Freescale Semiconductor
Application Note

Document Number: AN3769
Rev. 0, 06/2009

Using the Engine Position (CRANK and CAM) eTPU Functions

by: Geoff Emerson
East Kilbride

1 Introduction

This application note describes simple C interface functions to CRANK and CAM eTPU functions. These functions can be used on any product that has an eTPU module. Example code is available for the MPC5633M device. This application note should be read in conjunction with application note *AN2864 — General C Functions for the eTPU*. The CAM and CRANK functions are part of an automotive function set. For more details refer to *AN3769 — eTPU Automotive Set (Set 2)* .

2 Function Overview

These functions process signals from crank and cam sensors on an engine to provide angular position and speed information to a control system. The engine position functions (CRANK and CAM) together generate an angle counter. The angle counter is synchronized with the crank wheel angle and provides a counter representation of the instantaneous position of the engine. The crank toothed wheel signal is processed by a combination of eTPU hardware and eTPU software to create an angle counter. The cam signal is used to differentiate between the two 360° halves of the 720° engine cycle. The hardware and software support the following features:

- Support for wide range tooth wheel configurations
  - Up to 255 teeth on the wheel are supported
  - Between 1 and 3 missing teeth are supported
- Support for rising and falling active edges
- Unique position information over a multiple of 720° revolutions is generated
- Angle counts are added in the gap (to compensate for missing teeth)

Contents

- Synchronization is achieved by identifying the gap and confirming normal teeth spacing follows the gap.
- On– going gap testing is performed to ensure synchronization is maintained. If the gap testing fails a re–synchronization is initiated.
- Status reporting for:
  - CRANK
  - CAM
  - Overall engine position system

- Crank tooth periods are divided by between 1 and 1024 to create an angle counter. This happens on every crank tooth edge occurrence.
- Noise rejection for the crank signal is provided by using a ratio and time based windowing technique. Window position is based on most recent tooth measurements. Window width is based on a ratio of most recent tooth measurements. To provide versatile windowing ratios they may be separately specified for:
  - Normal teeth (all other teeth not dealt with below)
  - The tooth after the gap
  - The second tooth after the gap
  - The tooth after a single timeout has occurred. This allows the window to be wider if a timeout happens.

- Noise rejection for cam is provided by using an angle based windowing technique.
- Single tooth crank errors are dealt with as follows:
  - A single tooth occurring outside the window is allowed and reported by the CRANK function raising an interrupt request flag.
  - If the tooth edge is not in the window the CRANK function adds counts to compensate as if the tooth edge had occurred in the window.
  - A new window is scheduled for the next tooth based on the most recent physical tooth information and a special windowing ratio.
  - This means, the tooth occurring after a single tooth timeout event can have a wider acceptance window if necessary.

- Two consecutive teeth occurring outside of their respective windows are considered an error. In this case the CRANK function initiates a re-synchronization.
- When a stall or another error is detected a list of channels may be signalled. This allows for a rapid shutdown of other functions in the event that the engine position becomes unknown.
- When a stall is detected, the software initiates a re-synchronization.
- CAM and CRANK are separate functions. Either CRANK or CAM may be replaced by a custom function as required.
- The CAM function supports a single lobe CAM.
- The following features are associated with the CRANK function start-up behaviour
  - A programmable number of teeth may be ignored prior to the CRANK function beginning to attempt synchronization.
  - A programmable amount of time may be programmed during which the CRANK function waits before attempting to synchronize
  - A timeout for the first tooth is supported. If a second tooth edge is not recognized before this time the software reverts to seeking a first edge again.

- The created angle counter does not reset at 720°. It increments from 0 to 0xFFFFFF and numerous engine cycles would have occurred before it rolls over to 0.

3 Functional Description

The CRANK function processes the signal from a tooth wheel connected to the crank shaft. The tooth wheel has one or more missing teeth, these are referred to as the gap. The gap allows positional information for the engine to be derived. This is becasue the gap has a fixed angular relationship to the engine's position. A typical tooth wheel might have 35 teeth spaced at 10° intervals with one missing tooth in the 36th position. This is referred to a 36 –1 configuration. A 60 – 2 configuration is also common. This wheel has 58 physical teeth spaced at 6° intervals with missing teeth in the 59th and 60th positions. Prior to beginning the search for the gap, the CRANK function ignores teeth edges for a specified period of time, BLANK_TIME. Then optionally a

number of teeth, BLANK_TEETH, may be ignored. The next edge after this initiates a timeout period, FIRST_TOOTH_TIMEOUT, during which a second edge must occur. If a second edge does not occur within the specified timeout period the function begins searching for the first edge again. The figure below shows these parameters graphically.

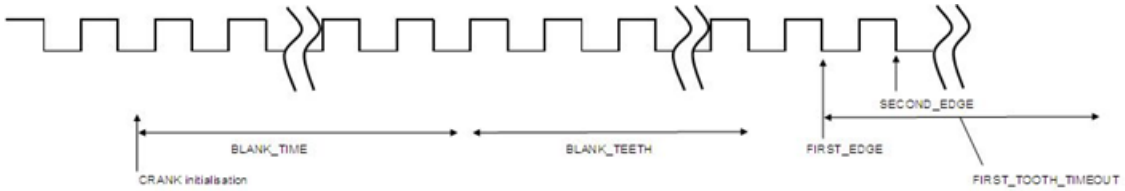


Figure 1. First Tooth Timeout

The CRANK function is responsible for identifying the gap in the crank wheel by measuring and analyzing the time between successive teeth. An ABa test is used to determine if the gap has occurred. The figure below is a graphical representation of the signal coming from a tooth wheel.

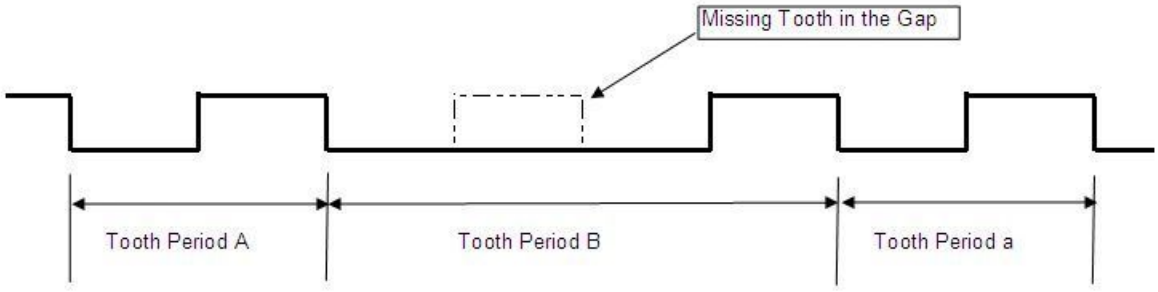


Figure 2. ABa Gap Test

A user specified Gap_Ratio parameter is used by the function to determine if the gap has been identified. The AB portion of the test is considered to have passed if:

*Tooth Period B * Gap_Ratio > Tooth Period A*

If the AB test passes then the Ba test is performed. The Ba test passes if:

*Tooth Period B * Gap_Ratio > Tooth Period a*

The timer counter register 2 (TCR2) contains the angle counter as derived by the eTPU Angle Counter hardware with assistance from the CRANK function. Prior to the gap being verified the TCR2 value is held by the CRANK function at a value less than one tooth worth of TCR2 counts. After the gap has been found and verified by the ABa test the CRANK function writes a starting value to the TCR2 register. The TCR2 count is incremented by a user defined number of counts for every Crank tooth which is detected. The rate at which the TCR2 counter is incremented is related to the previous tooth period as measured by the CRANK function. The aim of the angle counter hardware and CRANK function software is to provide a TCR2 that tracks the angular position of the crank shaft.

The CRANK function keeps track of the teeth and estimates when the next gap will occur. When the next and subsequent gaps occur, the CRANK function ensures that it has occurred in the correct place by performing an ABa test again. During the gap TCR2 counts are inserted even though there are no tooth edges.

Failure of the ABa test will result in the CRANK function signalling the CAM function and optionally other output channels (for example, FUEL, SPARK, and KNOCK_WINDOW) that there has been an error and atttempts to re-synchronize.

For a full explanation of the angle counter refer to *AN2897 — Using the eTPU Angle Clock*, available at www.freescale.com

3.1 Start-Up Scenarios

To achieve synchronization CAM and CRANK functions must be able to deal with various start-up scenarios. After initialization, the active CAM edge may happen either before or after the gap is identified. The following sections show the behavior of the functions under these conditions. Three status variables are used by the functions to control the behavior and to also communicate

with the host CPU. These are explained in detail in the Section 3.5 Status Variables. In the following figures, each of the three status variables are shown. In the following figures the Blank_Time period and any Blank_Teeth are not shown. Individual crank teeth signals are not shown. The vertical line against the crank signal indicates the gap. The active cam edge position is indicated by an arrow pointing up from a box bearing the word CAM.

3.1.1 Scenario 1 Cam Edge Occurs First

In this scenario, the cam edge occurs before the gap has been identified.

