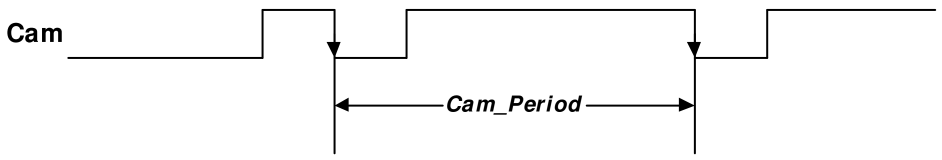
eTPU Primitive Implementation:

Cam Input Processing Algorithm for the eTPU

(CAMe)



MCD-5411

Revision 6.2

May 20, 2010

DELPHI

CONFIDENTIAL

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

Table of Contents

Cam Input Processing Algorithm for the eTPU

1. Introduction

This document describes the cam input algorithm as implemented in the eTPU. The CAMe algorithm will respond to both edges of the input signal, processing one edge as a critical edge and the other as a non-critical edge. The host CPU specifies the polarity of these edges. Either or both edges can be configured to generate an interrupt to the host (the current state of the pin indicates which edge to associate with the interrupt). When a non-critical edge occurs, an edge count and various edge times are updated. When a critical edge occurs, the period, the edge count, and various edge times are updated. Information from the engine speed sensor input primitive is also sampled and stored.

The CAMe primitive will not calculate cam phase angle, but will save the TCR2 value at the critical edge and at the non-critical edge. If TCR2 is set up as the angle clock, then this snapshot represents an extrapolated cam phase, subject to accel/decel errors. If an interpolation is desired, the CPU will be expected to use the EPPE array of buffered crank edge times (or keep its own array) to provide the crank information required for the cam phase determination.

The CAMe primitive can be configured to provide a replicated cam output, with a fixed lag time and polarity set by the user, on another eTPU channel.

2. System Overview

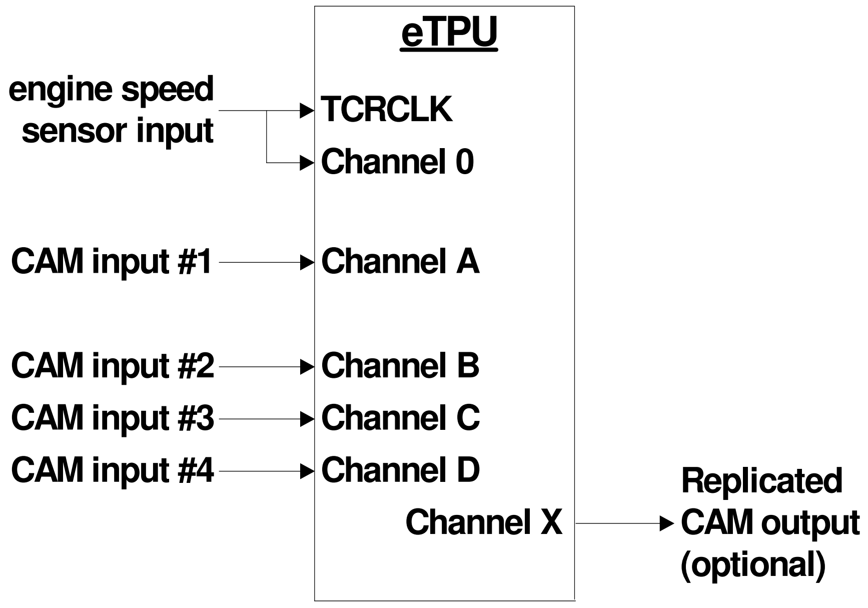


Figure 1 – CAMe Mechanization

2.1 Inputs

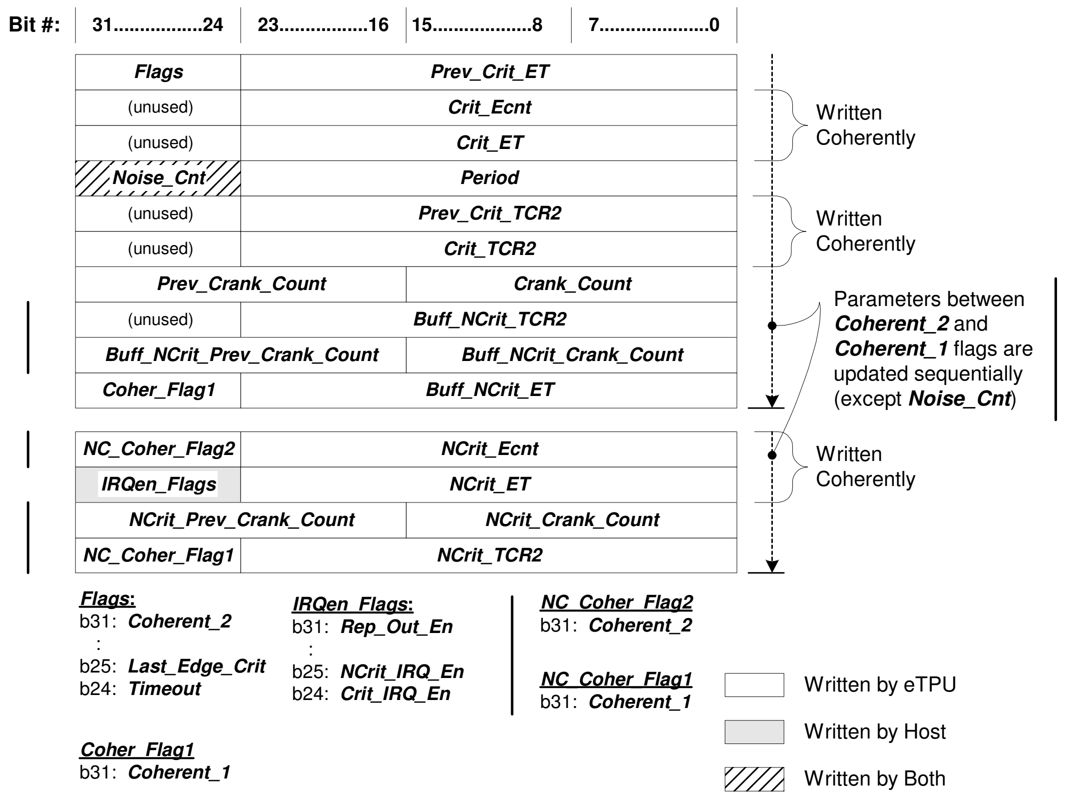
The CAMe algorithm requires one encoded cam position sensor input. Other cam inputs can be present, and can reside on any input channel on any eTPU in the system, as long as the eTPU shares parameter RAM with the eTPU processing the engine speed sensor input (for the purpose of capturing the critical crank count). See **Figure 1**. (Note, however, that the EPPe algorithm will not be able to sample the cam pin state if the cam input is on a different eTPU from the crank input.) This algorithm also requires a 58X medium-resolution engine position sensor input (see **MCD-5408** for details).

2.2 Outputs

If the replicated cam output feature is enabled by the user, the CAMe algorithm requires one hardware output to generate this signal. Otherwise, the CAMe algorithm requires no hardware outputs.

3. Memory Map

3.1 Channel RAM Parameters



Bit #:	31.................24	23.................16	15...................8	7.....................0
	Flags	***Prev_Crit_ET***		
	(unused)	***Crit_Ecnt***		
	(unused)	***Crit_ET***		
	▨ ***Noise_Cnt***	***Period***		
	(unused)	***Prev_Crit_TCR2***		
	(unused)	***Crit_TCR2***		
	Prev_Crank_Count		***Crank_Count***	
	(unused)	***Buff_NCrit_TCR2***		
	Buff_NCrit_Prev_Crank_Count		***Buff_NCrit_Crank_Count***	
	Coher_Flag1	***Buff_NCrit_ET***		
	NC_Coher_Flag2	***NCrit_Ecnt***		
	IRQen_Flags	***NCrit_ET***		
	NCrit_Prev_Crank_Count		***NCrit_Crank_Count***	
	NC_Coher_Flag1	***NCrit_TCR2***		

Written Coherently
Written Coherently
Parameters between Coherent_2 and Coherent_1 flags are updated sequentially (except Noise_Cnt)
Written Coherently

Flags:
b31: ***Coherent_2***
:
b25: ***Last_Edge_Crit***
b24: ***Timeout***

Coher_Flag1
b31: ***Coherent_1***

IRQen_Flags:
b31: ***Rep_Out_En***
:
b25: ***NCrit_IRQ_En***
b24: ***Crit_IRQ_En***

NC_Coher_Flag2
b31: ***Coherent_2***

NC_Coher_Flag1
b31: ***Coherent_1***

☐	Written by eTPU
▒	Written by Host
▨	Written by Both

Host Service Requests:
HSR 7: Shutdown
HSR 6: (Unused)
HSR 5: Initialize crit edge falling
HSR 4: (Unused)
HSR 3: Initialize crit edge rising
HSR 2: (Unused)
HSR 1: (Unused)

Entry Point Flags:
Flag0: ***Rise_Crit***
0 = falling edge critical
1 = rising edge critical
Flag1: (unused)

Channel Mode:
sm_st - Single match, single transition
match A - CAM timeout match
match B - (Disabled)
transition A - either CAM edge
transition B - (Disabled)

Function Mode Bits:
FM0 :***Use_TCR2***
0 = TCR1 timebase
1 = TCR2 timebase
FM1: ***Rep_Out_Inv***
0 = Rep. cam output mirrors input
1 = Rep. cam output inverts input

Interrupts:
Cam Critical Edge Occurred Interrupt
Cam Non-critical Edge Occurred Interrupt

Entry Table:
Standard Channel Condition Encoding Scheme

Figure 1 - Parameter RAM Map

3.2 Global RAM

The CAMe primitive accesses the following global RAM parameters: See the Application Implementation document for the actual memory locations of these parameters.

