Freescale Semiconductor
Application Note

Document Number: AN4907
Rev. 2, 01/2016

Engine Control eTPU Library

Covers all MPC5500, MPC5600, and MPC5700 Devices Featured with the Enhanced Time Processing Unit (eTPU)

by: Milan Brejl

Contents

1 Introduction

The Enhanced Time Processing Unit (eTPU) is a programmable I/O controller with its own core and memory system, allowing it to perform complex timing and I/O management independent of the CPU. The eTPU module is a peripheral for 32-bit Qorivva powertrain families of MPC5500, MPC5600, and MPC5700.

The eTPU is used as a coprocessor, specialized for advanced timing functions, such as handle complex engine control, motor control, and communication tasks independent of the CPU. The engine management is the most traditional area of eTPU usage.

A new complex library of eTPU functions enabling the eTPU to drive engine control applications was developed. This library represents a major step forward compared to its predecessor – eTPU automotive function set (set2) from 2009. In the updated library, there are new eTPU functions, which can be used to drive modern direct injection engines. The other functions have been widely updated and upgraded following the set2 user's experience, adding new features, simplifying their usage, and introducing much clear and easy to reuse code.

2 Library content and released items

Table 1 shows the engine control eTPU library, which consists of eight eTPU functions sorted to four groups.

Table 1. Engine control eTPU library function list

Group	eTPU Function	Code Name	Purpose
Engine Position	Cam	CAM	Receive and log a signal from a camshaft sensor.
	Crank	CRANK	Receive and process the tooth signal from a crankshaft sensor and generate the eTPU internal angle-base (TCR2).
	Crank Emulator	CRANK_EMUL	Generate the eTPU-internal angle-base (TCR2) without any signal on the input for testing purpose.
	Tooth Generator	TG	Generate a crank and cam tooth pattern for testing purpose.
Injection	Fuel	FUEL	Generate time or angle-based output pulses for driving port-injections.
	Direct Injection	INJ	Generate time or angle-based multi-channel output patterns for driving direct-injections.
Ignition	Spark	SPARK	Generate time or angle-based output pulses for spark ignitions.
Other	Knock Window	KNOCK	Generate a signal for gating and triggering an ADC, for sampling a knock sensor output.

For each of the eight eTPU functions in the library there are several software items released as shown in Table 2. All of them can be found as part of the AN4907SW package available to download at www.freescale.com.

Table 2. Engine control eTPU library software items

Item	File Name	Description
eTPU source code	etpuc_func.c etpuc_func.h	eTPU function C source code.
eTPU simulation scripts	<func>_*.tcl	eTPU function simulation script for use in CodeWarrior for eTPU.
Auto-generated interface files	etpu_<func>.h	eTPU-CPU interface files generated by the eTPU compiler for use by the API.
API source code	etpu_func.c etpu_func.h	CPU application interface (API) C source code.

Table continues on the next page...

Table 2. Engine control eTPU library software items (continued)

Item	File Name	Description
API low-level documentation	FUNC-doxyDoc.chm FUNC-doxyDoc.zip	Generated DoxyGen documentation of the API code, available in two formats. • Compiled Help File (.chm) • HTML (.zip package)

The new Engine Control eTPU Library benefits from new Freescale eTPU development tools from CodeWarrior. The eTPU(2) Development Suite is based on Eclipse IDE and includes the C and asm compiler, simulator, and debugger. Part of the new eTPU library release package is a CodeWarrior development project, including the eTPU source code, project settings, and also simulation scripts. Using the development project, it is straightforward to run a simulation and observe the internal functions' behavior.

Similarly, this application note describes how to get the eTPU functions up and running on the hardware. The application note AN4908 and its source code AN4908SW, both available to download at www.freescale.com, which demonstrates a typical usage of all the Engine Control eTPU Library functions, API calls, initialization, runtime control, and monitoring. The application is dedicated for MPC5674F and features a FreeMASTER graphical interface on a PC.

3 Changes from eTPU set2

The original Automotive eTPU Function Set, called set2, was released as AN3768 and AN3768SW at www.freescale.com in 2009. It consists of five eTPU functions.

- Crank and Cam (together as Engine Position driver)
- Fuel
- Spark
- Knock Window
- Tooth Generator (for testing only)

These functions were capable of driving port injection engines. Modern direct injection engines were not supported. Also, the range of supported crank & cam patterns was limited.

The new Engine Control eTPU Library overcomes these limitations. It aims to serve as ready to use functions for automotive applications. Moreover, it brings the eTPU code for developers to make them understand or even to modify the code for their specific needs.

The new Crank and Cam eTPU functions are generally able to synchronize to any crank and cam signal pattern. A crank wheel with multiple gaps, with an additional tooth instead of a gap, and also multiple cam signals are supported.

The new eTPU function Direct Injection is dedicated to direct solenoid injection drive. The function is able to generate a pattern of synchronous angle or time based output pulses on several output channels, including common bank channels and individual injector channels. For a closed loop solenoid current control, this function can provide the timing for either an external (MC33816) or internal (Reaction Channel) closed loop logic.

The updated functions of Fuel, Spark, and Knock Window are ported to new coding style, which is MISRA compliant, easy to read, reuse, and modify. The functionality is slightly extended from the original set2 versions.

3.1 Changes in API

As before, the usage of eTPU functions in an application is supported by CPU application interface routines, called API. The API is able to initialize an eTPU function and control it during runtime. The new eTPU library introduces a new API style. All eTPU function APIs use a general approach of data structures used to access the eTPU functions.

- **Instance** structure includes parameters used to initialize the eTPU function.
- **Config** structure includes configuration parameters, which are used for initialization and can also be changed in runtime.
- **States** structure includes internal states reflecting the eTPU function operation.

API routines manipulate data between these structures and the eTPU function itself. Based on the eTPU function implementation, it may require to read or write the eTPU data RAM and/or to access eTPU registers. All of that is covered by a simple API interface including functions as given below.

- **fs_etpu_<func>_init()**
- **fs_etpu_<func>_config()**
- **fs_etpu_<func>_get_states()**

The user can create additional API routines for specific needs. For example, the _config() function sets all eTPU function parameters, which can be changed in runtime. In case, a single parameter needs to be frequently changed, a new function can easily be created based on the fs_etpu_<func>_config() function code.

The API routines do not provide any conversion of units. All parameter values must be entered in eTPU- internal raw units.

- **time** values as a number of time-base (TCR1) ticks
- **angle** values as a number of angle-base (TCR2) ticks

TCR1 and TCR2 are two eTPU-internal free-running counters, which serve as a time-base (TCR1) and an angle-base (TCR2). The TCR1 frequency setting is part of eTPU module configuration. The TCR2 angle resolution is configured by parameters of the Crank eTPU function. The unit conversion can be easily handled by a set of macros. An example is provided in the demo application AN4908SW, file named etpu_gct.h.

All cylinder related angles are set relative to the particular cylinder Top Dead Center (TDC). The convention followed by the eTPU Engine Control library defines positive and negative TDC relative angles so that positive angles precede the TDC and negative angles come after, see Figure 1.

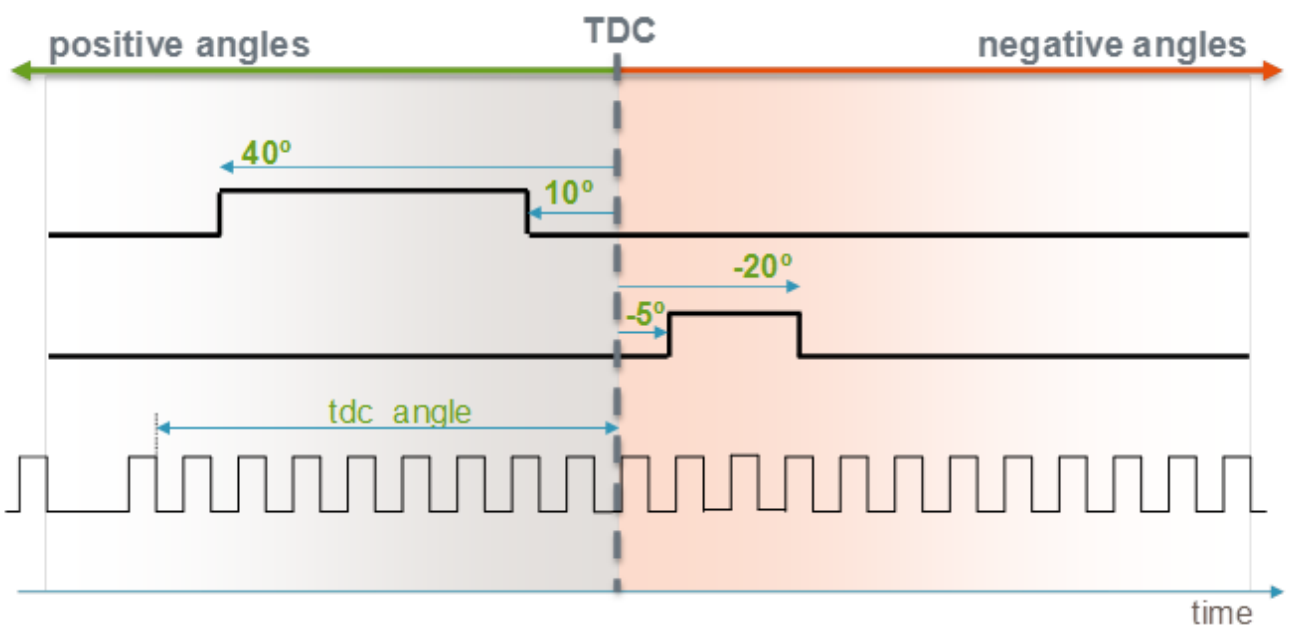

Figure 1. Positive and negative TDC-relative angles

The API routines use a coding style that is able to generate DoxyGen documentation. The generated API documentation is part of the library release.

4 Feature description

The need of precise timing together with the exact motor angle-position knowledge is typical for engine control applications. Especially on the lowest processing level, which is handled by the eTPU hardware and eTPU software, the time and angle needs to be used side by side. The eTPU module features two internal free running counters, called TCR1 and TCR2, which serve as a time-base (TCR1) and an angle-base (TCR2) for all engine control eTPU functions. Then, any input or output action can be related to either time, or angle, or both.

While the time-base is derived from system clocks, the angle base needs to be derived from the engine crankshaft and camshaft position sensors. This is enabled by the eTPU angle logic and the engine position library functions.

Once the angle-base is running and synchronized, the other library functions can start to generate output signals with edges that are defined in time or angle or both. These signals drive the injection and ignition, or generate another time or angle related output signal.

In case the engine angle synchronization is lost, the engine position function sends an eTPU-internal link to all the output generating functions. On the link, the outputs are stopped. Once the synchronization is achieved again, the outputs automatically start again.

4.1 Engine position

There are four eTPU functions in the library related to the engine position processing. The **Crank** eTPU function together with the **Cam** function process the input signals from crankshaft and camshaft sensors.

For testing purpose, there are other two supporting eTPU functions. When the real signal from the crankshaft and camshaft sensors are not present, the **Tooth Generator** function is able to generate similar signals. Then, the Tooth Generator outputs need to be connected to the Crank and Cam inputs through an external connection. Such a connection might not be an option on a production board. In this case, there is a pure software solution. The Crank eTPU function can be replaced by the Crank Emulator function, which controls the eTPU internal angle-base without any signal on the input pin. Instead, the CPU can set the engine speed to be emulated.

An example of the engine synchronization process is depicted in Figure 2.

