AN2577
Application note

Using the eTPU angle clock

Introduction

This document describes the angle clock feature of the enhanced time processor unit and offers practical advise on the application and use of this feature. The note assumes a familiarity with the eTPU and the eTPU_C language used in the byte craft compiler.

The eTPU is particularly suited to applications where input and output actions are synchronized to the position of a rotating wheel. The eTPU angle clock provides hardware which, when combined with the proper software, can track the wheel angle through accelerations, decelerations, and a wide range of speeds.

If a wheel is tracked by the eTPU angle clock, many output matches and input captures can operate on an angle count as well as a time count, and even on combinations of the two. Compensation for speed changes are made in the angle clock only. An angle triggered event will occur on the best estimate of the match angle.

The angle clock is complex to set up, but when running it offers an excellent integration of fast operating hardware with the complex control of a software program. A reference design has been built to demonstrate the operation of a periodic angle clock with other engine control functions. See your STMicroelectronics representative for details.

Contents

1 Overview . . . 4

2 eTPU angle clock hardware . . . 6

2.1 Tooth wheel input . . . 8

2.2 Tick counter . . . 8

2.3 Missing teeth . . . 9

2.4 Zero count synchronization . . . 10

2.5 Error recovery . . . 10

3 System design . . . 12

3.1 Steady state system design . . . 12

3.1.1 Periodic angle clock . . . 12

3.1.2 Free running angle clock . . . 13

3.1.3 Angle resolution . . . 13

3.1.4 Reference point . . . 14

3.1.5 Missing teeth . . . 14

3.1.6 Tick rate . . . 14

3.2 Acceleration and deceleration . . . 14

3.2.1 Acceleration . . . 15

3.2.2 Deceleration . . . 16

3.3 Startup system design . . . 17

3.3.1 Initial parameter values . . . 17

3.3.2 Acquisition and synchronization . . . 17

3.3.3 Starting the angle clock . . . 18

3.4 Anomalies . . . 19

3.4.1 Error prevention . . . 19

3.4.2 Error recovery . . . 20

Inserting a tooth . . . 20

Deleting a tooth . . . 20

3.5 Tooth circuit failure . . . 21

4 eTPU software design . . . 22

4.1 Host setup . . . 22

4.2 State machine description 22
4.2.1 Initialize crank 24
4.2.2 First edge 25
4.2.3 First period 26
4.2.4 Testing for a possible gap 26
4.2.5 Apparent gap 27
4.2.6 Possible gap verifying 27
4.2.7 Gap verified 28
4.2.8 Counting 28
4.2.9 Gapping 29
4.2.10 New revolution 30
4.2.11 Adjust angle 30
4.2.12 Stall 30

5 Angle clock application 31
5.1 Free running angle counter 31
5.2 Periodic angle clock 31
5.3 Angle math 32

6 Revision history 33

1 Overview

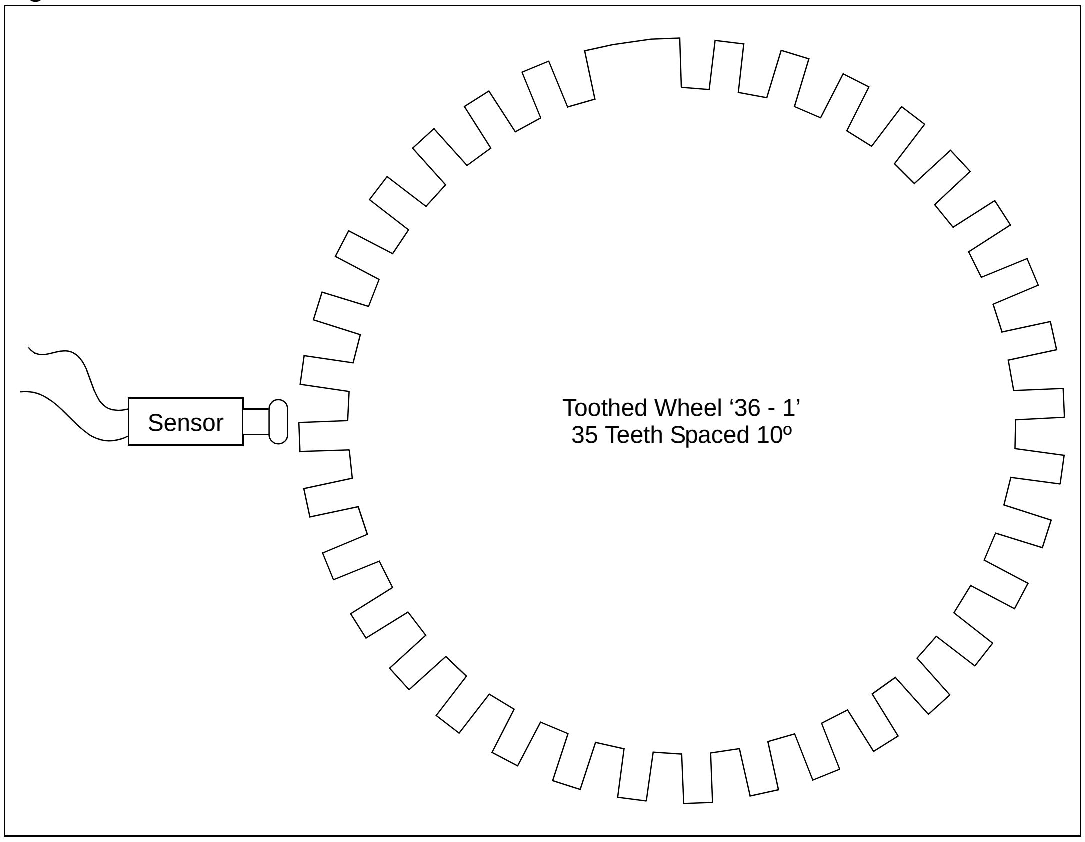
The enhanced time processor unit (eTPU) is an autonomous slave processor offered on several STMicroelectronics microcontrollers designed for real time automotive applications.The processor is tightly coupled to up to 32 channels, each associated with an input and an output signal. The input/output channels each have a pair of Match and Capture units interfaced to one of two timer/counter (TCR) registers. Logic in the channel enabled the hardware to detect or drive pin transitions with a high degree of timing precision.

In a typical application, at least one of the TCRs is driven by a real time clock derived from the MCU clock frequency. The second TCR can be driven by an asynchronous external signal, by a real time clock with a different time base, or by special angle clock circuitry on the eTPU. The purpose of the angle clock is to synchronize with the angle of a spinning shaft and provide a counter representation of the instantaneous position of the shaft. If the shaft provides an input signal related to teeth spaced along the circumference, the angle clock extrapolates the number of ticks between the adjacent pairs of teeth, thereby providing a finer resolution of angle than the shaft signal alone can provide. The angle clock was designed for automotive engine control, but finds application in a variety of rotational devices.

An angle clock can be implemented in hardware or in software, but either implementation has limitations. A software angle clock cannot divide an input tooth signal into very fine ticks without overwhelming the processor. Even before the tick rate reaches its limit, latency on the processor will begin to distort the ideal count pattern. A hardware angle clock can provide very fine resolution of the ticks, but the hardware becomes complex and large when the logic is required to handle errors and exceptional input signals.

The eTPU angle clock has been implemented as a software supported hardware subsystem, effectively eliminating both of these significant limitations. The hardware subsystem has provisions for fast tick counting plus compensation for missing teeth, which are often used to provide an angle key for the shaft. The hardware also provides automatic compensation for acceleration and deceleration, and assists in error recovery. The software handles synchronization and exception handling.

