TriCore™ AURIX™ 家族

32 位

使用 GTM(通用定时器模块)的发动机位置驱动程序

AP32212

应用笔记

V1.2　2014-05

微控器

版本 2014/05

出版发行：
英飞凌科技公司
中国，上海

免责条款

本应用笔记中给出的信息仅作为实现英飞凌器件的建议，不得被视为英飞凌器件的任何特定功能、条件或质量作出的任何说明或保证。此应用笔记的接受者必须在实际应用中判定此种描述的任何功能。英飞凌科技在此否认承担此应用笔记中任何和所有信息相关的任何形式的保证和责任（包括但不限于不侵犯第三方知识产权）。

信息

有关技术、交货条款及条件和价格，请与您最近的英飞凌科技公司办事处联系。（www.infineon.com）。

警告

由于技术要求，组件可能含有危险物质。如需相关型号的信息，请与您最近的 英飞凌科技公司办事处联系。如果可能合理地预期此类组件的故障会导致生命支持器件或系统发生故障或影响该器件或系统的安全性或有效性，则英飞凌科技公司提供的组件仅可用于获得英飞凌科技公司明确书面批准的生命支持器件或系统。生命支持器件或系统的目的是植入人体或支持和/或保持并维持和/或保护生命。如果出现故障，则可能危及使用者或他人的人身安全。

文献修订史

日期	版次	修订人	修订详情
06-08-2012	V1.1	Alfredo Baratta	Added paragraph 3.6 - Relative
19-10-2012	V1.2	Alfredo Baratta	Added paragraphs to chapter 4:Synchronization process, VRS Basic Check and Performance, Hardware

Trademarks

TriCore™ 是英飞凌科技公司所属商标。

请留下您的宝贵建议

您是否认为本文档中的任何信息存在错误，含糊不清或遗漏？您的宝贵意见和建议将帮助我们持续不断地改进文档质量。请将您的建议（请注明文档的索引号）发送至：

ctdd@infineon.com

目录

1 导言

发动机管理系统底层驱动程序,包括了曲轴和凸轮轴管理系统的所有功能。在设计过程中，它要求在使用英飞凌 AURIX™家族系列中引进的新一代通用定时器模块（GTM)时采用新的设计理念以便优化系统性能。

1.1 对读者说

请注意该文件面向在发动机位置管理领域已有一定经验的用户，即已经具备了一定专业知识的读者。

1.2 范围和目的

该文件涵盖了实现采用 AURIX™家族微控器的发动机位置底层驱动程序所需的 GTM 硬件和软件架构的各个方面。

该文件描述了顶层设计和驱动程序设计所需的思想，包括主要的架构决策。这样，就为开发组所有的成员提供了一个涉及设计，实现，验证和集成活动的通用框架。此外，在产品生命周期的部署和维护阶段，该文件还充当了知识转移的资料来源。

1.3 参考文献

[1] AURIX™ User Manual TC27x V1.0 2012-06 (or later)

[2] TC27x Errata Sheet Step AA V1.0 (or later)

2 发动机位置驱动程序概览

2.1 系统架构

对于基本的发动机位置驱动程序的实现，通用定时器模块 GTM 扮演着至关重要的角色。尤其是对于 DPLL(数字锁相环）这个模块，它需要与多个其它 GTM 子模块协同运行，以便能够实现一个较低 CPU 负荷并仍能管理曲轴（凸轮轴）的系统。

接下来将对整个系统和所涉及的所有子模块进行基本的描述。

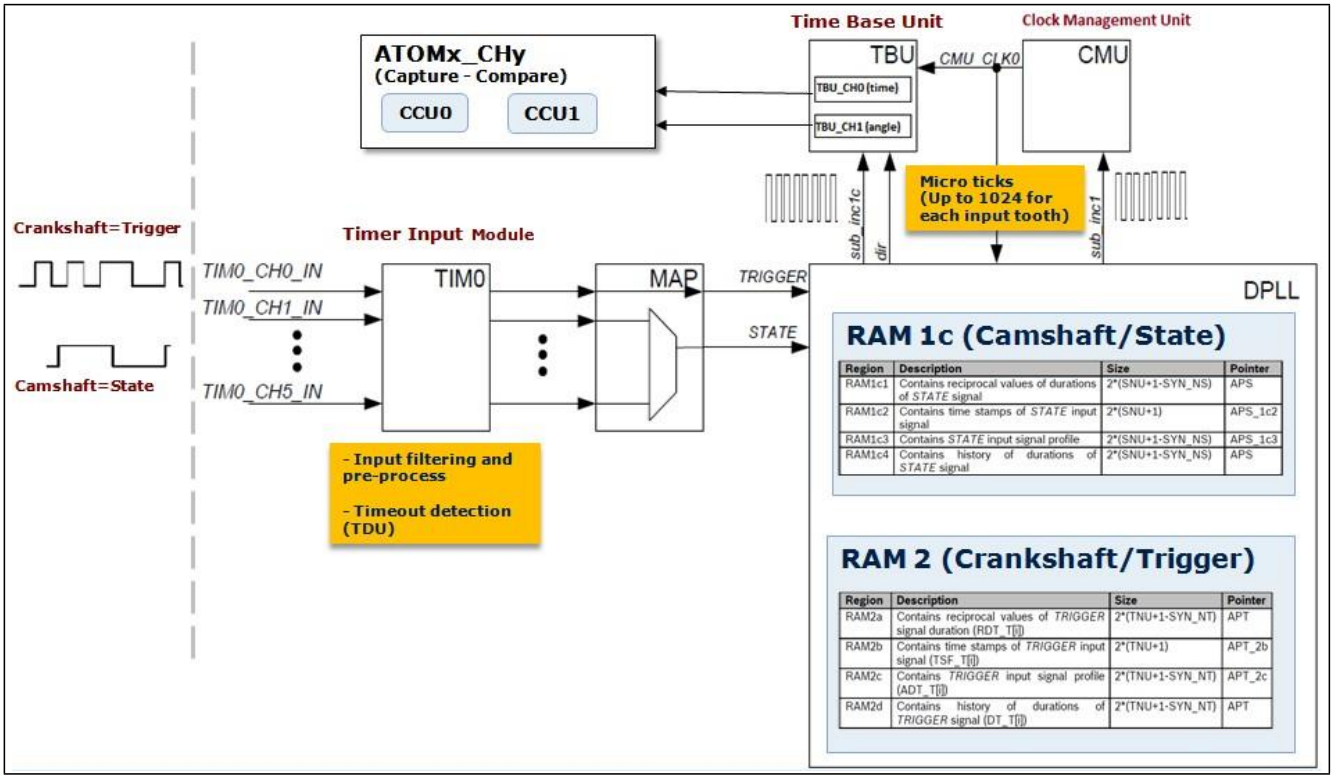

图 1 基于 GTM 的发动机位置驱动程序：系统架构

2.2 系统概览

GTM DPLL 可以通过生成高频微齿信号来增加一个或多个输入信号的分辨率。

DPLL 可被配置为能为每个输入的信号周期（齿）生成高达 1024 个的微齿。在此处我们所提及的这些输入信号，即 TRIGGER 和 STATE，必须被接至子模块 TIM0（定时器输入模块）。

原则上，有两种可行方案可为 DPLL 提供输入信号。

第一种可行方案，如上图所示，通过 MAP(输入映像）子模块将 TIM0 通道 0 接至 DPLL 的 TRIGGER（曲轴）输入信号。

第二种可选方案则是将凸轮轴信号接至其余 5 个 TIM0 输入通道（从通道 1 至通道 5）中的一个，并且通过在 MAP 中执行的一个多路复用器接至 DPLL 的 STATE 输入信号。

为了生成微齿，DPLL 需要知道输入信号的时序状态。而该时序状态可从**时基单元（TBU)**所提供的时间戳获取。

ATOM(连接先进路由单元(ARU)的定时器输出模块),可被用于触发"角度事件"或者是在驱动程序启动期间实现延时等待。

以下章节更加详细地描述了系统模块。

3 使用 GTM

TBU 通过时钟信号生成时间戳，该时钟信号起源于**时钟管理单元（CMU)**内部。

正如输出一样，DPLL 生成不同的信号，这些信号可被分为以下两组：

- 有关 **TRIGGER**（曲轴)：
  - sub_inc1
  - sub_inc1c
- 有关 **STATE**（凸轮轴)：
  - sub_inc2
  - sub_inc2c

sub_inc1 (sub_inc2)可被发送到位于 CMU 内部的一个特定时钟，sub_inc1c (sub_inc2c) 可以控制位于 TBU(CH1 和/或 CH2)内部的计数器。

为了生成微齿并进行其它计算，DPLL 采用了内置 ALU（算数逻辑单元）以及可保存数据以便用于计算的外部 RAM（随机存取存储器），。

- RAM1a
  -用于动作计算（位置减时间请求）。“动作计算”指那里可能有由先进路由单元（ARU)控制的请求，其中未来时间和角度点必须由 DPLL 代表输入信号来进行预测。
- RAM1bc
  -为 DPLL 保存计算参数，以用于计算 TRIGGER 和 STATE 的输入。同时，他还保存 STATE 输入特征值。
- RAM2
  -用于储存 TRIGGER 输入信号特征。

3.1 时钟管理单元（CMU)

CMU 为 GTM 内置计数器提供时钟预分频器。CMU 生成不同的时钟：

- 外设的(CMU_ECLK)
- 完全可配置的(CMU_CLKs)
- 预分频的(FXCLK)

注释：以这里我们所说的发动机位置驱动程序的执行为例，仅仅只需要 CMU_CLKs。

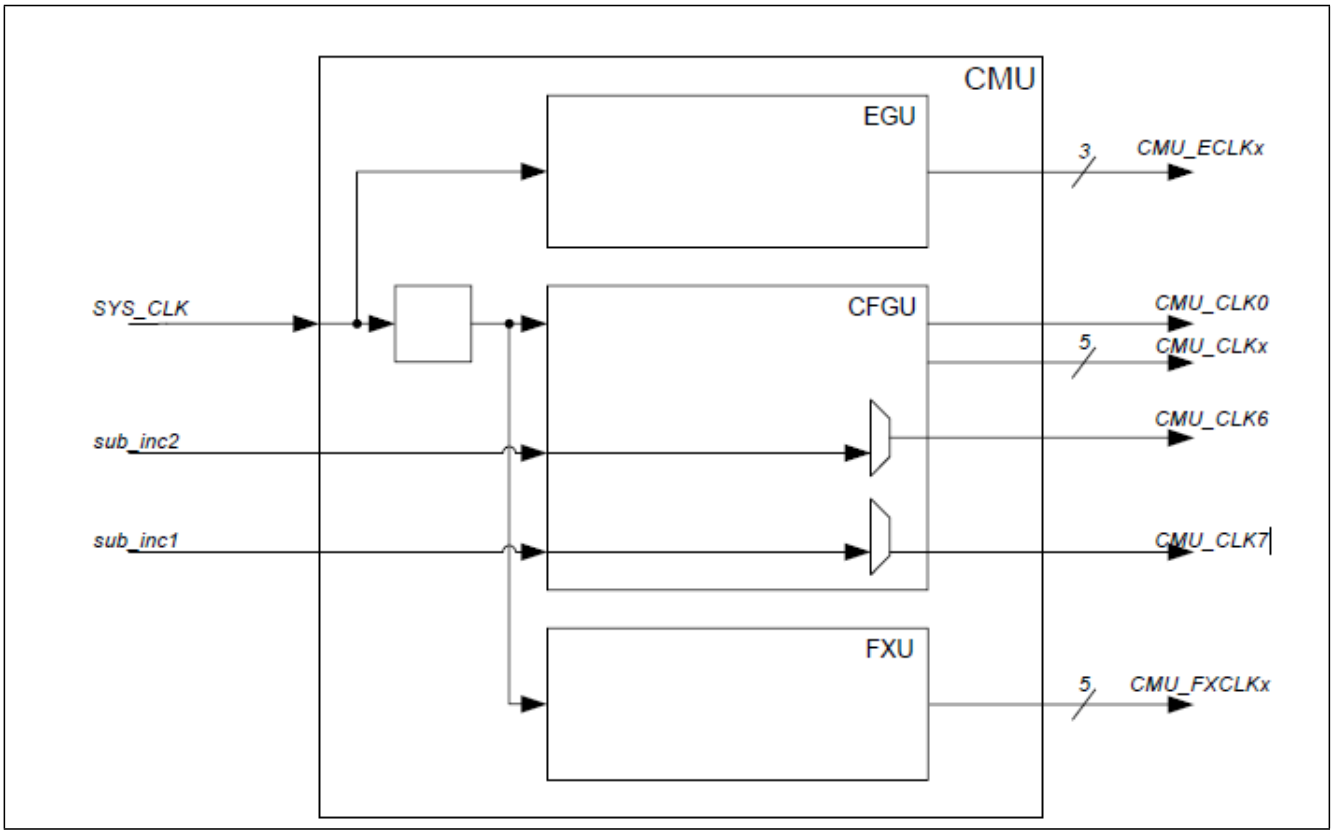

图 2　CMU 框图

用户必须在多个 CMU 预分频器之间做出选择,并且对他们进行合理配置。这一点非常重要，因为 TBU 需要一个时钟以生成时间戳，有了这个时间戳，就可将 TIM0 输入信号特征化，并将其发送到 DPLL。因此，预分频器的设置决定时间戳最小单位，并且因为被选的时间戳时钟也被馈送到 DPLL，所以该预分频器还决定微时钟的分辨率。

第二个重要的时钟预分频器是连接到时钟信号 CMU_CLK0 的那一个。

如果 CMU_CLK0 被配置且有必要，DPLL 总是使用它的频率生成缺失的微时钟或者是校正微时钟。

除了这两条时钟信号线外，也可使用 CMU_CLK6 和 CMU_CLK7 时钟信号将 DPLL 生成的微时钟 sub_inc2 和 sub_inc1 分配到其它的 GTM 子模块。

3.2　时基单元（TBU）

时基单元为 GTM 提供共同的时基。

典型情况下，对于一个发动机管理系统而言，时间戳 TBU_TS0 （其中 TS 代表有效沿的时间戳）提供 24 位时间相关的信息，其中 TBU_TS1 (TS2) 为系统提供发动机的角度。

图 3 显示了 TBU 框图。对于分辨率，TBU_TS0 时间戳是 27 位宽。

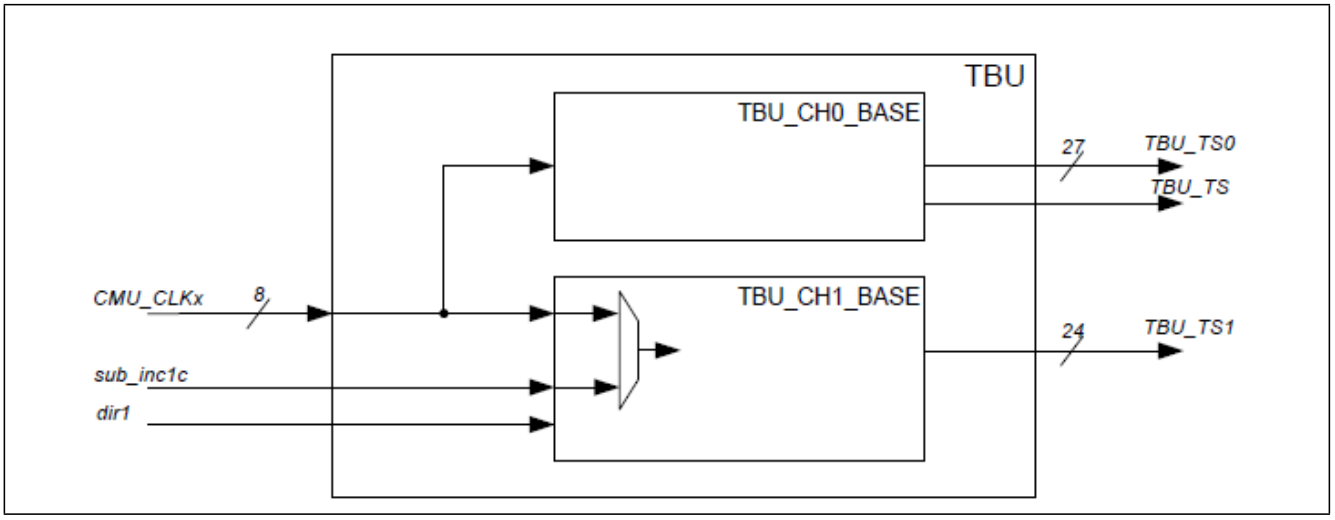

图 3 TBU 框图

请注意对于通道 1 的时基，应通过选择 TBU 通道控制寄存器的 CH_MODE 位选取 sub_inc1c (或用于凸轮轴的 sub_inc2c)输入时钟。可通过选择正向/反向计数器模式来实现上述选取。