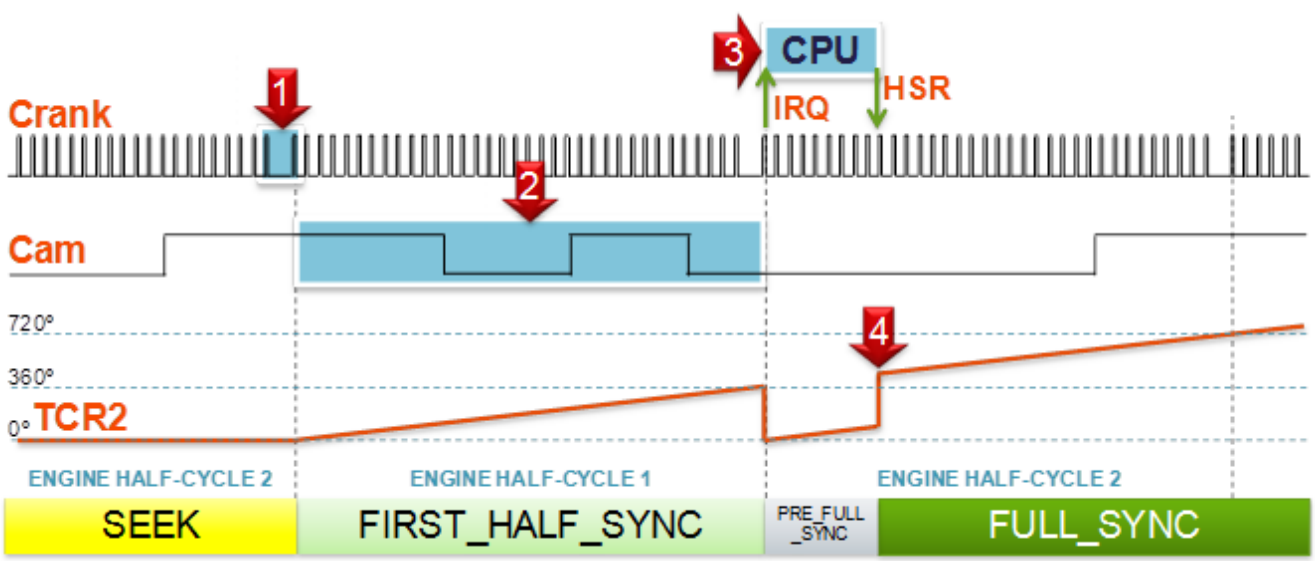

Figure 2. Engine synchronization example

The synchronization of eTPU TCR2 angle counter to the physical rotation of the Crank wheel is, from the software point of view, a result of a sequence of processing on both the eTPU and the CPU site.

1. eTPU function Crank recognizes the gap or the additional tooth on the crank wheel. eng_pos_state is set to FIRST_HALF_SYNC.
2. eTPU function Cam logs cam signal transitions for teeth_per_sync crank teeth. After that Crank sets channel interrupt flag (IRQ), the eng_pos_state is set to PRE_FULL_SYNC.
3. On the Crank interrupt, CPU uses the Cam log to decode the engine position. The CPU writes the decoded TCR2 engine angle (corresponding to the first tooth after gap) together with a Host Service Request (HSR) to Crank eTPU function (call of function fs_etpu_crank_set_sync).
4. eTPU function Crank adjusts the engine angle TCR2 value and sets eng_pos_state to FULL_SYNC. Full synchronization is achieved.

4.1.1 Cam

The Cam eTPU function (CAM) uses one eTPU channel to log input signal transitions. More instances of CAM can be initialized to log more inputs.

Cam has the following features.

- Based on the selected mode, rising, falling or both signal transitions are detected and logged.
- The log buffer size is configurable.
- The log can be reset automatically by a link from Crank eTPU function or manually by the CPU application. Resetting the log means setting the buffer index to the buffer start position so that the next transition overwrites the first value in the buffer.
- Number of transitions logged during the last engine cycle (between last two log resets) and the actual position in the log buffer are available to read.
- Two error conditions are reported.
  - FS_ETPU_CAM_ERROR_ZERO_TRANS – no input transition was logged during the last engine cycle (between last two log resets). It means the cam signal is lost.
  - FS_ETPU_CAM_ERROR_LOG_OVERFLOW – the log buffer size is not enough to log all input transitions. The last transition was not logged.
- Channel interrupt is generated on detecting an error condition.

A single item in the log buffer is a 32-bit word which includes the following.

- Transition TCR2 angle in the lower 24 bits
- Transition polarity (0-falling, 1-rising) in the upper 8 bits

Figure 3 depicts the CAM logging functionality.

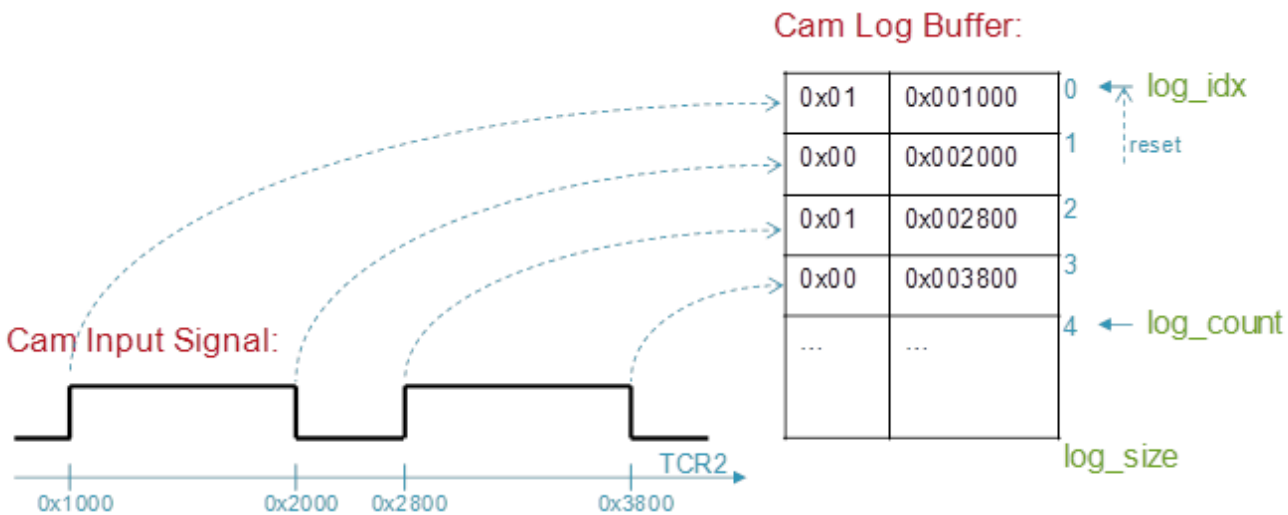

Figure 3. CAM logging functionality

The Cam API is used to control and monitor the eTPU function using the following functions.

- fs_etpu_cam_init()
- fs_etpu_cam_init()
- fs_etpu_cam_get_states()
- fs_etpu_cam_copy_log()
- fs_etpu_cam_reset_log()

These API functions handle the following data structures.

- cam_instance
- cam_config
- cam_states

Figure 4 shows Cam API functions, the state machine of calls, and data structures.

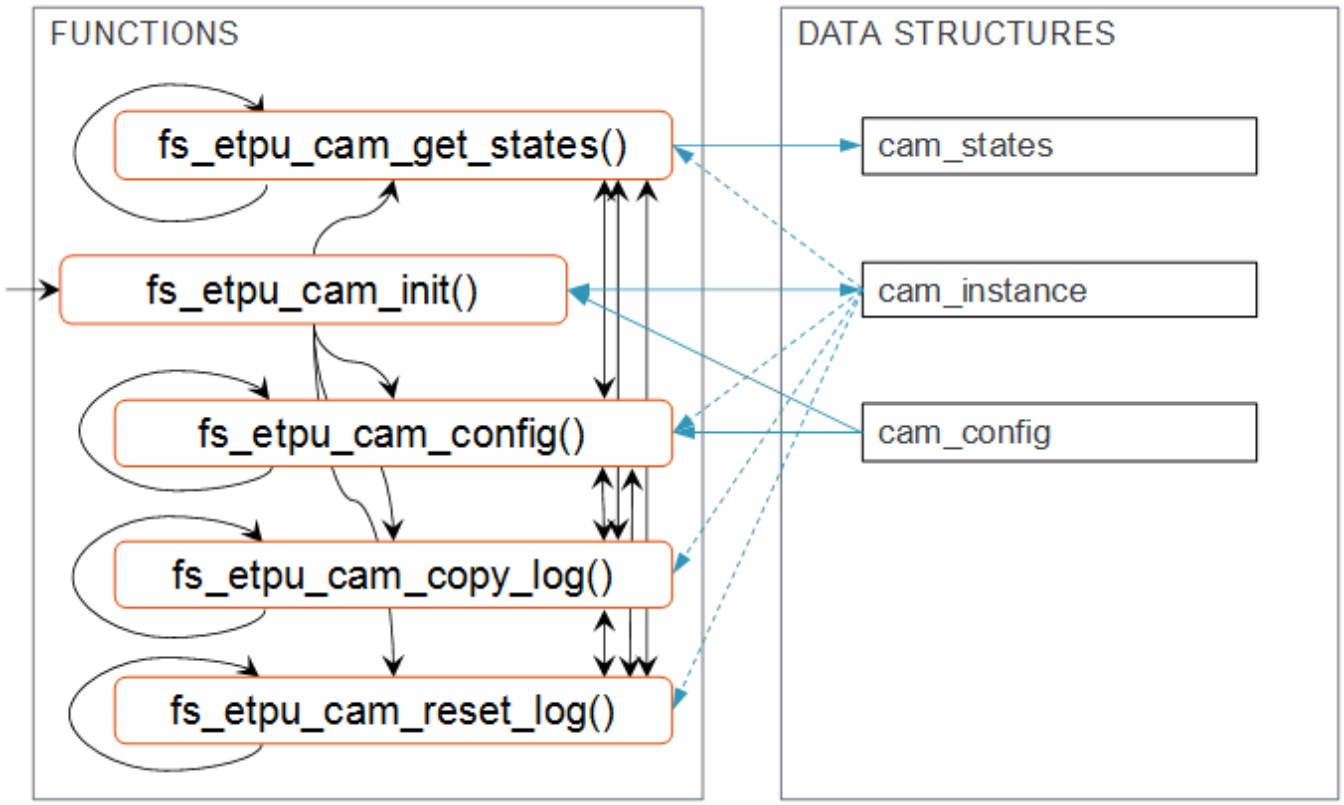

Figure 4. Cam API functions and data structures

The data structures include the following fields.

```
struct cam_instance_t:
        const uint8_t     chan_num
        const uint8_t     priority
        const uint8_t     log_size
              uint32_t  * cpba
              uint32_t  * cpba_log

struct cam_config_t:
              uint8_t     mode

struct cam_states_t:
              uint8_t     error
              uint8_t     log_count
              uint8_t     log_idx
```

For API description details, refer to CAM-DoxyDoc.chm or CAM-DoxyDoc.zip files, which are included in AN4907SW.

4.1.2 Crank

The Crank eTPU function uses one eTPU channel to process the tooth signal from a crankshaft sensor and generate the eTPU-internal angle-base (TCR2). The Crank eTPU function uses the Enhanced Angle Counter (EAC) eTPU hardware.

The Crank eTPU function can be assigned to one of (remember to set the TBCR.AM field correspondingly).

- eTPU channel 0, input signal connected to TCRCLK
- eTPU channel 1, input signal connected to channel 1 input (eTPU2 only)
- eTPU channel 2, input signal connected to channel 2 input (eTPU2 only)

Crank has the following features.