Figure 1. Toothed wheel



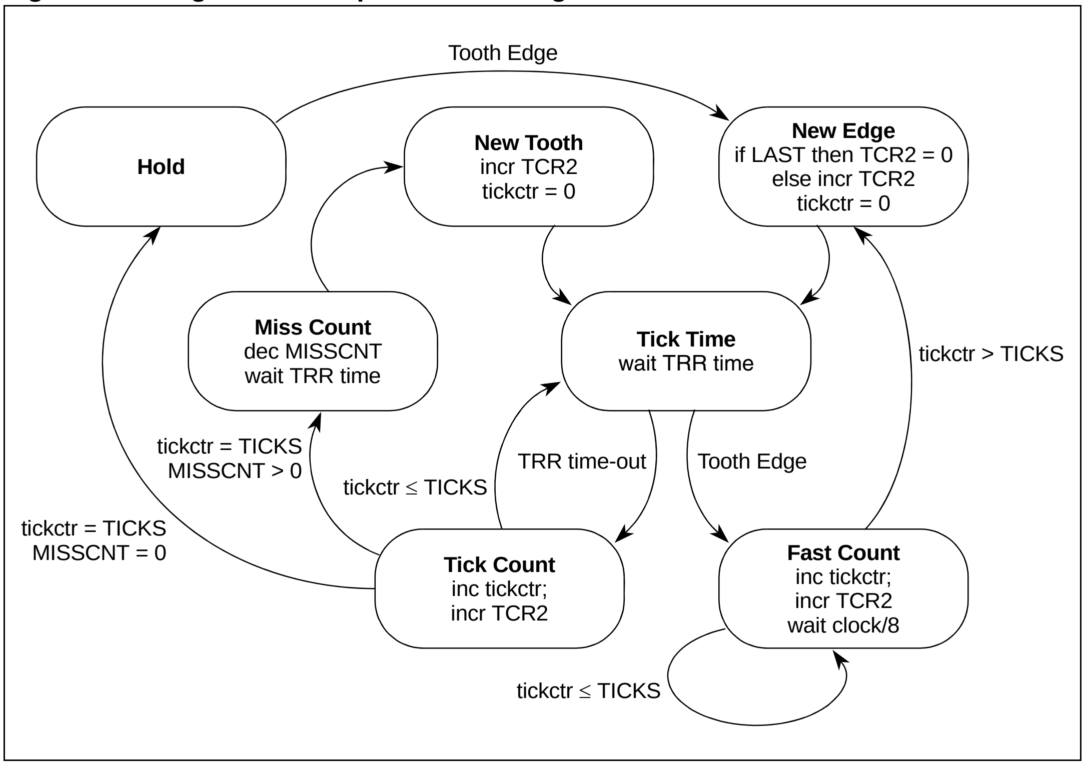
2 eTPU angle clock hardware

The eTPU angle clock provides a counter that can be synchronized to a periodic input signal such as would be produced by a sensor detecting teeth on a wheel attached to a rotating shaft. A typical toothed wheel used in automotive applications might have 35 teeth spaced at 10 degree intervals with a missing tooth in the 36th position (*Figure 1*). The angle clock hardware detects and times the teeth, compensates for the missing teeth, and inserts a fixed number of additional counts (ticks) between each pair of teeth at a rate determined by eTPU software. Typically, a system would insert 100 counts between each pair of teeth, giving an angle resolution of 0.1 degree.

The angle clock is maintained in TCR2, one of two selectable timer counter registers in the eTPU, which can be used by any channel to cause pin action and/or service requests on a match, or which can be captured as a result of an input pin action. In addition, the eTPU can export the angle clock to another eTPU or to an eMIOS device connected to a common shared timer and counter (STAC) bus. The angle clock can be read in real time by the host.

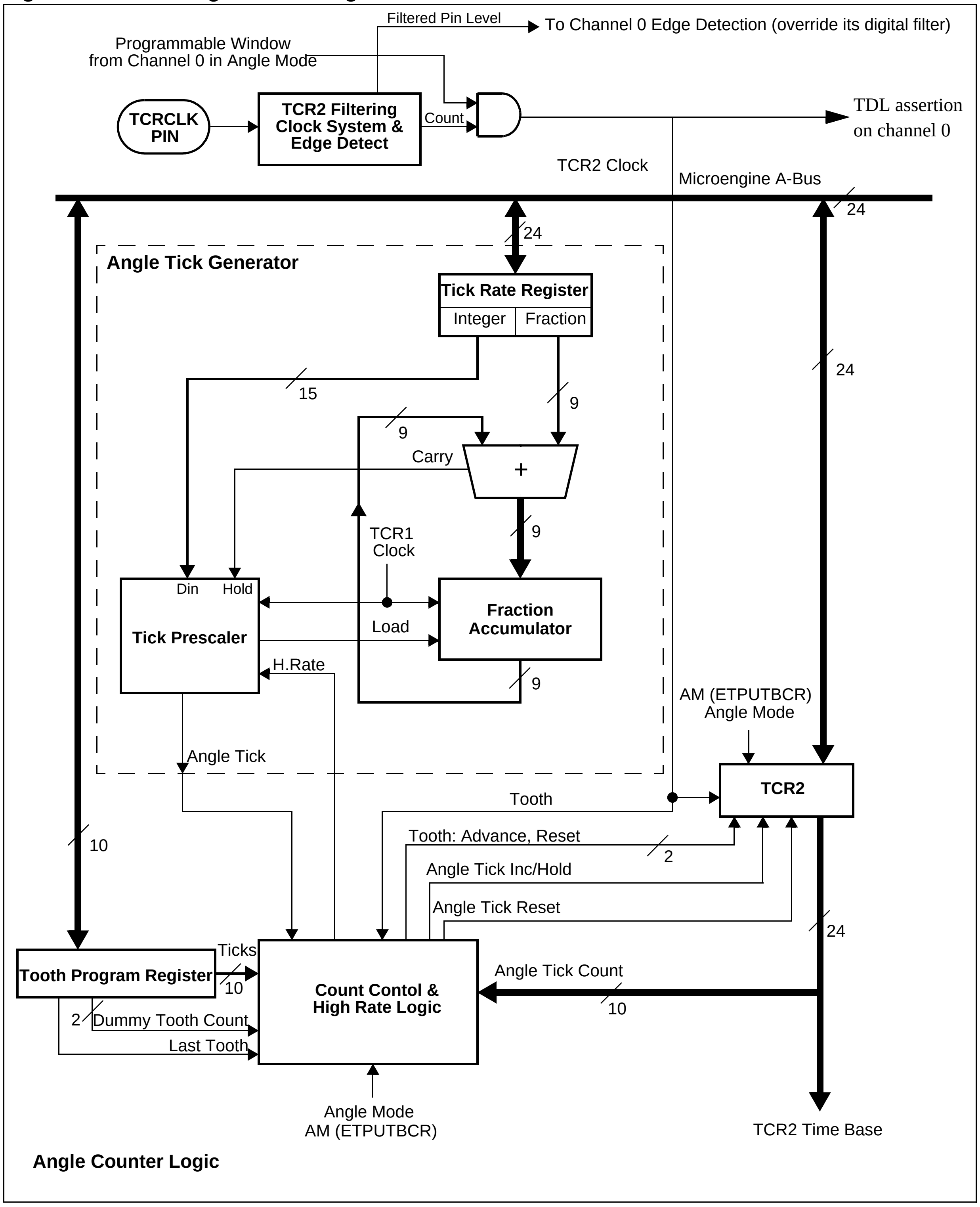
The eTPU angle clock operates as a state machine, processing inputs from the input pin and using parameters provided by the eTPU software. A simplified diagram of the eTPU angle clock logic is shown in *Figure 2*.

Figure 2. Angle clock simplified state diagram



The heart of the angle clock is a tick generator that inserts a selected number of ticks between each pair of teeth. A block diagram of the tick generator is shown in *Figure 3*.

Figure 3. eTPU angle clock tick generator



2.1 Tooth wheel input

When an eTPU is placed into angle clock mode by programming the AM bit in the register ETPUTBCR, the input to Channel 0 of the device is connected internally to the external clock input TCRCLK. This input has an independently programmed hardware filter circuit which enables the user to select a different input filter treatment from the rest of the eTPU input pins if desired. The filtered input signal is applied to the angle clock hardware circuitry. This hardware is part of channel 0 input and the signal to the angle clock can be conditioned by matches according to the selected mode for the channel. For example, the channel can be placed in *m2_st (Match2SingleTransition*) mode, and the eTPU software can establish a window for accepting a tooth signal from the wheel. A signal arriving too early might indicate noise and could be ignored, while one arriving too late could indicate a stall and might require special exception processing.

2.2 Tick counter

When the angle clock is running on channel 0 of an eTPU, an input signal detected by the channel increments TCR2. When this count occurs, a tick counter is started in the angle clock circuitry. This counter increments TCR2 at a rate set by the value in the tick rate register (TRR), which is programmed in terms of TCR1 cycles. Thus if the TRR were written with a value of 100, and the TCR1 count rate was set at 0.1 microsecond, the TCR2 counter would increment once every 10 microseconds.

The TRR is written by eTPU software from a value derived from the projected speed of the wheel. If the application requires 200 ticks per tooth, the eTPU software will typically measure the period of the last tooth pair in terms of TCR1, and divide the value by 200 to get the TRR value. The TRR can be programmed with a whole number and fraction of a count. As the count proceeds, the angle clock logic accumulates the fraction, and adjusts the count to compensate for the fraction. The software also programs the number of inserted counts into the field TICKS in the tooth program register (TPR). If there are 100 ticks per tooth period, then TICKS would be programmed with 99.