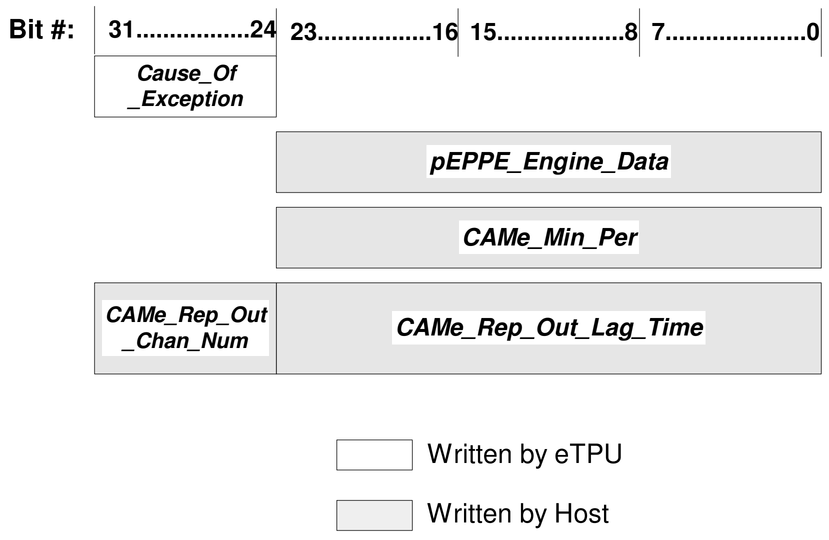


Figure 3 - Global RAM for CAMe

3.3 RAM Definitions

3.3.1 Global RAM Definition

Parameter	Range	Units	CPU Access	eTPU Access
Cause_Of_Exception	0 – 0x1F	eTPU Channel #	R	W
pEPPE_Engine_Data	0 – 0xFFFFFF	eTPU address	W	R
CAMe_Min_Per	0 – 0xFFFFFF	TCRx Units	W	R
CAMe_Rep_Out_Chan_Num	0 – 0x1F	eTPU Channel #	W	R
CAMe_Rep_Out_Lag_Time	0 – 0x7FFFFF	TCRx Units	W	R

Table 1 - Global RAM for CAMe

3.3.2 Parameter RAM Definition

Parameter	Range	Units	CPU Access	eTPU Access
Buff_NCrit_Crank_Count	0 – 0xFFFF	Count Units	R	R/W
Buff_NCrit_Prev_Crank_Count	0 – 0xFFFF	Count Units	R	R/W
Buff_NCrit_ET	0 – 0xFFFFFF	TCRx Units	R	R/W
Buff_NCrit_TCR2	0 – 0xFFFFFF	EPP Counts * 256 Units	R	R/W
Coherent_1	0 – 1	Flag	R	R/W
Coherent_2	0 – 1	Flag	R	R/W
Crank_Count	0 – 0xFFFF	Count Units	R	R/W
Crit_Ecnt	0 – 0xFFFFFF	Count Units	R	R/W
Crit_ET	0 – 0xFFFFFF	TCRx Units	R	R/W
Crit_IRQ_En	0 – 1	Flag	R/W	R
Crit_TCR2	0 – 0xFFFFFF	EPP Counts * 256 Units	R	R/W
Last_Edge_Crit	0 – 1	Flag	R	R/W
Ncrit_Crank_Count	0 – 0xFFFF	Count Units	R	R/W
NCrit_Ecnt	0 – 0xFFFFFF	Count Units	R	R/W
NCrit_ET	0 – 0xFFFFFF	TCRx Units	R	R/W
NCrit_IRQ_En	0 – 1	Flag	R/W	R
NCrit_Prev_Crank_Count	0 – 0xFFFF	Count Units	R	R/W
NCrit_TCR2	0 – 0xFFFFFF	EPP Counts * 256 Units	R	R/W
Noise_Cnt	0 – 0xFF	Count Units	R/W	R/W
Period	0 – 0xFFFFFF	TCRx Units	R	R/W
Prev_Crank_Count	0 – 0xFFFF	Count Units	R	R/W
Prev_Crit_ET	0 – 0xFFFFFF	TCRx Units	R	R/W
Prev_Crit_TCR2	0 – 0xFFFFFF	EPP Counts * 256 Units	R	R/W
Rep_Out_En	0 – 1	Flag	R/W	R
Rep_Out_Inv	0 – 1	Function Mode bit 1	W	R
Rise_Crit	0 – 1	Channel Flag 0	N/A	R/W
Timeout	0 – 1	Flag	R	R/W
Use_TCR2	0 – 1	Function Mode bit 0	W	R

3.3.3 Parameter RAM Written by Host CPU

3.3.3.1 Global Parameters

pEPPE_Engine_Data	(24 bits) Pointer to EPPE engine position data (edge time, edge count, period) used by the CAMe function. Note that ***pEPPE_Engine_Data*** must be set equal to the address of the EPPE parameter ***Crit_Edge_Time*** (see **MCD-5408**).
CAMe_Min_Per	(24 bits) This parameter is intended to limit the rate at which the eTPU can interrupt the CPU if some fault condition creates a high frequency signal at the CAM input. If the time since the last critical edge is less than ***CAMe_Min_Per***, then the eTPU will ignore the current edge and no variables will be updated. Non-critical edges will not be ignored, but only one non-critical edge interrupt will be issued (if enabled) for each critical edge when ***CAMe_Min_Per*** is any non-zero value. Units correspond with the ***Use_TCR2*** bit.
CAMe_Rep_Out_Chan_Num	(8 bits) The number of the eTPU channel on which the replicated cam output signal is to occur. Only valid when ***Rep_Out_En*** = 1.
CAMe_Rep_Out_Lag_Time	(24 bits) The amount of time by which the replicated cam output signal will lag the cam input signal. Units correspond with the ***Use_TCR2*** bit. Only valid when ***Rep_Out_En*** = 1.

Note: ***CAMe_Rep_Out_Lag_Time*** **must never be set to a value that exceeds the minimum possible period (either rising edge to rising edge or falling edge to falling edge) of the CAMe input signal, as this will result in an incorrect replicated cam output signal.**

3.3.3.2 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:

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

Rep_Out_Inv	**FM1** - Set (1) if the replicated cam output signal is to invert the input signal; cleared (0) if the replicated cam output signal is to mirror the input signal. Only valid if ***Rep_Out_En*** = 1.

Note: Any time *Rep_Out_Inv* is changed, the new value will not take effect until an Initialize HSR is issued.

3.3.3.3 Channel Parameters

Rep_Out_En	A flag indicating the host desires to enable a replicated cam output on the eTPU channel indicated by ***CAMe_Rep_Out_Chan_Num***.

Note: This flag is located in the upper byte of an eTPU parameter and extreme care should be exercised to prevent over-writing the lower 24 bits.

NCrit_IRQ_En	A flag indicating the host desires an interrupt from the eTPU when it completes the processing of non-critical cam edges.

Note: This flag is located in the upper byte of an eTPU parameter and extreme care should be exercised to prevent over-writing the lower 24 bits.

Crit_IRQ_En	A flag indicating the host desires an interrupt from the eTPU when it completes the processing of critical cam edges.

Note: This flag is located in the upper byte of an eTPU parameter and extreme care should be exercised to prevent over-writing the lower 24 bits.