- Based on the selected polarity, either rising or falling signal transitions are detected.
- There are various tooth patterns supported.
  - Single gap
  - Multiple equally spaced gaps
  - An additional tooth instead of a gap
- The tooth pattern is described as given below (see Figure 5).
  - Number of teeth between two gaps (teeth_till_gap)
  - Number of missing teeth in one gap (teeth_in_gap)
  - Number of teeth per one engine cycle (teeth_per_cycle). An additional tooth instead of a gap is characterized by teeth_in_gap = 0.
- The number of angle-base counts per one tooth is configurable (ticks_per_tooth).
- The gap is recognized using an ABA test and a gap_ratio parameter.
  - AB part: tooth_period_A1 < gap_ratio * tooth_period_B
  - BA part: gap_ratio * tooth_period_B > tooth_period_A2. where tooth_period_A1, tooth_period_B and tooth_period_A2 are consecutive tooth periods.
- A noise immunity and check of unexpected acceleration/deceleration are achieved using tooth acceptance windows, characterized by (see Figure 6 till Figure 9)
  - win_ratio_normal
  - win_ratio_across_gap
  - win_ratio_after_gap
  - win_ratio_after_gap
- The measured tooth periods can optionally be logged to an array.
- A Crank state and a global eng_pos_state are handled. The Crank state can be one of the following.
  - FS_ETPU_CRANK_SEEK
  - FS_ETPU_CRANK_BLANK_TIME
  - FS_ETPU_CRANK_BLANK_TEETH
  - FS_ETPU_CRANK_FIRST_TRANS
  - FS_ETPU_CRANK_SECOND_TRANS
  - FS_ETPU_CRANK_TEST_POSSIBLE_GAP
  - FS_ETPU_CRANK_VERIFY_GAP
  - FS_ETPU_CRANK_COUNTING
  - FS_ETPU_CRANK_COUNTING_TIMEOUT
  - FS_ETPU_CRANK_TOOTH_BEFORE_GAP
  - FS_ETPU_CRANK_TOOTH_BEFORE_GAP_NOT_HRM

    (only when ERRATTA_2477 is defined)
  - FS_ETPU_CRANK_ADDITIONAL_TOOTH

    (only when a crank wheel with an additional tooth is handled)
  - FS_ETPU_CRANK_TOOTH_AFTER_GAP
  - The eng_pos_state can be one of the following.
    - FS_ETPU_ENG_POS_SEEK
    - FS_ETPU_ENG_POS_FIRST_HALF_SYNC
    - FS_ETPU_ENG_POS_PRE_FULL_SYNC
    - FS_ETPU_ENG_POS_FULL_SYNC
- Eight error conditions are reported.
  - FS_ETPU_CRANK_ERR_INVALID_TRANS - an internal error.
  - FS_ETPU_CRANK_ERR_INVALID_MATCH - an internal error.
  - FS_ETPU_CRANK_ERR_TIMEOUT - a transition was not detected in the expected window.
  - FS_ETPU_CRANK_ERR_STALL - the engine position cannot be handled any more. The synchronization algorithm will be restarted.
  - FS_ETPU_CRANK_ERR_INTERNAL - an internal error.
  - FS_ETPU_CRANK_ERR_TIMEOUT_BEFORE_GAP – a timeout on the last tooth before the gap was detected.

  - FS_ETPU_CRANK_ERR_TIMEOUT_AFTER_GAP – a timeout on the first tooth after the gap was detected.
  - FS_ETPU_CRANK_ERR_TOOTH_IN_GAP - a tooth was detected where the gap was expected.
- A single missing tooth, except the first or the last tooth, is handled without synchronization loss. Error flag FS_ETPU_CRANK_ERR_TIMEOUT is set. A second missing tooth in the same revolution causes a loss of synchronization, the error flag FS_ETPU_CRANK_ERR_STALL is set and the synchronization algorithm is restarted.
- Channel interrupt is generated when:
  - The eng_pos_state is changed
  - During synchronization, on the first tooth, when the Cam log is buffered and ready for recognition.
  - Once per engine cycle, on the first tooth, in full synchronization state (FS_ETPU_ENG_POS_FULL_SYNC).

Figure 5 shows examples of crank wheels, corresponding tooth patterns and their parameterization.

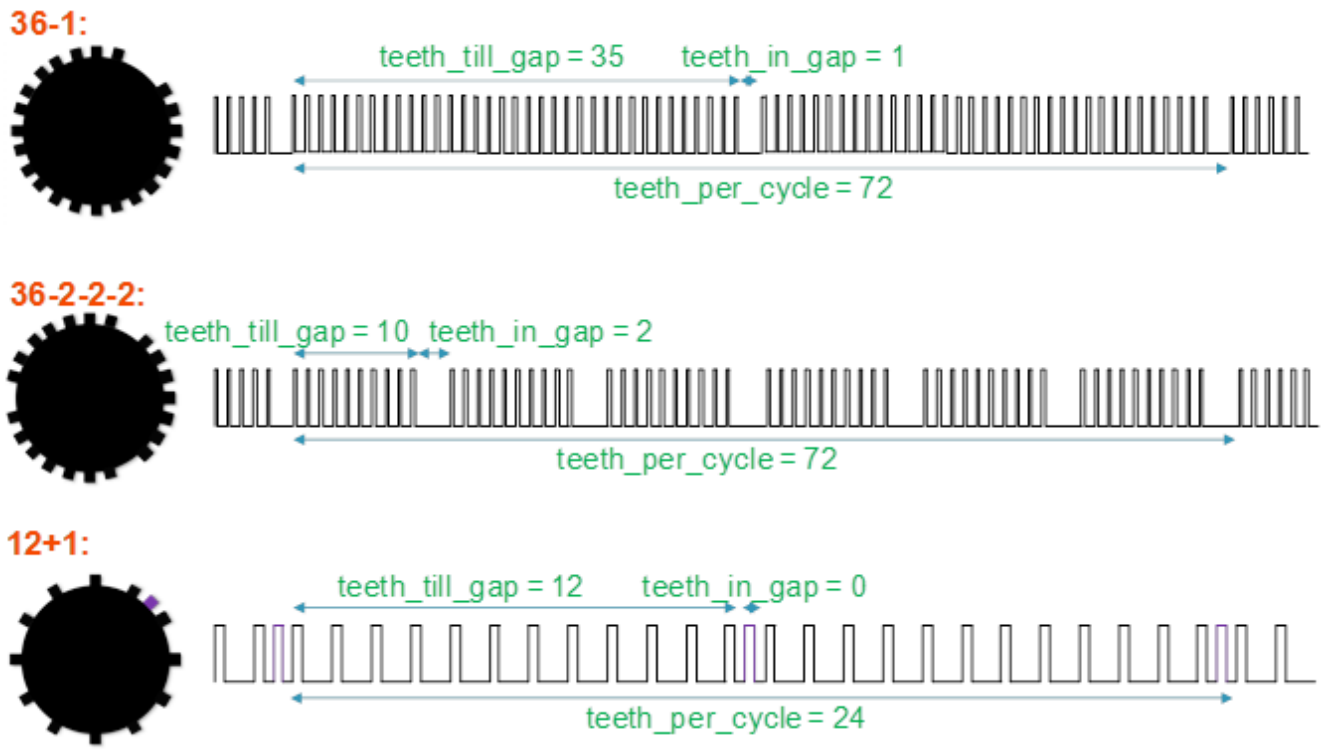

Figure 5. Examples of tooth patterns and their description by parameter values

The following figures explain the acceptance windows used to check the crank input signal (hww is an abbreviation for half window width). A wizard which helps to calculate the Crank gap_ratio and window ratio parameter values based on the maximum engine acceleration is included with the Crank API in excel file, named crank_ratios.xlsx.

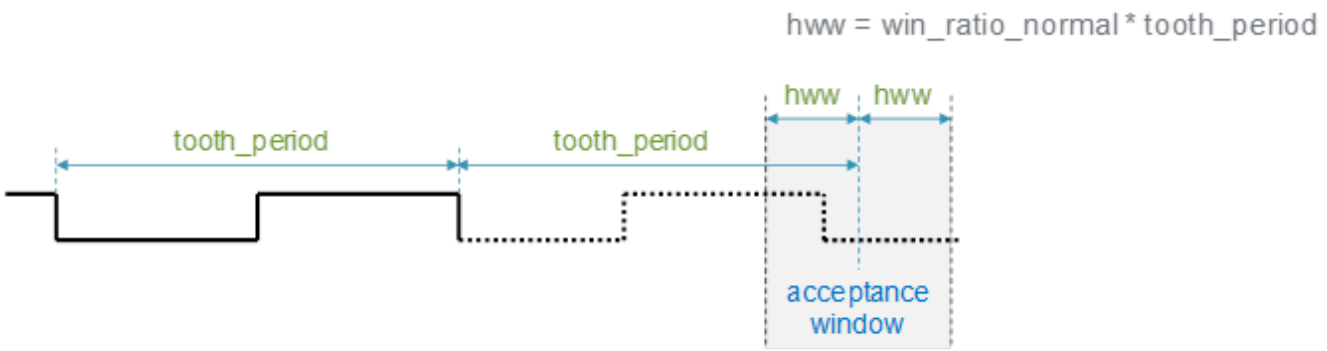

Figure 6. Acceptance window for a normal tooth using win_ratio_normal

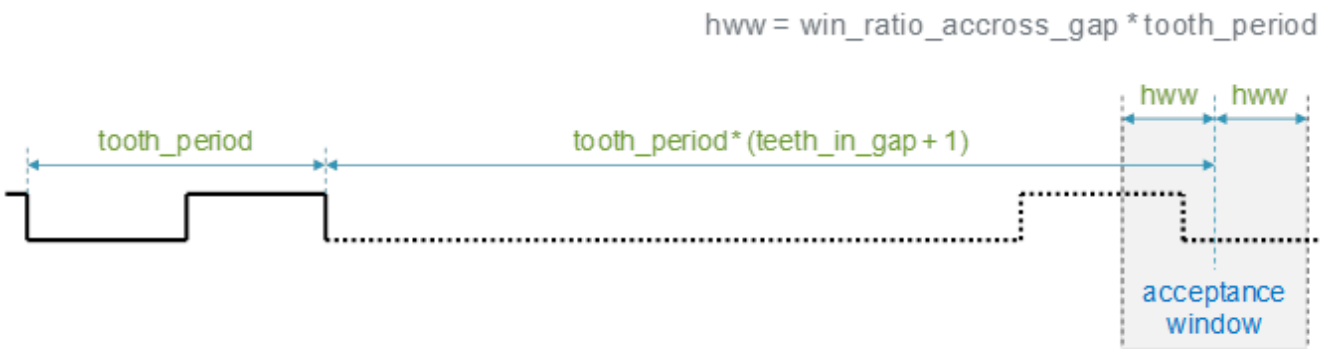

Figure 7. Acceptance window for the first tooth after gap using win_ratio_accross_gap

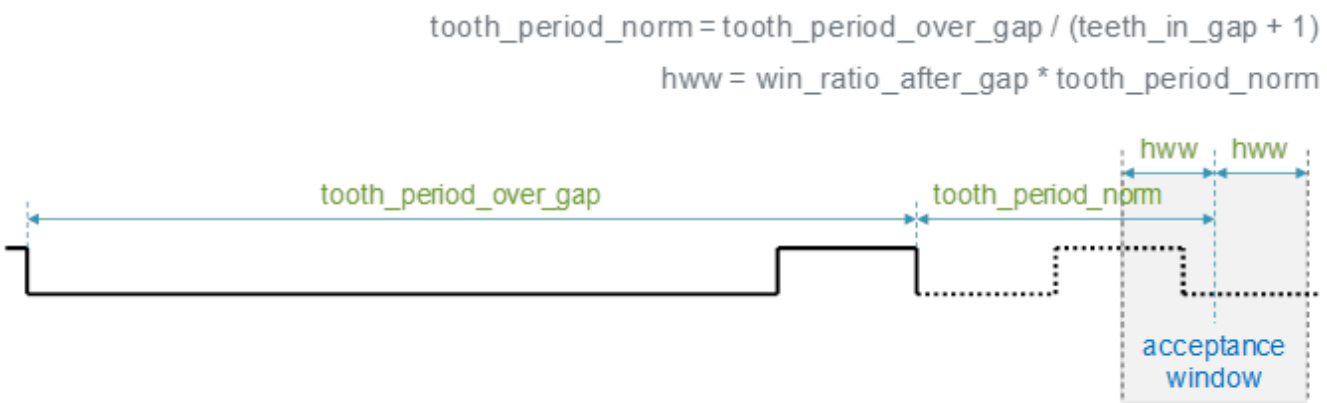

Figure 8. Acceptance window for the second tooth after gap using win_ratio_after_gap