注释：虽然可以通过 CMU_CLK7 选择 sub_inc1 时钟，但是不建议将该时钟作为输入，因为它并未准确地反应发动机的输入性能。

3.3 定时器输入模块（TIM0）

定时器输入模块 0 (TIM0)专用于为 DPLL 进行输入信号的采样及预处理。

TIM 包括了 8 个通道，并且在每个通道内都有可配置的滤波器（FLT)，超时检测（TDU)，和信号处理单元。

图 4 是一个 TIM 通道的原理图。

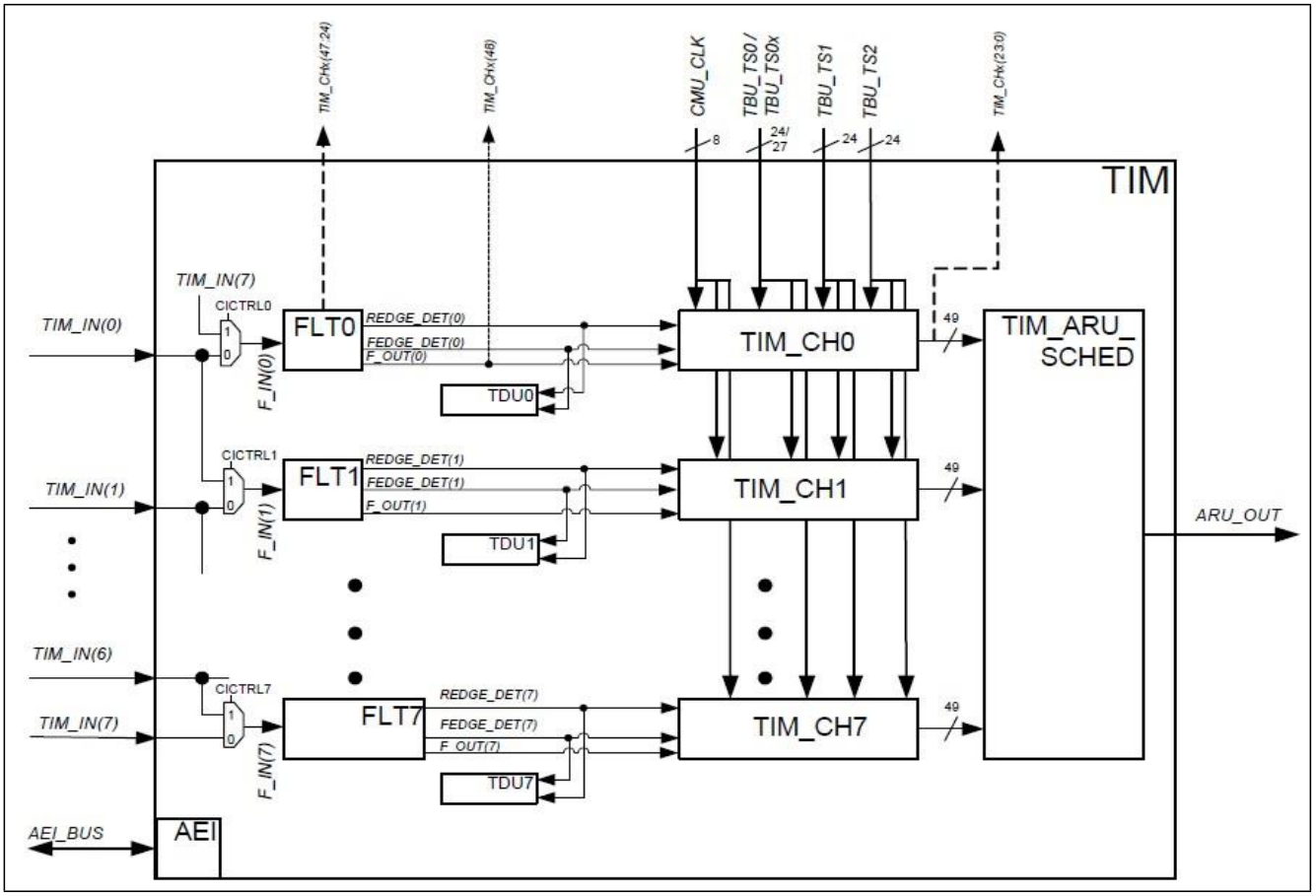

图 4　TIM 通道框图

一般来说,每个 TIM 通道都可被配置为能在不同的模式下工作,尤其是在以下模式:

- TPWM
  -TIM 脉宽调制(PWM)测量模式,以测量占空比和周期
- TIEM
  -TIM 输入事件模式,以捕获上升沿和/或下降沿的时间戳
- TIPM
  -TIM 输入预分频模式,以检测一个特定的沿

一个 TIMx_CH 的输出是 49 位宽:

- 24 位用于沿的时间戳
- 24 位用于滤波器的信息
- 1 位用于信号电平

这个 49 位宽的信号 TIMO_CH 可通过 MAP 的子模块被接至 DPLL。

为了能正确工作，DPLL 需要在相应的 TIM 通道内进行一些特定的设置：

- 滤波设置
- 在 TIM 通道内对时间戳格式进行取样

对于操作而言，DPLL 需要每一个输入沿（齿）的时间戳。

因此，应将相应的 TIMO 通道配置成在每一个输入沿上捕捉相关的时间戳（TIEM － TIM 输入事件模式)。

在 TIMO_CH［x］_CTRL 寄存器内部，有 3 个配置位用于设置（见图 5）。

ISL (Bit 14)	Ignore signal level: This bit has to be set, to force the TIM0 channel to react on each incoming edge.
GPR0_SEL (Bit 9:8)	Selection for GPR0 register: These two bits have to be set to "00" to sample the TBU_TS0 in the register. This value is transmitted to the DPLL when a valid edge occurs.
TBU0_SEL	TBU_TS0 bits input select: This bit has to be set according to the requested time stamp resolution.

图 5　TIM0_CH[x]_CTRL:寄存器设置

3.3.1　TIM0 滤波器配置

TIM0 子模块具有 3 种通用的滤波模式，这 3 种模式可根据每个沿进行单独配置。DPLL 可以按照滤波阈值分别更正每个沿的输入信号时序。

对于这些滤波更正，我们进行了一些必要的假设：

Filter mode	Meaning of FLT_RE	Meaning of FLT_FE
Immediate edge propagation	Acceptance time for rising edge	Acceptance time for falling edge
Individual de-glitch time (up/down counter)	De-glitch time for rising edge	De-glitch time for falling edge
Individual de-glitch time (hold counter)	De-glitch time for rising edge	De-glitch time for falling edge

图 6　TIM 滤波模式

图 7 是沿的立即传输模式（同时针对上升沿和下降沿）示例。

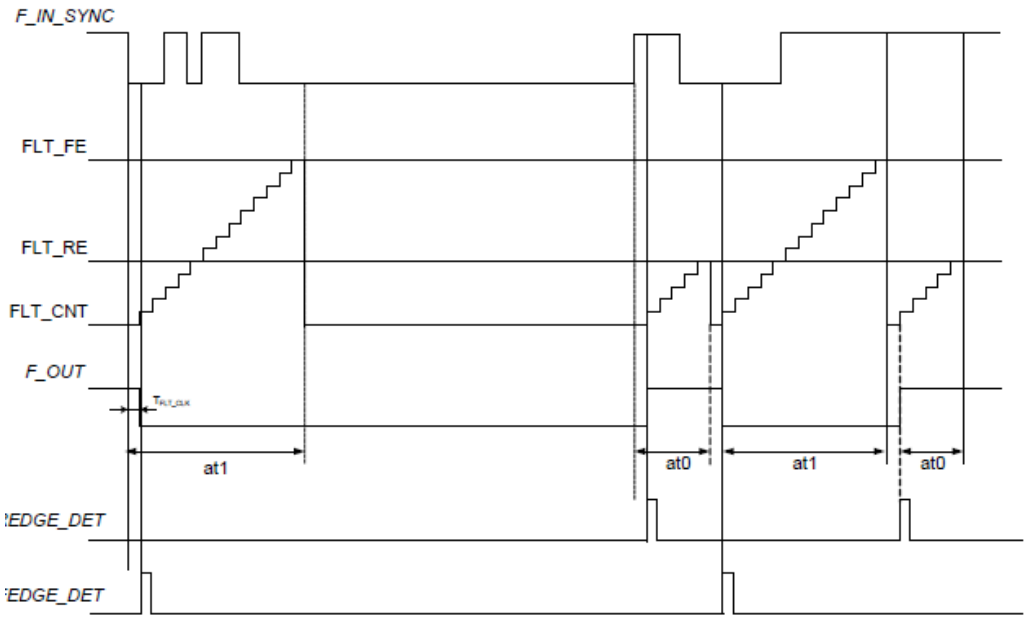

图 7　TIM 滤波：立即传输模式（RE/FE)示例

通过设置 DPLL_CTRL_0 寄存器的 IDT(输入延迟 TRIGGER)位和 IDS(输入延迟 STATE)位,就在 DPLL 内使能了滤波更正。与 IFP（输入滤波位置）位一起，就有可能确定 TIM 滤波器是代表 CMU_CLK 时钟计数，还是代表 sub_inc 时钟计数。

如果是针对 CMU_CLK 时钟计数基准进行的滤波更正，很重要的一点是要将滤波输入时钟配置成与 TBU_TS0 时间戳时钟一致。如果没有这样做，这两个频率无法相互匹配，且在 DPLL 内更正的时间戳将传递出不正确的结果。

另外，还要注意一个要点，即在沿的立即传输模式里配置 TIM 滤波器时，必须停用 DPLL 里的滤波更正。如果未停用该滤波更正，DPLL 将使用滤波停用窗口，同时也被编程为滤波阈值，以更正时间戳。而这会产生一个新的时间戳，该时间戳在立即传播沿之前就会被定位到。

3.3.2 TIM 超时单元(TDU)

超时检测单元(TDU)负责 TIM 输入信号的超时检测。

TIM 子模块的每一个通道都有各自的超时检测单元 (TDU, 8-bits)，在这里，可以在相应通道的过滤输入信号上建立一个超时事件。

TDU 架构如图 8 所示。

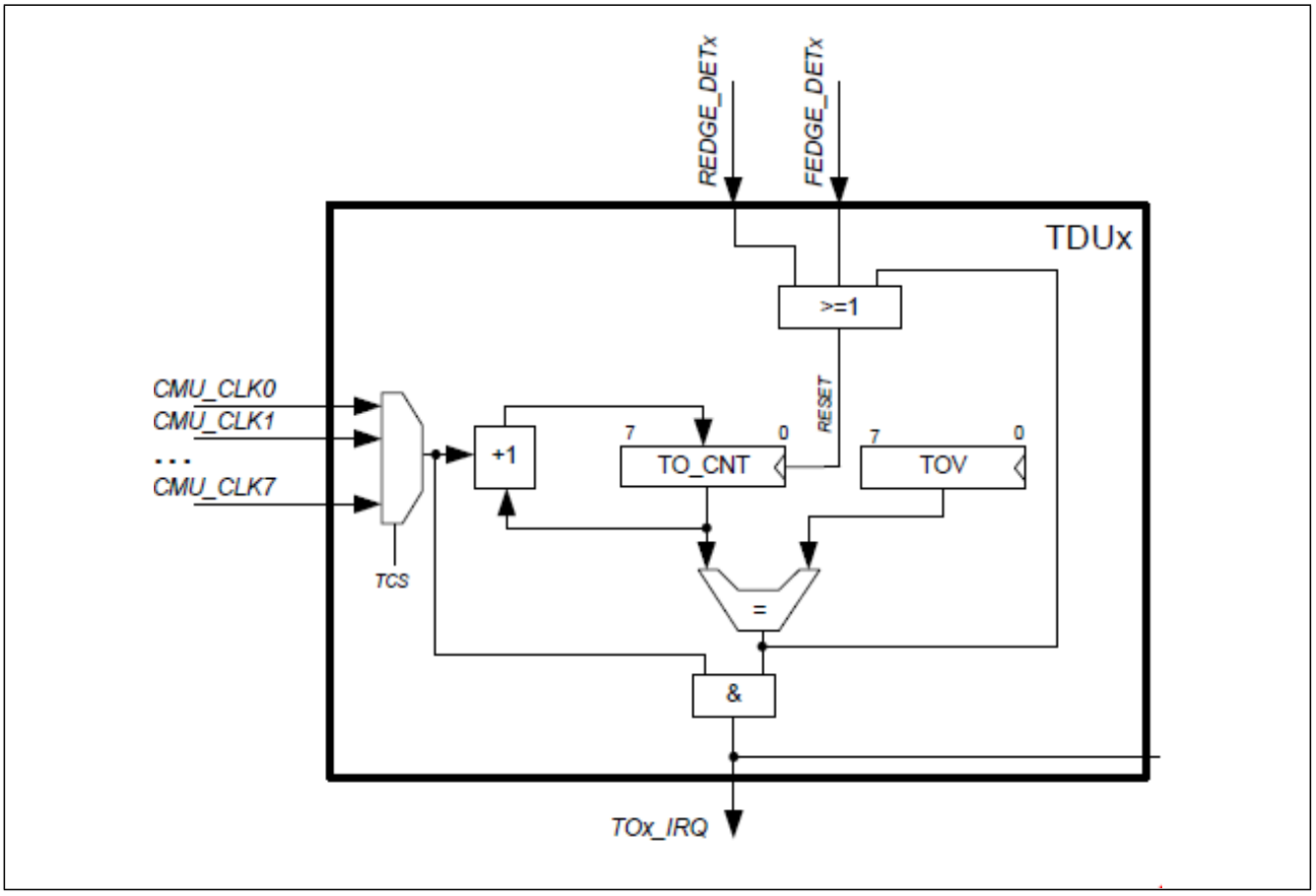

图 8 TDU 超时检测单元

采用从 TIM[i]_CH[x]_TDU 寄存器的 TCS(超时时钟选择)位段选择的特定 CMU_CLKx 输入信号的分辨率，就有可能检测出超时。单个超时值代表所选输入时钟信号有多少个时钟周期，且必须将它保存在 TIM 通道 x (x:0...7)超时值寄存器 TIM[i]_CH[x]_TDU 的 TOV(超时值）字段。

可通过下列公式计算出准确的超时值 TDU：

- TDU=(TOV+1)* T_{CMU_CLKx}

3.4 DPLL（数字锁相环）模块

3.4 概览

DPLL 是 GTM 一块配置性高，专用的子模块，它可以代表一个或多个低频输入信号生成高频微齿信号。

DPLL 可生成一个取决于 TRIGGER 输入(曲轴）的微时钟信号和一个取决于 STATE（凸轮轴）输入的微时钟信号，其中 TRIGGER 和 STATE 相互关联。

另外一种可能就是从两个独立的 TRIGGER 和 STATE 输入信号中生成两个独立的微时钟信号。

很多计算都是在 DPLL 内部执行，且这些计算需要大量的局部变量。因为为了获取这些局部变量而执行寄存器的相关开销高，因此尽可能将局部变量存储于 RAM 内。因此，在计算期间，就需要 DPLL 能够访问 RAM。

可将 DPLL 划分为多个功能模块（图 9）。有关 DPLL 更多细则，请参阅 GTM 技术规格书。

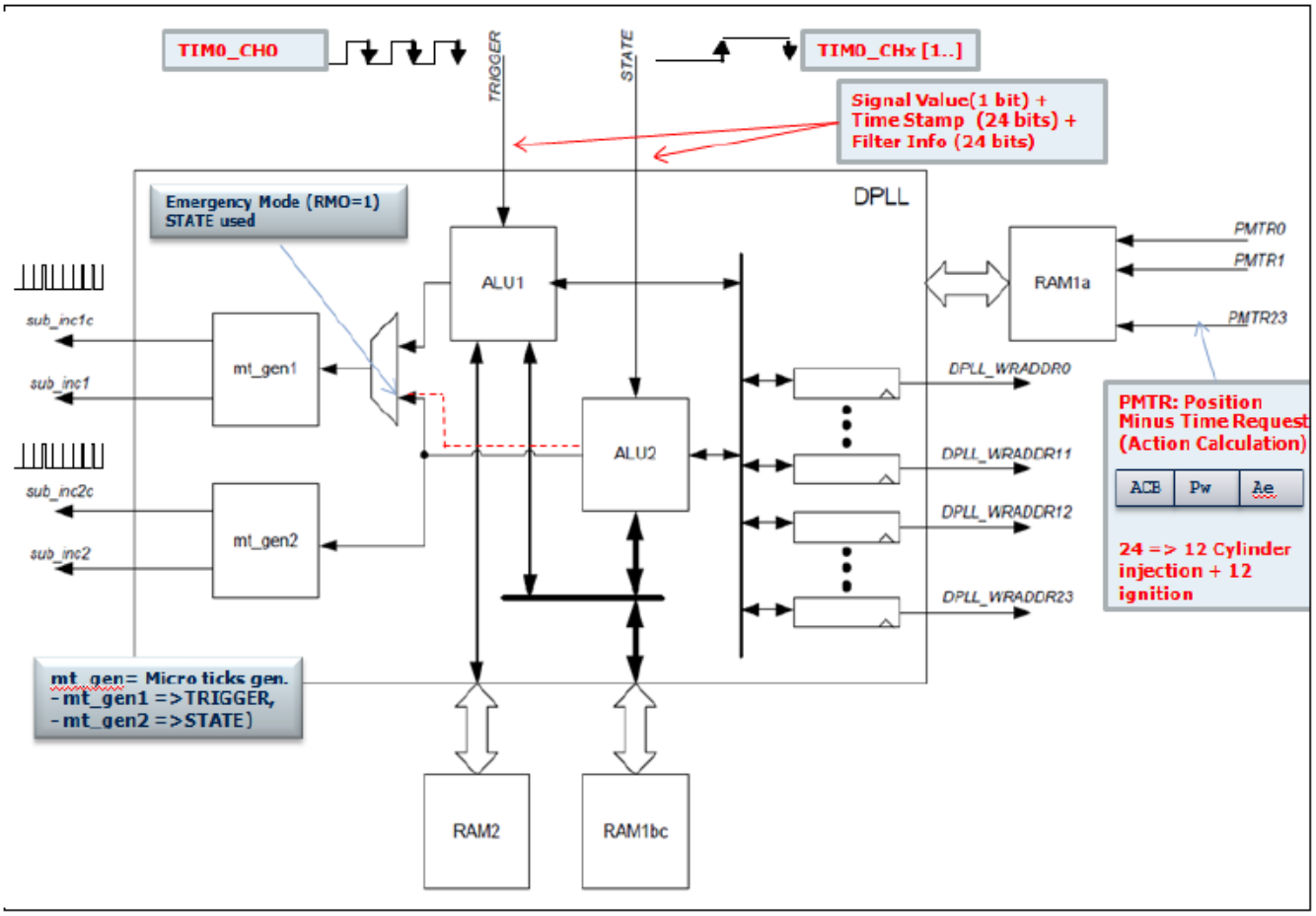

图 9 DPLL 框图

由两个独立的微时钟生成单元 mt_gen1，和 mt_gen2 在 DPLL 内部生成微齿。

[1] 最多控制两个同步电机（SMC=1）

由于这两个输入信号 TRIGGER 和 STATE 可以各自独立地到达 DPLL,因此就执行了两条独立的数据路径和 ALU 子单元,从而为微齿发生器计算参数,并为其它 GTM 系统计算行动值。

