TriCore™ AURIX™家族系列

32位

通用定时器（GTM），捕获和比较的示例驱动程序

AP32213

应用笔记

V1.0，2014-05

微控器

免责条款

为方便客户浏览，英飞凌以下所提供的将是有关英飞凌产品及服务资料的中文翻译版本。该中文翻译版本仅供参考，并不可作为任何论点之依据。虽然我们尽力提供与英文版本含义一样清楚的中文翻译版本，但因语言翻译和转换过程中的差异，可能存在不尽相同之处。因此，我们同时提供该中文翻译版本的英文版本供您阅读，请参见www.infineon-ecosystem.org。并且，我们在此提醒客户，针对同样的英飞凌产品及服务，我们提供更加丰富和详细的英文资料可供客户参考使用。请详见 www.infineon.com

客户理解并且同意，英飞凌毋须为任何人士由于其在翻译原来的英文版本成为该等中文翻译版本的过程中可能存在的任何不完整或者不准确而产生的全部或者部分、任何直接或者间接损失或损害负责。英飞凌对于中文翻译版本之完整与正确性不担负任何责任。英文版本与中文翻译版本之间若有任何歧异，以英文版本为准，且仅认可英文版本为正式文件。

您如果使用以下提供的资料，则说明您同意并将遵循上述说明。如果您不同意上述说明，请不要使用本资料。

版本2014/05

出版发行：
英飞凌科技公司
上海，中国

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 Infineon Technologies 办事处联系。

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 Infineon Technologies 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 Infineon Technologies 提供的组件仅可用于获得 Infineon Technologies 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	版本	修订人	修订内容
2012年9月	V1.0	A.Baratta	初版

商标：

Infineon®为英飞凌科技公司的注册商标。

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）发送电子邮件至：

ctdd@infineon.com

目录

1 介绍

本文档介绍了英飞凌AURIX家族系列产品的通用定时器模块（GTM），该模块扩展并替代之前GPTA模块功能。

GTM包括可以独立工作子模块，在系统设计时也可以把各子模块组合在一起使用。通过把不同的子模块组合在一起使用可以实现多种复杂功能。这样一来，一个GTM便具有多样且差异化的应用功能。

1.1 用途

本应用笔记提供了最具通用性的GTM子模块的基本驱动示例，从而帮助加快GTM学习进程，并且也是子模块复杂驱动程序开发的初期学习。

注意：本驱动程序文档不应在真实产品上“按原样”使用，仅作为平台开发的初期学习用。

1.2 参考文献

[1] AURIX™ Datasheet (A Step)

[2] GTM 1.4.2 Errata

2 英飞凌驱动程序的通用软件架构

2.1 介绍

本文档中参考的所有驱动程序示例均已实现，它们与英飞凌标准的软件架构具有兼容性。

该架构遵循“编译文件方法”，下面的小节解释说明了这种明确定义的文件结构。

2.2 目录结构

下图给出英飞凌默认通用的软件架构文件夹结构：

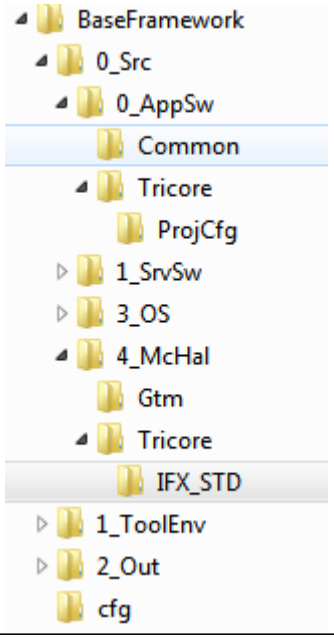

图1通用软件框架下的文件夹结构

2.2.1 控制器相关文件

源文件（*.c 和*.h或汇编源文件）是为特定的微控器而开发的，它们与库文件和目标文件一起存储在4_McHal文件夹和子文件夹下。

如下图所示示例，与多通道指令序列（MCS）相关的汇编代码会保存在4_McHal/Gtm文件夹下。

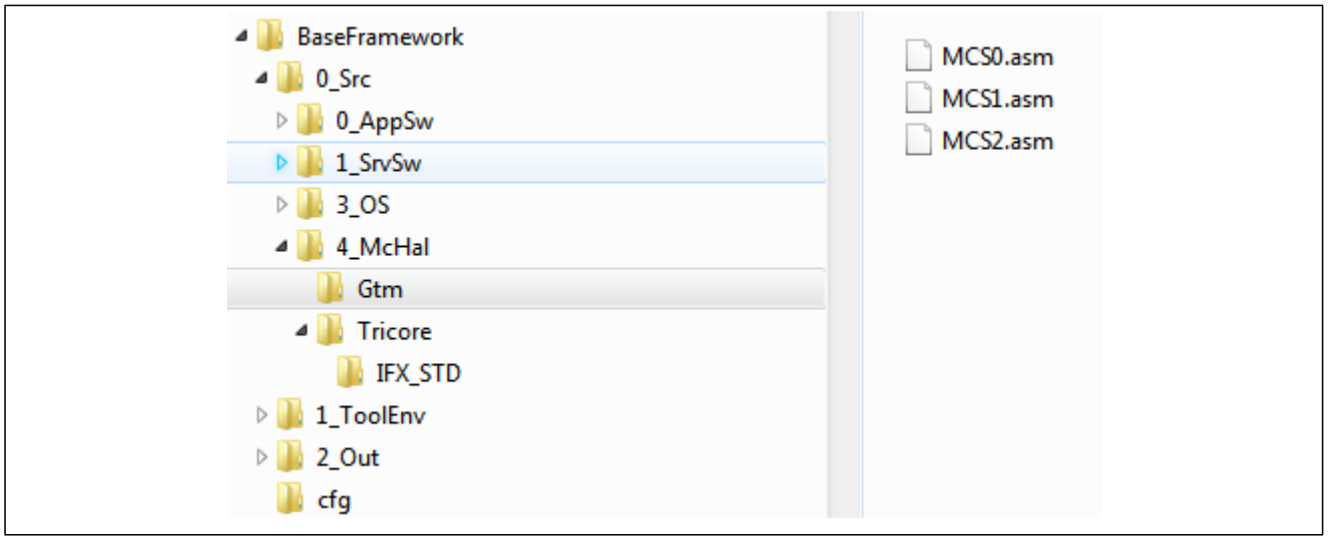

图2通用架构的文件夹结构下的MCS文件

2.2.2 控制器无关文件

当文件是控制器无关文件时，用户可以决定文件存储的最佳位置。

2.2.3 操作系统结构文件

当开发构建操作系统组件的源文件（*.c 和 *.h或汇编源文件）时，把他们与库文件和目标文件一起保存在3_OS文件夹下。

作为微控器家族系列端口开发的文件也存储在这里。

注释：用作OS Tick的文件保存在4_McHal。

2.2.4 桥接口和宏文件

在应用中，一些文件会用作桥接口，以便能够使源代码应用于不同编译器，并应用于不同架构下一些重复使用的组件中。这些源文件保存在1_SrvSW。

2.2.5 项目指定文件

算法实现和应用的源文件保存在0_AppSw。注意main.c文件也保存在0_AppSw。

3 PWM生成-PWM测试驱动器

3.1 介绍

开发的脉宽调制（PWM）驱动提供一套利用定时器输出模块（TOM）和ARU连接的定时器输出模块（ATOM）来生成PWM信号的基本函数。

PWM驱动不会使用到任何ARU交互。任何时候，CPU都可以更新周期，占空比和已配置的“PWM逻辑通道”的时钟。

本章中的各小节概述驱动器API及其使用方法。

3.2 PWM驱动程序系统结构

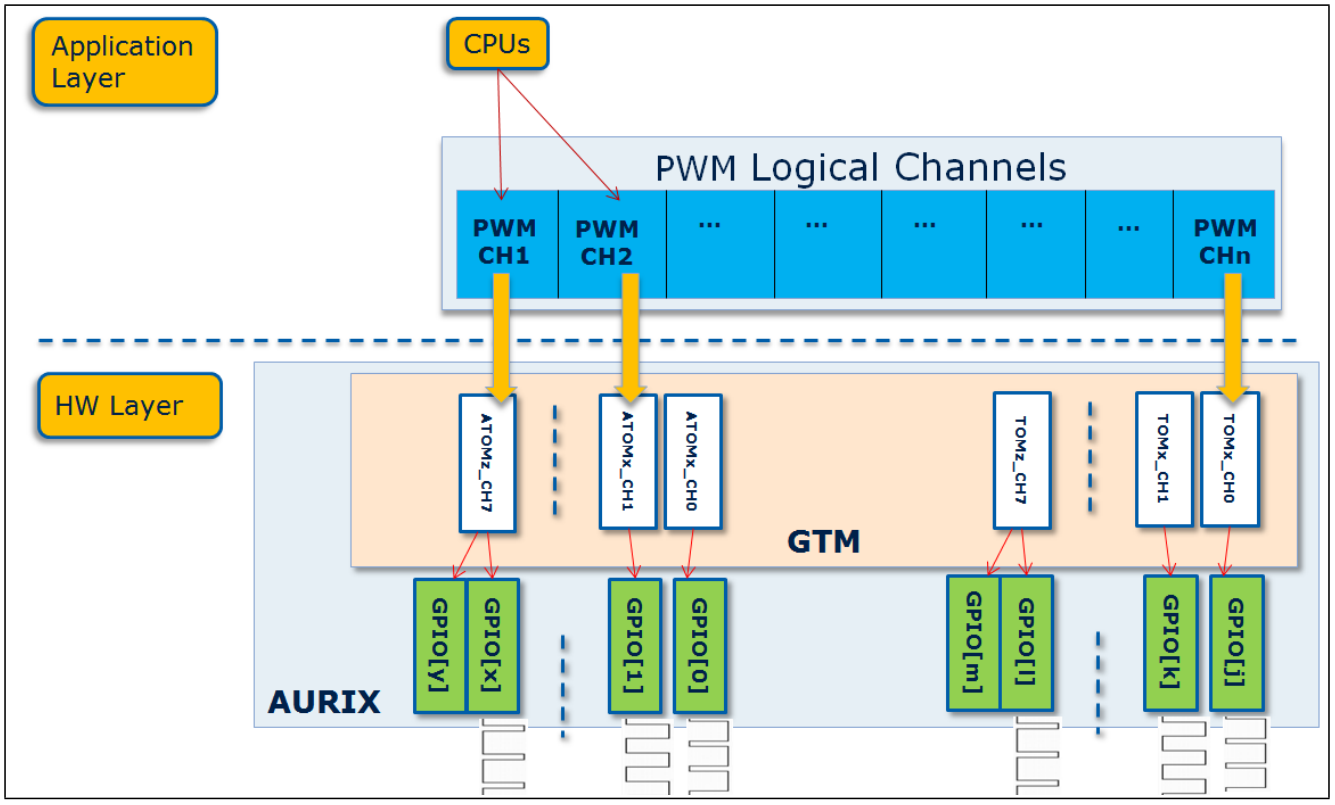

图3 PWM驱动系统结构

3.3 TOM和ATOM

使用TOM和ATOM生成独立信号或是独立的PWM信号。两者主要区别是：

- TOM和ATOM使用不同类型时钟。
- TOM内部计数器是16位，ATOM是24位。
- ATOM通过ARU与其它GTM模块共享信息。
- ATOM可以在四个不同运行模式下工作。

注释：关于TOM与ATOM模块的更多信息，请参考AURIX微控器的数据手册。

3.3.1 定时器输出模块（TOM）

使用TOM生成从属或是相互间独立的PWM信号。

TOM特性包括：

- 每条通道的专用PWM时基计数器（16位宽）。
- PWM更新和时钟预分频器选择的影子寄存器。
- 改变时钟预分频器允许宽范围PWM周期具有不同分辨率。
- 通过链接各个PWM时基计数器生成相互依赖的PWM。
- 用TOM子模块中的一个专用通道的脉冲计数调制输出信号来模拟DAC。
- 全面触发单元以启/停，启用和禁用多个并行的TOM通道。
- 用于PWM占空比和周期的异步修改的大于/等于比较器。