3.3.4 Parameter RAM Written by BOTH the eTPU and Host

Noise_Cnt	This parameter is intended to indirectly count critical edge events that are ignored due to the ***Min_Per*** requirement. It actually counts noise pulses as defined by a non-critical edge out of sequence (two in a row). It does not look for critical edges out of sequence. The CPU may clear ***Noise_Cnt*** and the eTPU limits it to 0xFF.

3.3.5 Parameter RAM Written by eTPU

3.3.5.1 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.

3.3.5.2 Channel Parameters

Coherent_2	One of two flags that can indicate data coherency when the Host or the other micro engine reads parameters from this primitive.
Last_Edge_Crit	Internal flag indicates the last edge processed was a critical edge.
Timeout	Internal flag used by the CAMe primitive to indicate when the time between two critical CAM edges has exceeded the 24 bit limit. When this occurs, the ***Period*** will be set to its maximum value of 0xFFFFFF and is not calculated on the following critical edge.
Prev_Crit_ET	The time of the previous critical cam edge in units specified by ***Use_TCR2***.
Crit_Ecnt	A counter that is incremented during the processing of every critical cam edge. Written coherently with ***Crit_ET*** by the eTPU. (Note that ***Crit_Ecnt*** is <u>independent</u> of ***NCrit_Ecnt***.)
Crit_ET	The time of the critical cam edge in units specified by ***Use_TCR2***. Written coherently with ***Crit_Ecnt*** by the eTPU.
Period	The time between the two most recent critical cam input edges in units specified by ***Use_TCR2***.
Prev_Crit_TCR2	The old value of ***Crit_TCR2*** that was captured at the previous critical cam edge. Written coherently with ***Crit_TCR2*** by the eTPU.

Crit_TCR2	A snapshot of the TCR2 taken at the critical cam edge, which is also the cam phase. If the accuracy of this cam phase is not adequate, the CPU may use the upper word of this data as a critical edge count to determine which crank data to use in the cam phase calculation, but remember that this data is not coherent with the crank data. Written coherently with ***Prev_Crit_TCR2*** by the eTPU.

Note: The upper word could be one count low if the angle clock is bursting!

Prev_Crank_Count	The previous value of ***Crank_Count*** which was captured at the previous critical cam edge. See the disclaimer in the note below.
Crank_Count	A snapshot of the last processed crank edge count that is captured during the processing of the critical cam edge (last crank edge processed by the crank primitive may not be the one that occurred before the cam edge- see note). The Crank count is accessed via the Global RAM pointer ***pEPPE_Engine_Data***.

Note: *Crank_Count* can be in error by + or – one count when the critical cam edge occurs very close to the critical crank edge, due to the scheduler in the eTPU.

Buff_NCrit_TCR2	Buffered value of ***NCrit_TCR2*** so as to be coherent with other critical edge data.

Note: The upper word could be one count low if the angle clock is bursting!

Buff_NCrit_Prev_Crank_Count	Buffered value of ***NCrit_Prev_Crank_Count*** so as to be coherent with other critical edge data. See the disclaimer in the note below.
Buff_NCrit_Crank_Count	Buffered value of ***NCrit_Crank_Count*** so as to be coherent with other critical edge data. See the crank count disclaimer in the note below.
Coherent_1	One of two flags that can indicate data coherency when the Host or the other micro engine reads parameters from this primitive.

Buff_NCrit_ET	A copy of ***NCrit_ET*** taken during the processing of the critical cam edge.
NCrit_Ecnt	A counter that is incremented during the processing of every non-critical cam edge. Written coherently with ***NCrit_ET*** by the eTPU. (Note that ***NCrit_Ecnt*** is independent of ***Crit_Ecnt***.)
NCrit_ET	The time of the non-critical cam edge in units specified by ***Use_TCR2***. Written coherently with ***NCrit_Ecnt*** by the eTPU.
NCrit_Prev_Crank_Count	The previous value of ***Crank_Count*** which was captured at the previous non-critical cam edge. See the disclaimer in the note below.
NCrit_Crank_Count	A snapshot of the last processed crank edge count that is captured during the processing of the non-critical cam edge (last crank edge processed by the crank primitive may not be the one that occurred before the cam edge- see note). The Crank count is accessed via the Global RAM pointer ***pEPPE_Engine_Data***.

Note: *NCrit_Crank_Count* can be in error by + or – one count when the non-critical cam edge occurs very close to the critical crank edge, due to the scheduler in the eTPU.

NCrit_TCR2	A snapshot of the TCR2 taken at the non-critical cam edge.

Note: The upper word could be one count low if the angle clock is bursting!

3.3.6 Parameters Used by eTPU Only

3.3.6.1 Channel Flags

Channel flag0 is used to provide separate entry points for critical and non-critical edges.

Rise_Crit (Flag0)	Set (1) by the eTPU in the HSR designated for critical rising edges on the cam input; cleared (0) in the HSR designated for critical falling edges.

(Flag1 is not used.)

3.4 Parameter Coherency

The parameters provided in this primitive can be accessed in different combinations for different purposes. Any access by the Host or the other eTPU engine should take the appropriate steps necessary to insure coherent data transfer.

3.4.1 Dual Parameter Coherency

For any set of two parameters indicated in Figure 1 as written coherently, the Host can use the CDC (Coherent Dual-parameter Controller) feature. This feature requires the Host to reserve a Transfer Parameter Area in SPRAM (Shared Parameter RAM). Alternatively, with interrupts disabled, the Host could read the two parameters twice and check to see if the two readings are the same. If not, one more read of the two parameters should result in coherent data, before enabling interrupts.

Dual back-to-back parameter accesses are not guaranteed to be atomic between the two micro engines. If the both micro engines access the two parameters back-to-back *in the same order*, however, coherency is assured.

3.4.2 Multiple Parameter Coherency

For other groupings of data, including groups larger than two parameters, the following sequence should be used. It is recommended that the Host inhibit interrupts while executing this sequence.

1. The first parameter read must include the ***Coherent_1*** flag.
2. Any other parameters can then be read.
3. The last parameter read must include the ***Coherent_2*** flag.
4. If ***Coherent_1*** is the same as ***Coherent_2***, then the data just read is coherent. If the two states are not the same, then the sequence must be repeated.

The sequence above works because the CAMe primitive will update the parameters in the opposite order, beginning with ***Coherent_2*** flag and ending with ***Coherent_1*** flag. The sequence should probably be repeated only two or three times before giving up and using old data. Note that ***Noise_Cnt*** does not require coherency.

Another option to insure coherent data is to disable the channel by writing 0 level priority, wait until the channel is not being serviced, read the data, then re-enable the channel.

4. Operation

An example illustrating how the various CAMe parameters operate and interact is shown in **Figure 4.**

