eTPU Function Implementation:

Pulse Width Modulation Input (PWMIE)

MCD – 5402

Revision 5.0

December 18, 2006

DELPHI

CONFIDENTIAL

Microcode Technology
Powertrain Electronics Software Technology
M/S CT-40D, Delphi Electronics & Safety, Kokomo, Indiana 46904-9005

This page intentionally left blank

Table of Contents

This page intentionally left blank

eTPU Function:
PWM Input

1. Introduction

The PWM Input function for the eTPU (PWMIE) is used to calculate the period and high time of a pulse-width modulated (PWM) input signal. The user may select whether the period is measured from falling edge to falling edge or from rising edge to rising edge. The time base used for the period calculation is selectable. This primitive also provides the accumulated period and accumulated high time, which may be used by the host to calculate the average duty cycle, as well as a critical edge counter.

2. System Overview

2.1 Inputs

One pulse-width modulated (PWM) input is required for the operation of this function (see **Figure 1**).

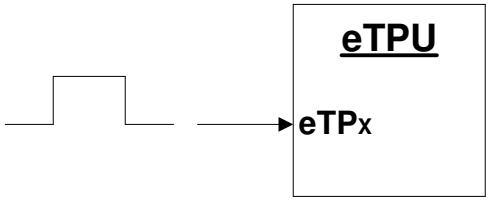


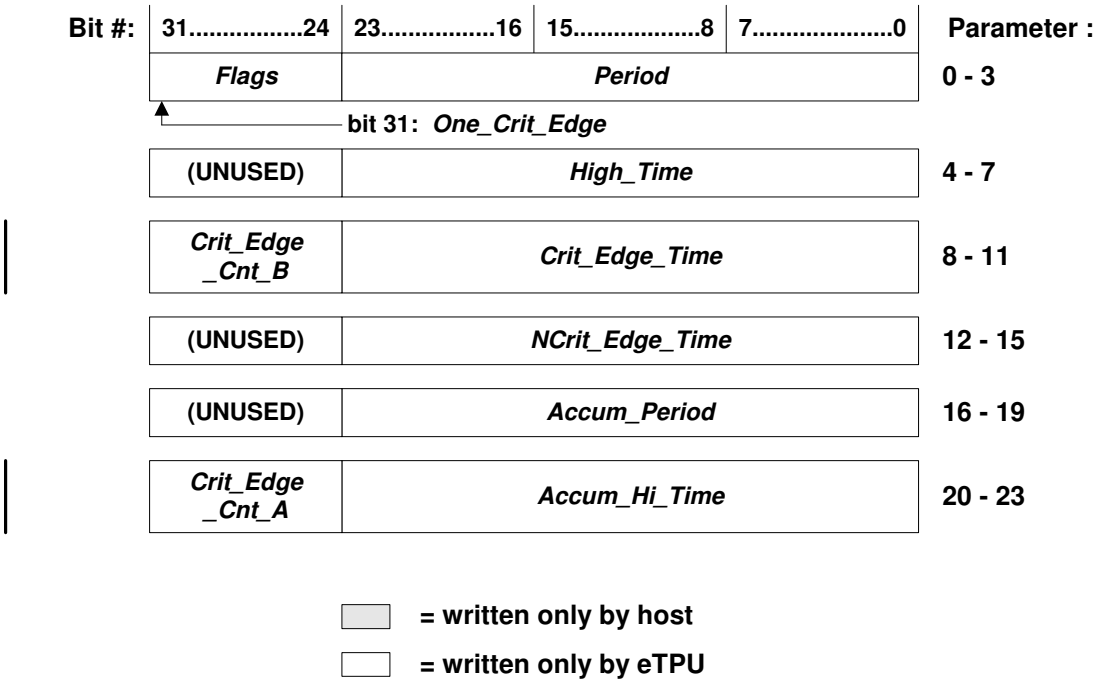
Figure 1 – Hardware Configuration

2.2 Outputs

There are no outputs produced by the operation of this function.

3. Memory Map

The memory map for the PWMIE function is shown in **Figure 2**.



Host Service Requests:

HSR7: Shutdown
HSR6: (Unused)
HSR5: Initialize
HSR4: (Unused)
HSR3: (Unused)
HSR2: (Unused)
HSR1: (Unused)

General Flags:

One_Crit_Edge
0 = No critical edge received
1 = 1+ critical edge received

Entry Point Flags:

Flag0: *Crit_Rise_Flag*
0 = Falling edges critical
1 = Rising edges critical

Flag1: (Unused)

Function Mode Bits:

FM0: *Use_TCR2*
0 = TCR1 time base
1 = TCR2 time base

FM1: *Crit_Rising*
0 = Falling edges critical
1 = Rising edges critical

Figure 2 – Memory Map

3.1 RAM Definition

The parameter RAM definitions are summarized in **Table 1**.

Parameter	Range	Units	CPU Access	eTPU Access
Parameter RAM				
One_Crit_Edge	0 – 1	Flag	R	R/W
Period	0 – 0x800000	TCRx Units	R	R/W
High_Time	0 – 0x800000	TCRx Units	R	R/W
Crit_Edge_Cnt_B	0 – 0xFF	Count Units	R	R/W
Crit_Edge_Time	0 – 0xFFFFFF	TCRx Units	R	R/W
NCrit_Edge_Time	0 – 0xFFFFFF	TCRx Units	R	R/W
Accum_Period	0 – 0xFFFFFF	TCRx Units	R	R/W
Crit_Edge_Cnt_A	0 – 0xFF	Count Units	R	R/W
Accum_Hi_Time	0 – 0xFFFFFF	TCRx Units	R	R/W
Crit_Rise_Flag **(flag0)**	0 – 1	Flag	None	R/W
Use_TCR2 **(FM0)**	0 – 1	Flag	R/W	R
Crit_Rising **(FM1)**	0 – 1	Flag	R/W	R

Table 1 - RAM Definitions

3.1.1 CPU Write / eTPU Read

3.1.1.1 Function Mode Bits

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

Use_TCR2 **(FM0)** Cleared (0) if the time base is to be TCR1. Set (1) if the time base is to be TCR2.