The tick counter is designed to count exactly TICKS+1 periods between detected physical teeth. Under constant speed operation, TICKS +1 times the TRR rate will be exactly equal to the period between the teeth. When the following tooth signal arrives late, the angle clock stops after incrementing the TCR2 by TICKS counts. Thus the angle clock is said to be in Halt mode while the slowing wheel catches up with the angle clock count. When the next tooth arrives, it is counted in TCR2 and the angle clock reverts to Normal mode with tick counts added for the following tooth period. Thus the angle clock cannot proceed to the next tooth count until the physical tooth is detected.

If the wheel is accelerating, the tooth signal may be detected before the TICKS count has completed. In this event, the angle clock hardware is put into a high rate count mode. The tick count completes at an accelerated rate equal to 1/8 of the MCU system clock (regardless of the TCR1 rate). This allows the angle clock to catch up with the physical teeth. After the last tick is counted, the angle clock will count the newly arrived tooth at the same rate. The high rate has been chosen to ensure that any eTPU or external action triggered by a specific value of the angle clock can be detected and acted upon.

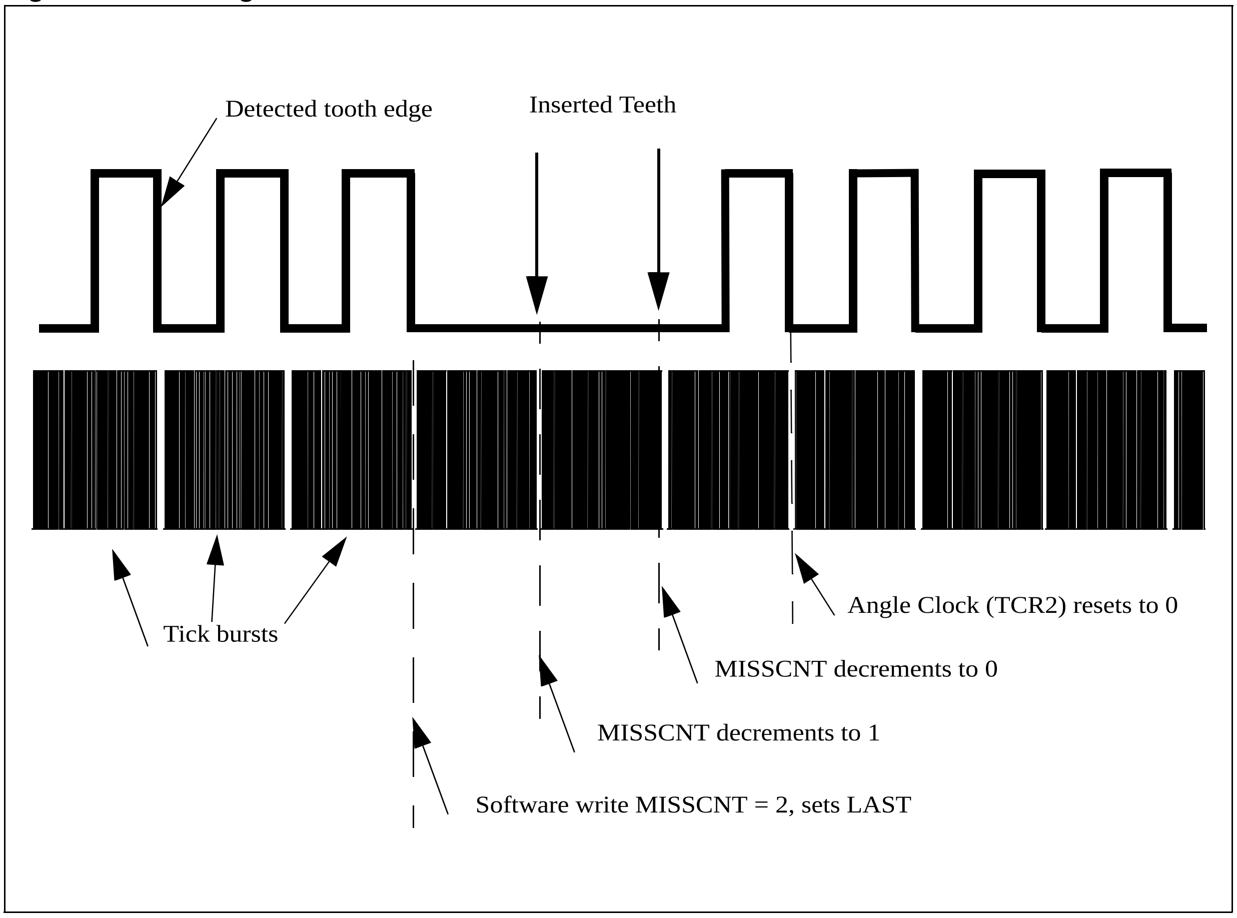
Note: *The angle clock is designed to count all TICKS between teeth, never skipping a count, and never proceeding with a count for the new tooth until the physical tooth is detected.*

2.3 Missing teeth

Typically a toothed wheel used in automotive applications has a series of equally spaced teeth with one or more teeth removed to provide a reference point. The angle clock hardware has circuitry to insert the tooth count for these missing teeth under control of eTPU software. One to three missing teeth can be added automatically by programming the field MISSCNT to a non-zero value after the last tooth edge before the gap.

When the eTPU software programs MISSCNT to a non-zero value, the tick counter proceeds as normal until the last count of the TICKS value is reached. Then after an additional TRR period, MISSCNT is tested, and when it is found to be non-zero, it is decremented and TCR is incremented as if a physical tooth had been detected. The TCR continues to increment at TRR rate according to the TICKS value for the period after the missing tooth. This process is repeated for all missing teeth. See *Figure 4*.

Figure 4. Missing teeth



The angle clock remains in normal mode throughout the missing tooth count unless a new physical tooth is detected before the final tick is counted. in that event the angle clock goes into high rate mode and completes the accelerated cycle as described above.

If the angle clock completes the count of the missing teeth plus all of the inserted ticks, and a new tooth is not detected, the angle clock goes into Halt mode as described above.

2.4 Zero count synchronization

For continuous synchronization of the angle clock with the position of the wheel, the angle clock has a feature to zero the TCR2 count on detection of a software designated edge. To use the automatic synchronization feature of the angle clock, the zero count must coincide with a physical tooth edge.

After the last tooth before the zero count synchronization, the software sets the LAST bit in the TPR to indicate that the current tooth is the last one in the count cycle. Then, when the next tooth is detected, the TCR2 count will automatically reset to 0 if the angle clock is in the Normal or Halt mode. If the angle clock is in the High Rate mode when a tooth is detected with the LAST bit set, the angle clock will continue the count of the ticks until it is complete before resetting the TCR2. When TCR2 is reset by a tooth edge with LAST set, the action also clears the LAST bit.

When the application uses the zero count synchronization feature to reset the TCR2 counter on a designated tooth each revolution, the TCR2 counter will continuously maintain a value proportional to the angle of the wheel over a 360 degree range. At steady state, if the software is properly implemented, the ticks between the teeth will be accurate within about one TCR1 count. Under deceleration, the tick positions may be in error, but the tooth angles will be accurate, while under acceleration, the tooth positions may be delayed while the tick count catches up at a high count rate.

If the software sets the LAST bit on alternate revolutions of the wheel, the TCR2 counter will count over an angle range of 720 degrees before resetting to zero. This is an important feature for 4-cycle automotive applications, where the control cycle is 720 degrees.

2.5 Error recovery

The eTPU angle clock hardware provides no automatic means to detect errors in the tooth count while tracking the wheel angle. If missing teeth are programmed between two normally spaced teeth, the hardware will absolutely insert the required counts, likely going into High Rate mode for a significant portion of the count. If no missing teeth are inserted in a gap, the counter will count no higher than TICKS counts and then go into Halt mode for the rest of the gap. If LAST is asserted on the wrong tooth, the TCR2 counter will reset, regardless of the TCR2 count, when the next tooth is detected. If the angle clock is untracked by missed detection of a tooth or spurious additional tooth, it is incumbent on the software, either in the host or the eTPU, to detect the error and initiate corrective action.

The angle clock hardware does provide means to make a correction in the TCR2 count when an error is verified by the software. If the application determines that the angle clock has missed a tooth count and is running with a one tooth deficit in the count, the eTPU software can insert an additional tooth by asserting the IPH bit in the TPR. When this signal is asserted, the hardware reacts exactly as if a physical tooth was detected. If the tick counter is counting in normal mode, it switches to high rate mode, completes the current tick count, then counts the inserted tooth and continues in normal mode. If the angle clock is in halt mode, it counts the inserted tooth and starts a tick count in normal mode. Note that if the IPH is asserted in High Rate mode, it is ignored.

Note: Inserting a tooth by asserting IPH will increase the TCR2 count by exactly one tooth angle by counting forward to the new angle. No counts are skipped.