每个TOM通道都有各自的16位计数器，它可产生PWM沿。

计数器频率从CMU提供的五个预分频时钟中选择。

每个TOM通道具有两个捕获比较单元（CCU0和CCU1）来把计数器与可配置的值进行比较。CCU0用来确定PWM周期时间，CCU1定义占空比持续时间。

当TOM通道计数器由前一通道复位时，把两个CCU设置为在PWM周期产生任意的脉冲，PWM周期也由前一通道决定。

每个CCU都有影子寄存器用来存储下一个PWM周期的PWM特性。这些影子寄存器可用于定义新的占空比和周期长度。此外，还有一个影子寄存器给新的PWM周期选择不同的时钟预分频器。这意味着，可以在一个PWM循环生成具有高分辨率的PWM周期，并在下一循环中，可以用一个较低的时钟频率产生一个非常长的，分辨率低的PWM周期。

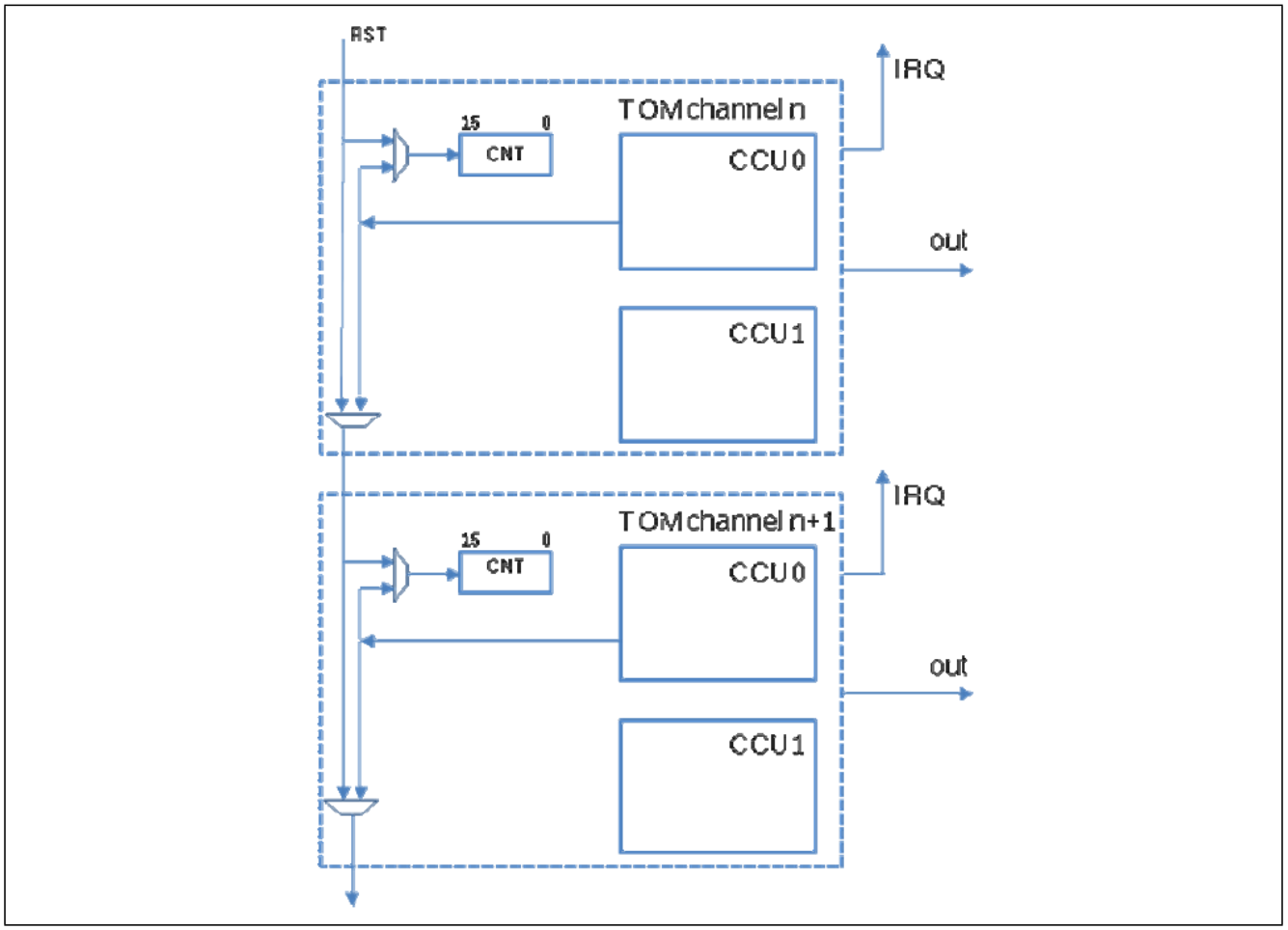

图4 TOM通道结构

3.3.2 ARU连接定时器输出模块（ATOM）

从硬件结构角度来看，ATOM的硬件结构非常类似于TOM子模块。然而，与TOM比起来，ATOM内部寄存器是24位长度，并且根据通道模式通道影子寄存器或比较寄存器可从与ARM连接的源端重新装载。

ATOM特征包括：

- 每个通道有专用PWM时基计数器（24位宽）。
- 两个比较寄存器的专用影子寄存器以及通道控制位的影子寄存器。
- 改变时钟预分频器允许宽范围PWM周期具有不同分辨率。
- 全局触发单元可以启/停，启用和禁用多个并行的ATOM通道。
- 带符号的大于/等于比较器来检测时基溢出。
- ARU连接自动重载新数据值以及控制信号，而不需要CPU介入。
- 提供四种不同工作模式，不同通道可以以不同模式运行。

每个ATOM通道由2个比较单元（CCU0和CCU1），自有的24位时基，以及信号输出和参数重载独立运行的影子寄存器构成。每个ATOM通道具有两个额外接口-一个是TBU的，另一个是GTM路由单元。

通过TBU接口，ATOM通道与GTM全局时基进行比较。ATOM通道内的比较单元（CCU0和CCU1）进行带符号的大于-等于比较，所以ATOM通道也可检测全局时基溢出。

