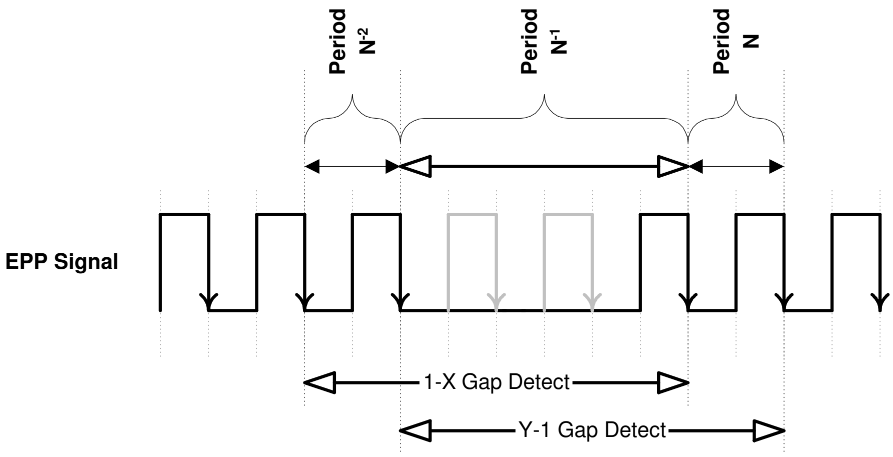
eTPU Primitive Implementation:

Engine Position Processing for the eTPU

(EPPE)



MCD-5408

Revision 18.0

February 15, 2011

DELPHI

CONFIDENTIAL

Complex I/O – Microcode
Powertrain Electronics Software
M/S CT-40D, Delco Electronics, Kokomo, Indiana 46904-9005

This page intentionally left blank

Table of Contents

Engine Position Processing for the eTPU

1.0 Introduction

This document describes the Engine Position Processing algorithm as implemented in the eTPU. When an input edge is received, the EPPE algorithm records the time of the input edge, computes a period and increments a tooth counter. The edge time is also copied into an array so the data from past edges can be evaluated. The algorithm is also able to generate synthetic "teeth" to facilitate the smooth operation of EST and other functions through the gap.

The algorithm stores a copy of the tooth counter if it detects a gap using either the Percent Period method between the last three periods, or a Threshold (1_X, Y_1) method, in which the last two periods are analyzed for gap determination.

The algorithm can record the edge time of up to four specified input edges, and optionally generate an interrupt to the host CPU when the specified edge(s) occurs.

The algorithm can filter out unwanted noise by ignoring input pulses with a period smaller than a user-defined threshold.

The algorithm has the capability to generate a tachometer output on a user-selected eTPU channel.

On each critical input edge, the algorithm records the state of the CAM input on a user-selected eTPU channel.

Finally, the algorithm has the capability to read a bi-directional crank sensor to detect backwards movements of the crankshaft ("rock-back") when the engine comes to a stop, so that the engine position at start-up may be known with greater accuracy.

The EPPE primitive also supports a backup mode for limp-home capability. When in backup mode, EPPE continues to process real input edges while creating backup matches where the host determines the crank edges should be occurring. The times of these backup matches are saved in the array. TCR2 is also updated per the backup matches.

2.0 System Overview

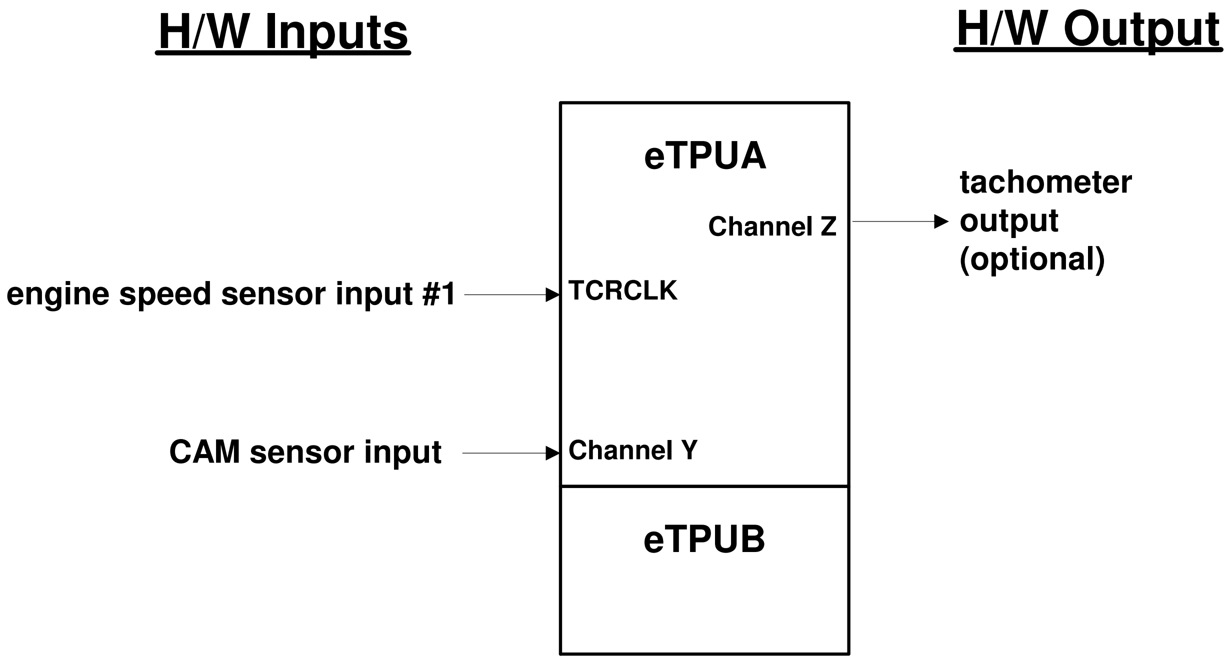


Figure 1: EPPE Mechanization

Note: Engine speed input, tachometer (if enabled) and cam sensor input (if using cam history feature) must be mechanized on the same eTPU.

2.1 Inputs

In a typical TPU application, two inputs are used: one with fine resolution and one with coarse resolution. The eTPU however, contains 24-bit timers providing the range and resolution required with just one input signal.

The crank input signal shall always to be brought in on the TCRCLK pin of the microprocessor. In angle mode and edge count mode, TCR2 will be derived from this input signal. In time mode, TCR2 will be derived from a separate eTPU via the STAC bus.

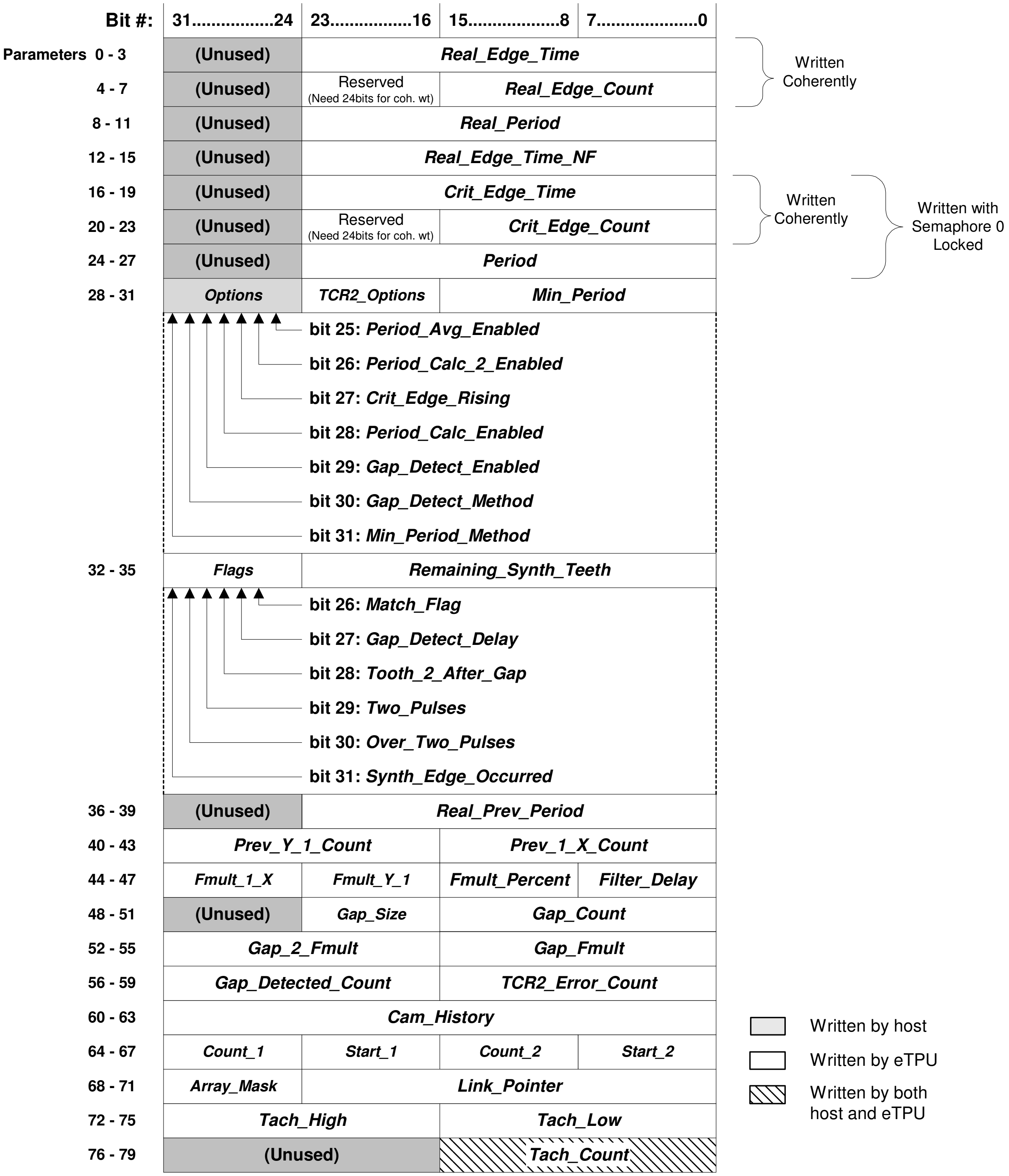
This primitive also reads the state of the CAM input when processing every edge of the engine position input. This input is shifted into a 32-bit value and saved in the variable called ***Cam_History***. If cam history is required, the CAM input must be mapped to the same eTPU as EPPE.

2.2 Outputs

This algorithm requires one hardware output if the user enables the tachometer output feature. This output may be on any channel of the same eTPU as EPPE, **except channel 0**.

3.0 Memory Map

3.1 Channel RAM Parameters



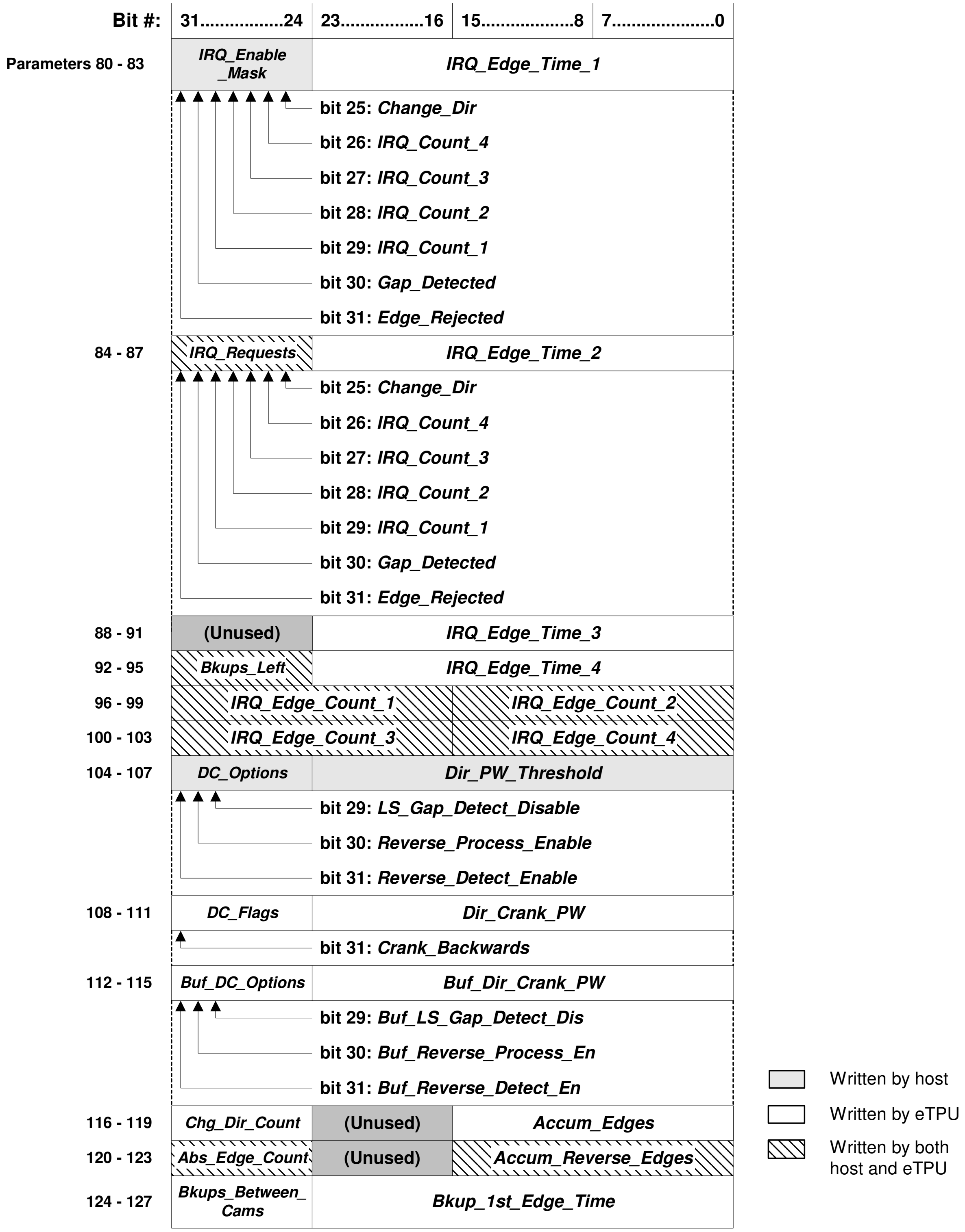
Bit #:	31.................24	23................16	15...................8	7.....................0	
Parameters 0 - 3	(Unused)	*Real_Edge_Time*			Written Coherently
4 - 7	(Unused)	Reserved (Need 24bits for coh. wt)	*Real_Edge_Count*		
8 - 11	(Unused)	*Real_Period*			
12 - 15	(Unused)	*Real_Edge_Time_NF*			
16 - 19	(Unused)	*Crit_Edge_Time*			Written Coherently Written with Semaphore 0 Locked
20 - 23	(Unused)	Reserved (Need 24bits for coh. wt)	*Crit_Edge_Count*		
24 - 27	(Unused)	*Period*			
28 - 31	*Options*	*TCR2_Options*	*Min_Period*		
	bit 25: *Period_Avg_Enabled* bit 26: *Period_Calc_2_Enabled* bit 27: *Crit_Edge_Rising* bit 28: *Period_Calc_Enabled* bit 29: *Gap_Detect_Enabled* bit 30: *Gap_Detect_Method* bit 31: *Min_Period_Method*				
32 - 35	*Flags*	*Remaining_Synth_Teeth*			
	bit 26: *Match_Flag* bit 27: *Gap_Detect_Delay* bit 28: *Tooth_2_After_Gap* bit 29: *Two_Pulses* bit 30: *Over_Two_Pulses* bit 31: *Synth_Edge_Occurred*				
36 - 39	(Unused)	*Real_Prev_Period*			
40 - 43	*Prev_Y_1_Count*		*Prev_1_X_Count*		
44 - 47	*Fmult_1_X*	*Fmult_Y_1*	*Fmult_Percent*	*Filter_Delay*	
48 - 51	(Unused)	*Gap_Size*	*Gap_Count*		
52 - 55	*Gap_2_Fmult*		*Gap_Fmult*		
56 - 59	*Gap_Detected_Count*		*TCR2_Error_Count*		
60 - 63	*Cam_History*				
64 - 67	*Count_1*	*Start_1*	*Count_2*	*Start_2*	
68 - 71	*Array_Mask*	*Link_Pointer*			
72 - 75	*Tach_High*		*Tach_Low*		
76 - 79	(Unused)		*Tach_Count*		

░ Written by host

□ Written by eTPU

▨ Written by both host and eTPU

Figure 2A – Parameter RAM



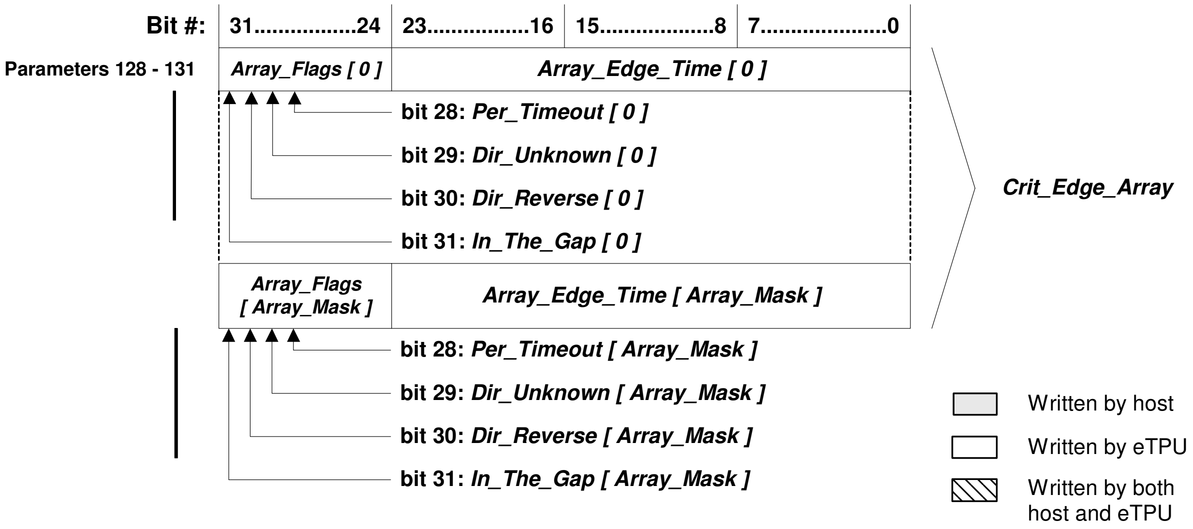
Bit #:	31.................24	23.................16	15..................8	7....................0
Parameters 80 - 83	*IRQ_Enable _Mask*	*IRQ_Edge_Time_1*		
	bit 25: *Change_Dir* bit 26: *IRQ_Count_4* bit 27: *IRQ_Count_3* bit 28: *IRQ_Count_2* bit 29: *IRQ_Count_1* bit 30: *Gap_Detected* bit 31: *Edge_Rejected*			
84 - 87	*IRQ_Requests*	*IRQ_Edge_Time_2*		
	bit 25: *Change_Dir* bit 26: *IRQ_Count_4* bit 27: *IRQ_Count_3* bit 28: *IRQ_Count_2* bit 29: *IRQ_Count_1* bit 30: *Gap_Detected* bit 31: *Edge_Rejected*			
88 - 91	(Unused)	*IRQ_Edge_Time_3*		
92 - 95	*Bkups_Left*	*IRQ_Edge_Time_4*		
96 - 99	*IRQ_Edge_Count_1*		*IRQ_Edge_Count_2*	
100 - 103	*IRQ_Edge_Count_3*		*IRQ_Edge_Count_4*	
104 - 107	*DC_Options*	*Dir_PW_Threshold*		
	bit 29: *LS_Gap_Detect_Disable* bit 30: *Reverse_Process_Enable* bit 31: *Reverse_Detect_Enable*			
108 - 111	*DC_Flags*	*Dir_Crank_PW*		
	bit 31: *Crank_Backwards*			
112 - 115	*Buf_DC_Options*	*Buf_Dir_Crank_PW*		
	bit 29: *Buf_LS_Gap_Detect_Dis* bit 30: *Buf_Reverse_Process_En* bit 31: *Buf_Reverse_Detect_En*			
116 - 119	*Chg_Dir_Count*	(Unused)	*Accum_Edges*	
120 - 123	*Abs_Edge_Count*	(Unused)	*Accum_Reverse_Edges*	
124 - 127	*Bkups_Between_ Cams*	*Bkup_1st_Edge_Time*		

░ Written by host

□ Written by eTPU

▨ Written by both host and eTPU

Figure 2B – Parameter RAM (cont.)


Bit #:
31................24
23................16
15..................8
7....................0
Parameters 128 - 131
Array_Flags [0]
Array_Edge_Time [0]
bit 28: Per_Timeout [0]
bit 29: Dir_Unknown [0]
bit 30: Dir_Reverse [0]
bit 31: In_The_Gap [0]
Crit_Edge_Array
Array_Flags [Array_Mask]
Array_Edge_Time [Array_Mask]
bit 28: Per_Timeout [Array_Mask]
bit 29: Dir_Unknown [Array_Mask]
bit 30: Dir_Reverse [Array_Mask]
bit 31: In_The_Gap [Array_Mask]
Written by host
Written by eTPU
Written by both host and eTPU

Note: the size for the array depends on the channel parameter ***Array_Mask***.
EPPE indexes into the array by 'ANDing' ***Crit_Edge_Count*** with ***Array_Mask***.

Crit_Edge_Array has a minimum size of 1 element (***Array_Mask*** = 0).

Figure 2C – Parameter RAM (cont.)

Host Service Requests:
HSR 7: Shutdown
HSR 6: (Unused)
HSR 5: Initialization
HSR 4: Request_Backup
HSR 3: Exit_Backup
HSR 2: Update
HSR 1: (Unused)

HW Semaphores:
Semaphore #0: Used to keep ***Period, Crit_Edge_Time*** and ***Crit_Edge_Count*** coherent between eTPUs.

Entry Point Flags:
Flag0: (unused)
Flag1: (unused)

Function Mode Bits:
FM0: ***EPPE_Use_TCR2***
0 = TIMEBASE_TCR1
1 = TIMEBASE_TCR2
FM1 ***EPPE_In_BACKUP***
0 = FALSE
1 = TRUE

Figure 2D – HSR Definitions, etc.

3.2 Global RAM

The following global RAM parameters may be accessed by the EPPE primitive. See the Application Implementation document for the actual memory locations of these parameters.

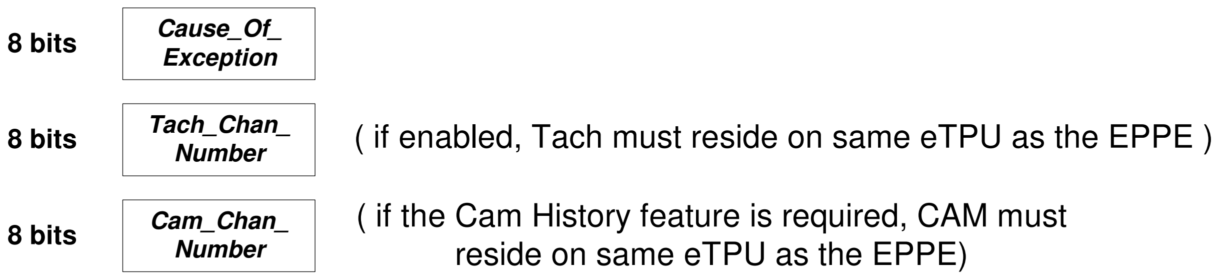

8 bits
Cause_Of_ Exception
8 bits
Tach_Chan_ Number
(if enabled, Tach must reside on same eTPU as the EPPE)
8 bits
Cam_Chan_ Number
(if the Cam History feature is required, CAM must reside on same eTPU as the EPPE)

Figure 3A – Global RAM

3.3 Link Array RAM

The user may define an array of link information (***EPPE_Link_Array***) in any section of RAM desired, addressed by the parameter ***Link_Pointer***.

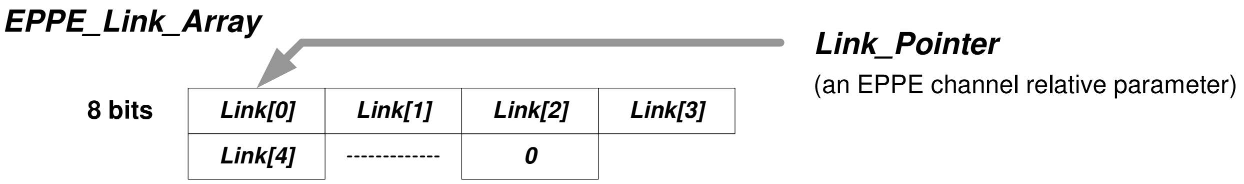


Figure 3B – Link Array RAM

3.4 RAM Definitions

3.4.1 Global RAM Definition

Parameter	CPU Useful	Range	Units	CPU Access	eTPU Access
Cam_Chan_Number	√	0 - 0x1F	eTPU Channel #	W	R
Tach_Chan_Number	√	0 - 0x1F	eTPU Channel #	W	R
Cause_Of_Exception	√	0 - 0x1F	eTPU Channel #	R	W

Table 1: Global RAM for EPPE

3.4.2 Link Array RAM Definition

Parameter	CPU Useful	Range	Units	CPU Access	eTPU Access
EPPE_Link_Array	(last element = 0x00)				
Link	√	0-0xFF	ETPU | Chan Num	W	R

Table 2: Link Array RAM

3.4.3 Parameter RAM Definition

Parameter	CPU Useful	Range	Units	CPU Access	eTPU Access
Real_Edge_Time	√	0 - 0xFFFFFF	Timer Units	R	W
Real_Edge_Count	√	0 - 0xFFFF	Count Units	R	W
Real_Period	√	0 - 0xFFFFFF	Timer Units	R	R/W
Real_Edge_Time_NF	√	0 - 0xFFFFFF	Timer Units	R	W
Crit_Edge_Time	√	0 - 0xFFFFFF	Timer Units	R	R/W
Crit_Edge_Count	√	0 - 0xFFFF	Count Units	R	R/W
Period	√	0 - 0xFFFFFF	Timer Units	R	R/W
Min_Period_Method	√	0,1	Flag	W	R
Gap_Detect_Method	√	0,1	Flag	W	R
Gap_Detect_Enabled	√	0,1	Flag	W	R
Period_Calc_Enabled	√	0,1	Flag	W	R
Crit_Edge_Rising	√	0,1	Flag	W	R
Period_Calc_2_Enabled	√	0,1	Flag	W	R
Period_Avg_Enabled	√	0,1	Flag	W	R
TCR2_Options	√	0,1,2,3	Field	W	R
Min_Period	√	0 - 0xFFFF	Timer Units	W	R
Synth_Edge_Occurred	No	0,1	Flag	-	R/W
Over_Two_Pulses	No	0,1	Flag	-	R/W
Two_Pulses	No	0,1	Flag	-	R/W
Tooth_2_After_Gap	No	0,1	Flag	-	R/W
Gap_Detect_Delay	No	0,1	Flag	-	R/W
Match_Flag	No	0,1	Flag	-	R/W
Remaining_Synth_Teeth	No	0 - 0xFFFFFF	Count Units	-	R/W

Parameter	CPU Useful	Range	Units	CPU Access	eTPU Access
Real_Prev_Period	√	0 - 0xFFFFFF	Timer Units	R	R/W
Prev_Y_1_Count	√	0 - 0xFFFF	Count Units	R	W
Prev_1_X_Count	√	0 - 0xFFFF	Count Units	R	W
Fmult_1_X	√	0 - 0xFF	0-1 Fractional Mult. (* 256)	W	R
Fmult_Y_1	√	0 - 0xFF	0-1 Fractional Mult. (* 256)	W	R
Fmult_Percent	√	0 - 0xFF	0-1 Fractional Mult. (* 256)	W	R
Filter_Delay	√	0 - 0xFF	Timer Units	W	R
Gap_Size	√	0 - 0xFF	Count Units	W	R
Gap_Count	√	0 - 0xFFFF	Count Units	R	W
Gap_2_Fmult	√	0 - 0xFFFF	0-1 Fractional Mult. (* 65536)	W	R
Gap_Fmult	√	0 - 0xFFFF	0-1 Fractional Mult. (* 65536)	W	R
Gap_Detected_Count	√	0 - 0xFFFF	Count Units	R	W
TCR2_Error_Count	√	0 - 0xFFFF	Count Units	R	W
Cam_History	√	0 - 0xFFFFFFFF	Cam States	R	W
Count_1	√	0 - 0xFF	Count Units	W	R
Start_1	√	0 - 0xFF	ETPU # | Channel #	W	R
Count_2	√	0 - 0xFF	Count Units	W	R
Start_2	√	0 - 0xFF	ETPU # | Channel #	W	R
Array_Mask	√	See description	Mask	W	R
Link_Pointer	√	0 - 0xFFFFFF	ETPU pram address	W	R
Tach_High	√	0 - 0xFFFF	Count Units	W	R
Tach_Low	√	0 - 0xFFFF	Count Units	W	R
Tach_Count	√	0 - 0xFFFF	Count Units	W	R/W
IRQ_Enable_Mask	√	0 - 0xFF	Mask	W	R
IRQ_Edge_Time_1	√	0 - 0xFFFFFF	Timer Units	R	W
IRQ_Requests	√	0 - 0xFF	Mask	R	R/W
IRQ_Edge_Time_2	√	0 - 0xFFFFFF	Timer Units	R	W
IRQ_Edge_Time_3	√	0 - 0xFFFFFF	Timer Units	R	W
Bkups_Left	No	0xFF	Counts	R/W	R/W
IRQ_Edge_Time_4	√	0 - 0xFFFFFF	Timer Units	R	W

Parameter	CPU Useful	Range	Units	CPU Access	eTPU Access
IRQ_Edge_Count_1	√	0 - 0xFFFF	Count Units	W	R
IRQ_Edge_Count_2	√	0 - 0xFFFF	Count Units	W	R
IRQ_Edge_Count_3	√	0 - 0xFFFF	Count Units	W	R
IRQ_Edge_Count_4	√	0 - 0xFFFF	Count Units	W	R
Reverse_Detect_Enable	√	0,1	Flag	W	R
Reverse_Process_Enable	√	0,1	Flag	W	R
LS_Gap_Detect_Disable	√	0,1	Flag	W	R
Dir_PW_Threshold	√	0 – 0xFFFFFF	Timer Units	W	R
Crank_Backwards	√	0,1	Flag	R	R/W
Dir_Crank_PW	√	0 – 0xFFFFFF	Timer Units	R	R/W
Buf_Reverse_Detect_En	No	0,1	Flag	-	R/W
Buf_Reverse_Process_En	No	0,1	Flag	-	R/W
Buf_LS_Gap_Detect_Dis	No	0,1	Flag	-	R/W
Buf_Dir_Crank_PW	√	0 – 0xFFFFFF	Timer Units	R	R/W
Chg_Dir_Count	√	0 – 0xFF	Count Units	R	R/W
Accum_Edges	√	0 – 0xFFFF	Count Units	R	R/W
Abs_Edge_Count	√	0 – 119	Count Units	R/W	R/W
Accum_Reverse_Edges	√	0 – 0xFFFF	Count Units	R/W	R/W
Bkups_Between_Cams	√	0xFF	Counts	W	R
Bkup_1st_Edge_Time	√	0xFFFFFF	Count Units	W	R
Crit_Edge_Array	(*Array_Mask* + 1 elements)				
In_The_Gap [X]	√	0,1	Flag	R	W
Dir_Reverse [X]	√	0,1	Flag	R	W
Dir_Unknown [X]	√	0,1	Flag	R	W
Per_Timeout [X]	√	0,1	Flag	R	W
Array_Edge_Time [X]	√	0 – 0xFFFFFF	Timer Units	R	W

Table 3: Parameter RAM for EPPE

3.4.4 Parameters Written by the CPU / Read by the eTPU

3.4.4.1 Function Mode Bits

The two Function Mode (FM) bits are located in the ETPUCxSCR register. They are written by the host CPU and read by the eTPU:

EPPE_Use_TCR2

FM0 – Set (1) if the time base is to be TCR2; cleared (0) if the time base is to be TCR1.

EPPE_In_Backup

FM1 – Set (1) if EPPE is operating in backup mode; cleared (0) if operating in normal mode. Refer to Section 4.8.12 for details on backup mode.

3.4.4.2 Global Parameters

Cam_Chan_Num The number (0x0 - 0x1F) of the eTPU channel that is used as the cam signal input.

Tach_Chan_Num

The number (0x1 - 0x1F) of thee TPU channel that is used as the tachometer output. **To disable the tachometer output feature, set *Tach_Chan_Num* = 0x0!**

3.4.4.3 Link Array Parameters

EPPE_Link_Array

An array of link information which the user may place anywhere in RAM. Each of the array elements indicates a specific channel and eTPU module to be linked to on every tooth (including "synthetic" teeth). Bits 4–0 represent the channel number while bits 7 & 6 indicate which eTPU engine to link to. Bit 5 is unused. See **Section 4.8.7** for details. ***Link_Array*** is pointed to by the channel relative parameter ***Link_Pointer.*** EPPE steps through the array linking to the appropriate channels until it encounters an entry of 0x00. An entry of 0x00 indicates there are no more channels to link to.

3.4.4.4 Channel Parameters

Min_Period_Method

Selects whether Min_Period filtering is performed in hardware or software. Refer to section 4.8.5 for details

HW_MIN_PERIOD = 1
SW_MIN_PERIOD = 0

Gap_Detect_Method

Selects whether Percent Period or Threshold (1_X, Y_1) gap detection algorithm is activated. See section 4.8.4.