Crit_Rising **(FM1)** Cleared (0) if the falling edge is to be critical (ie, ***Period*** measured from falling edge to falling edge and both ***Period*** and ***High_Time*** updated on the falling edge). Set (1) if the rising edge is to be critical (ie, ***Period***

measured from rising edge to rising edge and both ***Period*** and ***High_Time*** updated on the rising edge).

3.1.2 eTPU Write / CPU Read

The following parameters are written by the eTPU and read by the CPU:

One_Crit_Edge	**(bit 31)** Cleared (0) when an **Initialize** HSR is issued and when a period time-out occurs. Set (1) when the first critical input edge is detected. The polarity of the critical edge is determined by ***Crit_Rising***.
Period	The most recent period of the PWMIE signal. Measured from falling edge to falling edge if ***Crit_Rising*** = 0; measured from rising edge to rising edge if ***Crit_Rising*** = 1.
High_Time	The amount of time between the rising edge and the falling edge. Calculated on the falling edge if ***Crit_Rising*** = 0; calculated on the rising edge if ***Crit_Rising*** = 1.

NOTE: ***Period*** and ***High_Time*** are always updated coherently by the eTPU.

Crit_Edge_Cnt_B	Copy of ***Crit_Edge_Cnt_A,*** only updated coherently with ***Crit_Edge_Time***.
Crit_Edge_Time	The time of the most recent critical edge. The polarity of the critical edge is determined by ***Crit_Rising***.

NOTE: ***Crit_Edge_Cnt_A*** and ***Crit_Edge_Time*** are always updated coherently by the eTPU.

NCrit_Edge_Time	The time of the most recent non-critical edge. The polarity of the non-critical edge is determined by ***Crit_Rising***.
Accum_Period	The accumulated ***Period*** (see above) of the PWMIE signal. Used for Duty Cycle Averaging.
Crit_Edge_Cnt_A	Free-running counter, incremented by one on every critical edge. Updated coherently with ***Accum_Period*** and ***Accum_Hi_Time***.

Accum_Hi_Time The accumulated ***High_Time*** (see above) of the PWMIE signal. Used for Duty Cycle Averaging.

NOTE: ***Accum_Period***, ***Crit_Edge_Cnt_A*** and ***Accum_Hi_Time*** are always updated coherently by the eTPU.

3.1.2.1 Channel Flags

The Channel Flags are invisible to the CPU:

Crit_Rise_Flag **(Flag0)** Set to the same state as ***Crit_Rising*** when an **Initialize** HSR is issued.

Flag1 Unused.

3.2 Initialization

The CPU should initialize the PWMIE function as follows:

1. Write the encoded value for the PWMIE primitive to the channel's field within the correct channel function select register (CFSx). See the Application Implementation document.
2. Set/clear the ***Use_TCR2*** and ***Crit_Rising*** bits appropriately.
3. Issue an **Initialize** Host Service Request (%110).
4. Enable the PWMIE channel by assigning to it a non-zero priority (CPR > %00).

When an **Initialize** HSR is issued, the eTPU will respond by performing the following actions:

1. Configure the PWMIE channel per ***Use_TCR2*** and ***Crit_Rising***.
2. Set ***Crit_Rise_Flag*** **(flag0)** to the same state as ***Crit_Rising***.
3. Set ***One_Crit_Edge*** = 0.
4. Set ***Period*** = 0x800000.
5. If input pin state is low, set ***High_Time*** = 0x000000; if input pin state is high, set ***High_Time*** = 0x800000.

6. Issue an interrupt to the CPU.

4. Operation

4.1 Modes of Operation

The PWMIE function has two different modes of operation, selectable by ***Crit_Rising***: falling edges critical and rising edges critical.

4.1.1 Falling Edges Critical

When ***Crit_Rising*** = 0, the falling edges of the PWM input signal will be considered critical, while the rising edges will be considered non-critical (see **Figure 3**). This means that the eTPU will perform the following actions on every falling edge (except the very first one):

1. Calculate ***Period*** (time since the previous falling edge).
2. Calculate ***High_Time*** and write it coherently with ***Period***.
3. Calculate ***Accum_Period***.
4. Increment ***Crit_Edge_Cnt_A***.
5. Calculate ***Accum_Hi_Time*** and write it coherently with ***Accum_Period*** and ***Crit_Edge_Cnt_A***.
6. Issue an interrupt to the CPU.
7. Issue a DMA trigger to the CPU.
8. Set ***Crit_Edge_Cnt_B*** = ***Crit_Edge_Cnt_A***.
9. Store the edge time as ***Crit_Edge_Time*** and write it coherently with ***Crit_Edge_Cnt_B***.
10. Set up a time-out match to occur 0x800000 timer counts in the future.

On the first falling edge, the eTPU will set ***One_Crit_Edge*** = 1.

On each rising edge, the eTPU will store the edge time as ***NCrit_Edge_Time***.