连接至GTM路由单元可以用运行和控制数据加载ATOM通道，并提供比较/匹配数给其他ARU目标单元。

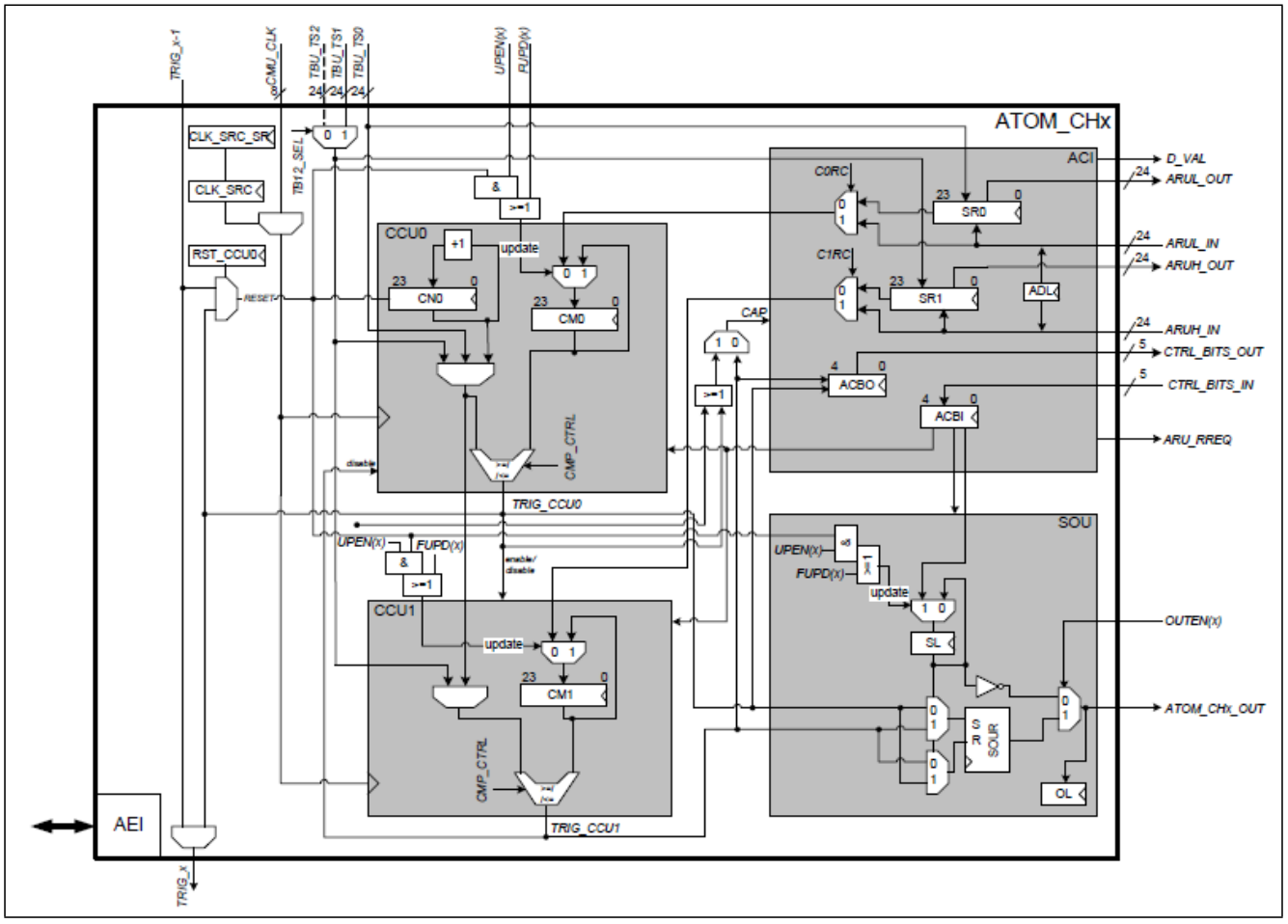

图5 ATOM通道结构

3.3.3 PWM生成

设置完成后，并启用寄存器TOM[i]_TGC[y]_ENDIS_STAT1中相应的位后，计数器CN0对选择的输入时钟（分辨率）的每个沿开始计数。

生成的输出信号的电平可以通过通道配置寄存器（A）TOM[i]_CH[x]_CTRL的SL配置位来设定。

如果计数器CN0从CM0往回复位到0，周期的第一条沿在TOM[i]_CH[x]_OUT上生成。如果CN0达到CM1，周期的第二条沿生成。

一般来讲，CM0代表PWM周期，CM1代表占空比。

[1] ATOM_AGC_ENDIS_STAT register for the ATOMs

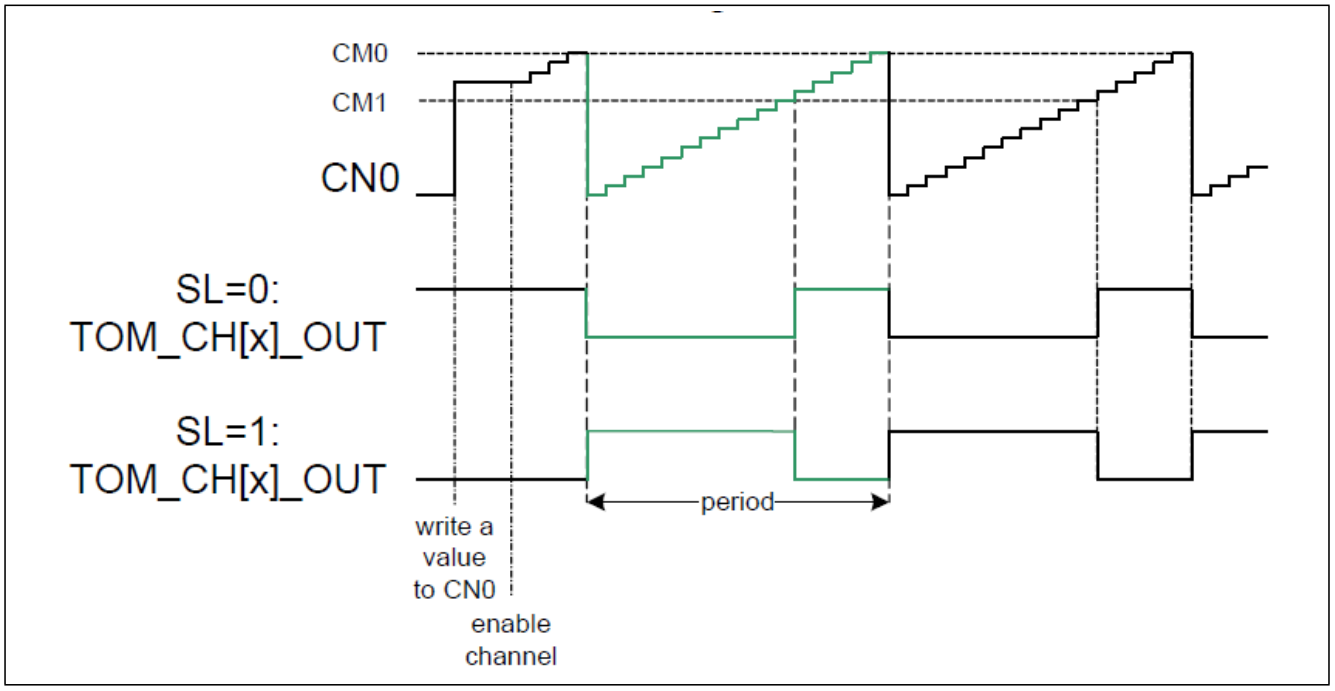

图6 （A）TOM PWM生成

CN0计数器的起始偏移可以通过访问指定寄存器进行设置。

CM0，CM1和 CLK_SRC寄存器更新时，可以在两种更新机制间选择：

- **同步更新**
  -使用SR0，SR1和 CLK_SRC_SR影子寄存器。
- **异步更新**
  -使用CM0，CM1和CLK_SRC寄存器。

当请求计数器寄存器CN0复位时（通过信号RESET），用各自影子寄存器内容更新CM0，CM1和CLK_SRC寄存器。如果CN0比较结果大于或是等于CM0，或是通过信号TRIG_[x-1]由另一个TOM通道[x-1] 触发复位时，CN0会复位。

注释：详细信息，请参阅AURIX数据手册。

3.4 公共函数-API

表1 PWM测试驱动器API

函数名称	函数签名	描述
IfxGtm_PwmTest_init	void **IfxGtm_PwmTest_init**(IfxGtm_PwmTest_Config* pwmChCfg);	驱动函数，初始化所有属性
IfxGtm_PwmTest_config	IfxGtm_PwmTest_Status **IfxGtm_PwmTest_config**(IfxGtm_PwmTest_Config* pwmChCfg);	PWM通道<pwmChCfg>配置
IfxGtm_PwmTest_start	IfxGtm_PwmTest_Status **IfxGtm_PwmTest_start**(IfxGtm_PwmTest_Config* pwmChCfg)	在通道上开始生成PWM：<pwmChCfg>
IfxGtm_PwmTest_updatePeriod	IfxGtm_PwmTest_Status **IfxGtm_PwmTest_updatePerio d**(IfxGtm_PwmTest_Config* pwmChCfg, uint32 pwmPeriod);	更新PWM通道<pwmChCfg>的PWM周期（CM0/SR0寄存器）
IfxGtm_PwmTest_updateDuty	IfxGtm_PwmTest_Status **IfxGtm_PwmTest_updateDuty**(IfxGtm_PwmTest_Config* pwmChCfg, uint32 dutyCycle);	更新PWM通道<pwmChCfg>的PWM占空比（CM1/SR1寄存器）。
IfxGtm_PwmTest_updateOffset	IfxGtm_PwmTest_Status **IfxGtm_PwmTest_updateOffse t**(IfxGtm_PwmTest_Config* pwmChCfg, uint32 offset);	更新PWM通道<pwmChCfg>的ATOM/TOM计数器起始值（CN0寄存器）。
IfxGtm_PwmTest_updateDutyAndPeriod	IfxGtm_PwmTest_Status **IfxGtm_PwmTest_updateDutyA ndPeriod**(IfxGtm_PwmTest_Config* pwmChCfg, uint32 dutyCycle,uint32 period);	更新PWM通道<pwmChCfg>的PWM占空比和周期。
IfxGtm_PwmTest_getSigLevel	uint32 **IfxGtm_PwmTest_getSigLevel**(IfxGtm_PwmTest_Config* pwmChCfg, IfxGtm_PwmTest_Status *status);	获得PWM通道<pwmChCfg>的当前PWM信号电平。
IfxGtm_PwmTest_getDuty	uint32 **IfxGtm_PwmTest_getDuty**(IfxGtm_PwmTest_Config* pwmChCfg, IfxGtm_PwmTest_Status *status);	获得PWM通道<pwmChCfg>的PWM占空比(CM1寄存器)。

函数名称	函数签名	描述
IfxGtm_PwmTest_getPeriod	uint32 **IfxGtm_PwmTest_getPeriod**(IfxGtm_PwmTest_Config* pwmChCfg, IfxGtm_PwmTest_Status *status);	返回PWM通道<pwmChCfg>的当前PWM周期。
IfxGtm_PwmTest_init	void **IfxGtm_PwmTest_init**(IfxGtm_PwmTest_Config* pwmChCfg);	驱动函数，初始化所有属性
IfxGtm_PwmTest_config	IfxGtm_PwmTest_Status **IfxGtm_PwmTest_config**(IfxGtm_PwmTest_Config* pwmChCfg);	PWM通道<pwmChCfg>配置
IfxGtm_PwmTest_start	IfxGtm_PwmTest_Status **IfxGtm_PwmTest_start**(IfxGtm_PwmTest_Config* pwmChCfg)	通道上开始生成PWM：<pwmChCfg>
IfxGtm_PwmTest_updatePeriod	IfxGtm_PwmTest_Status **IfxGtm_PwmTest_updatePerio d**(IfxGtm_PwmTest_Config* pwmChCfg, uint32 pwmPeriod);	更新PWM通道<pwmChCfg>的PWM周期（CM0/SR0寄存器）。

3.5 PWM测试驱动程序-PWM通道配置

为了能够配置ATOM/TOM，必须给驱动器传递一个配置变量。配置变量包括下面参数：

表2 PWM通道配置变量

参数名称	描述
ifxGtm_OutputModuleBasePtrsAddr	指向ATOM/TOM HW寄存器指针
ifxGtm_PwmOutputModuleType	指明是否必须使用TOM或是ATOM。
ifxGtm_OutputModuleOffset	用于访问不同ATOM/TOM模块
ifxGtm_OutputChannelOffset	寻址不同ATOM/TOM通道
ifxGtm_SignalLevel	规定激活/非激活状态的默认信号电平
ifxGtm_PwmSyncUpdate	规定使用同步还是非同步更新
ifxGtm_ClockSource	规定使用的时钟
pwmChStatus	PWM通道状态(已配置/未配置)
trigOut	规定下个相邻通道的触发器。 0 :TRIG_CCU0 -> 当前通道 1 :TRIG[x-1] -> 之前通道
rstCCU0	CCU0复位源 0 :一旦与CM0匹配比较，把计数器寄存器CN0复位为0。 1 :触发器TRIG_[x-1]上复位计数器CN0到0。
ifxGtm_OutputModuleNumber	ATOM/TOM模块编号
ifxGtm_OutputChannelIdx	ATOM/TOM通道编号
pwmPeriod	ATOM/TOM的PWM周期
pwmDutyCycle	ATOM/TOM的PWM占空比
pwmCntOffset	ATOM/TOM计数器起始偏移

3.6 PWM测试驱动器-使用实例

采用下面的代码进行编程后，用四个PWM通道实现中心对齐的PWM。

```
#define PWM_PERIOD 5000

    float dutyB = 0.250f;
     float dutyC = 0.125f;
     float dutyD = (0.750f) / 2;

     IfxGtm_PwmTest_init((IfxGtm_PwmTest_Config*) &cfgAtom20);
     IfxGtm_PwmTest_init((IfxGtm_PwmTest_Config*) &cfgAtom21);
     IfxGtm_PwmTest_init((IfxGtm_PwmTest_Config*) &cfgAtom22);
     IfxGtm_PwmTest_init((IfxGtm_PwmTest_Config*) &cfgAtom23);

     //ATOM2_CH0 => provides the period and to trigger the PWMs updates
     cfgAtom20.ifxGtm_PwmOutputModuleType = IfxGtm_Atom;
     cfgAtom20.ifxGtm_OutputModuleNumber = IfxGtm_Atom2;
     cfgAtom20.ifxGtm_OutputChannelIdx = IfxGtm_AtomCh0;
     cfgAtom20.ifxGtm_PwmSyncUpdate = IfxGtm_PwmSyncUpdate;
     cfgAtom20.ifxGtm_ClockSource = IfxGtm_CmuClk0;
     cfgAtom20.ifxGtm_SignalLevel          =          IfxGtm_SignalLow;
     cfgAtom20.trigOut = 1; ➔ To trigger the other channels
     cfgAtom20.pwmPeriod = PWM_PERIOD;
     cfgAtom20.pwmDutyCycle= (PWM_PERIOD * 0.5); //50% Duty Cycle
     cfgAtom20.pwmCntOffset= 0;
     IfxGtm_PwmTest_config((IfxGtm_PwmTest_Config*) &cfgAtom20);

     cfgAtom21.ifxGtm_PwmOutputModuleType = IfxGtm_Atom;
     cfgAtom21.ifxGtm_OutputModuleNumber = IfxGtm_Atom2;
     cfgAtom21.ifxGtm_OutputChannelIdx = IfxGtm_AtomCh1;
     cfgAtom21.ifxGtm_PwmSyncUpdate = IfxGtm_PwmSyncUpdate;
     cfgAtom21.ifxGtm_ClockSource = IfxGtm_CmuClk0;
     cfgAtom21.ifxGtm_SignalLevel = IfxGtm_SignalHigh;
     cfgAtom21.trigOut = 0;
     cfgAtom21.rstCCU0 = 1;
     cfgAtom21.pwmPeriod = PWM_PERIOD*(1 - dutyB);
     cfgAtom21.pwmDutyCycle= (PWM_PERIOD * dutyB);
     cfgAtom21.pwmCntOffset= 0;
     IfxGtm_PwmTest_config((IfxGtm_PwmTest_Config*) &cfgAtom21);

     cfgAtom22.ifxGtm_PwmOutputModuleType = IfxGtm_Atom;
     cfgAtom22.ifxGtm_OutputModuleNumber = IfxGtm_Atom2;
     cfgAtom22.ifxGtm_OutputChannelIdx = IfxGtm_AtomCh2;
```

```
cfgAtom22.ifxGtm_PwmSyncUpdate = IfxGtm_PwmSyncUpdate;
cfgAtom22.ifxGtm_ClockSource = IfxGtm_CmuClk0;
cfgAtom22.ifxGtm_SignalLevel          =          IfxGtm_SignalHigh;
cfgAtom22.trigOut = 0;
cfgAtom22.rstCCU0 = 1;
cfgAtom22.pwmPeriod = PWM_PERIOD*(1 - dutyC);
cfgAtom22.pwmDutyCycle= (PWM_PERIOD * dutyC);
cfgAtom22.pwmCntOffset= 0;
IfxGtm_PwmTest_config((IfxGtm_PwmTest_Config*) &cfgAtom22);

cfgAtom23.ifxGtm_PwmOutputModuleType = IfxGtm_Atom;
cfgAtom23.ifxGtm_OutputModuleNumber = IfxGtm_Atom2;
cfgAtom23.ifxGtm_OutputChannelIdx = IfxGtm_AtomCh3;
cfgAtom23.ifxGtm_PwmSyncUpdate = IfxGtm_PwmSyncUpdate;
cfgAtom23.ifxGtm_ClockSource = IfxGtm_CmuClk0;
cfgAtom23.ifxGtm_SignalLevel = IfxGtm_SignalHigh;
cfgAtom23.trigOut = 0;
cfgAtom23.rstCCU0 = 1;
cfgAtom23.pwmPeriod = PWM_PERIOD*(1 - dutyD);
cfgAtom23.pwmDutyCycle= (PWM_PERIOD * dutyD);
cfgAtom23.pwmCntOffset= 0;
IfxGtm_PwmTest_config((IfxGtm_PwmTest_Config*) &cfgAtom23);

IfxGtm_PwmTest_start(&cfgAtom20); //Start All Together
```