PERCENT_PERIOD = 0
1_X__Y_1 = 1

Gap_Detect_Enabled

Determines whether or not the gap detection method specified by ***Gap_Detect_Method*** is enabled. Disabling the gap detection algorithm may be of use if execution time is a concern.

Period_Calc_Enabled

At the first tooth after the gap, when this flag is set ***Period*** is calculated to be the actual period of the gap (ie, ***Real_Period***) multiplied by ***Gap_Fmult***. When this flag is cleared, ***Period*** is not updated at the first tooth after the gap.

<u>NOTE</u>: Care must be taken when ***Period_Calc_Enabled*** is selected and the gap is misplaced. ***Period*** is still calculated as ***Real_Period*** * ***Gap_Fmult***. The resulting period may have adverse effects on other primitives.

Crit_Edge_Rising

Set (1) if the rising edge on the engine position input is critical; cleared (0) if the falling edge is critical.

Period_Calc_2_Enabled

At the second tooth after the gap, when this flag is set ***Period*** is calculated to be the sum of the actual period of the gap and the first actual period after the gap (ie, ***Real_Period*** + ***Real_Prev_Period***), multiplied by ***Gap_2_Fmult***. When this flag is cleared, ***Period*** is calculated normally (ie, as the time between the last two input edges) at the second tooth after the gap.

<u>NOTE</u>: Care must be taken when ***Period_Calc_2_Enabled*** is selected and the gap is misplaced. ***Period*** is still calculated as (***Real_Period*** + ***Real_Prev_Period***) * ***Gap_2_Fmult***. The resulting period may have adverse effects on other primitives.

Period_Avg_Enabled

When this flag is set, ***Period*** is calculated as (***Real_Period*** + ***Real_Prev_Period***) / 2. When this flag is cleared, ***Period*** is calculated normally (ie, as the time between the last two input edges).

<u>NOTE</u>: ***Period_Avg_Enabled*** is ignored on the first two teeth after the gap. On these teeth, the value of ***Period*** will be calculated according to the ***Period_Calc_Enabled*** and ***Period_Calc_2_Enabled*** bits.

TCR2_Options Selects the input to TCR2. Refer to section 4.8.3 for details on each of these modes.

EPPE_TIME_MODE = 0
EPPE_ANGLE_HW = 1
EPPE_EDGE_COUNT = 2

Min_Period On each critical input edge, the time since the last critical input edge (***Real_Period***) will be tested against ***Min_Period***. If the current value of ***Real_Period*** is smaller than ***Min_Period***, the eTPU will ignore the edge and no variables will be updated. If no filtering by the eTPU is desired, set ***Min_Period*** = 0x0000. ***Min_Period_Method*** selects whether this filtering is done by hardware or software. Refer to section 4.8.5 for details.

Fmult_1_X This byte is a 0-1 fractional multiplier or ratio of the previous period to the current period. When:

(current period * ***Fmult_1_X***) > previous period

the eTPU will store the current value of ***Crit_Edge_Count*** as ***Prev_1_X_Count***.

Fmult_Y_1 This byte is a 0-1 fractional multiplier or ratio of the current period to the previous period. When:

(previous period * ***Fmult_Y_1***) > current period

the eTPU will store the current value of ***Crit_Edge_Count*** as ***Prev_Y_1_Count*** and interrupt the host (if enabled). Refer to section 4.8.4.2.

Fmult_Percent This byte is a 0-1 fractional multiplier or ratio of period N^{-1} to the summation of periods N, N^{-1} and N^{-2}. ***Fmult_Percent*** = N^{-1} / (N + N^{-1} + N^{-2}). Refer to section 4.8.4.1 for details on this gap detection algorithm.

Filter_Delay This value represents the total digital filter delay on the engine position input. On every real critical edge, ***Filter_Delay*** is subtracted from ***Real_Edge_Time*** and stored in ***Real_Edge_Time_NF***. See section 4.8.5.1

Gap_Size The number of teeth in the gap. When ***Crit_Edge_Count*** equals ***Gap_Count***, ***Gap_Size*** synthetic teeth are created. If ***TCR2_Options*** = ANGLE_HW, then the maximum ***Gap_Size*** supported is 4.

Gap_Count ***Gap_Count*** represents the tooth immediately preceding the gap. When ***Crit_Edge_Count*** equals ***Gap_Count*** the eTPU begins setting up a series of "synthetic" edges based on the current period. The CPU will have to keep moving ***Gap_Count*** ahead to disable it! A suggested initialization value for ***Gap_Count*** is (***Crit_Edge_Count*** + 0xFFFF).

Gap_2_Fmult Fractional value used to calculate ***Period*** on the second tooth after the gap. It represents the 0 – 1 multiplier [65536 / (***Gap_Size*** + 2)]. The eTPU reads this value at the second tooth after the gap and multiplies it by (***Real_Period*** + ***Real_Prev_Period***) to determine ***Period***. Setting the flag ***Period_Calc_2_Enabled*** enables this calculation. When the flag is cleared, ***Period*** is calculated normally (ie, as the time between the last two input edges) at the second tooth after the gap.

Gap_Fmult Fractional value used to calculate ***Period*** on the first tooth after the gap. It represents the 0 – 1 multiplier [65536 / (***Gap_Size*** + 1)]. The eTPU reads this value at the first tooth after the gap and multiplies it by ***Real_Period*** to determine ***Period***. Setting the flag ***Period_Calc_Enabled*** enables this calculation. When the flag is cleared, ***Period*** is not updated at the first tooth after the gap.

Count_1, Start_1
Count_2, Start_2 The parameters ***Count_1*** and ***Start_1*** along with ***Count_2*** and ***Start_2*** are used in conjunction to indicate what channels to link to. A link is a way to communicate to another channel. For this primitive, it is a way to tell the other channel that fresh input data is available. A link is a lot like an IRQ to another channel. These parameters are used as in the TPU design where ***Start_x*** is the first channel where this primitive will link to on every input edge. In contrast to the TPU design, the upper two bits of ***Start_x*** must also indicate which eTPU module to link to. ***Count_x*** is the number of consecutive channels to link to starting with ***Start_x***. Links can be disabled by setting ***Count_x*** to 0. Requested link(s) are generated on every tooth (including "synthetic" teeth). See section 4.8.7 for details.

Array_Mask Mask supplied by the host to determine the size of the critical edge time array, where the size of the array equals ***Array_Mask*** + 1. EPPE indexes into the array by 'ANDing' ***Crit_Edge_Count*** by ***Array_Mask*** thus the valid array sizes limited. See below:

Valid values of ***Array Mask***	Array Size
0x00	1
0x01	2
0x03	4
0x07	8
0x0F	16
0x1F	32
0x3F	64

0x7F	128
0xFF	256

Link_Pointer Pointer to the location in the eTPU parameter RAM containing the array of channel numbers to link to (***EPPE_Link_Array***). Each of the array elements indicate a specific channel and eTPU module to be linked to on every tooth (including "synthetic" teeth). Set ***Link_Pointer*** to 0 if no channels require a link. Write a channel number of 0x00 into the array to indicate there are no more channels to link to. See **Section 4.8.7** for details.

Tach_High The number of EPPE periods that the tachometer output signal will be in the high state. See section 4.8.11.

Tach_Low The number of EPPE periods that the tachometer output signal will be in the low state. See section 4.8.11.

IRQ_Enable_Mask

Determines which interrupt sources are enabled within the EPPE primitive. See section 4.8.8.

Enabled = 1
Disabled = 0

Reverse_Detect_Enable

Set (1) to enable detection of reverse crank pulses; cleared (0) otherwise. See **Section 4.10** for details.

Reverse_Process_Enable

Set (1) to enable processing of reverse crank pulses; cleared (0) otherwise. See **Section 4.10** for details.

LS_Gap_Detect_Disable

Set (1) to disable gap detection at very low engine speed; cleared (0) otherwise. When ***LS_Gap_Detect_Disable*** is cleared (0), gap detection will be delayed via the ***Gap_Detect_Delay*** bit. See **Section 4.8.4.3** for details.

Dir_PW_Threshold

Crank pulse width threshold used to determine crank direction. Applied only when ***Reverse_Detect_Enable*** = 1.

Bkups_Between_Cams

The number of backup matches (or desired crank edges) that need to occur between this **Request_Backup** HSR (corresponding to the previous cam edge) and the next.

Bkup_1st_Edge_Time

The absolute time of the first backup match to occur after the **Request_Backup** HSR (may be in the past).

3.4.5 Parameters written by the eTPU / Read by the Host CPU

3.4.5.1 Global Parameters

Cause_Of_Exception:
The number of the eTPU channel on which a global exception fault has occurred. This condition is caused by an unexpected event (output match, input transition, link, Host Service Request, etc) occurring on the channel in question. See **Section 4.6** for details on what conditions can cause a global exception.

3.4.5.2 Channel Parameters

Real_Edge_Time Time of the most recent **actual** input edge (ie, not including "synthetic" edges).

Real_Edge_Count
Count of **actual** input edges (ie, not including "synthetic" edges).

Real_Period Time between **actual** input edges (ie, not including "synthetic" edges). Unlike ***Period*** (when ***Gap_Calc_Enabled*** is clear), ***Real_Period*** is updated on the tooth immediately following the gap. The eTPU will write 0xFFFFFF into ***Real_Period*** if 0xFFFFFF timer counts elapsed since the last input edge.

Real_Edge_Time_NF
Time of the most recent **actual** input edge (ie, not including "synthetic" edges) subtracted by ***Filter_Delay***. See **Section 4.8.5.1**.

Crit_Edge_Time Time of the most recent edge (including "synthetic" edges).

Crit_Edge_Count
Incremented on every edge (including "synthetic" edges).

Period When ***Period_Avg_Enabled*** = 0, ***Period*** is calculated as the time between the last two input edges (including "synthetic" edges). When ***Period_Avg_Enabled*** = 1, ***Period*** will be calculated as (***Real_Period*** + ***Real_Prev_Period***) / 2. This is the normal condition, and it applies to all input edges except the first two teeth after the gap (***Period_Avg_Enabled*** is ignored on these teeth).

On the first tooth after the gap, when ***Period_Calc_Enabled*** = 0, ***Period*** is not updated at all; when ***Period_Calc_Enabled*** = 1, ***Period*** is calculated as ***Real_Period*** * ***Gap_Fmult***.

On the second tooth after the gap, when ***Period_Calc_2_Enabled*** = 0, ***Period*** is calculated normally (ie, as the time between the last two input edges); when ***Period_Calc_2_Enabled*** = 1, ***Period*** is calculated as (***Real_Period*** + ***Real_Prev_Period***) * ***Gap_2_Fmult***.

The eTPU will write 0xFFFFFF into ***Period*** if 0xFFFFFF timer counts have elapsed since the last input edge, and also on the first input edge received.

Real_Prev_Period

Time between **actual** input edges of previous period (ie, not including "synthetic" edges). The eTPU will write 0xFFFFFF into ***Period*** if 0xFFFFFF timer counts elapsed since the last input edge.

Prev_1_X_Count

The value of ***Crit_Edge_Count*** at which the current period times ***Fmult_1_X*** is greater than the previous period (ie, at the tooth immediately following the gap). Not updated until at least 3 input edges have been received after initialization. See **Section 4.8.4.2** for details on this gap detection algorithm.

Prev_Y_1_Count

The value of ***Crit_Edge_Count*** at which the previous period times ***Fmult_Y_1*** is greater than the current period (ie, at the second tooth following the gap). Not updated until at least 3 input edges have been received after initialization. See **Section 4.8.4.2** for details on this gap detection algorithm.

Gap_Detected_Count

Variable ***Gap_Detected_Count*** is used when Percent Period method of Gap Detection is selected via ***Gap_Detect_Method***. The value of ***Crit_Edge_Count*** at the last occurrence of input period where $N^{-1} > (N + N^{-1} + N^{-2})$ * ***Fmult_Percent***. If enabled via ***IRQ_Enable_Mask***, an interrupt is generated to the host CPU and indicated with bit ***Gap_Detected*** in ***IRQ_Requests***. See **Section 4.8.4.1** for details on this gap detection algorithm.

TCR2_Error_Count

When TCR2 is in angle mode (indicated by ***TCR2_Options***) the upper word of TCR2 is compared to ***Crit_Edge_Count***. If TCR2 is not within the predetermined limits, ***Crit_Edge_Count*** is written to the upper word of TCR2, the number of expected microticks is written to the lower word of TCR2 and ***TCR2_Error_Count*** is incremented. Refer to **Section 4.8.3.3** for details on angle mode.

Cam_History The last 32 pin states of the cam input indicated by the global variable ***Cam_Chan_Number***. Updated on every EPPE input edge. Bit 0 represents the most recent cam pin state.

IRQ_Edge_Time_x (x = 1-4)

When ***Crit_Edge_Count*** equals or is greater than ***IRQ_Edge_Count_x***, a copy of ***Array_Edge_Time[X]*** read from the ***Crit_Edge_Array*** corresponding to ***IRQ_Edge_Count_x*** is written to ***IRQ_Edge_Time_x***. This will allow the host CPU to determine the time of the edge corresponding to ***IRQ_Edge_Count_x*** even when several new input edges have been detected. If this interrupt source is enabled via the ***IRQ_Count_x*** bit in ***IRQ_Enable_Mask***, the corresponding interrupt source bit is set in ***IRQ_Requests*** and an interrupt is generated to the host CPU.

Crank_Backwards

Set (1) when crank is moving backwards; cleared (0) otherwise. Applied only when ***Reverse_Process_Enable*** = 1.

Dir_Crank_PW Width of most recent crank pulse, measured from critical edge to non-critical edge. Updated only when ***Reverse_Detect_Enable*** = 1.

Buf_Dir_Crank_PW

Buffered copy of ***Dir_Crank_PW***, updated on critical edge. Updated only when ***Reverse_Detect_Enable*** = 1.

Chg_Dir_Count Free-running counter that is incremented whenever a change in crank direction is detected. Updated only when ***Reverse_Process_Enable*** = 1.

Accum_Edges Free-running counter that mirrors ***Real_Edge_Count***, except that it is decremented whenever a genuine reverse crank pulse is detected.

Crit_Edge_Array An array containing the times of the critical edge events (***Array_Edge_Time [X]***) with associated flags (***Array_Flags [X]***). The size of the array is the channel relative parameter ***Array_Mask*** + 1 and has a minimum size of 1 element. The eTPU indexes into the array by 'ANDing' ***Crit_Edge_Count*** with ***Array_Mask***, thus the size of the array is limited. Refer to definition of ***Array_Mask*** for valid array sizes.

***In_The_Gap* [X]** Set (1) if the associated tooth is a synthetic tooth occurring in the gap; otherwise cleared (0).

Dir_Reverse* [X]** Set (1) when the associated tooth is in the reverse direction; otherwise cleared (0). If ***Reverse_Process_Enable = 0, then ***Dir_Reverse*[X]** will always be cleared (0). If ***Reverse_Process_Enable*** = 1, then ***Dir_Reverse*[X] will not be updated until the non-critical edge** (since crank direction cannot be determined until then).

<u>NOTE</u>: *Dir_Reverse*[X] is <u>not valid</u> when *Dir_Unknown*[X] = 1!

Dir_Unknown* [X]** Set (1) if the direction of the associated tooth has not yet been determined (i.e., between the critical and non-critical edges); otherwise cleared (0). If ***Reverse_Process_Enable = 0, then ***Dir_Unknown*[X]** will always be cleared (0). If ***Reverse_Process_Enable*** = 1, then ***Dir_Unknown*[X]** will be set (1) on the critical edge and cleared (0) on the non-critical edge (when crank direction becomes known).

***Per_Timeout* [X]** Set (1) if a period timeout occurred prior to the associated tooth; otherwise cleared (0). (If ***Per_Timeout*[X]** = 1, then the associated edge time cannot be used to calculate the period ending with that edge time.)

***Array_Edge_Time* [X]**

Critical edge time. If acceleration occurred and a real edge was received before all synthetic edges were complete, the synthetic edges are "made up" by recording the time of the real edge in the array for each of these made-up edges.

<u>CAUTION:</u> If ***Array_Edge_Time*[X]** values are used to calculate the period between two teeth, that period will be 0 in the case of "made up" teeth.

3.4.6 Parameters written by BOTH the eTPU and the host CPU

Tach_Count Down-counter, decremented on every EPPE critical input edge (plus synthetic edges). When the value of ***Tach_Count*** reaches zero (or less), the eTPU will invert the tachometer output signal and re-load ***Tach_Count*** with either ***Tach_High*** or ***Tach_Low*** (depending on the pin state). **<u>It is up to the user to initialize *Tach_Count* to some desired value before the first EPPE edge occurs!</u>** However, it is assumed that the user will **not** write to ***Tach_Count*** again after initializing it, since this could interfere with the eTPU's attempts to update ***Tach_Count*** .

IRQ_Requests

Contains a bit for each interrupt source from the EPPE primitive to the host CPU. A set bit indicates an interrupt source is pending, a clear bit indicates the interrupt source is not pending. The host is responsible for clearing this byte once the interrupt is acknowledged, but care must be taken to prevent conflicts with the eTPU accessing this byte. Section 4.8.8 describes how the host should disable the EPPE channel before accessing this byte.

Interrupt requested = 1
No interrupt requested = 0

NOTE: EPPE will continue to issue interrupts to the host until the host clears ***IRQ_Requests***.

NOTE: the host must disable the EPPE channel (as described in section 4.8.8) before writing to ***IRQ_Requests*** to prevent contention issues between the host and the eTPU.

Bkups_Left Temporary variable used by the eTPU to track the number of backup matches that still need to be issued.

IRQ_Edge_Count_x (x = 1 – 4)

The value of ***Crit_Edge_Count*** when the CPU desires the edge time to be buffered. The eTPU buffers ***Array_Edge_Time[X]*** from ***Crit_Edge_Array*** into ***IRQ_Edge_Time_x.*** The eTPU does this if ***Crit_Edge_Count*** equals or is greater than ***IRQ_Edge_Count_x***. The eTPU will automatically increase ***IRQ_Edge_Count_x*** to ***Crit_Edge_Cnt*** **+ 0x8000** once the data is buffered to prevent future "buffering". The CPU has plenty of time after the edge to modify ***IRQ_Edge_Count_x*** back to normal. The CPU can determine which edge set an interrupt by subtracting 0x8000 from ***IRQ_Edge_Count_x***. Then the CPU can issue a new ***IRQ_Edge_Count_x*** to set the next request. If this interrupt source is enabled via the ***IRQ_Count_x*** bit in ***IRQ_Enable_Mask***, the corresponding interrupt source bit is set in ***IRQ_Requests*** and an interrupt is generated to the host CPU.

Abs_Edge_Count

Counter, incremented or decremented on every critical or non-critical crank edge. The range of ***Abs_Edge_Count*** is from 0 to 119, where 1 and 61 each represent the first tooth after the gap. When ***Reverse_Process_Enable*** = 0, the eTPU will increment ***Abs_Edge_Count*** on every critical edge, but it is up to the CPU to ensure that the counter is in sync. (In fact, the value of ***Abs_Edge_Count*** is meaningless until the CPU synchronizes it.) When ***Reverse_Process_Enable*** = 1, then the CPU should do nothing but allow the eTPU to increment or decrement ***Abs_Edge_Count*** on every non-critical edge according to the direction in which the crank is moving.

NOTE: Assuming a 58X crank sensor, *Abs_Edge_Count* will never have the values of 59, 60, 119, or 0. These values represent synthetic teeth in the gap, which are skipped by *Abs_Edge_Count*.

NOTE: ***Abs_Edge_Count*** **will not be updated on broken tooth matches or on backup matches.**

Accum_Reverse_Edges

Counts number of reverse crank pulses (genuine or otherwise) detected since power-on. Limited to 0xFFFF. Updated only when ***Reverse_Detect_Enable*** = 1. Reset to 0x0000 by host CPU.

3.4.7 Parameters accessed by the eTPU ONLY

3.4.7.1 Channel Parameters

Synth_Edge_Occurred

This flag is set when synthetic edges are being processed. If this flag is set when a real edge is being processed, the EPPE knows it is the first real edge after the gap and ***Period*** is NOT updated.

Over_Two_Pulses

This flag indicated to the EPPE primitive that at least three edges have occurred. When this flag is set, the primitive performs gap analysis. Otherwise, the gap analysis is skipped until more edges have occurred.

Two_Pulses This flag is used as an entry table condition to indicate whether it is about to process the first edge, or two or more. If clear, EPPE does not calculate a period.

Tooth_2_After_Gap

This flag is set on the first real tooth after a gap and cleared on the next real tooth. If this flag is set when a real edge is being processed, the EPPE knows it is the second real edge after the gap and ***Period_Calc_2_Enabled*** is tested.

Gap_Detect_Delay

This flag is set (1) when the ***LS_Gap_Detect_Disable*** bit is set. It is cleared (0) one crank pulse after the ***LS_Gap_Detect_Disable*** bit is cleared, in order to delay gap detection (thus eliminating false gaps) when coming out of a very low engine speed condition.

Match_Flag This flag is set (1) when any type of match occurs (synthetic tooth, backup match, time-out). It is cleared (0) whenever any type of edge event occurs (critical edge, non-critical edge).

Remaining_Synth_Teeth

This count is used by the EPPE primitive to indicate the number of synthetic teeth that are yet to be created.

Buf_Reverse_Detect_En
Copy of ***Reverse_Detect_Enable*** bit, buffered on critical edge.

Buf_Reverse_Process_En
Copy of ***Reverse_Process_Enable*** bit, buffered on critical edge.

Buf_LS_Gap_Detect_Dis
Copy of ***LS_Gap_Detect_Disable*** bit, buffered on critical edge.

4.0 Host Interface

4.1 Initialization of EPPE channel

The CPU should initialize the EPPE function as follows:

1. TCR1 Time Base :

   a. Select system clock divided by 2 as the clock source for the TCR1 prescaler by writing %10 to the TCR1CTL field of the ETPUTBCR register.

   b. Initialize the prescaler for TCR1 using the TCR1P field in the ETPUTBCR register. The prescaler divides its input by (TCR1P + 1).

2. TCR2 Time Base:

   To initialize the TCR2 time base first initialize the channel parameter ***TCR2_Options*** to the selected mode. %00 indicates time mode where EPPE does not control TCR2, %01 indicates angle mode, %10 indicates Edge Count mode where the EPPE primitive writes the edge count shifted 8 bits to the TCR2 register and %11 indicated backup mode. The following definitions are provided:

   ```
   #define EPPE_TIME_MODE        ( 0x00 )
   #define EPPE_ANGLE_HW         ( 0x01 )
   #define EPPE_EDGE_COUNT       ( 0x02 )
   ```

   To initialize the eTPU control registers controlling TCR2, perform the following steps.

   In a single write to the ETPUTBCR register:

   i. Disable angle mode logic by writing %0 to the AM bit.

   ii. Disable hardware's control of TCR2 by writing %111 to the TCR2CTL field.

   In a single write to the ETPUTBCR register:

   If in angle mode, ensure that the **Initialize HSR** has completed before enabling the angle mode hardware and TCR2 (see Step 21).

3. <u>EPPE Critical Edges</u>: Determine which edges of the engine position input are the critical edges and write the desired selection to ***EPPE_Crit_Edge_Rising***. The following definitions are provided:

   #define EPPE_CRIT_FALLING (0)

   #define EPPE_CRIT_RISING (1)

4. <u>EPPE Time Base</u>: Write the desired time base for the captured EPPE edge times to ***EPPE_Use_TCR2*** (FM0) in the FM (channel function mode) field in the ETPUCxSCR register. The following definitions are provided:

   #define EPPE_TIMEBASE_TCR1 (0)

   #define EPPE_TIMEBASE_TCR2 (1)

5. <u>Backup Mode</u>: Ensure EPPE is set up to operate in normal mode by clearing bit ***EPPE_In_Backup*** (FM1) in the FM (channel function mode) field in the ETPUCxSCR register. The following definitions are provided:

   #define EPPE_CAM_NORMAL_MODE (0)
   #define EPPE_CAM_BACKUP_MODE (1)

6. <u>Input Filtering</u>:

   a. If additional filtering is desired on the engine position input edge, initialize the parameters ***Min_Period*** and ***Min_Period_Method*** as described in Section 4.8.5. The following definitions are provided:

      #define EPPE_SW_MIN_PERIOD (0)

      #define EPPE_HW_MIN_PERIOD (1)

   b. If no additional filtering is desired, set ***Min_Period*** to 0x0000 and ***Min_Period_Method*** to EPPE_HW_MIN_PERIOD (1).

   c. Initialize the variable ***Filter_Delay***, in the selected time base, representing the total filter delay of all hardware filters. See **Section 4.8.5.1**.

   d. **NOTE**: If initializing EPPE in Backup mode, be sure to set ***Min_Period_Method*** to SW_MIN_PERIOD.

7. <u>Edge Time Array</u>: Write the mask corresponding to the desired size of the edge time array into ***Array_Mask*** (see definition of ***Array_Mask*** for details).

8. <u>Gap Detection Algorithms</u>:

   a. If it is desired to have EPPE perform gap analysis, enable this feature by writing to ***Gap_Detect_Enabled.*** The following definitions are provided:

      #define EPPE_DETECT_DISABLED (0)
      #define EPPE_DETECT_ENABLED (1)

   b. If it is desired to disable gap detection until the engine speed is above a certain threshold, set (1) the ***LS_Gap_Detect_Disable*** bit. See **Section 4.8.4.3**.

   c. Initialize the flag ***Gap_Detect_Method*** selecting either Percent Period or Threshold (1_X/1_Y) gap detection algorithms. See **Section 4.8.3** for details on these algorithms. The following definitions are provided:

      #define EPPE_PERCENT_PERIOD (0)

      #define EPPE_1_X__Y_1 (1)

   d. Initialize ***Gap_Count*** far into the future, such as ***Crit_Edge_Count*** + 0xFFFF.

e. Initialize the size of the gap in ***Gap_Size***. (Note: if angle mode is selected, the maximum ***Gap_Size*** supported is 4.)

f. If using the Threshold algorithm for gap detection, initialize ***Fmult_1_X*** and ***Fmult_Y_1***.

g. If using the Percent Period algorithm for gap detection, initialize ***Fmult_Percent***.

9. Bi-directional crank: If the bi-directional crank algorithm is to be enabled, set (1) the ***Reverse_Detect_Enable*** and ***Reverse_Process_Enable*** bits and write a desired value to ***Dir_PW_Threshold***; otherwise, clear (0) the ***Reverse_Detect_Enable*** and ***Reverse_Process_Enable*** bits. See **Section 4.10**.

10. Linking to other channels: If no links are required, set ***Link_Pointer*** to 0 along with ***Count_1***, ***Count_2*** to 0. Otherwise, write the list of channels requiring a link to the array ***EPPE_Link_Array***. Initialize ***Link_Pointer*** to point to the global array. If contiguous channels require linking, set ***Start_1*** to the first channel and the number of channels (including that in ***Start_1***) in ***Count_1 and Start_2, Count_2*** indicating which channels need to be linked to after an edge is processed (both real and synthetic). See **Section 4.8.7**.

11. Tachometer:

a. If the tachometer output feature is to be disabled, then set ***Tach_Chan_Num* = 0x0**.

b. If the tachometer output feature is to be enabled:

i. Write the appropriate (non-zero) value to ***Tach_Chan_Num***.

ii. Write a desired value to ***Tach_Cnt***, ***Tach_High*** and ***Tach_Low***. See **Section 4.8.11**.

12. Cam History: Write the channel number of the CAM input to the global variable ***CAM_Chan_Number.***

13. IRQ Count Interrupts: Determine where the host desires to be interrupted and provide to ***IRQ_Edge_Count_1, IRQ_Edge_Count_2, IRQ_Edge_Count_3*** and ***IRQ_Edge_Count_4***.

14. Interrupts:

a. Select which interrupt sources from the EPPE primitive should generate an interrupt from the host in the ***IRQ_Requests*** parameter and set the corresponding bit(s) in ***IRQ_Enable_Mask***. See **Section 4.8.8**.

b. There are two ways of enabling the hardware to cause an interrupt from EPPE to the host:

i. If any interrupts from EPPE to the host are desired, write %1 to the CIE bit for the EPPE channel in the ETPUCIER register, otherwise write %0 to disable interrupts. Note the state of this bit is reflected in the ETPUCxCR register.

OR

ii. Interrupts can also be enabled from EPPE to the host by writing %1 to the CIE bit in the ETPUCxCR register, otherwise write %0 to disable interrupts. Note the state of this bit is reflected in the ETPUCIER register.

15. DMA Trigger:

If the host desires the eTPU to trigger the DMA when it completes the processing of each edge (both real and synthetic) the following steps are recommended:

a. There are two ways of enabling the hardware to allow the eTPU to cause a DMA trigger:

i. If any DMA triggers from EPPE are desired, write %1 to the DTRE bit for the EPPE channel in the ETPUCDTRER register, otherwise write %0 to disable transfer requests. Note the state of this bit is reflected in the ETPUCxCR register.

OR

ii. DMA trigger requests can also be enabled from EPPE by writing %1 to the DTRE bit in the ETPUCxCR register, otherwise write %0 to disable transfer requests. Note the state of this bit is reflected in the ETPUCDTRER register.

16. ETPU Channel Configuration Register (ETPUCxCR):

a. Select the standard entry table condition for the EPPE primitive by writing a 0 to the ETCS field. This value is passed from the microcode set as:

#define EPPE_ENTRYTABLE_TYPE (0)

b. Write the EPPE function number to the Channel Function Select (CFS) field. The following definition is provided:

#define EPPE_FUNCTION_NUM (x)

where ‘x’ value depends on the microcode set.

c. Write the EPPE channel’s parameter base address to the CPBA field.

17. Configure the SIU so the pins are configured as inputs to the eTPU. This includes the TCRCLK pin when in angle mode or edge count mode.

18. If using edge count mode, perform the following steps.

In a single write to ETPUTBCR:

a. Set the angle mode bit (AM), enabling the angle hardware.

b. Ensure that hardware’s control of TCR2 is disabled by writing %111 to the TCR2CTL field.

19. Initialize HSR: Issue an **Initialize** Host Service Request by writing (%101) to the HSR field in the ETPUCxHSRR register. The following definition is provided.

#define EPPE_HSR_INIT (5)

20. Enable the channel by assigning a priority to the CPR field of the ETPUCxCR register. High priority (%11) is recommended.

21. If using angle mode, **wait until the completion of the initialization HSR** before enabling the angle mode hardware and TCR2, as described in the following steps.

In a single write to ETPUTBCR:

a. Set the angle mode bit (AM), enabling the angle hardware.

b. Initialize TCR2CTL field with the critical edge by writing either %001 for rising edges or %010 for falling edges.

> **NOTE:** The correct critical edge must be selected for both TCR2CTL field in the ETPUTBCR register AND the flag ***Crit_Edge_Rising***.

4.2 Shutdown

This function can be disabled by simply issuing a **Shutdown** Host Service Request (*%111*), waiting for the HSR bits to clear, and setting the Priority level to Disabled (*%00*).

4.3 Reinitializing EPPE

4.3.1 Angle Mode

If the application desires to shutdown and reinitialize EPPE while in angle mode, care must be taken to keep TCR2 synchronized with ***Crit_Edge_Count***.

1. Ensure the port logic (in the SIU) has Channel 0 configured such that the eTPU does not have any signal being brought in through TCRCLK (i.e. change SIU so the channel 0 pin is a discrete I/O). This step is important since while angle mode is disabled (AM=0) and after the channel is initialized, input edges from channel 0 are passed to EPPE instead of the input on TCRCLK when AM = 1. If channel 0 is not modified via the SIU, EPPE would increment ***Crit_Edge_Count***, but the angle hardware would not be incrementing TCR2.
2. In a single write to ETPUTBCR:

   a. Disable TCR2 by writing %111 to the TCR2CTL field.

   b. Clear the angle mode bit (AM), disabling the angle hardware.
3. Issue a **Shutdown** Host Service Request (*%111*), waiting for the HSR bits to clear, and setting the Priority level to Disabled (*%00*).
4. Issue an **Initialization** Host Service Request (%101), waiting for the HSR bits to clear.
5. In a single write to ETPUTBCR:

   a. Set the angle mode bit (AM), enabling the angle hardware.

   b. Initialize TCR2CTL field with the critical edge by writing either %001 for rising edges or %010 for falling edges.

4.3.2 Time or Edge Count Modes

To reinitialize the EPPE primitive when using either time or edge count modes, simply:

1. Issue a **Shutdown** Host Service Request (*%111*), waiting for the HSR bits to clear, and setting the Priority level to Disabled (*%00*).
2. Issue an **Initialization** Host Service Request (%101), waiting for the HSR bits to clear.

4.4 Switching from EPPE to Another Function

If the CPU wishes to switch from the EPPE function to another function, the CPU needs to follow the directions given above to Shutdown EPPE and then continue with the directions for initialization for the new function.

4.5 Switching from Another Function to EPPE

Contrary to the TPU primitives, all eTPU primitives contain a **Shutdown** HSR. To change the channel's function to EPPE, the host CPU must follow shutdown procedure described about in **Section 4.2** and then initialize the EPPE function by following the steps outlined in **Section 4.1**.

