参考文献

Application Note

MSPM0 中的高级计时器技术

Gaurang Gupta, Sal Ye, Baibhav Tripathy, Luke Ledbetter

摘要

MSPM0 器件中的高级计时器 (TIMA) 外设旨在满足现代嵌入式应用的需求。本应用手册探讨了高级计时器模块的多方面功能。它深入研究了这些模块在各种场景中的应用，范围从基于软件的位操作通信协议到复杂的反馈控制 PWM 系统。通过利用输入捕获、PWM 输出等功能以及与 ADC 和比较器等模拟外设同步，开发人员可实现响应迅速且高效的控制环路。通过详细的示例和用例分析，本手册演示了高级计时器架构如何助力开发高级嵌入式解决方案、自适应电源调节以及实时信号处理。本文提供的见解旨在帮助工程师充分发挥计时器外设的全部潜力，以实现创新的应用设计。本应用手册中详细介绍的工程配套资料可从 MSPM0 中的高级计时器技术下载。

内容

1 简介........2
2 空闲低电平状态：**PWM** 输出通道低电平状态配置........3
3 非对称 **PWM**：具有相移控制功能的双路同步 **PWM** 生成........4
3.1 使用相位加载功能........4
3.2 使用辅助捕获/比较通道........7
4 位操作 **(Bit-Banging)** 仿真：基于软件的通信协议实现........11
4.1 使用 TIMA 仿真 UART Rx........11
4.2 使用 TIMA 仿真 UART Tx........14
5 基于反馈的 **PWM** 生成........19
5.1 基于反馈的 PWM 信号复制........19
5.2 使用输入基准生成延迟的 PWM 信号........20
6 延迟计时器启动：具有可配置延迟的同步计时器实例启动........23
7 基于硬件事件停止正在运行的计时器........25
8 动态 **PWM** 更新：占空比和时间周期调整........27
8.1 影子加载和影子比较功能........27
8.2 使用 DMA 生成任意信号........29
9 总结........33
10 参考资料........34

商标

所有商标均为其各自所有者的财产。

English Document: SDAA286

Copyright © 2026 Texas Instruments Incorporated

1 简介

本应用手册探讨了 MSPM0 微控制器中高级计时器 (TIMA) 模块的多用途功能，演示了对现代嵌入式应用至关重要的高级时序和控制功能。本文档提供了全面的实现策略，涵盖各种基于计时器的解决方案，范围从电机控制到通信协议仿真。

本指南中详细介绍的主要应用包括以电机为中心的 PWM 配置（支持空闲低电平状态维护），以及复杂的 PWM 生成技术（具有同步输出及精确相位控制）。本手册演示了实现相移 PWM 的两种不同的方法：利用相位负载功能和利用辅助捕获/比较通道。对于需要自定义通信协议的应用，本文探讨了位操作实现方案，重点介绍使用 TIMA 资源的 UART 发送器和接收器仿真。

此外，本文档还介绍了各关键时序控制方面，例如多个计时器实例之间的协调延迟启动、闭环系统基于反馈的 PWM 生成以及硬件触发的计时器控制。本指南最后介绍了动态 PWM 参数修改技术，该技术能够实时调整占空比和周期值。每种实现方案都包含特定于 MSPM0 器件的实际示例、时序图和优化的配置设置。

English Document: SDAA286
Copyright © 2026 Texas Instruments Incorporated

2 空闲低电平状态：PWM 输出通道低电平状态配置

TIMA 提供了一种机制，让所有捕获/比较 (CC) 通道都具有互补通道。该互补通道可以生成与主 CC 通道输出反相的信号。对于电机应用，两条输出通道最初必须均为低电平。

当 CTRCTL.EN 位为零时，CC 通道输出取决于 OCTL.CCPIV。如果 CC 和互补通道都要保持低电平，则解决方案是使互补通道的 IOMUX 保持禁用状态，并在以下情况下启用：