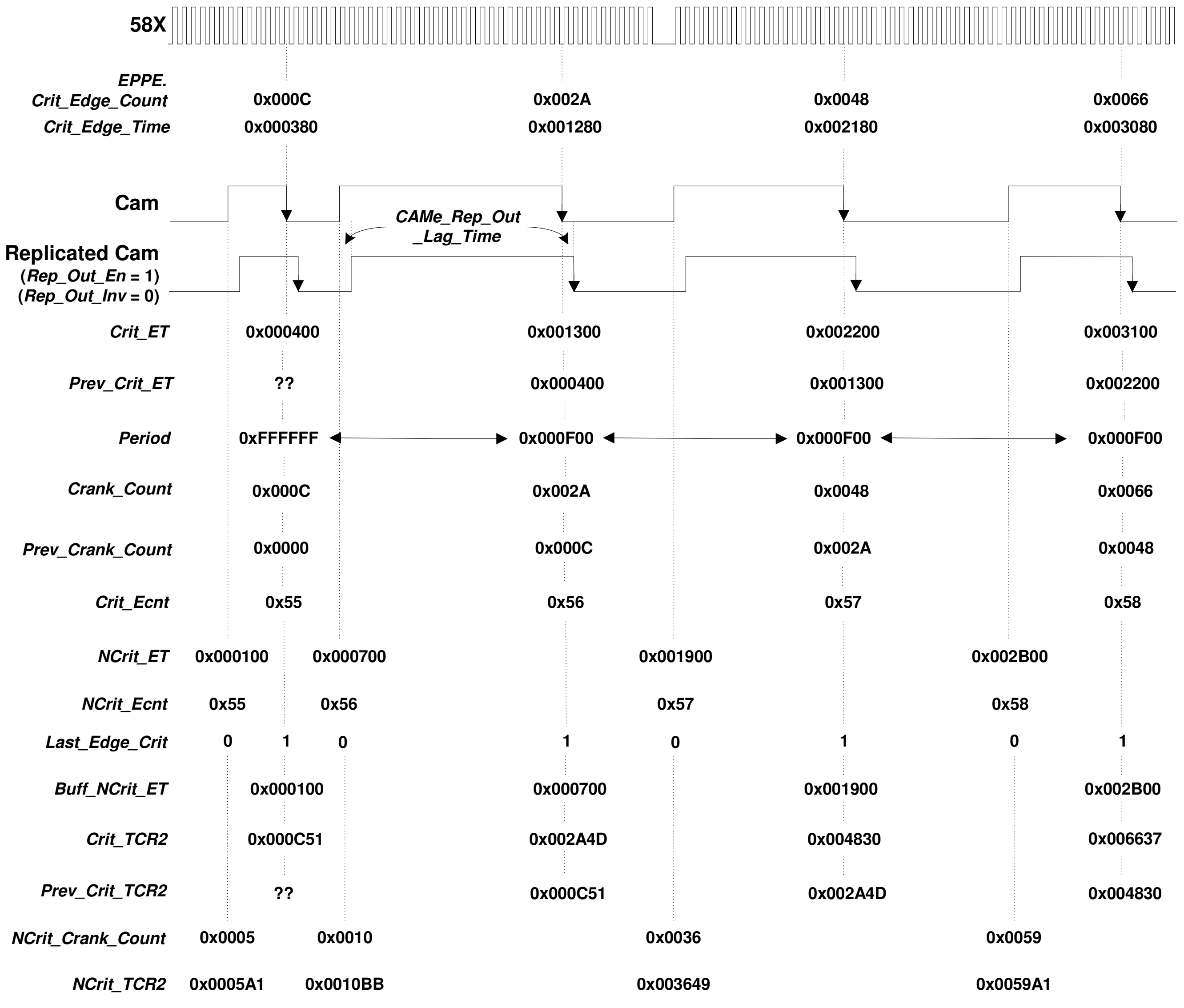


Figure 4 - Example of CAMe Operation (Falling Edges Critical)

4.1 Initialization

The CPU should initialize the CAMe function as follows:

1. Set ***CAMe_Min_Per*** to the desired value in global ram (0 disables).
2. Parameter RAM should be cleared.
3. If a replicated cam output is desired, set ***Rep_Out_En*** = 1, select the polarity of the signal with ***Rep_Out_Inv***, select the channel of the signal with ***CAMe_Rep_Out_Chan_Num***, and set ***CAMe_Rep_Out_Lag_Time*** to the desired value.
4. Enable interrupts with flags in parameter ram:
   a. If an interrupt is desired for critical edges, set ***Crit_IRQ_En*** = 1, otherwise set it to 0.
   b. If an interrupt is desired for non-critical edges, set ***NCrit_IRQ_En*** = 1, otherwise set it to 0.
5. Select the desired time base by setting Function Mode bit0 (FM0) to the appropriate value with the definitions provided:
   ```
   #define CAME_TIMEBASE_TCR1    (0)
   #define CAME_TIMEBASE_TCR2    (1)
   ```
6. Initialize the ETPU Channel Configuration Register (ETPUCxCR):
   a. Select the standard entry table condition for the CAMe primitive by writing a 0 to the ETCS field. This value is passed from the microcode set as:
      ```
      #define CAME_ENTRYTABLE_TYPE   ( 0 )
      ```
   b. Write the CAMe function number to the Channel Function Select (CFS) field. The following definition is provided:
      ```
      #define CAME _FUNCTION_NUM      ( x ),
      ```
      where 'x' value depends on the microcode set.
   c. Write the CAMe channel's parameter base address to the CPBA field.
7. Issue an **Initialize** Host Service Request by writing the appropriate value to the HSR field in the ETPUCxHSRR register. This will define the polarity of the critical edges with the following definition provided.
   ```
   #define CAME _HSR_INIT_CRIT_RISE      ( 3 )
   ```

#define CAME _HSR_INIT_CRIT_FALL (5)

8. Enable the channel by assigning a priority to the CPR field of the ETPUCxCR register. The eTPU will now schedule the above HSR for execution.

4.2 Critical Input Edge

When a critical edge occurs, the CAMe primitive will do the following:

1. Ignore the event if the time since the last critical edge is less than ***CAMe_Min_Per***.

2. Compute and save the period if this is not the first pulse since a timeout.

3. Save the edge time and increment the edge count.

4. Capture the TCR2 value.

5. Save a snapshot of the critical crank count from the EPPE primitive using the global pointer ***pEPPE_Engine_Data***.

6. Update the buffered values of the non-critical edge time, TCR2 snapshot, and crank count parameters.

7. Re-initialize the timeout.

8. Generate an interrupt for the host if enabled.

9. If ***Rep_Out_En*** = 1, set up a critical output edge for the replicated cam output signal on the channel indicated by ***CAMe_Rep_Out_Chan_Num***.

4.3 Non-Critical Input Edge

When a non-critical edge occurs, the CAMe primitive increments ***NCrit_Ecnt***, saves the captured edge time in ***NCrit_ET*** , saves a snapshot of crank count from EPPE, saves a snapshot of the TCR2 value and generates an interrupt to the host if enabled. ***Noise_Cnt*** is incremented whenever the non-critical edge is out of sequence (two in a row). If ***Rep_Out_En*** = 1, a non-critical output edge for the replicated cam output signal will be set up on the channel indicated by ***CAMe_Rep_Out_Chan_Num***. Also, if ***CAMe_Min_Per*** is non-zero, only one non-critical interrupt will be issued for each critical edge (the first one after a non-critical edge). This helps to limit the rate of interrupts if a fault condition results in a high-frequency signal applied to the cam input.

4.4 Replicated Cam Output

If enabled by the host (***Rep_Out_En*** = 1), the CAMe primitive will generate a replicated cam output signal on an eTPU channel indicated by ***CAMe_Rep_Out_Chan_Num***. The signal will lag the cam input signal by ***CAMe_Rep_Out_Lag_Time***. Its polarity is determined by ***Rep_Out_Inv***.

4.5 Angular Engine Position

Variable cam phase control software must know the engine position at the time of the critical cam edge (cam phase). Generally, the cam phase is determined by interpolating between the two crank edges that bound the cam edge. If a crank edge has not yet occurred after the cam edge, the cam phase can be determined by extrapolating from the two crank edges preceding the cam edge, but it will be less accurate due to accelerations or decelerations. Either calculation requires period and edge time information from the crank signal primitive that is synchronous with the cam edge.

The CAMe primitive will not calculate cam phase angle, as the TPU CAM4x primitive did, but will save the TCR2 value at the critical edge (and non-critical edge). This represents an extrapolated cam phase if TCR2 is set up as an angle clock. If an interpolation is desired, the CPU can use ***Crit_Array***, which contains the last 32 crank edge times, from the EPPE primitive or keep its own array of crank edge times. The critical edge count in the upper word of ***Crit_TCR2*** can be used to index into the array, but remember that the TCR2 value is a snapshot and coherency to the engine data is not guaranteed. Comparing the crank event times to ***Crit_ET*** will validate the data used, before calculating cam phase.

If TCR2 is not set up to reflect the critical edge count, then ***Crank_Count*** can be used to index the crank data array. Remember that this variable can be off by one count in the future or past when the cam edge is very near the crank edge due to the eTPU scheduler, and the crank event data should be validated as described above. Reference the TPU CAM4x MCD-2470 for more detail on these calculations and the special the cases that must be considered.

4.6 CPU Interrupts

The Host can enable interrupts for critical edges and non-critical edges with the appropriate flags defined in the upper byte of a parameter ram. The Host must be careful, however, not to overwrite the variable, ***NCrit_ET***, in the lower 24 bits of the same parameter.

The CAMe primitive issues a channel interrupt at the appropriate edge if enabled. The Host must determine if the interrupt was sourced by a critical or non-critical edge when both are enabled and may use ***Last_Edge_Crit*** flag. The interrupt overflow status bit for the cam channel alerts the Host of a missed interrupt.

When ***CAMe_Min_Per*** is any non-zero value, only one non-critical edge interrupt is issued for each critical edge processed. In other words, if a critical edge is ignored due to the min per requirement, then the non-critical edge that follows will not issue a host interrupt.

4.7 Shutdown

The CAMe function can be disabled by:

1. Issue a Shutdown Host Service Request (%111).
2. Wait for the HSR bits to clear.
3. Set the Priority level to disabled (%00).

Note that, if ***Rep_Out_En*** = 1, a Shutdown HSR will also disable the replicated cam output channel and drive the output low (regardless of the state of the cam input or ***Rep_Out_Inv***).

4.8 Switching Functions

To switch from one eTPU function to another, the CPU should shut down the original function and then follow directions for initializing the new function. Remember to allocate enough parameter RAM in initialization for the worst case intended function.