局部变量，行动参数和系统特征数据分别位于三个独立的 RAM 模块：

- RAM1a
  -保存行动计算所需的参数[1]
- RAM1bc
  -保存局部变量和 STATE 信号特征化数据
- RAM2
  -保存 TRIGGER 输入信号特征化数据

当将它们用于发动机管理应用上时,DPLL 可在如下两种模式下操作:

- 正常模式(RMO=0)
  -曲轴(TRIGGER)信号被用于生成 *sub_inc1* 和 *sub_inc1c* 微齿。
- 紧急模式(RMO=1)
  -凸轮轴(STATE)信号被用于生成 *sub_inc1* 和 *sub_inc1c* 微齿。

3.4.2 DPLL 微齿的生成

用于 TRIGGER 和 STATE 输入信号的微时钟由两个子单元 mt_gen1 和 mt_gen2 生成，并通过后面这四条信号线分配：*sub_inc1*, *sub_inc1c*, *sub_inc2* 和 *sub_inc2c*。

微齿由 24 位加法器生成，每当 24 位累加器寄存器溢出时，它就生成一个时钟计数。mt_gen1 子单元原理如图 10 所示。

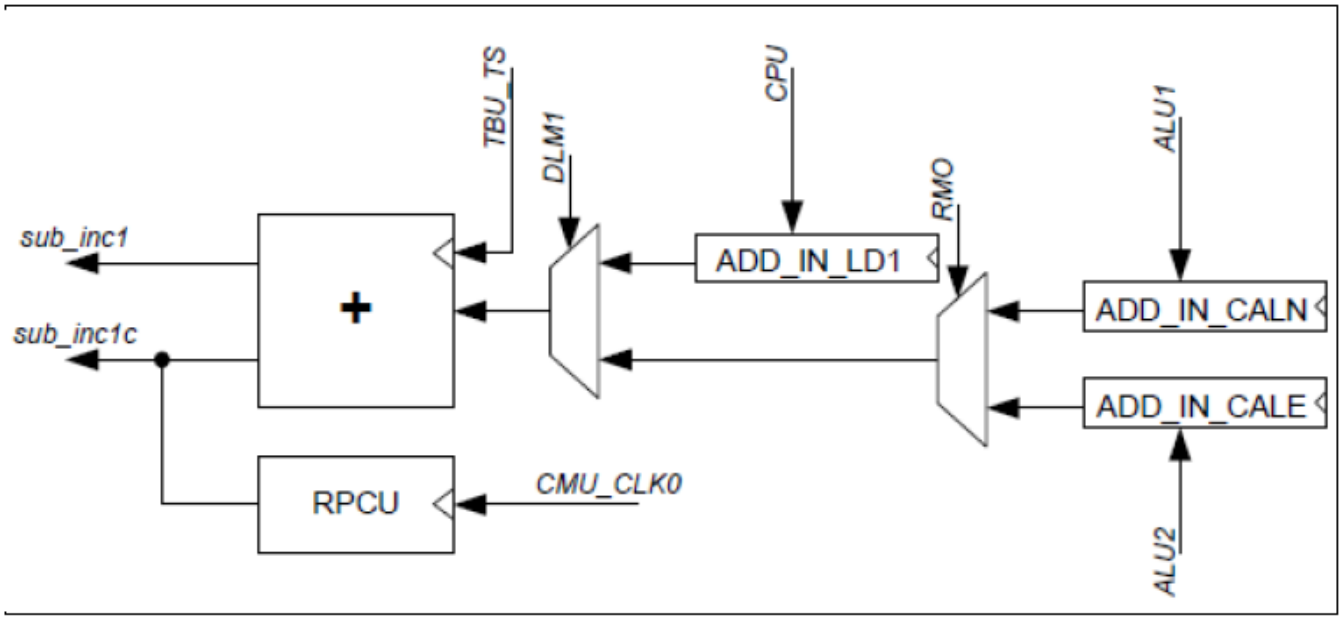

图 10 mt_gen1 子单元原理

加法器有两条输入路径：

- 其中一条路径可通过将加法器的值加载到 ADD_IN_LD1 寄存器而被 CPU 控制。
  该路径可通过 DPLL_CTRL_1 寄存器里的 DLM1 位使能。
- 另外一条路径被 DPLL 内部逻辑控制，其中两个 ALUs 代表他们各自的输入信号分别计算两个加法器的值。ALU1 计算 ADD_IN_CALN， 且 ALU2 计算 ADD_IN_CALE。

[1] PMTR (Position Minus Time Request)., 本文件不涉及这部分。

CPU 可以通过设置 DPLL_CTRL_1 寄存器(紧急模式/正常模式)里的 RMO 位控制使用哪一个加法器值(二选一)。
对于加法器值的计算，重要的一点是，来自 TBU 通道 0（与时间相关）的时间戳时钟 TBU_TS 给该加法器计时。
可以使用下列公式计算在每一个输入的有效沿上所生成的微齿数：

$$ADD_{INCAL} = \frac{[(MLT+1)*SYN_T + MP + PD_{store} + MPVAL1] + 0.5}{[(DT_{Tactual} + MED_T)*QST_T[p+q-1]]*SYN_T}$$

(MLT+1) = 1..1024 => number of microticks wanted for each tooth
SYN_T =virtual increments of last tooth (1 or 3),(stored into ADT_T profile)
MP=Missing pulses (to be added)
PD_Store=Phisical Deviation factor (stored into ADT_T profile)
MPVAL1 = Correction factor (from CPU)

图 11 ADD_IN_CAL 公式

要强调的一点是，在运行时，CPU 可以通过更新 MPVAL1 寄存器调整生成的微齿数。所有其它的参数要么是在驱动器初始化阶段被设置好，要么就是由 DPLL 自动计算出来。
而另外一个输入时钟 CMU_CLK0 则在某些微齿缺失或者是检测到方向的改变时，用于生成快脉冲。
因此，必须仔细挑选 TBU 通道 0 输入时钟和 CMU_CLK0 时钟。同样，如图 12 的描述，生成的输出微时钟之间也有差异。

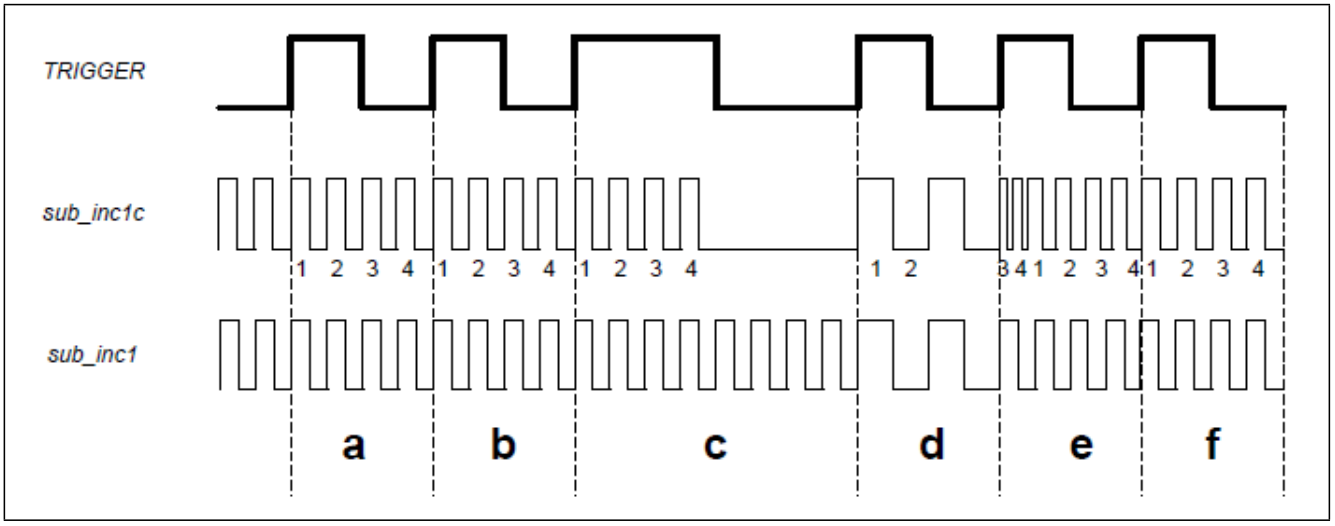

图 12 补偿的和未补偿的微时钟的生成

该示例表明了 sub_inc1c 和 sub_inc1 信号加速或减速时，会发生什么。
在上述例子中，我们将 DPLL 配置成在 TRIGGER 输入信号上的每一个输入 tick 上（从上升到上升沿是一个 tick）生成 4 个微时钟。这 4 个微时钟是为了间隔 a 和间隔 b 而生成的。

DPLL 为间隔 c 预测了相同的加法器值，但是其输入信号频率却降低了（减速）。

当 DPLL 在自动结束模式下编程时，sub_inc1c 在 4 个 tick 生成后输出 tick 结束。这个自动结束模式是由 DPLL_CTRL_1 寄存器里的位 DMO 控制的。对于下一个间隔，又一次计算了加法器值，从而使 sub_inc 信号频率更小。现在，由于输入信号加速，因此没有生成足够的微齿。

对此，DPLL 采取了两种可行的补偿措施。一种可行措施是为下一个间隔平均分配 6 个微齿。另一种可行措施是快速生成两个微齿，并在下一个间隔平均生成 4 个常规的微时钟。上述所需的行为可通过 DPLL_CTRL_1 寄存器里的 COA 位来配置。

如果要快速生成微时钟，则 CMU_CLK0 被用作时钟计时单位频率。

微齿的生成适用于信号 sub_inc1c。如图 12 所示，总是以从最后一次增长周期计算出的频率生成 sub_inc1。因此，sub_inc1 时钟计时单位不反应 TRIGGER 输入信号的物理位置。

3.4.3 DPLL RAM 组织

DPLL 有三个相关的 RAM 功能块，DPLL 自身和 CPU 可以独立地对其进行访问：

- RAM1a
  -用于行动计算(PMTR)。当 DPLL 被停用时，仅能被 CPU 访问。
- RAM1bc
  -在 DPLL 计算期间用于储存局部变量和 STATE 输入数据。
- RAM2
  -用于储存 TRIGGER 输入信号相关的数据。

DPLL 通过内部指针定位 RAM 内的数据。

除了这三个不同的物理 RAM，每一个 RAM 都被划分成若干个区域，且每一个区域都有其自己的 RAM 指针。了解这些指针对于运行 DPLL 非常重要。

下面的表格描述了 RAM2(RAM1bc)区域和相关的指针。

Region	Description	Size	Pointer
RAM1c1	Contains reciprocal values of durations of *STATE* signal	2*(SNU+1-SYN_NS)	APS
RAM1c2	Contains time stamps of *STATE* input signal	2*(SNU+1)	APS_1c2
RAM1c3	Contains *STATE* input signal profile	2*(SNU+1-SYN_NS)	APS_1c3
RAM1c4	Contains history of durations of *STATE* signal	2*(SNU+1-SYN_NS)	APS

图 13 RAM1bc 区域/指针（STATE）

Region	Description	Size	Pointer
RAM2a	Contains reciprocal values of *TRIGGER* signal duration (RDT_T[i])	2*(TNU+1-SYN_NT)	APT
RAM2b	Contains time stamps of *TRIGGER* input signal (TSF_T[i])	2*(TNU+1)	APT_2b
RAM2c	Contains *TRIGGER* input signal profile (ADT_T[i])	2*(TNU+1-SYN_NT)	APT_2c
RAM2d	Contains history of durations of *TRIGGER* signal (DT_T[i])	2*(TNU+1-SYN_NT)	APT

图 14 RAM2 区域/指针(TRIGGER)

重要的是要明白 DPLL 自动操控这些指针。然而，CPU 必须监视 TRIGGER(STATE)输入信号，并且根据与输入对应的实际档案位置（DPLL 同步）设置 APT_2c (APS_1c3)指针。

DPLL 设置 DPLL_STATUS 寄存器里的 LOCK(1)位，以发送如下信号：DPLL 可以按照在 RAM 区域 RAM2 (RAM2c)里用户给出的存储信息(ADT_T[i])判断输入信号。

B::d.dump 0xf012c400

address	0	4	8	C
D:F012C400	00324B80	00327C54	0032AD28	0032DDFC
D:F012C410	00330ED0	00333FA4	00337078	0033A14C
D:F012C420	0033D220	003402F4	003433C8	0034649C
D:F012C430	00349570	0034C644	0034F718	003527EC
D:F012C440	003558C0	00358994	0035BA68	0035EB3C
D:F012C450	00361C10	00364CE4	00367DB8	0036AE8C
D:F012C460	0036DF60	00371034	00374108	003771DC
D:F012C470	0037A2B0	0037D384	00380458	0038352C
D:F012C480	00386600	003896D4	0038C7A8	0038F87C
D:F012C490	00392950	00395A24	00398AF8	0039BBCC
D:F012C4A0	0039ECA0	003A1D74	003A4E48	003A7F1C
D:F012C4B0	003AAFF0	003AE0C4	003B1198	003B426C
D:F012C4C0	003B7340	003BA414	003BD4E8	003C05BC
D:F012C4D0	003C3690	003C6764	003C9838	003CC90C
D:F012C4E0	003CF9E0	003D8C5C	0050BA50	0050EB24
D:F012C4F0	003DBD30	003DEE04	003E1ED8	003E4FAC
D:F012C500	003E8080	003EB154	003EE228	003F12FC
D:F012C510	003F43D0	003F74A4	0028C218	0028F2EC
D:F012C520	002923C0	00295494	00298568	0029B63C
D:F012C530	0029E710	002A17E4	002A48B8	002A798C
D:F012C540	002AAA60	002ADB34	002B0C08	002B3CDC
D:F012C550	002B6DB0	002B9E84	002BCF58	002C002C
D:F012C560	002C3100	002C61D4	002C92A8	002CC37C
D:F012C570	002CF450	002D2524	002D55F8	002D86CC
D:F012C580	002DB7A0	002DE874	002E1948	002E4A1C
D:F012C590	002E7AF0	002EABC4	002EDC98	002F0D6C
D:F012C5A0	002F3E40	002F6F14	002F9FE8	002FD0BC
D:F012C5B0	00300190	00303264	00306338	0030940C
D:F012C5C0	0030C4E0	0030F5B4	00312688	0031575C
D:F012C5D0	00318830	00321AAC	006C65EC	005C5CD4

图 15　DPLL TSF_T[i]信息转储

注释：TSF 代表 TRIGGER 事件的时间戳字段。

3.4.4　TRIGGER 和 STATE 输入信号特征

为了合理操纵（及预测）DPLL，CPU 必须特征化 RAM1bc 和 RAM2 里存储区域里的 TRIGGER 和/或 STATE 输入信号。这些存储区域指的是 **RAM1c3** 和 **RAM2c**。