图7 PWM测试驱动程序API使用示例

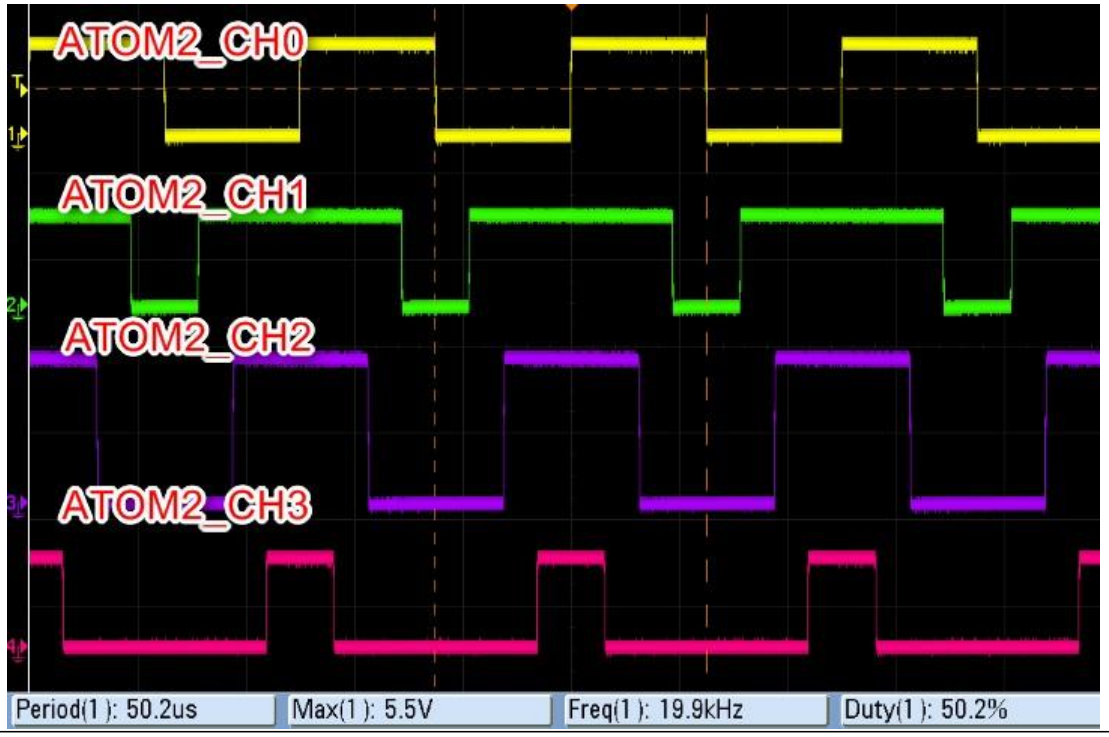

图8 PWM中心对齐示例

下图的基本理念阐述了使用GTM生成了中心对齐PWM。

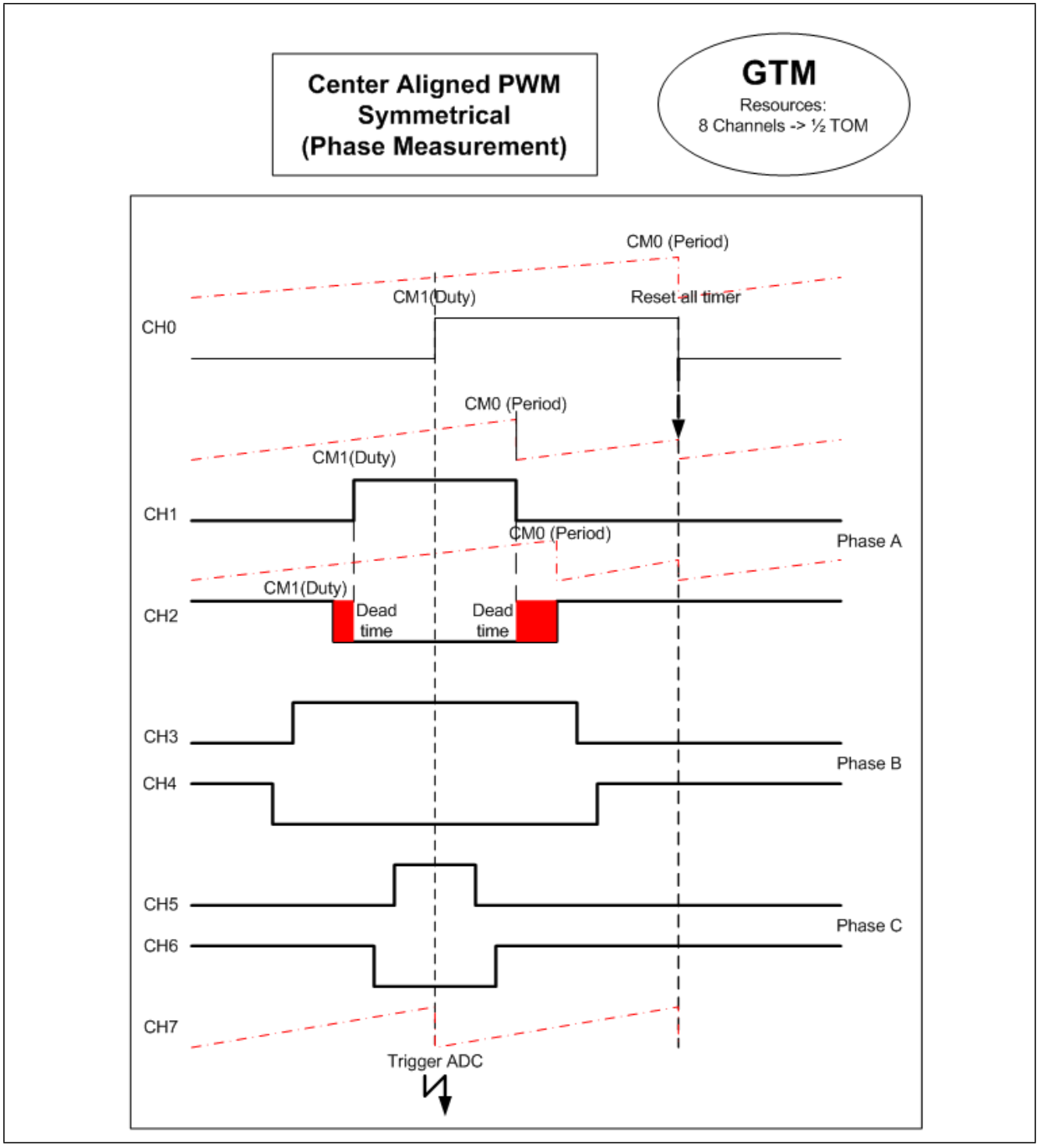

图9 使用GTM生成中心对齐PWM

第一个通道（ATOM2_CH0）触发其它所有通道。其它通道有其指定并明确定义的周期以及占空比（CM0，CM1），即同步更新（通常情况下），当ATOM2_CH0计数器达到ATOM2_CH0_CM1值（占空比）。

当ATOM2_CH0_CN0计数器达到所定义的周期值ATOM2_CH0_CM0，其它通道也重置复位为默认值（由SL位定义）。

4 基本TIM（定时器输入模块）示例

4.1 介绍

所开发的基本TIM驱动程序给出不同TIM模块时的工作方式。驱动器包括下面的模式：

- TPWM - TIM PWM 测量模式
- TIEM - TIM 输入事件模式
- TIPM - TIM 输入预分频模式

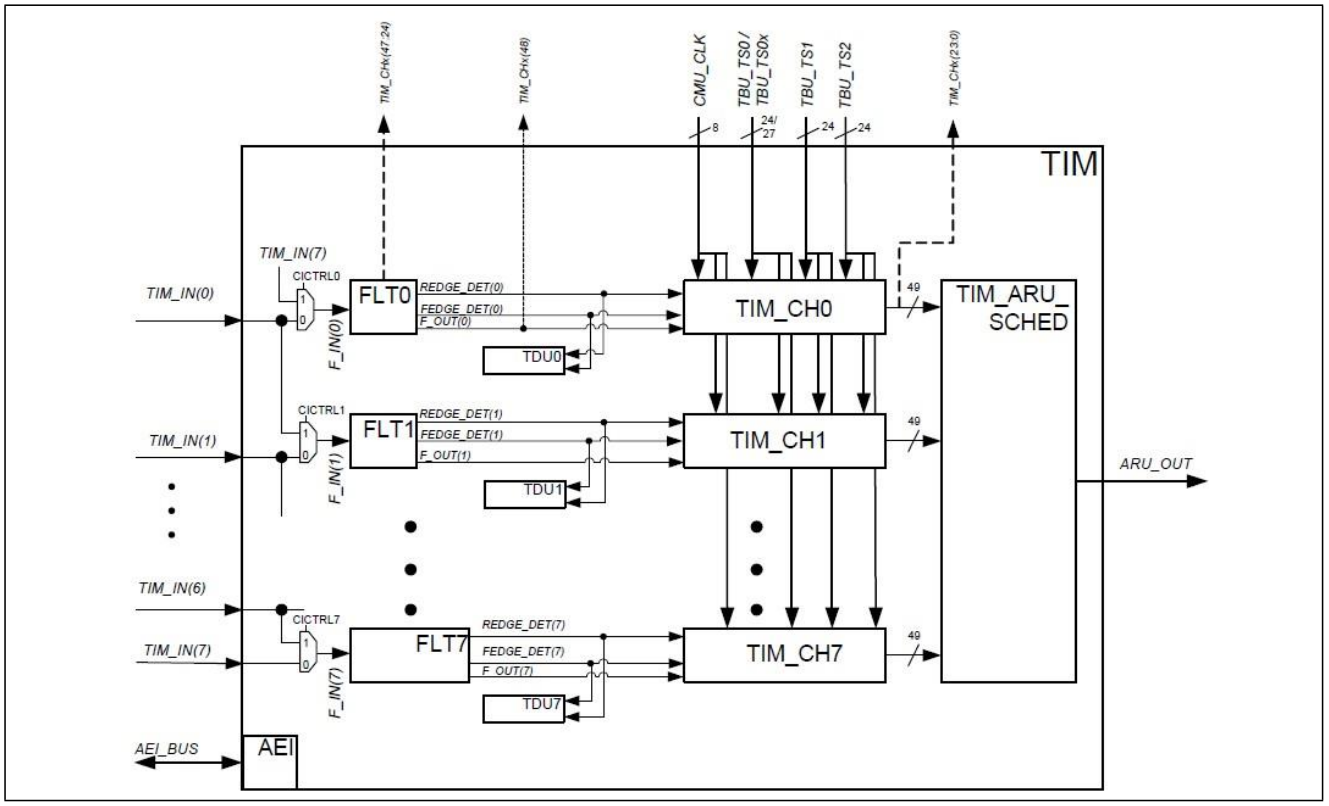

图10 TIM模块框图

驱动程序API包括两类主要函数：

- IfxGtm_timExampleInit
  -为了可以在特定模式下工作，初始化TIM模块。
- IfxGtm_tim00Src
  -TIM0_CH0中断服务程序。