5. Primitive Timing

Table 2 shows the worst-case execution times for the various phases of the CAMe primitive (see the Application Implementation document for information on converting µcycles into real time). Note that these values do not take into account latencies due to other primitives in the application, priorities, etc.

CAMe Primitive Phase	Worst-Case µcycles *	RAM Accesses
Crit Edge (TransA)		
- ***Rep_Out_En* = 0**	**40**	**22**
- ***Rep_Out_En* = 1**	**41**	**23**
NCrit Edge (TransA) – not noise		
- ***Rep_Out_En* = 0**	**29**	**11**
- ***Rep_Out_En* = 1**	**30**	**12**
Time-out (MatchA)	**7**	**3**
Init Crit Rise/Fall HSR		
- ***Rep_Out_En* = 0**	**26**	**6**
- ***Rep_Out_En* = 1**	**50**	**7**
Shutdown HSR		
- ***Rep_Out_En* = 0**	**11**	**1**
- ***Rep_Out_En* = 1**	**19**	**2**

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

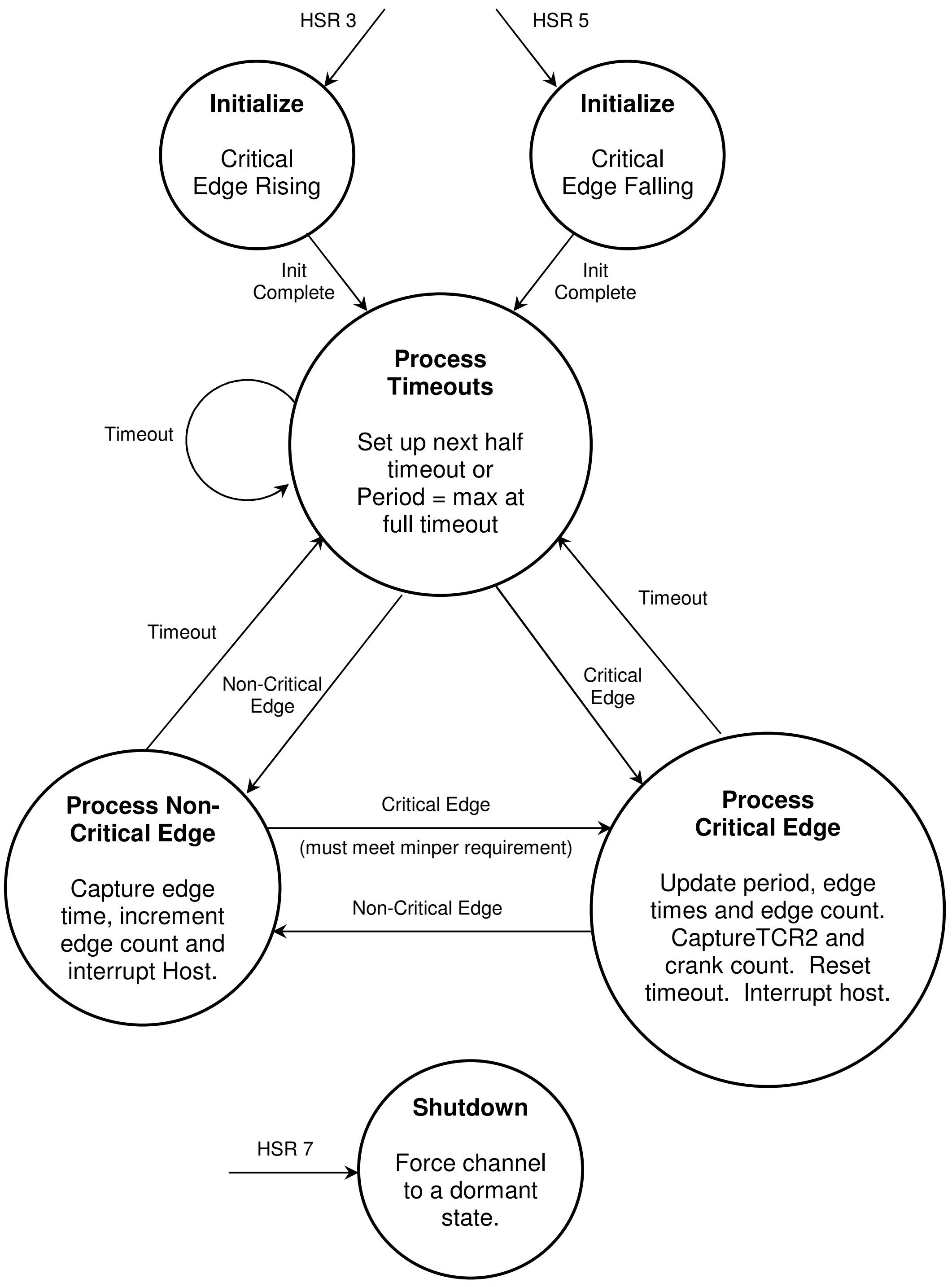
Code size = 150 microcode instructions (32-bit) = 600 bytes