当已知输入信号位置时（同步， 3.4.5)，CPU 必须为这些区域校正关联的指针。同步之后，“DPLL”被锁定在输入信号上，且可以执行所有所要求的计算。

图 16 是一个档案示例。

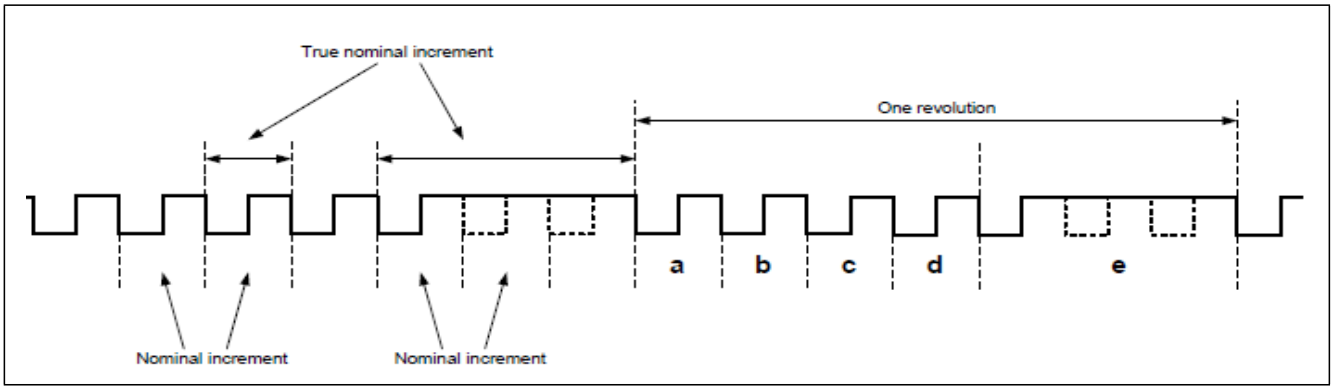

图 16　DPLL 输入特征示例

该图表示了一个齿轮的两圈旋转,其中有 2 个齿缺失。其中下降沿被定义为有效的触发沿，该沿定义一个有效的齿。有效沿在 DPLL_CTRL_1 里定义。

这些有效沿定义真实的实际齿增加。然而，为了将这些特征生效，必须将齿圈划分成等距的额定增加。

应用按照实际齿增加和他们的周期在 RAM 区域 RAM1c3 (STATE) 和/或 RAM2c（TRIGGER)里定义特征。

图 17 显示了在 DPLL RAM 里详细的输入特征。

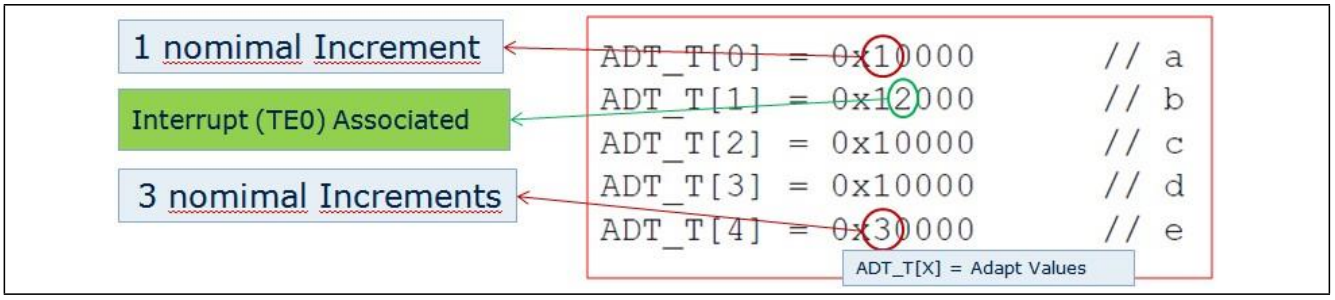

Figure 17　RAM2 Trigger ADT_T Entries

调整值 ADT_T[x]，代表齿轮的存储数据。对于真实的标称增长，标称增长的数量被储存在 NT 位字段（ADT_T[i]寄存器）。

对于上述示例中的 ADT_T 0 到 3，有一个标称增长。对于 ADT_T[4]，有三个标称增长。

对于真实的标称增长 ADT_T[1]，规定了一个用户定义的中断，在检测到齿时（如果被使能）会产生一个中断(TINT) 。

总共，最多会有 5 次与任意输入齿关联的用户特定中断。一个齿的物理偏移（PD)可以在 RAM 地址的低 13 位被指定出来。

PD 值代表将被添加或减去的若干微齿（如校正系数）。

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Bit	Reserved								Reserved					NT			TINT			PD												

图 18 ADT_T[i] 寄存器条目

3.4.5 DPLL TRIGGER/STATE 同步

同步过程是将输入信号(TRIGGER 和/或 STATE)与"预期"信号相匹配的重要过程,该"预期"信号档案由 CPU 储存在相关 DPLL RAM 内部。

一个指针(APT_2c 用于触发器)指向 ADT_T 数组。ADT_T 数组包含了预期的存储数据（针对 TRIGGER 和 STATE)：

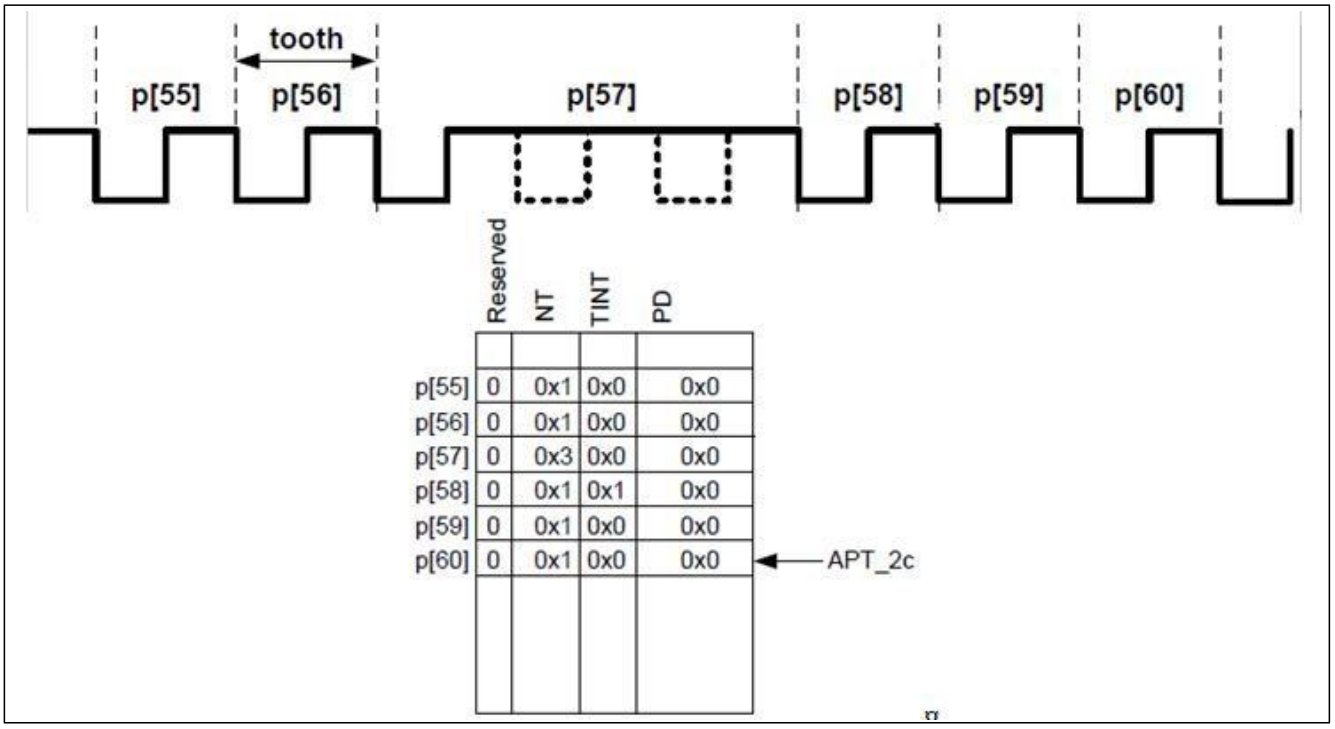

图 19 TRIGGER 指针

在实践中，一旦得知准确的飞轮位置，*CPU* 通过将 *apt_2c 指针设置到一个规定值来实施同步。*

也可以采用另外一种不同的方案执行此同步过程。一种可行的方案是使用 TIM0 模块的 TIM0_NEWVAL_IRQ 中断，测量输入信号的周期并检测空隙（“缺齿”）。

当当前这个齿的周期是前一个齿的两倍时，应用便得知曲轴现在处于间隙，且可以将 DPLL 指针设置为指向间隙之后的下一个齿（如，如果是 60-2 的飞轮，则为 57+3）。

下面给出了一个 60-2 齿飞轮的 TIM0_NEWVAL_IRQ 服务程序：

```
void isr_tim0_newVal(void) {
        TIM0_CH0_IRQ_NOTIFY = 0x1; // disable NOTIFY bit
        actTS = TIM0_CH0_GPR0; // save actual time stamp
        // tooth starts with falling edge
       if (!(actTS & 0x01000000)) { // falling edge detected
            oldTS = newTS;
            oldDiff = diffTS;
           newTS = actTS & 0xFFFFFF; // determine new time stamp
            if (++iter > 1) {          // gap detection after second valid edge
             diffTS = newTS - oldTS;
                  if (oldDiff >= 2 * diffTS) {
                         // gap characteristic for two missing teeth
                        GTM_DPLL_APT_2C.U = (57 + 3);
                         // synchronize DPLL: use first gap + 3 tooth
```

```
                        // enable DPLL sub_inc1c generation
                    DPLL_CTRL_1 = 0x80020032;
                     TIM0_CH0_IRQ_EN = 0x0;
                    // disable TIM0 channel 0 NEWVAL interrupt
                } //gap detected
            } // second valid edge
} // falling edge
```

图 20 代码片段:TIM new_val 中断服务路径 (ISR)

使用 TIM0_NEWVAL_IRQ 中断检测间隙的一个可能同步序列如下图所示：

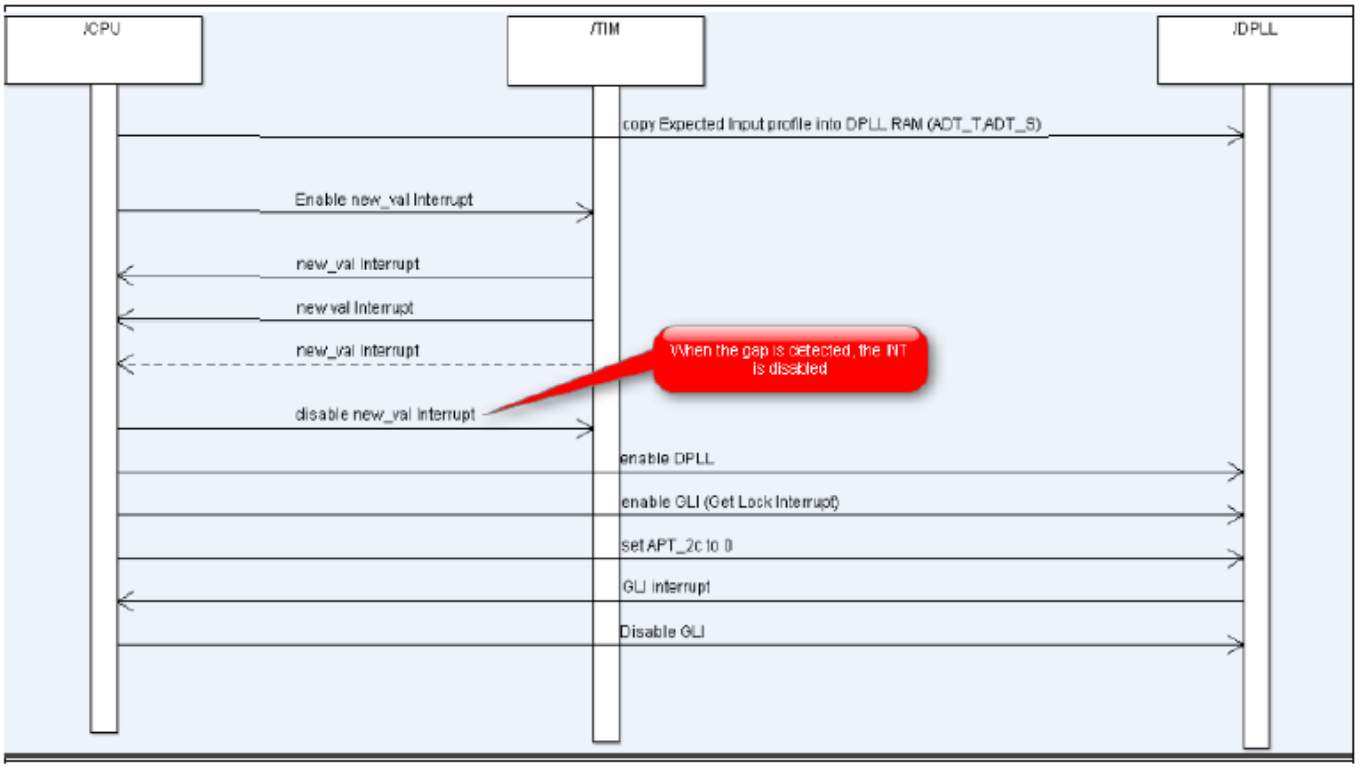

图 21　使用 TIM new_val 中断的 DPLL 同步

同步 DPLL 的另外一个可选方案是使用缺齿的触发中断（MTI），该 MTI 在检测到一个缺失的齿时由 DPLL 所触发。

由于 MTI 只有在同步之后才可靠，因此，在使能 MTI 中断之前，有必要将 APT_2c 存储指针至少 3 次复位到 0。

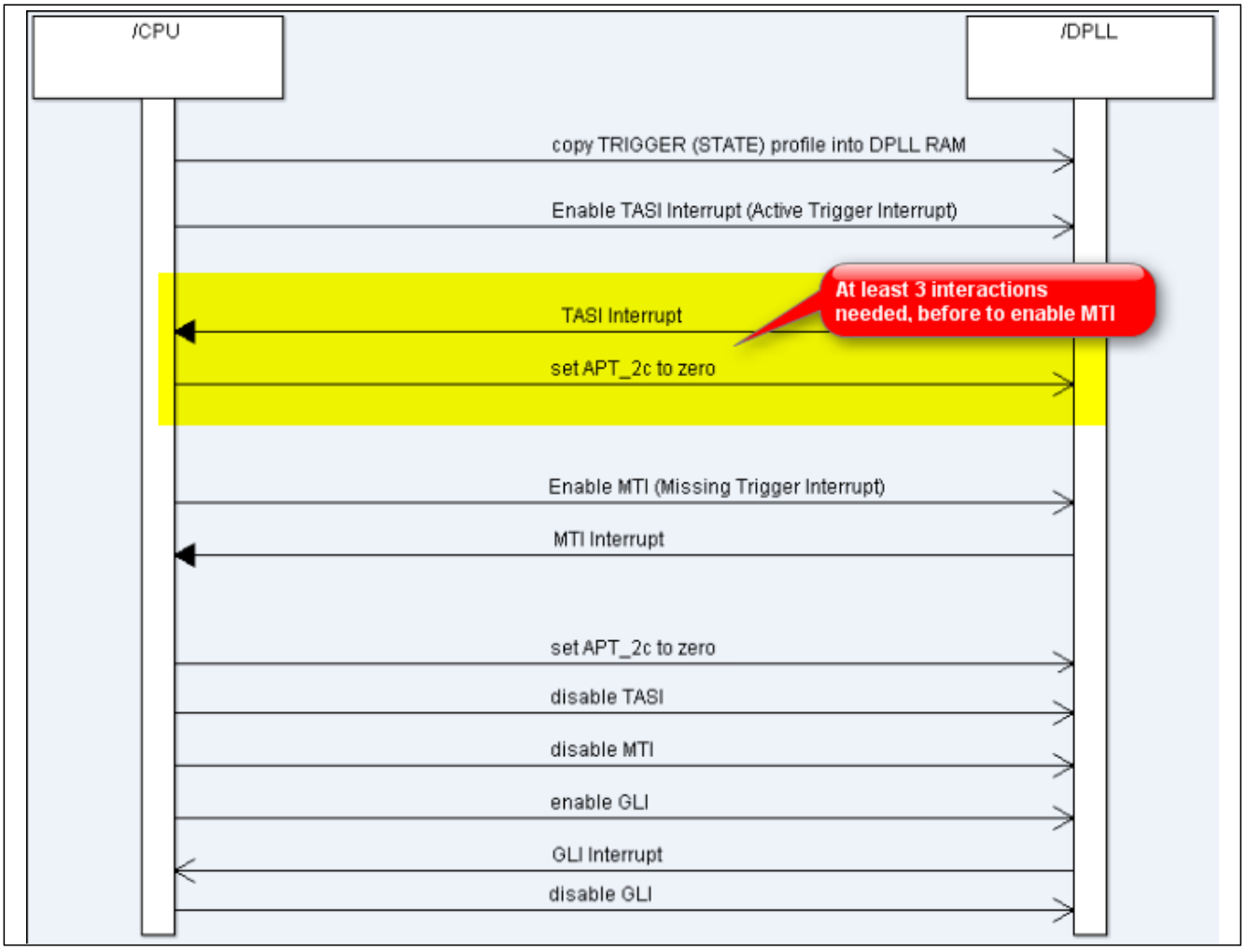

图 22　使用 MTI 中断的 DPLL 同步

3.4.6 DPLL 输入信号真实性检查

在每一个齿上,DPLL 执行一系列需要众多计算的任务。

一般而言，DPLL 可以预测 TRIGGER（和/或 STATE)信号沿之间当前的间隔。

执行的一些基本操作如下：

•齿真实性检查 (PVT)
•预估当前齿的持续时间和下一个齿的持续时间(CDT_X)
•上一个预测误差和加权平均误差
•位置减去时间行动计算(PMTR)
•漏脉冲计算
•中断的产生（有效/无效斜率，缺失触发…）