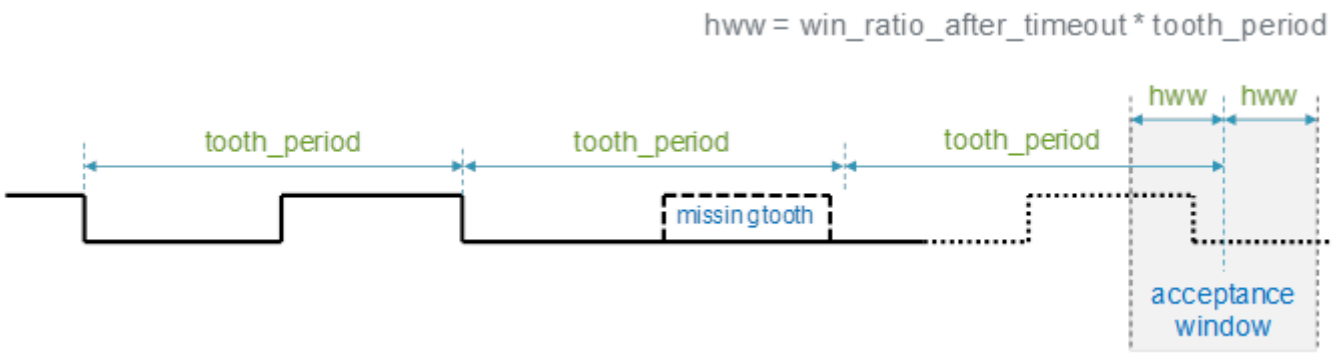

Figure 9. Acceptance window for a tooth after timeout using win_ratio_after_timeout

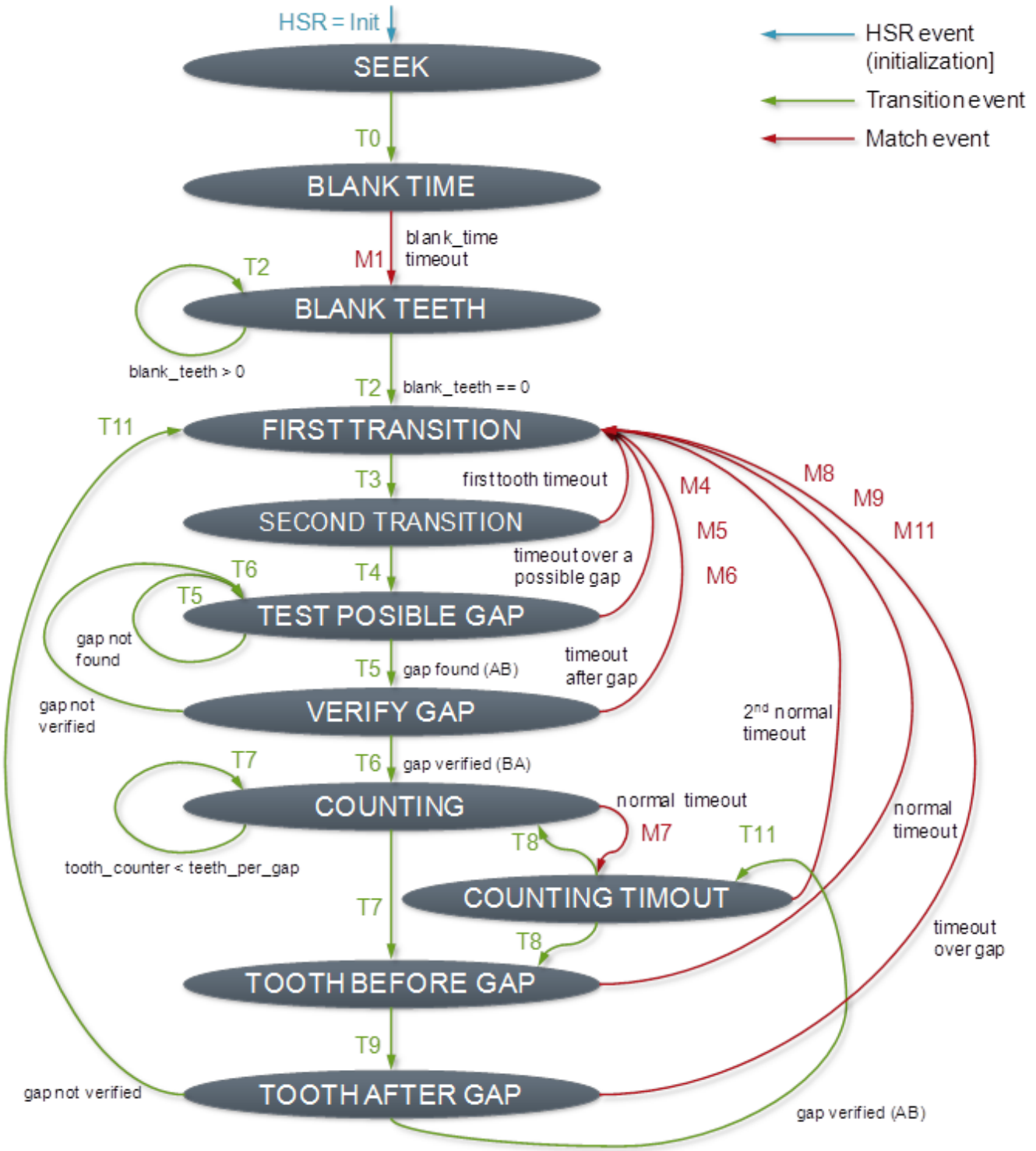

Figure 10. Crank eTPU function state machine

The Crank API is used to control and monitor the eTPU function using the following functions.

- fs_etpu_crank_init()
- fs_etpu_crank_config()
- fs_etpu_crank_get_states()
- fs_etpu_ copy_tooth_period_log()
- fs_etpu_crank_ set_sync()

These API functions handle the following data structures.

- crank_instance
- crank_config
- crank_states

Figure 11 shows Crank API functions, state machine of calls, and data structures.

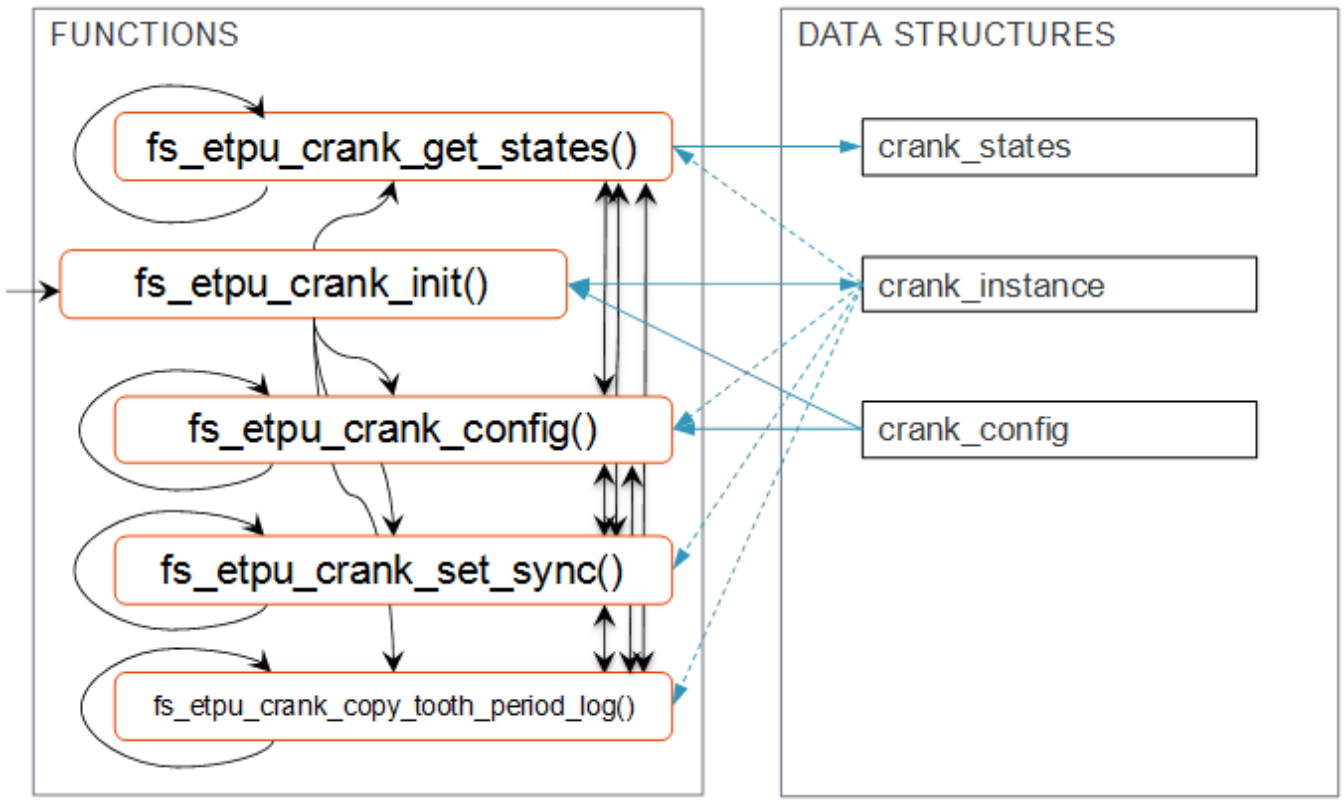

Figure 11. Crank API functions and data structures

The data structures include the following fields.

```
struct crank_instance_t:
       const uint8_t       chan_num
       const uint8_t       priority
       const uint8_t       polarity
       const uint8_t       teeth_till_gap
       const uint8_t       teeth_in_gap
       const uint8_t       teeth_per_cycle
       const uint24_t      tcr2_ticks_per_tooth
       const uint8_t       log_tooth_periods
       const uint32_t      link_cam
       const uint32_t      link_1
       const uint32_t      link_2
       const uint32_t      link_3
       const uint32_t      link_4
             uint32_t  *   cpba
             uint32_t  *   cpba_tooth_period_log

struct crank_config_t:
             uint8_t       teeth_per_sync
             uint24_t      blank_time
             uint8_t       blank_teeth
             ufract24_t    gap_ratio
             ufract24_t    win_ratio_normal
             ufract24_t    win_ratio_across_gap
```

```
              ufract24_t   win_ratio_after_gap
              ufract24_t   win_ratio_after_timeout
              uint24_t     first_tooth_timeout

struct crank_states_t:
              uint8_t     error
              uint8_t     state
              uint8_t     eng_pos_state
              uint8_t     tooth_counter_gap
              uint8_t     tooth_counter_cycle
              uint24_t    last_tooth_period
```

For API description details, refer to CRANK-DoxyDoc.chm or CRANK-DoxyDoc.zip which are included in AN4907SW.

4.1.3 Crank emulator

The Crank Emulator eTPU function (CRANK_EMUL) uses eTPU channel 0 (on eTPU2 optionally channel 1 or 2) to generate internal TCR2 angle-base without processing any input crank signal. For testing purpose, when the Crank signal is not available, the Crank Emulator function can be used to replace the Crank function. Crank emulator drives the internal angle-base at a given speed and consequently enables the injection, ignition, and other output functions to generate outputs.