4.6 Global Exception

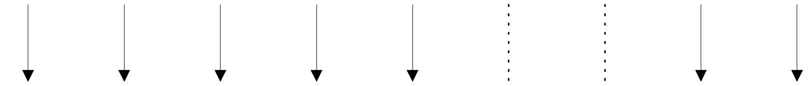
Under certain error conditions, the EPPE primitive issues a global exception to the host and writes its channel number to the global variable ***Cause_Of_Exception***. Below are the conditions that cause EPPE to generate a global exception:

1. An HSR is issued to the EPPE primitive that has not been defined.
2. A link has been issued to the EPPE primitive (links are not supported by EPPE).
3. HW error allows servicing of an EPPE input transition, when the ***Min_Period*** match has not been satisfied.

4.7 Example

The following picture is intended to explain how some of the EPPE parameters interact. It shows the gap with 2 “synthetic” edges created by the ETPU (***Gap_Size*** = 2).

Engine Speed Sensor (58X)



Real_Edge_Time		500	2116	3684	5332	6868			11636	13236
Real_Edge_Cnt		56	57	58	59	60			61	62
Real_Period		???	1616	1568	1648	1536			4768	1600
Real_Edge_Time_NF		490	2106	3674	5312	6848			11616	13226
Crit_Edge_Time		500	2116	3684	5332	6868	8407	9945	11636	13236
Crit_Edge_Cnt		78	79	80	81	82	83	84	85	86
Period		???	1616	1568	1648	1536				1600
***Two_Pulses* (flag0)**	T									
Over_Two_Pulses	T									
Remaining_Synth_Teeth	0					2	1	0		
Real_Prev_Period		???	???	1616	1568	1648			1536	4768
Prev_1_X_Cnt									85	
Prev_Y_1_Cnt										86
Fmult_1_X	(80)									
Fmult_Y_1	(80)									
Fmult_Percent	(80)									
Filter Delay		(10)			(20)					(10)
Gap_Size	(2)									
Gap_Count	(82)									
Gap_Detected_Count										86
IRQ_Count_1		(79)	32847							
IRQ_Count_2		(83)					32851			
IRQ_Count_3		(90)								
IRQ_Count_4		(120)								
IRQ_Edge_Time_1			2116							
IRQ_Edge_Time_2							8407			
IRQ_Edge_Time_3										
IRQ_Edge_Time_4										
IRQ			**Y** IRQ_Count_1				**Y** IRQ_Count_2			**Y** Gap_Count

Figure 4 – Example Interaction of Parameters when in Normal mode (not Backup)

4.8 Features

4.8.1 Engine Stalls

In the past, the host software detected an engine stall condition and would potentially re-initialize the engine position primitive. If EPPE is re-initialized and it is operating in angle mode, TCR2 becomes inconsistent with ***Crit_Edge_Count***. See **Section 4.8.3.3** for details.

4.8.2 Engine Position Information

Other applications (such as spark) use the engine position information provided by the EPPE algorithm (ie, period, edge time, and edge count). The TPU-based engine position primitive maintained two copies of this information in fine and coarse resolutions. The eTPU has 24 bit timers. It is assumed that all range and resolution combinations can be met with this one 24-bit timer. This design frees up the second timer channel (TCR2) for angle or edge time information (see **Section 4.8.3**).

4.8.3 Options for TCR2

There are three modes supported for the second eTPU time base called TCR2: time mode, edge count mode and angle mode.

4.8.3.1 Edge Count Mode

Edge count mode is similar to angle mode, but the EPPE primitive is driving TCR2 without hardware intervention. On every edge, whether real or synthetic, the EPPE writes the edge count shifted left 8, to the TCR2 register. The lower byte of TCR2 is always 0xFF.

TCR2 = ((***Crit_Edge_Count*** << 8) OR 0xFF)

4.8.3.2 Time Mode (TCR2 NOT driven by EPPE)

In time mode, the EPPBE primitive is not involved in driving TCR2. TCR2 is derived from a separate eTPU via the STAC bus.

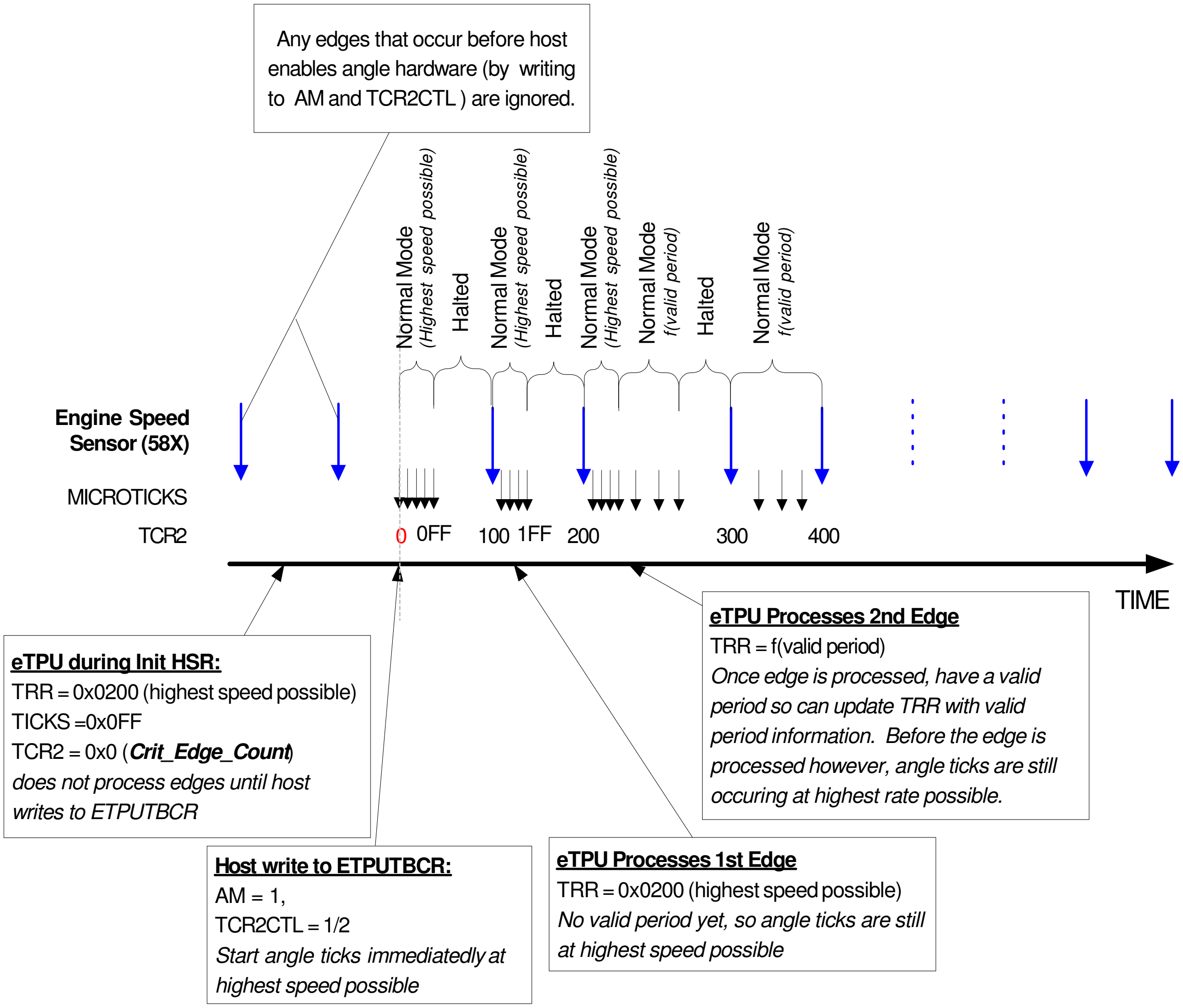
4.8.3.3 Angle Mode

When angle mode is selected, the eTPU HW controls TCR2 in a digital PLL mode that is relative to the speed of the engine. The lower 8 bits of TCR2 (called angle ticks) are incremented at a rate of (last period / 256). The most significant 16 bits of TCR2 track ***Crit_Edge_Count***. TCR2 runs from 0 to 2^{24} and then wraps to 0.

If the engine decelerates, the angle hardware completes generating the 255 angle ticks and halts while waiting for the next edge. When the edge arrives, the hardware continues to generate angle ticks.

When the engine accelerates, the edge occurs before the angle ticks have completed. In this case, the angle hardware generates the angle ticks at a rate of (system clock / 8) until all angle ticks have been generated. This mode is referred to as bursting.

For more details on the angle hardware, refer to Section 4.7 - "EAC-ETPU Angle Counter" - in Freescale's eTPU Block Guide.



Any edges that occur before host enables angle hardware (by writing to AM and TCR2CTL) are ignored.

eTPU during Init HSR:
TRR = 0x0200 (highest speed possible)
TICKS =0x0FF
TCR2 = 0x0 (***Crit_Edge_Count***)
does not process edges until host writes to ETPUTBCR

Host write to ETPUTBCR:
AM = 1,
TCR2CTL = 1/2
Start angle ticks immediately at highest speed possible

eTPU Processes 1st Edge
TRR = 0x0200 (highest speed possible)
No valid period yet, so angle ticks are still at highest speed possible

eTPU Processes 2nd Edge
TRR = f(valid period)
Once edge is processed, have a valid period so can update TRR with valid period information. Before the edge is processed however, angle ticks are still occuring at highest rate possible.

Figure 5: Initialization of TCR2 in Angle Mode

Preventing Errors in TCR2:

Care is taken by the EPPE primitive to keep TCR2 consistent with ***Crit_Edge_Count***. The first step to keep TCR2 consistent with ***Crit_Edge_Count*** is to get it started correctly. **Figure 5** illustrates the steps to initialize TCR2 when in angle mode so that it starts up synchronized with ***Crit_Edge_Count***. See **Section 4.1** for details on the general initialization sequence.

When EPPE is initialized, the angle hardware is disabled (AM bit in ETPUTBCR register = 0) so any edges coming in on TCRCLK pin are ignored. At initialization, EPPE sets up the angle clock tick rate in the trr (the Tick Rate Register) to be its highest possible. When the host does enable the angle hardware and sets the TCR2CTL register appropriately, the angle hardware ticks at this high rate and then halts waiting for an edge.

When the first edge is processed, a valid period is not available. The tick rate is left in its highest possible rate. The angle hardware ticks at this high rate and then halts waiting for the second edge.

When the second edge arrives, the angle hardware starts ticking again at the highest rate. When EPPE processes the edge and calculates a valid period, the tick rate is changed to be appropriate for this valid period. The angle ticks slow down to this more accurate speed. However (assuming steady state), since many of the angle ticks occurred while at high speed, the ticks complete before the third edge arrives and the angle hardware halts awaiting the third tooth.

Monitoring Errors in TCR2:

On the third and subsequent critical edges, EPPE monitors TCR2 and compares it to ***Crit_Edge_Count***. If an error is detected, and TCR2 is not bursting, ***TCR2_Error_Count*** is incremented. An error is determined to be when:

TCR2 < ***Crit_Edge_Count*** - 1

OR

TCR2 > ***Crit_Edge_Count*** + 64 (or 25% of expected period)

Resynchronizing TCR2:

TCR2 is then modified in an attempt to restore synchronization with Crit_EdgeCount. To correct TCR2, the number of angle ticks that are expected by this time is calculated:

expected_ticks = (current time – time of actual edge) / time between angle ticks

TCR2 is then updated with the value:

TCR2 = [***Crit_Edge_Count*** * TICKS_PER_TEETH (256)] OR expected_ticks

Note some error in this calculation is expected.

Controlling TCR2 Roll-Over:

As a further means of keeping TCR2 synchronized with ***Crit_Edge_Count***, the hardware is used to reset TCR2 when it is expected to roll-over to 0x0000. When tooth 0xFFFF is processed, a bit called LAST in the TPR register (Tooth Program Register) is set by the primitive. When this bit is set, the hardware forces TCR2 = 0x0000 on the next tooth event.

4.8.4 Gap Detection Algorithms

4.8.4.1 Percent Period Method :

The eTPU performs Gap Detection on every EPP edge received. If ***Gap_Detect_Method*** = 0 (Percent Period Gap Detection) is selected, then Gap evaluation is as follows:

$N^{-1} > (N + N^{-1} + N^{-2})$ * ***Fmult_Percent***

If evaluation is true, the value of ***Crit_Edge_Count*** is stored into ***Gap_Detected_Count***. **See Figure 6.**

An interrupt will be provided to the Host on every Gap detection. If the Gap detect interrupt is NOT desired by the Host, it can be disabled via ***IRQ_Enable_Mask***.

NOTE: The edge count contained in ***Gap_Detected_Count*** will actually be (***Gap_Size*** + 2) greater than EPPE edge that preceded the actual Gap.

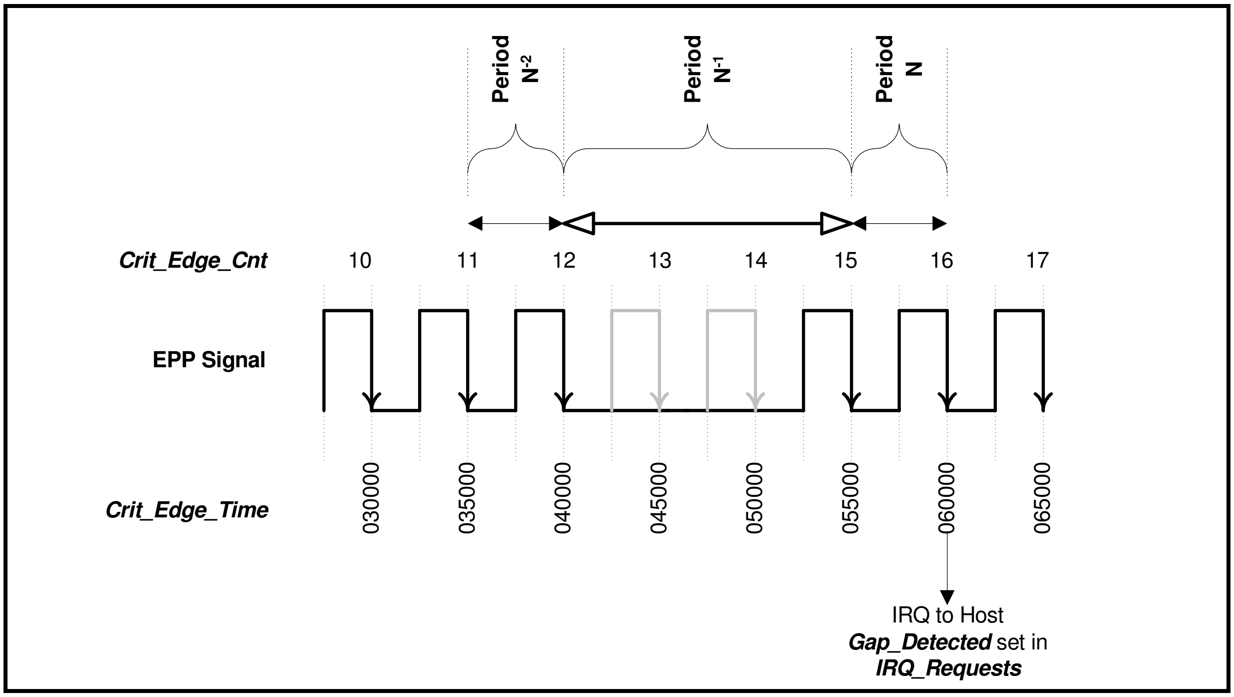


Figure 6 – Percent Period Gap Detection

Fmult_Percent = 0.5

Period N = (current edge time – ***Crit_Edge_Time***)
= (060000– 055000)
= 5000

Period N^{-1} = ***Real_Period***
= (055000 – 040000)
= 15000

Period N^{-2} = ***Real_Prev_Period***
= (040000 – 035000)
= 5000

Gap detected if: $N^{-1} > (N + N^{-1} + N^{-2}) * $ ***Fmult_Percent***

$$15000 > (5000 + 15000 + 5000) * 0.5$$

$$15000 > 25000 * 0.5$$

$$15000 > 12500$$

Gap_Detected_Count = 16

Gap_Detected flag set in ***IRQ_Requests***

Interrupt generated to host CPU

Note: Actual Gap occurred following edge at: (***Gap_Detected_Count*** – 4) or (16 – 4) = 12

4.8.4.2 Threshold Method (1_X / Y_1)

4.8.4.2.1 Prev_1_X_Cnt

The eTPU performs Gap Detection on every EPP edge received. If ***Gap_Detect_Method*** = 1 (1_X__Y_1) is selected, then Gap evaluation is as follows:

N* ***Fmult_1_X*** > N^{-1}.

If evaluation is true, the value of ***Crit_Edge_Count*** is stored into ***Prev_1_X_Cnt***. **See Figure 7.**

4.8.4.2.2 Prev_Y_1_Cnt

The eTPU performs Gap Detection on every EPP edge received. If ***Gap_Detect_Method*** = 1 (1_X__Y_1) is selected, then Gap evaluation is as follows:

N^{-1} * ***Fmult_Y_1*** > N.

If evaluation is true, the value of ***Crit_Edge_Count*** is stored into ***Prev_Y_1_Count***. **See Figure 7.**

An interrupt will be provided to the Host on every Gap detection. If the Gap detect interrupt is NOT desired by the Host, it can be disabled via ***IRQ_Enable_Mask.***

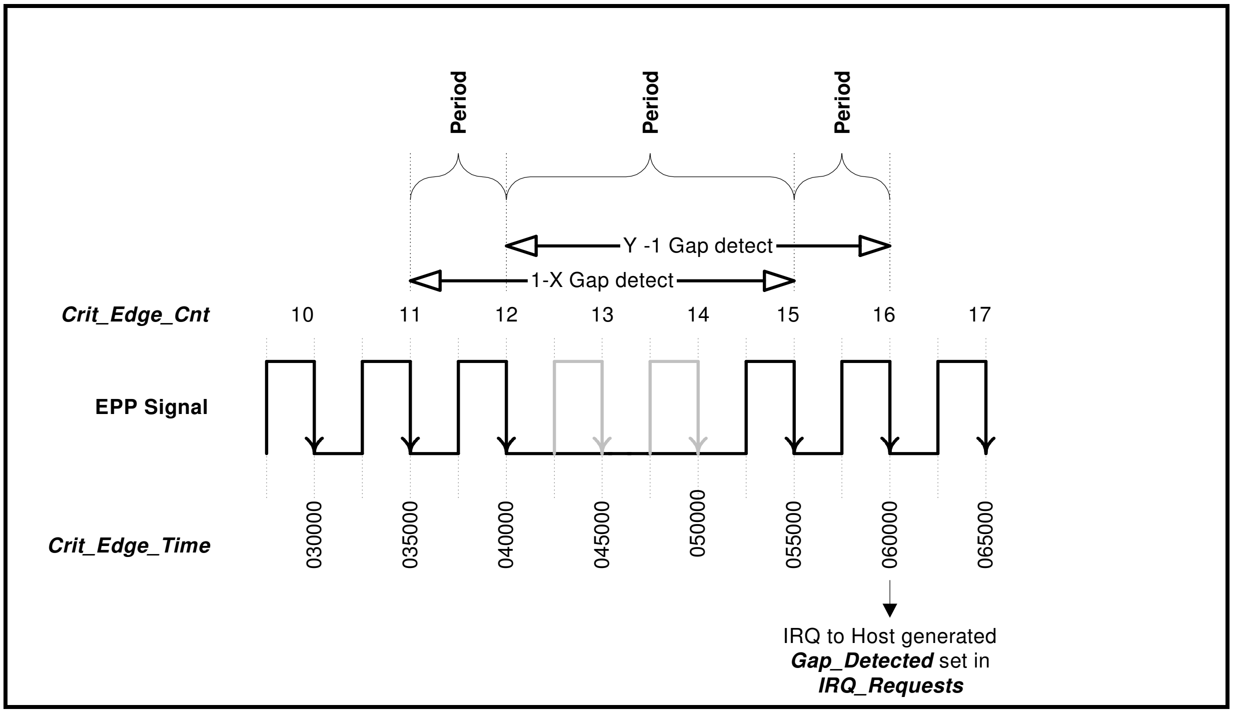


Figure 7 – Threshold (1-X, Y-1) Gap Detection

$$\textbf{\textit{Fmult\_1\_X}} = 0.5$$
$$\textbf{\textit{Fmult\_Y\_1}} = 0.5$$
$$\text{period}^{-1} = \textbf{\textit{Real\_ Prev\_Period}}$$

Gap evaluation is: N* ***Fmult_1_X*** > N^{-1}.

1-X Gap test: (current period * ***Fmult_1_X***) > period^{-1}
(055000 – 040000) * 0.5 > (040000 – 035000)
015000 * 0.5 > 05000
7500 > 5000
Prev_1_X_Count = 15

Gap evaluation is: N^{-1} * ***Fmult_Y_1*** > N.

Y-1 Gap test: (period^{-1} * ***Fmult_Y_1***) > current period
(055000 – 040000) * ***Fmult_Y_1*** > (060000 – 055000)
1500 * 0.5 > 500
7500 > 5000
Prev_Y_1_Count = 16
Gap_Detected flag set in ***IRQ_Requests***
Interrupt generated to host CPU

4.8.4.3 Low Speed Gap Detection Delay

To disable gap detection at very low engine speeds, the user should set (1) the ***LS_Gap_Detect_Disable*** bit. When the engine speed is high enough to permit reliable gap detection, the user should clear (0) the ***LS_Gap_Detect_Disable*** bit. When this occurs, the eTPU will delay gap detection for one more crank pulse to ensure legitimate gap detection. This will be done regardless of the gap detection method (ie, Percent Period or Threshold). See **Figure 8**.

If this delay in gap detection at low engine speeds is not desired, then the user should disable gap detection by clearing the ***Gap_Detect_Enabled*** bit instead.

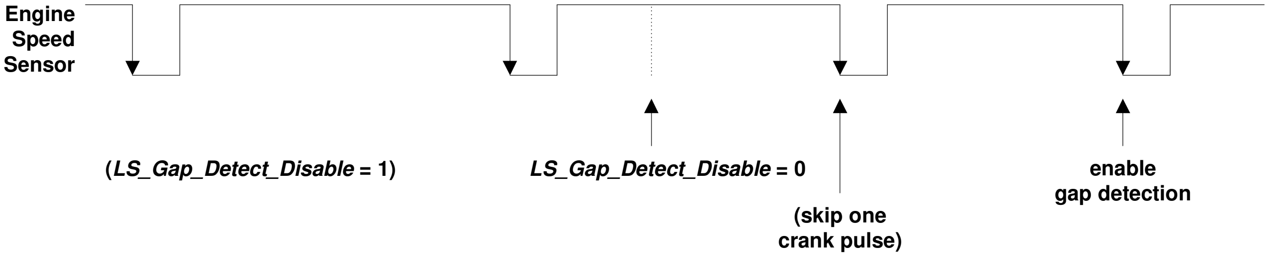


Figure 8 - Low Speed Gap Detection Delay

4.8.5 Input Signal Filtering

4.8.5.1 Recording the Non-Filtered Critical Edge Time

On every critical edge, EPPE calculates the time ***Crit_Edge_Time_NF***. This variable represents the time of the actual edge event, before being filtered by the hardware. The host CPU is responsible for writing the total filter delay into the variable ***Filter_Delay***. EPPE simply subtracts this value from ***Crit_Edge_Time*** to produce ***Crit_Edge_Time_NF***.

Crit_Edge_Time_NF = Crit_Edge_Time – Filter_Delay

4.8.5.2 Filtering with *Min_Period*

There are two options for additional filtering beyond the digital filtering that is provided for all eTPU channels. Both options are based on the value written to ***Min_Period***.

a. When ***Min_Period_Method*** = HW_MIN_PERIOD (1), the hardware in the eTPU is used to filter out any edge that occurs before ***Min_Period*** time since the previous edge.

b. When ***Min_Period_Method*** = SW_MIN_PERIOD (0), the software provides the additional filtering by processing each critical edge event. If an edge occurs before ***Min_Period*** time since the previous edge, that edge is rejected. The flag ***Edge_Rejected*** is set in ***IRQ_Requests***. If this interrupt source is enabled via ***IRQ_Enable_Mask***, the EPPE channel causes an interrupt to the host indicating an edge was rejected.

NOTE: When using angle mode, it is strongly recommended to set ***Min_Period_Method*** = HW_MIN_PERIOD so the hardware can not only filter "noise" pulses from the edge processing, but also prevent TCR2 from incrementing due to the rejected edge. Keep in mind an "edge rejected" interrupt cannot be instigated when using HW_MIN_PERIOD feature.

NOTE: When in backup mode, ***Min_Period_Method*** must be set to SW_MIN_PERIOD, even if TCR2 is in Angle Mode. See **Section 4.9.1** for details.

4.8.6 IRQ_Edge_Count_x

When the host CPU requires edge data for specific edges, the ***IRQ_Edge_Count_x*** variables can be used. The eTPU performs a "greater than or equal to" comparison of ***Crit_Edge_Count*** with ***IRQ_Edge_Count_x*** at every edge event. For example, if the CPU provides an ***IRQ_Edge_Count_x*** of 91 when ***Crit_Edge_Count*** is 92, the eTPU will buffer ***Array_Edge_Time[X]*** stored in the ***Crit_Edge_Array*** corresponding to an edge count of 91 into ***IRQ_Edge_Time_x***. The eTPU will set ***IRQ_Edge_Count_x*** = (***Crit_Edge_Count*** + 0x8000) to prevent further events.

The CPU can determine which edge sent an interrupt by subtracting 0x8000 from ***IRQ_Edge_Count_x***.

If the host CPU has requested an interrupt for this event, indicated by the flag ***IRQ_Count_x*** in ***IRQ_Enable_Mask,*** the flag ***IRQ_Count_x*** is set in ***IRQ_Requests*** and an interrupt to the host is generated.

The comparison between ***IRQ_Edge_Count_x*** and ***Crit_Edge_Count*** is also performed whenever an **Update** HSR (%010) is issued.

NOTE: It is up to the user to avoid writing to ***IRQ_Edge_Count_x*** at the same time the eTPU is attempting to update it!

4.8.7 Linking to other Channels

It is the host CPU's responsibility to determine if any other eTPU channels require a link whenever an engine position edge is processed. A link is a method for communicating with other eTPU channels. For this primitive, it is a way to tell other channels that fresh input data is available. A link is much like an IRQ to another channel. Requested link(s) will be generated on **every** tooth (including "synthetic" teeth).

Two methods are provided for linking to other channels. The first method is similar to that in TPU primitives. ***Start_X*** indicates the first channel to link to while ***Count_X*** indicates how many channels to link to. It is important to note however, the format of the channel number written to ***Start_X*** is different than the TPU implementation. ***Start_X*** must also indicate which eTPU module requires the link. The required format is detailed in “Link Engine Selection” table below.

Bit #:	7	6	5	4	3	2	1	0
	Engine Selection		0[1]	*Channel Number*				

[1] 1: Bit 5 is reserved and must be written to 0.

Engine Selection	Description
00	This Engine
01	Engine 1
10	Engine 2
11	The Other Engine

Table 4: Link Engine Selection

Note: The user must ensure that the combination of ***Start_X*** and ***Count_X*** does not exceed the number of available eTPU channels.

To disable this link method, simply write 0x00 to both ***Count_1*** and ***Count_2*** parameters.

The second link method uses an array of channel numbers. The host writes the channel numbers (along with the eTPU engine, as described above) in the array ***EPPE_Link_Array***. The end of the list is indicated by a 0x00 entry. A pointer to the array is written to the channel relative variable ***Link_Pointer***. If this linking feature is not required, simply write 0x000000 to ***Link_Pointer***

Note: Links to EPPE are not supported. If a link occurs, the EPPE channel number is written to the global variable ***Cause_Of_Exception*** and a global exception is forced.

4.8.8 Interrupt Sources

EPPE contains six conditions that can cause an interrupt. Since all interrupt causes generate the same HW interrupt from the EPPE channel to the host CPU, the following variables are added so the host CPU can determine via SW the source of the interrupt. The following bits represent the interrupt sources in each of the following variables:

Bit 31: ***Edge_Rejected***

Bit 30: ***Gap_Detected***

Bit 29: ***IRQ_Count_1***

Bit 28: ***IRQ_Count_2***

Bit 27: ***IRQ_Count_3***

Bit 26: ***IRQ_Count_4***

Bit 25: ***Change_Dir***

Bit 24: (not used)

The variable ***IRQ_Enable_Mask*** determines which interrupt sources are enabled.

1 = Enabled (will cause interrupt)

0 = Disabled (will not cause interrupt)

If EPPE determines an interrupt condition is met, and the interrupt source is enabled in ***IRQ_Enable_Mask***, it sets the corresponding bit in ***IRQ_Requests*** and causes the HW Interrupt. The host CPU can then interrogate ***IRQ_Requests*** to determine the cause(s) of the interrupt.

1 = Interrupt pending

0 = No interrupt pending

Note: ***IRQ_Requests*** may set multiple bits on one edge event.

Note: ***IRQ_Requests*** may also contain active interrupt indications from past edge events that are not yet cleared by the host.

Note: The host must be sure to handle the case where an interrupt is sent when ***IRQ_Requests*** = 0. This situation can occur if ***IRQ_Requests*** is read just before the interrupt is sent and the interrupt sources are then processed. When the interrupt is then processed, ***IRQ_Requests*** could already have been cleared.

Note: If an interrupt is issued to the host and the host does not clear the corresponding bit in ***IRQ_Requests*** before EPPE executes again (either because of an edge or an **Update** HSR) the interrupt is issued again.

Once the host CPU has processed an interrupt, along with clearing the HW interrupt from the EPPE channel, it needs to clear the pending interrupt source(s) from within EPPE. Refer to **Section 4.8.8.1** for the steps required in clearing these interrupt sources.

4.8.8.1 Clearing Interrupts from EPPE

To clear the interrupt sources in EPPE the host needs to:

1. Disable interrupts.
2. Suppress the EPPE channel from being serviced by setting its priority to 0.

3. Check if the EPPE channel is currently active by reading the appropriate bit in the ETPU Channel Service Status Register (ETPUCSSR). The host must delay until the EPPE channel is not active.
4. Read ***IRQ_Requests*** and save as a local copy.

> **NOTE:** Temporary data is stored in ***IRQ_Requests*** during the processing of a thread. Host must only read this variable when EPPE has been disabled.

5. Clear the appropriate bit for the interrupt source(s) in ***IRQ_Requests***.
6. Re-enable the EPPE channel by restoring its priority level.
7. Enable interrupts.

4.8.9 Update HSR (%010)

The host CPU can issue an **Update** HSR whenever new ***IRQ_Edge_Count_x*** information is available.

When the EPPE receives an **Update** HSR, it performs the following steps:

1. Handles any synthetic or real edge that may be pending.
2. Performs each ***IRQ_Edge_Count_x*** comparison.
3. Interrupts the host if any new interrupt sources are determined.

> **NOTE:** In bi-directional crank mode, the eTPU updates certain parameters on the **non-critical** crank edge. Therefore, if an **Update** HSR is issued and an interrupt received back from the eTPU **during** the bi-directional crank pulse, these parameters will not have been updated yet for the current tooth! (See **Section 4.10**.)

4.8.10 Cam History

The Cam channel input, defined by Global variable ***Cam_Chan_Number***, is sampled every EPP edge. The state of the Cam channel input is then saved in the variable ***Cam_History***, where Bit 0 is the current Cam status and Bit 31 is the state of the Cam input 31 EPP edges previously.

4.8.11 Tachometer Output

The EPPE algorithm is capable of generating a tachometer output on another eTPU channel selected by the user. The tachometer channel is selected by the global ***Tach_Chan_Number***, which may be set to 0x1 - 0xF. **If *Tach_Chan_Number* is set to 0x0, the tachometer feature will be disabled.**

The first transition of the tachometer signal will occur ***Tach_Cnt*** EPP pulses after start-up. It is up to the user to write this initial value of ***Tach_Cnt***. After the first transition, the output signal will transition as determined by the values of ***Tach_High*** and ***Tach_Low***.

If the host CPU needs the current state of the Tachometer output, it can read the pin state directly from the host and does not need to make a request to the eTPU.

NOTE: The tachometer output may be inaccurate when in backup mode (the matches representing the critical edge inputs made-up during the Request_HSR do not all process the tachometer output).

4.8.12 DMA

The EPPE primitive is designed to issue a trigger to the DMA once it completes the processing of each critical edge. A trigger is NOT generated for any rejected edges.

CAUTION: If acceleration occurs in the gap and the real edge arrives before all the synthetic teeth have been processed, a DMA trigger is never issued for these made-up teeth.

When in backup mode, the DMA trigger is issued for every real tooth edge that is processed, not the backup matches. With this implementation, the host can set up the DMA to buffer each ***Real_Edge_Time*** so it can perform its own gap analysis while in backup.

4.9 Backup Mode

When the crank input signal is faulty, information from the cam input can be used instead to provide enough engine position information to enable the system to "limp-home". The host uses the expected relationship between the CAM input and crank input to specify how many crank edges would normally occur before the next cam edge. The host also uses the period between the CAM edges to calculate the expected period of the crank. These two pieces of information are passed to the EPPE primitive when in backup mode.

The information is used by EPPE to create backup matches where the host expected the crank transitions to occur. These matches are processed as if they were real crank inputs. When in backup, EPPE continues to manage TCR2 (when in angle mode or edge count mode) so the expected crank events (or backup matches) are filtered through the system to the other primitives.

Some systems are required to monitor the actual crank input even when in backup mode. If the crank input returns to a healthy state, EPPE can be transitioned back to normal mode by using the **Exit_Backup** HSR.

Thus when EPPE is in backup mode, it must process both the real crank input transitions along with the backup matches. **Table 5** lists the features available in EPPE and which are associated with the actual crank input when in backup mode and which are processed by the backup match.