RAM size = 14 parameters (32-bit) = 56 bytes

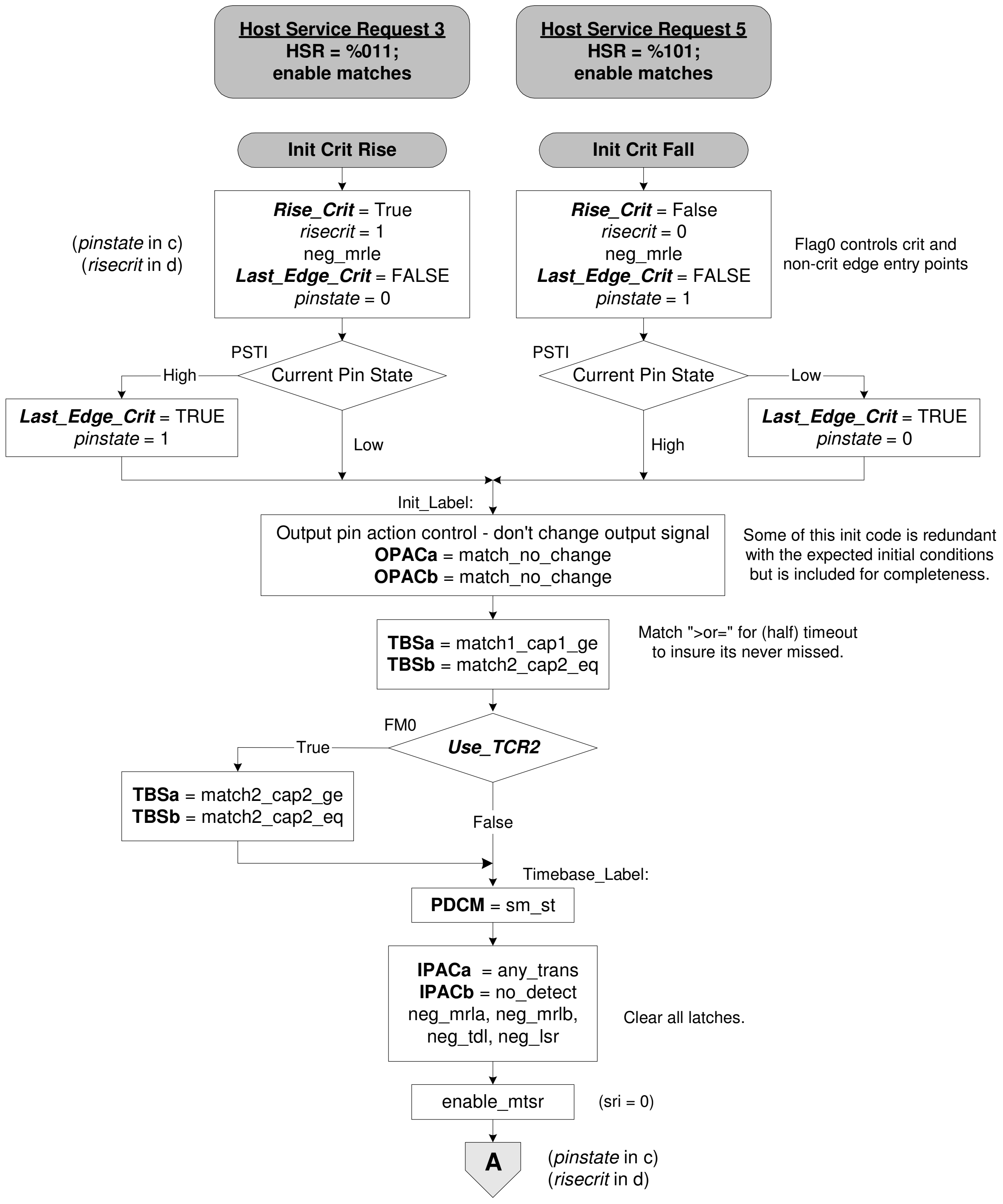
Table 2 – Worst-Case Execution Times

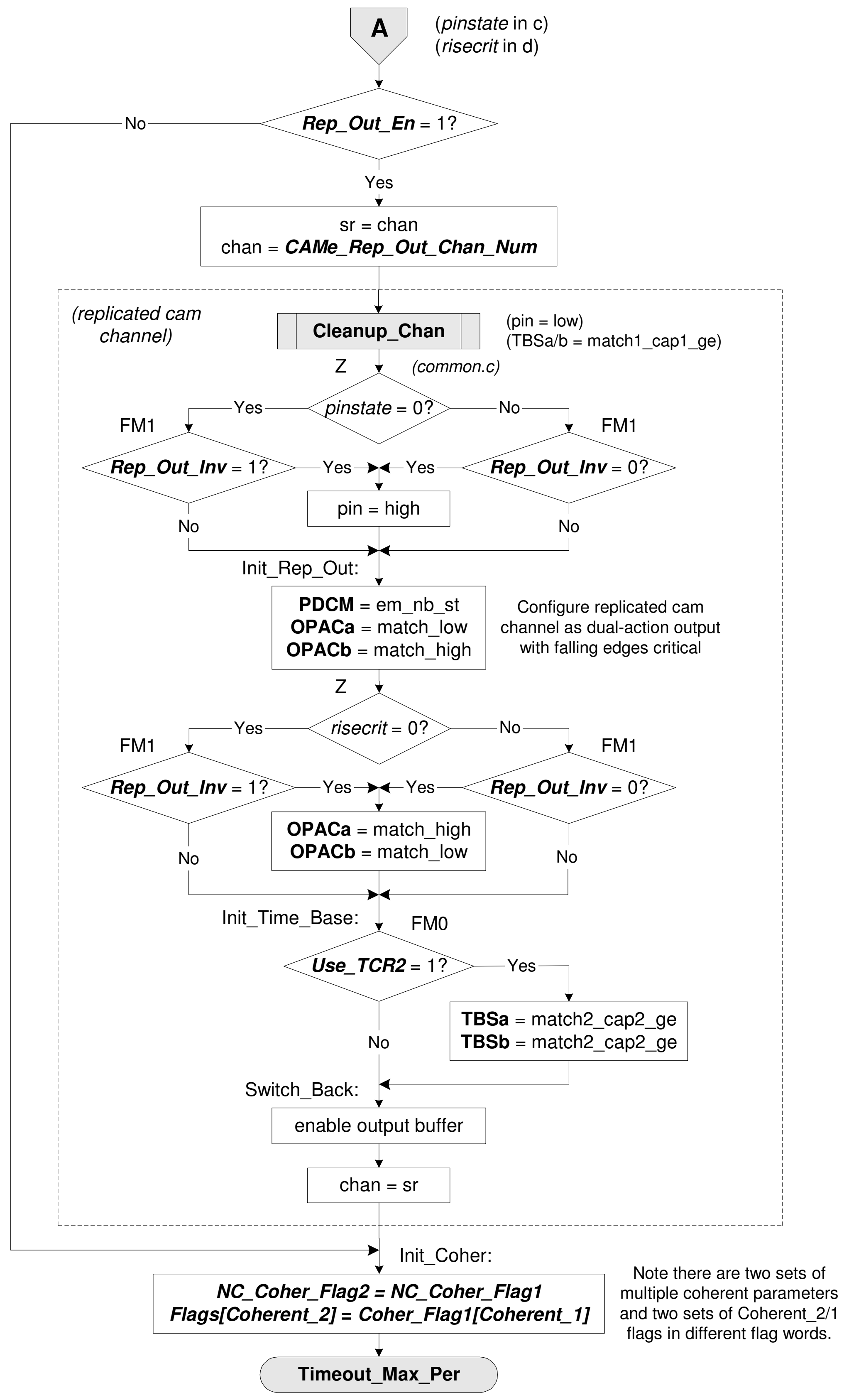
6. State Diagram

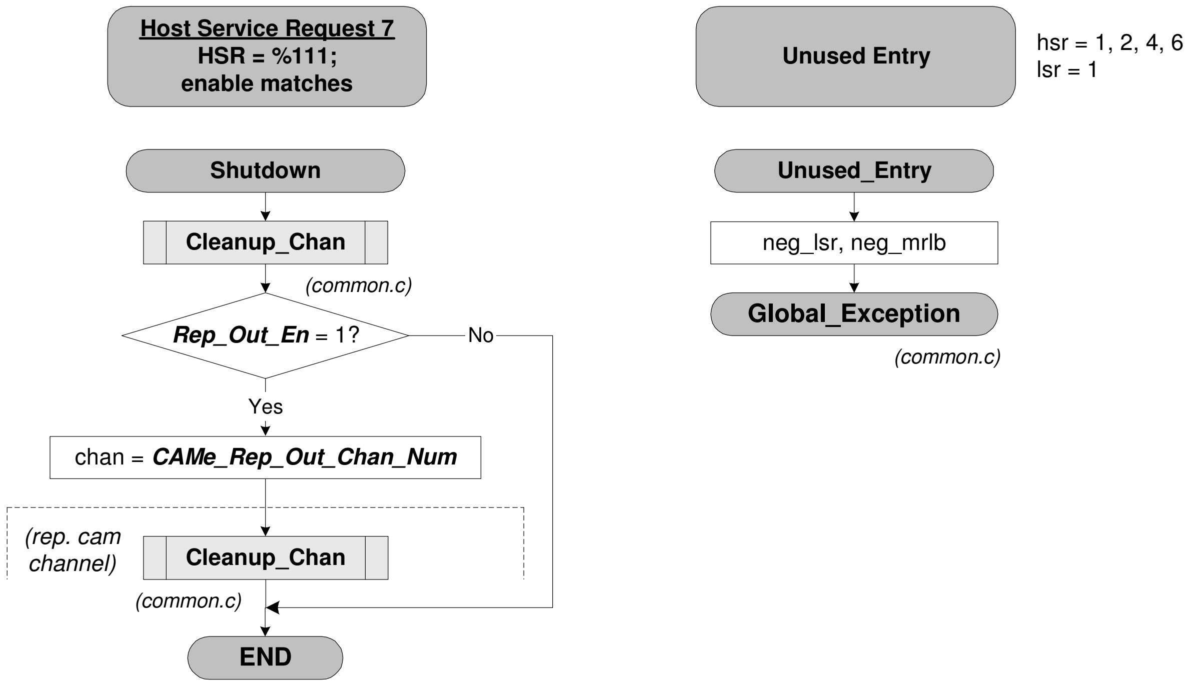
CAMe Primitive


HSR 3
HSR 5
Initialize
Critical Edge Rising
Initialize
Critical Edge Falling
Init Complete
Init Complete
Process Timeouts
Set up next half timeout or Period = max at full timeout
Timeout
Timeout
Non-Critical Edge
Critical Edge
Timeout
Process Non-Critical Edge
Capture edge time, increment edge count and interrupt Host.
Critical Edge
(must meet minper requirement)
Non-Critical Edge
Process Critical Edge
Update period, edge times and edge count. CaptureTCR2 and crank count. Reset timeout. Interrupt host.
Shutdown
HSR 7
Force channel to a dormant state.