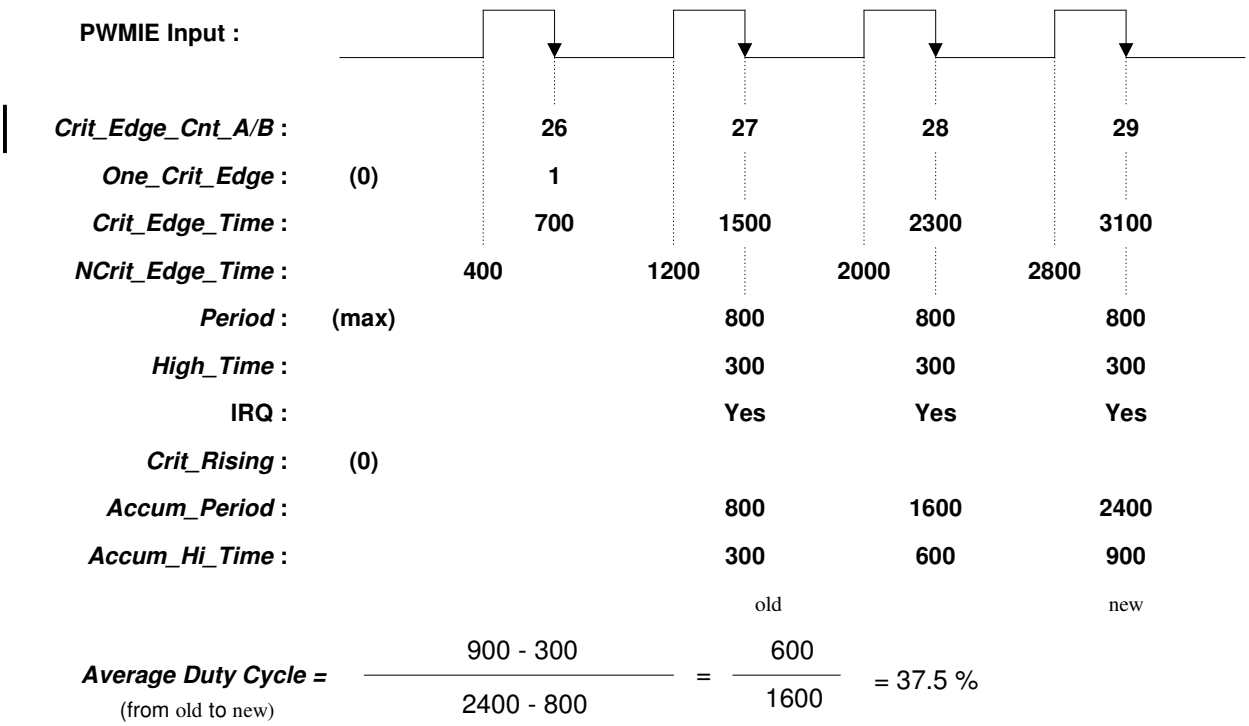


Figure 3 – Falling Edges Critical

4.1.2 Rising Edges Critical

When ***Crit_Rising*** = 1, the rising edges of the PWM input signal will be considered critical, while the falling edges will be considered non-critical (see **Figure 4**). This means that the eTPU will perform the following actions on every rising edge (except the very first one):

1. Calculate ***Period*** (time since the previous rising edge).
2. Calculate ***High_Time*** and write it coherently with ***Period***.
3. Calculate ***Accum_Period***.
4. Increment ***Crit_Edge_Cnt_A***.
5. Calculate ***Accum_Hi_Time*** and write it coherently with ***Accum_Period*** and ***Crit_Edge_Cnt_A***.
6. Issue an interrupt to the CPU.
7. Issue a DMA trigger to the CPU.

8. Set ***Crit_Edge_Cnt_B*** = ***Crit_Edge_Cnt_A***.
9. Store the edge time as ***Crit_Edge_Time*** and write it coherently with ***Crit_Edge_Cnt_B***.
10. Set up a time-out match to occur 0x800000 timer counts in the future.

On the first rising edge, the eTPU will set ***One_Crit_Edge*** = 1.

On each falling edge, the eTPU will store the edge time as ***NCrit_Edge_Time***.