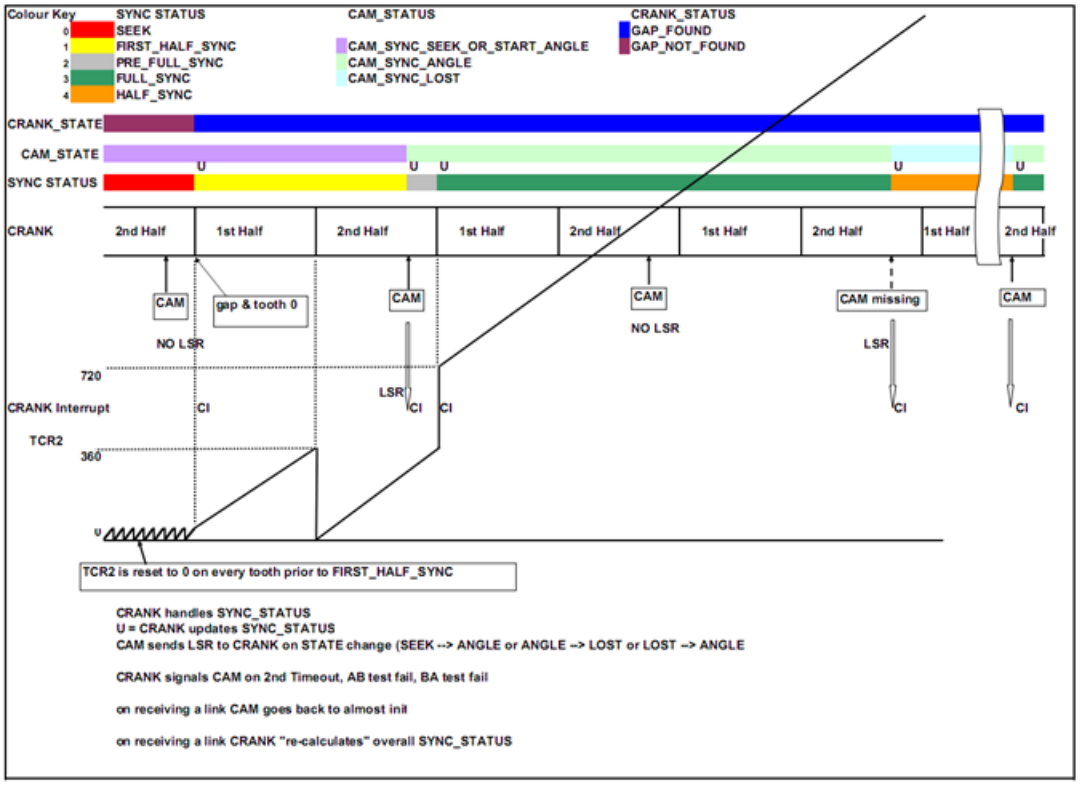


Figure 3. Cam Edge Before the Gap

The first cam edge is effectively ignored by the CAM function. This is because the CRANK function has not yet identified the gap. When the gap is identified by the CRANK function it begins incrementing TCR2 beyond a single tooth's count. Prior to this point TCR2 was reset to zero on every active crank tooth edge by the CRANK function. TCR2 is reset to zero by the CRANK function when the second gap occurs because no valid cam edge has yet been identified. When the third gap occurs, a valid cam edge has been identified and TCR2 is set to the number of TCR2 counts representing 720° by the CRANK function. From this point the TCR2 value is incremented by the specified number of counts per tooth on every valid tooth edge. This approach ensures that the cam signal occurs in the second half of each engine revolution. It also means that the TCR2 values do not exceed the 720° TCR2 value until both the cam position and crank position are known. Therefore, if output events for example FUEL, SPARK, and KNOCK_WINDOW are scheduled at initialization for angles greater than 720° the output events can not occur until the engine position is known. This technique is shown for the KNOCK_WINDOW function in the software package associated with this application note, *AN3769SW*.

3.1.2 Scenario 2 CRANK Gap Occurs First (before the CAM edge)

In this case the gap is identified before the cam edge occurs.

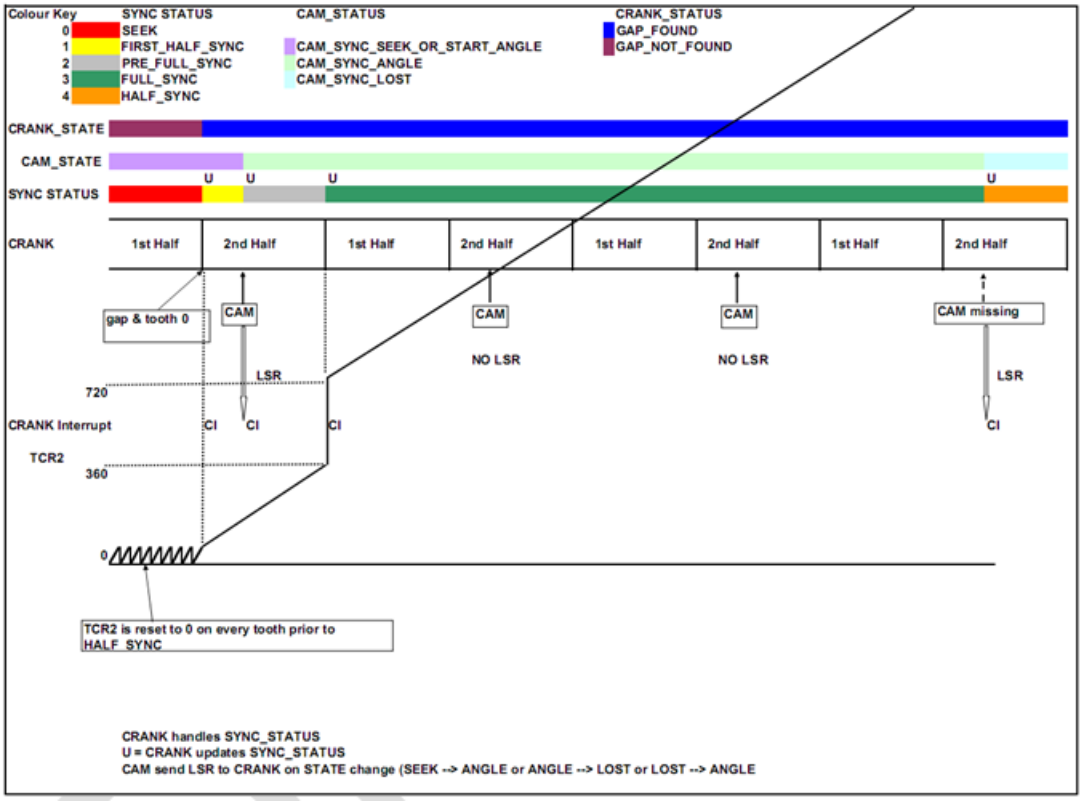


Figure 4. Cam Edge After the Gap

When the gap is identified by the CRANK function it begins incrementing TCR2 beyond a single tooth's count. Prior to this point TCR2 was reset to zero on every active crank tooth edge by the CRANK function. When the second gap occurs, a valid cam edge has been identified and TCR2 is set to the number of TCR2 counts representing 720° by the CRANK function. From this point the TCR2 value is incremented by the specified number of counts per tooth on every valid tooth edge. This approach ensures that the CAM signal occurs in the second half of each engine revolution. It also means that the TCR2 values do not exceed the 720° TCR2 value until both the CAM position and CRANK position are known. Therefore, if output events, for example, FUEL, SPARK, and KNOCK_WINDOW are scheduled at initialization for angles greater than 720°. The output events cannot occur until the engine position is known. This technique is shown for the KNOCK_WINDOW function in the software package associated with this application note *AN3769SW*.

3.2 Windowing Signals

The eTPU channel hardware includes various channel modes to help reject noise on signals. One of these is used in the example function set. Various window parameters must be specified in calls to the APIs so that the functions work efficiently. These parameters are described in the Section 4. C-Level API for eTPU Engine Position Functions CRANK and CAM.

The mechanism for the windowing is described here.

The fundamental components of the windowing system for the crank signal are shown in the following figure.

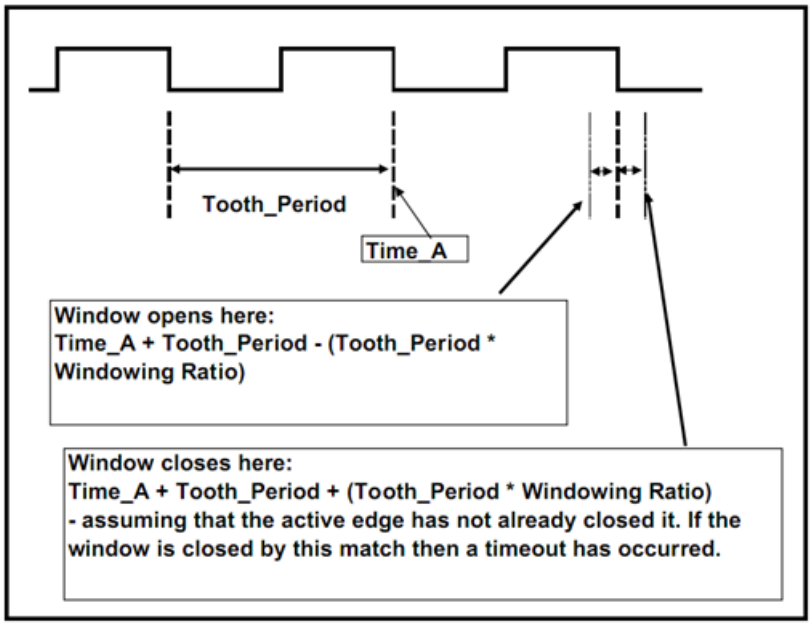


Figure 5. Mecanism for Windowing Out Noise

The eTPU channel hardware is programmed with values in its match registers that define when the window opens and closes. Transitions before the window opens are rejected. A transition in the acceptance window prevents further transitions from being accepted until the hardware is re-programmed. When the window closes without a transition in the window, further transitions are not recognized until the hardware is re-programmed. This situation is referred to as a timeout. The CRANK function can distinguish between a timeout and a valid transition in the acceptance window. Timeouts are reported by setting a TIMEOUT bit in an error status word. The cam signal is windowed using the same channel mode and a similar technique. However, the cam window is based in the angle domain, whereas the crank signal is windowed in the time domain.