IfxGtm_timExampleInit如下面示例代码所示：

```
void  IfxGtm_timExampleInit(short  op_mode,  uint32  clk_div,  uint32
event_TIPM, uint32 timeout,short aru_en) {

      set_clock(clk_div); //Resolution 1us
      _op_mode = op_mode;
      _event_to_count = event_TIPM;
      //TIM PWM Measurement Mode
```

```
        if (op_mode == TPWM) {
                GTM_TIM0_CH0_CTRL.B.TIM_MODE= TPWM; //TPWM
                GTM_TIM0_CH0_CTRL.B.GPR0_SEL  =  3;  //==>  CNTS  Input  (Duty
Cycle)
                GTM_TIM0_CH0_CTRL.B.GPR1_SEL  =  3;  //==>  CNT  as  Input
(Period)
                GTM_TIM0_CH0_CTRL.B.DSL = 1; //Start From Rising Edge
        }
        //TIM Input Prescaler Mode Rising/Falling
        if (op_mode == TIPM_RF) {
                GTM_TIM0_CH0_CTRL.B.TIM_MODE = TIPM;
                GTM_TIM0_CH0_CTRL.B.ISL = 1; //ignore DSL,both edges active
                GTM_TIM0_CH0_CNTS.B.CNTS = _event_to_count;
        }

        if (op_mode == TIPM_R) {
                GTM_TIM0_CH0_CTRL.B.TIM_MODE = TIPM;
                GTM_TIM0_CH0_CTRL.B.ISL  =  0;  //DSL  select  active  signal
level
                GTM_TIM0_CH0_CTRL.B.DSL = 1; //Start From Rising Edge
                GTM_TIM0_CH0_CNTS.B.CNTS = _event_to_count;
        }
        if (op_mode == TIPM_F) {
                GTM_TIM0_CH0_CTRL.B.TIM_MODE = TIPM;
                GTM_TIM0_CH0_CTRL.B.ISL = 0; //active signal level
                GTM_TIM0_CH0_CTRL.B.DSL  =  0;  //Meas.  starts  with  falling
edge
                GTM_TIM0_CH0_CNTS.B.CNTS = _event_to_count;
        }
        //TIM Input Prescaler Mode
        if (op_mode == TIEM_RF) {
                GTM_TIM0_CH0_CTRL.B.TIM_MODE = TIEM;
                GTM_TIM0_CH0_CTRL.B.ISL = 1;
                GTM_TIM0_CH0_CTRL.B.GPR1_SEL = 3;
        }
        // TIM Input Event Mode
        if (op_mode == TIEM_R) {
                GTM_TIM0_CH0_CTRL.B.TIM_MODE = TIEM;
                GTM_TIM0_CH0_CTRL.B.ISL = 0; //DSL ->active signal level
                GTM_TIM0_CH0_CTRL.B.DSL = 1; //Start From Rising Edge
        }
        if (op_mode == TIEM_F) {
                GTM_TIM0_CH0_CTRL.B.TIM_MODE = TIEM;
                GTM_TIM0_CH0_CTRL.B.ISL = 0; //DSL-> active signal level
                GTM_TIM0_CH0_CTRL.B.DSL  =  0;  //Meas.  starts  with  falling
```

```
edge
        }
        GTM_TIM0_CH0_CNT.U = 0;
        GTM_TIM0_CH0_IRQ_EN.U = 0x2F;     // Enable All Interrupts
        GTM_TIM0_CH0_CTRL.B.TIM_EN = 1; //Start TIM
        if (timeout!=0) {
                GTM_TIM0_CH0_TDU.U = timeout<<8 | 0x1;
    }
        if (aru_en!=0) {
                GTM_TIM0_CH0_IRQ_MODE.B.IRQ_MODE = 1;
                GTM_TIM0_CH0_CTRL.B.ARU_EN = 1;
                /*Enable also TIM1*/
                GTM_TIM1_CH0_CTRL.B.ARU_EN = 1;
                GTM_TIM1_CH0_CTRL.B.ISL = 1;
                GTM_TIM1_CH0_CTRL.B.GPR1_SEL = 3;
                GTM_TIM1_CH0_CTRL.B.TIM_MODE = TIEM;
                GTM_TIM1_CH0_CNT.U = 0;
                GTM_TIM1_CH0_IRQ_EN.U = 0x2F;  // Enable All Interrupts
                GTM_TIM1_CH0_CTRL.B.TIM_EN = 1; //Start TIM
        }
}
```

图11 初始化TIM模块

该驱动程序的使用实例如下表所示：

表3 TIM驱动使用实例

使用实例	描述
PWM测量(TPWM模式)	TIM寄存器GPR0和GPR1控制输入信号的周期和占空比。
TIM输入预分频模式（TIPM）- 上升/下降	用户指定的（上升和下降）沿数之后，产生一次中断。 GTM_TIM0_CH0_CNTS.B.CNTS = event_to_count;
TIM输入预分频模式（TIPM）- 上升	用户指定的上升沿之后，产生一次中断。 GTM_TIM0_CH0_CNTS.B.CNTS = event_to_count;
TIM输入预分频模式（TIPM）- 下降	用户指定的下降沿之后，产生一次中断。 GTM_TIM0_CH0_CNTS.B.CNTS = _event_to_count;
TIM输入模式（TIEM）- 上升/下降	每次输入沿时，会给一个时间戳。
TIM输入模式（TIEM）-上升	每次输入上升沿时，会给一个时间戳。
TIM输入模式（TIEM）-下降	每次输入下降沿时，会给一个时间戳。
TIM - ARU - MCS - DMA	更加复杂的一个例子是，TIM测量时间戳和沿数，通过ARU并发送给MCS，触发DMA传输。

重要的程序是TIM0_CH0中断服务程序：

```
sint32 IfxGtm_tim00Isr(void) {
        DRIVER_STATUS_REG.B.GLITCH_DET = 0;
        DRIVER_STATUS_REG.B.DATA_OVERFLOW = 0;
        DRIVER_STATUS_REG.B.CNT_OVERFLOW = 0;
        DRIVER_STATUS_REG.B.ECNT_OVERFLOW = 0;
        DRIVER_STATUS_REG.B.TIMEOUT = 0;
        DRIVER_STATUS_REG.B.NEW_VAL = 0;
        //GLITCHDET
        if (GTM_TIM0_CH0_IRQ_NOTIFY.U & 0x20) {
                DRIVER_STATUS_REG.B.GLITCH_DET = 1;
                  GTM_TIM0_CH0_IRQ_NOTIFY.U = 0x20;
        }
        //CNT Overflow
        if (GTM_TIM0_CH0_IRQ_NOTIFY.U & 0x4) {
                DRIVER_STATUS_REG.B.CNT_OVERFLOW = 1;
                GTM_TIM0_CH0_IRQ_NOTIFY.U |= 4;
        }
        //ECNT Overflow
        if (GTM_TIM0_CH0_IRQ_NOTIFY.U & 0x2) {
                DRIVER_STATUS_REG.B.ECNT_OVERFLOW = 1;
                GTM_TIM0_CH0_IRQ_NOTIFY.U |= 2;
        }
        //data overflow
        if (GTM_TIM0_CH0_IRQ_NOTIFY.U & 0x8) {
                GTM_TIM0_CH0_IRQ_NOTIFY.U |= 0x8;
                GTM_TIM0_CH0_CTRL.B.TIM_EN = 0;
                GTM_TIM0_CH0_CTRL.B.TIM_EN = 1;
                DRIVER_STATUS_REG.B.DATA_OVERFLOW = 1;
        }
        //Timeout
        if (GTM_TIM0_CH0_IRQ_NOTIFY.U & 0x10) {
                GTM_TIM0_CH0_IRQ_NOTIFY.U |= 0x10;
                DRIVER_STATUS_REG.B.TIMEOUT = 1;
        }
        //NEW_VAL
        if (GTM_TIM0_CH0_IRQ_NOTIFY.U & 0x1) {
                DRIVER_STATUS_REG.B.NEW_VAL = 1;
                if ((_op_mode == TPWM) || (_op_mode == TIPM)) {
                        current_T[0] = (0x00FFFFFF & GTM_TIM0_CH0_GPR0.U);//DC
                        current_T[1]          =          (0x00FFFFFF          &
GTM_TIM0_CH0_GPR1.U);//Period
                }
```

```
            if ((_op_mode == TIEM_RF)|| (_op_mode == TIEM_R) ||
              (_op_mode == TIEM_F) || (_op_mode == TIPM_RF) ||
              (_op_mode == TIPM_R) || (_op_mode == TIPM_F)) {
                  current_T[event_count++] =
                       (0x00FFFFFF & GTM_TIM0_CH0_GPR0.U); //TS0
                  if (event_count > 1) {
                        current_T[2] = current_T[1] - current_T[0];
                        event_count = 0;
                  }
            }
            GTM_TIM0_CH0_IRQ_NOTIFY.U |= 1;
            return 0;
      }
}
```

图12 TIM中断服务程序代码

GTM PSM FIFO与ATOM模块(stimuli.c)一起使用来模拟输入信号。

通过传递元组阵列（<period,duty>）到stimuli_init函数，它可以产生复杂的PWM信号（参见IfxGtm_PsmExample驱动概述），信号被送到到TIM。

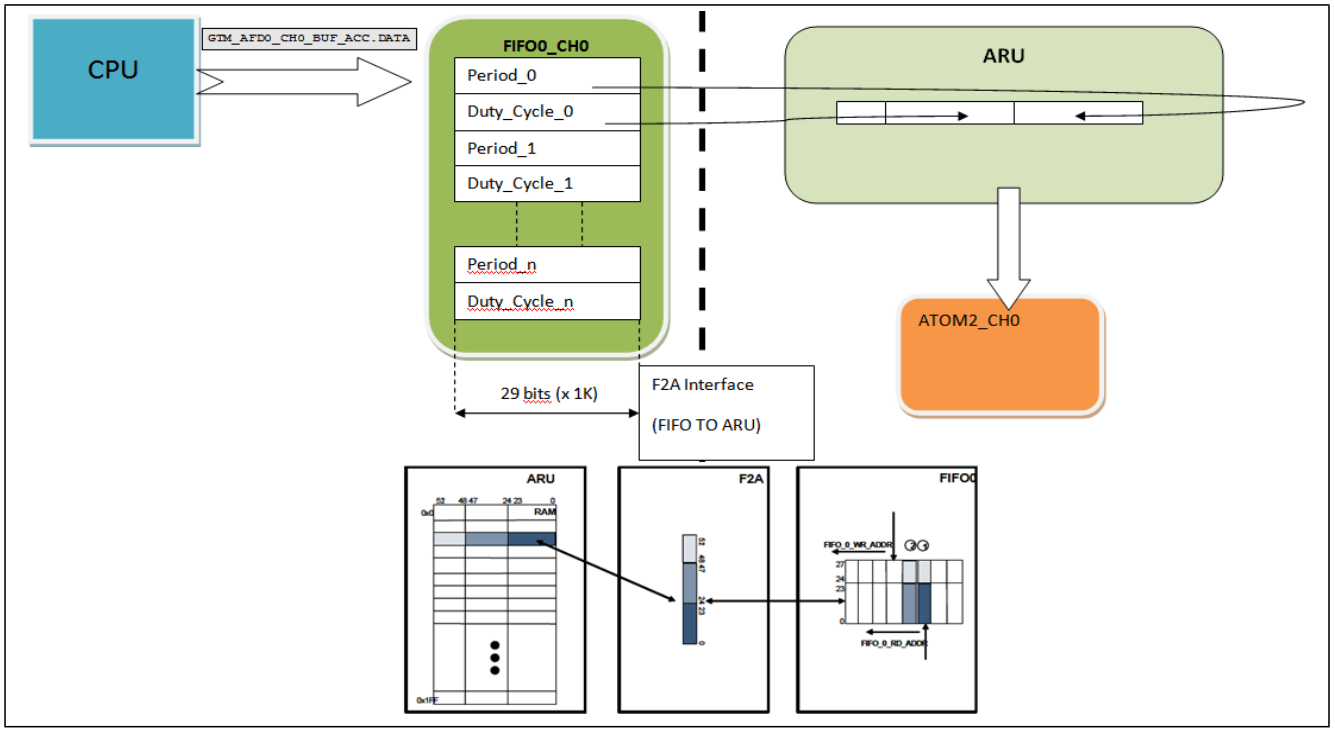

图13 VRS驱动程序API使用示例

4.2 TIM PWM测量模式（TPWM）应用实例

PWM测量模式，TIM通道测量输入的PWM信号的占空比和周期。DSL位定义待测的PWM信号的极性（上升/下降沿）。当请求测量脉冲高时间和周期（PWM具有高电平的占空比，DSL= 1）时，滤波器检测到第一个上升沿后通道开始测量。