The Crank emulator API function prototypes and data structures are similar to the Crank API, so that the Crank API can be easily replaced by Crank Emulator API in an application by replacing the header file inclusion. The main difference between Crank Emulator and Crank is that the tooth_period is an input parameter instead of an output. Therefore, the engine speed can be set by the application and the output functions can be tested at known timing conditions.

The Crank Emulator API is used to control and monitor the eTPU function using the following functions.

- fs_etpu_crank_init()
- fs_etpu_crank_config()
- fs_etpu_crank_get_states()
- fs_etpu_ copy_tooth_period_log()
- fs_etpu_crank_ set_sync()
- fs_etpu_crank_set_speed()

The function *fs_etpu_crank_ set_speed()* is unique for Crank Emulator. It sets the *tooth_period* parameter value and starts the angle-base if not started. The other functions are similar to Crank API. Note that the usage of *fs_etpu_crank_config()*, which sets crank signal processing parameters, has no effect when used with Crank Emulator eTPU function.

These API functions handle the following data structures.

- crank_instance
- crank_config
- crank_states

Figure 12 shows Crank Emulator API functions, state machine of calls, and data structures.

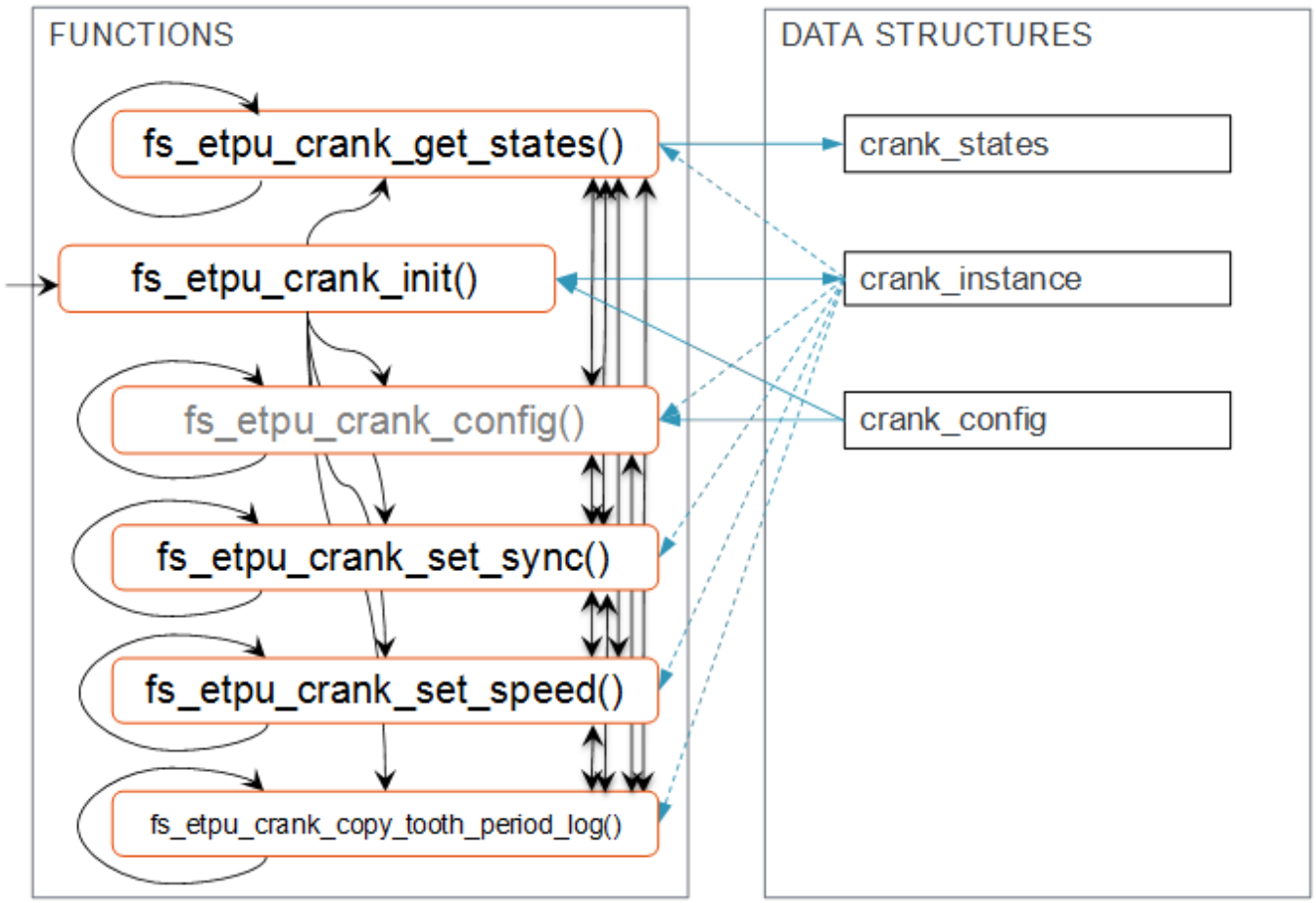

Figure 12. Crank emulator API functions and data structures

The data structures include the following fields. Not all of them are used, the structures are kept similar to Crank API.

```
struct crank_instance_t:
          const uint8_t     chan_num
          const uint8_t     priority
          const uint8_t     polarity
          const uint8_t     teeth_till_gap
          const uint8_t     teeth_in_gap
          const uint8_t     teeth_per_cycle
          const uint24_t    tcr2_ticks_per_tooth
          const  uint8_t    log_tooth_periods
          const uint32_t    link_cam
          const uint32_t    link_1
          const uint32_t    link_2
          const uint32_t    link_3
          const uint32_t    link_4
                uint32_t  * cpba
                uint32_t  * cpba_tooth_period_log

struct crank_config_t:
                uint8_t     teeth_per_sync
                uint24_t    blank_time
                uint8_t     blank_teeth
              ufract24_t    gap_ratio
              ufract24_t    win_ratio_normal
              ufract24_t    win_ratio_across_gap
              ufract24_t    win_ratio_after_gap
              ufract24_t    win_ratio_after_timeout
                uint24_t    first_tooth_timeout
```

```
struct crank_states_t:
                uint8_t     error
                uint8_t     state
                uint8_t     eng_pos_state
                uint8_t     tooth_counter_gap
                uint8_t     tooth_counter_cycle
                uint24_t    last_tooth_period
                uint24_t    last_tooth_period_norm
```

For API description details, refer to CRANK_EMUL-DoxyDoc.chm or CRANK_EMUL-DoxyDoc.zip files, which are included in AN4907SW.

4.1.4 Tooth generator

The Tooth Generator eTPU function (TG) uses two eTPU channels to generate two output signals, which can serve as the crank and cam tooth pattern for testing purpose. An external connection can be used to connect these signals to the cam and crank inputs.

Tooth generator eTPU function has the following features.

- Crank tooth pattern is generated based on a defined number of teeth and missing teeth.
- Cam signal pattern of any number of edges is generated. Cam edge positions are defined by an array of corresponding crank tooth numbers.
- The initial polarity of each output signal is selectable.
- An exponential acceleration/deceleration speed profile can be generated.
- A switch can be used to disable the crank output.
- Tooth counters are available to read.
- Channel interrupt is generated on every gap.

An example of Tooth Generator output signals and corresponding parameter values is shown in Figure 13.

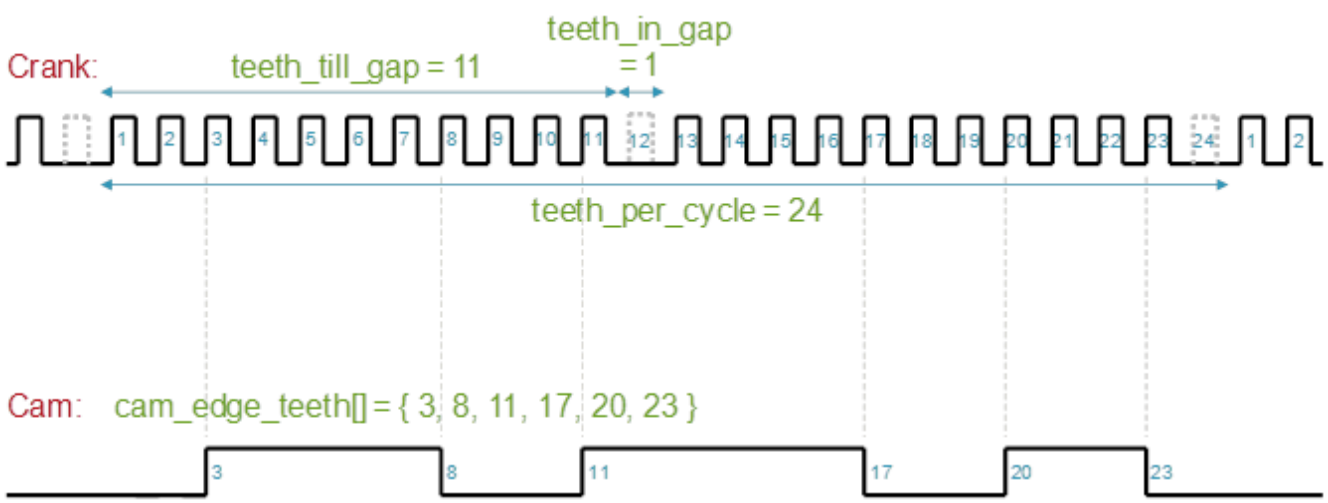

Figure 13. Tooth generator output signals and their parameters

The Tooth Generator eTPU function is able to automatically generate an exponential acceleration/deceleration speed profile. Figure 14 shows how each tooth period is calculated until the target tooth period value is reached.

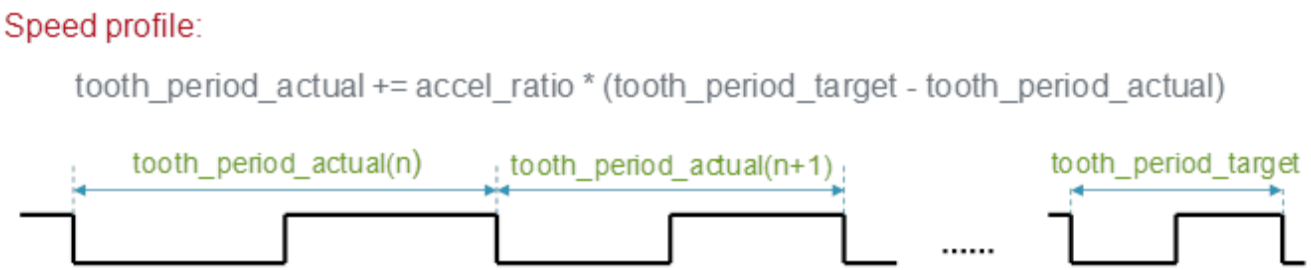

Figure 14. Tooth generator acceleration/deceleration speed profile generation

The Tooth Generator API is used to control and monitor the eTPU function using the following functions.

- fs_etpu_tg_init()
- fs_etpu_tg_config()
- fs_etpu_tg_get_states()

These API functions handle the following data structures.

- tg_instance
- tg_config
- tg_states

Figure 15 shows Tooth Generator API functions, state machine of calls, and data structures.