Feature	Actual Crank Input	Backup Match
Crit_Edge_Time, ***Crit_Edge_Count*** and ***Period***		X
Crit_Edge_Array		X
IRQ_Edge_Count_x & ***IRQ_Edge_Time_x***		X

Feature	Actual Crank Input	Backup Match
Tachometer		X
Links		X
Real_Edge_Time, ***Real_Edge_Count***, ***Real_Period***, ***Real_Prev_Period***, ***Real_Edge_Time_NF***	X	
Min_Period (must be set to SW_MIN_PERIOD)	X	
Gap analysis	X	
Cam_History	X	
DMA trigger	X	
Gap_Count	Not analyzed in backup mode (gap is not filled in when in backup mode)	
Timeout between Critical Edges[1]	Disabled when in backup mode	

[1] Resets ***Period***, ***Real_Period*** and ***Real_Prev_Period*** to 0xFFFFFF if the time between critical edges exceeds the range of the timers.

Table 5: Mapping of EPPE Features to Real Edges or Backup Edges when in Backup Mode

Note: When in backup mode, the ***Min_Period*** **filtering method must be set to SW_MIN_PERIOD**, even if operating in angle mode. This setup is required or the backup match can cancel the min period match and block the processing of any future edges!

Figure 9 illustrates the relationship between the expected crank signal, the CAM input, and the backup matches along with the **Request_Backup** HSRs. Shown is the case where there is a slight delay before the second **Request_Backup** HSR is issued. TCR2 holds with 0xFF in its lower byte awaiting the next backup match. When the **Request_Backup** HSR does arrive, the next backup match is set up. In this case, the match is in the past and occurs immediately. The match logic starts TCR2 counting again. It also sets up the next backup match to occur at time (requested time of the 1st backup edge + ***Period***).

Figure 9 illustrates the case where an acceleration caused the CAM edge to occur earlier than predicted. The **Request_Backup** HSR is issued before all the backup edges from the previous **Request_Backup** HSR have completed. When the HSR is processed, these backup matches are "made up" by recording the current time in the ***Crit_Edge_Array*** for each made-up edge. ***Crit_Edge_Count*** and the upper byte of TCR2 are also incremented by the number of made-up edges.

CAUTION: It is not recommended to use the edge times in ***Crit_Edge_Array*** to calculate the period between two contiguous events. The time between a made-up backup edge and the previous backup edge may be misleadingly small. Also be aware that the period between two made-up edges may be 0. The array can be easily used however to calculate the average period between backup matches.

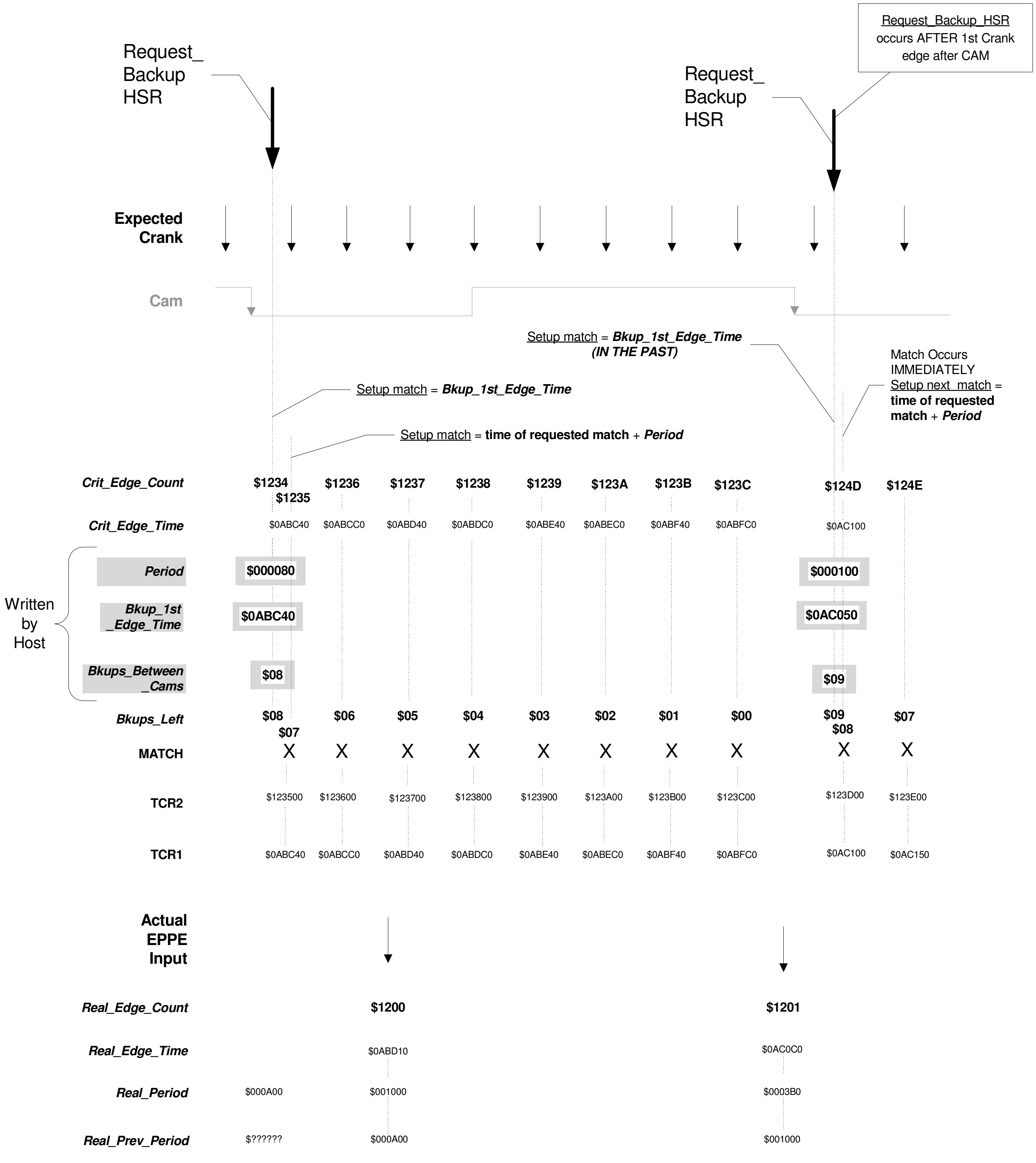
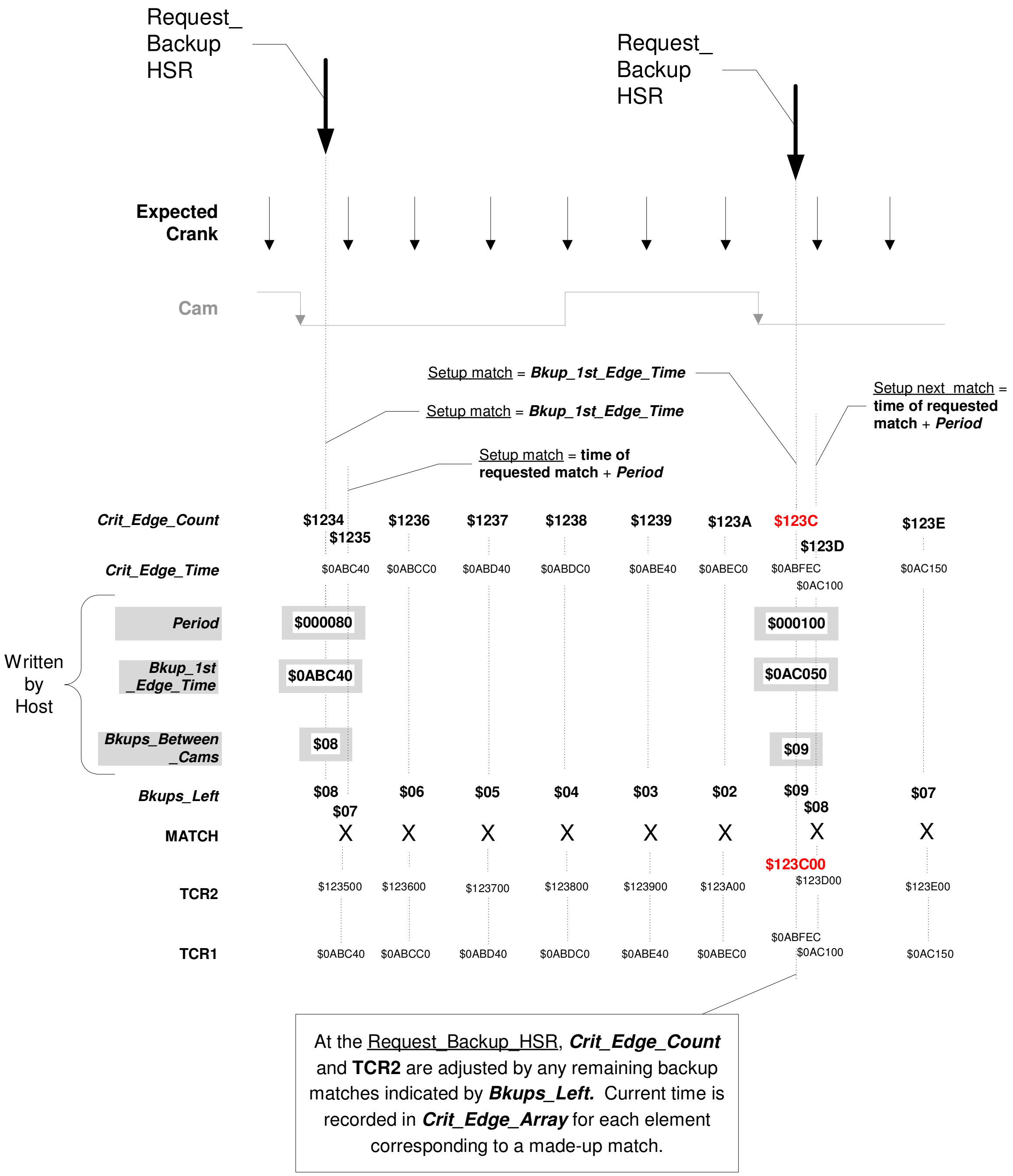


Figure 9: Overview of Backup Mode



At the Request_Backup_HSR, ***Crit_Edge_Count*** and **TCR2** are adjusted by any remaining backup matches indicated by ***Bkups_Left.*** Current time is recorded in ***Crit_Edge_Array*** for each element corresponding to a made-up match.

Figure 10: Acceleration in Backup Mode

4.9.1 Request_Backup HSR

To enter Cam backup mode:

1. Disable the EPPE channel by assigning a priority of 0 (%00) to the CPR field of the ETPUCxCR register and wait until EPPE completes processing.
2. Make sure channel 0 is routed to the eTPU in the SIU.

3. In a single write to the ETPUTBCR register:
    a. **TCR2CTL = %111** so EPPE inputs are no longer routed to the angle hardware in the eTPU, allowing EPPE to manually control TCR2. Inputs are instead routed to the Channel 0 hardware so the edges can still be serviced.
    b. **AM = %1** to ensure the angle mode hardware in the eTPU is enabled.
4. Set ***Min_Period_Method*** = SW_MIN_PERIOD (so the match used for this feature does not interfere with the backup matches).
5. If interrupts from a rejected pulse are not desired, disable them via the ***Edge_Rejected*** flag in ***IRQ_Enable_Mask***.
6. Set ***EPPE_In_Backup*** (FM1) = EPPE_CAM_BACKUP_MODE (1)
7. Set ***Bkups_Left*** = 0 (only on initial **Request_Backup** HSR).
8. Write the absolute time of the first backup edge to ***Bkup_1st_Edge_Time***.
9. Write the requested period between backup matches to ***Period***.
10. Write the total number of desired backup matches from this **Request_Backup** HSR to the next HSR in ***Bkups_Between_Cams***.
11. Issue **Request_Backup** HSR (%100)
12. Enable the EPPE channel by assigning a non-zero priority to the CPR field of the ETPUCxCR register.

Subsequent Request_Backup_HSRs need to:

1. Write the absolute time of the first backup edge to ***Bkup_1st_Edge_Time***.
2. Write the requested period between backup matches to ***Period***.
3. Write the total number of desired backup matches from this **Request_Backup** HSR to the next HSR in ***Bkups_Between_Cams***.
4. Issue **Request_Backup** HSR (%100)

EPPE responds to the **Request_Backup** HSR by executing the following steps:

1. Set up the ***Min_Period*** match to occur immediately so it does not interfere with the backup match.
2. Read the new Period information and writes to the eTPU angle hardware tick rate register (TRR) so the frequency of the microticks is adjusted as soon as possible.
3. Set up the first backup match since the HSR for the absolute time provided in ***Bkup_1st_Edge_Time***.
4. If any backup matches did not complete from the last **Request_Backup** HSR, they are made up by:
    a. Capturing current time into each array element corresponding to a made-up match.
    b. Capturing current time in ***Crit_Edge_Time***.
    c. Incrementing ***Crit_Edge_Count*** by the number of made-up matches (value in ***Bkups_Left***).

    d. Incrementing TCR2 by the number of made-up matches * number of microticks per tooth.

5. Snap-shot ***Bkups_Between_Cams*** into the local variable ***Bkups_Left***.

4.9.2 Exit_Backup HSR

To exit Cam backup mode:

1. Disable the EPPE channel and wait until EPPE completes processing.
2. If TCR2_Options = TIME_MODE or EDGE_COUNT then the ETPUTBCR register does not need to change. If TCR2_Options = ANGLE_MODE then:
    a. Make sure TCRCLK is routed to the eTPU in the SIU.
    b. In a single write to ETPUTBCR:
        ii. Set the angle mode bit (AM), enabling the angle hardware.
        iii. Initialize TCR2CTL field with the critical edge by writing either %001 for rising edges or %010 for falling edges.
3. ***Min_Period_Method*** can be set to either SW_MIN_PERIOD or HW_MIN_PERIOD.
4. Set ***EPPE_In_Backup*** (FM1) = EPPE_CAM_NORMAL_MODE (0)
5. Set ***Bkups_Left*** = 0.
6. Write ***Real_Edge_Time*** to ***Crit_Edge_Time*** so when the next edge is processed, ***Period*** is calculated relative to the last real edge and not the last backup edge. This step is important since ***Period*** is used to calculate the rate of the microticks.
7. Adjust the edge times stored in the ***Crit_Edge_Array*** to prevent a disconnect between the data collected from the backup match and the data from the real edge.
8. While in backup, EPPE performs gap analysis and reports the location of the gap in terms of ***Real_Edge_Count***. Before exiting backup, translate the location of the gap from ***Real_Edge_Count***s to ***Crit_Edge_Count***s and write to ***Gap_Count***.
9. Issue **Exit_Backup** HSR (%011)
10. Enable the EPPE channel.

EPPE responds to an **Exit_Backup** HSR by executing the following step:

1. Set up the timeout match to occur max time from the last real input edge.

4.10 Bi-directional Crank

The EPPE algorithm has the capability to detect reverse crank pulses when a bi-directional crank sensor is employed. There are two mode bits associated with bi-directional crank operation (***Reverse_Detect_Enable***, ***Reverse_Process_Enable***), four counters (***Accum_Edges***, ***Accum_Reverse_Edges***, ***Chg_Dir_Count***, ***Abs_Edge_Count***), and two flags in each array entry (***Dir_Reverse* [X]**, ***Dir_Unknown* [X]**).

Reverse_Detect_Enable is set (1) when the engine speed is below a user-defined threshold (typically about 2000 RPM) and enables reverse crank pulse detection.

Reverse_Process_Enable is set (1) when the engine speed is below a calibratable threshold (lower than the ***Reverse_Detect_Enable*** threshold, typically about 500 RPM) and enables reverse crank pulse processing.

Accum_Edges is a free-running counter that is incremented on every forward crank pulse and decremented on every genuine reverse crank pulse.

Accum_Reverse_Edges is an up-counter (limited to a maximum value of 0xFFFF) that is incremented on every reverse crank pulse detected (genuine or otherwise).

Chg_Dir_Count is incremented whenever it is determined that the crankshaft has changed directions. Whenever ***Chg_Dir_Count*** is incremented, an interrupt will be generated to the host CPU (if enabled via ***IRQ_Enable_Mask***) and indicated with bit ***Change_Dir*** in ***IRQ_Requests***.

Abs_Edge_Count counts from 0 to 119, where 1 and 61 each represent the first tooth after a gap. ***Abs_Edge_Count*** is incremented on forward crank pulses and decremented on genuine reverse crank pulses, being adjusted automatically for gaps in both directions. The value of ***Abs_Edge_Count*** must be synchronized by the CPU before entering reverse crank processing mode (see below); once this mode is entered, the EPPE algorithm will maintain ***Abs_Edge_Count*** without any input from the CPU.

When ***Reverse_Process_Enable*** = 1, ***Dir_Unknown*** **[X]** indicates whether or not a valid direction has been determined for a given array entry; ***Dir_Reverse*** **[X]** indicates what the actual direction is for the same array entry, and may only considered to be valid when ***Dir_Unknown*** **[X]** = 0. When ***Reverse_Process_Enable*** = 0, both ***Dir_Unknown*** **[X]** and ***Dir_Reverse*** **[X]** are always cleared (0).

In uni-directional crank mode (***Reverse_Detect_Enable*** = 0, ***Reverse_Process_Enable*** = 0), only critical edges are captured by the eTPU and the crank is assumed to be moving in a forward direction. ***Accum_Edges*** and ***Abs_Edge_Count*** are incremented on every critical edge, while ***Accum_Reverse_Edges*** and ***Chg_Dir_Count*** are not updated. (Note that ***Abs_Edge_Count*** is automatically adjusted for a gap when count 58 or count 118 is reached.) In this mode, all interrupts from the eTPU to the host CPU are issued on the critical edge. See **Figure 11**.

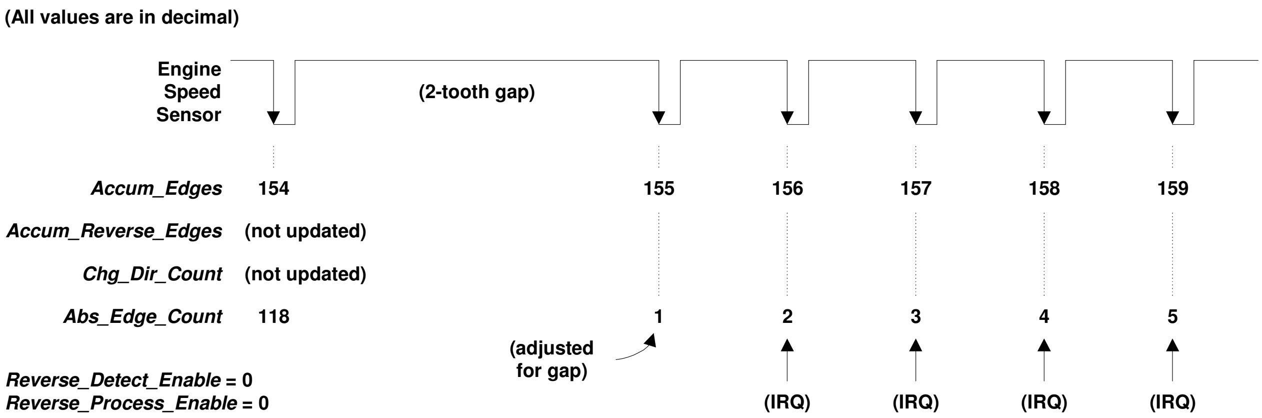


Figure 11: Uni-Directional Crank Mode

In reverse crank detection mode (***Reverse_Detect_Enable*** = 1, ***Reverse_Process_Enable*** = 0), on each non-critical edge, the time since the last critical edge is calculated and stored as ***Dir_Crank_PW*** and the direction of the crank pulse is determined. (On the following critical edge, the value of ***Dir_Crank_PW*** is transferred to ***Buf_Dir_Crank_PW***.) If ***Dir_Crank_PW*** is less than ***Dir_PW_Threshold***, the eTPU will interpret this as a forward crank pulse. If ***Dir_Crank_PW*** is greater than or equal to ***Dir_PW_Threshold***, the eTPU will interpret this as a reverse crank pulse.

In this mode, it is assumed that the crank is moving forwards even when a reverse crank pulse is seen and that a faulty crank sensor is responsible for the reverse crank pulse. Therefore, all the counters are updated (or not updated) as in uni-directional crank mode with the exception of ***Accum_Reverse_Edges***, which is incremented on the non-critical edge of every reverse pulse that is detected. In this mode, all interrupts from the eTPU to the host CPU continue to be issued on the critical edge. See **Figure 12**.

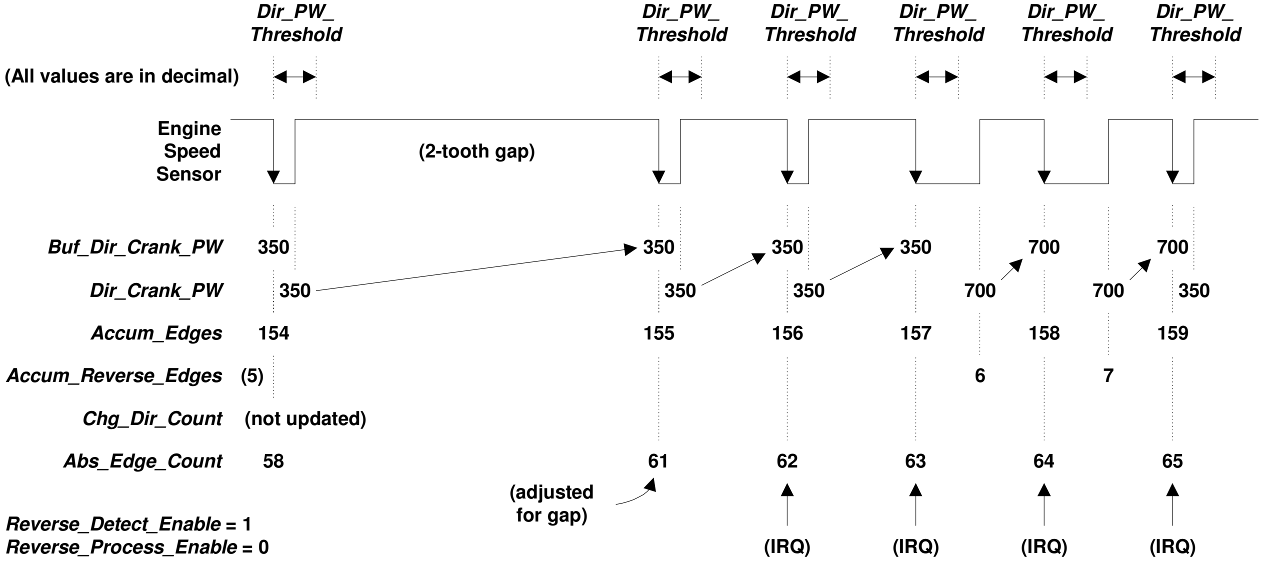


Figure 12 – Reverse Crank Detection Mode

In reverse crank processing mode (***Reverse_Detect_Enable*** = 1, ***Reverse_Process_Enable*** = 1), whenever a reverse crank pulse is detected, the eTPU assumes that the crankshaft is, in fact, moving backwards. In this mode, ***Accum_Edges***, ***Accum_Reverse_Edges***, ***Chg_Dir_Count*** and ***Abs_Edge_Count*** are all updated on every non-critical edge. (Note that the value of ***Accum_Edges*** will now decrement when a reverse crank pulse is received.) All interrupts from the eTPU to the host CPU will also be issued on the non-critical edge. See **Figure 13**.

Also in this mode, ***Dir_Unknown*** **[X]** is set (1) on the critical edge and cleared (0) on the non-critical edge. ***Dir_Reverse*** **[X]** is set to the state of the ***Crank_Backwards*** flag on the non-critical edge (i.e., once crank direction has been determined). Therefore, as long as ***Dir_Unknown*** **[X]** = 1, the state of ***Dir_Reverse*** **[X]** is invalid. See **Figure 14**.

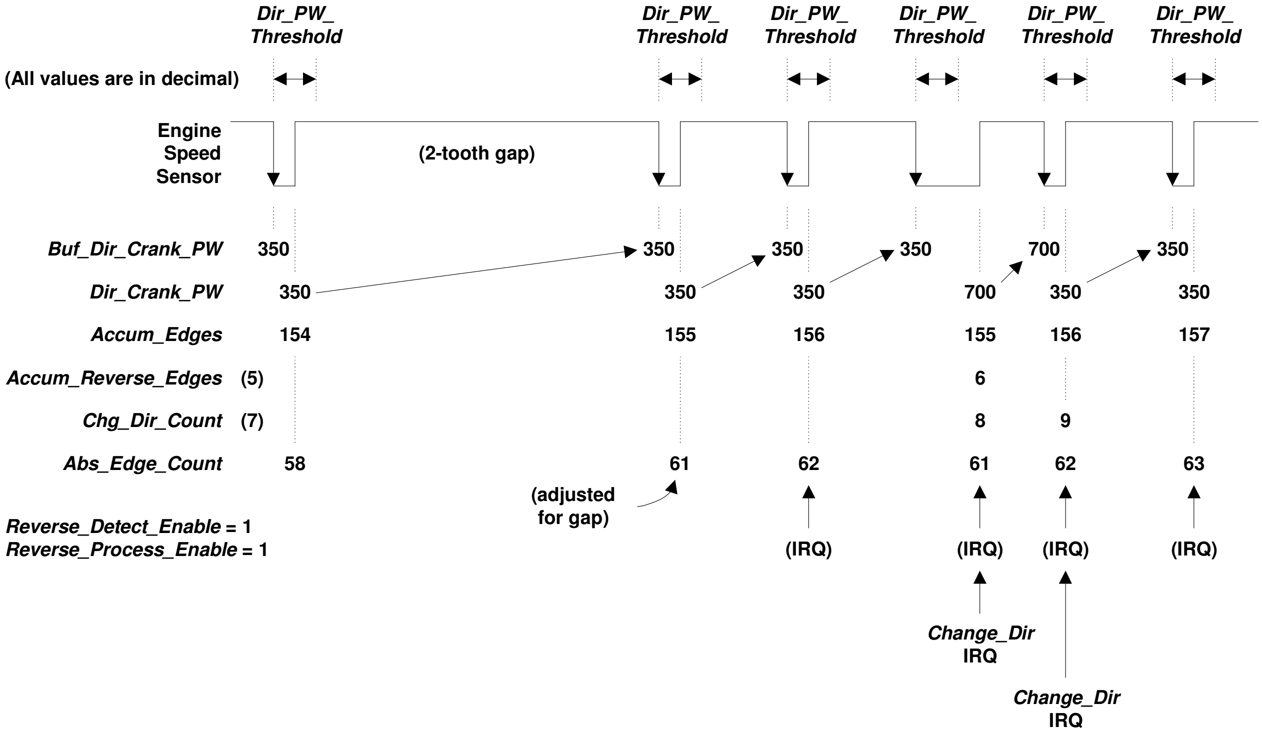


Figure 13 - Reverse Crank Processing Mode

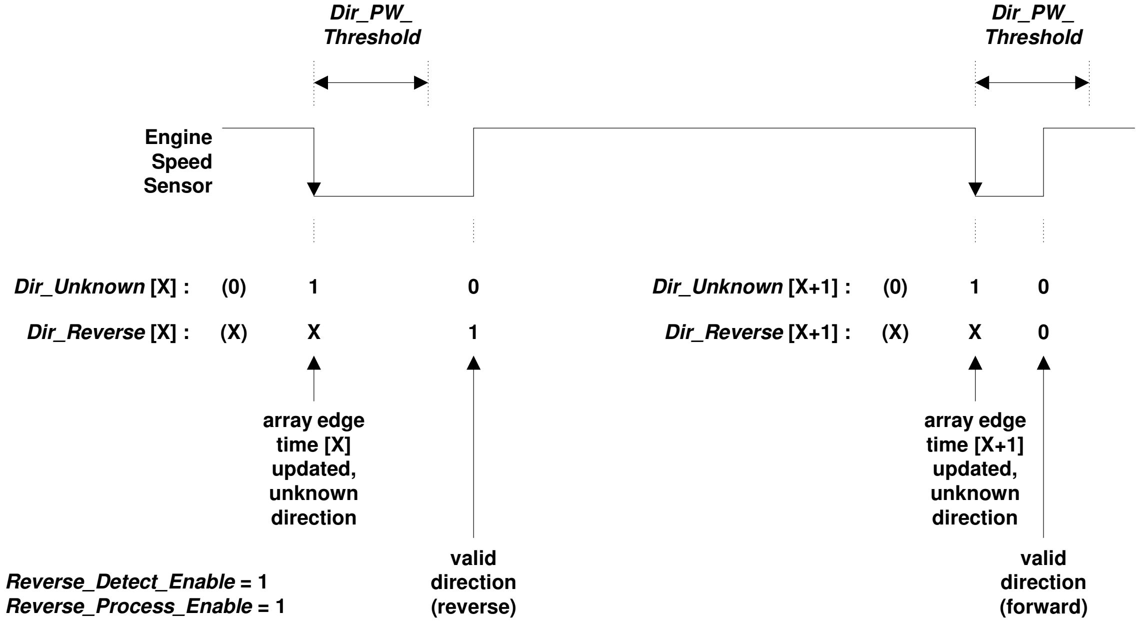

Dir_PW_ Threshold
Engine Speed Sensor
Dir_Unknown [X] : (0) 1 0
Dir_Reverse [X] : (X) X 1
Dir_Unknown [X+1] : (0) 1 0
Dir_Reverse [X+1] : (X) X 0
array edge time [X] updated, unknown direction
valid direction (reverse)
array edge time [X+1] updated, unknown direction
valid direction (forward)
Reverse_Detect_Enable = 1
Reverse_Process_Enable = 1

Figure 14 – Direction Flags in Array Entries

<u>NOTE:</u> After an Initialize HSR has been issued, the EPPE algorithm assumes that the engine was last moving in a forward direction (ie, *Crank_Backwards* = 0).

<u>NOTE:</u> If *Min_Period_Method* = EPPE_HW_MIN_PERIOD (1) while running in bi-directional crank mode, the value of *Min_Period* must not be so large that the non-critical edge is filtered out!

If *Min_Period_Method* = EPPE_SW_MIN_PERIOD (0), the non-critical edge will be accepted as valid regardless of the value of *Min_Period*.

<u>NOTE:</u> The bi-directional crank feature should not be enabled when there are problems with the crank sensor (ie, in Backup mode). Even if it is enabled, the bi-directional crank algorithm will not execute on Backup matches. If enabled, the bi-directional crank algorithm will execute on any real crank teeth that occur in Backup mode, but the results will be inherently unpredictable.

4.11 Limitations

4.11.1 Speed Limitations in Angle Mode

TCR1 is used to clock the tick generation hardware for TCR2 causing a direct relationship between the time base selected for TCR1 and TCR2 when in angle mode. The maximum frequency of TCR2 angle ticks is that of the TCR1 time base. For example, if TCR1 is selected to have a 1 usec time base, the maximum frequency of the TCR2 angle ticks is 1 / 1usec or 1 MHz. This limitation on the angle ticks frequency causes excessive bursting when operating above the RPM limit.

Another limitation is imposed on lower RPMs to ease the calculation of the size of the angle ticks for the TRR register (Tick Rate Register). For this calculation, the Period of the EPPE input is limited to 19 bits.

Table 6 and **Table 7** detail these limitations for a 60x and 24xe system respectively when using 256 angle ticks per tooth.