- 在边沿对齐模式下，保持互补 CC 通道的 IOMUX 处于禁用状态，直到启用 CTRCTL.EN
- 在中央对齐模式下，保持互补 CC 通道的 IOMUX 处于禁用状态，直到 CTRCTL.EN 被启用并且 CC 通道满足第一个 CC 事件

```
/* Configure Complementart Channel's IOMUX in PULL_DOWN state initially */
SYSCONFIG_WEAK void SYSCFG_DL_GPIO_init(void)
{
    DL_GPIO_initPeripheralOutputFunction(GPIO_PWM_0_C0_IOMUX,GPIO_PWM_0_C0_IOMUX_FUNC);
    DL_GPIO_enableOutput(GPIO_PWM_0_C0_PORT, GPIO_PWM_0_C0_PIN);
    DL_GPIO_setDigitalInternalResistor(GPIO_PWM_0_C0_CMPL_IOMUX,DL_GPIO_RESISTOR_PULL_DOWN);
}

/* Enable Counter with CTRCL.EN bit and configure the IOMUX back to PWM mode */
int main(void)
{
    SYSCFG_DL_init();
    DL_TimerA_startCounter(PWM_0_INST);
    DL_GPIO_initPeripheralOutputFunction(GPIO_PWM_0_C0_CMPL_IOMUX,GPIO_PWM_0_C0_CMPL_IOMUX_FUNC);
    while (1) {
        __WFI();
    }
}
```

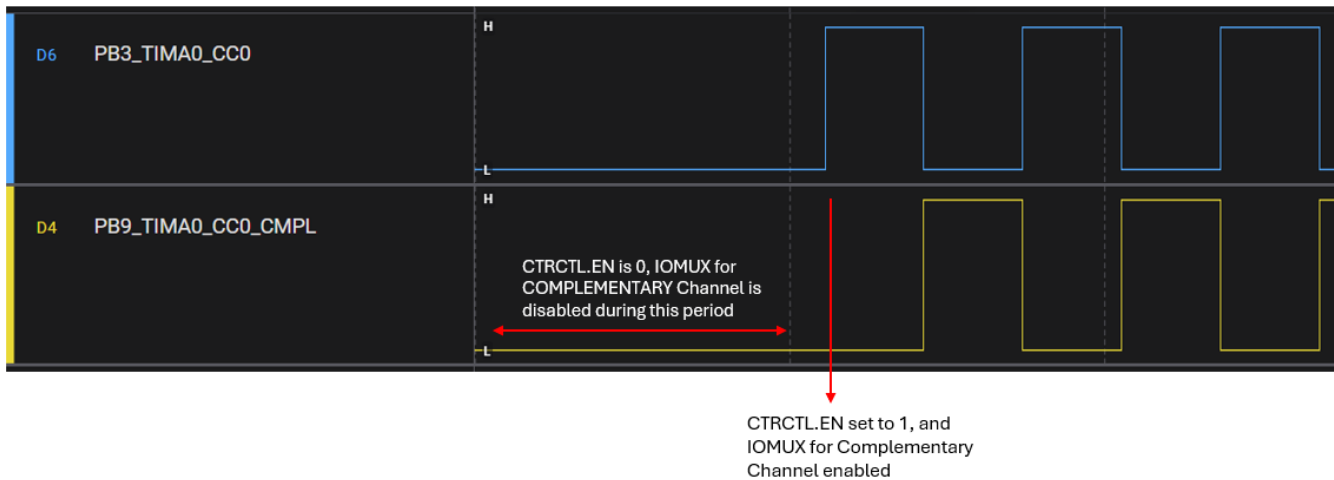

图 2-1. CC 输出设置为空闲低电平时的 PWM 波形

此外，TIMA 还支持一项功能，其中基于软件的配置可用于覆盖输出。CCACT 中的 SWFRCACT_CMPL（互补通道上的软件强制）字段可用于将 CC 输出覆盖为低电平。根据应用，可以使用外部事件或在中断服务例程中清除 SWFRCACT_CMPL 字段。

备注

SWFRCACT_CMPL 在设置 CTRCTL.EN 后生效。

Copyright © 2026 Texas Instruments Incorporated

3 非对称 PWM：具有相移控制功能的双路同步 PWM 生成

TIMA 可生成两个具有受控相移的同步中心对齐 PWM 信号，用于

- 电机控制，以更大限度地减少扭矩纹波并提高低速运行的平滑度
- 数字电源/直流/直流转换器，以降低开关损耗和 EMI
- 交流相位控制电路，以实现基于相位角的精确功率控制