If the application determines that the angle clock is ahead of the wheel, it can slow the count by asserting the HOLD bit in the TPR. When this signal is asserted, the angle clock hardware immediately suspends the count until an new tooth is detected. When the new tooth is detected, the angle clock operation resumes from the count it held when the HOLD bit was set.

Note: Asserting HOLD stops the TCR2 count for exactly one tooth angle.

There are certain restrictions and limitations to the use of the various signals in the angle clock TPR register.

- IPH may be used to reset the counter, if LAST is asserted when IPH is set.
- MISSCNT can be used to insert more than three missing teeth with the restriction that if MISSCNT is to be written on the same tooth as the LAST bit is set, the two fields must be written together.
- Do not use IPH to cancel a HOLD that is in progress. Simply clear the HOLD bit.
- When the tick counter is in high rate mode only IPH and HOLD can have immediate effect. All other fields in the TPR are buffered until the angle clock exits High Rate mode.
- If the TICKS field is changed before the angle clock goes into Halt mode, the tick counter will attempt to use the new value immediately after the current tick occurs. Note that the angle clock does not go into Halt mode until approximately one TRR time after the last tick is counted.

3 System design

There are four operating conditions that must be considered in the system design process: steady state operation, acceleration and deceleration, startup, and exception handling. The first two conditions covers a large majority of the operating time of the system but requires only a small portion of the development time. Startup is more complex, while exception processing tries to anticipate large numbers of exceedingly rare error conditions, which must be handled competently without human intervention.

Since the eTPU is only a small slave processor associated with a powerful MCU central processor, the system approach should be to entrust the steady state operation an much as possible to the eTPU, with assistance in startup. The eTPU might help in error detection, but since the exception handling may be complex, the proper systems approach is to use the eTPU to detect anomalies, but correct the errors where possible in the CPU.

3.1 Steady state system design

There are two basic angle counting modes that can be supported by the eTPU. In the periodic mode, the TCR2 can be programmed to maintain a real time representation of the absolute angle of the wheel with respect to a reference position, while in the free running mode it can track a continuously increasing angle from startup, rolling over as the TCR1 (time) counter does after reaching 0XFFFFFF (16777215) counts. The choice between these modes depends on the desired mode of comparison of angle positions.

3.1.1 Periodic angle clock

If the angle clock software sets LAST at the end of a cycle, typically at 360 or 720 degrees, the TCR2 counter will roll over to 0 at the tooth edge after LAST. This means that comparison to the engine angle must be done as equals-only. The greater-or-equal compare depends on the count rolling over after 0XFFFFFF. Greater-or-equal compares will be satisfied for any number from 0 to 0X7FFFFF less than the programmed angle. This means that an attempt to setup a compare in the future will not work properly if the future projected angle rolls over the end angle. In other words, 0 degrees will always test less than any other angle, including 715 degrees, even though with the periodic angle clock, it is 5 degrees in the future.

Engineers are understandably uneasy about using an equal-only compare. If a spark is programmed to fire at 32 degrees, missing the angle under any exceptional condition will cause an undesirable misfire. However, the eTPU angle clock is designed not to miss any comparison counts regardless of the speed dynamics of the input signal. This means that any compare that is programmed to occur at a selected angle will absolutely occur at the next occurrence of that angle in TCR2.

The only danger is in writing the compare value just too late for the intended cycle. If the spark time was changed, for example, to 31 degrees just after the angle clock reached 31.1 degrees, the spark could be missed. Therefore it must be a requirement of all angle triggered driver software to test for any change that steps the compare point over the current angle, and force the compare immediately when that happens.

3.1.2 Free running angle clock

To avoid the step-over problem mentioned above, a designer might choose to use the greater-or-equal comparator in the eTPU channel with a free running angle clock. In this case, the trigger angle for any function must be offset by the angle at a reference point on the wheel. For example, the software driver could maintain the top dead center (TDC) value for the current cylinder. If the spark is set to 12 degrees before TDC, the driver need only subtract the 12 degrees from the TDC angle to get an absolute firing point. Regardless when the software is updated, the greater-or-equal compare will ensure that a event is not missed. Then at some point in the cycle, for each driver the TDC angle must be updated for the next cycle.

The disadvantage of using a free running angle clock is that a host read of TCR2 will not directly yield the absolute position of the engine. The value will have to be reduced by a reference angle, modulus 360/720 degrees, and the reference angle will have to be updated every revolution.

Either the periodic or the free running angle clock can be applied successfully to an engine control system.

3.1.3 Angle resolution

The eTPU angle clock can resolve the angle between the teeth into as many as 1024 ticks. However the proper selection of the TICKS value in the TPR must take into account other considerations. To understand the limits for TICKS in a given system, the engineer must consider the configuration of the toothed wheel, the maximum and minimum speed that must be processed, and the selected resolution of the TCR1 counter.

The TCR1 counter resolution is usually a compromise between the finest resolution required by the eTPU channel drivers and the maximum pulse width that must be measured or driven. In fact, arbitrarily long pulses can be processed regardless of the TCR1 resolution by using software intervention, so the maximum time is seldom a great consideration. The minimum TCR1 resolution is two times the period of the MCU system clock. However, many systems engineers choose a convenient value derived from the clock. A typical TCR1 resolution might be 100 nanoseconds. The would enable the eTPU to time out pulses up to about 0.8 seconds with a single write to a match register. (Using the greater-than-or-equal comparison, the future equals half the range of the counter. Values larger than 0x800000 above the TCR value are considered in the past, and will match immediately using the greater-than-or-equal comparison.)

Now consider the TRR. It is organized as a 1-bit integer and a 9-bit fraction.This means that the largest value for a single tick would be 0X7FFF.1FF (32767.998) times the period of the TCR. Since a zero value in the integer field is processed as 32768, the minimum TRR value is actually 1. There are reasons to avoid both of these extremes, so the user should build in a little margin when setting the limits.

The maximum number of ticks per tooth should ensure that there is more than one TCR1 time per tick at the maximum speed for the wheel. The minimum number of ticks per tooth should ensure that the maximum value of TRR is not exceeded at the minimum wheel speed.

Example

Assume a TCR1 time of 100 nanoseconds, a maximum speed of 10,000 RPM and a 36 tooth (10 degree) wheel,

1 *MaxTicks < DegreesPerTooth * TRC1_Frequency / (60 * RPM_Maximum)

MaxTicks < 1 * 10.0 * 10,000,000 / 60 * 10,000 = 166. ticks per tooth

Assume a TCR1 time of 100 nanoseconds, a minimum speed of 20 RPM and a 36 tooth (10 degree) wheel,

32768 * MinTicks > DegreesPerTooth * TRC1_Frequency / (60 * RPM_Minimum)

MinTicks > 10.0 * 10,000,000 / (60 * 20 * 32768) = 2.543 ticks per tooth

choose 100 for 10 ticks per degree (0.1 degree per tick)

3.1.4 Reference point

The reference point for a periodic angle clock should be on a selected physical tooth edge. Typically this might be the first tooth edge after the gap, but any edge can be chosen. Note that if the angle clock is to cycle through 720 degrees rotation, the resetting tooth is found on alternate rotations of the wheel. Selecting an absolute reference for the 720 degree cycle must be done using an external reference such as a cam shaft sensor.

The angle clock will automatically reset to zero if a physical tooth edge is detected with the LAST bit set in the TPR. This means that the system software must keep track of the tooth count and write the LAST bit at the appropriate time.

3.1.5 Missing teeth

In an automotive engine control system, the toothed wheel on the crankshaft typically has one or more missing teeth to provide an angular reference point. The angle clock hardware can provide compensation for these missing teeth, adding a tooth count and repeating the tick count for each missing tooth to bridge the gap in the teeth. The system software must keep track of the tooth count and, after the last edge before the gap, write the number of expected missing teeth to the MISSCNT field in the TPR.

3.1.6 Tick rate

The eTPU can automatically record the time of a tooth transition on the Channel 0 input. If the software has stored the previous edge time, then the period of the tooth signal can be calculated by subtracting the times. To derive the value for the TRR, the software must divide the time for a tooth period by the number of ticks per tooth. This calculation must be adjusted if the most recent period was measured over the gap.

Note that the tooth period is a measure of the rotational speed of the engine. Typically this value is kept in a global memory location. In that case, it should be updated with the adjusted period when measured over the gap.

3.2 Acceleration and deceleration

Speed changes in the wheel can be easily handled by the eTPU angle clock hardware if it is properly managed by the eTPU software. Since the tick counter is an extrapolation of the speed of the wheel, the accuracy of the ticks is determined by the accuracy of the TRR value plus a possible error due to software latency. The systems engineer needs to understand the maximum acceleration expected in the wheel and design adequate compensation in the software.