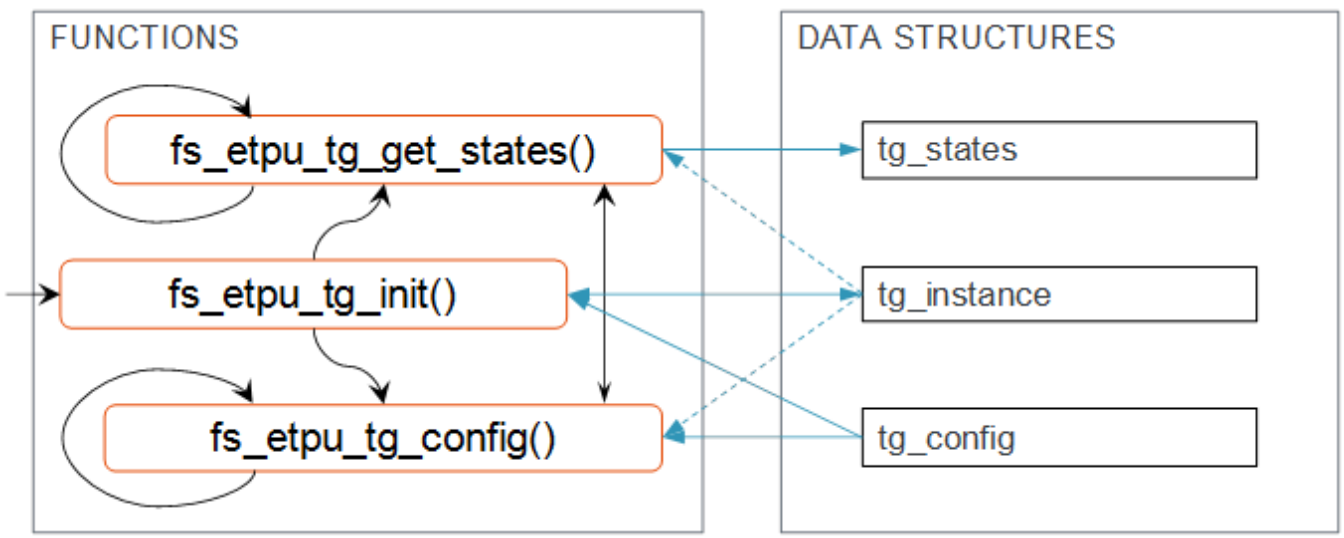

Figure 15. Tooth generator API functions and data structures

The data structures include the following fields.

```
struct tg_instance_t:
       const uint8_t    chan_num_crank
       const uint8_t    chan_num_cam
       const uint8_t    priority
       const uint8_t    polarity_crank
       const uint8_t    polarity_cam
       const uint8_t    teeth_till_gap
       const uint8_t    teeth_in_gap
       const uint8_t    teeth_per_cycle
       const uint8_t    cam_edge_count
       const uint8_t  * p_cam_edge_tooth
             uint32_t * cpba
             uint8_t  * cpba8_cam_edge_tooth
```

```
struct tg_config_t:
              int24_t   tooth_period_target
            fract24_t   accel_ratio
             uint8_t    generation_disable

struct tg_states_t:
             uint8_t    tooth_counter_cycle
              int24_t   tooth_period_actual
```

For API description details, refer to TG-DoxyDoc.chm or TG-DoxyDoc.zip files, which are included in AN4907SW.

4.2 Injection

There are two eTPU functions in the Engine Control eTPU Library related to injection drive. The Fuel function is targeted to port injection engines while the Direct Injection function targets the modern direct injection engines.

4.2.1 Fuel

The Fuel eTPU function (FUEL) uses one eTPU channel to generate an output signal that can drive an injector. Each instance of the FUEL eTPU function controls a single Fuel output.

Fuel has the following features.

- There is a FUEL parameter tdc_angle, relative to which all angles are defined. Positive angles precede the tdc_angle and negative angles come after.
- Fuel output pulse width is injection_time long even if the engine accelerates or decelerates.
- In case of an extreme acceleration, the angle_stop limits the injection pulse length.
- The CPU can update the amount of injected fuel anytime using the function fs_etpu_fuel_update_injection_time(), which not only sets the injection_time value for the next engine cycles, but also updates the current injection - shorts the pulse, extends the pulse or generates an additional pulse.
- In order to
  - Immediately disable output generation, set injection_time to 0.
  - Disable the injection generation from the next cycle, but finish the running injection pulse, use generation_disable configuration flag.
- Two error conditions are reported
  - FS_ETPU_FUEL_ERROR_STOP_ANGLE_APPLIED - a fuel injection pulse has been stopped and shortened by the stop_angle. Hence, the commanded injection_time and the injection_time_applied may differ.
  - FS_ETPU_FUEL_ERROR_MINIMUM_INJ_TIME_APPLIED - a fuel injection pulse, the main or an additional one, is shorter than the injection_time_minimum and hence not generated, skipped. The commanded injection_time and the injection_time_applied may differ.
- Channel interrupt is generated once every engine cycle, at the angle_stop.

Figure 16 shows the Fuel output signal and the signal parameters.

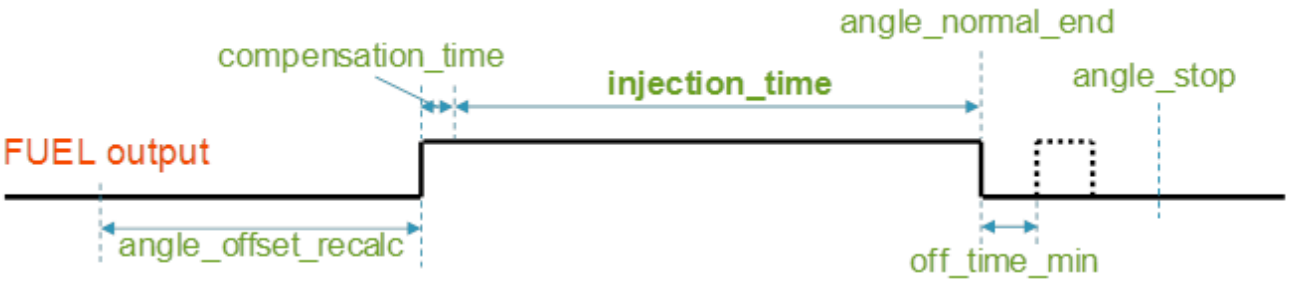

Figure 16. Fuel output signal and parameters

The Fuel API is used to control and monitor the eTPU function using the following functions.

- fs_etpu_fuel_init()
- fs_etpu_fuel_config()
- fs_etpu_fuel_get_states()
- fs_etpu_fuel_update_injection_time ()

These API functions handle the following data structures.

- fuel_instance
- fuel_config
- fuel_states

Figure 17 shows Fuel API functions, state machine of calls, and data structures.

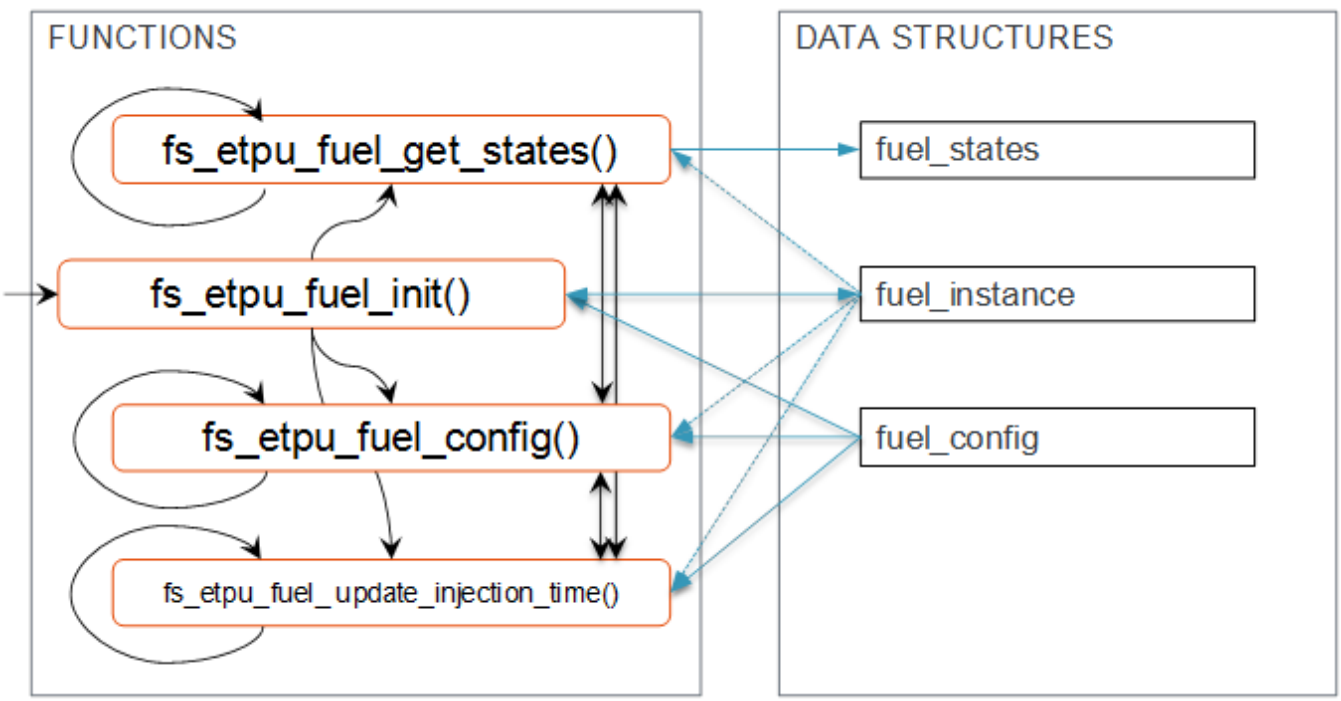

Figure 17. Fuel API functions and data structures

The data structures include the following fields.

```
struct fuel_instance_t:
        const uint8_t    chan_num
        const uint8_t    priority
        const uint8_t    polarity
        const uint24_t   tdc_angle
```

```
              uint32_t * cpba

struct fuel_config_t:
               int24_t   angle_normal_end
               int24_t   angle_stop
               int24_t   angle_offset_recalc
              uint24_t   injection_time
              uint24_t   compensation_time
              uint24_t   injection_time_minimum
              uint24_t   off_time_minimum
              uint8_t    generation_disable

struct fuel_states_t:
              uint8_t    error
              uint24_t   injection_time_applied
               int24_t   injection_start_angle
```

For API description details, refer to FUEL-DoxyDoc.chm or FUEL-DoxyDoc.zip files, which are included in AN4907SW.

4.2.2 Direct injection

The Direct Injection (INJ) eTPU function is able to generate complex angle-based and time-based output patterns, primarily dedicated to direct injection timing control. The function enables to generate a pattern of synchronous angle or time based output pulses on several output channels, including common bank channels and individual injector channels. For a closed loop solenoid current control, this function can provide the timing for either external (MC33816) or internal (Reaction Channel) closed loop logic.

Each instance of the INJ eTPU function controls a single INJ channel together with up to three BANK channels (For example, Boost and Batt). The BANK channels can be common for more INJ instances.

Direct injection has the following features.

- The INJ eTPU function is able to generate an injection sequence each engine cycle.
- An injection sequence consists of injections. The number of injections in the sequence is configurable and can be reconfigured every engine cycle.
- Each injection starts at a defined angle (angle_start) and consists of phases. The number of phases is configurable and can be reconfigured every engine cycle.
- Each injection phase is defined by output states of the INJ and all the BANK channels, a phase duration, and options to generate DMA requests at the beginning of the phase.
- More INJ channels (individual injectors) may use separate injection sequences, or can share the same injection sequence definition array.
- There is an INJ parameter tdc_angle, relative to which all angles are defined. Positive angles precede the tdc_angle, and negative angles come after.
- INJ parameter angle_irq defines an angle at which an IRQ request is generated. The CPU may reconfigure the injection sequence setting on this interrupt, but not later than the first injection angle_start is reached. If the CPU does not provide new new data, the last injection sequence definition is used.
- INJ parameter angle_stop defines the latest angle when the whole injection sequence must be finished. In case of an extreme acceleration, the INJ channel and all the BANK channel outputs are turned to inactive state at the angle_stop, whatever injection phase is active.
- Four error conditions are reported.
    - FS_ETPU_INJ_ERROR_PREV_INJ_NOT_FINISHED - injection sequence cannot start while another INJ channel occupies the BANK channels. The injection sequence was not generated.
    - FS_ETPU_INJ_ERROR_LATE_START_ANGLE_1ST - the first injection angle_start was about to be scheduled in past, hence the whole injection sequence was skipped.