3.1 使用相位加载功能

要生成非对称 PWM 信号，可以使用 TIMA 中的相位加载特性。此方法使用两个计时器实例，其中一个必须是 TIMA（仅 TIMA 支持相位加载功能）。两个 TIMER 实例均使用交叉触发功能进行同步。当计时器交叉触发时，如果启用了相位加载功能，则相位加载生效，计时器从相位加载值开始计数。这一可配置的相位加载值支持两个 PWM 之间的受控相移。以下各节介绍配置。

3.1.1 主要计时器（主计时器）的配置

主要计时器是生成交叉触发信号以同步其他计时器的计时器：

- 通过设置 TIMx.CTTRIGCTL.CTEN 位来启用交叉触发输出功能。
- 通过选择适当的触发源来配置交叉触发生成事件。
  - 对于基于软件的交叉触发，设置 TIMx.CTTRIG.TRIG 位。
  - 对于基于硬件的交叉触发，使用 TIMx.CTTRIGCTL.EVTCTTRIGSEL 选择触发源，并通过设置 TIMx.CTTRIGCTL.EVTCTEN 启用硬件触发。
- 计时器还提供自触发功能

```
/* Configuration for Main Timer to generate Cross Trigger */
static const DL_TimerA_ClockConfig gPWM_0ClockConfig = {
    .clockSel = DL_TIMER_CLOCK_BUSCLK,
    .divideRatio = DL_TIMER_CLOCK_DIVIDE_1,
    .prescale = 0U
};

static const DL_TimerA_PWMConfig gPWM_0Config = {
    .pwmMode = DL_TIMER_PWM_MODE_CENTER_ALIGN,
    .period = 1600,
    .isTimerWithFourCC = false,
    .startTimer = DL_TIMER_STOP,
};

SYSCONFIG_WEAK void SYSCFG_DL_PWM_0_init(void) {

    DL_TimerA_setClockConfig(
        PWM_0_INST, (DL_TimerA_ClockConfig *) &gPWM_0ClockConfig);

    DL_TimerA_initPWMMode(
        PWM_0_INST, (DL_TimerA_PWMConfig *) &gPWM_0Config);

    // Set Counter control to the smallest CC index being used

DL_TimerA_setCounterControl(PWM_0_INST,DL_TIMER_CZC_CCCTL0_ZCOND,DL_TIMER_CAC_CCCTL0_ACOND,DL_TIMER_
CLC_CCCTL0_LCOND);

    DL_TimerA_setCaptureCompareOutCtl(PWM_0_INST, DL_TIMER_CC_OCTL_INIT_VAL_LOW,
        DL_TIMER_CC_OCTL_INV_OUT_DISABLED, DL_TIMER_CC_OCTL_SRC_FUNCVAL,
        DL_TIMERA_CAPTURE_COMPARE_0_INDEX);

    DL_TimerA_setCaptCompUpdateMethod(PWM_0_INST, DL_TIMER_CC_UPDATE_METHOD_IMMEDIATE,
DL_TIMERA_CAPTURE_COMPARE_0_INDEX);
    DL_TimerA_setCaptureCompareValue(PWM_0_INST, 500, DL_TIMER_CC_0_INDEX);

    DL_TimerA_setCaptureCompareOutCtl(PWM_0_INST, DL_TIMER_CC_OCTL_INIT_VAL_LOW,
        DL_TIMER_CC_OCTL_INV_OUT_DISABLED, DL_TIMER_CC_OCTL_SRC_FUNCVAL,
        DL_TIMERA_CAPTURE_COMPARE_1_INDEX);

    DL_TimerA_setCaptCompUpdateMethod(PWM_0_INST, DL_TIMER_CC_UPDATE_METHOD_IMMEDIATE,
DL_TIMERA_CAPTURE_COMPARE_1_INDEX);
    DL_TimerA_setCaptureCompareValue(PWM_0_INST, 500, DL_TIMER_CC_1_INDEX);

    DL_TimerA_enableClock(PWM_0_INST);
```