3.2.1 Acceleration

Acceleration is manifested in the angle clock by a tooth edge arriving earlier than expected whenever the period of the last tooth separation is less than the pervious period. When the previous tooth edge was detected, the TRR was written with a value that assumed evenly spaced ticks between that tooth and the next. Acceleration of the wheel means that the latest tooth is detected before the tick count is completed. When this occurs, the angle clock switches automatically into High Rate mode. The balance of the ticks are counted at a rate equal to the MCU system clock (not TCR1) divided by eight. Once the tick count is complete, one additional count is added for the newly arrived tooth. The angle clock then reverts to the Normal mode.

Typically, the detection of the tooth edge causes the Channel 0 logic to request service from the eTPU engine. This service request is granted by a scheduler in a manner determined by the following:

1. If the eTPU is idle and no other channel is requesting service, the scheduler will grant the Channel 0 request immediately. The eTPU will require 6 system clocks to start executing the service thread.
2. If more than one channel is requesting service, the scheduler will arbitrate the requests according to the priorities and recent history of the scheduler. See the Reference Manual for a detailed treatment of the scheduler process.
3. If another channel is being serviced, the eTPU will complete the thread being serviced, then revert to 2.

Presumably, the service thread will determine the previous period and calculate a new TRR value. Then, at some point in the thread, the TRR will be written. However, depending on latency, the new tick stream may have started counting before the updated TRR was written. In that event, all ticks that have been timed during the latency period will presumable be slow and, after the TRR is corrected, the current period may run late as well.This effect is actually rather small in real automotive systems, except perhaps during the very first firing while cranking.

Example

A 36-1 (35 ten-degree teeth with one missing tooth) tooth wheels is running at 720 RPM when the engine is accelerated at a rate of 5000 RPM per second. If the angle clock was inserting 100 ticks per tooth, and the latency for the tooth service was 10 microseconds, what is the tick distribution in the first two gaps after acceleration starts? (system clock = 100 MHz)

Original TRR value:

720 RPM * 1/60 * 36 = 432 teeth per second => 2.315 msec/tooth => 23.15 µsec/tick

Speed change over the first tooth:

5000 RPM/sec * 0.002315 sec = 11.6 RPM

Second TRR value:

(720 + 11.6) RPM * 1/60 * 36 = 439 teeth per second => 2.278 msec/tooth => 22.78 µsec/tick

Second tick count:

2.278 msec/23.15 µsec = 98.4 ticks

The edge arrives when the count is 98, therefore count 99 and 100 (tooth count) occur at 0.08 and 0.16 µsec after the tooth arrives. The largest tick error would be tick 98 which would occur about (100 - 98.4) 1.6 ticks (0.16 degrees) late.

Speed change over the second tooth:

5000 RPM/sec * 0.002278 sec = 11.4 RPM

Third TRR value:

(731.6 + 11.4) RPM * 1/60 * 36 = 446 teeth per second => 2.243 msec/tooth => 22.43 µsec/tick

Third tick count:

2.243 msec/22.78 µsec = 98.5 ticks

...which we must adjust for the high rate counts plus the error in the first tick position:

(0.16 µsec + (22.78µsec - 22.43µsec))/ 22.43 µsec/tick = 0.02 ticks

The edge arrives when the count is about 98.5, therefore count 99 and 100 (tooth count) occur at 0.08 and 0.16 µsec after the tooth arrives. The largest tick error would be tick 98 which would occur about 1.5 tick positions (0.15 degrees) late.

As can be seen from the example, the angle clock can track a typical automotive toothed wheel quite well under normal operating conditions. Also, the faster the speed, the less the effect of a given acceleration rate.

3.2.2 Deceleration

When the angle clock completes the tick count and the next tooth does not arrive, it goes into halt mode. Then when the next tooth does arrive it is counted and the tick count starts again with the previous low TRR value. When a new TRR value is calculated, the tick rate is decreased. The angle tracking under deceleration is similar to acceleration tracking.

Example

In the example above, what is the error when the wheel decelerated from 731.6 RPM to 720 RPM in one tooth time?

Second count theoretical minus second count actual:

99 * 23.15 µsec - 99 * 22.78 µsec = 36.6 µsec => 36.6 / 22.78 = 1.6 ticks

The 99th tick arrives 36.6 µsec or 1.6 ticks early. This is the largest angle error (0.16 degrees).

See also discussion of excessive acceleration and deceleration in *Section 3.4: Anomalies on page 19*.

3.3 Startup system design

Starting the angle clock requires three essential steps. First, the initial values must be written to the function parameters and control registers. Second, the tooth signal is acquired, the periods are tracked, and the tooth pattern is tested for a reference point, usually the first gap. Third, when the angle is determined, the actual angle is written to the TCR2, synchronizing it to the wheel and the rest of the angle based control system can be started.

3.3.1 Initial parameter values

The angle clock hardware must be setup in the eTPU by writing certain control registers by the host. This includes writing the AM bit in the ETPUTBCR and configuring the TCRCLK input for the selected eTPU. If the angle clock is to be used as a server for another eTPU or for the eMIOS, the shared timing and control (STAC) bus must be set up. The essential software steps will be detailed below.

The Channel 0 parameters must be written to initial values by the host at startup. Most important are the angle clock hardware register values (which are subsequently written by the eTPU software to the registers) and the initial position of the angle clock.

While the software is searching for the reference point, the angle clock may be held to a particular value, or it may be counting from some arbitrary point. The system designer should take care that other drivers in his system do not trigger on arbitrary value of TCR2 before the angle clock in synchronized.

There are several ways to avoid spurious matches and pin action during startup:

- The eTPU may be kept out of Angle Mode until the channel hardware has searched and found the reference point. TCR2 can be disabled by writing 0b111 to the TCR2CTL field of the ETPUTBCR.

Note: *Do not set TCR2CTL to 0b111 unless the angle mode is disabled by setting AM to 0.*

- The angle clock can be allowed to run during startup, but other drivers dependent on the angle clock must be kept disabled until synchronization. While this strategy is used in the Reference Design, it is not the preferred strategy.
- TCR2 can be maintained at a large negative and/or out of range value during acquisition and synchronization.

The main advantage of letting the angle clock run during startup and synchronization is to reduce the number of special cases handled in the software.

3.3.2 Acquisition and synchronization

The first detected tooth edge indicates that the wheel is moving. The only information that can be gathered is the TCR1 time of the edge.

The second tooth provides a new edge time and the first tooth period. Note that in most systems the period value alone cannot distinguish between a normal tooth separation or the multiple tooth width of the gap. However, with the first period recorded, the system can now start to search for the first gap.

Once the third tooth edge is detected, a second tooth period is known and the search for the gap can begin. There are various ways to identify the gap using the time between consecutive pairs of teeth. The larger the gap, that is the more teeth are missing, the easier it is to discriminate between normal teeth and the tooth gap. However, variations in period size can also indicate acceleration or deceleration of the wheel. If the size of tooth period is approximately twice the previous period, this might indicate a gap, or a severe deceleration of the wheel.

A common way to ensure that a gap has been detected is to confirm the measurement on the following tooth edge. Whatever test is used to identify a long period can be applied in reverse to the following period to test for an ABA pattern of periods. The presumption here is that the wheel would not make a sudden sharp deceleration, followed one tooth later by a sudden sharp acceleration. Whether this or any other test is valid for a given system must be determined by the systems engineer. The ABA test is considered valid for automotive applications because of the significant inertia of the crankshaft.

EXAMPLE

Four consecutive tooth edge times are found to be:

Capture[1] = 0x1B238A

Capture[2] = 0x1CBA70

Capture[3] = 0x1FAB33

Capture[4] = 0x211145

Subtracting adjacent tooth times yields the following periods:

Period[1] = 0x196E6

Period[2] = 0x2F0C3

Period[3] = 0x16612

We can use a “Gap Ratio” algorithm to test if this represents a gap detection.

Use Gap_Ratio = 0.7

Is_Gap = if ((Period[2] * Gap_Ratio > Period[1]) && (Period[3] < Gap_Ratio * Period[2])

Period[2] * Gap_Ratio = 0x203BB

This is larger than Period[1] and Period[3]...

...therefore the Is_Gap condition is satisfied.

3.3.3 Starting the angle clock

Once the gap has been found and verified, the application knows the position of the wheel at the time of the last tooth edge. The angle clock can now be initialized with this angle. There are two considerations for the systems engineer at this point. First, if the application is synchronizing to a 720 degree cycle, there may or may not be sufficient information to determine the correct half of the engine cycle. Additional information might be required to completely synchronize the angle clock to the wheel. Second, there is typically some time lost between the detection of the last edge and the software conclusion that the wheel angle is verified. If the angle clock is arbitrarily started at this time, there may be some error in the indicated angle.