3.3 Error Handling

A limited amount of error handling is supported by CAM and CRANK functions. To make the function set easily understood error handling is treated in a fairly simple manner. Other more complex error handling solutions are possible. It is left to the user to determine if the measures described are adequate for their purposes and meet their system requirements.

The following errors are dealt with:

3.3.1 CRANK Timeout Prior to Synchronization

Situation — The crank signal did not have a valid edge in the acceptance window prior to the gap having been identified.

Treatment — In this case the internal state machine of the CRANK function reverts to seeking the first edge again. The timeout bit is set in the crank error status word.

3.3.2 AB Test Failure After Synchronization is Achieved

This can be caused by one of the following scenarios:

3.3.2.1 Timeout on CRANK Tooth Prior the Gap

Situation — A timeout occurred on the tooth before the gap.

Treatment — This is considered a serious error as the position of the engine is no longer certain. The AB portion of the ABa test cannot be performed as the A measurement is not available. The TIMEOUT and STALL bits are set in the CRANK error status word. The internal state machine of the CRANK function reverts to seeking for the first edge again. The CRANK channel issues a channel interrupt. Optionally, links are sent to a specified list of channels. This allows output channels to shutdown quickly in the event that the engine position becomes unknown.

3.3.2.2 Timeout on the CRANK Tooth After the Gap

Situation — A timeout occurred on the tooth after the gap.

Treatment — This is considered a serious error as the position of the engine is no longer certain. The AB portion of the ABa test cannot be performed as the B measurement is not available. The STALL bit is set in the CRANK error status word. The internal state machine of the CRANK function reverts to seeking for the first edge again. The CRANK channel issues a channel interrupt. Optionally, links are sent to a specified list of channels. This allows output channels to shutdown quickly in the event that the engine position becomes unknown.

3.3.2.3 CRANK Gap Ratio Test Fail

Situation — The gap ratio test fails on the tooth after the gap.

Treatment — This is considered a serious error as the position of the engine is no longer certain. The STALL bit is set in the CRANK error status word. The internal state machine of the CRANK function reverts to seeking for the first edge again. The CRANK channel issues a channel interrupt. Optionally, links are sent to a specified list of channels. This allows output channels to shutdown quickly in the event that the engine position becomes unknown.

3.3.3 Single CRANK Tooth Timeout After Synchronization is Achieved

Situation — A valid edge was not detected in the acceptance window.

Treatment — An internal flag within the CRANK function (Timeout_flag) is used to register that a single tooth timed out. The CRANK channel issues a channel interrupt. The next tooth is windowed with a different windowing ratio (called crank_windowing_ratio_timeout). This approach allows the user to specify a different (probably wider) window for the next tooth after a single tooth timeout. If the next window has an active edge within it the internal CRANK flag (Timeout_flag) is cleared.

3.3.4 Two Consecutive CRANK Tooth Timeouts

Situation — A valid edge was not detected in the acceptance window and the internal flag within the CRANK function (Timeout_flag) is already set.

Treatment — This is considered a serious error as the position of the engine is no longer certain. The STALL bit is set in the CRANK error status word. The internal state machine of the CRANK function reverts to seeking for the first edge again. The CRANK channel issues a channel interrupt. Optionally, links are sent to a specified list of channels. This allows output channels to shutdown quickly in the event that the engine position becomes unknown.

3.3.5 Active CAM Edge Not in Angle Window

Situation — The cam edge is not in the angle acceptance window.

Treatment — Assuming that prior to the cam edge becoming lost its position was known (it was found in the angle window) the CAM status changes from CAM_SYNC_ANGLE to CAM_SYNC_LOST. CAM sends a link service request to CRANK. A link service request is the means by which one channel can request service from another channel. When this is serviced CRANK updates the overall engine position status. Assuming this had previously been FULL_SYNC the overall engine position status would be changed to HALF_SYNC. The CRANK function issues a channel interrupt when the overall engine position status changes. In this situation the application software must decide what to do. One approach is to to widen the angle window in which the active cam edge is being searched.

3.4 Software Limitations

This section describes the limitations of the CAM and CRANK functions.

Single lobe CAM — Only a single lobe cam is supported by the CAM function. Where the majority of crank wheels are one of two configuration (36 – 1 or 60 – 2), cam wheels have numerous configurations. The example CAM function was written for the simplest case which is a single lobe. It is anticipated that some users will substitute the CAM function with one that is better suited to their own cam configuration.

Single gap on the crank wheel — In this design all of the missing (gap) teeth must be contiguous. Some more specialized crank wheels have multiple blocks of missing teeth to accommodate fast start synchronization strategies. These are not supported by this function set.

Single instance of the engine position functions — Only a single instance of these functions is supported. This is because the overall engine position status variable is global within the eTPU. This means that on a 2 engine eTPU system it is possible to run only one instance of the engine position functions.

Minimum achievable RPM — The following tables show the minimum RPM values that can be achieved without computational error for various TCR1 timebase frequency and common crank wheel configurations.

Table 1. Minimum RPM for a 36 – 1 Crank Wheel for Various Angular Resolutions and TCR1 Frequencies

Crank Type	Angular Resolution (degrees)	TCR1 Resolution (ns)	Min RPM
36-1	0.1	15.2	52.5
36-1	0.05	15.2	52.5
36-1	0.01	15.2	52.5
36-1	0.1	25.0	31.8
36-1	0.05	25.0	31.8
36-1	0.01	25.0	31.8
36-1	0.1	30.3	26.2
36-1	0.05	30.3	26.2
36-1	0.01	30.3	26.2
36-1	0.1	50.0	15.9
36-1	0.05	50.0	15.9
36-1	0.01	50.0	15.9
36-1	0.1	60.6	13.1
36-1	0.05	60.6	13.1
36-1	0.01	60.6	13.1
36-1	0.1	100.0	7.9
36-1	0.05	100.0	7.9
36-1	0.01	100.0	7.9
36-1	0.1	121.2	6.6
36-1	0.05	121.2	6.6
36-1	0.01	121.2	6.6
36-1	0.1	200.0	4.0
36-1	0.05	200.0	4.0
36-1	0.01	200.0	4.0

Table 2. Minimum RPM for a 60–2 Crank Wheel for Various Angular Resolutions and TCR1 Frequencies

crank type	Angular Resolution (degrees)	TCR1 resolution (ns)	Min RPM
60-2	0.05	25	19.1
60-2	0.01	25	19.1

crank type	Angular Resolution (degrees)	TCR1 resolution (ns)	Min RPM
60-2	0.05	50	9.5
60-2	0.01	50	9.5
60-2	0.05	100	4.8
60-2	0.01	100	4.8
60-2	0.05	200	2.4
60-2	0.01	200	2.4
60-2	0.05	15.15152	31.5
60-2	0.01	15.15152	31.5
60-2	0.05	30.30303	15.7
60-2	0.01	30.30303	15.7
60-2	0.05	60.60606	7.9
60-2	0.01	60.60606	7.9
60-2	0.05	121.2121	3.9
60-2	0.01	121.2121	3.9

NOTE

In this case 0.1 degree resolution has been omitted. . This is because in these cases the TRR calculation overflows.

Limitations in calculating the tick rate — To provide a TCR2 value that tracks the position the engine CRANK function calculates a value for the tick rate register (TRR). The number of TCR1 counts per TCR2 count is stored in the tick rate register. TRR adjusts according to whether the engine is accelerating or decelerating. It is recalculated on every active tooth edge based on the most recent tooth period measurements. The CRANK function is responsible for measuring Tooth_Period using timer counter register 1 (TCR1). This Tooth_Period is divided by a user defined parameter, Ticks_Per_Tooth to establish the rate at which the TCR2 angle counter register is incremented. The TRR in the eTPU angle counter hardware must be populated with this rate. TRR has a 9.15 fractional format. To achieve this format the following calculation is performed:

trr = ((Tooth_Period_A << 3) / (Ticks_Per_Tooth)) << 6;

Under certain conditions the calculated TRR value can have an error due to rounding errors introduced by this simple method of calculation. If the Ticks_Per_Tooth parameter is more than 200 or a slow TCR1 frequency is being used (which would result in small Tooth_Period_A values (< 100) then errors of more than 2% in the TRR value may be encountered. However, these are considered corner cases and have not been dealt with in this design. It is up to the user to ascertain if this method is suitable for their requirements. More complex calculation methods are possible which deal with the limitations presented by this simple method, these involve more computational steps.

Additionally, this calculation will fail to give an accurate TRR value if either of the left shift operations results in a value greater than 0xFFFFFF. If the TCR1 prescaler is small and the RPM is low then Tooth_Period_A will be large. This is more of an issue on a 36 – 1 CRANK than on a 60 – 1 CRANK. Careful selection of TCR1 prescaler is required to avoid this issue. If Ticks_Per_Tooth is small then this can produce an overflow situation. A more comprehensive method is described in the eTPU Reference Manual.