English Document: SDAA286
Copyright © 2026 Texas Instruments Incorporated

```
    DL_TimerA_setCCPDirection(PWM_0_INST , DL_TIMER_CC0_OUTPUT | DL_TIMER_CC1_OUTPUT );

    DL_TimerA_configCrossTrigger(PWM_0_INST, DL_TIMER_CROSS_TRIG_SRC_ZERO,
                             DL_TIMER_CROSS_TRIGGER_INPUT_ENABLED,
DL_TIMER_CROSS_TRIGGER_MODE_ENABLED
        );//Configuration to Generate Hardware Based Cross Trigger on Zero Event

    DL_TimerA_setCaptureCompareInput(PWM_0_INST, DL_TIMER_CC_INPUT_INV_NOINVERT,
DL_TIMER_CC_IN_SEL_TRIG, DL_TIMER_CC_0_INDEX);

    DL_TimerA_setCaptureCompareInput(PWM_0_INST, DL_TIMER_CC_INPUT_INV_NOINVERT,
DL_TIMER_CC_IN_SEL_TRIG, DL_TIMER_CC_1_INDEX);

    /*
     * Determines the external triggering event to trigger the module (self-triggered in main
configuration)
     * and triggered by specific timer in secondary configuration
     */
    DL_TimerA_setExternalTriggerEvent(PWM_0_INST,DL_TIMER_EXT_TRIG_SEL_TRIG_1);
    DL_TimerA_enableExternalTrigger(PWM_0_INST);
    uint32_t temp;
    temp = DL_TimerA_getCaptureCompareCtl(PWM_0_INST, DL_TIMER_CC_0_INDEX);
    DL_TimerA_setCaptureCompareCtl(PWM_0_INST, DL_TIMER_CC_MODE_COMPARE, temp | (uint32_t)
DL_TIMER_CC_ZCOND_TRIG_RISE, DL_TIMER_CC_0_INDEX);

    temp = DL_TimerA_getCaptureCompareCtl(PWM_0_INST, DL_TIMER_CC_1_INDEX);
    DL_TimerA_setCaptureCompareCtl(PWM_0_INST, DL_TIMER_CC_MODE_COMPARE, temp | (uint32_t)
DL_TIMER_CC_ZCOND_TRIG_RISE, DL_TIMER_CC_1_INDEX);

}
```

3.1.2 辅助计时器的配置

辅助计时器接收交叉触发信号并将其运行与主要计时器同步。

- 根据交叉触发映射（请参阅特定于器件的数据表）以及使用的主要和辅助计时器实例配置 TIMx.TSEL.ETSEL 字段。
- 通过将 TIMA.TSEL.TE 位设置为 1 来启用输入触发器功能。
- 将 TIMAx.IFCTL_01.ISEL 位设置为 3 以选择交叉触发作为输入源。
- 如果计数器以递减计数模式计数，则将 CCCTL.LCOND 设置为 1 以使用上升沿触发加载事件，这将在收到交叉触发后将计数器值设置为加载值，并且计数器将开始递减计数。类似地，递增计数模式将 CCCTL.ZCOND 设置为 1
- 如果是自交叉触发场景和递增/递减计数模式，例如 TIMx.CTTRIGCTL.EVTCTTRIGSEL 设置为零事件，则 CCCTL.ZCOND 应设置为 1
- 在满足 LCOND 或 ZCOND 条件时会设置 TIMx.CTRCTL.EN 位，并且计数器值分别更改为加载值或零值。

备注

在交叉触发模式下工作的计时器应具有相同的 TIMCLK 频率。如果生成的交叉触发脉冲宽度小于接收触发信号的计时器的一个 TIMCLK 周期，则交叉触发将不会启动计数器。

```
/* Configuration for Secondary Timer to Receive a Cross Trigger */
static const DL_TimerA_ClockConfig gPWM_1ClockConfig = {
    .clockSel = DL_TIMER_CLOCK_BUSCLK,
    .divideRatio = DL_TIMER_CLOCK_DIVIDE_1,
    .prescale = 0U
};

static const DL_TimerA_PWMConfig gPWM_1Config = {
    .pwmMode = DL_TIMER_PWM_MODE_CENTER_ALIGN,
    .period = 1600,
    .isTimerWithFourCC = true,
    .startTimer = DL_TIMER_STOP,
};

SYSCONFIG_WEAK void SYSCFG_DL_PWM_1_init(void) {

    DL_TimerA_setClockConfig(
```