If the angle clock is required to track a 720 degree cycle, then it is possible that the information on which half of the cycle the gap was detected is not yet available to the application. While full synchronization may not have been achieved, the application can declare “Half_Sync.” This means that the crank shaft position is known but the cam shaft position may subsequently require the shaft angle to be corrected by 360 degrees. In an automotive application, Half_Sync is sufficient to operate the engine using a wasted spark and split fueling strategy. If the system is operated in a half sync mode until full sync is possible, it is important to consider the effect on the system of the angle clock getting a sudden 360 degree correction if this should be required.

Various cam shaft detection strategies may be used to effectively eliminate the cam uncertainty at the time the crank shaft is synchronized. If such a strategy is available, Half_Sync is not required.

When synchronization is verified, the angle at the last tooth edge is known. The software should immediately write that angle to TCR2, write the TRR value derived from the last period, and start the angle clock. The first series of ticks will be delayed by the latency of the function to this point, but if necessary the angle clock will go into High Rate mode at the next tooth edge to catch up.

3.4 Anomalies

There are two ways to minimize the effects of anomalies that can cause loss of synchronization in the eTPU angle clock: prevent as many as possible, and provide a means to detect and correct those that cannot be prevented. The prevention is typically done by the eTPU hardware system, properly setup by the software. However, once an anomaly is detected and processed by the system either as a superfluous or a missing tooth edge signal, the angle clock design can only help detect the problem, and pass the information on to the host for corrective action.

3.4.1 Error prevention

The eTPU angle clock provides means to reject tooth edges detected outside of certain limits. This rejection is done by an input pin hardware filter and by gating the tooth signal with a match registers in the channel.

The TCRCLK input pin filter for the angle clock can use the input pin filters of the other eTPU channels, or it can be programmed separately. These filters are designed to reject very short apparent pin transitions such as might be caused by high frequency induced noise. In any case, this low pass filtering of the tooth signal will introduce a time delay in the signal from the tooth. Since the filter time is generally in the order of a few system clock times, the angle represented by this delay is very small even at high wheel speeds. See the Reference Manual for details of the TCRCLK filter.

When Channel 0 is in one of the *m2_* modes, the MatchA register can be used as a blocking timer for input signals. In this mode, the eTPU software can provide a blanking time after the detection of a tooth edge, during which time a (presumably spurious) additional edge will be blocked. This feature is particularly useful when an imperfect tooth detection circuit presents a chopped or hashed edge to the eTPU, which often happens at low speed. If the frequency of the edge noise is lower than the pin filter rejection, a noise pulse could be detected and counted as a tooth edge signal unless this blanking is used. The blanking match can be programmed as an absolute time or calculated by the eTPU as a percentage of the previous period. The choice depends on the expected characteristics of the edge noise.

If a noisy edge is possible, then a noisy return edge might also be detected as an active transition. If blanking is used to prevent noise from being processed as a tooth edge, then the blanking must be applied to both edges of the tooth signal.

If a tooth signal is completely lost to the eTPU channel by some means, the hardware alone cannot distinguish the long period from any other deceleration. However, a second match can be setup in Channel 0 to request service if a transition has not occurred within a specified time.

3.4.2 Error recovery

The tick counter in the eTPU angle clock ensures that there are exactly TICKS counts between physical teeth. If the angle clock acquires an error less than one tooth count, the only possible reason is a software error. Nevertheless, if even if the count acquires a small error, the count will rectify when the LAST tooth is processed.

Note: *The most common cause of a small TCR2 angle error is writing the initial angle to the angle clock when synchronization occurs. When gap synchronization is first verified, if the tick rate timer expires just as the TCR2 value is incremented by the tick counter, the tick counts for subsequent teeth count may fall short by one tick until the last tooth re-synchronizes the tick counter.*

All other sources of error for the angle clock involve detecting an extra tooth or missing one which should have been detected. The angle clock hardware cannot automatically correct a tooth detection error. If a lost or extra tooth error is possible, the application software must provide some means to detect the error. The most obvious method is to continually monitor the position of the gap by repeating the gap detection and verification algorithm. If the gap is detected at the wrong tooth position, or not detected at the expected one, the angle count can be corrected. The decision to correct an apparent error can be made in the CPU or the eTPU software, although prudence dictates that the decision is left to the more powerful host processor.

The eTPU angle clock provides two features to correct a TCR2 counter that has become untracked by one tooth. Each of these corrections can be repeated to make a multiple tooth correction.

Inserting a tooth

If the angle clock hardware has missed counting a tooth, the software can assert IPH in the TPR to add a single tooth. The effect of this assertion is to force the eTPU to recognize a physical tooth immediately. If the tick counter is in Halt mode, the angle clock counts the inserted tooth, then begins a new sequence of tick counts exactly as if a tooth had been detected by the channel. If the tick counter is in Normal mode, it immediately goes into High Rate mode until the tick count is complete, then the inserted count is counted as a physical tooth, and finally the angle clock returns to Normal mode. Note that if IPH is asserted when the angle clock is in High Rate mode, it will be ignored.

The effect of inserting a tooth is that the angle clock will advance the angle count by one tooth without missing any ticks. If a match is scheduled for an angle between the asserting of IPH and the time the count is rectified, it will occur.

Note: *Asserting IPH (except when asserted during High Rate mode) will always add exactly one tooth to the TCR2 angle count. The correction will be made using the High Rate mode, and no count will be skipped.*

Deleting a tooth

If the angle clock hardware has mistakenly counted an extra tooth, the software can assert HOLD in the TPR to stop the count for exactly one tooth count. The effect of this assertion is to stop the tick count immediately until the next tooth is detected. When that tooth is detected, the tick counter continues where it left off, meaning that the tick count will count a portion of TICKS before the assertion of HOLD, and the balance after the next tooth.

Do not use IPH to cancel a HOLD in progress. To correct the correction, simply negate HOLD.

Note: Asserting HOLD will always subtract exactly one tooth from the TCR2 angle count. The correction is made by stopping the count, and the angle count does not decrement or repeat any count.

3.5 Tooth circuit failure

If the tooth circuit fails and the failure is detected by the host, the software may implement a limp home strategy. The eTPU angle clock provides a method to disconnect the angle clock system from the Channel 0 input so that spurious signals from the tooth circuit will not disrupt the limp home system. In that event, the host may configure the eTPU such that Channel 0 continues to monitor the tooth circuit for diagnostic purpose. See the Reference Manual for details.

4 eTPU software design

The software shown below is taken from an automotive reference design which demonstrates the operation and application of the eTPU angle clock in a practical automotive engine control system. Code that is directly copied is displayed in `Courier New` font. The reference design can be obtained from your ST representative.

4.1 Host setup

The host must setup the registers and parameters in the eTPU to enable the angle clock. In addition to the setup required for any eTPU operation, use of the angle clock requires the following steps.

a) The Angle Mode bit in the ETPUTBCR must be set.

b) The TCRCLK Filter Mode must be setup.

c) Initial Channel 0 parameters must be written.

d) Channel 0 must be initialized.

e) The TCR2 Clock Source in the ETPUTBCR must be set for rising or falling edges to match the IPAC setting for the tooth in Channel 0.

The last step may be delayed until Channel 0 has confirmed synchronization.

4.2 State machine description

Table 1 shows a simplified state table for an angle clock system, similar to the implementation in the reference design. The states are referenced in the following description of an angle clock system implementation.