Table 3. 36 – 1 Crank Avoiding Overflows in the TRR Calculation (overflows are highlighted)

RPM	Edges/ Second	Ticks_ Per_ Tooth	Fys (MHz)	TCR1 freq	Tooth_ Period_A (TCR1 counts)	Tooth_ Period_A <<3	(Tooth_ Period_A <<3)/ ticks_ per_tooth	((Tooth_ Period_A <<3)/ ticks_ per_tooth)) <<6
30	18	1024	132	66000000	3666667	29333336	28646	1833344
30	18	1024	132	33000000	1833333	14666664	14323	916672
30	18	1024	132	257812.5	14323	114584	112	7168
30	18	1024	80	40000000	2222222	17777776	17361	1111104
30	18	1024	80	20000000	1111111	8888888	8681	555584
30	18	1024	80	156250	8681	69448	68	4352
30	18	1	132	66000000	3666667	29333336	29333336	1877333504
30	18	1	132	33000000	1833333	14666664	14666664	938666496
30	18	60	132	33000000	1833333	14666664	244444	15644416
30	18	1	132	257812.5	14323	114584	114584	7333376
30	18	1	80	40000000	2222222	17777776	17777776	1137777664
30	18	1	80	20000000	1111111	8888888	8888888	568888832
30	18	35	80	20000000	1111111	8888888	253968	16253952
30	18	1	80	156250	8681	69448	69448	4444672
1200	7200	1024	132	66000000	9167	73336	72	4608
1200	7200	1024	132	257812.5	36	288	0	0
1200	7200	1024	80	40000000	5556	44448	43	2752
1200	7200	1024	80	156250	22	176	0	0
1200	7200	1	132	66000000	9167	73336	73336	4693504
1200	7200	1	132	257812.5	36	288	288	18432
1200	7200	1	80	40000000	5556	44448	44448	2844672
1200	7200	1	80	156250	22	176	176	11264

Table 4. 60 – 2 CRANK Avoiding Overflows in the TRR Calculation

RPM	Edges/ Second	Ticks_ Per_ Tooth	Fys (MHz)	TCR1 freq	Tooth_ Period_A (TCR1 counts)	Tooth_ Period_A <<3	(Tooth_ Period_A P<<3)/ ticks_ per_tooth	((Tooth_ Period_A <<3)/ ticks_ per_tooth))<<6
30	30	1024	132	66000000	2200000	17600000	17188	1100032
30	30	1024	132	33000000	1100000	8800000	8594	550016
30	30	1024	132	257812.5	8594	68752	67	4288

RPM	Edges/ Second	Ticks_ Per_ Tooth	Fys (MHz)	TCR1 freq	Tooth_ Period_A (TCR1 counts)	Tooth_ Period_A <<3	(Tooth_ Period_A P<<3)/ ticks_ per_tooth	((Tooth_ Period_A <<3)/ ticks_ per_tooth))<<6
30	30	1024	80	40000000	1333333	10666664	10417	666688
30	30	1024	80	156250	5208	41664	41	2624
30	30	1	132	66000000	2200000	17600000	17600000	1126400000
30	30	50	132	33000000	1100000	8800000	176000	11264000
30	30	1	132	257812.5	8594	68752	68752	4400128
30	30	1	80	40000000	1333333	10666664	10666664	682666496
30	30	10	80	8000000	266667	2133336	213334	13653376
30	30	1	80	156250	5208	41664	41664	2666496
1200	1200	1024	132	66000000	5500	44000	43	2752
1200	1200	1024	132	257812.5	21	168	0	0
1200	1200	1024	80	40000000	3333	26664	26	1664
1200	1200	1024	80	156250	13	104	0	0
1200	1200	1	132	66000000	5500	44000	44000	2816000
1200	1200	1	132	257812.5	21	168	168	10752
1200	1200	1	80	40000000	3333	26664	26664	1706496
1200	1200	1	80	156250	13	104	104	6656

3.5 Status Variables

Both CAM and CRANK functions have status variables associated with them. The CRANK function also maintains a status variable that reflects the overall status of the engine position system. These status variables and all their respective values are described.

3.5.1 CRANK_Status

The Crank_Status variable has one of the following values:

FS_ETPU_CRANK_BLANK_TIME — The function is waiting until a user specified blank time has expired. Crank teeth are ignored during this period.

FS_ETPU_CRANK_BLANK_TEETH — The function is ignoring a user specified number of teeth.

FS_ETPU_CRANK_FIRST_EDGE — The function is waiting for a valid active first edge (after Blank_time and Blank_teeth).

FS_ETPU_CRANK_SECOND_EDGE — The function is waiting for a valid active second edge (after the first edge).

FS_ETPU_CRANK_TEST_POSSIBLE_GAP — On the next active edge the function has made at least 2 tooth period measurements and can perform the AB portion of the ABa test.

FS_ETPU_CRANK_VERIFY_GAP — The AB test has passed. On the next active edge the function can perform the Ba portion of the ABa test.

FS_ETPU_CRANK_GAP_VERIFIED — The ABa test has passed.

FS_ETPU_CRANK_COUNTING — The function is counting normal teeth. Normal teeth are teeth that are not the tooth before or the tooth after the gap.

FS_ETPU_CRANK_TOOTH_BEFORE_GAP — The function is expecting the next valid active edge to be the tooth before the gap.

FS_ETPU_CRANK_TOOTH_AFTER_GAP– The function is expecting the next valid active edge to be the tooth after the gap.

FS_ETPU_CRANK_TOOTH_AFTER_GAP_NOT_HRM — Similar to FS_ETPU_CRANK_TOOTH_AFTER_GAP. This state guarantees that the angle counter hardware is not in high rate mode. This state is related to a workaround for Errata 2477 that affects particular versions of the eTPU. A conditional compilation flag is provided. See the errata list for target device to determine if this is relevant.

A software finite state machine within the CRANK function determines the status of the CRANK function. See Appendix A for a state transition diagram describing the finite state machine.

3.5.2 CAM_State

The CAM_State variable has one of the following values:

FS_ETPU_CAM_SYNC_SEEK_OR_START_ANGLE — The CAM signal is not yet being filtered by the eTPU channel.

FS_ETPU_CAM_SYNC_ANGLE — The CAM signal is being filtered by the eTPU channel. The active edge is in the angle window.

FS_ETPU_CAM_SYNC_LOST — The CAM signal is being filtered by the eTPU channel. The active edge is not in the angle window.

A software finite state machine within the CAM function determines the status of the CAM function. See Appendix B for a state transition diagram describing the finite state machine.

3.5.3 Engine Position Status

The engine position status variable has one of the following values:

ENG_POS_SEEK — The position of the CRANK and CAM are both unknown. TCR2 is not valid.

ENG_POS_FIRST_HALF_SYNC — The position of the CRANK is known, the position of the CAM is not known. TCR2 has a value between 0 and a half engine cycle (360°) worth of TCR2 counts.

ENG_POS_PRE_FULL_SYNC — The position of the CRANK and CAM are both known. It has a value between 0 and a half cycle (360°) worth of TCR2 counts. TCR2 becomes valid the next time the gap occurs.

ENG_POS_FULL_SYNC — The position of the CRANK and CAM are both known. TCR2 is now valid.

ENG_POS_HALF_SYNC — The position of the CRANK is known; the position of the CAM has become unknown. A valid active CAM edge has not been found in the acceptance angle window. TCR2 may be considered valid if it is acceptable to the application. The CAM edge was not found where it was expected to be.

A software finite state machine within the CRANK and CAM functions determines the value of the engine position status variable. See Appendix 3 for a State Transition diagram describing the finite state machine.

3.6 Function Interactions

The CRANK and CAM functions each provide part of the functionality for the engine positioning system. The role of the CRANK function is, wth assistanc from the eTPU hardware, to create TCR2 in such a way that TCR2's value tracks the position of the crank wheel. The crank wheel rotates twice for every cam wheel rotation. The engine position cannot be fully known without the position of both the cam and crank wheels being known.

In this design the CRANK function holds the overall engine position status variable. When the status of the CAM function changes the CAM function communicates with the CRANK function by populating the link service request register with the value which corresponds to the CRANK function channel’s number (0 on a single engine eTPU microcontroller). In this situation the eTPU will service the CRANK channel and update the overall position status variable according to the following table:

Table 5. Engine Position Status Derivation

Current engine position status	CAM_State = SYNC_ANGLE	CAM_State = LOST
ENG_POS_FIRST_HALF_SYNC	ENG_POS_PRE_FULL_SYNC	X
ENG_POS_HALF_SYNC	ENG_POS_FULL_SYNC	X
X	X	ENG_POS_HALF_SYNC

NOTE

- If the CAM_State is lost then the engine position status will be ENG_POS_HALF_SYNC irrespective of the present engine position status.
- The value in the table is the new engine position status and is derived based on the current engine position status and the CAM_State variable
- Refer to the state transition diagrams in Appendix C

At start-up any cam edge is effectively ignored until the CRANK gap has been identified. In this case if a CAM edge occurs, the TCR1 value is captured. No angular information is yet available or valid. Once the gap has been found, CAM edges can be processed and the CAM function is allowed to signal the CRANK function to update the overall engine position status variable.

The CAM function signals CRANK via a link service request on the following events:

- Active cam edge and overall engine position status variable does not equal ENG_POS_SEEK (for example CRANK function has recently found and verified the gap). Software ensures that this link service request is generated once per gap finding sequence.
- Active cam edge and cam status variable has changed.
- Cam window timed out (no active edge in the angle window) and the cam status variable has changed.