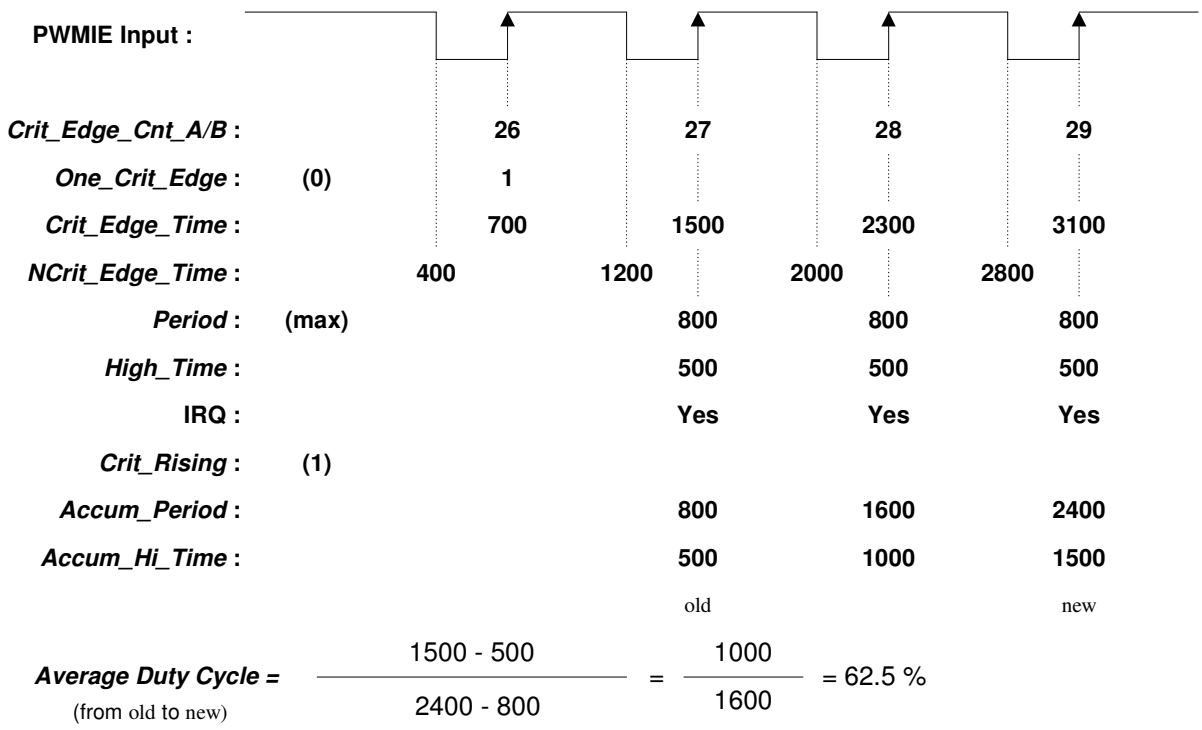


Figure 4 - Rising Edges Critical

4.2 Duty Cycle Averaging

The PWMIE primitive provides 2 parameters for the host CPU to use to calculate average duty cycle, ***Accum_Period*** and ***Accum_Hi_Time***. The CPU must read these parameters coherently.

$$\textbf{Average Duty Cycle} = \frac{\textbf{Accum\_Hi\_Time}_{new} - \textbf{Accum\_Hi\_Time}_{old}}{\textbf{Accum\_Period}_{new} - \textbf{Accum\_Period}_{old}}$$

For falling edges critical, if it is desired to have the average "low time" duty cycle:

Inverse Average Duty Cycle = 100% - Average Duty Cycle

4.3 Period Time-Out

If 0x800000 timer counts elapse between critical edges of the PWM input signal, the eTPU will detect a period time-out condition and perform the following actions:

1. Set ***One_Crit_Edge*** = 0.
2. Set ***Period*** = 0x800000.
3. If input pin state is low, set ***High_Time*** = 0x000000; if input pin state is high, set ***High_Time*** = 0x800000.
4. Issue an interrupt to the CPU.

4.4 Limitations of the PWMIE Function

The PWMIE function is designed to accurately and reliably capture up to two closely spaced input edges. If more than two closely spaced edges (noise spikes) occur, the PWMIE function may not perform correctly. An edge or edges may be missed entirely, or a critical edge may be taken to be a non-critical edge (or vice-versa), etc. This in turn will cause ***Period*** and ***High_Time*** to be calculated incorrectly.

.

4.5 Global Exception

Under certain error conditions, the PWMIE function will issue a global exception to the host and write its channel number to the global variable ***Cause_Of_Exception***. Below are the conditions that cause PWMIE to generate a global exception:

1. An HSR is issued to the PWMIE function that has not been defined.
2. A link to the PWMIE function occurs.

4.6 Shutting Down the PWMIE Function

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

4.7 Switching from PWMIE to Another eTPU Function

To switch from the PWMIE function to another eTPU function, the CPU should shut down the PWMIE function (see **Section 4.6**) and then follow the directions for initializing the new function.

4.8 Function Latency

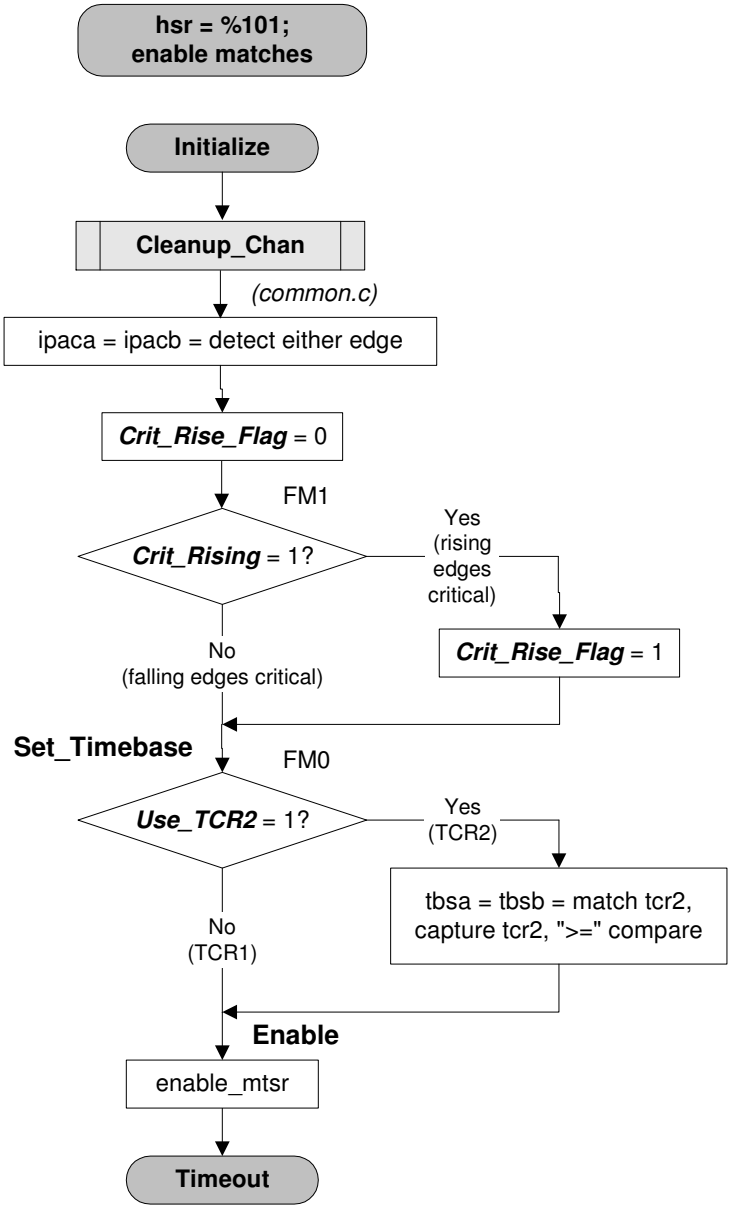
The worst-case execution times for the various threads of the PWMIE function are shown below (see the Application Implementation document to convert microcycles into real time based on crystal frequency). Note that these values do not take into account latencies due to other functions in the application, priorities, etc.