测量是通过CNT寄存器用CMU_CLKx配置的时钟来进行计数，直到检测到下降沿。

计数器的值存储在影子寄存器CNTS（如果CNTS_SEL= 0），计数器CNT连续计数，直到下一个上升沿为止。

在下一个上升沿上，假设GPR0_SEL= 1和GPR1_SEL= 1，CNTS寄存器的内容传送到GPR0，CNT寄存器的内容传送到GPR1。GPR0包括占空比长度，GPR1包括周期。

注释：ECNT的位1-7检查GPR0与GPR1寄存器数据一致性。

CNT寄存器清零并且TIM[i]_CH[x]_IRQ_NOTIFY状态寄存器内NEWVAL状态位。TIM_NEWVALx_IRQ中断发生取决于相应中断使能状态。

下图给出使用驱动程序测量PWM的示例。

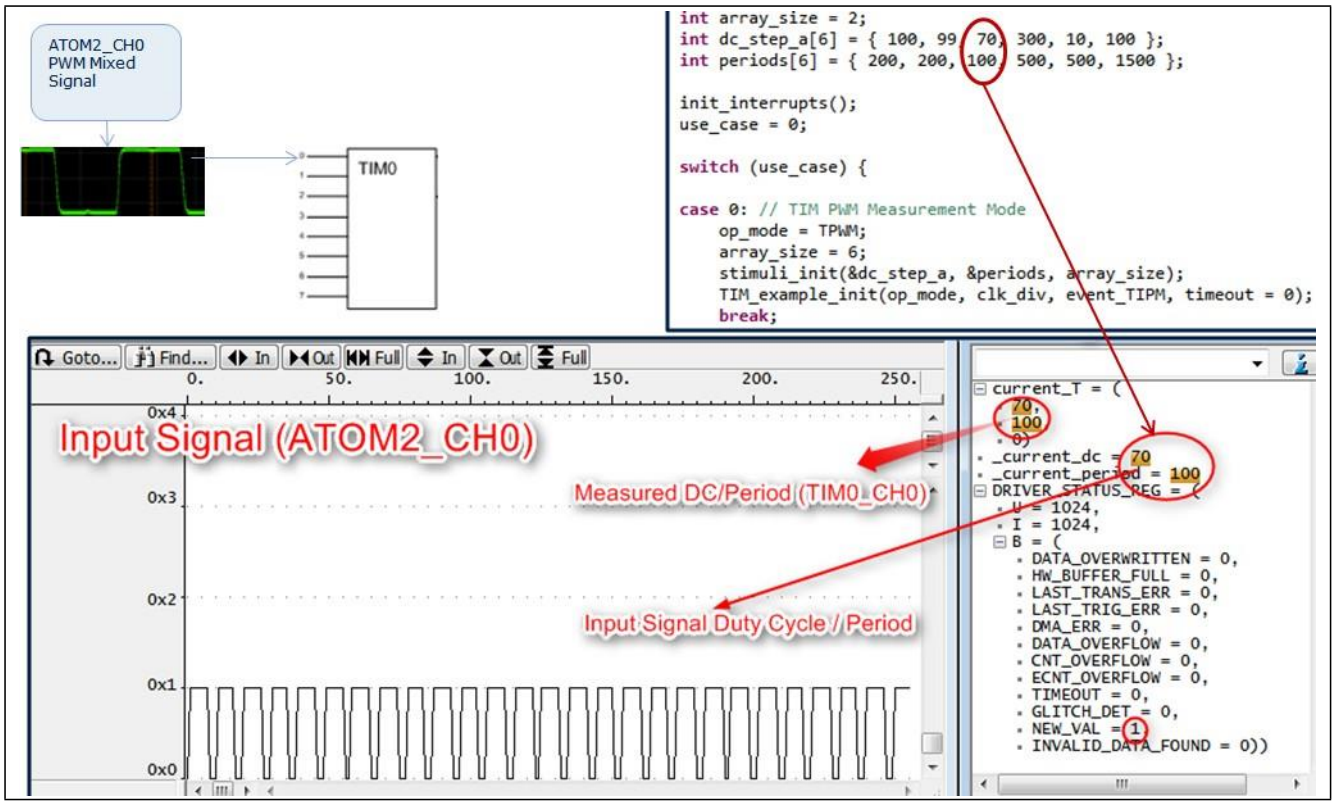

图14 TIM PWM测量模式示例

4.3 TIM输入事件模式（TIEM）应用实例

在输入事件模式时，TIM通道能够对沿进行计数。

通过配置可以规定是否对上升、下降或是两种沿进行了计数。在TIM[i]_CH[x]_CTRL寄存器的DSL和ISL位字段进行配置。

当收到配置的沿，发生TIM[i]_NEWVAL[x]_IRQ中断，自此中断使能。

计数寄存器CNT用来对沿数进行计量。

位字段GPR0_SEL，GPR1_SEL，和CNTS_SEL用来更新GPR0，GPR1以及CNTS寄存器的值。

下面的图给出了TIEM模式驱动程序的运行示例。

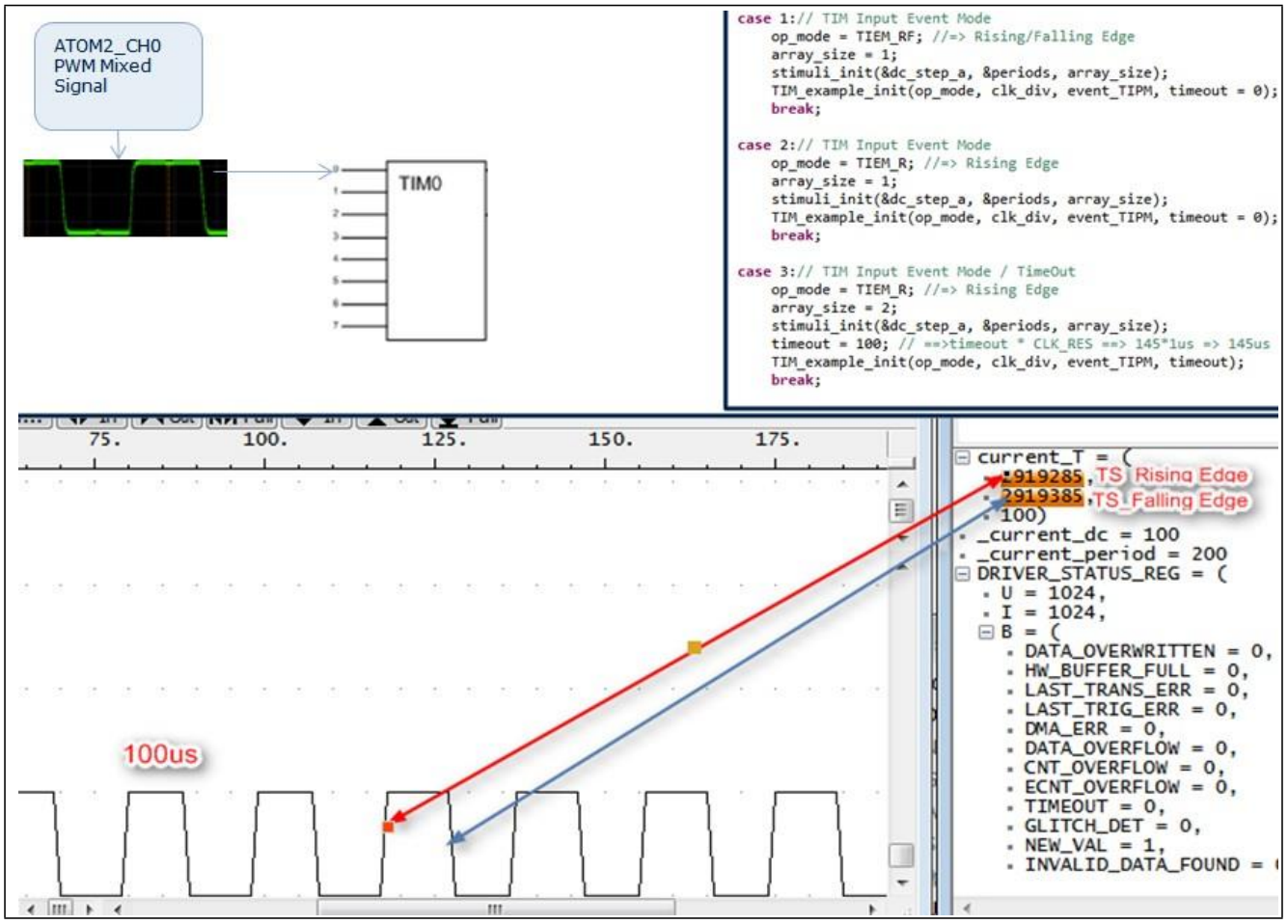

图15 TIEM示例

4.4 TIM输入预分频模式（TIPM）应用实例

TIM输入预分频模式下，TIM[i]_NEWVAL[x]_IRQ发生前检测到的沿数是可编程的。CNTS寄存器中明确指明了中断发生之前的沿数。

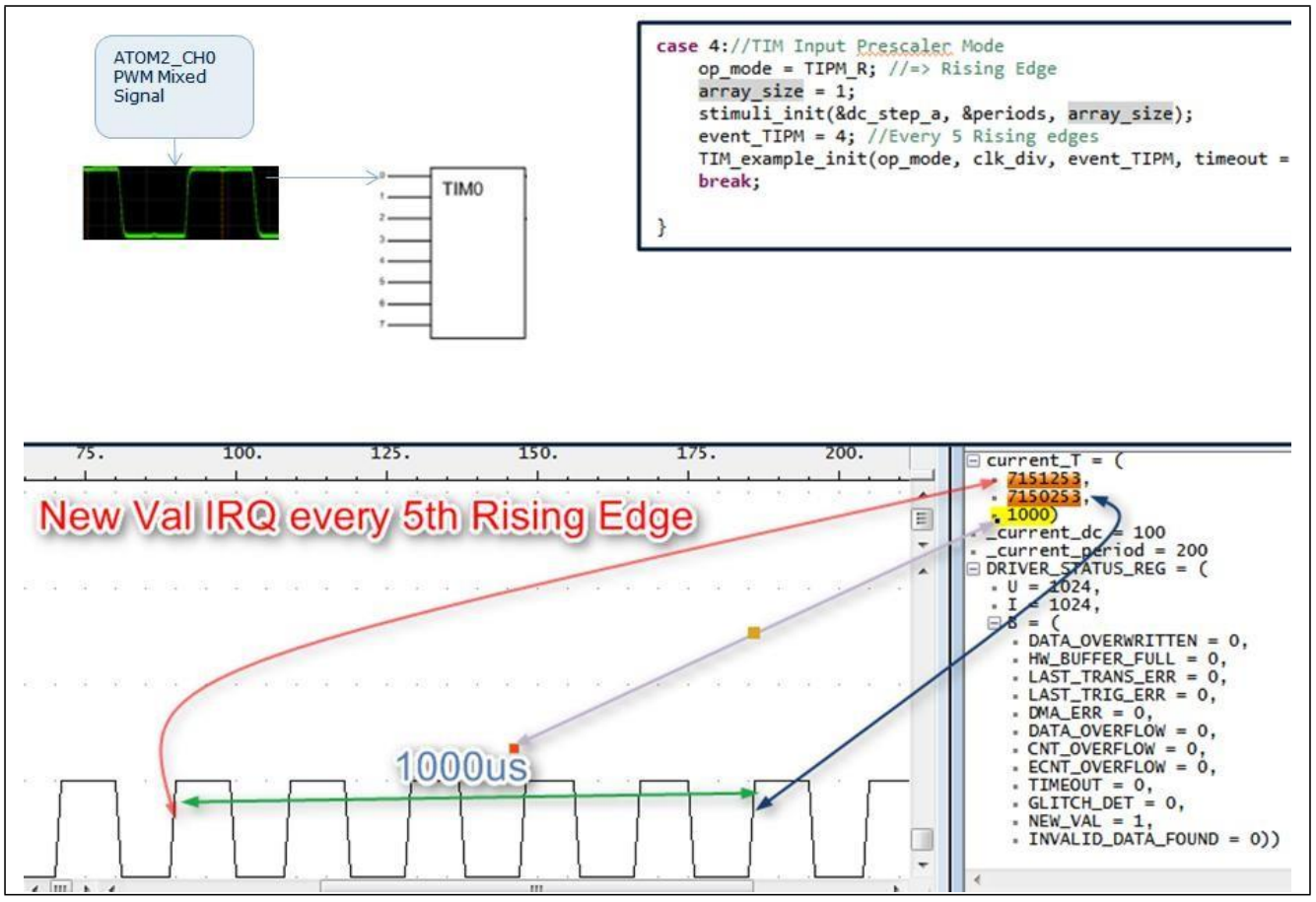

图16 TIPM示例

4.5 TIM－ARU-MCS应用实例

此处给出更为复杂的示例说明TIM模块与多通道序列（MCS）的交互方式。

使用实例包括TIM模块，其使用两条通道来测量时间戳以及两个输入信号的沿数。

数据发送到MCS，并把MCS信息存储在内部缓冲器。一旦它收集齐所有数据，触发DMA传输以把信息存储在AURIX存储器空间的某个地方。

下图设计了使用实例。

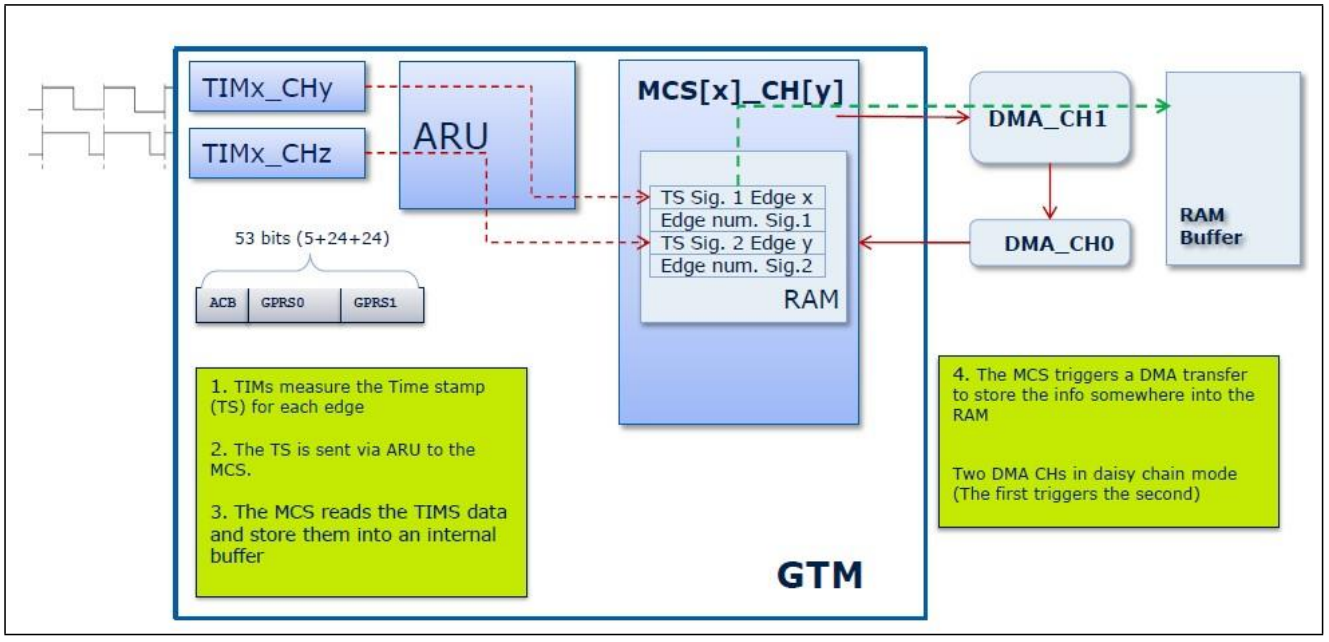

图17 TIM-ARU-MCS-DMA使用实例

下面给出了该示例中MCS汇编程序代码。

```
tsk0_init:
    movl R6 0
    movl R5 0
    movl R2 0
    movl R4 0

tsk0_main:
      movl R0 0
      movl R1 0
      ard R0 R1 TIM0_WRADDR0 ; Wait for capture (TIM0_CH0)
      mov R5 R0 ;<=== R5 <== Rising Edge TS