在同步之后，DPLL 检查是否每个齿都满足一套可配置的溢出定时。

这些定时可以被分成两组：

•有效斜率到有效斜率定时
•有效斜率到无效斜率定时

3.4.6.1 有效沿到有效沿定时

- TS_T_CHECK
- PVT_CHECK
- TLR_CHECK

下图描述了如何使用这些超时:

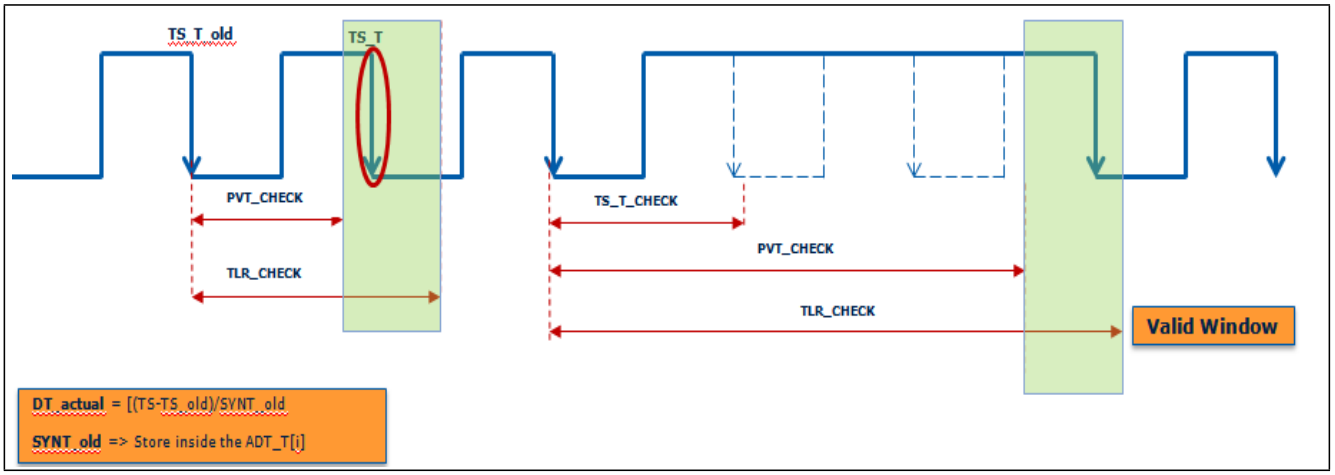

图 23 有效斜率到有效斜率时序

下表总结了时序:

表 1 DPLL 超时时序

时序名称	描述	公式	注释
TS_T_CHECK	检测到下一个有效沿的最长时间	TOV*DT_T[1]$_{actual}$	如果在 TS_T_CHECK 内未出现一个有效的沿,那么 DPLL 引发 MTI(缺失的触发中断)。一般该时序在每个间隙上被扰乱。
PVT_CHECK	出现下一个有效沿的最小时间	PVT_val* DT_T$_{actual}$	如果在 PVT 时间之前出现，该沿被压缩。 CPU 必须调节 APT_2c 指针
TLR_Check	2 个有效沿之间的最大时间	TLR* DT_T$_{actual}$	如果一个有效斜率太晚出现,则引发 TORI 中断(如被使能且 TOR!=0)

任何 TS_T_CHECK 或者是 PVT 违例都不会造成 DPLL 引发的同步丢失。然而，对于 PVT_CHECK，CPU 必须重新调节 APT_2c 指针，以指向正确的齿。TLR 违例会造成同步 2 的丢失。

3.4.6.2 有效沿到无效沿定时

•THMI(最小触发保持时间）检查
•THMA(最大触发保持时间）检查
下图描述了 THMI 和 THMA 时序：

1 **DT_Tactual = (TS –TSold)/SYNT_old，** 表示前一个齿的持续时间
2 默认停用**TLR**检查（TLR=0）

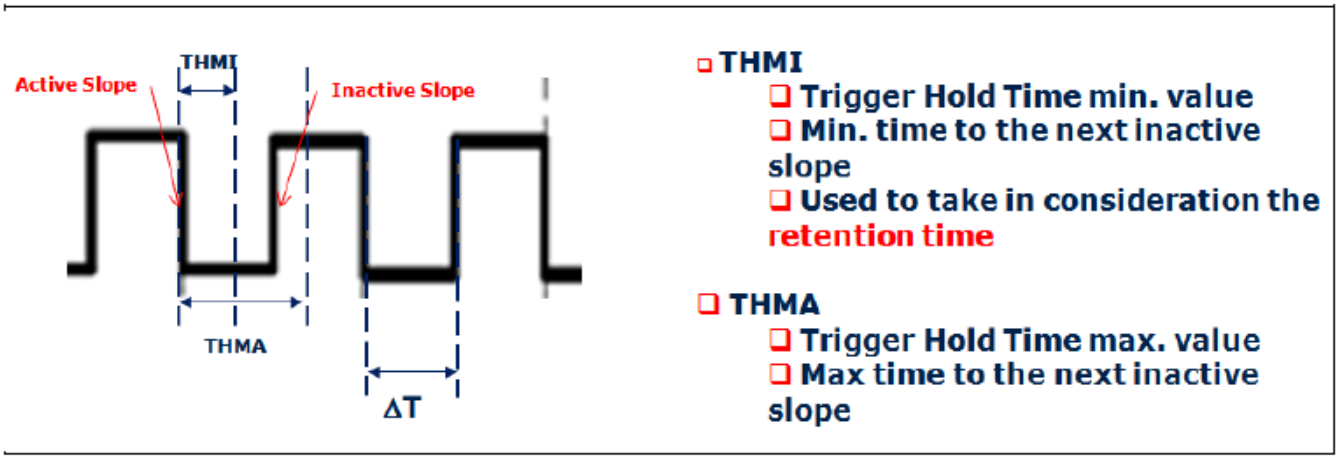

图 24　有效沿到有效沿溢出定时

在某些特定应用里,可将 THMI 和 THMA 超时用于检测转动方向的变化。

3.4.7 DPLL 简化流程图

下图是正常模式下 (RMO=0, SMC = 0)的 DPLL 简化流程图,图中描述了 DPLL 基本操作的概况。
正常模式指将 TRIGGER 信号（曲轴）用于发动机同步。在紧急模式下(RMO=1)使用 STATE(凸轮轴）。

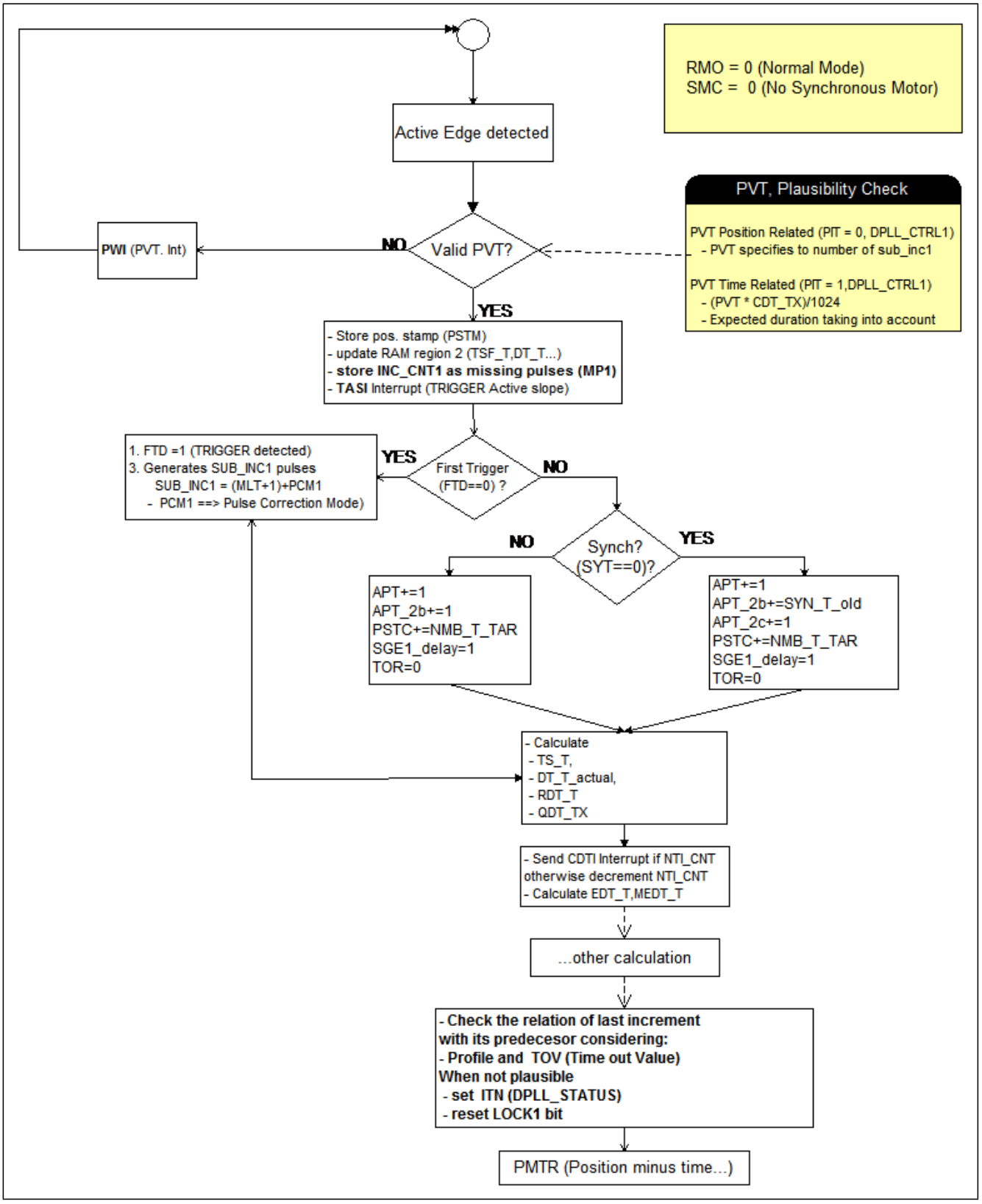
RMO = 0 (Normal Mode)
SMC = 0 (No Synchronous Motor)
Active Edge detected
PVT, Plausibility Check
PVT Position Related (PIT = 0, DPLL_CTRL1)
- PVT specifies to number of sub_inc1
PVT Time Related (PIT = 1,DPLL_CTRL1)
- (PVT * CDT_TX)/1024
- Expected duration taking into account
PWI (PVT. Int)
NO
Valid PVT?
YES
- Store pos. stamp (PSTM)
- update RAM region 2 (TSF_T,DT_T...)
- store INC_CNT1 as missing pulses (MP1)
- TASI Interrupt (TRIGGER Active slope)
1. FTD =1 (TRIGGER detected)
3. Generates SUB_INC1 pulses
SUB_INC1 = (MLT+1)+PCM1
- PCM1 ==> Pulse Correction Mode)
YES
First Trigger (FTD==0) ?
NO
NO
Synch? (SYT==0)?
YES
APT+=1
APT_2b+=1
PSTC+=NMB_T_TAR
SGE1_delay=1
TOR=0
APT+=1
APT_2b+=SYN_T_old
APT_2c+=1
PSTC+=NMB_T_TAR
SGE1_delay=1
TOR=0
- Calculate
- TS_T,
- DT_T_actual,
- RDT_T
- QDT_TX
- Send CDTI Interrupt if NTI_CNT otherwise decrement NTI_CNT
- Calculate EDT_T,MEDT_T
...other calculation
- Check the relation of last increment with its predecesor considering:
- Profile and TOV (Time out Value)
When not plausible
- set ITN (DPLL_STATUS)
- reset LOCK1 bit
PMTR (Position minus time...)

图 25 DPLL 流程图

3.4.8 DPLL 基本寄存器配置

图 26 是基本的 DPLL 寄存器的概况:

DPLL part	Description
DPLL_CTRL_0	Contains input signal characteristics, like *TRIGGER/STATE* event characteristic, input filter characteristic etc.
DPLL_CTRL_1	Contains DPLL configuration for operation mode, micro tick generation, time stamp resolutions, etc.
DPLL_APT_2c	Actual RAM pointer address for RAM region 2c. This pointer has to be written in or after synchronization condition was met. In this application note, the pointer is written in the ISR `void isr_toothdet`(void).
DPLL_THMI	Minimum time to the next inactive *TRIGGER* slope. The time should be given in number of time stamp ticks TBU_TS.
RAM2c	This RAM region has to be initialized with the *TRIGGER* signal input profile for FULL_SCALE. Please note, that there could be a RAM2 initialization after reset. Therefore, the programmer has to wait until this RAM initialization ended.
DPLL_TOV	Timeout value for the actual *TRIGGER* slope. This value has to be provided to the DPLL since otherwise, the DPLL would generate timeout events after the first valid *TRIGGER* event.

图 26 DPLL 寄存器

因为在此例中无 STATE(曲轴)信号输入,因此未显示 RAM 区域 RAM1c3。

以下代码片段以 60-2 齿飞轮为例，给出其 DPLL 配置:

```
void init_dpll(void) {
      unsigned int ram_ini_v = 0;
      unsigned int i = 0;
      gtm_ptr p;
      // initialize RAM;
      // make sure that no RAM initialization takes place in parallel
       ram_ini_v = DPLL_RAM_INI;
       while (ram_ini_v) { // wait until RAM initialization ends
              ram_ini_v = DPLL_RAM_INI;
      }
      p = (uint32 *) ((uint32) &GTM_DPLL_RR2 + 0x800);
      //copying the 60-2 expected profile into the DPLL RAM
      for (i = 0; i < 57; i++) { // file profile for regular tooth
            p[i] = 0x10000; // first HALF SCALE
      }
       p[57] = 0x30000;     //  58th is special one!
       p[58] = 0x12000;     //  Optional:
```

```
                        // gen TINT0 IRQ at the 59th tooth in FULL_SCALE
    for (i = 59; i < 115; i++) { // file profile for regular tooth
         p[i] = 0x10000; // second HALF SCALE
    }
    p[115] = 0x30000;
    DPLL_IRQ_EN = 0x00040000; // enable TINT0 interrupt
    DPLL_TOV = 0x780;
        // configure timeout value for actual TRIGGER slope (TS_T_CHECK)
     DPLL_CTRL_0 = 0x403B0257;
    DPLL_CTRL_1 = 0x80020000;
    DPLL_CTRL_1 = 0x80020012; // now enable the DPLL
                                        }
```

图 27 代码片段:60-2 齿飞轮的 DPLL 初始化示例

3.4.9 DPLL 错误情况和同步丢失

当 DPLL 被同步时,DPLL_STATUS 寄存器里的 FTD, SYT 和 LOCK1 位置位,并持续不断地检查间隙之间的有效 TRIGGER(有效沿)事件的次数。

如果出现以下事件，DPLL 将失去同步：

- 在预期为间隙时（在 PVT 时间之后），检测到额外的有效沿。
  - 在这种情况下，LOCK1 位复位，且 DPLL_STATUS 寄存器里的 ITN 位置位。如被使能，LL1I（或者是用于凸轮轴的 LL2I）被引发以报告“失锁”情况。
- 出现意料之外的间隙（遗漏 TRIGGERS)。
  - 在这种情况下，NUTC 寄存器里的 NUTE 值被设置为 1，LOCK1 位复位，且 DPLL_STATUS 寄存器里的 ITN 位置位。地址指针在出现下一个有效的 TRIGGER 斜率时相应增加。
- TLR 违例
  - 触发锁定范围违例

3.4.10 DPLL 中断

DPLL 提供一系列中断以报告与曲轴/凸轮轴管理相关的所有可能事件。

可以通过 DPLL_IRQ_EN 在任何时间使能或停用所有的中断。

表 2 DPLL 中断

中断信号	描述	注解
DPLL_TORI_IRQ DPLL_SORI_IRQ	TRIGGER(曲轴）超出范围 STATE(凸轮轴）超出范围	如果 TOR 和/或 SOR 被置位(!=0)，则执行一个最大超时检查以便检测 TOR*DT_T_{actual}（或 SOR*DT_S_{actual})之后出现的有效沿。
DPLL_CDSI_IRQ DPLL_CDTI_IRQ	为上一次增加计算 TRIGGER 持续时间 为上一次增加计算 STATE 持续时间	对于 CDTI_IRQ, 有可能编写一个在 NTI_CNT 有效沿之后引发的中断程序。 在 STATE 公式计算结尾引发 CDSI_IRQ。
DPLL_TE0_IRQ DPLL_TE1_IRQ	TRIGGER 事件中断请求	可在某个特定的齿上引发的一系列中断。 仅在同步(SYT=1)之后有效。