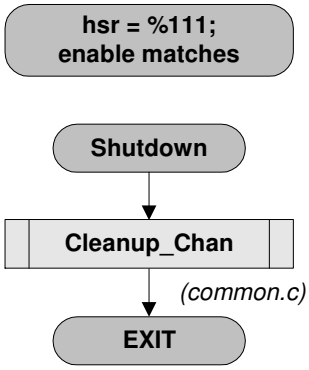
PWMIE Thread	Worst-Case µcycles *	RAM Accesses
Shutdown HSR (%111)	**8**	**0**
Initialize HSR (%101)	**23**	**3**
Non-Critical Edge	**27**	**10**
Critical Edge	**29**	**10**
Non-Critical and Critical Edges	**26**	**10**
Critical and Non-Critical Edges	**25**	**10**

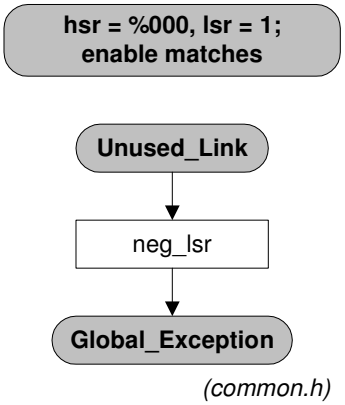
* These numbers do not consider potential collisions while accessing PRAM.

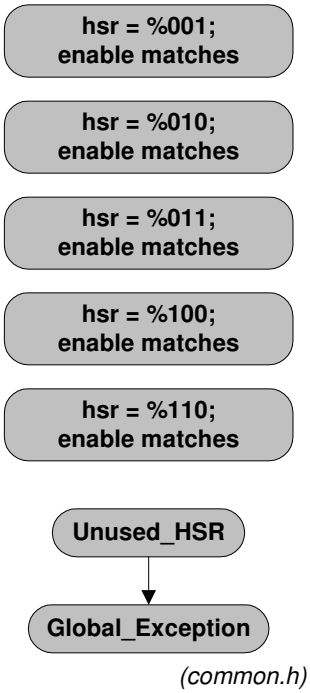
PWMIE primitive code size = 83 microcode instructions