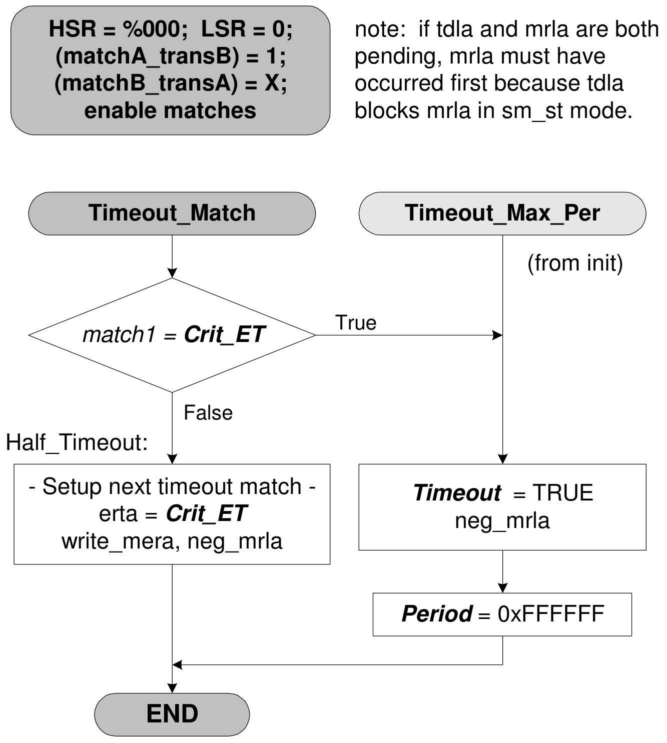
7. Flowcharts



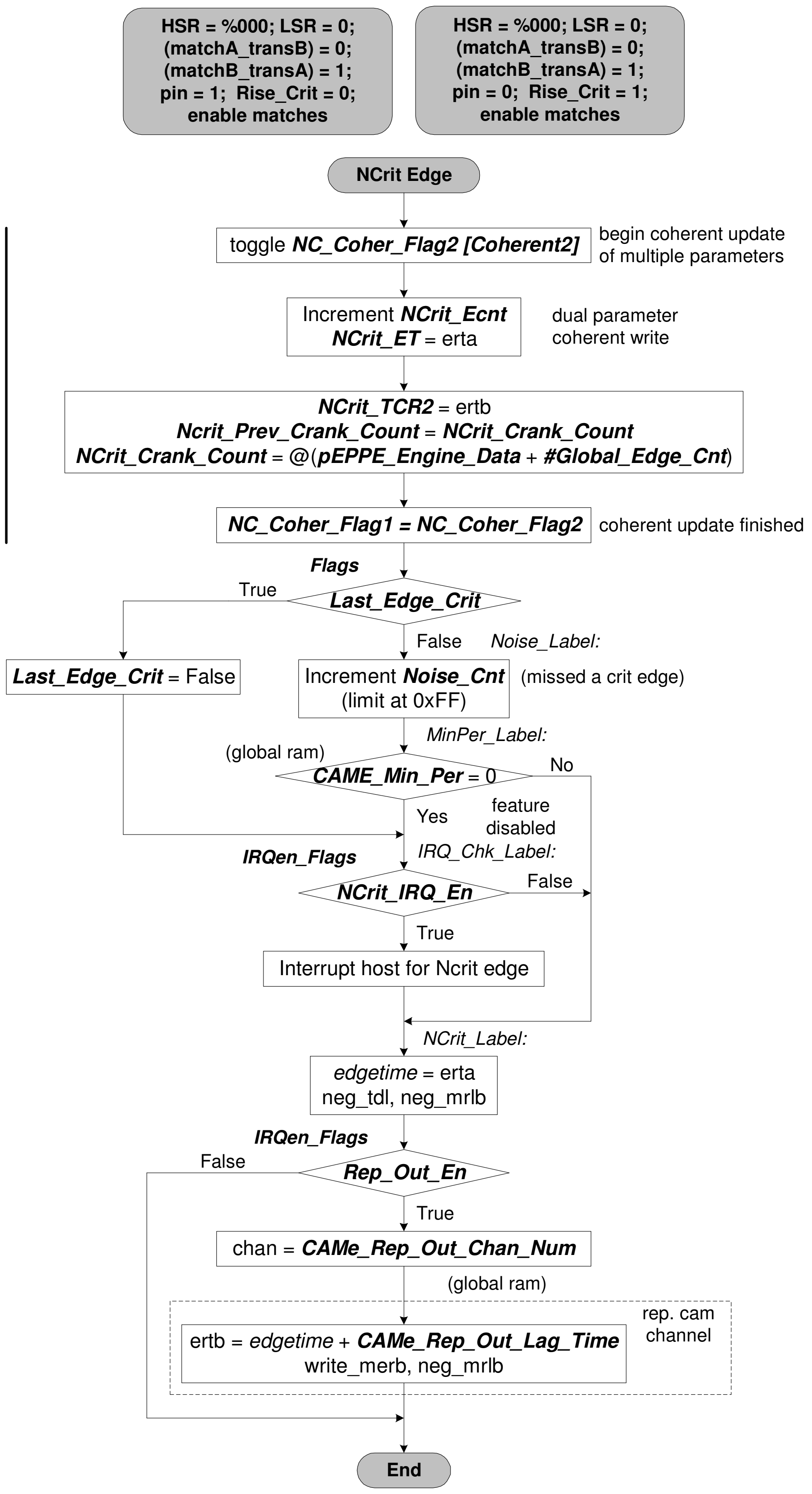

A
(*pinstate* in c)
(*risecrit* in d)
Rep_Out_En = 1?
No
Yes
sr = chan
chan = ***CAMe_Rep_Out_Chan_Num***
(replicated cam channel)
Cleanup_Chan
(pin = low)
(TBSa/b = match1_cap1_ge)
Z
(common.c)
pinstate = 0?
Yes
No
FM1
FM1
Rep_Out_Inv = 1?
Yes
Yes
Rep_Out_Inv = 0?
pin = high
No
No
Init_Rep_Out:
PDCM = em_nb_st
OPACa = match_low
OPACb = match_high
Configure replicated cam channel as dual-action output with falling edges critical
Z
risecrit = 0?
Yes
No
FM1
FM1
Rep_Out_Inv = 1?
Yes
Yes
Rep_Out_Inv = 0?
OPACa = match_high
OPACb = match_low
No
No
Init_Time_Base:
FM0
Use_TCR2 = 1?
Yes
No
TBSa = match2_cap2_ge
TBSb = match2_cap2_ge
Switch_Back:
enable output buffer
chan = sr
Init_Coher:
NC_Coher_Flag2 = NC_Coher_Flag1
Flags[Coherent_2] = Coher_Flag1[Coherent_1]
Note there are two sets of multiple coherent parameters and two sets of Coherent_2/1 flags in different flag words.
Timeout_Max_Per



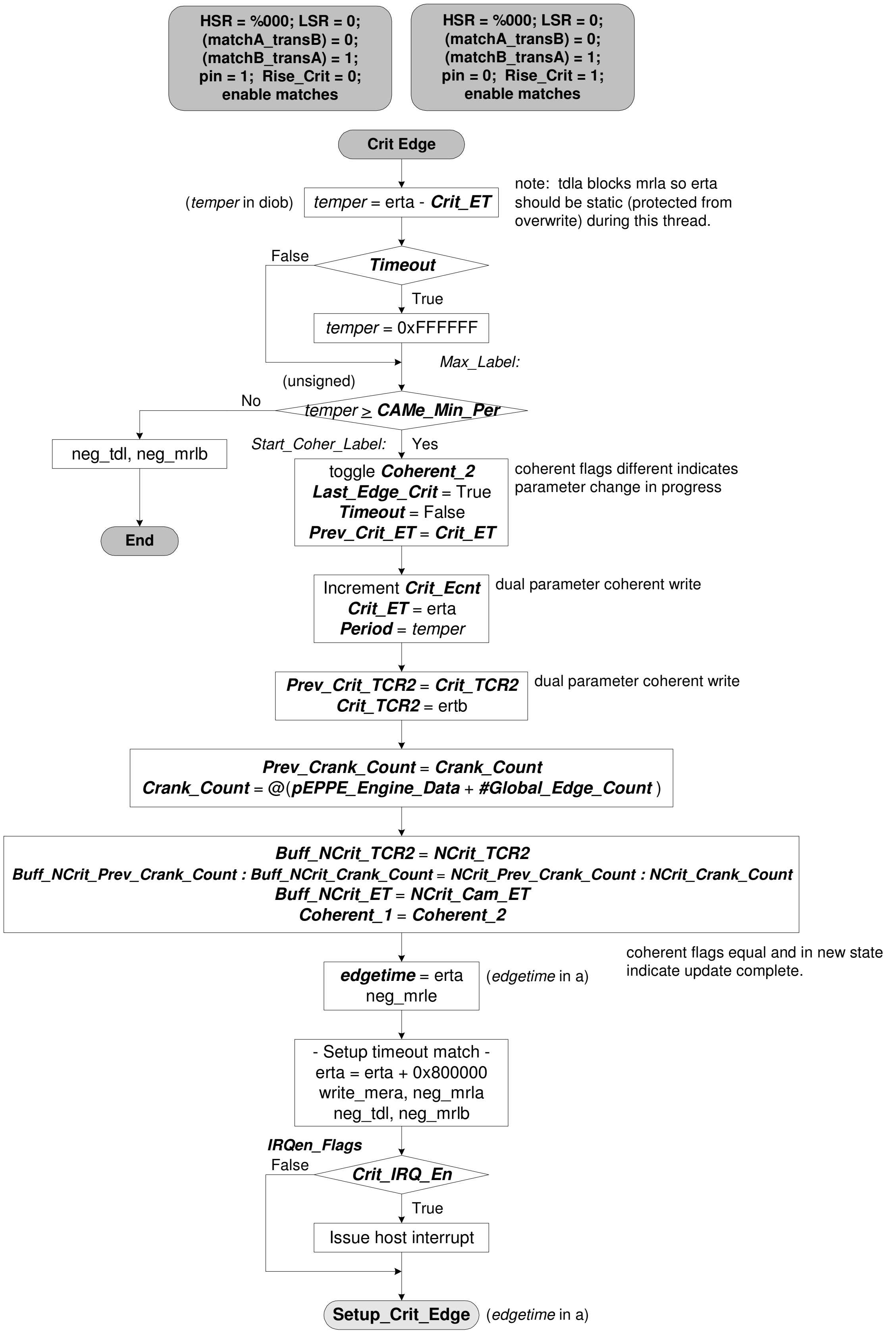
Host Service Request 7
HSR = %111;
enable matches
Shutdown
Cleanup_Chan
(common.c)
Rep_Out_En = 1?
No
Yes
chan = CAMe_Rep_Out_Chan_Num
(rep. cam channel)
Cleanup_Chan
(common.c)
END
Unused Entry
hsr = 1, 2, 4, 6
lsr = 1
Unused_Entry
neg_lsr, neg_mrlb
Global_Exception
(common.c)