TCR1 Time Base (usec)	Min RPM Limitation	Max RPM Limitation
0.0625	30.5	62500
0.125	15.3	31250
0.25	7.6	15625
0.5	3.8	7813
1	1.9	3906
2	1.0	1953
4	0.5	977

Table 6: RPM Limitation in <u>60X</u> System (256 angle ticks per tooth)

TCR1 Time Base (usec)	Min RPM Limitation	Max RPM Limitation
0.0625	76.3	156250
0.125	38.1	78125
0.25	19.1	39063
0.5	9.5	19531
1	4.8	9766
2	2.4	4883
4	1.2	2441

Table 7: RPM Limitation in 24Xe System (256 angle ticks per tooth)

4.11.2 Speed Limitations for Gap Detection

Minimum RPM for Gap Detection using 24-bit Math:

58x Input (60 teeth/rev) 2 tooth gap

Assume:

- Five period overflow for Gap Detection **(Percent Period Method)**
- 24 bits a data available = 16,777,216 counts
- Timer resolution = 0.25 usec / count

Total counts available for each of the 5 periods is:

16,777,216 counts / 5 periods (gap=3 periods) = 3,355,443.2 counts/period

Translating this into time for each period is:

3,355,443.2 counts/period * 0.25 usec/count = 0.839 sec./period

Now translate seconds / period to revolutions / minute:

0.839 sec / period * (1 min / 60 sec) * (60 periods or teeth / 1 rev) = 0.839 min / rev
or
1 / 0.839 rev / min = 1.19rev / min

Minimum Speed that can be measured = 1.19 rev / min

Assume:

- Three period overflow for Gap Detection **(Threshold 1_Y, X_1)**
- 24 bits a data available = 16,777,216 counts
- Timer resolution = 0.25 usec / count

Total counts available for each of the 3 periods is:

16,777,216 counts / 3 periods (gap=3 periods) = 5,592,405 counts/period

Translating this into time for each period is:

5,592,405 counts/period * 0.25 usec/count = 1.40 sec./period

Now translate seconds / period to revolutions / minute:

1.40 sec / period * (1 min / 60 sec) * (60 periods or teeth / 1 rev) = 1.40 min / rev
or
1 / 1.40 rev / min = 0.715 rev / min

Minimum Speed that can be measured = 0.715 rev / min

58x Input (60 teeth/rev) <u>4 tooth gap</u>

Assume:

- Seven period overflow for Gap Detection **(Percent Period Method)**
- 24 bits a data available = 16,777,216 counts
- Timer resolution = 0.25 usec / count

Total counts available for each of the 5 periods is:

16,777,216 counts / 7 periods (gap=3 periods) = 2,396,745 counts/period

Translating this into time for each period is:

2,396,745 counts/period * 0.25 usec/count = 0.599 sec./period

Now translate seconds / period to revolutions / minute:

0.599 sec / period * (1 min / 60 sec) * (60 periods or teeth / 1 rev) = 0.599 min / rev
or
1 / 0.599 rev / min = 1.67 rev / min

Minimum Speed that can be measured = 1.67 rev / min

Assume:

- Five period overflow for Gap Detection **(Threshold 1_Y, X_1)**
- 24 bits a data available = 16,777,216 counts
- Timer resolution = 0.25 usec / count

Total counts available for each of the 3 periods is:

16,777,216 counts / 5 periods (gap=3 periods) = 3,355,443 counts/period

Translating this into time for each period is:

3,355,443 counts/period * 0.25 usec/count = 0.839 sec./period

Now translate seconds / period to revolutions / minute:

0.839 sec / period * (1 min / 60 sec) * (60 periods or teeth / 1 rev) = 0.839 min / rev
or
1 / 0.839 rev / min = 1.19 rev / min

Minimum Speed that can be measured = 1.19 rev / min

5.0 Primitive Resources & Timing

EPPE Primitive Phase		**µcycles**
Execution Threads		
Initialization HSR		**33**
Shutdown HSR		**8**
Request_Backup_HSR[1]		**237**
Exit_Backup_HSR		**6**
Update HSR (No Pulse Pending, 1 IRQ match)		**48**
Unused Link		**6**
Unused HSR	No ***Min_Period*** pending	**6**
Unused HSR	w/ ***Min_Period*** pending	**11**
Critical Edge Processing[2] (Normal mode)	Time Mode	**250**
	Edge Count Mode	**250**
	Angle Mode	**288**
Critical Edge Processing[3] (Backup mode)	Time Mode	**95**
	Edge Count Mode	**95**
	Angle Mode	**95**
Backup Match[4]	Time Mode	**156**
	Edge Count Mode	**156**
	Angle Mode	**181**
Non-Critical Edge Processing (Bi-directional crank mode only)		**68**
Features		
Gap Analysis	Percent Period	18 additional
	Threshold	16 additional
TCR2 Error Correction (Angle mode only)		35 additional to correct
Links – 4 individual links in array, 2 link sets of 4 each		58 additional
Tachometer		11 additional
One IRQ Count Match		14 additional when match
Period Calculation on 1st Tooth After Gap		2 additional
Period Calculation on 2nd Tooth After Gap		7 additional
Period Averaging on Every Tooth		0 additional
Make up 2 teeth		44 additional

[1] Includes making up of 10 teeth which corresponds to 8000 RPM / sec acceleration at 600 RPM, one *IRQ_Edge_Count_x* match on made-up teeth.

[2] Includes: 1 *IRQ_Edge_Count_x* match on tooth after gap, along with period after the gap calculation Does NOT include: TCR2 error correction, links, tachometer, making up of teeth or gap analysis

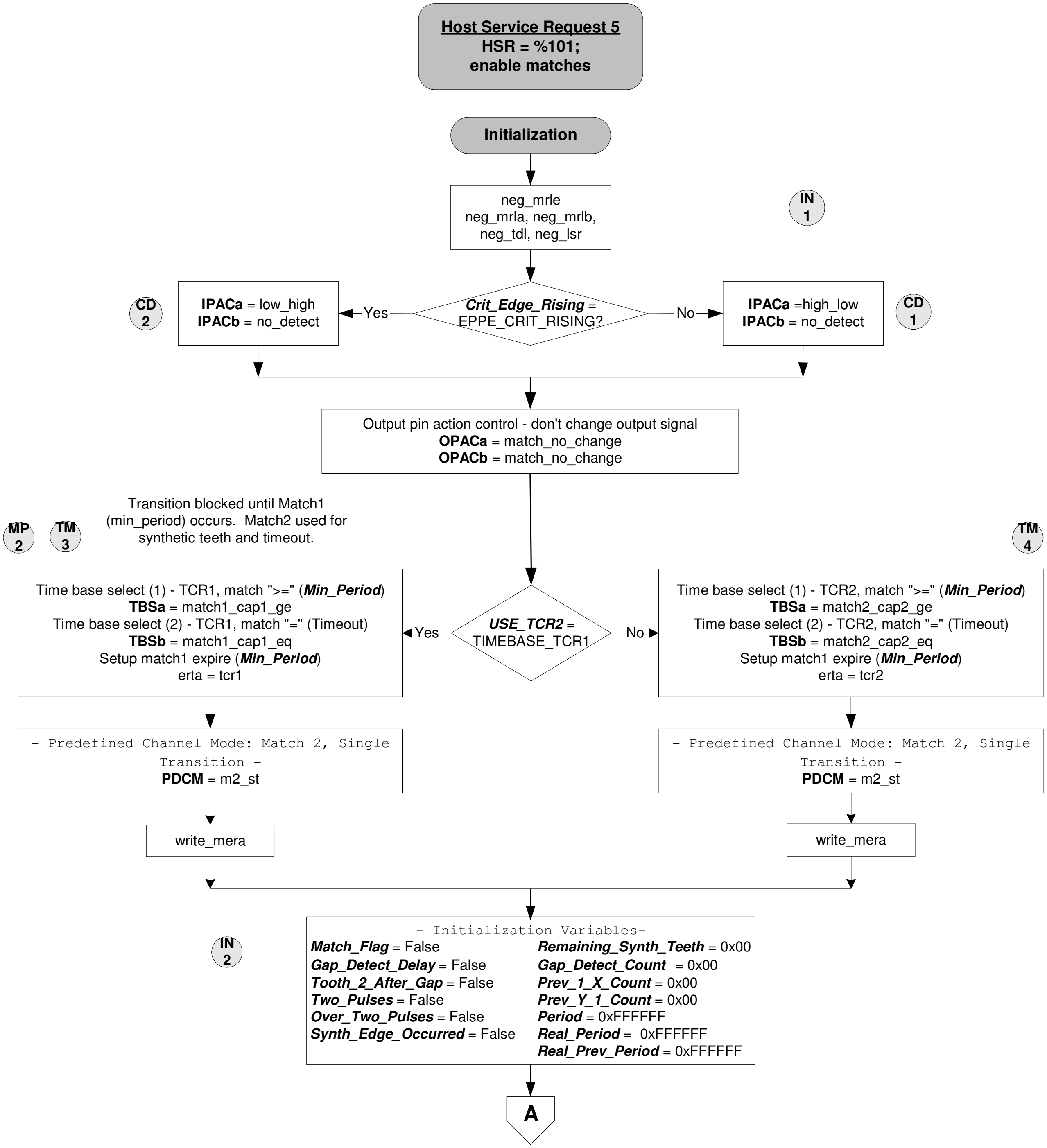
[3] Does NOT include: gap analysis

[4] Includes: 1 *IRQ_Edge_Count_x* match. Does NOT include: TCR2 error correction, links, or tachometer

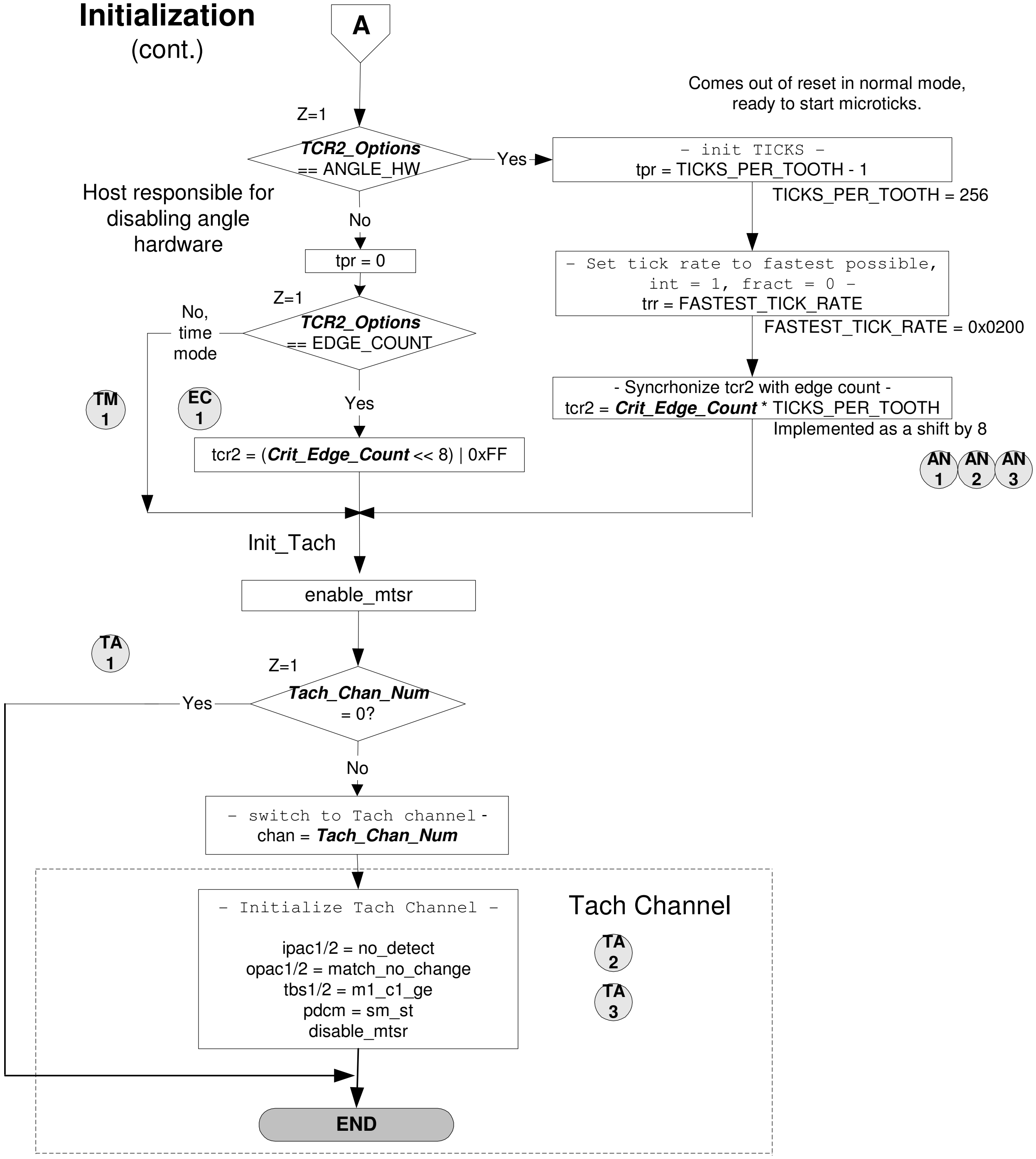
Size of Parameter RAM	=	**32 + (Size of array) words** (32 bits)
Size of Code	=	**732 words** (32 bits)

6.0 Flowcharts

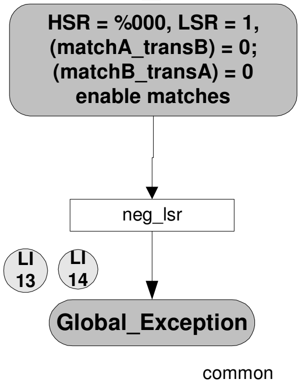
X Indicates the test case associated with the flow chart. Details on the test case are included in the software test plan document entitled "STP_5408.doc" available in CM Synergy in the kok_pt1 database, under ComplexIO/eTPU/Primitives/CamCrank/Eppe/Verification/VerificationTestPlan.


Host Service Request 5
HSR = %101;
enable matches
Initialization
neg_mrle
neg_mrla, neg_mrlb,
neg_tdl, neg_lsr
IN 1
Crit_Edge_Rising = EPPE_CRIT_RISING?
Yes
No
CD 2
IPACa = low_high
IPACb = no_detect
IPACa =high_low
IPACb = no_detect
CD 1
Output pin action control - don't change output signal
OPACa = match_no_change
OPACb = match_no_change
Transition blocked until Match1 (min_period) occurs. Match2 used for synthetic teeth and timeout.
MP 2
TM 3
TM 4
USE_TCR2 = TIMEBASE_TCR1
Yes
No
Time base select (1) - TCR1, match ">=" (Min_Period)
TBSa = match1_cap1_ge
Time base select (2) - TCR1, match "=" (Timeout)
TBSb = match1_cap1_eq
Setup match1 expire (Min_Period)
erta = tcr1
Time base select (1) - TCR2, match ">=" (Min_Period)
TBSa = match2_cap2_ge
Time base select (2) - TCR2, match "=" (Timeout)
TBSb = match2_cap2_eq
Setup match1 expire (Min_Period)
erta = tcr2
- Predefined Channel Mode: Match 2, Single Transition -
PDCM = m2_st
- Predefined Channel Mode: Match 2, Single Transition -
PDCM = m2_st
write_mera
write_mera
IN 2
- Initialization Variables-
Match_Flag = False
Gap_Detect_Delay = False
Tooth_2_After_Gap = False
Two_Pulses = False
Over_Two_Pulses = False
Synth_Edge_Occurred = False
Remaining_Synth_Teeth = 0x00
Gap_Detect_Count = 0x00
Prev_1_X_Count = 0x00
Prev_Y_1_Count = 0x00
Period = 0xFFFFFF
Real_Period = 0xFFFFFF
Real_Prev_Period = 0xFFFFFF
A

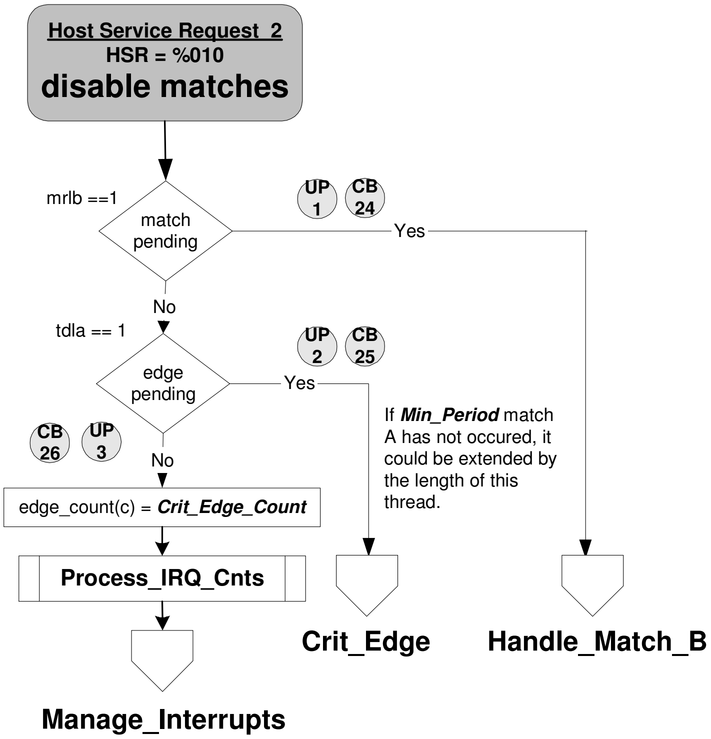
Initialization
(cont.)



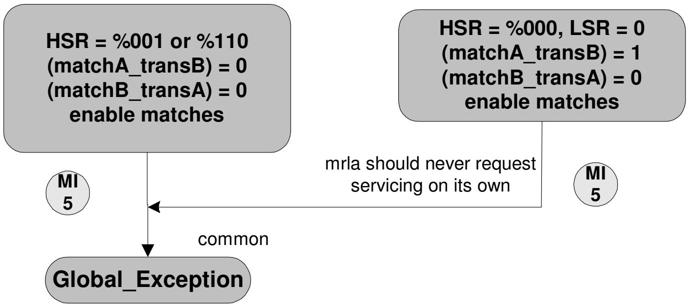
Link with no *Min_Period* Match
Link with *Min_Period* Match



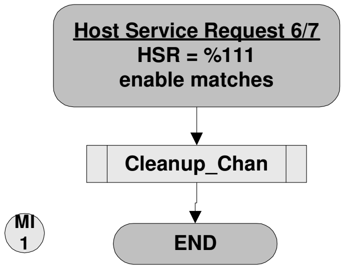
Update

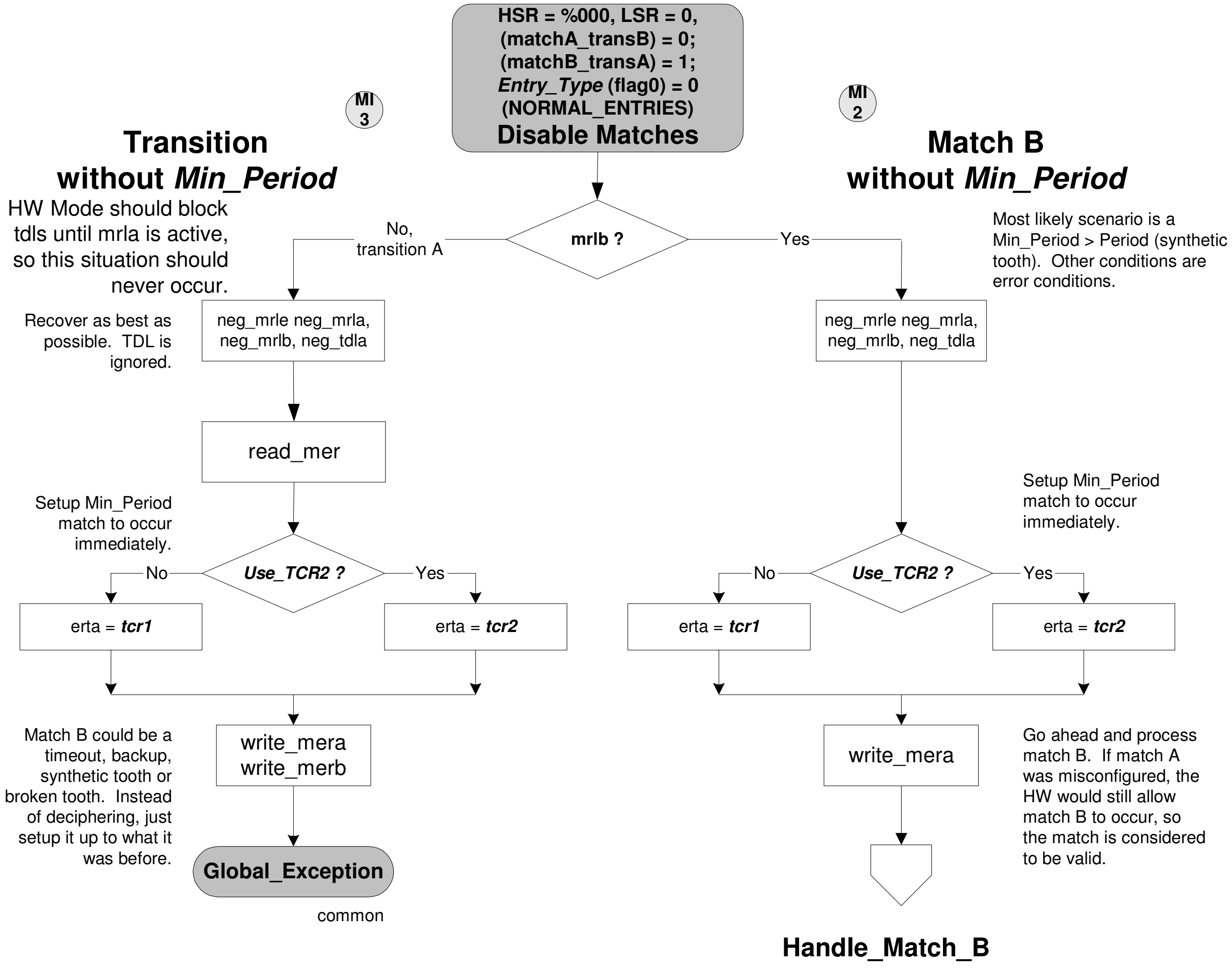


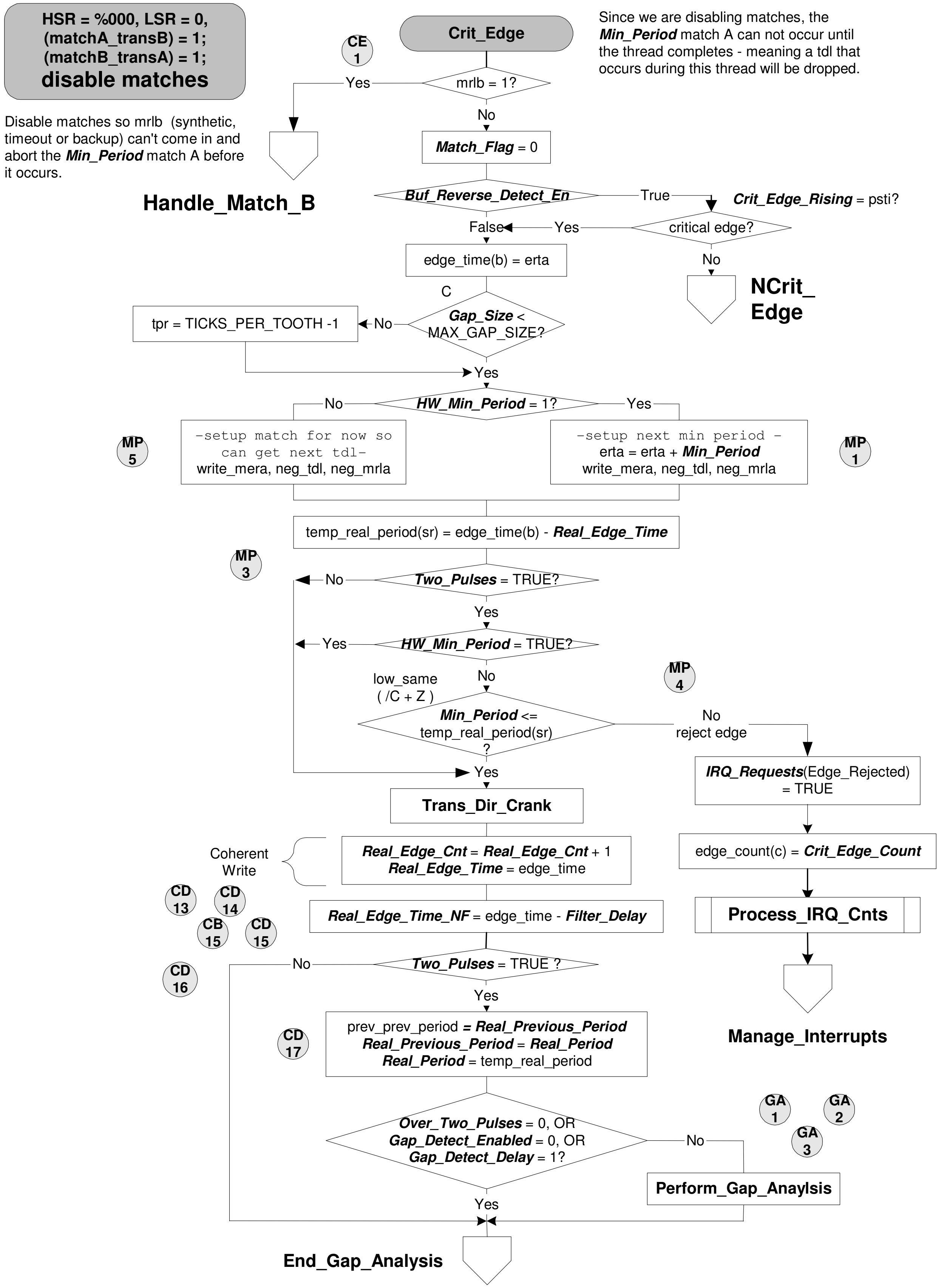
Unused HSRs
(HSR = 1,3,4,6)



Shutdown

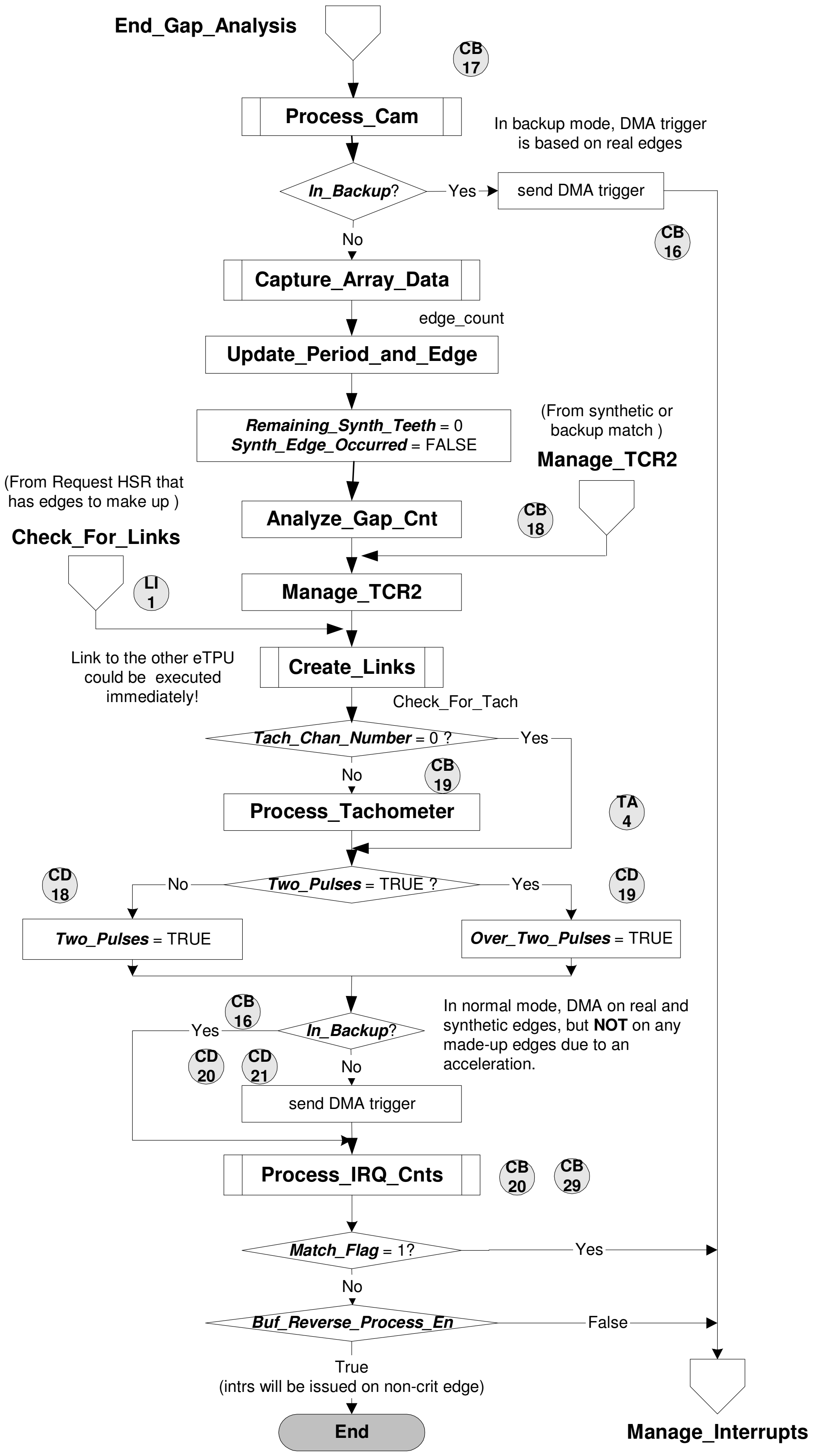


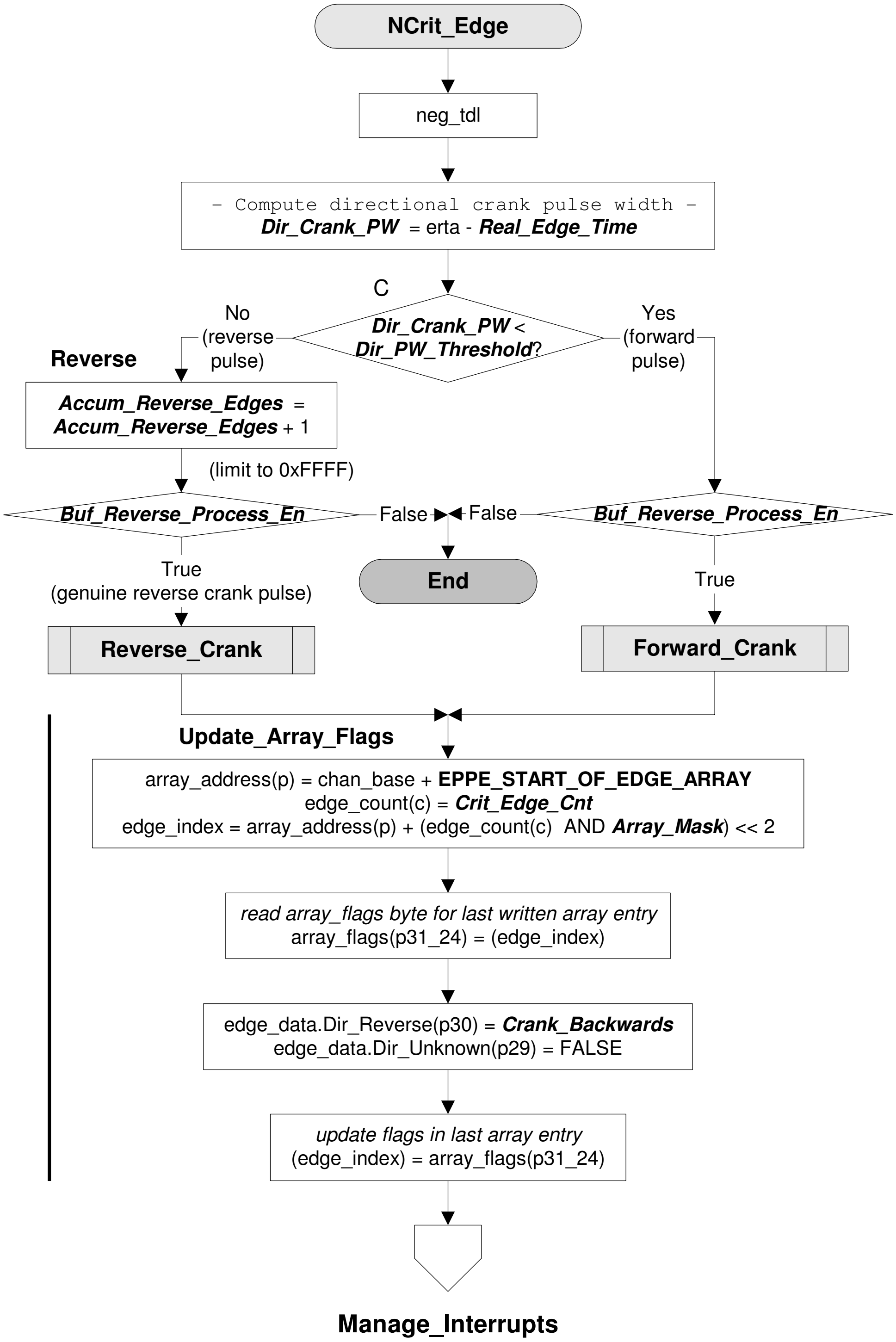

HSR = %000, LSR = 0,
(matchA_transB) = 0;
(matchB_transA) = 1;
Entry_Type (flag0) = 0
(NORMAL_ENTRIES)
Disable Matches
MI 3
MI 2
Transition
without Min_Period
HW Mode should block tdls until mrla is active, so this situation should never occur.
Match B
without Min_Period
Most likely scenario is a Min_Period > Period (synthetic tooth). Other conditions are error conditions.
mrlb ?
No, transition A
Yes
Recover as best as possible. TDL is ignored.
neg_mrle neg_mrla, neg_mrlb, neg_tdla
neg_mrle neg_mrla, neg_mrlb, neg_tdla
read_mer
Setup Min_Period match to occur immediately.
Setup Min_Period match to occur immediately.
Use_TCR2 ?
No
Yes
erta = tcr1
erta = tcr2
Use_TCR2 ?
No
Yes
erta = tcr1
erta = tcr2
Match B could be a timeout, backup, synthetic tooth or broken tooth. Instead of deciphering, just setup it up to what it was before.
write_mera
write_merb
Global_Exception
common
write_mera
Go ahead and process match B. If match A was misconfigured, the HW would still allow match B to occur, so the match is considered to be valid.
Handle_Match_B

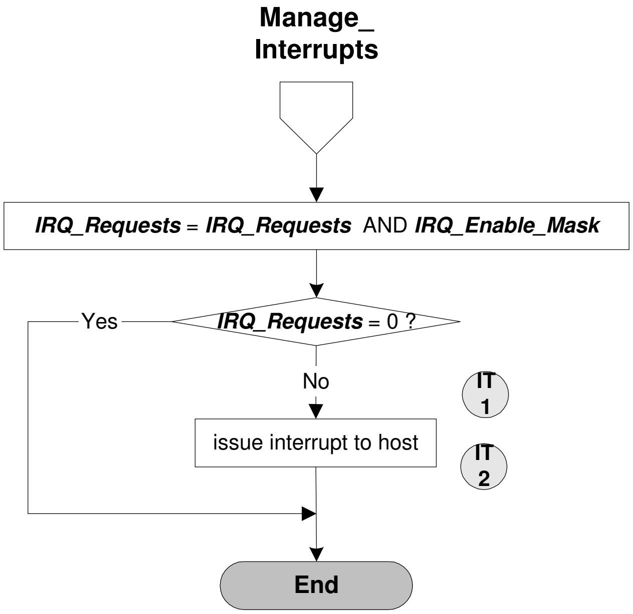


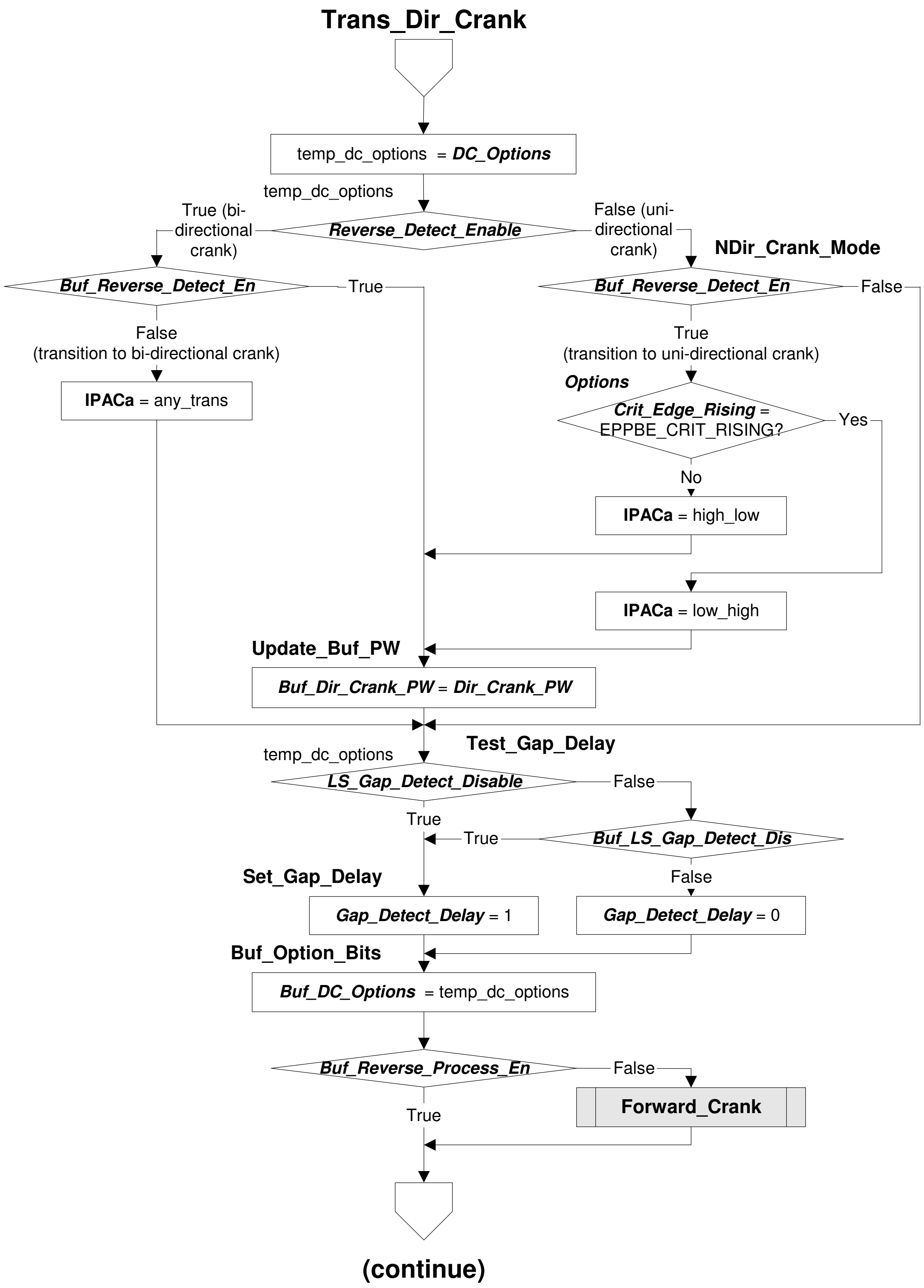
Since we are disabling matches, the ***Min_Period*** match A can not occur until the thread completes - meaning a tdl that occurs during this thread will be dropped.