- FS_ETPU_INJ_ERROR_LATE_START_ANGLE_NTH - the second or later injection angle_start was about to be scheduled in past, hence the rest of the injection sequence was skipped.
- FS_ETPU_INJ_ERROR_STOPPED_BY_STOP_ANGLE - the injection sequence was not finished before the angle_stop and hence the injection was hard-stopped at the angle_stop.

- Channel interrupt is generated before each injection sequence, at the angle_irq.

Figure 18 shows an example of the generated signals and their correspondence to a solenoid injector current profile.

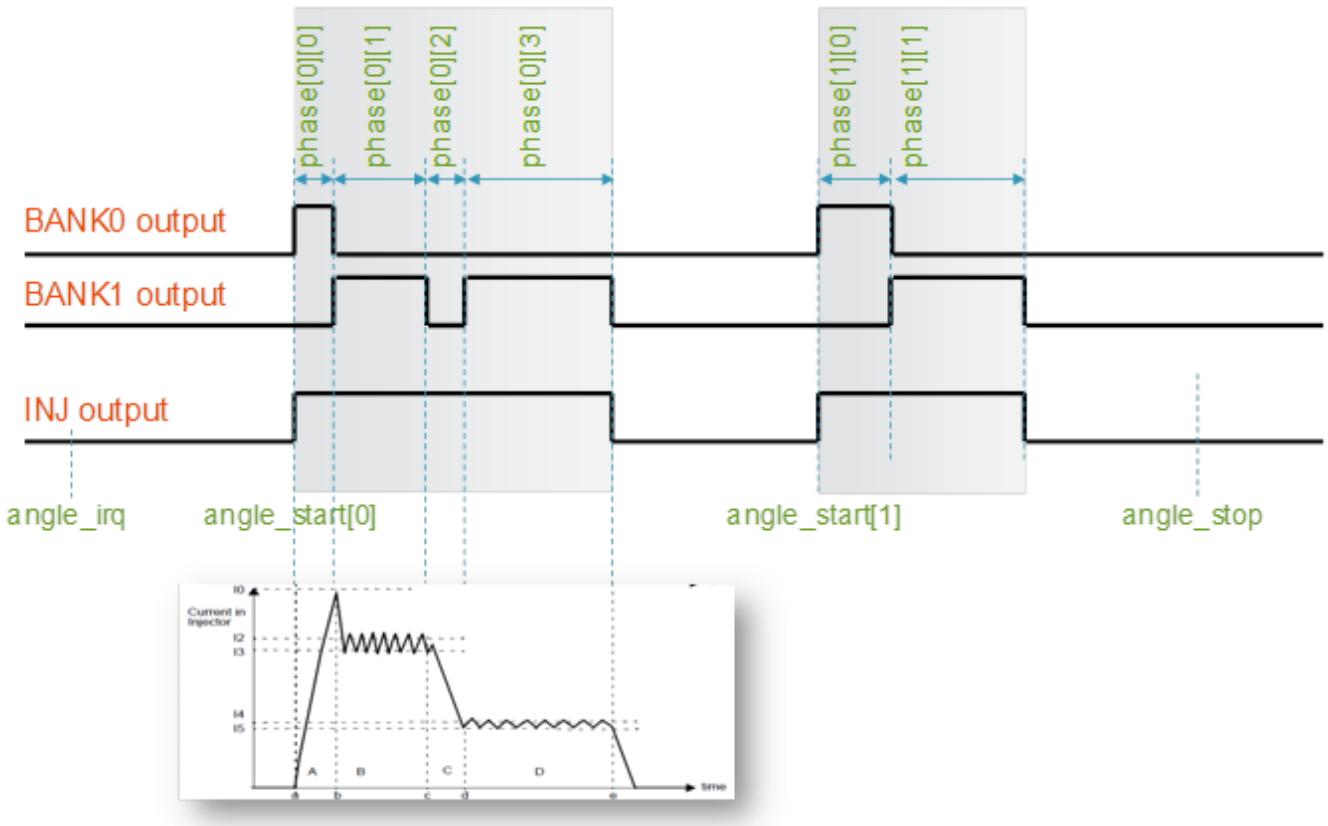

Figure 18. Direct injection output signals – an injection sequence consisting of two injections

The Direct injection API is used to control and monitor the eTPU function using the following functions.

- fs_etpu_inj_init()
- fs_etpu_inj_config()
- fs_etpu_ inj_get_states()

These API functions handle the following data structures.

- inj_instance
- inj_config
- inj_injection_config
- inj_states

Figure 19 shows direct injection API functions, state machine of calls, and data structures.

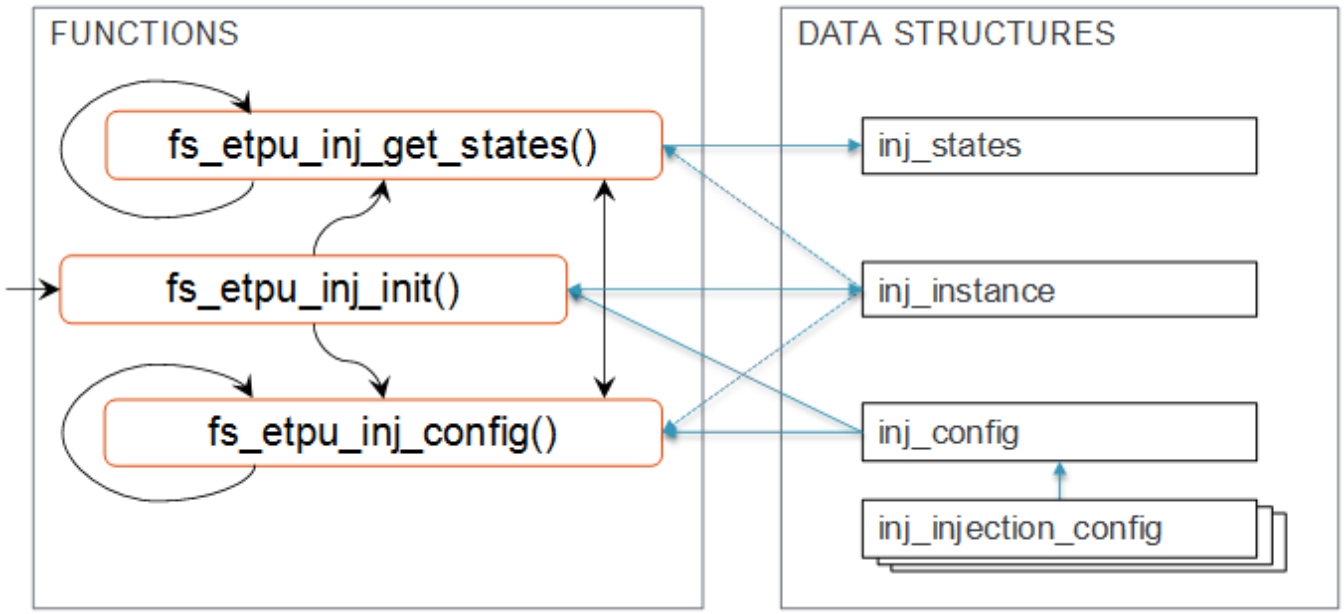

Figure 19. Direct injection API functions and data structures

The data structures include the following fields.

```
struct inj_instance_t:
       const uint8_t     chan_num
       const uint8_t     priority
       const uint8_t     polarity
       const uint24_t    tdc_angle
             uint32_t  * cpba
             uint32_t  * cpba_single_spark

struct inj_config_t:
              int24_t    angle_irq
              int24_t    angle_stop
             uint8_t     injection_count
       struct inj_injection_config_t *p_injection_config

struct inj_injection_config_t:
              int24_t    angle_start
             uint8_t     phase_count
             uint32_t  * p_phase_config

struct inj_states_t:
             uint8_t     error
             uint8_t     injection_idx
             uint8_t     phase_idx
```

For API description details, refer to INJ-DoxyDoc.chm or INJ-DoxyDoc.zip files, which are included in AN4907SW.

4.3 Ignition

There is one eTPU function in the Engine Control library targeted to drive ignitions.

4.3.1 Spark

The Spark eTPU function (SPARK) uses one eTPU channel to generate an output signal, which can drive an ignitor. Each instance of the SPARK eTPU function controls a single Spark output.

Spark has the following features.

- There is a SPARK parameter tdc_angle, relative to which all angles are defined. Positive angles precede the tdc_angle and negative angles come after.
- The SPARK eTPU function generates one or more spark output pulses per engine cycle.
- The number of sparks per engine cycle are configurable. The individual sparks are defined by an array of single spark structures. The single spark structure defines one spark output pulse.
- Each spark output pulse includes the main pulse, defined by end_angle and dwell_time, and limited by min_dwell_time and max_dwell_time. The main pulse is optionally followed by a sequence of multi-pulses, defined by multi_on_time, multi_off_time, and pulse_count.
- The Spark pulse ends at end_angle even if the engine accelerates or decelerates.
- In case of an extreme acceleration or deceleration, the dwell_time is limited by dwell_time_min or dwell_time_max.
- In order to disable the spark output generation, use generation_disable configuration flag.
- Two error conditions are reported.
  - FS_ETPU_SPARK_ERROR_MIN_DWELL_APPLIED - the spark main pulse has been limited by min_dwell_time and the pulse ended at a later angle than the eng_angle. Hence, the commanded dwell_time and the dwell_time_applied may differ.
  - FS_ETPU_SPARK_ERROR_MAX_DWELL_APPLIED - the spark main pulse has been limited by max_dwell_time and the pulse ended sooner than at the end_angle. Hence, the value of the output parameter dwell_time_applied have a different value than the value of the input parameter dwell_time.
- Channel interrupt is generated before each single spark, on the recalc_angle.

Figure 20 shows the spark output signal and the signal parameters.

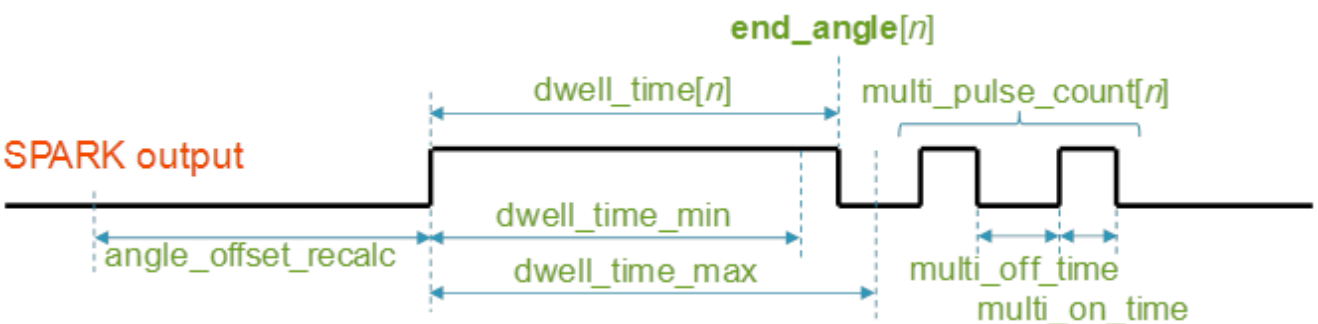

Figure 20. Spark output signal and parameters

The Spark API is used to control and monitor the eTPU function using the following functions.