Table 1. Angle clock system state

State	Reference name	Action	Exit condition	GoTo State
	InitializeCrank	TICKS = TicksPerToot - 1	If ActiveEdge	
		ToothCount = *0*		
		CrankStatus = `Stall;`		
	First_edge	**CrankStatus** = `WaitingGap`	If timeout (StallTimer)	
		StallTimer = *StallPeriod* at startpoint Time(activeEdge)	If ActiveEdge	
		ToothTime = Time(ActiveEdge)		

Table 1. **Angle clock system state (continued)**

State	Reference name	Action	Exit condition	GoTo State
	First Period	StallTimer = *StallPeriod* at startpoint Time(activeEdge)	If ActiveEdge	
		Period = Time(ActiveEdge) - **ToothTime**	If timeout (StallTimer)	
		ToothTime = Time(ActiveEdge)		
		ToothCount += 1		
		TickRate = **Period**/TicksPerTooth		
		BlankTimer = **Period** * *BlankingRatio*		
	Testing_Possible_Gap	StallTimer = *StallPeriod* at startpoint Time(activeEdge)	If **Period**>*GapRatio* * PeriodLast	
		PeriodLast = **Period**	If ActiveEdge	
		Period = Time(ActiveEdge) - **ToothTime**	If timeout (StallTimer)	
		ToothTime = Time(ActiveEdge)		
		ToothCount += 1		
		TickRate = **Period**/TicksPerTooth		
		BlankTimer = **Period** * *BlankingRatio*		
	Apparent_Gap	**Period** = PeriodLast	If ActiveEdge	
		TickRate = **Period**/TicksPerTooth	If timeout (StallTimer)	
	PossibleGapVerifying	StallTimer = *StallPeriod* at startpoint Time(activeEdge)	If Tooth_Period*Gap_ Ratio < PeriodLast	
		PeriodLast = **Period**	If Active Edge	
		Period = Time(ActiveEdge) - **ToothTime**	If timeout (StallTimer)	
		ToothTime = Time(ActiveEdge)		
		ToothCount += 1		
		TickRate = **Period**/TicksPerTooth		
		BlankTimer = **Period** * *BlankingRatio*		
	GapVerified	**CrankStatus** = `HalfSync`	If ActiveEdge	
		ToothCount = *AngleSync*/TicksPerTooth	If timeout (StallTimer)	
		Period = PeriodLast		
		TickRate = **Period**/TicksPerTooth		
		BlankTimer = **Period** * *BlankingRatio*		
		EngineAngle = *AngleSync*		

Table 1. Angle clock system state (continued)

State	Reference name	Action	Exit condition	GoTo State
	Counting	StallTimer = *StallPeriod* at startpoint Time(activeEdge)	If ToothCount == ToothBeforeGap	
		PeriodLast = **Period**	If ActiveEdge	
		Period = Time(ActiveEdge) - **ToothTime**	If AdjustAngle() //return	
		ToothTime = Time(ActiveEdge)	If timeout (StallTimer)	
		ToothCount += 1	If Receive(Signal)	12
		if (ToothCount == LastRealTooth) then LAST = `true`		
		TickRate = **Period**/TicksPerTooth		
		BlankTimer = **Period** * *BlankingRatio*		
	Gapping	MISSCNT = *NumberMissing*	If ActiveEdge	
		if ((FirstGap && !CamDetected)||(SecondGap && CamDetected) then CrankStatus = `Full_Sync` else CrankStatus = `Error`	If AdjustAngle() //return	
		StallTimer = (*StallPeriod + NumberMissing** **Period)** at startpoint Time(activeEdge)	If timeout (StallTimer)	
		BlankTimer = **Period** * (*NumberMissing + BlankingRatio)*		
	NewRev	if AngleClock = 0 then ToothCount = 0	If ActiveEdge	
		ToothTime = Time(ActiveEdge)	If AdjustAngle() //return	
		BlankTimer = **Period** * *BlankingRatio*	If timeout (StallTimer)	
	AdjustAngle	**EngineAngle** = **EngineAngle** + *AngleCorrection*	Return to last state.	x
		Do Not Change State Number		

4.2.1 Initialize crank

The first state is entered by a host service request. The eTPU parameters are initialized and hardware is setup for the specific application.

Example

```
   if (hsrInitAngleClock)   // Sets Entry Vector for the eTPU
   {
/* Initialize Parameters */
   State1:
      // Set the channel mode to detection one input transition and
two match events
      SetChannelMode(m2_st);
/* ... */
     // Setup action unit A to capture high to low transition
     SetupCaptureTrans_A(Capture_tcr1, high_low);
/* Match2SingleTransition mode requires a MatchA before the
transition can be detected. Force the first match in a couple of
microseconds. */
     // Setup match to block out transition too close to current
time
     SetupMatch_A((tcr1+20), Mtcr1_Ctcr1_ge, match_no_change);
   }
```

4.2.2 First edge

The eTPU will wait until a transition is detected on the input pin. The channel service request vector for TransitionA is shared with Match B. Also, the channel cannot distinguish between the initial startup tooth transitions and subsequent ones. Therefore the software must sort out the service requests from the channel when they are received.

Example

```
   else if (matchB_transA) //Here on MatchB (stall) or TransitionA
(Tooth edge)
   {
     if (IsTransALatched())   //If the Tooth edge is detected
     {
/* Do some common tasks... */
        switch(Last_State)
        {
           case InitializeCrank:
              // State 2
              CrankStatus = Waiting_Gap;
              SetChannelInterrupt();//Inform the host that the crank
status is updated
              Last_State = First_edge;
              break;
/* Additional cases inserted here */
        }    // end of Switch...case statement
        // The transition capture only need to setup once at init.
        // Clear transition latch will re-arm the transition
detection
        ClearTransLatch();
        // Setup match event to block out input trans too close to
the valid falling edge.

        // The transition detection is disabled until the match event
on the action unit A occur.

        SetupMatch_A((EdgeCaptureTime + ToothPeriod*BlankingRatio),
Mtcr1_Ctcr1_ge, match_no_change);


        // Setup match event to detect stall condition

        // If the crank edge transition does not occur again for a
StallPeriod, it is stalled

        SetupMatch_B((EdgeCaptureTime + StallPeriod),
Mtcr1_Ctcr1_ge, match_no_change);

     }  // active edge
```

4.2.3 First period

When the second edge is detected, the function can make the first measurement of the tooth period. The period should be stored as a global variable, as it is the best measure of the wheel speed and may be used by other drivers. Before it is overwritten, the previous period should be stored off for use in detecting the gap.

4.2.4 Testing for a possible gap

When the third edge is detected, enough information finally exists to test for the missing tooth gap. This routine is now the default for subsequent teeth. This state is repeated until the gap test is satisfied after which a confirming test needs to performed on the following period.

It is necessary from this point on to count the teeth that are detected. For example, if a 36-1 tooth wheel produces more than 35 tooth periods without a gap detection, an exception might be raised with the host. Typically the tooth count is rectified when the first gap is detected. After that, the software can anticipate subsequent gap detections.

If the wheel is to be tracked over 720 degrees as is necessary for a four cycle automotive engine, a separate search needs to be conducted for the unique cam information indicating which of the two crankshaft cycles are currently being tracked. Note that the cam information and the crank synchronization are both needed to determine full synchronization.

In addition to tracking the tooth period and count, the angle clock software should provide status to the host and other applications using the angle information. The status selections might include:

1. *Stall* indicates that the angle clock has not yet determined a valid period, or the time since the last tooth transition exceeded a specified limit.
2. *Searching_Gap* is used to indicate that teeth have been detected at an acceptable rate, but the first gap has not yet been detected and confirmed.
3. *Half_Sync* in a 720 degree system indicates that the gap has been verified, but the cam cycle information has not been determined.
4. *Full_Sync* indicates that the TCR2 counter is synchronized with the wheel and the cam shaft.
5. *Error* indicates that the angle clock system has detected an incorrect state change.

Note: In a four-cycle engine, some limited operation is possible in Half_Sync mode.

Example

```
case Testing_Possible_Gap:
          {
            ToothPeriodLast = ToothPeriod;
            ToothPeriod= TempToothPeriod;
            ToothTime = EdgeCaptureTime;
            ToothCount = ToothCount + 1;


            if (ToothPeriod > (ToothPeriodLast*GapRatio +
ToothPeriodLast))
            {
              Last_State = Apparent_Gap;
            }
            else
            {
              Last_State = Testing_Possible_Gap;
/* if the test fails, we continue testing... */
            }
            break;
```

4.2.5 Apparent gap

When the gap test is satisfied it may be necessary to confirm the fact by testing the period of the following gap.

4.2.6 Possible gap verifying

One possible means to verify a probable gap is to determine that the period following appears to be shorter by a test similar to the one used to find the gap. The algorithm and parameters used for gap detection and verification must satisfy the requirements for the target system. If the gap is not verified, the system must return to the testing for a possible gap state.

Example

```
          case Apparent_Gap:
            ToothPeriodLast = ToothPeriod;
            ToothPeriod= TempToothPeriod;
            ToothTime = EdgeCaptureTime;
            ToothCount = ToothCount + 1;
            if (ToothPeriod >= ToothPeriodLast*GapRatio)//verify
            {
              CrankStatus = Half_Sync;
              SetChannelInterrupt();//Inform the host that the crank
status is updated
              ToothCount = 1;
/* More */
              Last_State = GapVerified;
            }
            else
              Last_State = PossibleGapVerifying;
/* More */
            break;
```

4.2.7 Gap verified

Once the gap is verified, the system can go to *Half_Sync* or even *Full_Sync* mode. The tooth position is now known, and subsequent gaps can be anticipated. An angle can be written to the TCR2, although if the mode is *Half_Sync,* the angle could be off by 360 degrees.

4.2.8 Counting

Once wheel sync has been achieved, the angle clock system executes repeated cycles of tooth detection and period timing, driving the TCR2 in wheel synchronization. The rate of the ticks following a tooth are extrapolated from the previous period by the software when it programs the Tick Rate register.