3.7 Engine_Cycle_Zero_Count – Reference Point for Each Engine Cycle

In this design a non-resetting angle counter has been implemented. The TCR2 value is not reset to zero at some point during every engine cycle. CAM and all other output functions in this design maintain a variable that is a reference point for the start of each engine cycle.

The CRANK function API calculates the number of TCR2 counts in an engine cycle. This is based on the number of physical teeth, the number of missing teeth (in the gap), and the number of ticks specified in a tooth, calculated as follows:

```
tcr2_counts_per_engine_cycle = crank_tcr2_ticks_per_tooth * ((crank_number_of_physical_teeth +
crank_number_of_missing_teeth) *2);
```

Both the CAM and CRANK function have a parameter called Angle_Start_Count. These parameters are both set to tcr2_counts_per_engine_cycle.

This Angle_Start_Count value is written to TCR2 when the gap has been identified, verified, and the cam position is known. The CRANK function begins incrementing TCR2 and tcr2_counts_per_engine_cycle is added on every engine revolution (720°).

The CAM function maintains the Engine_Cycle_Zero_Count variable for events to be scheduled relative to it. At initialization, Engine_Cycle_Zero_Count is also set to Angle_Start_Count, after the CRANK function has verified the gap. On every subsequent active cam edge or on every subsequent CAM acceptance angle window timeout (these events are mutually exclusive) Engine_Cycle_Zero_Count is incremented by tcr2_counts_per_engine_cycle. The cam edge can be windowed in an angle relative to Engine_Cycle_Zero_Count. The figure below shows the principle involved.

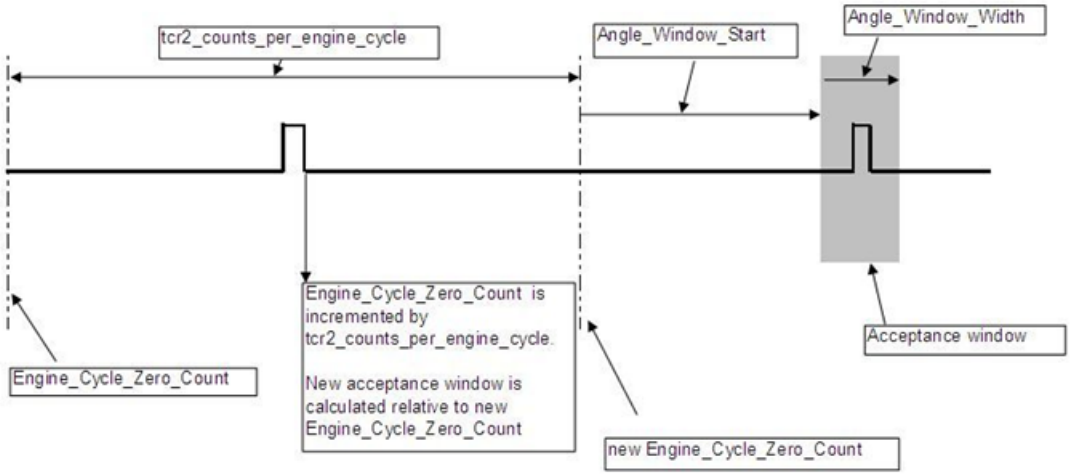


Figure 6. CAM Engine_Cycle_Zero_Count

3.8 Communication with Output Functions

Each output function in this design maintains an Engine_Cycle_Zero_Count variable similar to that maintained by CAM. Similar to CAM this must be initialized to tcr2_counts_per_engine_cycle. As explained in the section EngineCycle Zero Count, the Engine_Cycle_Zero_Count parameter must be updated every cycle. It is recommended this be updated on the last event of each engine cycle.

If CRANK detects a serious error that means the engine position is no longer certain, it sends link service requests (LSRs) to a list of channels as specified by the API parameters crank_link_1, crank_link_2, crank_link_3, and crank_link_4. See Intialization Function. Up to 15 channels in addition to the CAM channel can be linked in this manner. See C Level API for eTPU Engine Positions for a description of these API parameters.

When an output channel services a link service request it must drive the output to a defined state. The engine position is no longer certain and to continue driving FUEL or SPARK may result in damage to the engine. The output channels must also reset the Engine_Cycle_Zero_Count parameter to tcr2_counts_per_engine_cycle. This is because the CRANK function begins looking again for the gap. After the gap has been found and verified the CRANK function begins generating TCR2 values beginning at Angle_Start_Count.

3.9 Custom CAM Function Requirements

The CAM function provided in the example software supports only a single lobe. The majority of crank wheels are one of two configurations (36–1 or 60–2). Cam wheels have numerous configurations allowing for fast start strategies. The example CAM function was written for the simplest case, a single lobe. It is anticipated that some users may want to substitute the CAM function with another better suited for their own specific CAM configuration.

To interface correctly with the example CRANK function a custom CAM function must:

1. Have an Angle_Start_Count parameter, Host, or eTPU software is required to set this to tcr2_counts_per_engine_cycle prior to the function being initialized.
2. Maintain an Engine_Cycle_Zero_Count type parameter. See section Engine Cycle Zero Count. This parameter needs to be set to the value of Angle_Start_Count under the following conditions:
   a. At initialization
   b. Whenever the CAM channel services a link service request.
3. The Engine_Cycle_Zero_Count parameter must be incremented by tcr2_counts_per_engine_cycle once per revolution. It is recommended to perform this on the last service event for the CAM of each engine revolution.
4. Reference a global variable called Eng_Pos_Sync_Status_g to determine CAM's status.
5. Maintain a CAM_State variable of type uint8. The CRANK function has a C pointer to this parameter and uses it to determine the overall engine position status. The CAM_State variable has the following valid values:
   a. #define FS_ETPU_CAM_SYNC_SEEK_OR_START_ANGLE 0

b. #define FS_ETPU_CAM_SYNC_ANGLE 1
c. #define FS_ETPU_CAM_SYNC_LOST 2

6. The CAM_State variable changes under the following conditions:
   a. From FS_ETPU_CAM_SYNC_SEEK_OR_START_ANGLE to FS_ETPU_CAM_SYNC_ANGLE when Eng_Pos_Sync_Status_g (the overall engine position status) is no longer ENG_POS_SEEK. This means that the CRANK function has identified and verified the gap.
   b. From FS_ETPU_CAM_SYNC_ANGLE to FS_ETPU_CAM_SYNC_LOST when the active CAM edge is not found in the angle acceptance window.
   c. From FS_ETPU_CAM_SYNC_LOST to FS_ETPU_CAM_SYNC_ANGLE when the active CAM edge is found in the angle acceptance window (where it has previously been found).

Refer to the example CAM function source code and state transition diagram in Appendix 3 to gain a better understanding of these recommendations.

3.10 Performance and Use of eTPU CRANK and CAM Functions

Performance — Like all eTPU functions, performance of the CRANK and CAM functions in an application is to some extent dependent upon the service time (latency) of other active eTPU channels. This is due to the operational nature of the eTPU scheduler. Increased eTPU loading potentially results in increased latencies. However, worst-case latency in any eTPU application can be closely estimated. To analyze the performance of an application that appears to approach the limits of the eTPU, use the guidelines given in the eTPU reference manual and the information provided in the eTPU engine position software release available from Freescale.

Changing operation modes — To re-configure either the CAM function or the CRANK function on a channel while it is still running, both channels must first be disabled. This can be done using the fs_etpu_disable function that can be found in file etpu_utils.h.

4 C Level API for eTPU Engine Position Functions CRANK and CAM

The following functions provide easy access to interface to the CRANK and CAM functions. Use of these functions eliminates the need to directly control the eTPU registers. These functions can be found in the etpu_eng_pos.h and etpu_eng_pos.c files. The functions are described below and are available from Freescale. In addition, the eTPU compiler generates files called etpu_crank_auto.h and etpu_cam_auto.h. These files contain information relating to the eTPU CRANK and CAM functions, details on how the eTPU data memory is organized, and definitions for various API parameters.

The API consists of 20 functions:

- Initialization function
  fs_etpu_app_eng_pos_init

- Read parameter functions
  fs_etpu_eng_pos_get_cam_edge_angle
  fs_etpu_eng_pos_get_tooth_number
  fs_etpu_eng_pos_get_tooth_time
  fs_etpu_eng_pos_get_engine_speed

- Read status functions
  fs_etpu_eng_pos_get_engine_position_status
  fs_etpu_eng_pos_get_cam_status
  fs_etpu_eng_pos_get_crank_status

- Read error status functions
  fs_etpu_eng_pos_get_cam_error_status
  fs_etpu_eng_pos_get_crank_error_status

- Clear error status functions

fs_etpu_eng_pos_clear_cam_error_status
fs_etpu_eng_pos_clear_crank_error_status

- Update parameter functions
  fs_etpu_eng_update_cam_window
- Set windowing ratio functions
  fs_etpu_eng_pos_set_wr_normal
  fs_etpu_eng_pos_set_wr_after_gap
  fs_etpu_eng_pos_set_wr_across_gap
  fs_etpu_eng_pos_set_wr_timeout
- Advanced functions
  fs_etpu_eng_pos_insert_tooth
  fs_etpu_eng_pos_adjust_angle
- Debug functions
  fs_etpu_eng_pos_get_engine_cycle_0_tcr2_count
  fs_etpu_eng_pos_set_tooth_number

4.1 Initialization Function — fs_etpu_app_eng_pos_init

```
int32_t fs_etpu_app_eng_pos_init (uint8_t cam_channel,
                                  uint8_t cam_priority,
                                  uint8_t cam_edge_polarity,
                                  uint32_t cam_angle_window_start,
                                  uint32_t cam_angle_window_width,
                                  uint8_t crank_channel,
                                  uint8_t crank_priority,
                                  uint8_t crank_number_of_physical_teeth,
                                  uint8_t crank_number_of_missing_teeth,
                                  uint8_t crank_blank_tooth_count,
                                  uint32_t crank_tcr2_ticks_per_tooth,
                                  ufract24_t crank_windowing_ratio_normal,
                                  ufract24_t crank_windowing_ratio_after_gap,
                                  ufract24_t crank_windowing_ratio_across_gap,
                                  ufract24_t crank_windowing_ratio_timeout,
                                  ufract24_t crank_gap_ratio,
                                  uint32_t crank_blank_time_ms,
                                  uint32_t crank_first_tooth_timeout_us,
                                  uint32_t crank_link_1,
                                  uint32_t crank_link_2,
                                  uint32_t crank_link_3,
                                  uint32_t crank_link_4,
                                  uint32_t tcr1_timebase_freq );
```

This function initializes two channels to use CAM and CRANK functions.