中断信号	描述	注解
DPLL_TE2_IRQ DPLL_TE3_IRQ DPLL_TE4_IRQ		TEi_IRQ 取决于 ADT_T[i] 里的 TINT 值(预期特征).
DPLL_GL1_IRQ DPLL_GL2_IRQ	用于 SUB_INC1 请求的"get lock(取锁)"中断(曲轴) 用于 SUB_INC2 请求的"get lock（取锁）"中断(凸轮轴)	在 CPU 调整 APT_2c 指针之后(同步),如果下一个间隙如期出现,则 DPLL 引发 GL1 (或 GL2)中断。
DPLL_LL1_IRQ DPLL_LL2_IRQ	用于 SUB_INC1 请求的"loss of lock(失锁)"中断 用于 SUB_INC2 请求的"loss of lock(失锁)"中断	当在间隙中检查的有效沿个数与预期不相符时,引发中断
DPLL_PW_IRQ	TRIGGER（曲轴）真实性窗口（PW)违例中断请求	有效沿出现在规定的超时之前： (PVT_val * DT_Tactual)/1024 if PIT=0 PVT_val (PIT=1=>number of sub_inc1) 对于 PW,不考虑齿,且 CPU 必须调整 APT_2c 指针。
DPLL_TAS_IRQ	当 NTI_CNT 为 0 时，TRIGGER 有效沿中断请求	当 CDT1 未被编程时(NTI_CNT=0)，在每个触发器（曲轴）有效斜率上引发中断
DPLL_SAS_IRQ	STATE 有效沿（SAS)中断请求	在每个 STATE（凸轮轴）有效沿上引发该中断。
DPLL_MT_IRQ DPLL_MS_IRQ	遗漏 TRIGGER(曲轴)中断请求 遗漏 STATE(凸轮轴)中断请求	在 TS_T_CHECK 时间内(或 TS_S_CHECK 时间)未检测到有效沿时被引发。 TS_T_CHECK=TS_T + DT_T_actual*(TOV); TS_S_CHECK=TS_S+DT_S_actual*(TOV_S); 仅有 RMO=1 (紧急模式)
DPLL_TIS_IRQ DPLL_SIS_IRQ	TRIGGER 无效沿（TIS)中断请求 STATE 无效沿中断请求	这些中断在所有的 TRIGGER/STATE 无效沿上都会被引发。
DPLL_TAX_IRQ	TRIGGER 最大保持时间违反中断请求	如果无效沿在来自有效沿的 THMA 时间之后出现。(THMA>0)
DPLL_TIN_IRQ	TRIGGER 最小保持时间违反中断请求	如果无效沿在来自有效沿的 THMI 时间之前出现。(THMI>0)
DPLL_E_Interrupt	DPLL 误差中断	见 DPLL_STATUS 寄存器位 31

3.5 ATOM:连接 ARU 的定时器输出模块

连接 ARU 的定时器输出模块(ATOM)无需 CPU 介入便可生成复杂的输出信号。

每个 ATOM 子模块包含了 8 个输出通道，这 8 个输出通道在多个可配置操作模式下可相互独立工作。

ATOM 子模块框图如下所示：

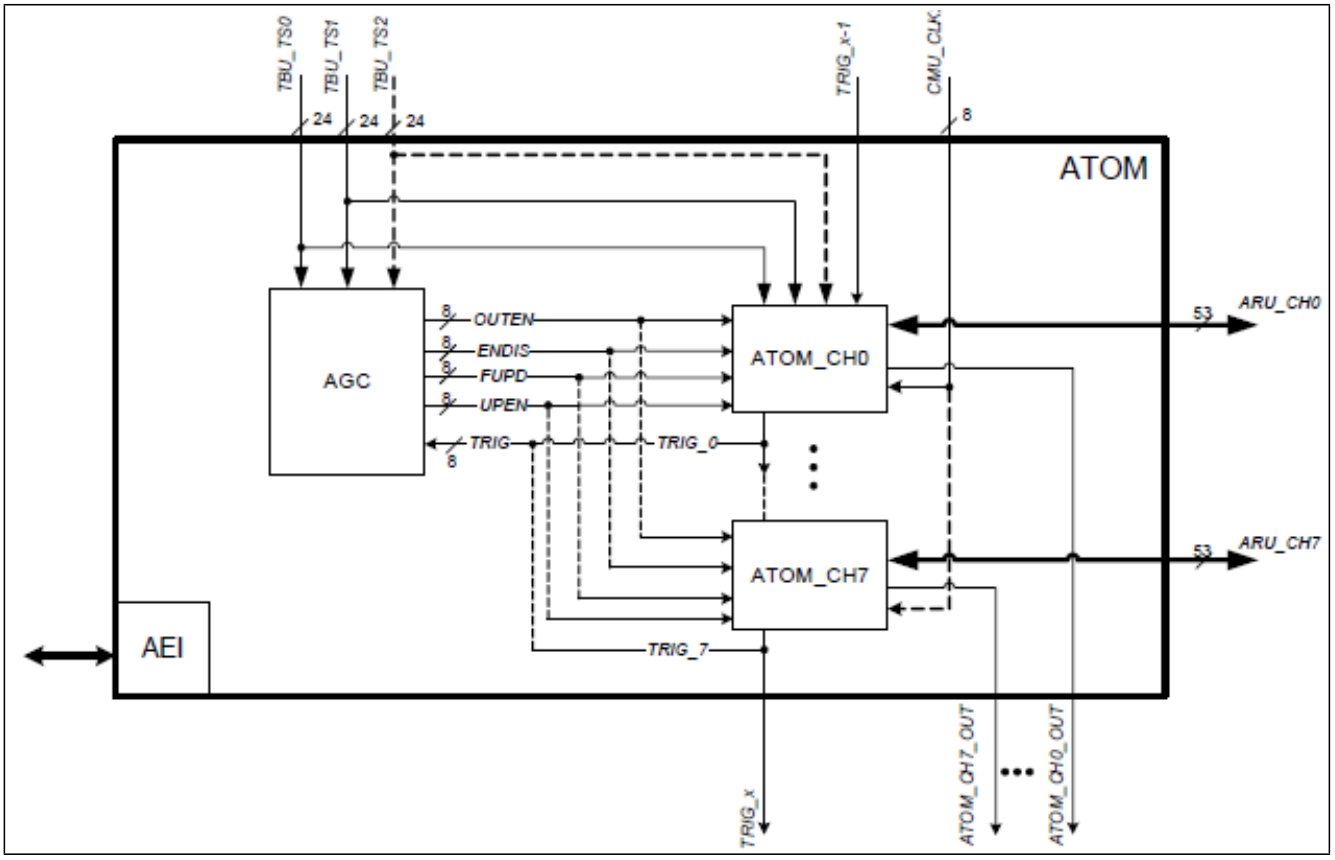

图 28 ATOM 框图

每个 ATOM 通道可在 4 种不同的信号输出模式下操作：

- ATOM 信号立即输出模式 (SOMI)
- ATOM 信号比较输出模式 (SOMC)
- ATOM 信号 PWM 输出模式 (SOMP)
- ATOM 信号连续输出模式 (SOMS)

该文件专注于 SOMC 模式。

3.5.1 ATOM SOMC

每个 ATOM 通道有两个比较单元 (CCU0 和 CCU1)，且在 SOMC 信号输出模式下可以使用这两个单元（也可以组合使用），以在发生匹配事件时引发中断和/或触发另外一个比较单元。

一般来说，CCU0 比较单元用于执行基于时间的比较，CCU1 用于基于位置/角度的比较。

图 29 所示为典型的 ATOM 示例。

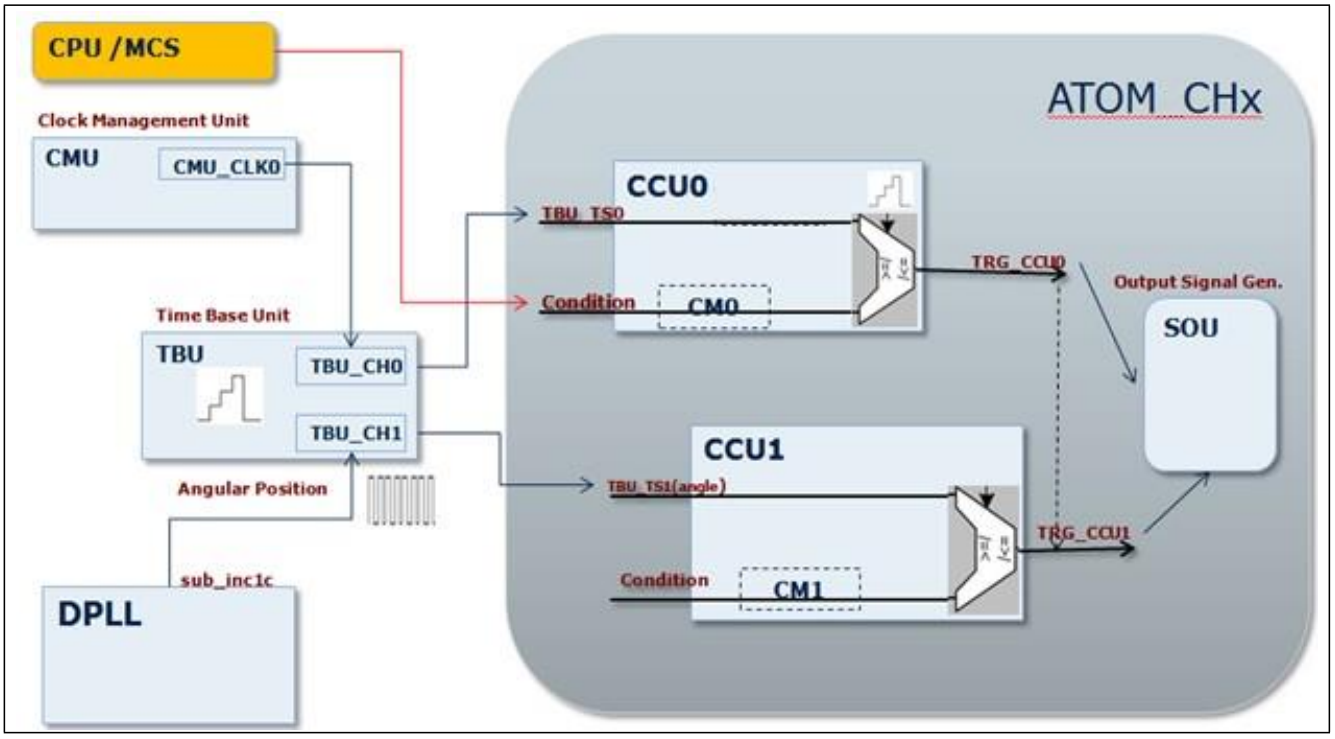

图 29 ATOM SOMC 示例

在 SOMC 模式下,按照如下方式控制两个比较单元 CCU0 和 CCU1 的行为状态:

- 停用 ARU
  - ACB 位(ATOM[i]_CH[x]_CTRL 寄存器)的第 4 位到第 2 位
- 使能 ARU
  - 通过 ARU 控制位第 52 位到第 48 位更新 ACBI 位字段(ATOM[i]_CH[x]_STAT),及 ACB 位字段。

根据 SL 位里预定义的信号电平，并结合在 ATOM[i]_CH[x]_CTRL 或 ATOM[i]_CH[x]_STAT 寄存器里的 ARU 或 CPU 定义两个控制位，CCUx 触发器信号 TRIG_CCU0 和 TRIG_CCU1 产生沿。

更多有关 ATOM 和 SOMC 模式，请参考 AURIX™数据手册。

3.6 相对角 vs.绝对角

为了避免任何微齿的丢失，并使用 PMTR(位置减去时间请求），DPLL 应总是以相对角工作。
角度计数器 TBU_CH1 是一个 24 位的计数器。但有时候，建立在发动机位置驱动（如 MPG 驱动器），需要以绝对角工作。
例如，为了从 10 度角到 360.5 度角时产生角-角脉冲，可以使用如下参数：

- <AngleStartAbs> = 100 (10.0 degrees)
- <AngleStopAbs> = 3605 (360.5 degrees)

发动机位置驱动应该可以提供一系列的功能以从绝对角开始计算相对角。
一个可行的方案是在每转之后，储存用作偏移的“相对角”，与绝对角相加，从而给出相对角。

- <AngleStartRel> = IfxVrs_getAngleOffset () + <AngleStartAbs>

因为有 24 位的角度计数器,因此偏移是必须的;在 720° 角时,最大的角度值不是生成的微齿总数的倍数。例如:

- 对于每个齿（6 度），256 个微齿
- 120 个齿=> 256*120 = 30720 个微齿/720 度
- (0x00FFFFFF+1) / 30720 => 546,13

这意味着由于在每次角度计数器重启之后（转出），相对角度 0x00000000 对应一个不同的绝对角度，因此无法简单地使用一个“模块”操作从相对角度计算出绝对角度（如 relativeAngle>%30720)。
在每次旋转之后，为了保存当前的相对角度，可以使用 5 个 DPLL_TE[i]_IRQ 中断其中之一。考虑到 60-2 齿曲轴，需在 ADT_T[i] 曲轴档案第 1 个元素（ADT_T[0]）上使能该中断。下列伪代码是一个实现的示例：

```
static IfxVrsPhs_setCrankProfileHook IfxVrs_initCrankProfile(void) {
  uint8 i;
     for (i = 0; i < 57; i += 1) {
       pmem_ADT_Tx[i] = 0x10000;
  }
  pmem_ADT_Tx[0] = 0x18000; // TE3_IRQ=> update rel. angle offset
  pmem_ADT_Tx[55] = 0x12000;
  pmem_ADT_Tx[56] = 0x14000;
  pmem_ADT_Tx[57] = 0x36000;
  for (i = 58; i < 115; i += 1) {
      pmem_ADT_Tx[i] = 0x10000;
  }
  pmem_ADT_Tx[113] = 0x12000;
  pmem_ADT_Tx[114] = 0x14000;
  pmem_ADT_Tx[115] = 0x36000;
}

// DPLL_TE0 Interrupt Service Routine
static sint32 IfxVrs_te3Isr(void) {
  ......
  ......
  vrsAttributes.globalTeethOffset = (GTM_TBU_CH1_BASE.U + 1);
  ...
  ...
  return (0);
}
```

图 30 代码片段:相对角偏移示例

有关该主题所采用的解决方案不可一概而论,在不同的应用中应采用不同的解决方案,并且上述建议不是唯一可行的解决方案。
另外一种可行方案，就是使用在 ADT_T 数组档案里的 PD(物理偏移）字段，使每转所产生的微时钟总数是 2 的幂。

在这种方式下，在每次 TBU_CH1 计数器溢出之后的相对角 0x00000000 永远是参照绝对角 0° 。另一方面，对于每个曲轴齿，DPLL 所产生的微时钟数并不是都相同。

4 代码实例：发动机位置驱动程序

4.1 简介

开发了一个基础的发动机位置驱动程序作为“概念证明”。驱动程序的目的是表示如何使用 GTM 解决与曲轴/凸轮轴管理相关的最重要的话题。

注意：　并未打算将该驱动程序用于实际产品，而是仅仅将其用作入门级开发的起点。

以下章节概述了驱动程序的 API(应用程序界面）及其用途。

如同驱动程序代码，还可以获得 doxygen 文档及 windows 帮助文件。

4.2 驱动程序概览

驱动程序使用输入曲轴飞轮信号（在 TIM0_CH0 上），执行以下功能：

- 产生飞轮角度信号以控制喷油和点火功能。
- 测量齿周期(VRS_TP, 其中 VRS 代表可变磁阻传感器)
- 测量 TDC（上止点）周期(VRS_TDCPeriod)
- 从 0 到 VRS_NTEETH_CYCLE -1 开始数齿
- 产生“同步”标志(flywheelStat)
- 测量 TDC 周期
- 产生“发动机停止”异常(VRS_EX_EngineStopped)
- 产生与齿和角度比例信号相关的事件
- 生成“错误类型”诊断信息(VRS_Error)
- 生成 CLC 缓冲以获取齿周期
- 紧急模式管理（曲轴用于生成角度信息）