      mov R6 R1 ;<=== R6 <== Falling Edge TS
      ;Copy value inside TRASMIT_VALUES
      movl R3 0
      movl R3 TRANSMIT_VALUES ;internal buffer ptr
```

```
        mwri R5 R3
        addl R3 4
        mwri R6 R3
        ;Waiting TIM1_CH0 to caput
        movl R0 0
        movl R1 0
        ard R0 R1 TIM1_WRADDR0 ; Wait for capture (TIM1_CH0)
        mov R5 R0 ;<=== R5 <== Rising Edge TS
        mov R6 R1 ;<=== R6 <== Falling Edge TS
        ;Copy value inside TRASMIT_VALUES
        addl R3 4
        mwri R5 R3
        addl R3 4
        mwri R6 R3
        orl    STA IRQ_H_MSK ;MCS0_CH0 Interrupt ==> DMAs chain start
        MOVL R0 $4
          WURM R0 STRG $4 ;Stop the MCS exec. an wait for special cond.
          MOVL CTRG $4
         jmp tsk0_main
tsk0_done:
        orl    STA IRQ_H_MSK       ; rise IRQ flag
        andl STA EN_L_MSK   ; disable task
```

图18 TIM-ARU-MCS使用实例示例代码

5 PWM灵活生成方式-IfxGtm_PsmExample驱动程序

5.1 IfxGtm_PsmExample驱动程序概览

参数存储模块（PSM）是GTM基础模块的一部分。PSM子模块包括下面三个子单位：

AEI-to-FIFO数据接口(AFD)
FIFO-to-ARU接口(F2A)
FIFO

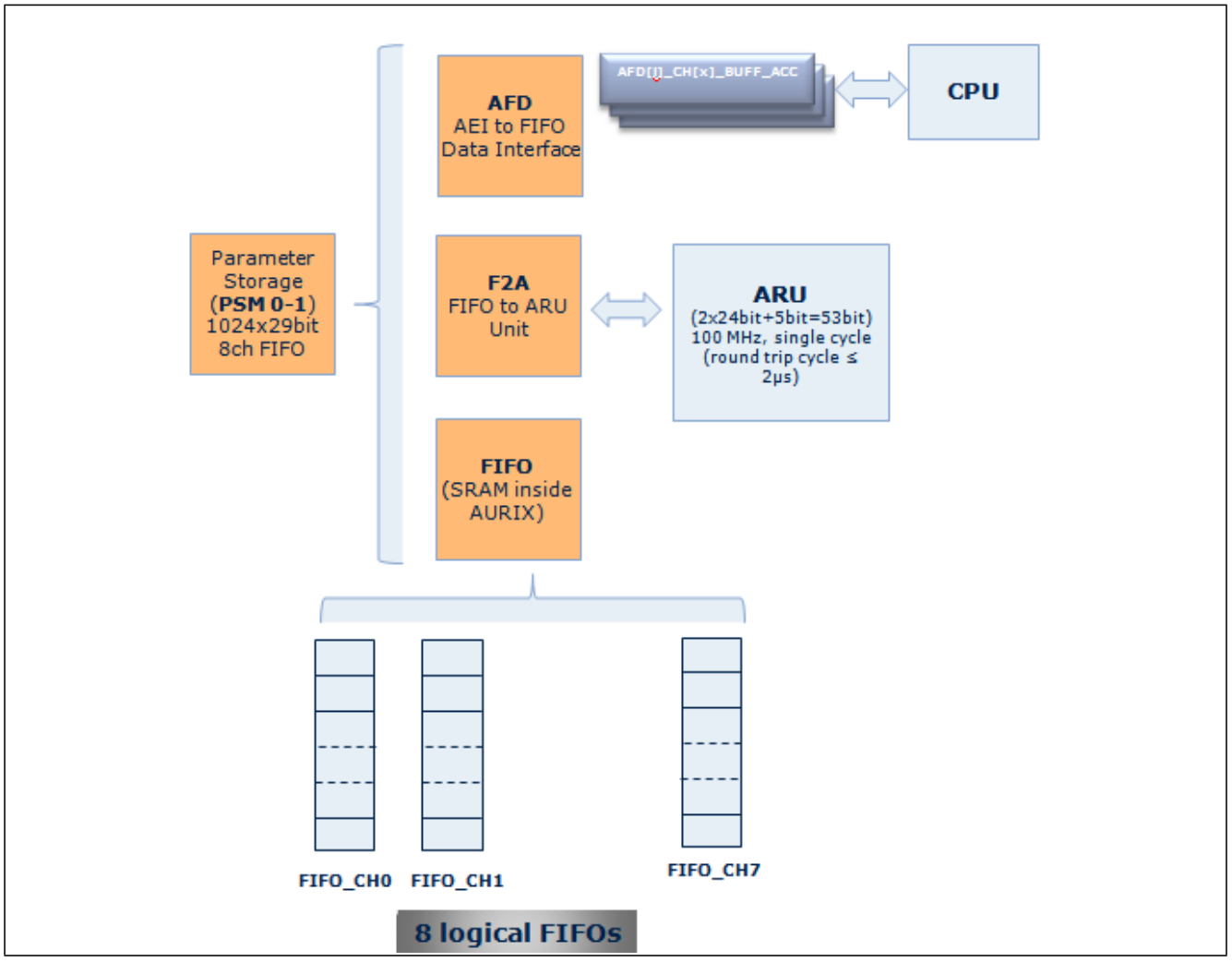

图19 PSM框图

PSM可以作为输入数据的数据存储或输出数据的参数存储用。RAM数据逻辑上置于FIFO子单位内。
此处给出了PSM如何与ATOM一起使用以生成复杂/灵活度高的PWM信号的基本示例。
很重要的是，要知道在配置之后，是不需要CPU交互来生成复杂的PWM。
这个例子中，FIFO编程在环形缓冲模式下工作。
环形缓冲模式是一种功能强大的工具，给另一GTM子模块提供一个连续的数据或配置流，而无需CPU交互。
环形缓冲模式下，FIFO给F2A子模块提供了一个连续的数据流。