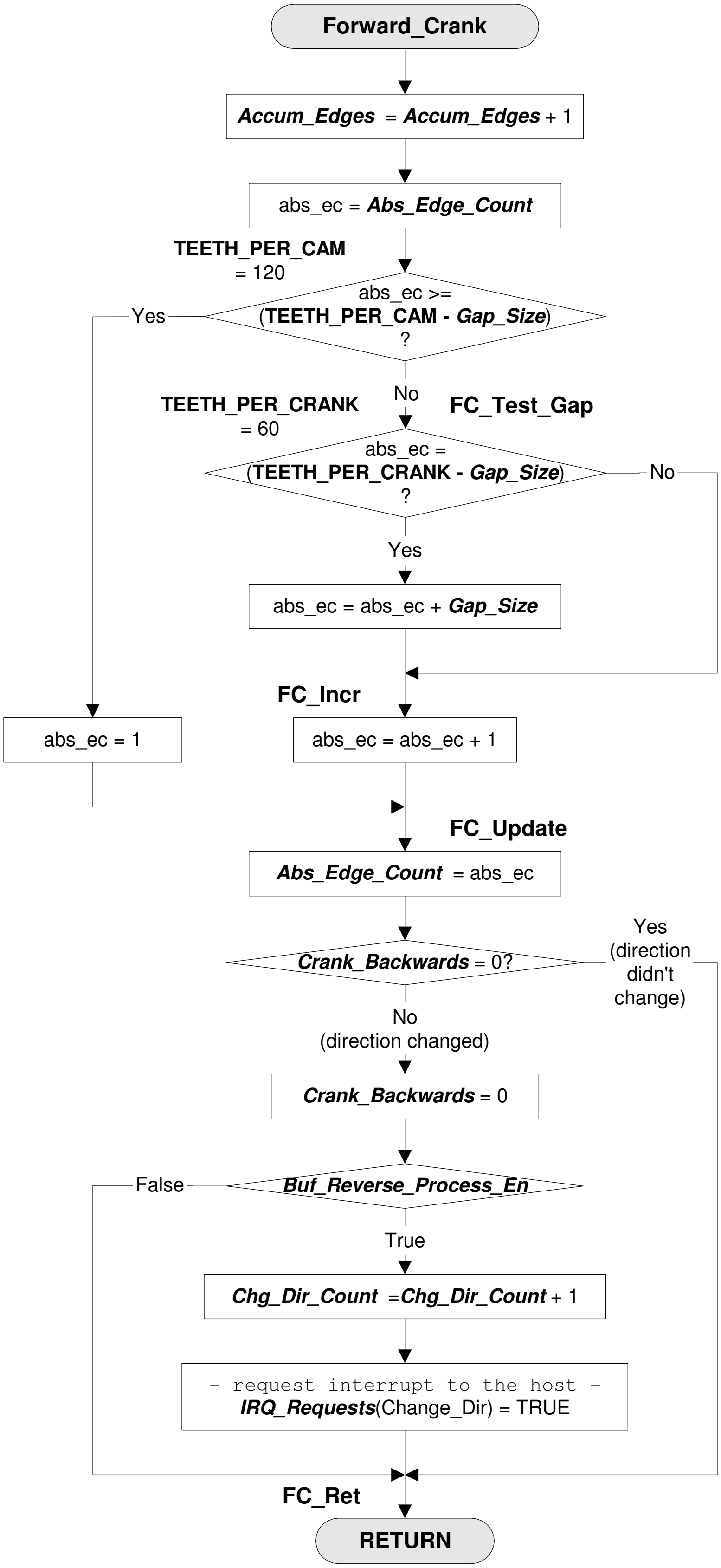
Disable matches so mrlb (synthetic, timeout or backup) can't come in and abort the ***Min_Period*** match A before it occurs.

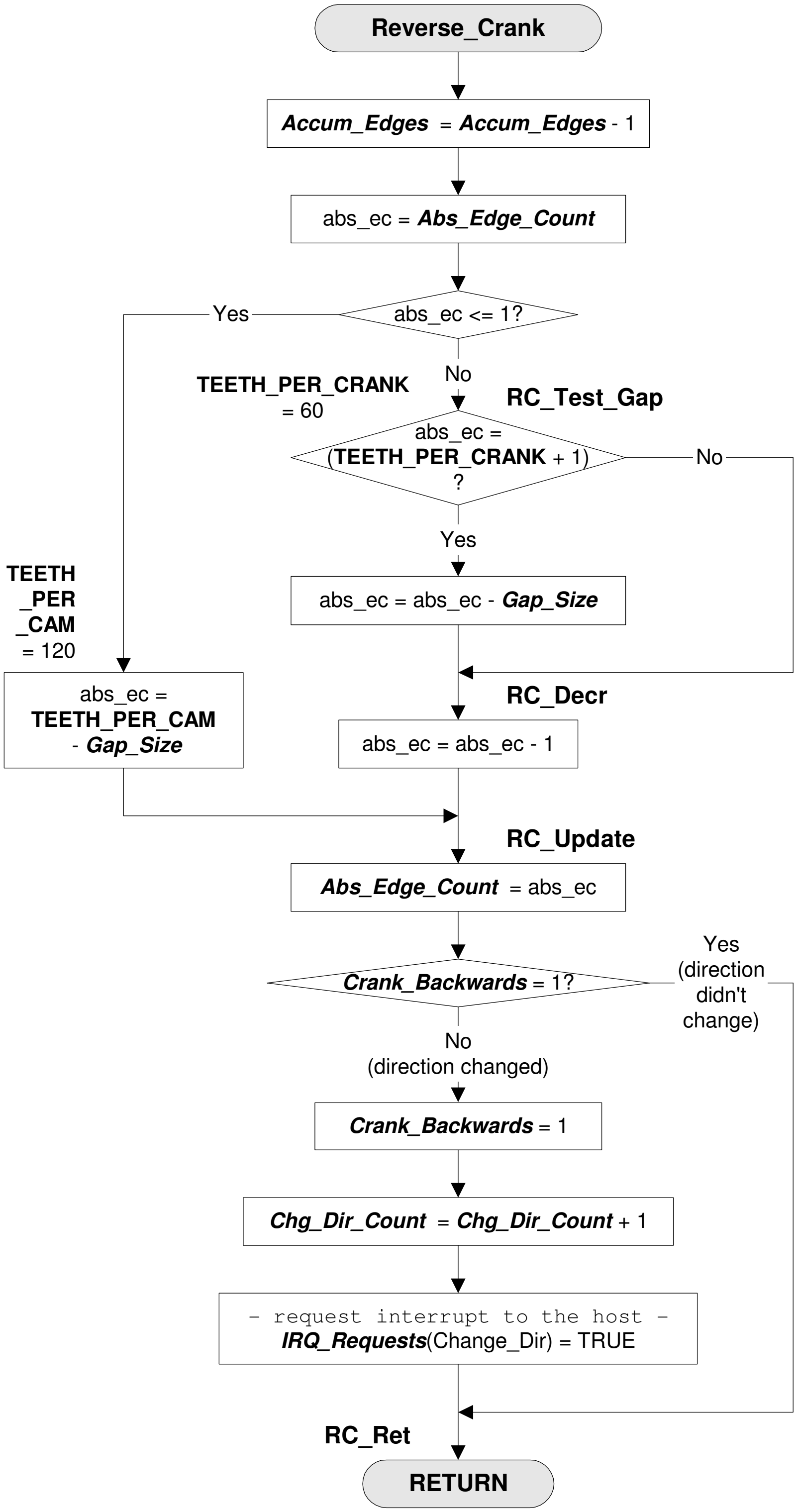

End_Gap_Analysis
CB 17
Process_Cam
In backup mode, DMA trigger is based on real edges
In_Backup?
Yes
send DMA trigger
CB 16
No
Capture_Array_Data
edge_count
Update_Period_and_Edge
Remaining_Synth_Teeth = 0
Synth_Edge_Occurred = FALSE
(From synthetic or backup match)
Manage_TCR2
(From Request HSR that has edges to make up)
Analyze_Gap_Cnt
CB 18
Check_For_Links
LI 1
Manage_TCR2
Link to the other eTPU could be executed immediately!
Create_Links
Check_For_Tach
Tach_Chan_Number = 0 ?
Yes
No
CB 19
Process_Tachometer
TA 4
CD 18
No
Two_Pulses = TRUE ?
Yes
CD 19
Two_Pulses = TRUE
Over_Two_Pulses = TRUE
CB 16
Yes
In_Backup?
In normal mode, DMA on real and synthetic edges, but NOT on any made-up edges due to an acceleration.
CD 20
CD 21
No
send DMA trigger
Process_IRQ_Cnts
CB 20
CB 29
Match_Flag = 1?
Yes
No
Buf_Reverse_Process_En
False
True
(intrs will be issued on non-crit edge)
End
Manage_Interrupts

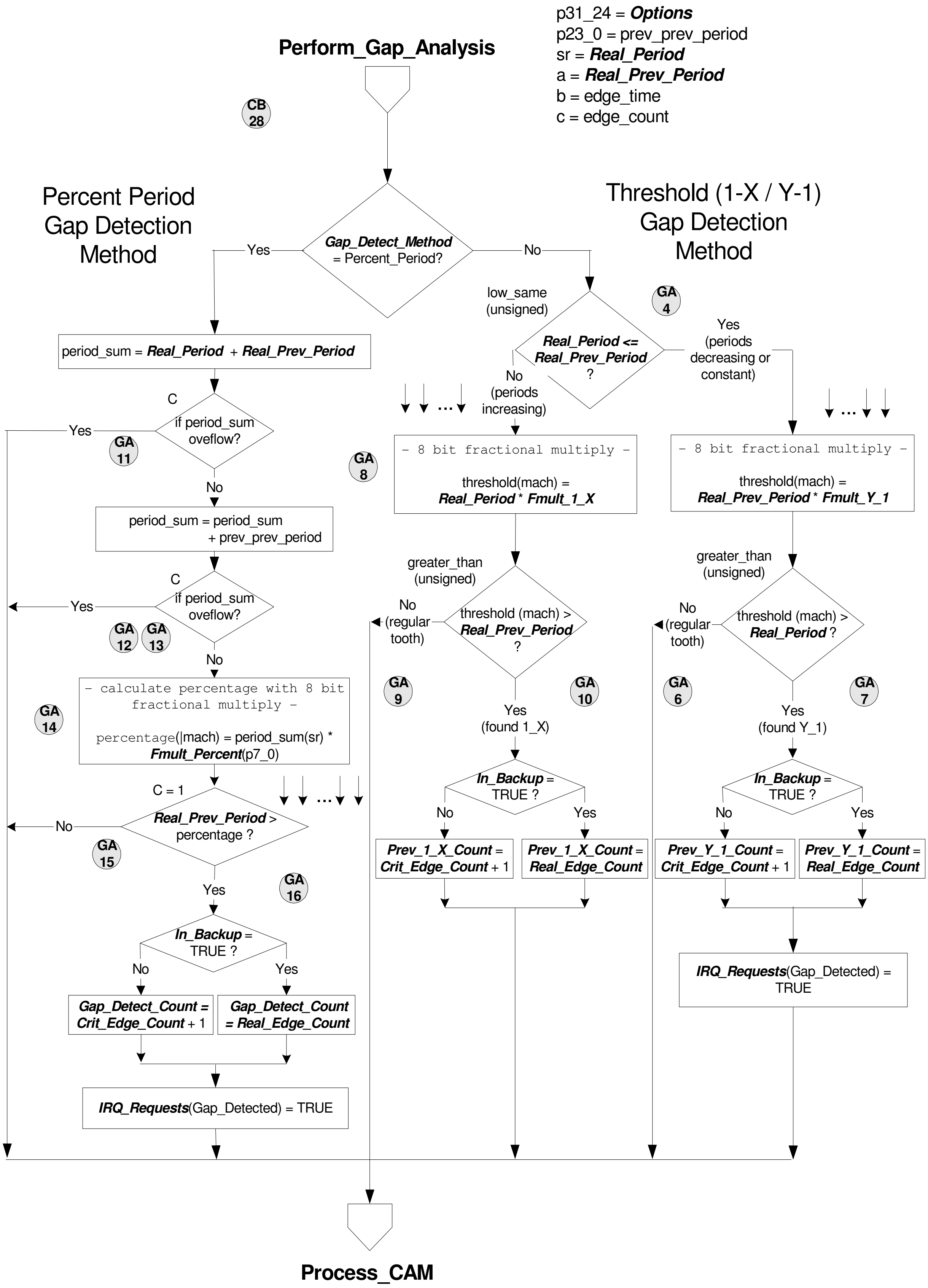

NCrit_Edge
neg_tdl
- Compute directional crank pulse width -
Dir_Crank_PW = erta - Real_Edge_Time
C
Dir_Crank_PW < Dir_PW_Threshold?
No (reverse pulse)
Yes (forward pulse)
Reverse
Accum_Reverse_Edges = Accum_Reverse_Edges + 1
(limit to 0xFFFF)
Buf_Reverse_Process_En
False
False
Buf_Reverse_Process_En
True (genuine reverse crank pulse)
End
True
Reverse_Crank
Forward_Crank
Update_Array_Flags
array_address(p) = chan_base + EPPE_START_OF_EDGE_ARRAY
edge_count(c) = Crit_Edge_Cnt
edge_index = array_address(p) + (edge_count(c) AND Array_Mask) << 2
read array_flags byte for last written array entry
array_flags(p31_24) = (edge_index)
edge_data.Dir_Reverse(p30) = Crank_Backwards
edge_data.Dir_Unknown(p29) = FALSE
update flags in last array entry
(edge_index) = array_flags(p31_24)
Manage_Interrupts


Manage_ Interrupts
IRQ_Requests = IRQ_Requests AND IRQ_Enable_Mask
IRQ_Requests = 0 ?
Yes
No
issue interrupt to host
IT 1
IT 2
End

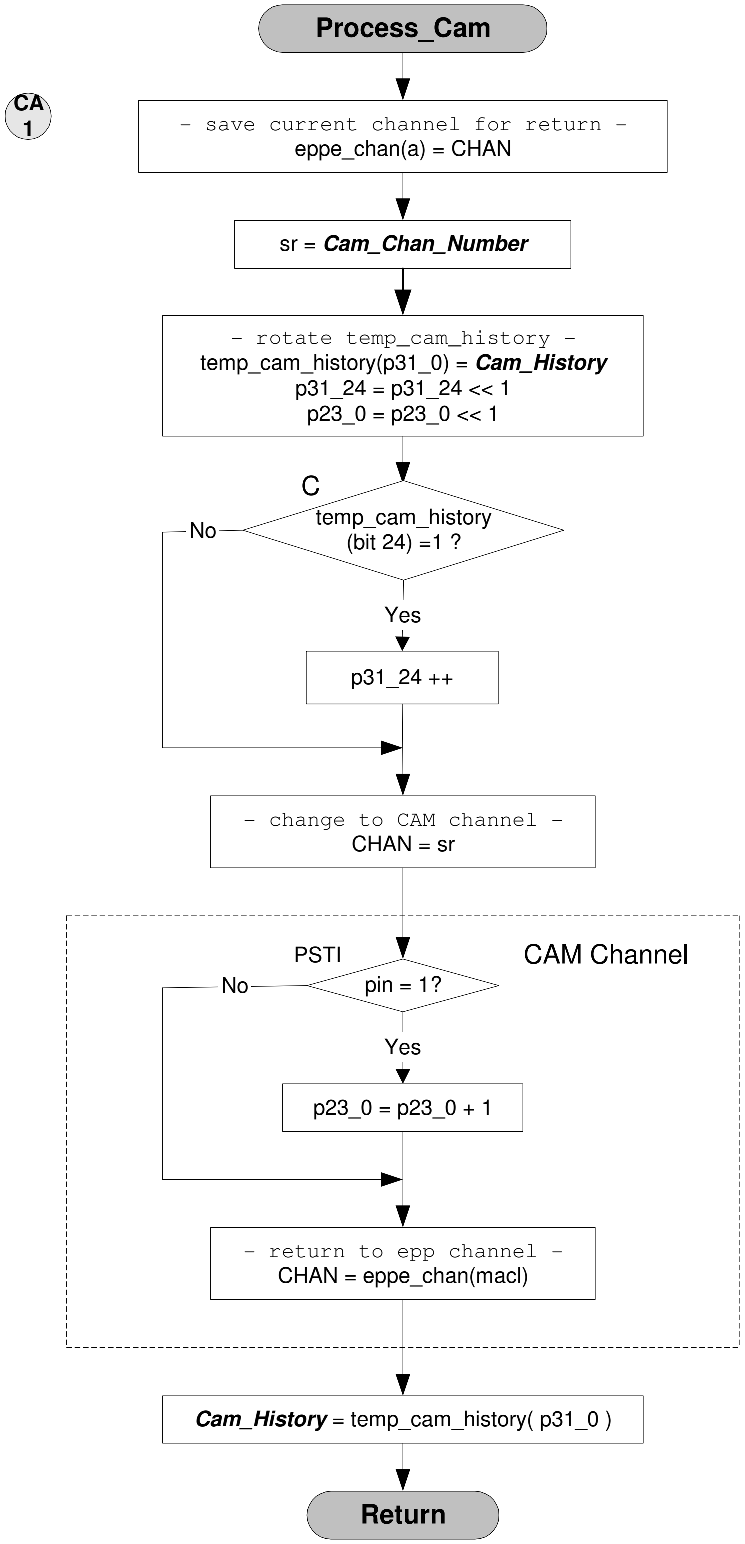

Trans_Dir_Crank
temp_dc_options = DC_Options
temp_dc_options
Reverse_Detect_Enable
True (bi-directional crank)
False (uni-directional crank)
NDir_Crank_Mode
Buf_Reverse_Detect_En
True
False
(transition to bi-directional crank)
IPACa = any_trans
Buf_Reverse_Detect_En
False
True
(transition to uni-directional crank)
Options
Crit_Edge_Rising = EPPBE_CRIT_RISING?
Yes
No
IPACa = high_low
IPACa = low_high
Update_Buf_PW
Buf_Dir_Crank_PW = Dir_Crank_PW
Test_Gap_Delay
temp_dc_options
LS_Gap_Detect_Disable
False
True
Buf_LS_Gap_Detect_Dis
True
False
Set_Gap_Delay
Gap_Detect_Delay = 1
Gap_Detect_Delay = 0
Buf_Option_Bits
Buf_DC_Options = temp_dc_options
Buf_Reverse_Process_En
False
True
Forward_Crank
(continue)

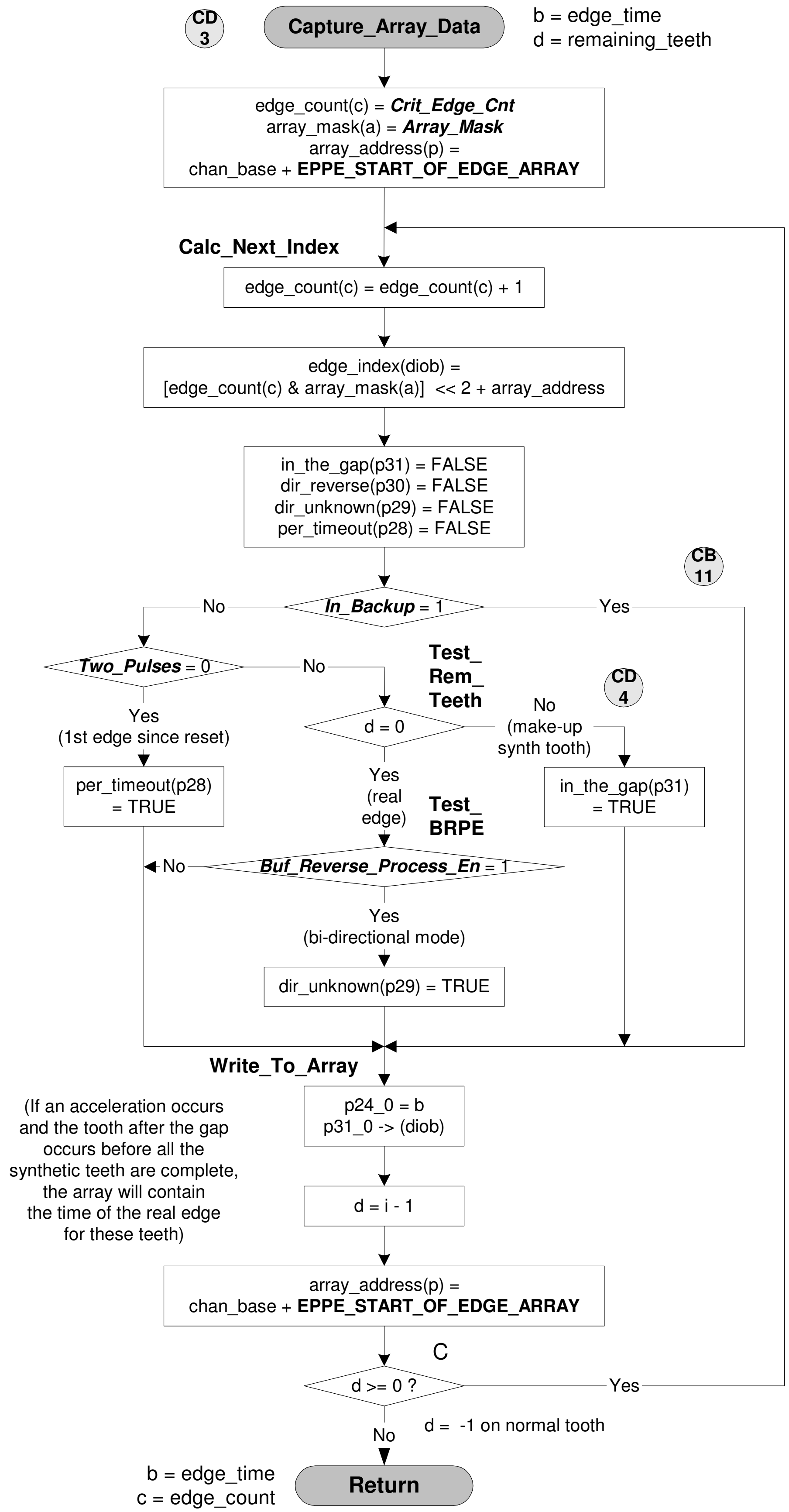

Forward_Crank
Accum_Edges = Accum_Edges + 1
abs_ec = Abs_Edge_Count
TEETH_PER_CAM
= 120
abs_ec >=
(TEETH_PER_CAM - Gap_Size)
?
Yes
No
TEETH_PER_CRANK
= 60
FC_Test_Gap
abs_ec =
(TEETH_PER_CRANK - Gap_Size)
?
No
Yes
abs_ec = abs_ec + Gap_Size
FC_Incr
abs_ec = 1
abs_ec = abs_ec + 1
FC_Update
Abs_Edge_Count = abs_ec
Crank_Backwards = 0?
Yes
(direction
didn't
change)
No
(direction changed)
Crank_Backwards = 0
False
Buf_Reverse_Process_En
True
Chg_Dir_Count =Chg_Dir_Count + 1
- request interrupt to the host -
IRQ_Requests(Change_Dir) = TRUE
FC_Ret
RETURN

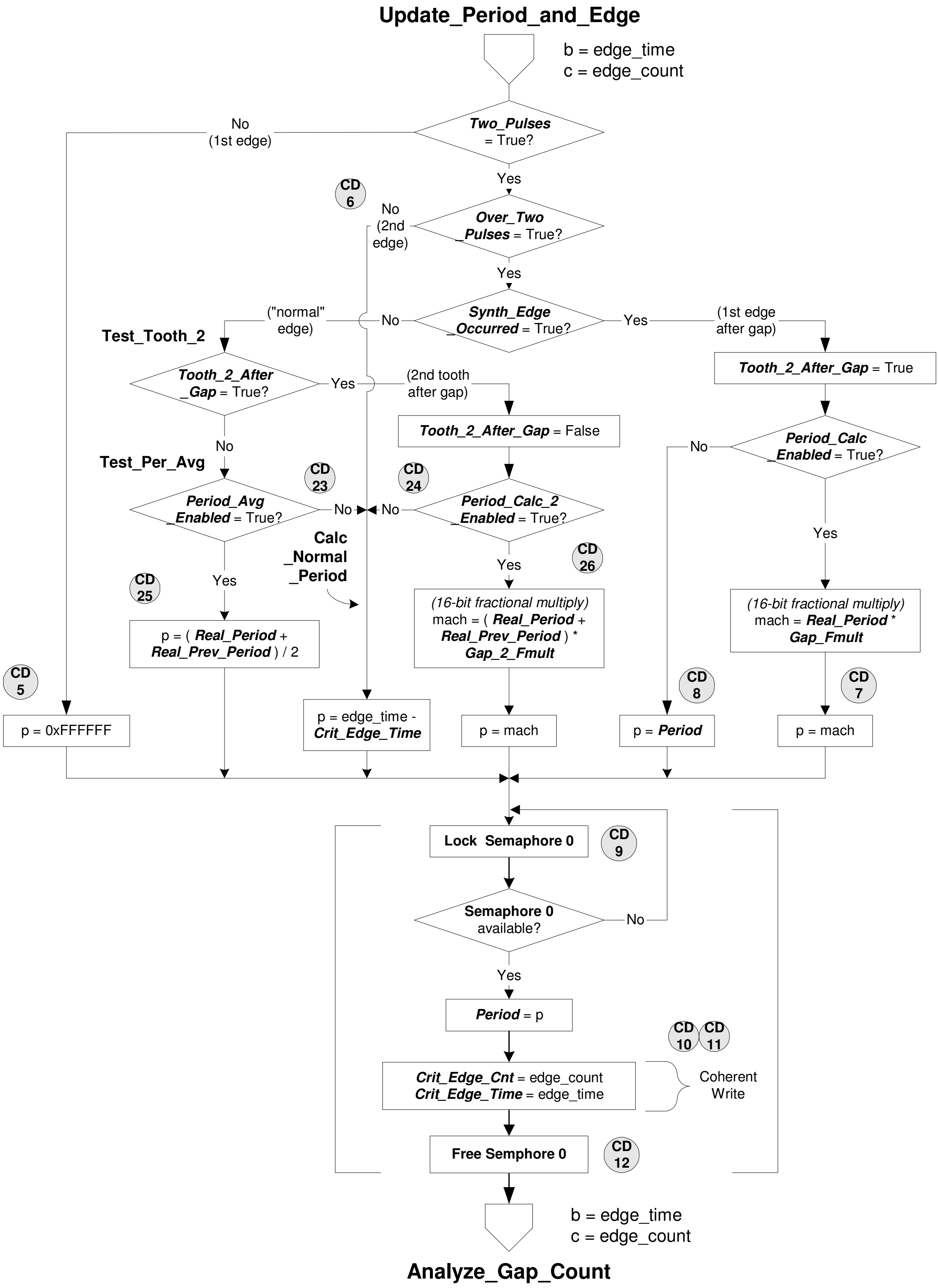

Reverse_Crank
Accum_Edges = Accum_Edges - 1
abs_ec = Abs_Edge_Count
abs_ec <= 1?
Yes
No
TEETH_PER_CRANK = 60
RC_Test_Gap
abs_ec = (TEETH_PER_CRANK + 1) ?
No
Yes
abs_ec = abs_ec - Gap_Size
TEETH _PER _CAM = 120
abs_ec = TEETH_PER_CAM - Gap_Size
RC_Decr
abs_ec = abs_ec - 1
RC_Update
Abs_Edge_Count = abs_ec
Crank_Backwards = 1?
Yes (direction didn't change)
No (direction changed)
Crank_Backwards = 1
Chg_Dir_Count = Chg_Dir_Count + 1
- request interrupt to the host -
IRQ_Requests(Change_Dir) = TRUE
RC_Ret
RETURN