The Tick Rate register is organized as a 15-bit whole number and a 9-bit fraction and is programmed in terms of TCR1 counts. The eTPU can automatically capture the TCR1 time of each tooth edge, so the period is trivial to determine. The new edge time is then stored for the next period calculation. To determine the tick period, the software need only divide the measured tooth period by the number of tick counts per tooth. This division can present some problems for the designer depending on the selected resolution of TCR1 and the speed range of the wheel. See *Section 3.1.3: Angle resolution on page 13*. Even when the TCR1 is properly selected, the software engineer needs to ensure that the intermediate result of the fixed point calculation does not overrun a register or truncate needed resolution of the tick rate.

At low wheel speeds, the period is long and a left shift can be done after dividing by the ticks per tooth. If the shift is done before the division, the value may overflow. However, at high speeds, the period is short and dividing by the ticks per tooth could result in gross errors before the shift is done. The algorithm in the example below is designed to implement the 9-bit shift in two steps to minimize loss of resolution.

Example

```
                  if (IsTransALatched())    //If the Tooth edge is
detected
      {
           EdgeCaptureTime = GetCapRegA();//Capture the edge
transition time
           TempToothPeriod = EdgeCaptureTime-ToothTime;
         ToothTime = EdgeCaptureTime;
         TempTicks = ((TempToothPeriod*8)/TicksPerTooth)*64;//
implements a left shift by 9 bits and a div by 60
               TickRate = TempTicks;
```

Note: The value in the tick rate register must never be programmed to less than 1.0. If the integer value is set to 0, it will be interpreted as 32768.

The angle clock tick counter begins timing out when the tooth edge occurs. The calculation of a new value for the Tick Rate Register will take some time to complete. While this is being done the old value is used for the tick rate. At steady state this makes no difference, but when the wheel is accelerating, the first tick is necessarily late. If the acceleration continues, all of the tick counts will be late and the angle clock will have to go into High Rate mode to catch up.

It is possible to compensate for the accelerating wheel by adjusting the tick rate so that the next tooth is counted a bit faster than a simple extrapolation. This adjustment can be added to the software design at the engineer's discretion.

This state is repeated for each new tooth detection until one of the following occurs:

1. The tooth count indicates that the gap is following next.
2. The tooth count indicates that the next tooth is the zero angle reference point.
3. The host requests service, for example to correct an error.
4. The following tooth is not detected within a specified time (stall).

4.2.9 Gapping

When the tooth count indicates that a new gap is expected, the system must verify that measured gap times pass the designated test. If the gap is correctly placed, no host action is required. If the gap period is outside the range of acceptability, the information should be passed up to the host by setting the status to *Error*. The application can then analyze the error and execute a correction procedure.

Note: Once synchronized, the angle clock system can proceed indefinitely without further host action, unless a signal anomaly occurs.

4.2.10 New revolution

When the count of the teeth indicates that the last tooth in the cycle has been detected, and if the system required a periodic angle count, the software must set the LAST bit so that the counter will reset at the next detected tooth edge. Note that the counter reset will not occur on a tooth count inserted by MISSCNT during the gap, although asserting the IPH when LAST is set will reset the TCR2.

In a 720 degree system, this point occurs every second revolution of the wheel. The phasing of the count may be done automatically (for example, by using a cam detected signal to enable the reset logic) or manually by having the host force *Full_Sync* based on the cam information.

4.2.11 Adjust angle

Provision may be made in the angle clock software for the host to alter the angle indicated by the TCR2 counter. This action could be taken as a result of an independent determination that the counter fell out of sync, perhaps as a result of detecting a spurious tooth. The system can use IPH or HOLD to advance or retard the angle count by one tooth. Remember to correct the tooth count as well so that the next reset occurs at the correct tooth position. When the Adjust Angle service request is processed, the function should revert to *Full_Sync* mode. If the correction is made in error, the system software will indicate this at the next gap.

4.2.12 Stall

Provision should be made to detect the cessation of the tooth signals. Typically the system resets a match timer at each tooth edge, and reverts to a *Stall* status if the timer expires before the next tooth is detected.

5 Angle clock application

The angle clock was implemented in the reference design with little more software than is listed in this section. Several variations of the angle clock software have been written for various applications. ST provides the reference design with a complete working angle clock and several drivers. See your ST representative.

As described above, the eTPU angle clock can implement a free running angle counter or a periodic wheel angle indicator. Each alternative has advantages as well as problems, which are discussed briefly below.

5.1 Free running angle counter

If the LAST bit is not set periodically by the angle clock software, the wheel angle will accumulate 16777215 ticks then roll over to zero. If this is applied to a 60-2 tooth wheel with 60 ticks per tooth, each revolution of the engine will increment the count by 0xE10 (3600) counts. Thus a specific point on the wheel which is represented by 0xABC in TCR2 will on the next revolution be represented by 0x18CC, and on the next cycle by 0x26DC. In order to find the absolute position on the wheel at any time, the software must subtract the angle at some reference point from the indicated angle. While this seems tedious, the free running angle counter offers one clear advantage: match comparisons for angles can be done using the greater-or-equal compare. Any angle which previously occurred up to about 2300 revolutions in the past can be immediately matched with respect to the current angle.

However, if a free running angle counter is used, each system function using the angle needs to use the reference point to schedule or capture a channel action. If spark is to be scheduled for 12 degrees before top dead center (BTDC), the spark driver function needs to know a reference point in order to program the correct TCR2 value into the match register. In general the host software may change the spark angle at any time. If a change is made which would require a spark to occur in the past, the greater-or-equal compare would trigger a match immediately.

The free running counter design is much simpler if the number of teeth on the wheel and number of ticks between the teeth were each powers of 2. For example, if a 64-2 tooth wheel was used in a system with 128 ticks per tooth, the number of ticks per revolution would be 64 (0x40) times 128 (0x80) or 8192 (0x2000). In this case the counter would count exactly 2048 (0x800) revolutions and roll over. The TCR2 counter value would simply be a concatenation of an 11-bit revolution number, a 6-bit tooth number, and a 7-bit tick number.

5.2 Periodic angle clock

The obvious advantage of the periodic angle clock is that there is a one-to-one correspondence between the TCR2 value and the instantaneous angle of the wheel, regardless of the tooth and ticks configuration. The angle can be read at any time both by the host and by the eTPU functions. Periodic matches are always programmed with the same value.

However, the greater-or-equal compare in the channel match registers will not function properly with a periodic angle counter. For example, an angle of 719 degrees will always test greater than 0 degrees.

In an equals-only setup, when a function sets up a channel for action at a future angle, that angle can be any value in the range of the angle clock and the match will occur in the next occurrence of that angle. This is true even if the angle is 719 degrees in the future. The corollary of this is that if a system update results in a decision that an action that was scheduled should occur in the past, the match will wait until the next revolution.

Note: In the eTPU angle clock, no counts will be skipped. Any in-range equals-only match will always be satisfied on the next occurrence of the angle.

The discrimination between reprogramming to the past rather than the future is left to the eTPU software. The designer must insure that, for example, if a spark is about to occur and the spark fire angle is updated to occur at an earlier angle, the match is forced immediately. If the earlier angle is simply written to the match register, the spark will be missed until the next cycle.

There is one additional situation to consider when using equals-only matches with the angle clock. If the tick rate is slow, it is possible to get a service request on an angle match and execute part of a thread before the angle changes. If a function is trying to set up a future match exactly 360 (or 720) degrees after the current one, care must be taken that the register is not written with the current value of TCR2.

Please refer to the reference design for examples of how to use a periodic angle clock.

5.3 Angle math

If the angle clock is used in the continuous angle mode, angles parameters can be added, subtracted, and compared like any other 24-bit integer. However, when the user selects a periodic reset of the angle clock, modulus arithmetic must be used. If the angle clock is reset every cam revolution (= two crank revolutions), the modulus is 720 degrees expressed in terms of the tick resolution. For example if a 60-tooth wheel is used with 60 ticks per tooth, there would be 7200 ticks per cam cycle (Ticks_Per_Cycle).

Adding two angles in such an application would be done thusly:

```
New_Angle = (First_Angle + Second_Angle) % Ticks_Per_Cycle;
```

This operation is mathematically correct, but it might not generate the most efficient code. This is because the modulus is normally generated as a divide instruction. A more efficient coding would be to add the numbers, and if the sum is greater than Ticks_Per_Cycle, subtract Ticks_Per_Cycle. Macros or subroutines for handling this math can be found in the reference design.

6 Revision history

Table 2. Document revision history

Date	Revision	Changes
29-May-2008	1	Initial release.
24-Sep-2013	2	Updated Disclaimer.