驱动程序采用一个状态机以管理飞轮的不同状态。图 31 是状态机简图：

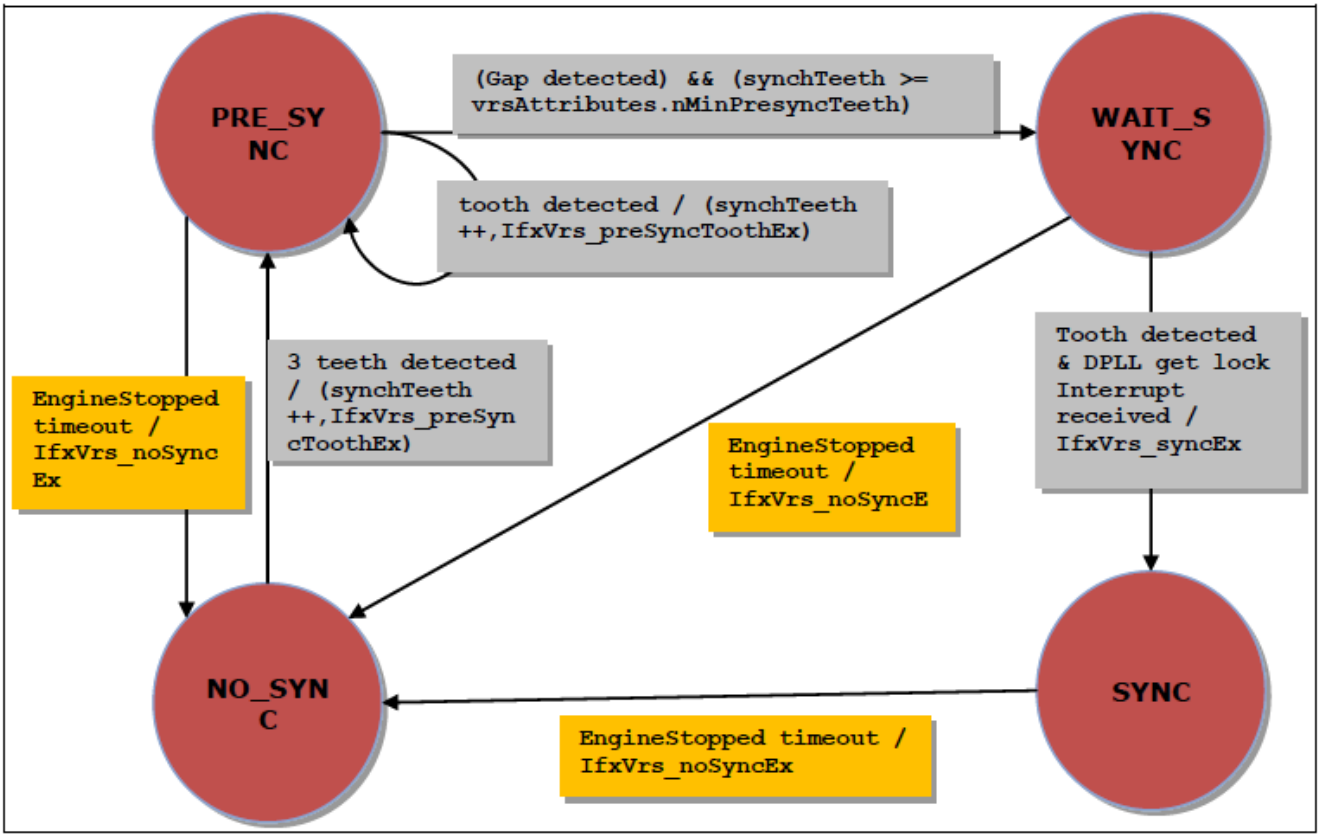

图 31　发动机位置驱动程序：飞轮状态机

4.3　同步过程

VRS 驱动程序同步过程如图 32 所示。

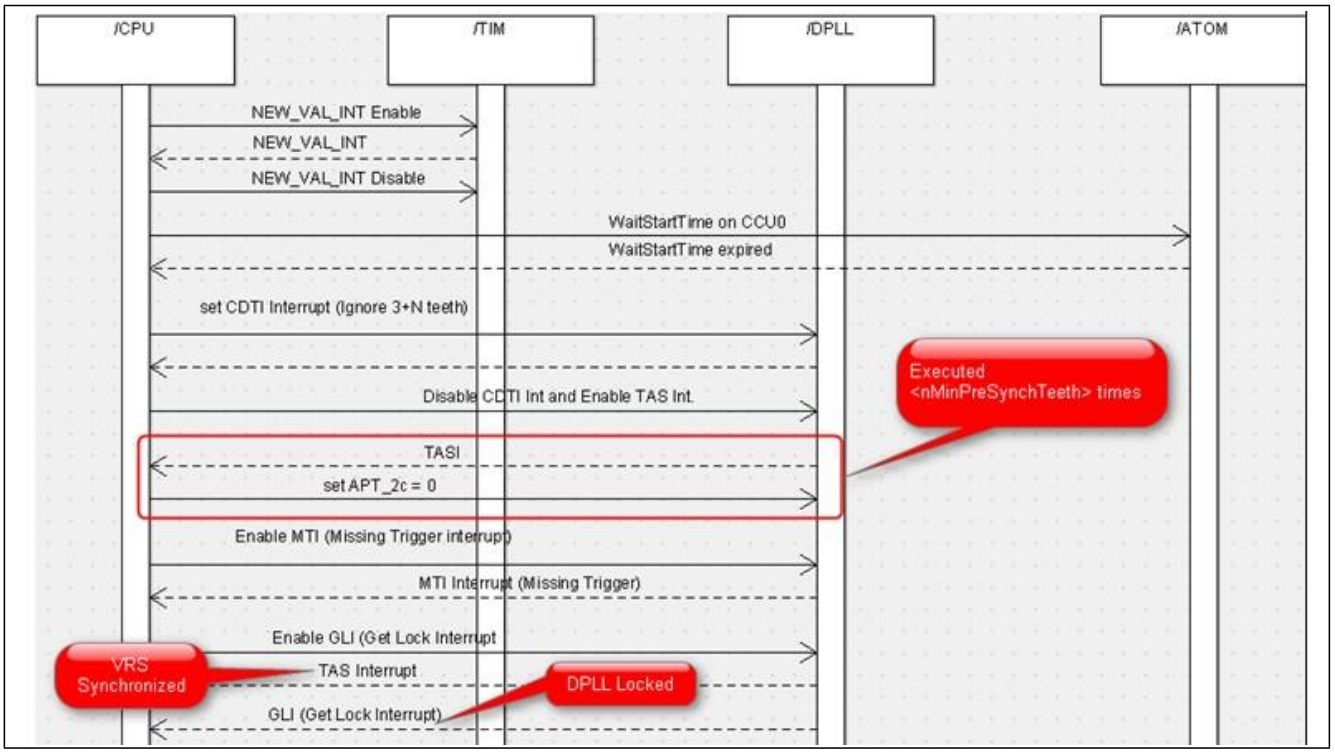

图 32　VRS 驱动程序同步

触发器有效沿中断(TASI),指在运行期间不需要在每个齿上都生成一个中断。

TIM 模块通过使用其超时检测单元（TDU)实现“发动机已停止”超时。

ATOM 模块用于在进程开始(WaitStartTime)的时候“计数”熄灭时间。

4.4 VRS 基础检查和性能

在每个曲轴齿上，驱动程序使用 DPLL 硬件执行一系列的检查以检测任何与系统相关的状况（如提前齿，缺齿检测,..)。在 DPLL 输入信号真实性检查章节中描述了该检查。

在具体实现中，我们决定在每转一圈时，多次改变 TS_T_CHECK 和 the PVT_CHECK 时序，以便在某些特殊的事件中有不同的状态（如间隙之后的第一个齿）。

为了实现该策略，使用了 ADT_T 数据可编程中断特性（TIN)。

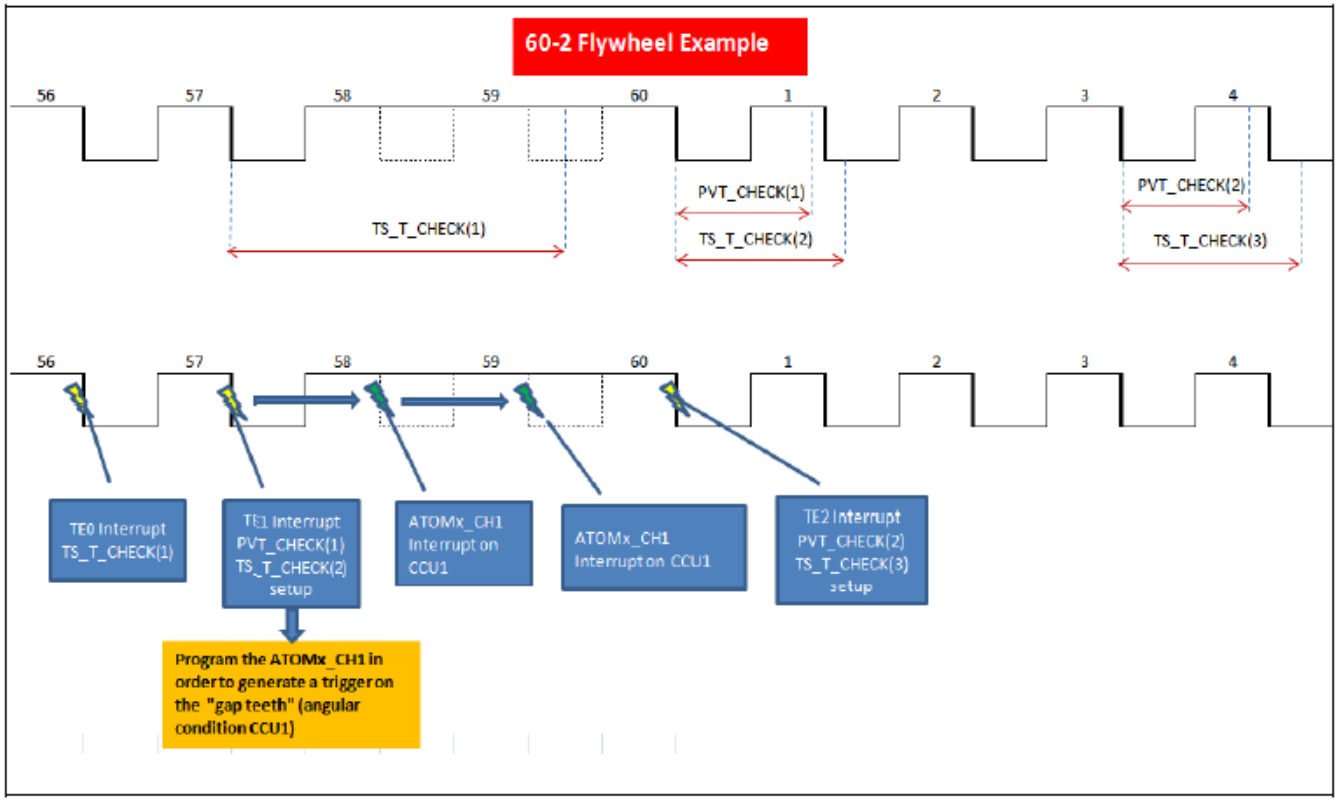

图 33 VRD 驱动程序中断流程

每次转动所需的最小中断数为 3 (TE0,TE1 和 TE2)。如果在间隙内的一个齿上编程任意齿事件（IfxVrs_setEventOnTooth)，使用一个 ATOM“模拟”间隙齿，并且随着间隙齿数量的增加，中断次数也相应增加。

4.5 公共函数 - API

表 3 VRS 驱动程序 API

函数名称	函数签名	描述
IfxVrs_initConfig	void IfxVrs_initConfig	

执行示例：发动机位置驱动程序

函数名称	函数签名	描述
	(IfxVrsPhs_Config *config)	该函数用于初始化 config 变量为默认值
IfxVrs_init	IfxVrs_Mstatus **IfxVrs_init**(IfxVrs Phs_Config *config)	VRS 驱动程序配置
IfxVrs_acqConfig	IfxVrs_Mstatus **IfxVrs_acqConfig**(IfxVrs _DiagnosticMode diagnosticMode)	用于在正常操作和紧急操作之间变换驱动器操作模式
IfxVrs_tdcConfig	IfxVrs_Mstatus **IfxVrs_tdcConfig** (IfxVrs_Nteeth firstTDCTooth, IfxVrs_Nteeth tdcTeeth)	配置上止点中心（Top Dead Centre）周期测量
IfxVrs_setActiveEdge	uint32 **IfxVrs_setActiveEdge** (*short activeEdge*)	设置 TRIGGER 有效沿
IfxVrs_start	void **IfxVrs_start** (void)	启动驱动程序
IfxVrs_getTdcPeriod	IfxVrs_Period **IfxVrs_getTdcPeriod** (*IfxVrs_Mstatus*status*)	检索测量的最后 TDC 周期
IfxVrs_getFlywheelStat	IfxVrs_FlywheelStat **IfxVrs_getFlywheelStat** (*void*)	返回飞轮状态
IfxVrs_getAngleOffset	uint32 IfxVrs_getAngleOffset (*void*)	返回用于计算相对角（相对绝对角）的当前角度偏移
IfxVrs_setAngleOffset	uint32 **IfxVrs_setAngleOffset** (*uint32 angleValue*)	设置用于计算绝对角的当前角度偏移
IfxVrs_getLastTs	uint32 IfxVrs_getLastTs (*void*)	检索最后一个齿的时间戳
IfxVrs_getAbsAngle	IfxVrs_FlywheelAngle **IfxVrs_getAngle** (*IfxVrs_Mstatus*status*)	返回当前发动机绝对角度(内分辨率)
IfxVrs_setEventOnTooth	IfxVrs_Nteeth **IfxVrs_setEventOnTooth** (IfxVrs_Nteeth toothNo, IfxVrs_Mstatus *status)	在某个特定的齿上设置一个一次性事件。
IfxVrs_setEventOnAngle	IfxVrs_FlywheelAngle **IfxVrs_setEventOnAngle**(IfxVrs_FlywheelAngle	在某个特定的角度上（绝对角度 - 0000-7200)设置一个一次性事件

函数名称	函数签名	描述
	absAngle, IfxVrs_Mstatus *status)	
IfxVrs_toothPeriod	IfxVrs_Period **IfxVrs_toothPeriod** (*IfxVrs_Mstatus *status*)	返回最后一个齿的周期
IfxVrs_setWaitStart	IfxVrs_Mstatus **IfxVrs_setWaitStart** (*IfxVrs_Period waitTime*)	在开始同步之前，设置一个最初的超时
IfxVrs_setNIgnoreTeeth	IfxVrs_Nteeth **IfxVrs_setNIgnoreTeeth**(IfxVrs_Nteeth teethNumber, IfxVrs_Mstatus *status)	在开始同步之前，设置在 NO_SYNC 等待的齿的数量
IfxVrs_setNMinPresyncTeeth	IfxVrs_Nteeth **IfxVrs_setNMinPresyncTeeth**(IfxVrs_Nteeth teethNumbers, IfxVrs_Mstatus *status)	设置保持在 VRS 状态机的 PRESYNC 状态的齿的最小数量
IfxVrs_getNTooth	IfxVrs_Nteeth **IfxVrs_getNTooth** (*IfxVrs_Mstatus *status*)	获取当前飞轮齿的数量。
IfxVrs_getClcBuffer	IfxVrs_Mstatus **IfxVrs_getClcBuffer** (*uint8 size, uint32 *buffer*)	检索包含齿时间戳的缓冲区
IfxVrs_clcEnable	IfxVrs_SwitchMode **IfxVrs_clcEnable** (IfxVrs_SwitchMode sw, IfxVrs_Mstatus *status)	使能/停用齿周期数组缓冲区

下列是驱动器配置和用途的代码示例。

所有最重要的驱动属性都包含在 vrsAttributes 全局变量内部。图 34 为驱动器示例。

```
IfxVrsPhs_Config vrsConfig;
IfxVrs_initConfig(&vrsConfig);

vrsConfig.camActiveEdge = IfxVrs_ActiveEdge_both;
vrsConfig.crankActiveEdge = IfxVrs_ActiveEdge_falling;
vrsConfig.crankTeethNr = 59;  // (59+1) 60 teeth per Cycle
vrsConfig.crankCycles = 2;    // 2 Cycles => 120 teeth
vrsConfig.camTeethNr = 0;     //1 Tooth Cam
vrsConfig.camCycles = 1;
vrsConfig.microTicksTooth = IFX_VRS_TOOTH_STEPS; //1024 microtick for each tooth
vrsConfig.feFilterValCh0 = 0x00000050;
vrsConfig.feFilterValCh1 = 0x00000050;
vrsConfig.reFilterValCh0 = 0x00000040;
vrsConfig.reFilterValCh1 = 0x00000040;
vrsConfig.crankSetupFunc = (IfxVrsPhs_setCrankProfileHook) setupCrankDpllProfile;
vrsConfig.camSetupFunc = (IfxVrsPhs_setCamProfileHook) setupCamDpllProfile;
vrsConfig.angleEventHandler = (IfxVrsPhs_callbackFuncPtr) vrsAngleEventHanlder;
vrsConfig.toothEventHandler = (IfxVrsPhs_callbackFuncPtr) vrsToothEventHanlder;

//Driver Init
if (useDefaultSetting == FALSE)
  IfxVrs_init(&vrsConfig);
else
  IfxVrs_init(NULL); //Use default Setting

IfxVrs_acqConfig(IfxVrs_DiagnosticMode_noFault);
IfxVrs_setActiveEdge(IfxVrs_ActiveEdge_falling);
IfxVrs_setWaitStart(15000);             //Wait 15000xtCLK after the 1st tooth detected
IfxVrs_setNIgnoreTeeth(5, &status);     //Wait 3 + [5] teeth in NO_SYNC
IfxVrs_setNMinPresyncTeeth(4, &status); //Wait 4 teeth is PRE-SYNC
IfxVrs_tdcConfig(2, 20);                //Config the Top Dead Centre period meas.
IfxVrs_start();                         //Start the driver

//Wait for the DPLL/Flywheel sync.
while (IfxVrs_getFlywheelStat() != IfxVrs_FlywheelStat_sync) ;

IfxVrs_gapExEnable(IfxVrs_SwitchMode_off);        //Enable callback function "gapEx"
IfxVrs_clcEnable(IfxVrs_SwitchMode_off, &status); //Enable clc
IfxVrs_setEventOnAngle(3600, &tmp_status);        //Enable an "one-time" Angle Event  Ang=360.0
IfxVrs_setEventOnTooth(110, &tmp_status);         //Enable an "one-time" Tooth Event  Tooth No 110 (1..120)

while (1 == 1) {
  prev_teeth = current_teeth;
  current_teeth = IfxVrs_getNTooth(&tmp_status);          // current tooth number
  current_teeth_period = IfxVrs_toothPeriod(&tmp_status); // current tooth duration
  current_angle = IfxVrs_getAbsAngle(&tmp_status);        // current absolut angle
  topDeadCPeriod = IfxVrs_getTdcPeriod(&tmp_status);
```