Copyright © 2026 Texas Instruments Incorporated

```
        PWM_1_INST, (DL_TimerA_ClockConfig *) &gPWM_1ClockConfig);

    DL_TimerA_initPWMMode(
        PWM_1_INST, (DL_TimerA_PWMConfig *) &gPWM_1Config);

    // Set Counter control to the smallest CC index being used

DL_TimerA_setCounterControl(PWM_1_INST,DL_TIMER_CZC_CCCTL0_ZCOND,DL_TIMER_CAC_CCCTL0_ACOND,DL_TIMER_
CLC_CCCTL0_LCOND);

    DL_TimerA_setCaptureCompareOutCtl(PWM_1_INST, DL_TIMER_CC_OCTL_INIT_VAL_LOW,
        DL_TIMER_CC_OCTL_INV_OUT_DISABLED, DL_TIMER_CC_OCTL_SRC_FUNCVAL,
        DL_TIMERA_CAPTURE_COMPARE_0_INDEX);

    DL_TimerA_setCaptCompUpdateMethod(PWM_1_INST, DL_TIMER_CC_UPDATE_METHOD_IMMEDIATE,
DL_TIMERA_CAPTURE_COMPARE_0_INDEX);
    DL_TimerA_setCaptureCompareValue(PWM_1_INST, 500, DL_TIMER_CC_0_INDEX);

    DL_TimerA_setCaptureCompareOutCtl(PWM_1_INST, DL_TIMER_CC_OCTL_INIT_VAL_LOW,
        DL_TIMER_CC_OCTL_INV_OUT_DISABLED, DL_TIMER_CC_OCTL_SRC_FUNCVAL,
        DL_TIMERA_CAPTURE_COMPARE_1_INDEX);

    DL_TimerA_setCaptCompUpdateMethod(PWM_1_INST, DL_TIMER_CC_UPDATE_METHOD_IMMEDIATE,
DL_TIMERA_CAPTURE_COMPARE_1_INDEX);
    DL_TimerA_setCaptureCompareValue(PWM_1_INST, 500, DL_TIMER_CC_1_INDEX);

    DL_Timer_enablePhaseLoad(PWM_1_INST);

    DL_TimerA_enableClock(PWM_1_INST);

    DL_Timer_setPhaseLoadValue(PWM_1_INST, 200);

    DL_TimerA_setCCPDirection(PWM_1_INST , DL_TIMER_CC0_OUTPUT | DL_TIMER_CC1_OUTPUT );

    DL_TimerA_setCaptureCompareInput(PWM_1_INST, DL_TIMER_CC_INPUT_INV_NOINVERT,
DL_TIMER_CC_IN_SEL_TRIG, DL_TIMER_CC_0_INDEX);

    DL_TimerA_setCaptureCompareInput(PWM_1_INST, DL_TIMER_CC_INPUT_INV_NOINVERT,
DL_TIMER_CC_IN_SEL_TRIG, DL_TIMER_CC_1_INDEX);

    /*
     * Determines the external triggering event to trigger the module (self-triggered in main
configuration)
     * and triggered by specific timer in secondary configuration
     */
    DL_TimerA_setExternalTriggerEvent(PWM_1_INST,DL_TIMER_EXT_TRIG_SEL_TRIG_1);
    DL_TimerA_enableExternalTrigger(PWM_1_INST);
    uint32_t temp;
    temp = DL_TimerA_getCaptureCompareCtl(PWM_1_INST, DL_TIMER_CC_0_INDEX);
    DL_TimerA_setCaptureCompareCtl(PWM_1_INST, DL_TIMER_CC_MODE_COMPARE, temp | (uint32_t)
DL_TIMER_CC_ZCOND_TRIG_RISE, DL_TIMER_CC_0_INDEX);

    temp = DL_TimerA_getCaptureCompareCtl(PWM_1_INST, DL_TIMER_CC_1_INDEX);
    DL_TimerA_setCaptureCompareCtl(PWM_1_INST, DL_TIMER_CC_MODE_COMPARE, temp | (uint32_t)
DL_TIMER_CC_ZCOND_TRIG_RISE, DL_TIMER_CC_1_INDEX);
}
```

3.1.3 交叉触发功能的实现

表 3-1 介绍了两种交叉触发生成方法。

表 3-1. 交叉触发

序号	软件交叉触发	硬件交叉触发
1	当应用程序代码向 TIMx.CTTRIG.TRIG 位写入 1 时，本质上是生成软件交叉触发。	根据 TIMx.CTTRIGCTL.EVTCTTRIGSEL 字段中配置的事件，硬件会在每个配置的事件发生时生成硬件交叉触发，而无需任何软件参与（在初始配置完成后）。例如，如果 EVTCTTRIGSEL 被设置为零事件，则会在主要计时器的每个零事件时生成硬件交叉触发。