- fs_etpu_spark_init()
- fs_etpu_spark_config()
- fs_etpu_ spark_get_states()

These API functions handle the following data structures.

- Spark_instance
- Spark_config
- Single_spark_config
- Spark_states

Figure 21 shows Spark API functions, state machine of calls, and data structures.

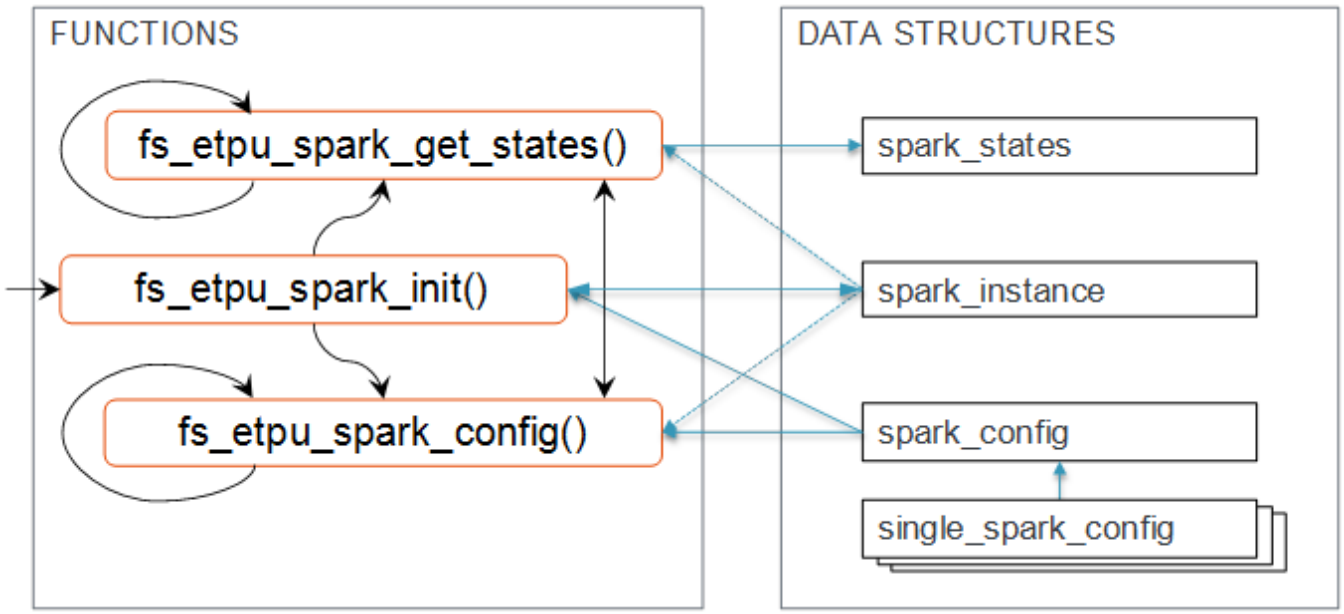

Figure 21. Spark API functions and data structures

The data structures include the following fields.

```
struct spark_instance_t:
          const uint8_t     chan_num
          const uint8_t     priority
          const uint8_t     polarity
          const uint24_t    tdc_angle
                uint32_t  * cpba
                uint32_t  * cpba_single_spark

struct spark_config_t:
                 int24_t   angle_offset_recalc
                uint24_t   dwell_time_min
                uint24_t   dwell_time_max
                uint24_t   multi_on_time
                uint24_t   multi_off_time
                uint8_t    spark_count
         struct single_spark_config_t    *p_single_spark_config
                uint8_t    generation_disable

struct single_spark_config_t:
                 int24_t   end_angle
                uint24_t   dwell_time
                uint8_t    multi_pulse_count

struct spark_states_t:
                uint8_t    error
                uint24_t   dwell_time_applied
```

For API description details, refer to SPARK-DoxyDoc.chm or SPARK-DoxyDoc.zip files, which are included in AN4907SW.

4.4 Other

This section describes the eTPU Engine Control library, which includes supportive functions not directly dedicated to the basic engine drive. There is one function included, the Knock. Other than this, all the standard eTPU functions, for example, PWM can be used together with the engine control functions, but these functions are not released as part of the Engine Control eTPU Library.

4.4.1 Knock

The Knock eTPU function (KNOCK) supports A/D conversion of a knock-detection signal. Each instance of the KNOCK eTPU function uses one eTPU channel and controls a single Knock output.

The Knock eTPU function has the following features.

- There is a KNOCK parameter tdc_angle, relative to which all angles are defined. Positive angles precede the tdc_angle and negative angles come after.
- The KNOCK eTPU function enables to generate one or more angular windows per engine cycle.
- The number of windows per engine cycle is configurable. The windows are defined by an array of window structures, including a TDC relative angle_start and angle_width.
- The KNOCK eTPU function can operate in one of the following modes.
  - Gate mode (FS_ETPU_KNOCK_FM1_MODE_GATE)

    In the Gate mode, simple angle-based pulses (angle-windows) are generated. The output signal can be used to gate the ADC running in continuous mode.

  - Trigger mode (FS_ETPU_KNOCK_FM1_MODE_TRIGGER)

    In the Trigger mode a 50% duty-cycle PWM signal is generated within the angle window. The output signal can be used to trigger the ADC

The KNOCK function enables to selectively generate channel interrupts and/or DMA requests at:

- Window start
- Window end
- Every trigger pulse (Trigger mode only)

Figure 22 shows the Knock output signal in both modes and signal parameters.

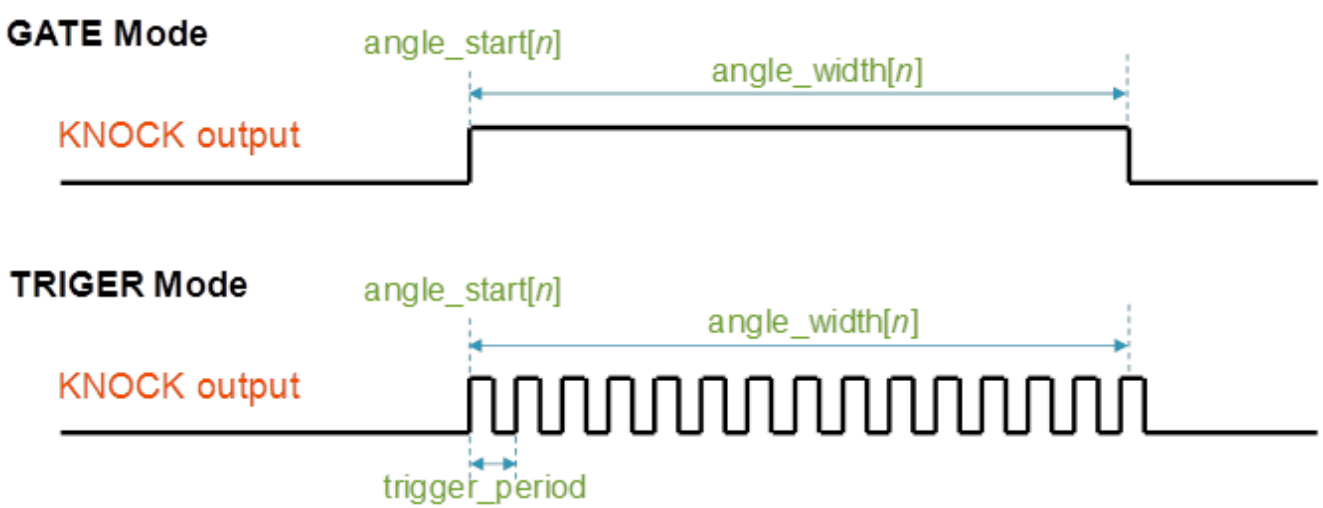

Figure 22. Knock output signal and parameters

The Knock API is used to control the eTPU function using the following functions.

- fs_etpu_knock_init()
- fs_etpu_knock_config()

These API functions handle the following data structures.

- knock_instance
- knock_config
- knock_window_config

Figure 23 shows Knock API functions, state machine of calls, and data structures.

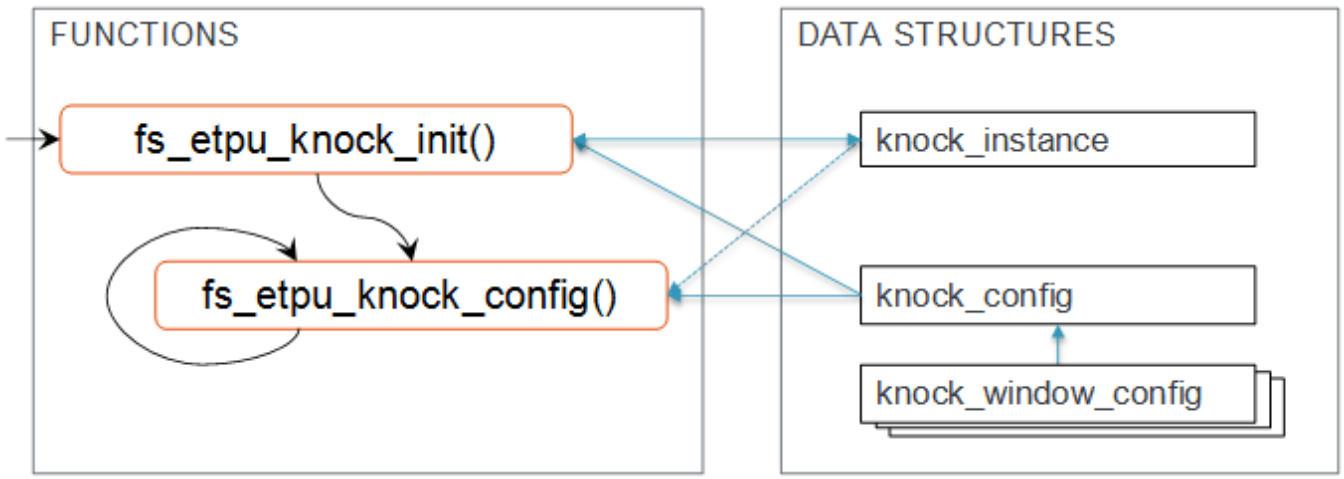

Figure 23. Knock API functions and data structures

The data structures include the following fields.

```
struct knock_instance_t:
       const uint8_t     chan_num
       const uint8_t     priority
       const uint8_t     polarity
       const uint24_t    tdc_angle
             uint32_t  * cpba
             uint32_t  * cpba_windows

struct knock_config_t:
             uint8_t     mode
             uint8_t     window_count
      struct knock_window_config_t     *p_knock_window_config
              int24_t    trigger_period
             uint8_t     irq_dma_options

struct knock_window_config_t:
              int24_t    angle_start
              int24_t    angle_width
```

For API description details, refer to KNOCK-DoxyDoc.chm or KNOCK-DoxyDoc.zip files, which are included in AN4907SW.

5 Summary

The new Engine Control eTPU Library enables the eTPU to drive engine control applications. This library represents a major step forward compared to its predecessor – eTPU Automotive Function Set (set2). In the updated library, there are new eTPU functions, which are able to drive modern direct injection engines, and the other functions are widely updated and upgraded, including new features, simplifying the usage and introducing much clear code. The new unified application interface (API) style enables to easily use the eTPU functions from the application.

6 References

The following references are available at freescale.com.

1. *Engine Control eTPU Demo Application* (document AN4908)
2. *Enhanced Time Processing Unit Reference Manual* (document ETPURM)
3. *eTPU2 Reference Manual Addendum* (document ETPU2RMAD)
4. *eTPU Automotive Function Set (Set2)* (document AN3768)
5. *General C Functions for the eTPU* (document AN2864)

7 Revision history

This section documents the changes done to this document.

Table 3. Revision history

Rev. number	Substantial changes
0	Initial release
2	Added one line of code to Crank emulator

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number AN4907
Revision 2, 01/2016