For these functions to run, they need to use some of the eTPU data memory. There is not any fixed amount of data memory associated with any channel in the eTPU. The memory needs to be allocated in a way that makes sure each channel has its own memory that will not be used by any other channels. There are two ways to allocate this memory: automatically or manually. Using automatic allocation to initialize each channel, reserves some of the eTPU data memory for its own use. With manual configuration, the eTPU data memory is defined when the system is designed.

Automatic allocation is simpler and is used in all of the program examples. The function uses automatic allocation if the channel parameter base address (CPBA) field for a channel is zero. This is the reset condition of the field, normally you do not need to do anything except call the initialization API function.

If the initialization function is called more than once, it only allocates data memory the first time it is called. The initialization function writes a value to the channel parameter base address field on subsequent calls. It does not allocate more memory.

If the eTPU data memory is allocated manually, then a value must be written to the channel parameter base address before the initialization function is called. This is normally only used if the user wants to pre-define the location of each channels' data memory.

This function has the following parameters:

- cam_channel (uint8_t) — The CAM channel number. For devices with two eTPUs, this parameter must be assigned a value of 1–31 for eTPU_A and 65–95 for eTPU_B. For products with a single eTPU, this parameter must be assigned a value of 1–31.
- cam_priority (uint8_t) — The priority to assign to the eTPU CAM channel. The priority setting is used by the eTPU scheduler. The following eTPU priority definitions are found in utilities file etpu_utils.h.
  FS_ETPU_PRIORITY_HIGH
  FS_ETPU_PRIORITY_MIDDLE
  FS_ETPU_PRIORITY_LOW
  FS_ETPU_PRIORITY_DISABLED

- cam_edge_polarity (uint8_t) — The polarity if the CAM edge, can be
  FS_ETPU_CAM_FM0_RISING_EDGE or
  FS_ETPU_CAM_FM0_FALLING_EDGE

- cam_angle_window_start (unit32_t) — The starting position of the CAM acceptance window. Range 0 to 71999. 71999 represents 719.99°.
- cam_angle_window_width (unit32_t) — The width of the CAM acceptance window. Range 0 to 71999. 71999 represents 719.99 °.
- crank_channel (uint8_t) — This is the CRANK channel number. Can be either 0 for ETPU_A or 64 for ETPU_B. These channels have special hardware that allows special filtering of the CRANK signal
- crank_priority (uint8_t) — The priority to assign to the eTPU CRANK channel. The priority setting is used by the eTPU scheduler. The following eTPU priority definitions are found in utilities file etpu_utils.h.
  FS_ETPU_PRIORITY_HIGH
  FS_ETPU_PRIORITY_MIDDLE
  FS_ETPU_PRIORITY_LOW
  FS_ETPU_PRIORITY_DISABLED

- crank_number_of_physical_teeth (uint8_t) — Number of physical teeth on the crank wheel. Can be 1 to 255. For a 36 –1 wheel configuration this parameter would be 35.
- crank_number_of_missing_teeth (uint8_t) — Number of missing teeth on the crank wheel. Can be 1, 2, or 3. For a 36 –1 wheel configuration this parameter would be 1.
- crank_blank_tooth_count (uint8_t) — Number of teeth to be ignored after first active edge. Can be 0 to 255.
- crank_tcr2_ticks_per_tooth (uint32_t) — Number of TCR2 counts per tooth. Can be 1 to 1024. Refer to Section 3.4 Software Limitations calculating the tick rate for details of how to determine a good value for this parameter.
- crank_windowing_ratio_normal (ufract24_t) — A fractional number. Used to window out noise. Can be between 0x0 and 0xFFFFFF. Applied to normal teeth. The ratios below are used for other teeth. Refer to Section 3.2 Windowing Signals.
- crank_windowing_ratio_after_gap (ufract24_t) — A fractional number. Used to window out noise. Can be between 0x0 and 0xFFFFFF. Applied to the second tooth after the gap. Refer to Section 3.2 Windowing Signals.
- crank_windowing_ratio_across_gap (ufract24_t) — A fractional number. Used to window out noise. Can be between 0x0 and 0xFFFFFF. Applied to the first tooth after the gap. Refer to Section 3.2 Windowing Signals.
- crank_windowing_ratio_timeout (ufract24_t) — A fractional number. Used to window out noise. Can be between 0x0 and 0xFFFFFF. Applied to the tooth immediately following a timeout. Refer to Section 3.2 Windowing Signals.
- crank_gap_ratio (ufract24_t) — A fractional number. Used to identify the gap. Can be between 0x0 and 0xFFFFFF. Used in the ABa test. Refer to Section 3 Functional Description.
- crank_blank_time_ms — The amount of time that teeth are ignored for at start up in milli-seconds. crank_blank_time_ms has a maximum value given by:
  (0xFFFFFF/tcr1_timebase_freq) * 1000.

- crank_first_tooth_timeout_us — The amount of time in micro-seconds after the first tooth times out. Maximum value given by:
  (0xFFFFFF/tcr1_timebase_freq) * 1000000

- crank_link_1 — This is a packed 32-bit parameter containing 4 x 8 bit channel numbers. These channels are signalled when CRANK has had to re-synchronize. (For example, after STALL). In this design the CAM channel number must be included in the list. Unused elements of this list should be padded with an unused channel number or with the number of a channel that has been assigned a function that does not respond to links. See the figure below for details. (0b11001000 would link to channel 8 on the other eTPU engine; 0b00001000 would link to channel 8 on the current eTPU engine.). On a single eTPU device the engine selection bit field should be set to 0b00 or 0b01.

7	6	5	4	3	2	1	0
Engine Selection		reserved[1]	Channel Number				

[1] Reserved bits must be written 0.

Engine Selection	Description
00	This engine
01	Engine 1
10[1]	Engine 2
11	The other engine

[1] This can occur only if the link service request came from the other engine or from serviced channel itself.

Figure 7. Micro Engine Link Register

- crank_link_2 (uint32_t) — See description of crank_link_1.
- crank_link_3 (uint32_t) — See description of crank_link_1.
- crank_link_4 (uint32_t) — See description of crank_link_1
- timebase_freq (uint32_t) — This is the pre-scaled frequency in hertz of the TCR1 timebase.

4.2 Read Parameter Functions

- uint32_t fs_etpu_eng_pos_get_cam_edge_angle() — This API function has no parameters. It returns the angle relative to Engine_Cycle_Zero_Count at where the last CAM edge occurred. The value returned is scaled so that 71999 represents 719.99°.
- int8_t fs_etpu_eng_pos_get_tooth_number() — This API function has no parameters. It returns the tooth number of the last tooth that was accepted by the CRANK function. The tooth immediately after the gap is numbered 1. Note that the tooth number is not incremented on the tooth immediately prior to the gap. The tooth numbering scheme for a 36–1 CRANK wheel is shown below.

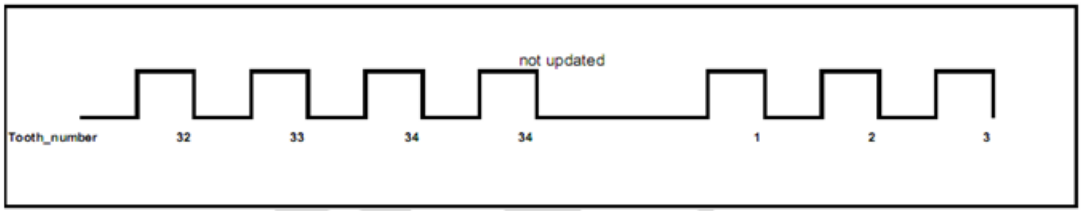


Figure 8. Tooth Numbering Scheme For a 36–1 Crank Wheel

- int32_t fs_etpu_eng_pos_get_tooth_time() — This API function has no parameters. It returns the TCR1 at which the last accepted CRANK tooth occurred.
- int32_t fs_etpu_eng_pos_get_engine_speed(uint32_t tcr1_timebase_freq) — This function has the following parameter:

  timebase_freq (uint32_t) — This is the pre-scaled frequency in hertz of the TCR1 timebase.

  It returns the engine speed in revs per minute (RPM).

4.3 Read Status Functions

- int8_t fs_etpu_eng_pos_get_engine_position_status() — This API function has no parameters. It returns the overall engine position status. Refer to Section 3.5.3 Engine Position Status for a description of the possible return values.