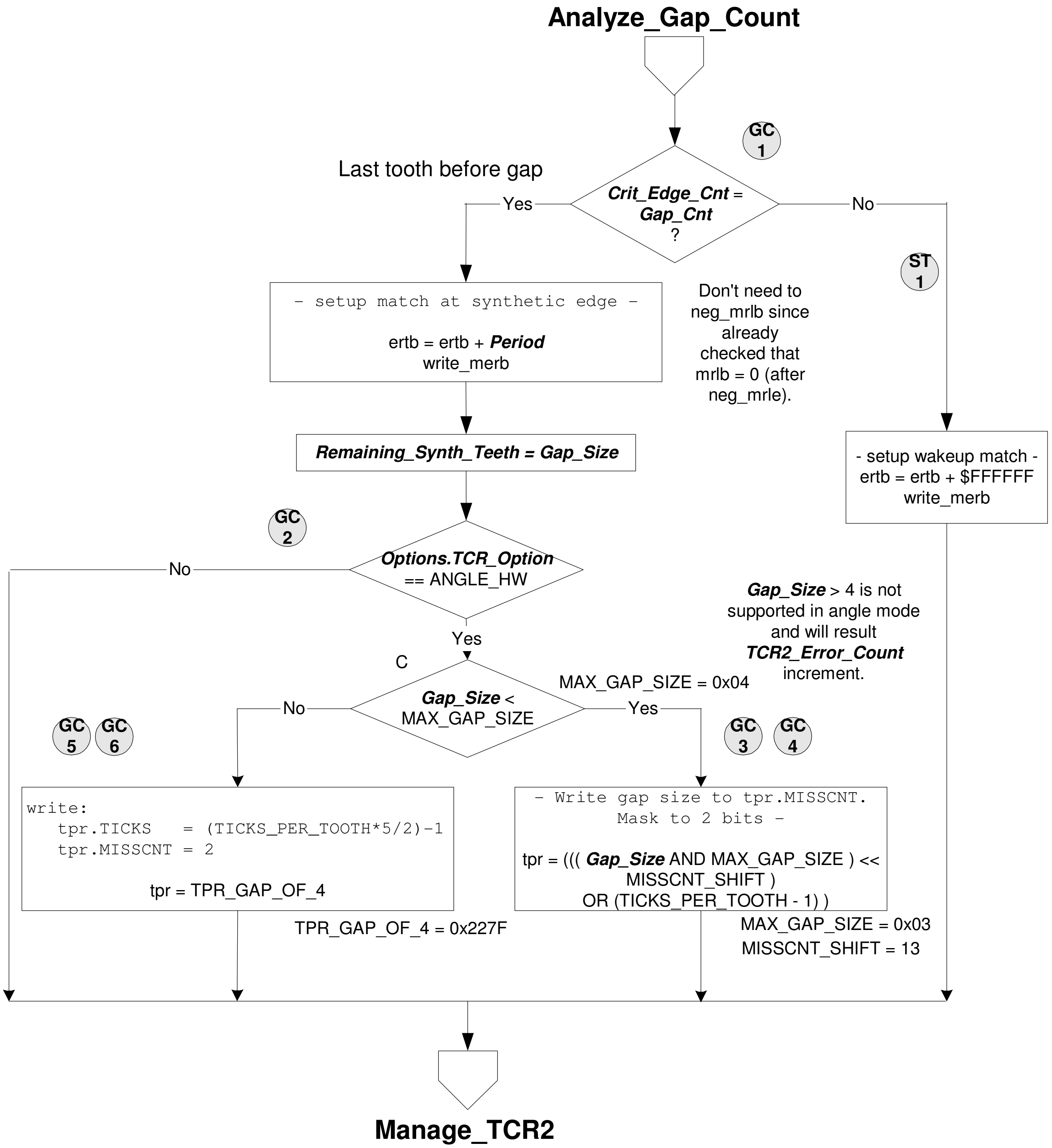


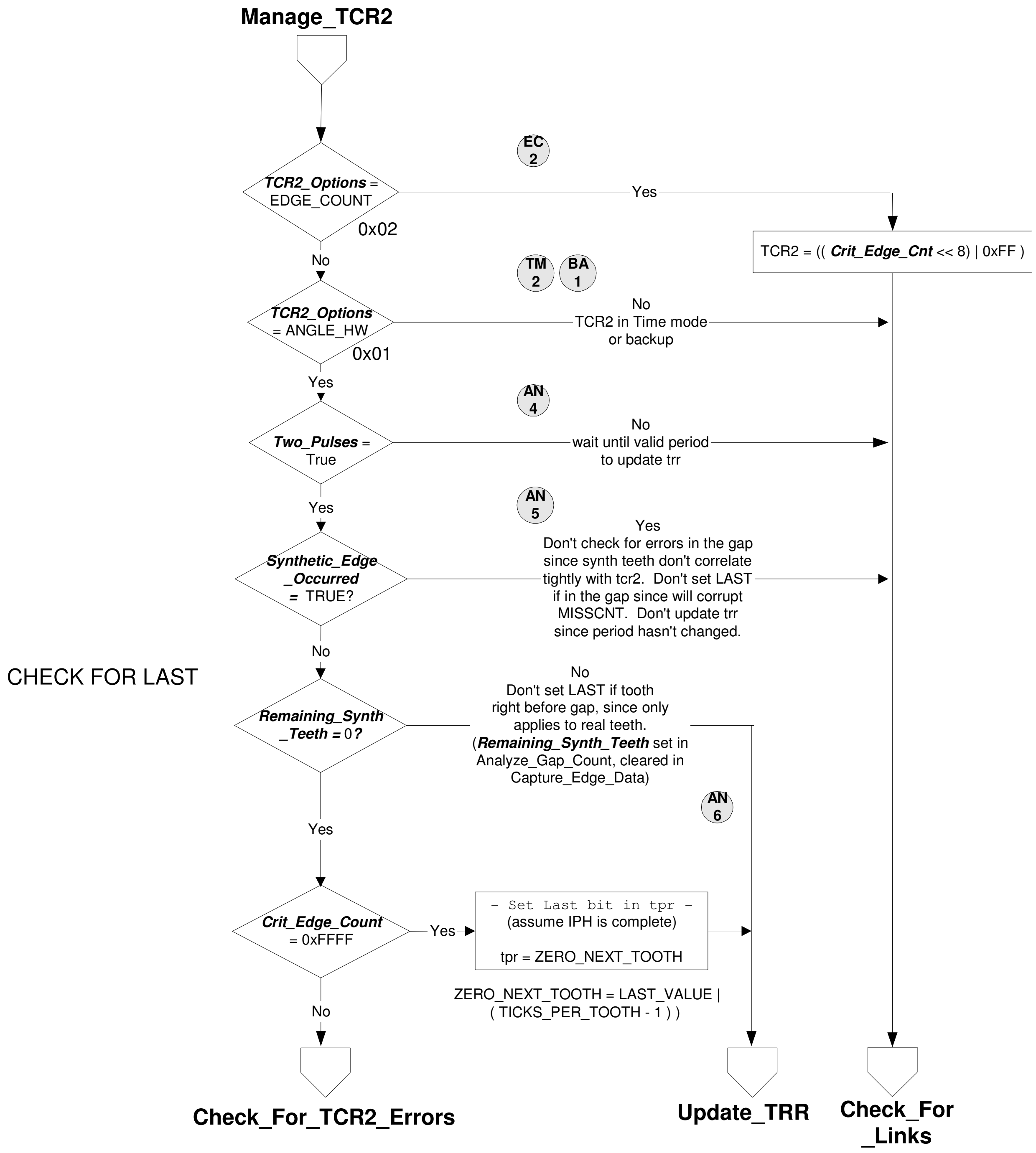
Perform_Gap_Analysis
p31_24 = Options
p23_0 = prev_prev_period
sr = Real_Period
a = Real_Prev_Period
b = edge_time
c = edge_count
CB 28
Percent Period Gap Detection Method
Threshold (1-X / Y-1) Gap Detection Method
Gap_Detect_Method = Percent_Period?
Yes
No
period_sum = Real_Period + Real_Prev_Period
C
if period_sum oveflow?
Yes
GA 11
No
period_sum = period_sum + prev_prev_period
C
if period_sum oveflow?
Yes
GA 12
GA 13
No
- calculate percentage with 8 bit fractional multiply -
percentage(|mach) = period_sum(sr) * Fmult_Percent(p7_0)
GA 14
C = 1
Real_Prev_Period > percentage ?
No
GA 15
Yes
GA 16
In_Backup = TRUE ?
No
Yes
Gap_Detect_Count = Crit_Edge_Count + 1
Gap_Detect_Count = Real_Edge_Count
IRQ_Requests(Gap_Detected) = TRUE
low_same (unsigned)
GA 4
Real_Period <= Real_Prev_Period ?
No (periods increasing)
Yes (periods decreasing or constant)
GA 8
- 8 bit fractional multiply -
threshold(mach) = Real_Period * Fmult_1_X
- 8 bit fractional multiply -
threshold(mach) = Real_Prev_Period * Fmult_Y_1
GA 5
greater_than (unsigned)
threshold (mach) > Real_Prev_Period ?
No (regular tooth)
GA 9
GA 10
Yes (found 1_X)
In_Backup = TRUE ?
No
Yes
Prev_1_X_Count = Crit_Edge_Count + 1
Prev_1_X_Count = Real_Edge_Count
greater_than (unsigned)
threshold (mach) > Real_Period ?
No (regular tooth)
GA 6
GA 7
Yes (found Y_1)
In_Backup = TRUE ?
No
Yes
Prev_Y_1_Count = Crit_Edge_Count + 1
Prev_Y_1_Count = Real_Edge_Count
IRQ_Requests(Gap_Detected) = TRUE
Process_CAM

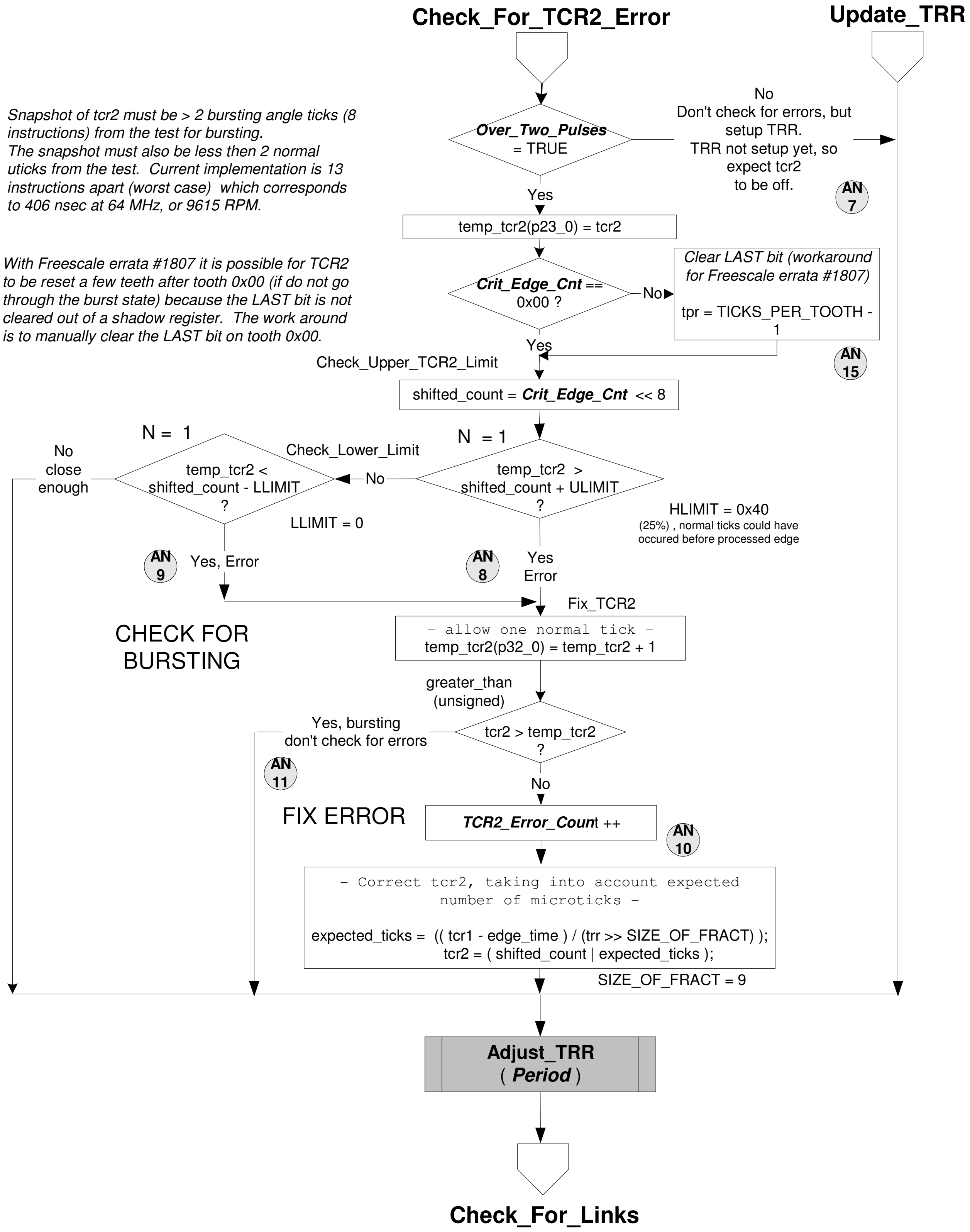

Process_Cam
CA 1
- save current channel for return -
eppe_chan(a) = CHAN
sr = ***Cam_Chan_Number***
- rotate temp_cam_history -
temp_cam_history(p31_0) = ***Cam_History***
p31_24 = p31_24 << 1
p23_0 = p23_0 << 1
C
temp_cam_history (bit 24) =1 ?
No
Yes
p31_24 ++
- change to CAM channel -
CHAN = sr
CAM Channel
PSTI
pin = 1?
No
Yes
p23_0 = p23_0 + 1
- return to epp channel -
CHAN = eppe_chan(macl)
Cam_History = temp_cam_history(p31_0)
Return


CD 3
Capture_Array_Data
b = edge_time
d = remaining_teeth
edge_count(c) = ***Crit_Edge_Cnt***
array_mask(a) = ***Array_Mask***
array_address(p) =
chan_base + **EPPE_START_OF_EDGE_ARRAY**
Calc_Next_Index
edge_count(c) = edge_count(c) + 1
edge_index(diob) =
[edge_count(c) & array_mask(a)] << 2 + array_address
in_the_gap(p31) = FALSE
dir_reverse(p30) = FALSE
dir_unknown(p29) = FALSE
per_timeout(p28) = FALSE
CB 11
No
In_Backup = 1
Yes
Two_Pulses = 0
No
Test_ Rem_ Teeth
CD 4
No (make-up synth tooth)
d = 0
Yes (1st edge since reset)
per_timeout(p28) = TRUE
Yes (real edge)
Test_ BRPE
in_the_gap(p31) = TRUE
No
Buf_Reverse_Process_En = 1
Yes (bi-directional mode)
dir_unknown(p29) = TRUE
Write_To_Array
p24_0 = b
p31_0 -> (diob)
(If an acceleration occurs and the tooth after the gap occurs before all the synthetic teeth are complete, the array will contain the time of the real edge for these teeth)
d = i - 1
array_address(p) =
chan_base + **EPPE_START_OF_EDGE_ARRAY**
C
d >= 0 ?
Yes
No
d = -1 on normal tooth
b = edge_time
c = edge_count
Return


Update_Period_and_Edge
b = edge_time
c = edge_count
Two_Pulses = True?
No (1st edge)
Yes
CD 6
No (2nd edge)
Over_Two_Pulses = True?
Yes
("normal" edge)
No
Synth_Edge_Occurred = True?
Yes
(1st edge after gap)
Test_Tooth_2
Tooth_2_After_Gap = True?
Yes
(2nd tooth after gap)
Tooth_2_After_Gap = True
Tooth_2_After_Gap = False
No
Period_Calc_Enabled = True?
No
Test_Per_Avg
CD 23
CD 24
Period_Avg_Enabled = True?
No
No
Period_Calc_2_Enabled = True?
Yes
Calc_Normal_Period
Yes
CD 26
Yes
CD 25
p = (Real_Period + Real_Prev_Period) / 2
(16-bit fractional multiply) mach = (Real_Period + Real_Prev_Period) * Gap_2_Fmult
(16-bit fractional multiply) mach = Real_Period * Gap_Fmult
CD 5
CD 8
CD 7
p = 0xFFFFFF
p = edge_time - Crit_Edge_Time
p = mach
p = Period
p = mach
Lock Semaphore 0
CD 9
Semaphore 0 available?
No
Yes
Period = p
CD 10
CD 11
Crit_Edge_Cnt = edge_count
Crit_Edge_Time = edge_time
Coherent Write
Free Semphore 0
CD 12
b = edge_time
c = edge_count
Analyze_Gap_Count

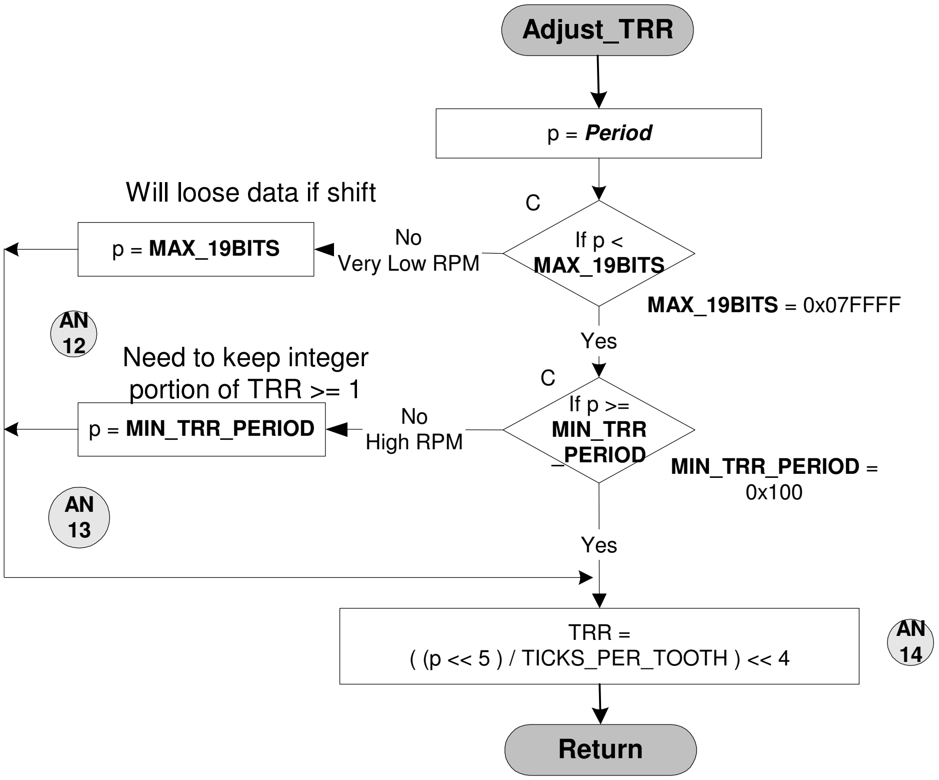

Analyze_Gap_Count
GC 1
Last tooth before gap
Crit_Edge_Cnt = Gap_Cnt ?
Yes
No
ST 1
- setup match at synthetic edge -
ertb = ertb + Period
write_merb
Don't need to neg_mrlb since already checked that mrlb = 0 (after neg_mrle).
- setup wakeup match -
ertb = ertb + $FFFFFF
write_merb
Remaining_Synth_Teeth = Gap_Size
GC 2
Options.TCR_Option == ANGLE_HW
No
Yes
Gap_Size > 4 is not supported in angle mode and will result TCR2_Error_Count increment.
C
MAX_GAP_SIZE = 0x04
Gap_Size < MAX_GAP_SIZE
No
Yes
GC 5
GC 6
GC 3
GC 4
write:
tpr.TICKS = (TICKS_PER_TOOTH*5/2)-1
tpr.MISSCNT = 2
tpr = TPR_GAP_OF_4
TPR_GAP_OF_4 = 0x227F
- Write gap size to tpr.MISSCNT. Mask to 2 bits -
tpr = (((Gap_Size AND MAX_GAP_SIZE) << MISSCNT_SHIFT) OR (TICKS_PER_TOOTH - 1))
MAX_GAP_SIZE = 0x03
MISSCNT_SHIFT = 13
Manage_TCR2

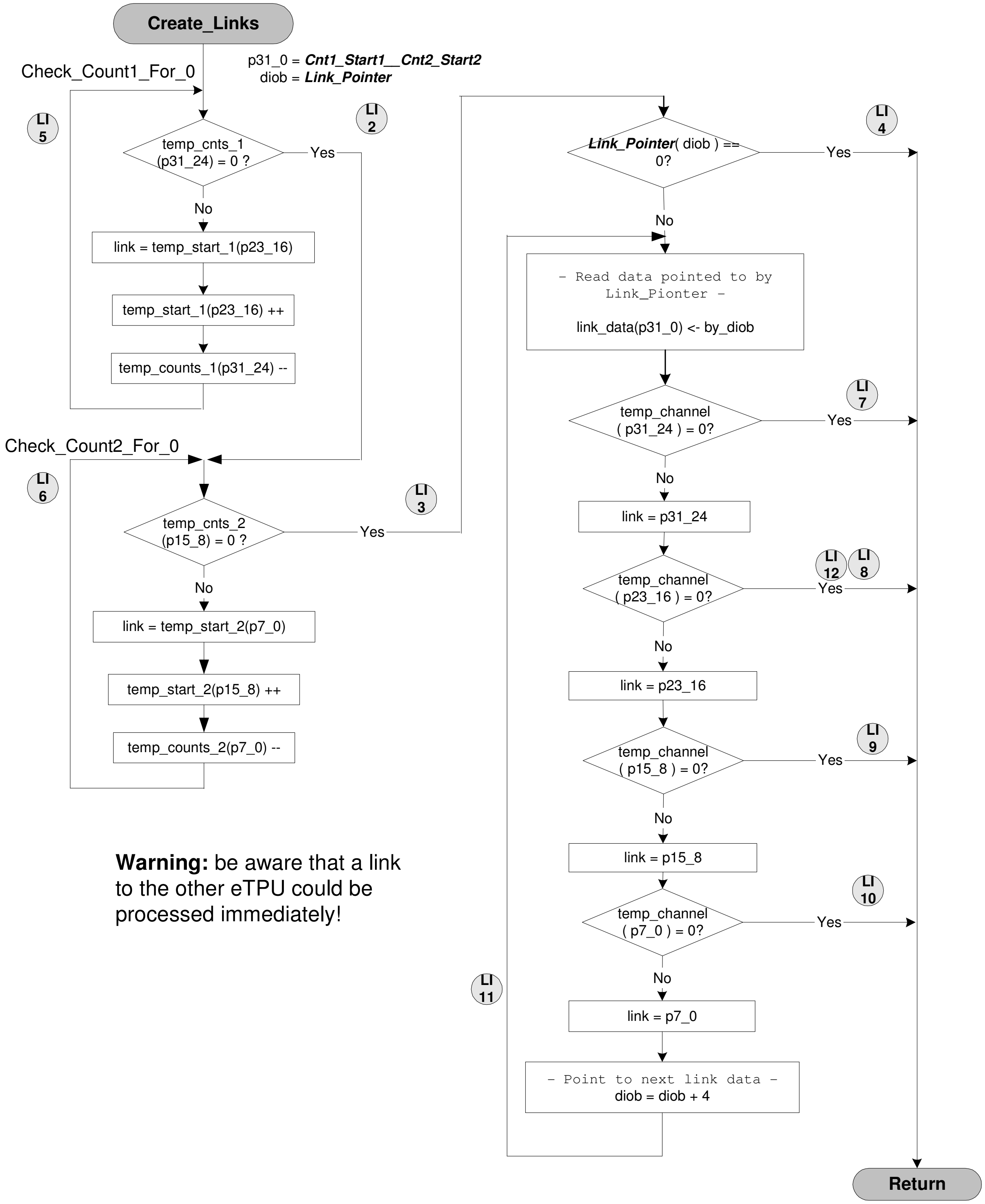

Manage_TCR2
EC 2
TCR2_Options = EDGE_COUNT
0x02
Yes
TCR2 = ((Crit_Edge_Cnt << 8) | 0xFF)
No
TM 2
BA 1
TCR2_Options = ANGLE_HW
0x01
No
TCR2 in Time mode
or backup
Yes
AN 4
Two_Pulses = True
No
wait until valid period
to update trr
Yes
AN 5
Synthetic_Edge _Occurred = TRUE?
Yes
Don't check for errors in the gap
since synth teeth don't correlate
tightly with tcr2. Don't set LAST
if in the gap since will corrupt
MISSCNT. Don't update trr
since period hasn't changed.
No
CHECK FOR LAST
Remaining_Synth _Teeth = 0?
No
Don't set LAST if tooth
right before gap, since only
applies to real teeth.
(Remaining_Synth_Teeth set in
Analyze_Gap_Count, cleared in
Capture_Edge_Data)
AN 6
Yes
Crit_Edge_Count = 0xFFFF
Yes
- Set Last bit in tpr -
(assume IPH is complete)
tpr = ZERO_NEXT_TOOTH
ZERO_NEXT_TOOTH = LAST_VALUE |
(TICKS_PER_TOOTH - 1))
No
Check_For_TCR2_Errors
Update_TRR
Check_For _Links


Check_For_TCR2_Error
Update_TRR
Over_Two_Pulses = TRUE
No
Don't check for errors, but setup TRR.
TRR not setup yet, so expect tcr2 to be off.
AN 7
Yes
temp_tcr2(p23_0) = tcr2
Crit_Edge_Cnt == 0x00 ?
No
Clear LAST bit (workaround for Freescale errata #1807)
tpr = TICKS_PER_TOOTH - 1
AN 15
Yes
Check_Upper_TCR2_Limit
shifted_count = Crit_Edge_Cnt << 8
N = 1
temp_tcr2 > shifted_count + ULIMIT ?
HLIMIT = 0x40
(25%) , normal ticks could have occured before processed edge
No
Check_Lower_Limit
N = 1
temp_tcr2 < shifted_count - LLIMIT ?
LLIMIT = 0
No close enough
AN 9
Yes, Error
AN 8
Yes Error
Fix_TCR2
- allow one normal tick -
temp_tcr2(p32_0) = temp_tcr2 + 1
CHECK FOR BURSTING
greater_than (unsigned)
tcr2 > temp_tcr2 ?
Yes, bursting don't check for errors
AN 11
No
FIX ERROR
TCR2_Error_Count ++
AN 10
- Correct tcr2, taking into account expected number of microticks -
expected_ticks = ((tcr1 - edge_time) / (trr >> SIZE_OF_FRACT));
tcr2 = (shifted_count | expected_ticks);
SIZE_OF_FRACT = 9
Adjust_TRR (Period)
Check_For_Links

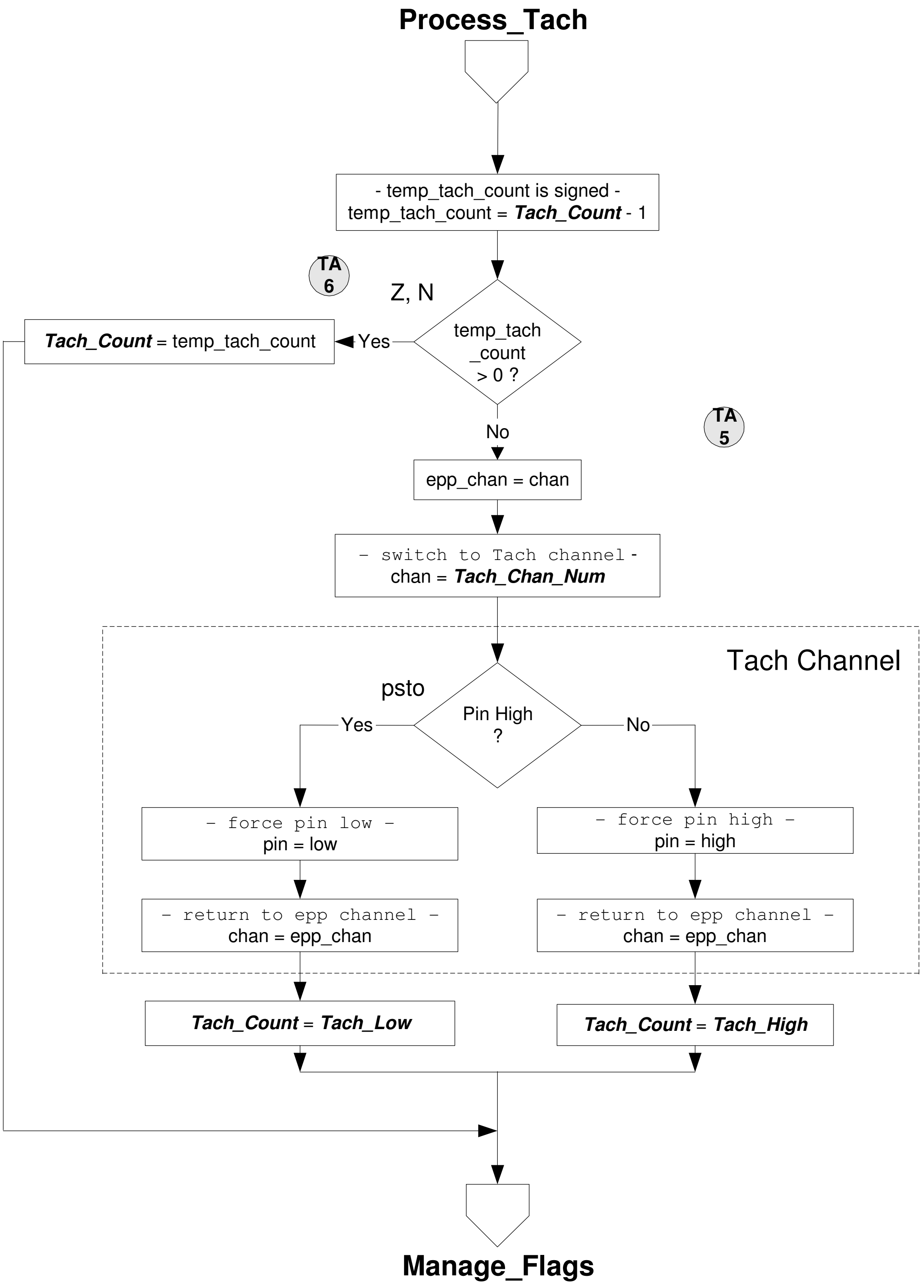
Snapshot of tcr2 must be > 2 bursting angle ticks (8 instructions) from the test for bursting.
The snapshot must also be less then 2 normal uticks from the test. Current implementation is 13 instructions apart (worst case) which corresponds to 406 nsec at 64 MHz, or 9615 RPM.

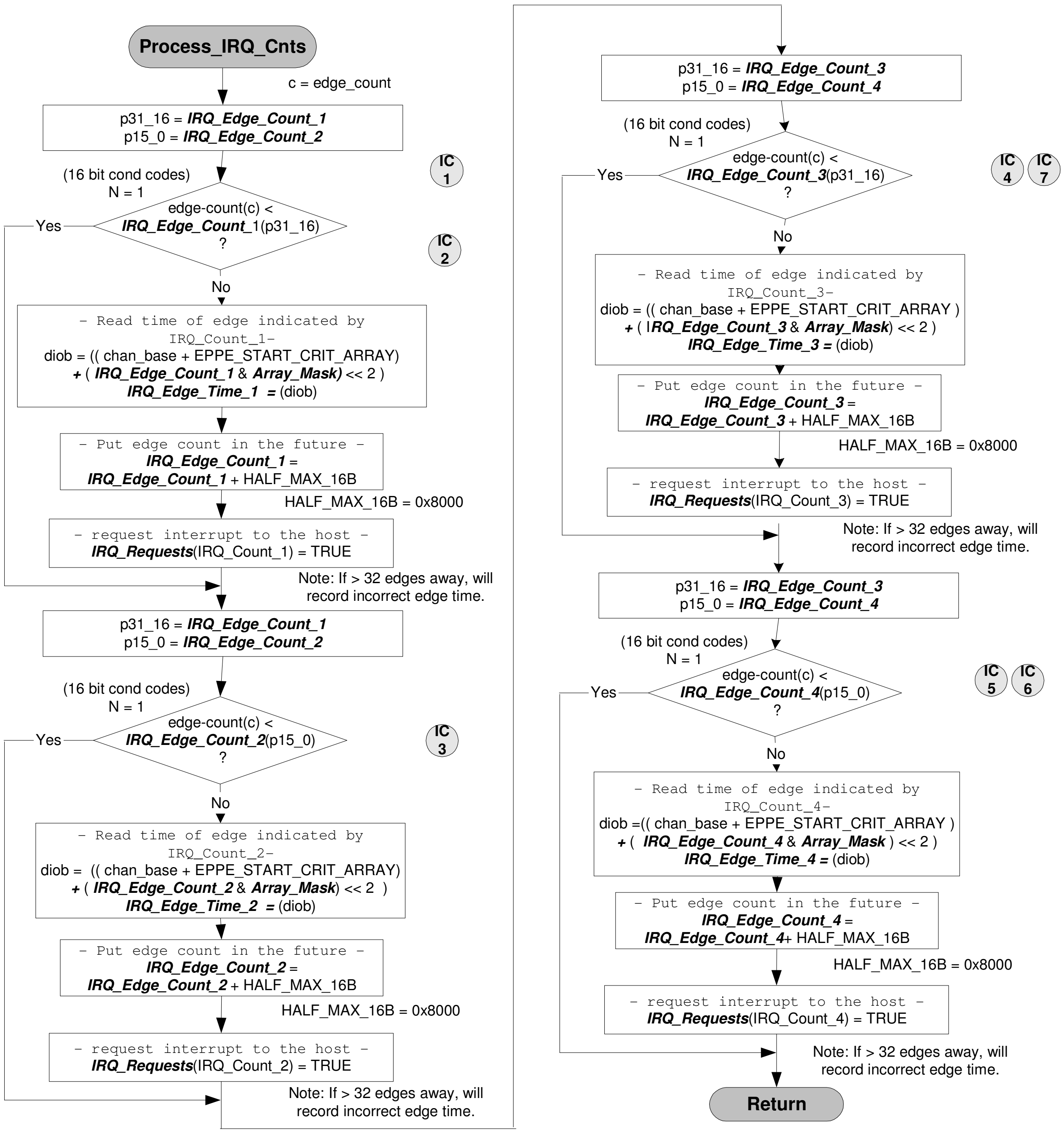
With Freescale errata #1807 it is possible for TCR2 to be reset a few teeth after tooth 0x00 (if do not go through the burst state) because the LAST bit is not cleared out of a shadow register. The work around is to manually clear the LAST bit on tooth 0x00.