首先传递FIFO的第一个字。FIFO把最后一个字提供给ARU后，会再次获得第一个字。

要求用户可以把连续数据流内的一些数据更改后给到GTM子模块或系统总线。这可能会通过FIFO AEI接口提供的直接存储器存取/访问完成。

下面所示的使用实例设计由基本驱动程序实现。

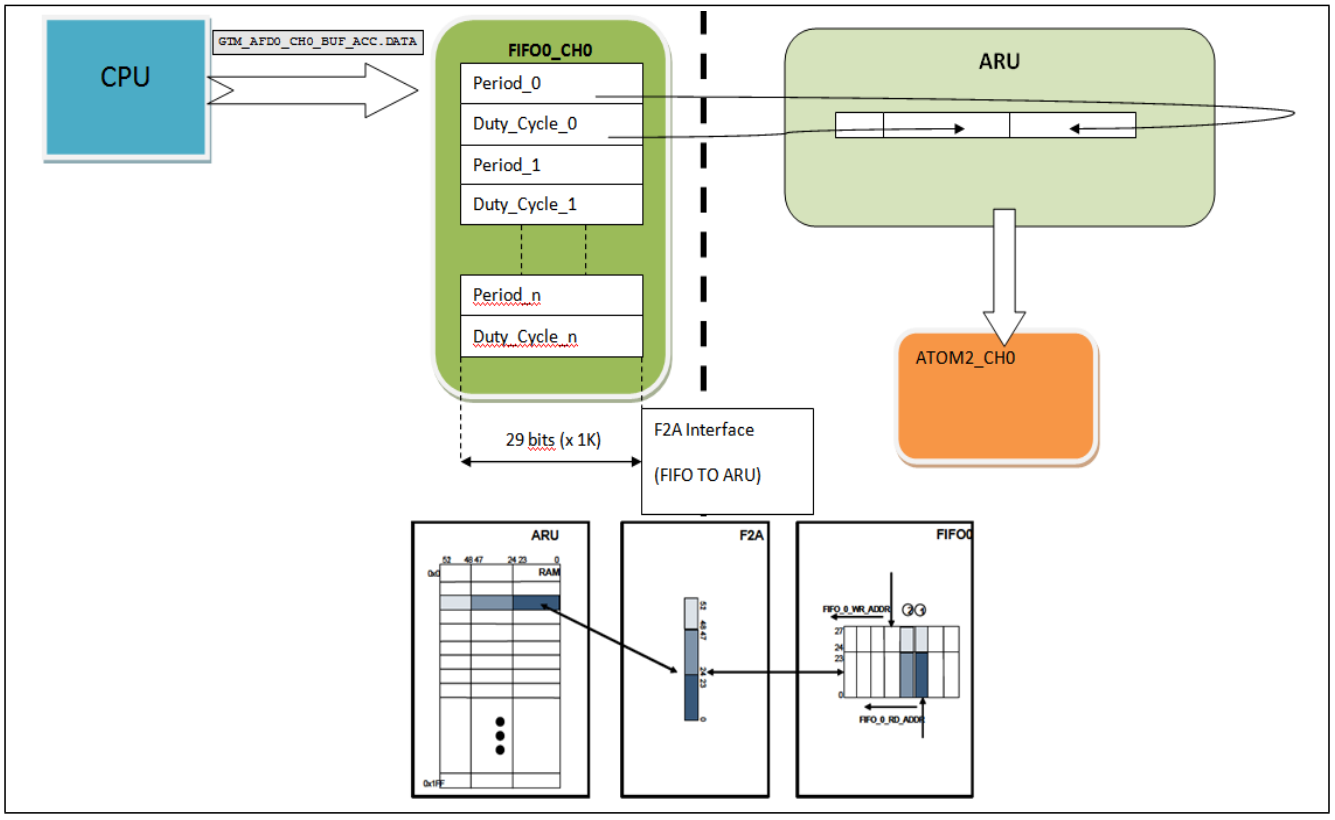

图20 PSM使用实例示例

需要CPU配置FIFO及所需ATOM模块。然后，CPU给PSM寄存器内的所需PWM写周期值和占空比值。

5.2 IfxGtm_PsmExample驱动程序API

表4 IfxGtm_PsmExample函数

函数名称	函数签名	描述
IfxGtm_PwmExampleConfig	void IfxGtm_PwmExampleConfig (sint32 atomNo, sint32 atomCh, sint32 fifoCh, IfxGtm_PeriodDutyTupla* perDutyList, sint32 fifoValNo, uint8 clockSrc, uint8 clockDiv);	驱动配置
IfxGtm_loadValues2Fifo	void IfxGtm_loadValues2Fifo (sint32 fifoCh, IfxGtm_PeriodDutyTupla* perDutyList,	PSM/FIFO更新：（周期，占空比）值的<fifoValNo> 元组复制到FIFO。
IfxGtm_pwmExampleCmuClock	void IfxGtm_pwmExampleCmuClock(uint8 CmuClkSrc, uint8 clkDiv);	时钟设置

下面所示为驱动程序的使用示例：

```
short fifoNoVal = 4;
short fifoCh = 3;
uint8 clkDiv = 0; // 1/100Mhz => 10ns

IfxGtm_PeriodDutyTupla periodDutyList[4] =            {
        { 1000, 0.90 }, //[4], period 1000 * 10ns => 10us, duty = 90%
        { 1000, 0.70 }, //[3],period 1000 * 10ns => 10us, duty = 70%
        { 1000, 0.40 }, //[2],period 1000 * 10ns => 10us, duty = 40%
        { 1000, 0.20 }, //[1],period 1000 * 10ns => 10us, duty = 20%
};

IfxGtm_PwmExampleConfig(ATOM2,CH0,fifoCh,periodDutyList,fifoNoVal,CMU_CLK
0,clkDiv);
```

图21 IfxGtm_PsmExample驱动示例配置

如下图所示，该配置在ATOM2_CH0生成PWM。

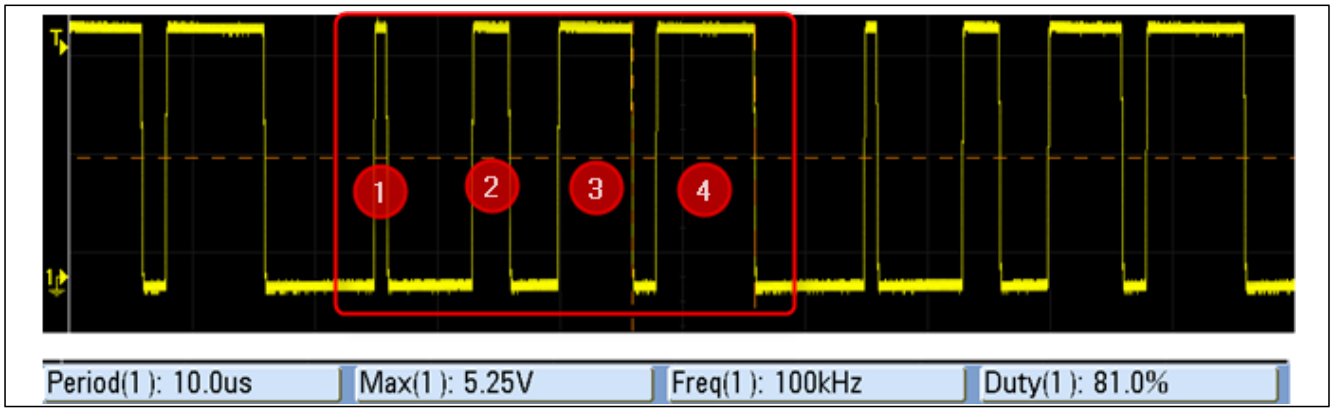

图22 PSM使用实例设计

5.3 驱动文件和文件夹结构

驱动文件结构框架如下图所示。

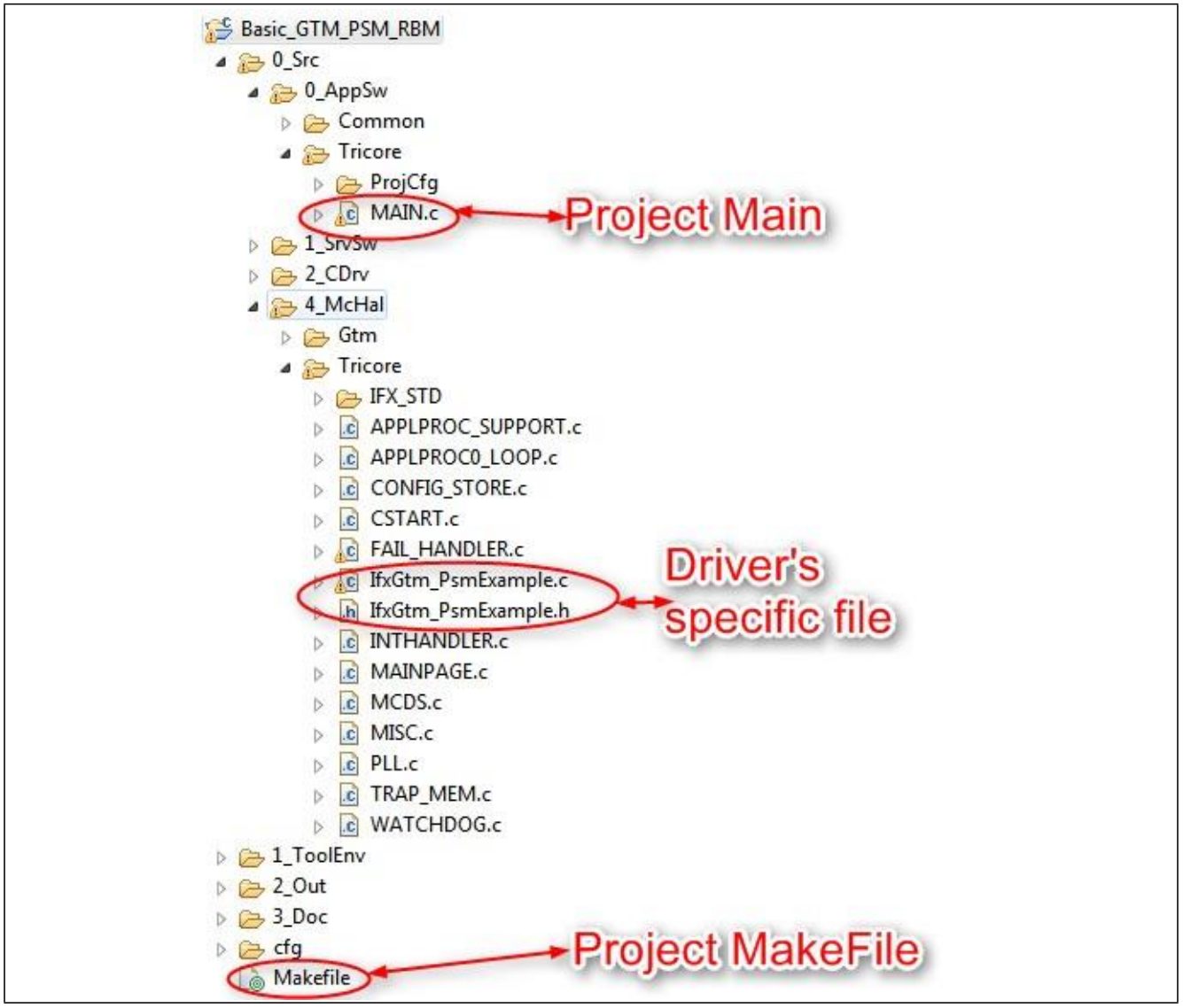

图23 IfxGtm_PsmExample文件结构

www.infineon.com

Infineon Technologies出版发行