- int8_t fs_etpu_eng_pos_get_cam_status() — This API function has no parameters. This API function returns the status of the CAM function. Refer to Section 3.5.2 CAM State for a description of the possible return values.

- int8_t fs_etpu_eng_pos_get_crank_status() — This API function has no parameters. This API function returns the status of the CRANK function. Refer to Section 3.5.1 CRANK Status for a description of the possible return values.

4.4 Read Error Status Functions

- int8_t fs_etpu_eng_pos_get_cam_error_status() — This API function has no parameters. This API function returns the error status of the CAM function. Possible return values are:

  FS_ETPU_CAM_NO_ERROR — Indicates no error has occurred.
  FS_ETPU_CAM_INVALID_M1 — Indicates an invalid entry condition within the CAM function.
  FS_ETPU_CAM_INVALID_M2 — Indicates an invalid entry condition within the CAM function.

- int8_t fs_etpu_eng_pos_get_crank_error_status() — This API function has no parameters. This API function returns the error status of the CRANK function. Possible return values are:

  FS_ETPU_CRANK_NO_ERROR — Indicates no error has occurred.
  FS_ETPU_CRANK_TIMEOUT — Indicates a single tooth timeout has occurred. An active edge did not occur in the acceptance window.
  FS_ETPU_CRANK_STALL — Indicates two consecutive teeth timed out.
  FS_ETPU_CRANK_INTERNAL_ERROR — Indicates the internal state machine control variable had an unexpected value.
  FS_ETPU_CRANK_INVALID_M1 — Indicates an invalid entry condition within the CRANK function.
  FS_ETPU_CRANK_INVALID_M2 — Indicates an invalid entry condition within the CRANK function.

4.5 Clear Error Status Functions

- int32_t fs_etpu_eng_pos_clear_cam_error_status() — This API function has no parameters. This API function resets any CAM error. The CAM error status word becomes FS_ETPU_ CAM _NO_ERROR.

- int32_t fs_etpu_eng_pos_clear_crank_error_status() — This API function has no parameters. This API function resets any CRANK error. The CRANK error status word becomes FS_ETPU_CRANK_NO_ERROR.

4.6 Update Parameter Functions

- int32_t fs_etpu_eng_update_cam_window (uint32_t cam_angle_window_start, uint32_t cam_angle_window_width); This function is used to update the starting angle and width of the CAM acceptance window. This function has the following parameters:
  - cam_angle_window_start (uint32_t) — The new starting position of the CAM acceptance window. Range 0 to 71999. 71999 represents 719.99°.
  - cam_angle_window_width (unit32_t) — The new width of the CAM acceptance window. Range 0 to 71999. 71999 represents 719.99°.

4.7 Set Windowing Ratio Functions

- int32_t fs_etpu_eng_pos_set_wr_normal (ufract24_t ratio) — This function is used to change the windowing ratio for normal teeth ratios for other types of teeth changed using the API functions listed below. Refer to Section 3.2 Windowing Signals. This function has one parameter:

- ratio (ufract24_t) — A fractional number. Can be between 0x0 and 0xFFFFFF. 0x0 means 0.0 and 0xFFFFFF means 1.0; 0x7fffff means 0.5.

- int32_t fs_etpu_eng_pos_set_wr_after_gap (ufract24_t ratio) — This function is used to change the windowing ratio for the second tooth after the gap. Refer to Section 3.2 Windowing Signals. This function has one parameter:
  - ratio (ufract24_t) — A fractional number. Can be between 0x0 and 0xFFFFFF. 0x0 means 0.0, 0xFFFFFF means 1.0, and 0x7fffff means 0.5.

- int32_t fs_etpu_eng_pos_set_wr_across_gap(ufract24_t ratio) — This function is used to change the windowing ratio for the first tooth after the gap. Refer to Section 3.2 Windowing Signals. This function has one parameter:
  - ratio (ufract24_t): A fractional number. Can be between 0x0 and 0xFFFFFF. 0x0 means 0.0 and 0xFFFFFF means 1.0; 0x7fffff means 0.5.

- int32_t fs_etpu_eng_pos_set_wr_timeout (ufract24_t ratio) — This function is used to change the windowing ratio for the tooth immediately following a timeout. Refer to Section 3.2 Windowing Signals. This function has one parameter:
  - ratio (ufract24_t) — A fractional number. Can be between 0x0 and 0xFFFFFF. 0x0 means 0.0 and 0xFFFFFF means 1.0; 0x7fffff means 0.5.

4.8 Advanced Functions

- int32_t fs_etpu_eng_pos_insert_tooth(uint8_t host_asserted_tooth_count) — This function is used to add a tooth worth of TCR2 counts (that is the crank_tcr2_ticks_per_tooth parameter of the fs_etpu_app_eng_pos_init function) to the TCR2 counter. This is achieved by eTPU software asserting the IPH bit in the tooth program register in the eTPU micro-engine. The eTPU software also sets the tooth number to the one supplied by the host. When this function is used all the inserted TCR2 counts appear on the counter bus, none are skipped and any events scheduled by output functions at these intervening counts will occur. This function has one parameter:
  - host_asserted_tooth_count (uint8_t) — The value the tooth number variable becomes after the eTPU software has added the TCR2 counts.

- int32_t fs_etpu_eng_pos_adjust_angle(int24_t angle_adjust) — This function is used to adjust the value of the TCR2 counter. The angle_adjust parameter value is added to the TCR2 counter value. This function has one parameter:
  - angle_adjust (int24_t) — The amount by which the TCR2 value is adjusted. Notice that this parameter is signed so the TCR2 counter can be incremented or decremented

4.9 Debug Functions

- int32_t fs_etpu_eng_pos_get_engine_cycle_0_tcr2_count() — This function returns the value of the Engine_Cycle_Zero_Count parameter as maintained by the CAM eTPU function. See Sectionh 3.7 Engine Cycle Zero Count for an explanation of the meaning of this parameter.
- int32_t fs_etpu_eng_pos_set_tooth_number(uint8_t tooth_number) — This function is used to set the tooth number which is maintained by the CRANK eTPU function. This function has one parameter:
  - tooth_number (uint8_t) — The value the tooth number variable becomes.

5 Use of Engine Position Functions

API Functions — Order of use

The initialization function, fs_etpu_app_eng_pos_init must be run before any of the other functions. The initialization function must also be run prior to any of the initialization functions for the output functions. In this design the output eTPU functions have a pointer to a variable written by the CAM eTPU function (this variable contains the number of TCR2 counts in an engine cycle). This variable must be initialized by the CAM function prior to any output function otherwise improper operation results.

6 Example of Function Use

This section describes an example use of the engine position function, how to initialize the eTPU module, and assign the eTPU CAM and CRANK functions to eTPU channels. To demonstrate these functions the KNOCK_WINDOW function is used so that an output waveform can be observed. Refer to *AN3772 — Using the Knock Window eTPU Function* for more details on the KNOCK_WINDOW function.

A FreeMASTER project has been created to demonstrate an example use of the CAM, CRANK, and KNOCK_WINDOW functions. To use the example software FreeMASTER software must be downloaded and installed on the user's PC. FreeMASTER software was designed to provide an application-debugging, diagnostic, and demonstration tool for the development of algorithms and applications. It runs on a PC connected to an evaluation booard (EVB) via an RS232 serial cable. A small program resident in the microprocessor communicates with the FreeMASTER software to return status information to the PC and process control information from the PC. FreeMASTER software, executing on a PC, uses part of Microsoft Internet Explorer as the user interface. The FreeMASTER application can be downloaded from the Freescale website.

The demonstration software is written for an MPC5633M device and an XPC56XX EVB Motherboard with an XPC563M 144QFP Mini-Module daughter card. These can be purchased via Freescale website.

A tooth generator eTPU function (TOOTHGEN) has been written to allow the eTPU to provide the input signals for both the CAM and CRANK functions. This function creates a programmable number of crank teeth and a programmable number of missing teeth. The TOOTHGEN function also allows the injection of various errors into the CRANK and CAM signal, for example, missing CRANK teeth, CAM noise, and CRANK noise. The FreeMASTER tool can be used to demonstrate how the CAM and CRANK functions deal with particular error scenarios.

The tooth generator output for the crank signal must be connected to the pin assigned the eTPU channel running the CRANK function. Also, the tooth generator output for the CAM signal must be connected to the pin assigned the eTPU channel running the CAM function. The channel assignments are shown in the following table.

Table 6. Channel Assignments

Channel	TOOTHGEN	Engine Position
CRANK	ETPUA1	ETPUA0
CAM	ETPUA15	ETPUA14
KNOCK_WINDOW		ETPUA10

A short video has been created to show some of the features of the engine position driver. To view this go to Freescale's YouTube channel . The Freemaster interface allows the user to create various scenarios to study the behaviour of the engine position driver. Various error situations can be examined and it is possible to change many of the API parameters, including those of the Tooth generator function. Example software is available from Freescale. For this FreeMASTER demo; look for *AN3769SW*.

A short video is available on Freescale's YouTube channel ASHWARE simulator running a simulation that shows various features of the entire set2 function set. All of the set 2 automotive functions are shown in this video. You can view this video at: ASHWARE set2 simulation . This video allows users to see the ASHWARE simulator in action without having to buy a license.

7 Conclusion

This eTPU Engine position application note provides the user with a description of the CRANK and CAM eTPU functions, usage, and examples. The simple C interface functions to the engine position eTPU functions enable easy implementation of the CAM and CRANK function in applications. The functions are targeted for the MPC55xx/MPC563xM family of devices, but they can be used with any device that contains an eTPU.