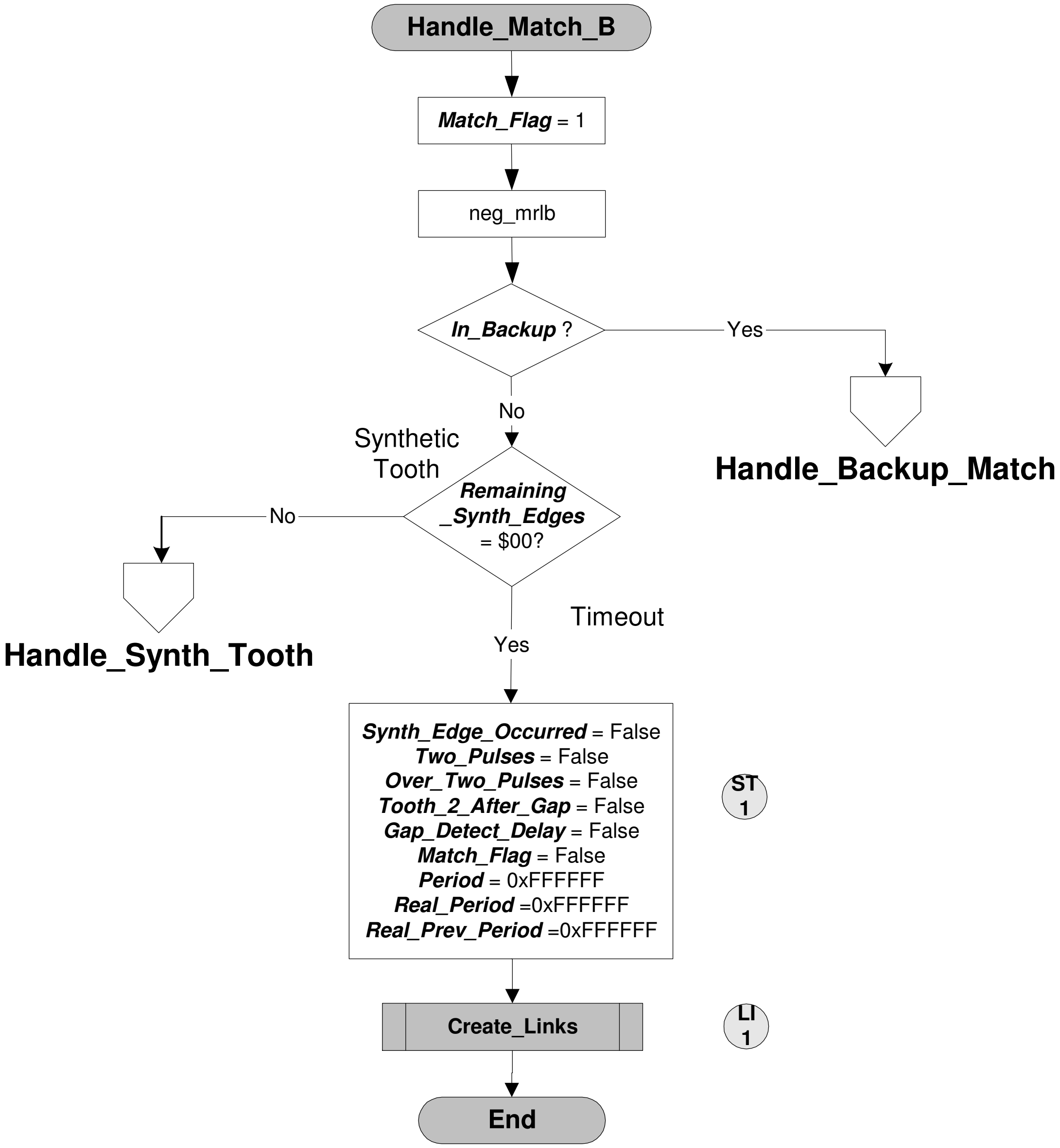

Adjust_TRR
p = Period
Will loose data if shift
C
p = MAX_19BITS
No
Very Low RPM
If p <
MAX_19BITS
MAX_19BITS = 0x07FFFF
Yes
AN
12
Need to keep integer
portion of TRR >= 1
C
If p >=
MIN_TRR
_PERIOD
p = MIN_TRR_PERIOD
No
High RPM
MIN_TRR_PERIOD =
0x100
AN
13
Yes
TRR =
((p << 5) / TICKS_PER_TOOTH) << 4
AN
14
Return


Create_Links
Check_Count1_For_0
p31_0 = Cnt1_Start1__Cnt2_Start2
diob = Link_Pointer
LI 5
LI 2
temp_cnts_1 (p31_24) = 0 ?
Yes
No
link = temp_start_1(p23_16)
temp_start_1(p23_16) ++
temp_counts_1(p31_24) --
Check_Count2_For_0
LI 6
LI 3
temp_cnts_2 (p15_8) = 0 ?
Yes
No
link = temp_start_2(p7_0)
temp_start_2(p15_8) ++
temp_counts_2(p7_0) --
Link_Pointer(diob) == 0?
Yes
LI 4
No
- Read data pointed to by Link_Pionter -
link_data(p31_0) <- by_diob
LI 7
temp_channel (p31_24) = 0?
Yes
No
link = p31_24
LI 12
LI 8
temp_channel (p23_16) = 0?
Yes
No
link = p23_16
LI 9
temp_channel (p15_8) = 0?
Yes
No
link = p15_8
LI 10
temp_channel (p7_0) = 0?
Yes
No
LI 11
link = p7_0
- Point to next link data -
diob = diob + 4
Return

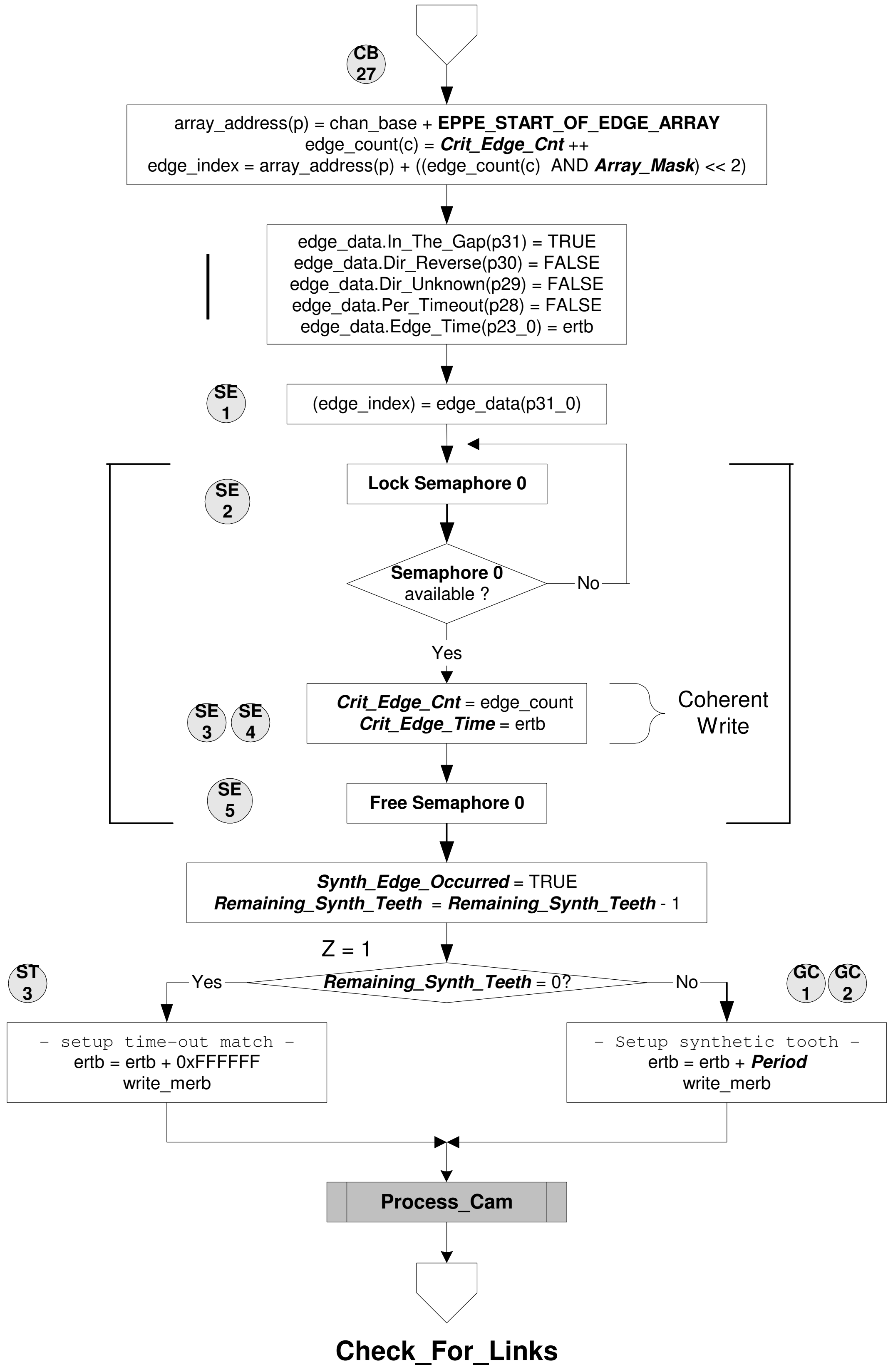
Warning: be aware that a link to the other eTPU could be processed immediately!


Process_Tach
- temp_tach_count is signed -
temp_tach_count = Tach_Count - 1
TA 6
Z, N
temp_tach _count > 0 ?
Yes
Tach_Count = temp_tach_count
No
TA 5
epp_chan = chan
- switch to Tach channel -
chan = Tach_Chan_Num
Tach Channel
psto
Pin High ?
Yes
No
- force pin low -
pin = low
- force pin high -
pin = high
- return to epp channel -
chan = epp_chan
- return to epp channel -
chan = epp_chan
Tach_Count = Tach_Low
Tach_Count = Tach_High
Manage_Flags

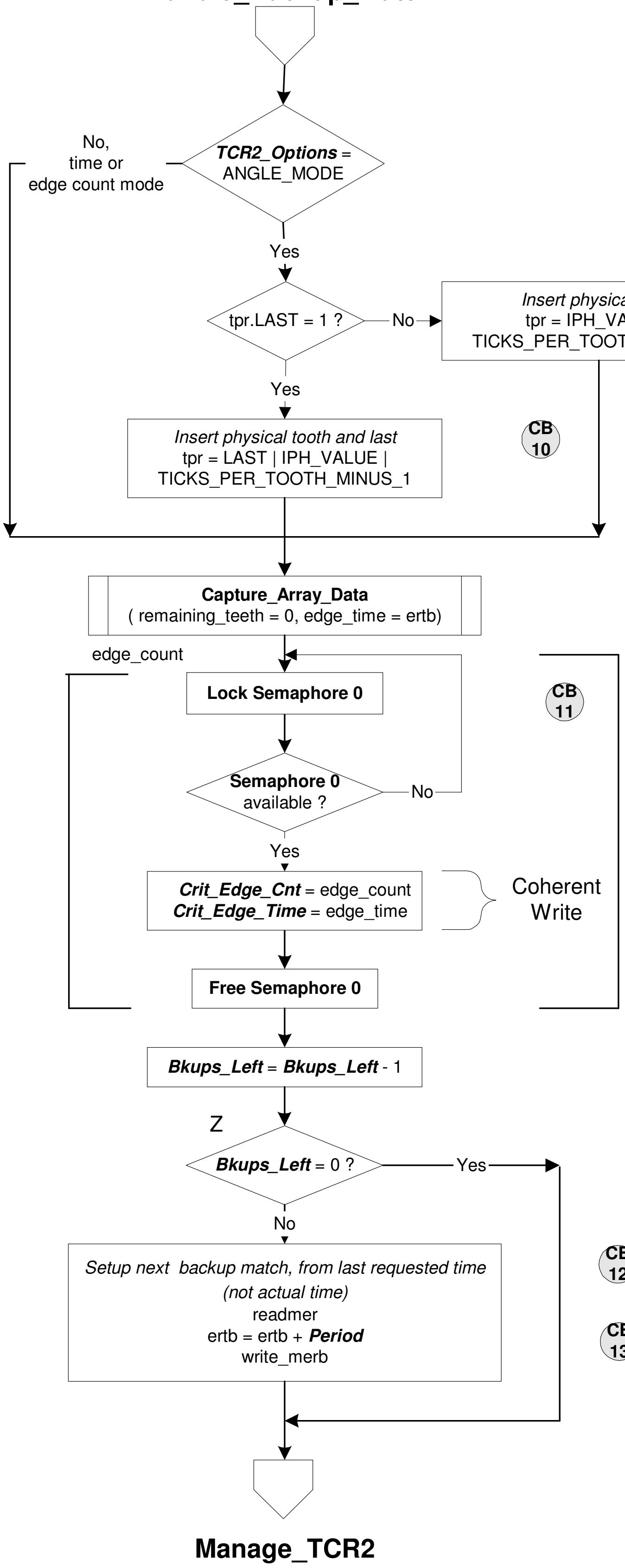

Process_IRQ_Cnts
c = edge_count
p31_16 = IRQ_Edge_Count_1
p15_0 = IRQ_Edge_Count_2
IC 1
(16 bit cond codes)
N = 1
edge-count(c) <
IRQ_Edge_Count_1(p31_16)
?
Yes
IC 2
No
- Read time of edge indicated by
IRQ_Count_1-
diob = ((chan_base + EPPE_START_CRIT_ARRAY)
+ (IRQ_Edge_Count_1 & Array_Mask) << 2)
IRQ_Edge_Time_1 = (diob)
- Put edge count in the future -
IRQ_Edge_Count_1 =
IRQ_Edge_Count_1 + HALF_MAX_16B
HALF_MAX_16B = 0x8000
- request interrupt to the host -
IRQ_Requests(IRQ_Count_1) = TRUE
Note: If > 32 edges away, will
record incorrect edge time.
p31_16 = IRQ_Edge_Count_1
p15_0 = IRQ_Edge_Count_2
(16 bit cond codes)
N = 1
edge-count(c) <
IRQ_Edge_Count_2(p15_0)
?
Yes
IC 3
No
- Read time of edge indicated by
IRQ_Count_2-
diob = ((chan_base + EPPE_START_CRIT_ARRAY)
+ (IRQ_Edge_Count_2 & Array_Mask) << 2)
IRQ_Edge_Time_2 = (diob)
- Put edge count in the future -
IRQ_Edge_Count_2 =
IRQ_Edge_Count_2 + HALF_MAX_16B
HALF_MAX_16B = 0x8000
- request interrupt to the host -
IRQ_Requests(IRQ_Count_2) = TRUE
Note: If > 32 edges away, will
record incorrect edge time.
p31_16 = IRQ_Edge_Count_3
p15_0 = IRQ_Edge_Count_4
(16 bit cond codes)
N = 1
edge-count(c) <
IRQ_Edge_Count_3(p31_16)
?
Yes
IC 4
IC 7
No
- Read time of edge indicated by
IRQ_Count_3-
diob = ((chan_base + EPPE_START_CRIT_ARRAY)
+ (IRQ_Edge_Count_3 & Array_Mask) << 2)
IRQ_Edge_Time_3 = (diob)
- Put edge count in the future -
IRQ_Edge_Count_3 =
IRQ_Edge_Count_3 + HALF_MAX_16B
HALF_MAX_16B = 0x8000
- request interrupt to the host -
IRQ_Requests(IRQ_Count_3) = TRUE
Note: If > 32 edges away, will
record incorrect edge time.
p31_16 = IRQ_Edge_Count_3
p15_0 = IRQ_Edge_Count_4
(16 bit cond codes)
N = 1
edge-count(c) <
IRQ_Edge_Count_4(p15_0)
?
Yes
IC 5
IC 6
No
- Read time of edge indicated by
IRQ_Count_4-
diob =((chan_base + EPPE_START_CRIT_ARRAY)
+ (IRQ_Edge_Count_4 & Array_Mask) << 2)
IRQ_Edge_Time_4 = (diob)
- Put edge count in the future -
IRQ_Edge_Count_4 =
IRQ_Edge_Count_4+ HALF_MAX_16B
HALF_MAX_16B = 0x8000
- request interrupt to the host -
IRQ_Requests(IRQ_Count_4) = TRUE
Note: If > 32 edges away, will
record incorrect edge time.
Return


Handle_Match_B
Match_Flag = 1
neg_mrlb
In_Backup ?
Yes
No
Handle_Backup_Match
Synthetic Tooth
Remaining _Synth_Edges = $00?
No
Handle_Synth_Tooth
Timeout
Yes
Synth_Edge_Occurred = False
Two_Pulses = False
Over_Two_Pulses = False
Tooth_2_After_Gap = False
Gap_Detect_Delay = False
Match_Flag = False
Period = 0xFFFFFF
Real_Period =0xFFFFFF
Real_Prev_Period =0xFFFFFF
ST 1
Create_Links
LI 1
End

Handle_Synth_Tooth

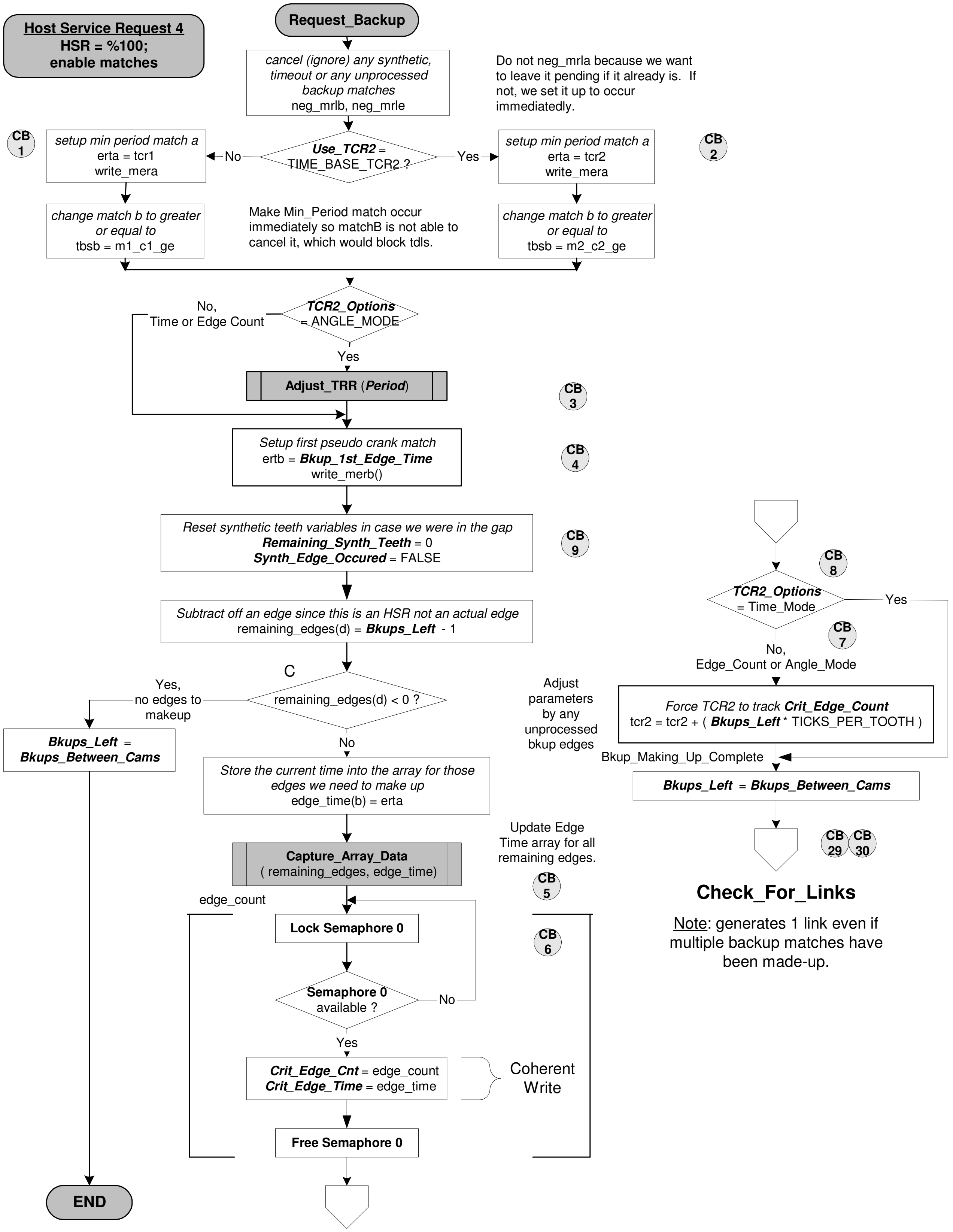

CB 27
array_address(p) = chan_base + EPPE_START_OF_EDGE_ARRAY
edge_count(c) = Crit_Edge_Cnt ++
edge_index = array_address(p) + ((edge_count(c) AND Array_Mask) << 2)
edge_data.In_The_Gap(p31) = TRUE
edge_data.Dir_Reverse(p30) = FALSE
edge_data.Dir_Unknown(p29) = FALSE
edge_data.Per_Timeout(p28) = FALSE
edge_data.Edge_Time(p23_0) = ertb
SE 1
(edge_index) = edge_data(p31_0)
SE 2
Lock Semaphore 0
Semaphore 0 available ?
No
Yes
SE 3
SE 4
Crit_Edge_Cnt = edge_count
Crit_Edge_Time = ertb
Coherent Write
SE 5
Free Semaphore 0
Synth_Edge_Occurred = TRUE
Remaining_Synth_Teeth = Remaining_Synth_Teeth - 1
Z = 1
Remaining_Synth_Teeth = 0?
Yes
No
ST 3
GC 1
GC 2
- setup time-out match -
ertb = ertb + 0xFFFFFF
write_merb
- Setup synthetic tooth -
ertb = ertb + Period
write_merb
Process_Cam
Check_For_Links

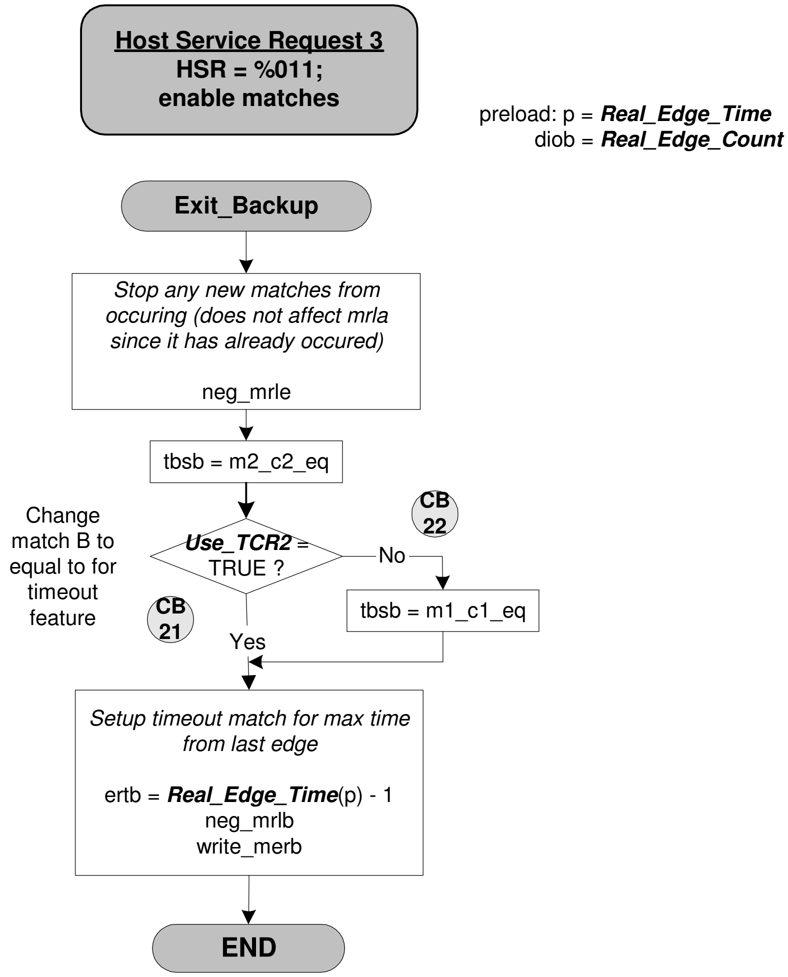
Handle_Backup_Match


TCR2_Options = ANGLE_MODE
No, time or edge count mode
Yes
tpr.LAST = 1 ?
No
Insert physical tooth
tpr = IPH_VALUE | TICKS_PER_TOOTH_MINUS_1
Yes
Insert physical tooth and last
tpr = LAST | IPH_VALUE | TICKS_PER_TOOTH_MINUS_1
CB 10
Capture_Array_Data
(remaining_teeth = 0, edge_time = ertb)
edge_count
Lock Semaphore 0
CB 11
Semaphore 0 available ?
No
Yes
Crit_Edge_Cnt = edge_count
Crit_Edge_Time = edge_time
Coherent Write
Free Semaphore 0
Bkups_Left = Bkups_Left - 1
Z
Bkups_Left = 0 ?
Yes
No
Setup next backup match, from last requested time
(not actual time)
readmer
ertb = ertb + Period
write_merb
CB 12
CB 13
Manage_TCR2

Insert physical tooth as soon as possible

IPH is ignored when bursting, but TCR2 will be caught up in Manage_TCR2 (later this thread). This sequence may occur if the Request HSR has missed > 1 crank edge.


Host Service Request 4
HSR = %100;
enable matches
Request_Backup
cancel (ignore) any synthetic, timeout or any unprocessed backup matches
neg_mrlb, neg_mrle
Do not neg_mrla because we want to leave it pending if it already is. If not, we set it up to occur immediatedly.
CB 1
setup min period match a
erta = tcr1
write_mera
No
Use_TCR2 = TIME_BASE_TCR2 ?
Yes
setup min period match a
erta = tcr2
write_mera
CB 2
change match b to greater or equal to
tbsb = m1_c1_ge
Make Min_Period match occur immediately so matchB is not able to cancel it, which would block tdls.
change match b to greater or equal to
tbsb = m2_c2_ge
No,
Time or Edge Count
TCR2_Options = ANGLE_MODE
Yes
Adjust_TRR (Period)
CB 3
Setup first pseudo crank match
ertb = Bkup_1st_Edge_Time
write_merb()
CB 4
Reset synthetic teeth variables in case we were in the gap
Remaining_Synth_Teeth = 0
Synth_Edge_Occured = FALSE
CB 9
Subtract off an edge since this is an HSR not an actual edge
remaining_edges(d) = Bkups_Left - 1
C
Yes,
no edges to
makeup
remaining_edges(d) < 0 ?
Bkups_Left = Bkups_Between_Cams
No
Store the current time into the array for those edges we need to make up
edge_time(b) = erta
Capture_Array_Data
(remaining_edges, edge_time)
Update Edge Time array for all remaining edges.
CB 5
edge_count
Lock Semaphore 0
CB 6
Semaphore 0 available ?
No
Yes
Crit_Edge_Cnt = edge_count
Crit_Edge_Time = edge_time
Coherent Write
Free Semaphore 0
END
CB 8
TCR2_Options = Time_Mode
Yes
CB 7
No,
Edge_Count or Angle_Mode
Adjust parameters by any unprocessed bkup edges
Force TCR2 to track Crit_Edge_Count
tcr2 = tcr2 + (Bkups_Left * TICKS_PER_TOOTH)
Bkup_Making_Up_Complete
Bkups_Left = Bkups_Between_Cams
CB 29
CB 30
Check_For_Links
Note: generates 1 link even if multiple backup matches have been made-up.


Host Service Request 3
HSR = %011;
enable matches
preload: p = Real_Edge_Time
diob = Real_Edge_Count
Exit_Backup
Stop any new matches from occuring (does not affect mrla since it has already occured)
neg_mrle
tbsb = m2_c2_eq
Change match B to equal to for timeout feature
CB 22
Use_TCR2 = TRUE ?
No
tbsb = m1_c1_eq
CB 21
Yes
Setup timeout match for max time from last edge
ertb = Real_Edge_Time(p) - 1
neg_mrlb
write_merb
END

7.0 Revision History

Each microcode document is assigned a revision number. The numbers are assigned according to the following scheme (used for all documents after May, 2003):

Rev x.y

x identifies the microcode, where

1 represents the original release of microcode
2 represents the first release of changed microcode
3 represents the second change to microcode etc.

y identifies the document, where

0 represents the original release of documentation for this microcode
1 represents the first document change for the same microcode
2 represents the second change to the document for the same microcode

7.1 Revision Log for this Application:

Revision	Date	Record	Author
1.0	04-21-04	Initial release of documentation.	Mary Hedges
1.1	04-27-04	Updated documentation.	Mary Hedges
2.0	05-27-04	• Implemented percent period gap detection method. • Initial implementation of angle logic. • Add DMA trigger at end of each edge logic thread (real and synthetic). • Fixed interrupt acknowledge sequence to support HSR or suppression of channel methods.	Mary Hedges
3.0	06-30-04	• Optimized init, cam history, tach, data collection • Added ***Gap_Detect_Enabled***. • Moved dma trigger to before IRQ_Cnts() so would not trigger when rejected pulse, changed in all threads to be consistent. • Improved angle logic: added re-synchronization of TCR2 with ***Crit_Edge_Count, a***dded details on initialization process, added use of LAST. • Implemented more efficient links design.	Mary Hedges
3.1	08-17-04	• Added test cases to flow charts	Mary Hedges

Revision	Date	Record	Author
4.0	08-31-04	• Added support for backup mode. • Added HW semaphore to keep Period, Edge_Count and Edge_Time coherent between eTPUs. • Minor cleanup to MCD, including Fig. 1 and 5. • Added few more test cases. • Check and fix TCR2 errors before calculating new trr, so current value is available.	Mary Hedges
5.0	01-06-05	• Added "EPPE" to TCR2_Options, such as **EPPE_Time_Mode**, **EPPE_Angle_HW** etc. • Fixed 16 bit math in ***IRQ_Edge_Count_x*** compares. • Lower byte of TCR2 in EDGE_COUNT mode is now 0xFF. • Added option ***(Gap_Calc_Enabled***) to multiply the period after the gap by the fraction ***Gap_Fmult*** instead of using the period before the gap. • Limit integer portion of tick rate register to 1 at high RPM. Added Speed limitations section.	Mary Hedges
6.0	SCR #3913 03-15-05	• No change to MCD. Fixed definition of HSRs passed to the host.	Mary Hedges
7.0	SCR #3969 04-25-05	• REWRITE IN ASSEMBLY • Made size of the critical edge time array configurable by adding ***Array_Mask***. • HSR method of clearing interrupts is no longer supported. Removed ***New_IRQ_Requests*** and ***IRQ_Clear_Mask***. • Created list of channels to link to in global. • Reordered ***IRQ_Edge_Count_1*** to 4. • Reordered link variables ***Count_1, Start_1*** etc. • Moved ***Two_Pulses*** from channel flag to param. • Moved ***Gap_Fract***, renamed ***Gap_Fmult***. • ***Real_Prev_Period*** now set to 0xFFFFFF at init and stall. • Renamed ***PLL_Error_Count*** to ***TCR2_Error_Count*** • Changed to standard entry table.	Mary Hedges

Revision	Date	Record	Author
		• Changed ***Mult_1_X, Mult_Y_1*** from 1-4 integer multipliers to 0-1 fractional multipliers called ***Fmult_1_X***, ***Fmult_Y_1***. Renamed ***Percent_Multipler*** to ***Fmult_Percent*** to indicate it as a 0-1 fractional multiplier. • Negate match enables (neg_mrle) at beginning of thread to prevent lock-up issue.	
8.0	SCR #4019 6-13-05	• Add cam backup mode	Mary Hedges
9.0	SCR #4201 8-18-05	• Fixed errata in software, no changes in MCD	Mary Hedges
10.0	SCR #4501 1-17-06	• Added work around for Freescale errata #1807 • Misc. MCD cleanup: corrected Fig. 8 and 9.	Mary Hedges
11.0	SCR #4622 3-24-06	• Added support for 56x • Changed ***IRQ_Count*** flow chart to match code. • Backup matches that are made-up on the Request_HSR now check for links and ***IRQ_Count***s.	Mary Hedges
12.0	SCR #4874 08-18-06	• Added option for always calculating ***Period*** as average of last 2 real periods. • Added option for special calculation of ***Period*** on 2nd tooth after gap.	Warren Donley
13.0	SCR #6899 01-10-08	• Added fix so if creating synthetic teeth with ***Period < Min_Period*** a global exception will not be issued. • Fixed definition of ***Min_Period_Method*** in memory map.	Mary Hedges
14.0	SCR #8355 02-03-09	• Added directional crank algorithm. Included ***Chg_Dir_Count***, excluded **Chg_Dir** interrupt.	Warren Donley
15.0	SCR #8595 03-19-09	• Moved ***Chg_Dir_Count***, and ***Accum_Reverse_Edges*** • Detect reverse crank pulses below 2000 RPM • Process reverse crank pulses below 500 RPM (calibratable) • Delay gap detection after engine speed goes above 20 RPM (calibratable)	Warren Donley

Revision	Date	Record	Author
16.0	SCR #1779 06-25-10	- Modified ***Abs_Edge_Count*** to increment or decrement immediately when the crank changes direction.	Warren Donley
17.0	SCR #1862 07-14-10	- Issued interrupt to CPU when the crank changes direction.	Warren Donley
18.0	SCR #2823 02-15-11	- Added bits ***Dir_Reverse***, ***Dir_Unknown***, and ***Per_Timeout*** to each ***Crit_Edge_Array*** value. - Corrected descriptions of how TCR2 clock control is configured in edge count mode.	Warren Donley