English Document: SDAA286
Copyright © 2026 Texas Instruments Incorporated

表 3-1. 交叉触发（续）

序号	软件交叉触发	硬件交叉触发
2	仅当应用需要时，软件交叉触发才会同步主要计时器与辅助计时器。 **备注** 如果加载/CC 值发生变化，由软件产生的同步可能会失效。	硬件交叉触发机制会反复同步主要计时器与辅助计时器。因此，即使主要计时器的加载/CC 值发生变化，辅助计时器仍会与主要计时器同步，因为每次事件发生时都会生成硬件交叉触发。

基于软件的交叉触发信号的机制

```
DL_TimerA_generateCrossTrigger(PWM_0_INST);//Mechanism to generate Software based Cross-Trigger,
this software cross trigger will enable TIMA0 and TIMA1
```

在发生零事件时生成基于硬件的交叉触发的配置

```
    DL_TimerA_configCrossTrigger(PWM_0_INST, DL_TIMER_CROSS_TRIG_SRC_ZERO,
    DL_TIMER_CROSS_TRIGGER_INPUT_ENABLED, DL_TIMER_CROSS_TRIGGER_MODE_ENABLED
        );  // Configuration to Generate Hardware based Cross-Trigger on Zero Event
```

相移 **PWM** 生成

图 3-1 展示了相移 PWM 生成的测试波形。

图 **3-1.** 使用交叉触发和相位加载生成相移 **PWM**

3.2 使用辅助捕获/比较通道

TIMA0 具有 4 条外部通道和 2 条内部 CC 通道，可独立用于生成 3 个相移 PWM。这可通过利用内部 CC 通道的 SECONDARY CC 事件来实现。GPTIMER 提供通过配置 CCCTL.CC2SELD 或 CCCTL.CC2SELU 来选择辅助 CC 通道的功能，具体取决于计数模式（递减、递增、递增/递减）。