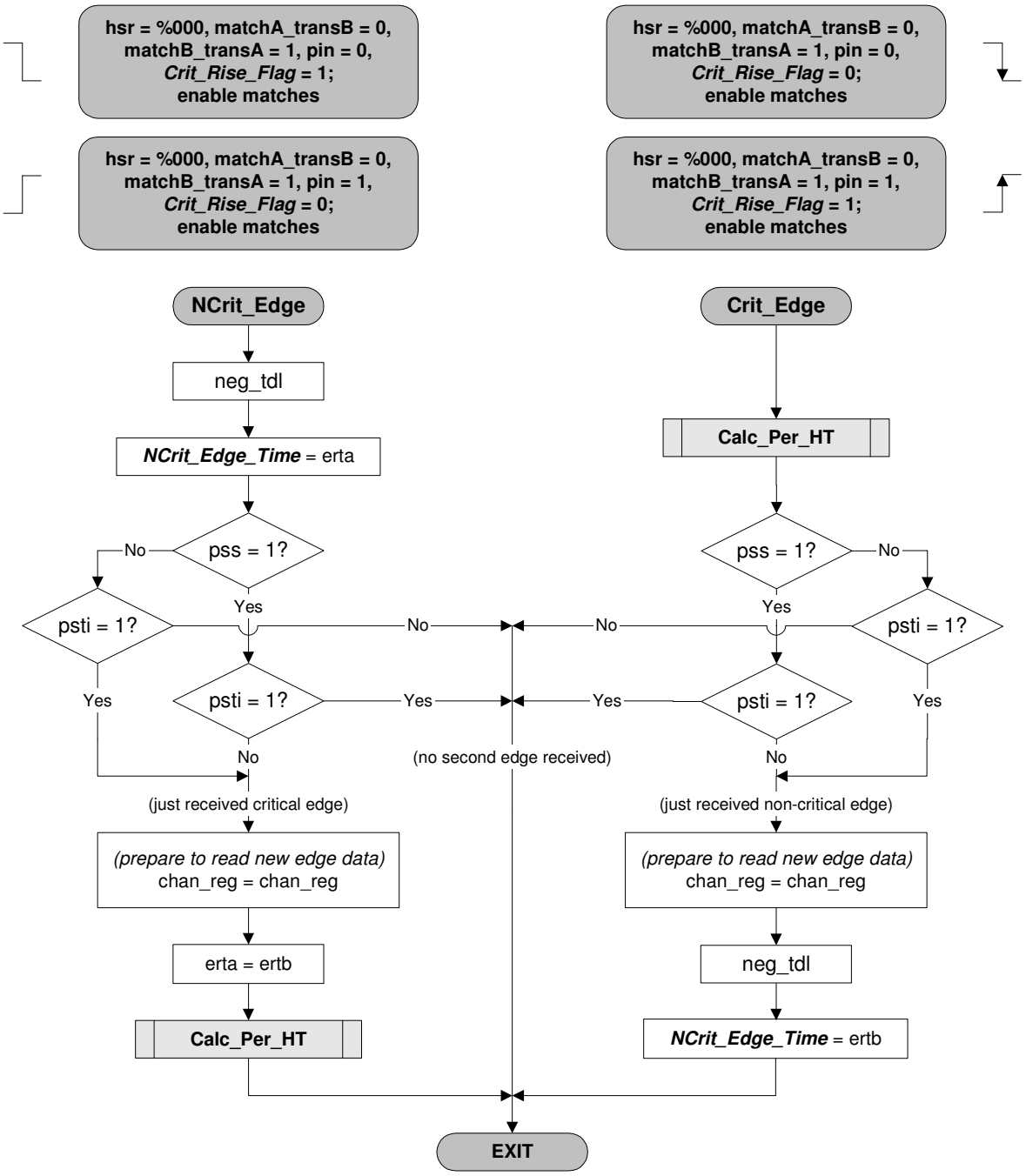
5. Flowcharts

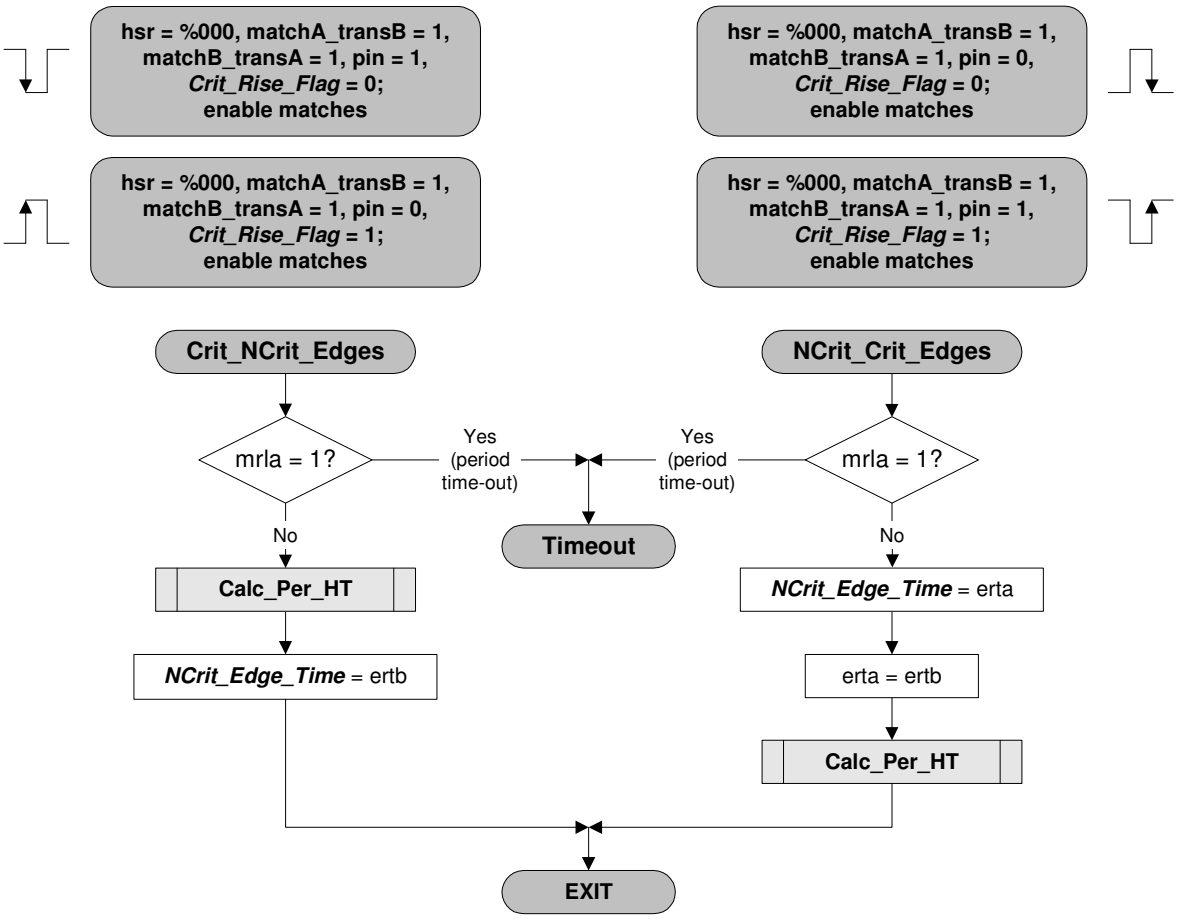


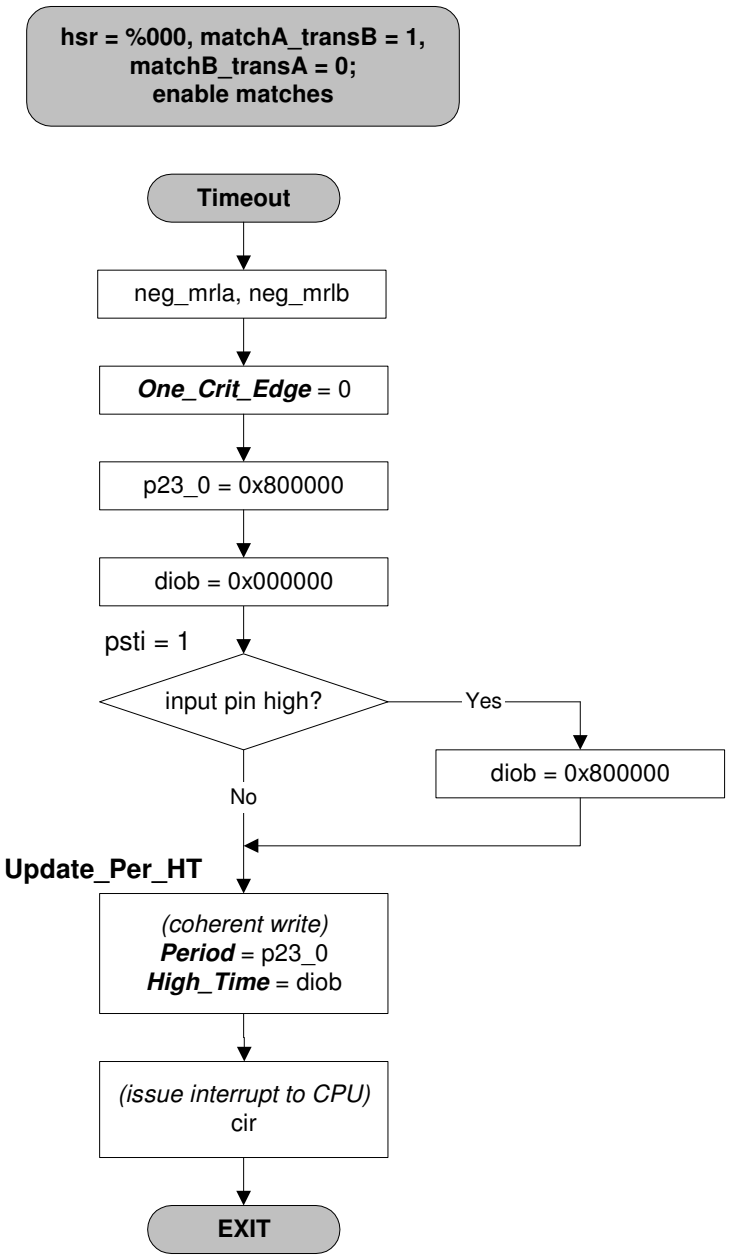

hsr = %111;
enable matches