Appendix A: Crank State Transition

Following is the state transition diagram for the CRANK function finite state machine.

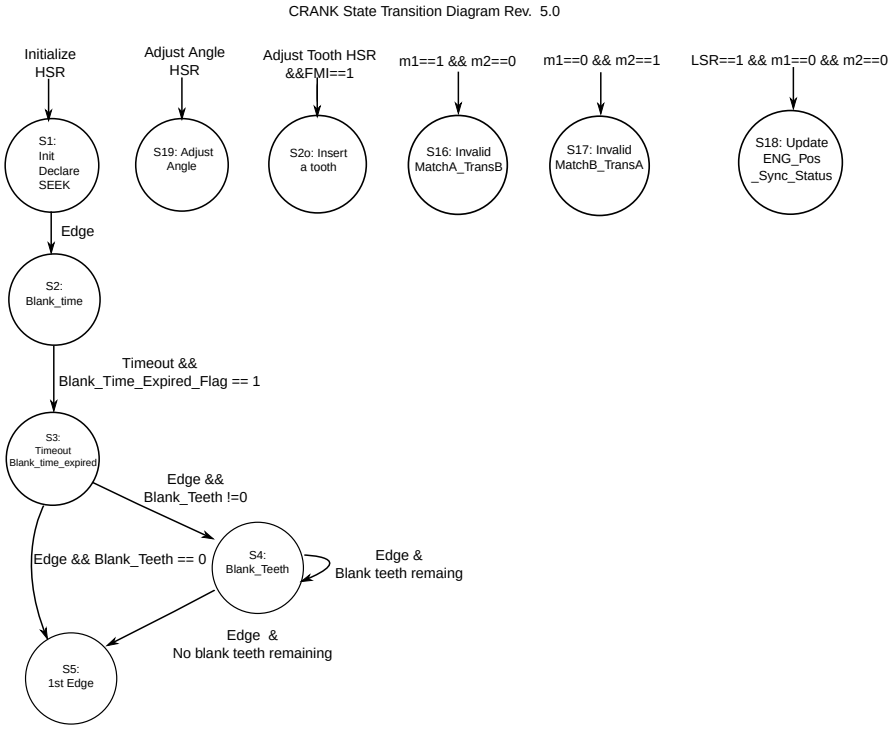
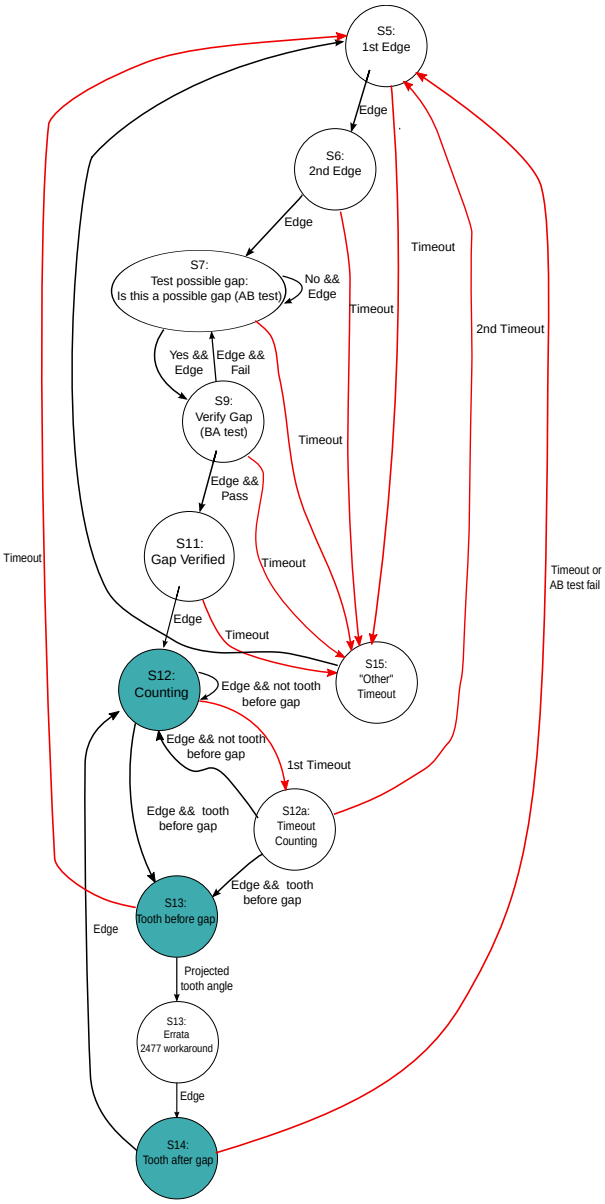


Figure A-1. Crank State Transition Part 1



Notes:
CAM signals CRANK when the overall status needs to change because of the CAM status change.

CRANK maintains SYSYEM_SYNC_STATE

Global Pram must be intialized prior to HSR for CAM and CRANK. This prevents spurious behavior caused by the CRANK function inadvertently seeing an untrue CAM status value

No timeout from BLANK_teeth thread, due to start up engine tooth times this would be excessive(>24 bits)

4 Flag states are used to determine entry to S10 (flags = 1), S11 (flags = 3), S12 (flags = 2) and all other threads (flag = 0).

All states prior to Gap_Verified will set tcr2 = 0

Figure A-2. Crank State Transition Part 2

Appendix B: CAM State Transition

Following is the state transition diagram for the CAM function finite state machine.

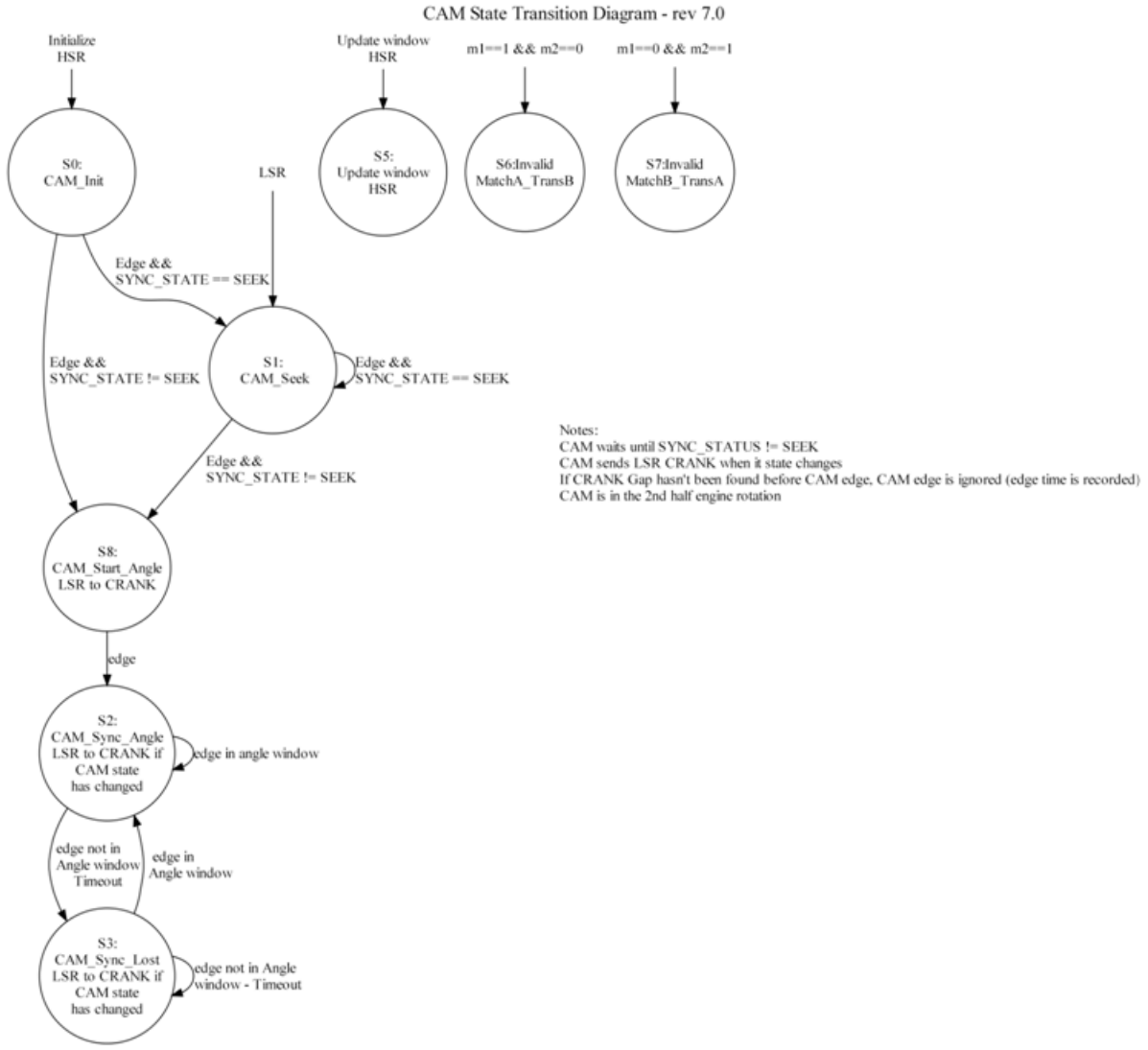


Figure B-1. CAM State Transition

Appendix C: System Sync State Transition

State transition diagram for engine position status finite state machine.

Figure C-1. System Sync State Transition

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
1-800-441-2447 or +1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN3769
Rev. 0, 06/2009