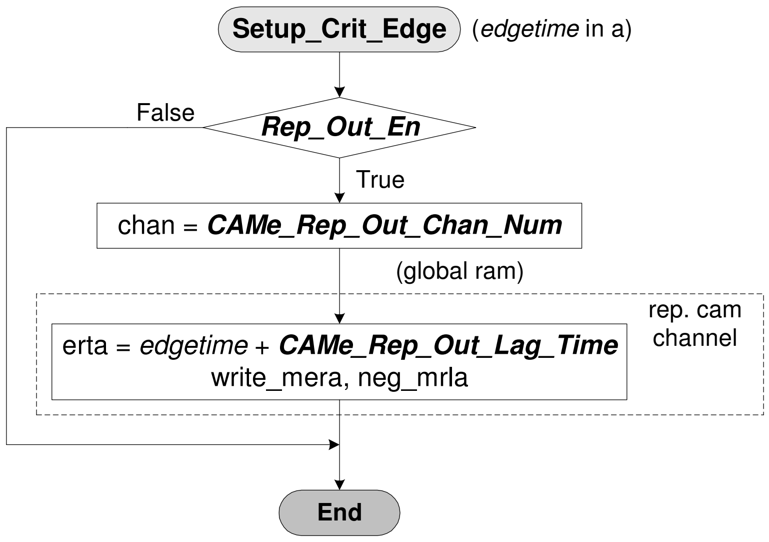
HSR = %000; LSR = 0;
(matchA_transB) = 1;
(matchB_transA) = X;
enable matches
note: if tdla and mrla are both pending, mrla must have occurred first because tdla blocks mrla in sm_st mode.
Timeout_Match
Timeout_Max_Per
(from init)
match1 = Crit_ET
True
False
Half_Timeout:
- Setup next timeout match -
erta = Crit_ET
write_mera, neg_mrla
Timeout = TRUE
neg_mrla
Period = 0xFFFFFF
END



HSR = %000; LSR = 0; (matchA_transB) = 0; (matchB_transA) = 1; pin = 1; Rise_Crit = 0; enable matches
HSR = %000; LSR = 0; (matchA_transB) = 0; (matchB_transA) = 1; pin = 0; Rise_Crit = 1; enable matches
NCrit Edge
toggle NC_Coher_Flag2 [Coherent2]
begin coherent update of multiple parameters
Increment NCrit_Ecnt
NCrit_ET = erta
dual parameter coherent write
NCrit_TCR2 = ertb
Ncrit_Prev_Crank_Count = NCrit_Crank_Count
NCrit_Crank_Count = @(pEPPE_Engine_Data + #Global_Edge_Cnt)
NC_Coher_Flag1 = NC_Coher_Flag2
coherent update finished
Flags
Last_Edge_Crit
True
False
Noise_Label:
Last_Edge_Crit = False
Increment Noise_Cnt (limit at 0xFF)
(missed a crit edge)
MinPer_Label:
(global ram)
CAME_Min_Per = 0
No
Yes
feature disabled
IRQ_Chk_Label:
IRQen_Flags
NCrit_IRQ_En
False
True
Interrupt host for Ncrit edge
NCrit_Label:
edgetime = erta
neg_tdl, neg_mrlb
IRQen_Flags
Rep_Out_En
False
True
chan = CAMe_Rep_Out_Chan_Num
(global ram)
rep. cam channel
ertb = edgetime + CAMe_Rep_Out_Lag_Time
write_merb, neg_mrlb
End


HSR = %000; LSR = 0; (matchA_transB) = 0; (matchB_transA) = 1; pin = 1; Rise_Crit = 0; enable matches
HSR = %000; LSR = 0; (matchA_transB) = 0; (matchB_transA) = 1; pin = 0; Rise_Crit = 1; enable matches
Crit Edge
(temper in diob)
temper = erta - Crit_ET
note: tdla blocks mrla so erta should be static (protected from overwrite) during this thread.
False
Timeout
True
temper = 0xFFFFFF
Max_Label:
(unsigned)
No
temper ≥ CAMe_Min_Per
neg_tdl, neg_mrlb
End
Start_Coher_Label:
Yes
toggle Coherent_2
Last_Edge_Crit = True
Timeout = False
Prev_Crit_ET = Crit_ET
coherent flags different indicates parameter change in progress
Increment Crit_Ecnt
Crit_ET = erta
Period = temper
dual parameter coherent write
Prev_Crit_TCR2 = Crit_TCR2
Crit_TCR2 = ertb
dual parameter coherent write
Prev_Crank_Count = Crank_Count
Crank_Count = @(pEPPE_Engine_Data + #Global_Edge_Count)
Buff_NCrit_TCR2 = NCrit_TCR2
Buff_NCrit_Prev_Crank_Count : Buff_NCrit_Crank_Count = NCrit_Prev_Crank_Count : NCrit_Crank_Count
Buff_NCrit_ET = NCrit_Cam_ET
Coherent_1 = Coherent_2
coherent flags equal and in new state indicate update complete.
edgetime = erta
neg_mrle
(edgetime in a)
- Setup timeout match -
erta = erta + 0x800000
write_mera, neg_mrla
neg_tdl, neg_mrlb
IRQen_Flags
False
Crit_IRQ_En
True
Issue host interrupt
Setup_Crit_Edge
(edgetime in a)

DELPHI CONFIDENTIAL


Setup_Crit_Edge
(edgetime in a)
False
Rep_Out_En
True
chan = CAMe_Rep_Out_Chan_Num
(global ram)
rep. cam channel
erta = edgetime + CAMe_Rep_Out_Lag_Time
write_mera, neg_mrla
End

8. Revision Log

8.1 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 10/16/92):

Rev **x**.**y**

x	identifies the microcode, where:
0	represents the original release of microcode
1	represents the first release of changed microcode
2	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 etc.

8.2 Revision History

Revision	Date	Record	Author
0.0			Mary Hedges
0.0	11/9/04	Remove back-up crank functionality. Re-structure entry points and parameter order.	Doug Tackitt
1.0	12/9/04 12/22/04	Make several small changes from design review. Add two flags that indicate data coherency. Make changes from code review. Change all names and labels to start with CAMe (ouch). Move ipac init after timebase init for obvious reasons. Change Timeout from eq compare to ge compare and process half-timeout.	DJT
2.0 (SCR 4083)	04/20/05	Made changes to work with compiler build 147 or later.	Warren Donley
3.0 (SCR 4113)	05/01/05	Add minper feature to limit the host interrupt rate during a signal fault. ***CAMe_Min_Per*** is in global ram. Add ***Noise_Cnt***. Removed CAMe from the beginning of all variable names in the MCD.	Doug Tackitt
4.0	05/17/05	SCR 4142: Fix a comment error in the .h file.	Doug Tackitt
5.0	09/26/05	SCR 4360: Added replicated CAMe output and	Warren Donley

		non-critical edge counter. Expanded both edge counters to 16 bits.	
5.1	02/24/06	SCR 4606: Added parentheses to certain #define statements.	Warren Donley
5.2	06/07/07	Corrected header block in Host Interface.	Warren Donley
6.0	11/19/09	Add crank cnt snapshot to ncrit edge.	Doug Tackitt
6.1	11/30/09	Fix mistake on PRAM map page- IRQ_En_Flags.	Doug Tackitt
6.2	5/20/10	Fix name typos on Last_Edge_Crit.	Doug Tackitt