Shutdown
Cleanup_Chan
(common.c)
EXIT

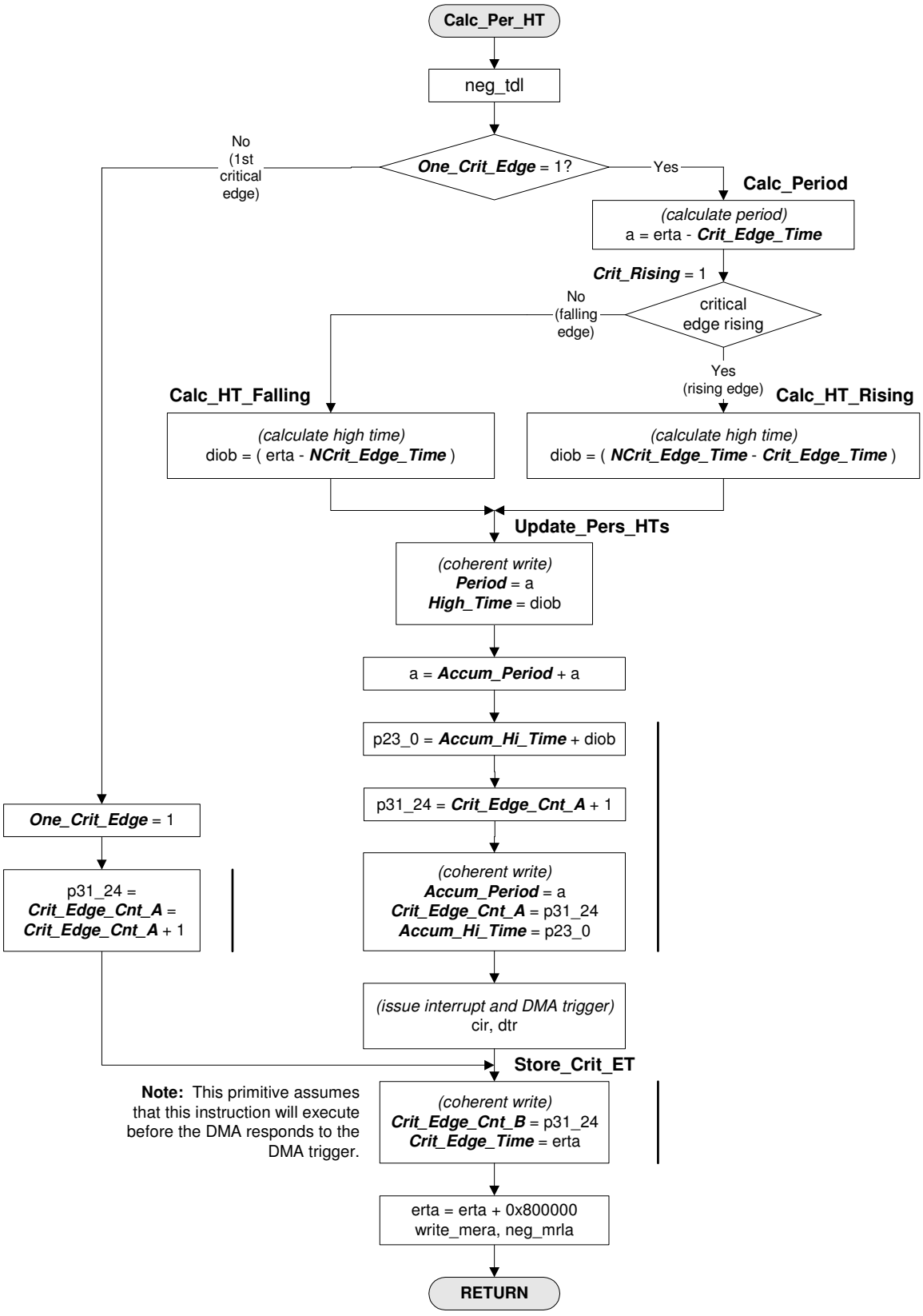

hsr = %000, lsr = 1;
enable matches
Unused_Link
neg_lsr
Global_Exception
(common.h)