图 34　VRS 驱动程序 API:用途示例

下列属性包含在 vrsAttributes:

表 4 IfxVrs 驱动程序:公共属性

属性名称	描述
IfxVrs_FlywheelStat currentState	当前飞轮状态
IfxVrs_FlywheelMode currentMode	当前飞轮工作模式
IfxVrs_DiagnosticMode diagnosticMode	飞轮诊断模式
IfxVrs_SwitchMode enableClc	使能齿持续时间环形缓冲区的标志
IfxVrs_SwitchMode ignoreStatus	表明是否必须需要额外的齿在 NO_SYNC 状态下等候
IfxVrs_ActiveEdge activeEdge	曲轴有效沿
IfxVrs_Nteeth toothEventProgrammed	对当前/最后一次事件的配置的齿
IfxVrs_FlywheelAngle angleEventProgrammed	对当前/最后一次事件的配置的角度
IfxVrs_FlywheelAngle globalTeethOffset	用于计算相对角度的角度偏移
IfxVrs_Period tdcPeriod	最后上止点(TDC)周期
IfxVrs_Period waitStart	检测到第一个齿之后,等待的 CLK 循环数
IfxVrs_Nteeth nMinPresyncTeeth	显示有多少个齿在 PRE_SYNC 状态等候
IfxVrs_Nteeth firstTDCTooth	用于上止点(TDC)周期计算的第一个齿
IfxVrs_Nteeth tdcTeeth	两个 TDC [0..HALF_SCALE-1]之间的齿数
IfxVrs_Nteeth nIgnoreTeeh	显示有多少个齿在 NO_SYNC 状态等候
IfxVrs_Bool isInit	驱动程序初始化标志
GTM_DPLL_STATUS_type* dpllStatus	指向 DPLL 状态寄存器的指针
IfxVrs_EnableCallback callbacksEn	用于使能/禁止回调函数
IfxVrs_Nteeth synchTeeth	在 PRS_SYNC 状态里的齿数计数器
IfxVrs_Period lastToothPeriod	最后一个齿的持续时间
IfxVrs_Bool isDpllLocked	DPLL 锁定状态

图 35 显示了驱动程序在运行时的表现状态。

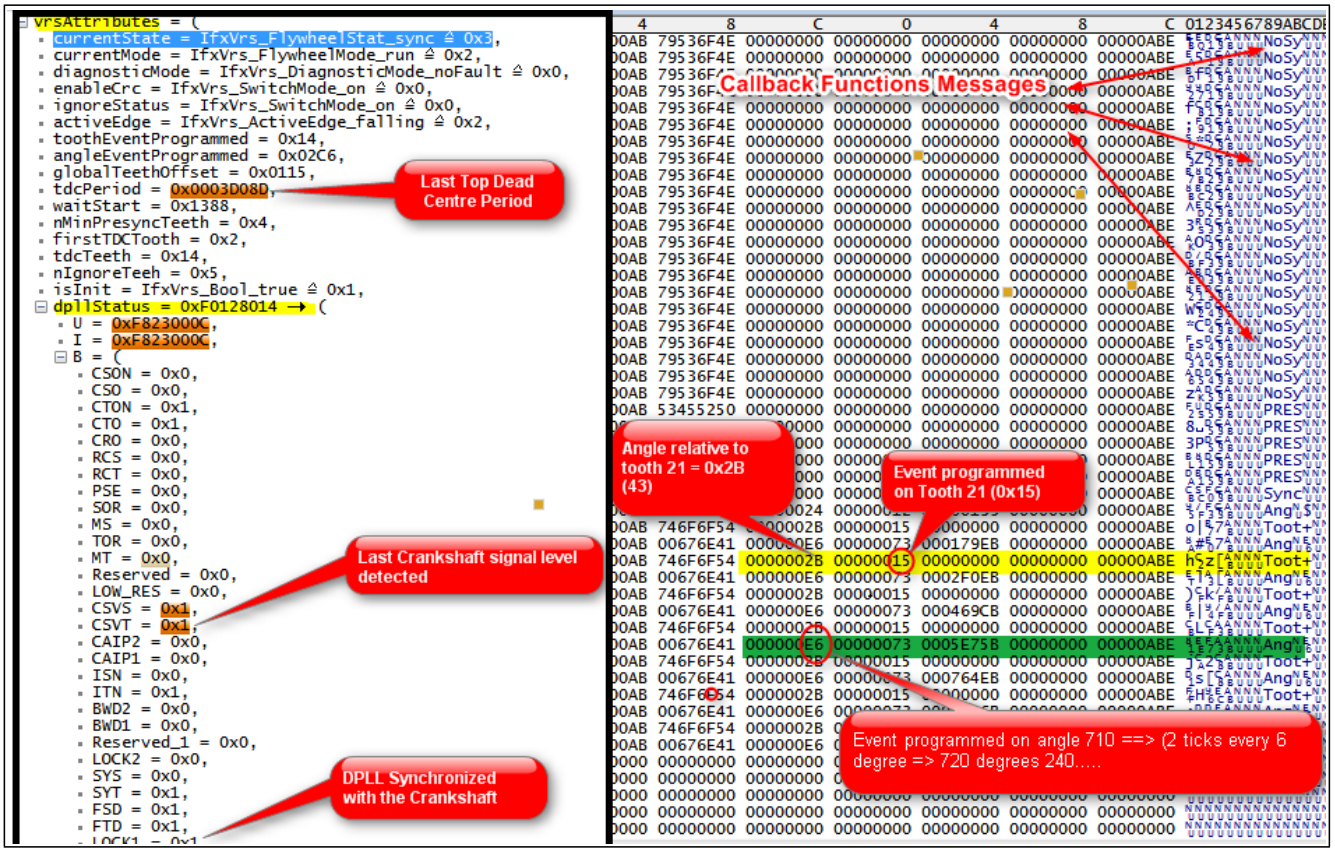

图 35　运行时的发动机位置驱动程序

dpllStatus 属性提供了许多有用的 DPLL 信息。例如，可检查：

•DPLL 是否与曲轴或凸轮轴同步
•检测到了多少个缺失的齿（MT 字段）
•用于曲轴和凸轮轴的输入信号电平(CSVT 和 CSVS)

注意：　重点是要记住在尝试使用驱动程序之前，必须配置曲轴和/或凸轮轴的所有参数。大多数参数都包含在 IfxPhsVrs_common.h 文件。

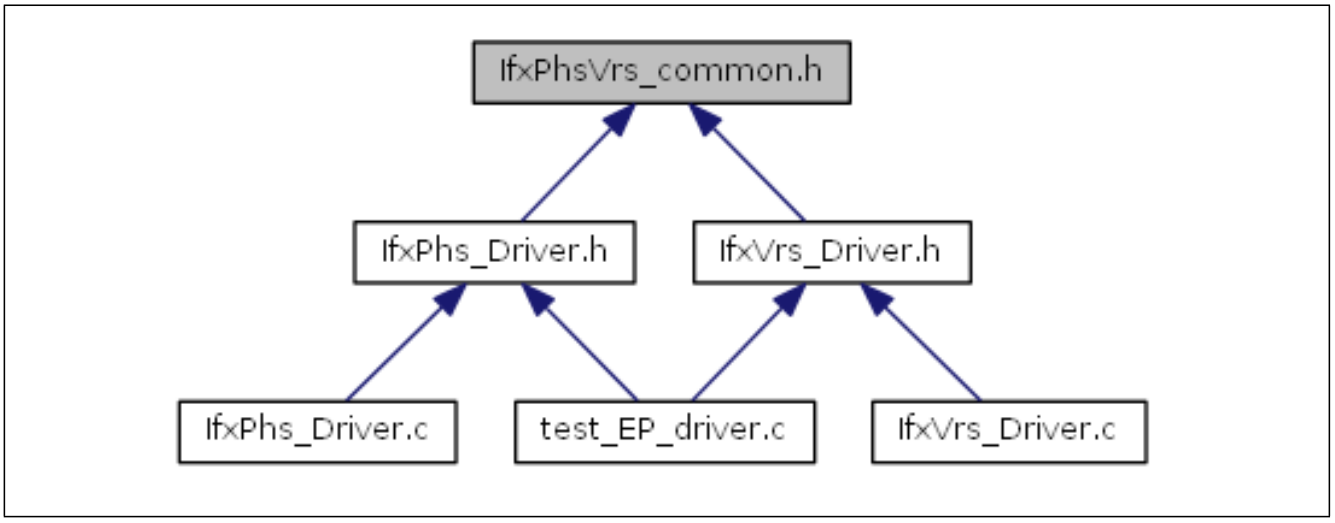

图 36 VRS 驱动程序文件相关性

4.6 私有函数

表 5 IfxVrs 驱动程序:私有函数

函数签名	描述
void **IfxVrs_timConfig** (uint32 reFilterValCh0, uint32 feFilterValCh0, uint32 reFilterValCh1, uint32 feFilterValCh1)	TIM0_CH0 和 CH1 配置
void **IfxVrs_enableTbu** (void)	使能时基单元计数器(CH0,CH1)
void **IfxVrs_configDpll** (uint8 triggerN, uint8 stateN, uint8 microtickMlt, uint8 triggerActiveEdge, uint8 stateActiveEdge)	DPLL 配置
void **IfxVrs_initCrankProfile** (*uint8 nbOfTeeth, uint8 nbOfMissingTeeth*)	曲轴输入数据特性(ADT_T:为所有的增加改写数值(TRIGGER))
void **IfxVrs_initCamProfile** (*uint8 nbOfTeeth, uint8 nbOfMissingTeeth*)	凸轮轴输入数据特性(ADT_S:为所有的增加改写数值(STATE))
void **IfxIrq_initIrqsVector** (void)	中断矢量初始化
sint32 **IfxVrs_tim00Isr** (void)	TIM0_CH0 ISR
sint32 **IfxVrs_atomMngtIsr** (void)	ATOM4_CH0 ISR, ATOM4_CH1 ISR
sint32 **IfxVrs_mtiIsr** (void)	DPLL 丢齿触发器 ISR
sint32 **IfxVrs_gliIsr** (void)	DPLL 取锁触发器 IST
sint32 **IfxVrs_pwiIsr** (void)	DPLL 真实性检查违例 ISR
sint32 **IfxVrs_lliIsr** (void)	DPLL 失锁 ISR
sint32 **IfxVrs_tasIsr** (void)	DPLL 有效斜率触发器 ISR
sint32 **IfxVrs_tisIsr** (void)	DPLL 无效斜率触发器 ISR
sint32 **IfxVrs_cdtiIsr** (void)	CDT 中断服务程序
sint32 **IfxVrs_te0Isr** (void)	通用中断 DPLL TEI0 ISR
sint32 **IfxVrs_te1Isr** (void)	通用中断 DPLL TEI1 ISR
sint32 **IfxVrs_te2Isr** (void)	通用中断 DPLL TEI2 ISR
sint32 **IfxVrs_te3Isr** (void)	通用中断 DPLL TEI3 ISR
sint32 **IfxVrs_te4Isr** (void)	通用中断 DPLL TEI4 ISR

4.7 回调函数

VRS 驱动程序通过使用一套回调函数与上层软件层相互作用。可以通过修改 callbacksEn 属性使能/停用该回调函数。

表 6 IfxVrs 驱动程序:回调函数

函数名称	描述
IfxVrs_noSyncEx	当 flywheelStat 属性被设置为 NO_SYNC 值时所引发的事件。
IfxVrs_engineStoppedEx	当暂时超时消逝时，驱动程序所引发的事件
IfxVrs_PreSyncToothEx	当 flywheelStat = PRE_SYNC 时，在每个齿上所引发的事件。
IfxVrs_syncEx	当 flywheelStat 变量被设置为 SYNC 时所引发的事件。
IfxVrs_toothEx	在某个特定的，用户定义的齿上所引发的事件。
IfxVrs_angleEx	在某个特定的，用户定义的角上所引发的事件。
IfxVrs_warningErrorEx	当检测到一个无效齿时（真实性检查误差）所引发的事件
IfxVrs_noSyncStateEx	当 VRS_FlywheelStat= NO_SYNC 时， BIOS_VRS_IDN_EV_NoSyncState 事件必须是在每个齿事件上由驱动程序所引发。
IfxVrs_gapEx	当检测到间歇且在 SYNC 状态运行时所引发的事件。
IfxVrs_dpllLockEx	在检测到第一个间隙时所引发的事件。之后，DPLL 被同步。

下图显示了 GTM 中断和回调函数之间的关系。

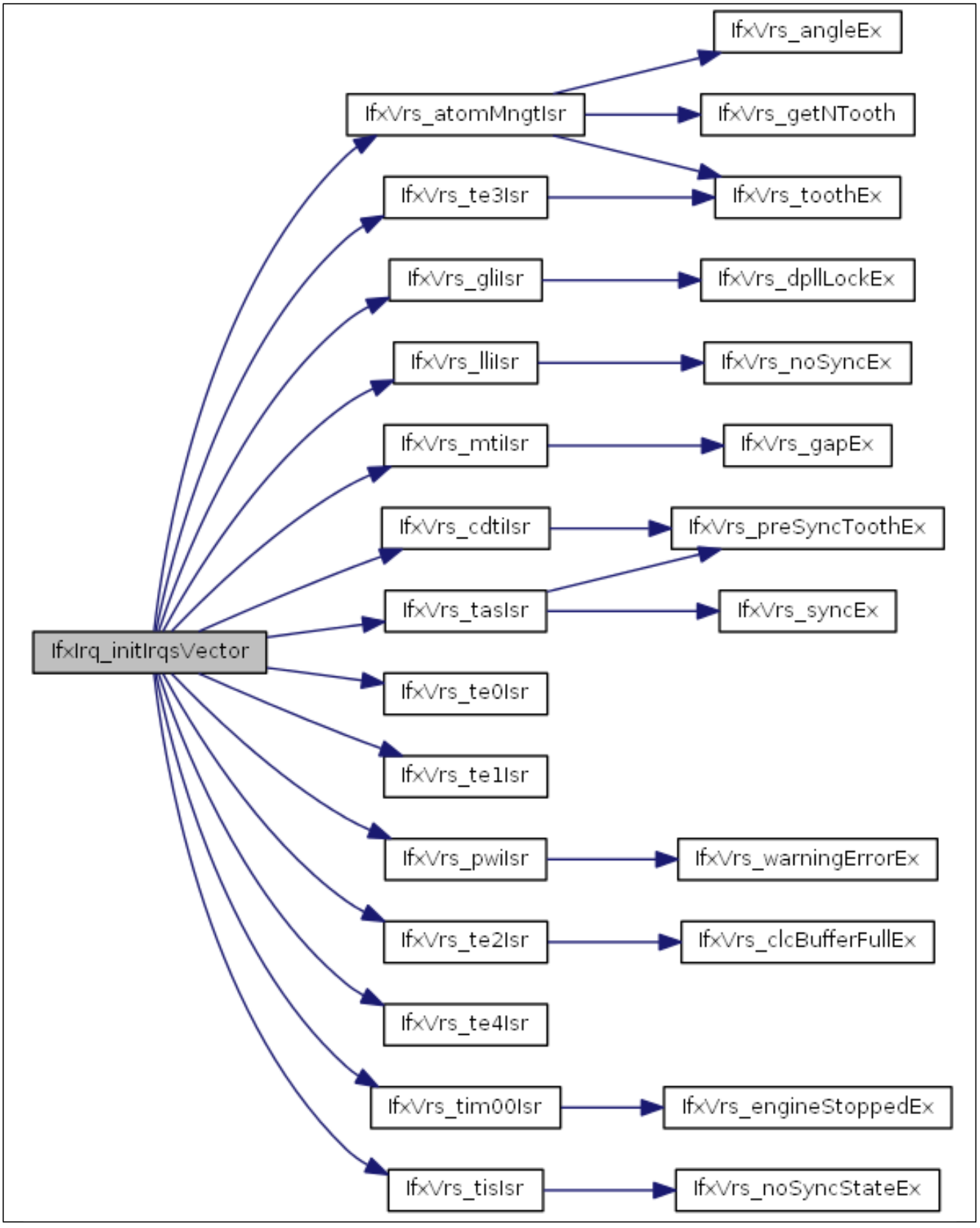

图 37　DPLL 中断和回调函数

4.8 硬件资源

表 7 IfxVrs 驱动程序:硬件资源

HW 模块	描述
TIM0_CH0	曲轴输入
TIM0_CH1	凸轮轴输入
ATOMx_CH0	CCU0:用于“waitStartTime” CCU1:用于在角度上触发一个事件
ATOMx_CH1	CCU0:不使用 CCU1:用于间隙齿模拟
CMU_CLK0	系统时钟(分辨率 100ns)
CMU_CLK1	TIM0_CH0_TDU - 超时分辨率 (1ms)

w ww.i nf ineon.com

Published by Infineon Technologies AG