hsr = %001;
enable matches
hsr = %010;
enable matches
hsr = %011;
enable matches
hsr = %100;
enable matches
hsr = %110;
enable matches
Unused_HSR
Global_Exception
(common.h)


hsr = %000, matchA_transB = 0, matchB_transA = 1, pin = 0, Crit_Rise_Flag = 1; enable matches
hsr = %000, matchA_transB = 0, matchB_transA = 1, pin = 0, Crit_Rise_Flag = 0; enable matches
hsr = %000, matchA_transB = 0, matchB_transA = 1, pin = 1, Crit_Rise_Flag = 0; enable matches
hsr = %000, matchA_transB = 0, matchB_transA = 1, pin = 1, Crit_Rise_Flag = 1; enable matches
NCrit_Edge
neg_tdl
NCrit_Edge_Time = erta
pss = 1?
No
Yes
psti = 1?
No
Yes
psti = 1?
No
Yes
(just received critical edge)
(prepare to read new edge data)
chan_reg = chan_reg
erta = ertb
Calc_Per_HT
Crit_Edge
Calc_Per_HT
pss = 1?
No
Yes
psti = 1?
No
Yes
psti = 1?
No
Yes
(just received non-critical edge)
(prepare to read new edge data)
chan_reg = chan_reg
neg_tdl
NCrit_Edge_Time = ertb
(no second edge received)
EXIT


hsr = %000, matchA_transB = 1,
matchB_transA = 1, pin = 1,
Crit_Rise_Flag = 0;
enable matches
hsr = %000, matchA_transB = 1,
matchB_transA = 1, pin = 0,
Crit_Rise_Flag = 0;
enable matches
hsr = %000, matchA_transB = 1,
matchB_transA = 1, pin = 0,
Crit_Rise_Flag = 1;
enable matches
hsr = %000, matchA_transB = 1,
matchB_transA = 1, pin = 1,
Crit_Rise_Flag = 1;
enable matches
Crit_NCrit_Edges
NCrit_Crit_Edges
mrla = 1?
Yes
(period
time-out)
Yes
(period
time-out)
mrla = 1?
No
No
Timeout
Calc_Per_HT
NCrit_Edge_Time = erta
NCrit_Edge_Time = ertb
erta = ertb
Calc_Per_HT
EXIT


hsr = %000, matchA_transB = 1,
matchB_transA = 0;
enable matches
Timeout
neg_mrla, neg_mrlb
One_Crit_Edge = 0
p23_0 = 0x800000
diob = 0x000000
psti = 1
input pin high?
Yes
diob = 0x800000
No
Update_Per_HT
(coherent write)
Period = p23_0
High_Time = diob
(issue interrupt to CPU)
cir
EXIT


Calc_Per_HT
neg_tdl
No (1st critical edge)
One_Crit_Edge = 1?
Yes
Calc_Period
(calculate period)
a = erta - Crit_Edge_Time
Crit_Rising = 1
No (falling edge)
critical edge rising
Yes (rising edge)
Calc_HT_Falling
Calc_HT_Rising
(calculate high time)
diob = (erta - NCrit_Edge_Time)
(calculate high time)
diob = (NCrit_Edge_Time - Crit_Edge_Time)
Update_Pers_HTs
(coherent write)
Period = a
High_Time = diob
a = Accum_Period + a
p23_0 = Accum_Hi_Time + diob
p31_24 = Crit_Edge_Cnt_A + 1
One_Crit_Edge = 1
p31_24 = Crit_Edge_Cnt_A = Crit_Edge_Cnt_A + 1
(coherent write)
Accum_Period = a
Crit_Edge_Cnt_A = p31_24
Accum_Hi_Time = p23_0
(issue interrupt and DMA trigger)
cir, dtr
Store_Crit_ET
Note: This primitive assumes that this instruction will execute before the DMA responds to the DMA trigger.
(coherent write)
Crit_Edge_Cnt_B = p31_24
Crit_Edge_Time = erta
erta = erta + 0x800000
write_mera, neg_mrla
RETURN

6. Revision Log

6.1 Document Revision Numbering

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

6.2 Revision History

Revision	Date	Record	Author
1.0	04-23-04	Initial release of documentation. PWMIE-1.0 WAS NEVER TESTED!!!	Warren Donley
2.0	05-11-04	Updated and tested PWMIE.	William Ditty
2.1	06-21-04	Correct flowchart branches for **input_pin_state_changed** decisions.	William Ditty
3.0 (SCR 3942)	02-23-05	Added duty cycle averaging.	Steven Hughes
4.0 (SCR 4581)	02-24-06	Converted from C to assembly language. Added **Cleanup_Chan** to **Initialize** HSR and deleted neg_tdl from **Timeout** routine.	Warren Donley
5.0 (SCR 5190)	12-18-06	Added critical edge counter, updated coherently with ***Accum_Period*** and ***Accum_Hi_Time***, and a copy, updated coherently with ***Crit_Edge_Time***.	Warren Donley