English Document: SDAA286
Copyright © 2026 Texas Instruments Incorporated

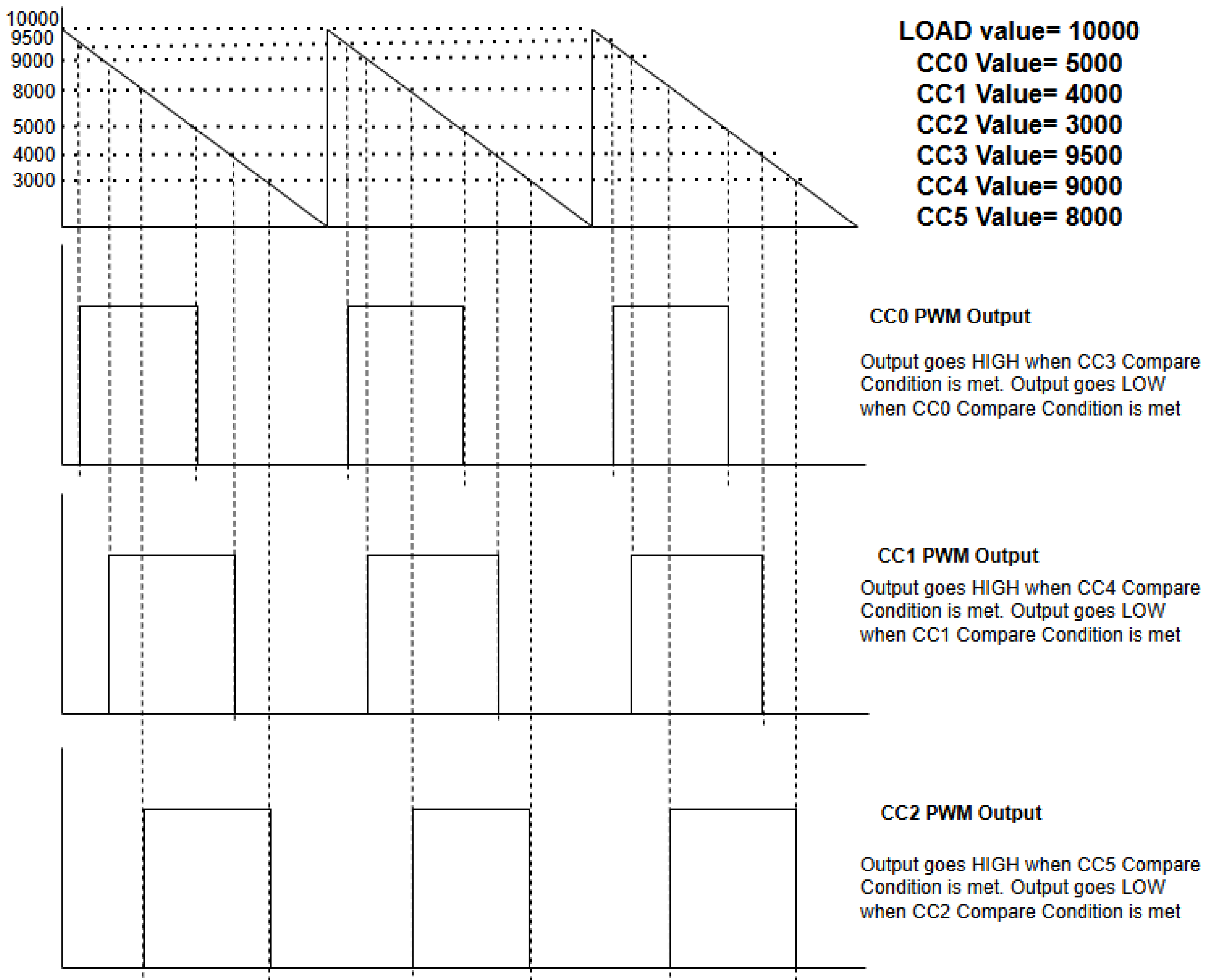

图 3-2. 以辅助通道方法使用 TIMA0 生成非对称 PWM 的示意图

根据所选的辅助通道，可以通过配置 CCACT.CC2UACT 或 CCACT.CC2DACT 来完成输出生成，具体取决于计数模式（递增、递减、递增/递减）。

例如，如果计数器加载值为 10000，并且计数器处于递减计数模式，且 CC4 值为 9000。

将 CCCTL[1].CC2SELD 配置为 0x4，这意味着选择 CC4 作为主通道 CC1 的辅助 CC 通道。

将 CCACT[1].CC2DACT 配置为 0x1，这意味着只要计数器的值达到 9000（即当前 CC4 值），CC 输出值就会设置为高电平。CC4 将用作 CC1 的**伪**加载值。

可以观察到，由于 CCACT[1].CC2DACT，每次计数器达到 9000 时，CC1 的输出都会变为高电平。当计数器达到 4000（CC1 值）时，PWM 输出将变为低电平。

实施此方法来生成三个相移 PWM 非常有用。使用交叉触发功能可通过多个计时器实例生成更多同步相移 PWM 输出通道。

按照以下**配置序列**设置辅助 CC 通道：

- 在 TIMA.CTRCTL 中，为以下各项设置所需的计数器控制设置：
  - 计数模式 (CM) 和启用后计数值 (CVAE)。
  - CLC、CZC 和 CAC，用于指定控制计数器清零、前进或加载的条件
- 设置 TIMA.LOAD 值以配置 PWM 周期

Copyright © 2026 Texas Instruments Incorporated

- 对于比较模式，设置 TIMA.CCCTL_xy[0/1].COC=0
- 通过设置 CCPD 寄存器中的相应位，将 CCP 配置为 CC 块的输出。
- 在 TIMA.OCTL_xy[0/1] 中，设置 CCPO = 0 以选择信号发生器输出。
- 通过针对相应计数器 n 将 ODIS.C0CCPn 设置为 0 来启用相应的 CCP 输出。
- 使用 CCPOINV 位配置信号的极性，并配置 CCPIV 以指定计数器禁用时的 CCP 输出状态。
- 将 CC0 的 CCCTL.CC2SELD 字段配置为 3，表示 CC3 用作辅助 CC 块。
- 将 CC1 的 CCCTL.CC2SELD 字段配置为 4，表示 CC4 用作辅助 CC 块。
- 将 CC2 的 CCCTL.CC2SELD 字段配置为 5，表示 CC5 用作辅助 CC 块。
- 根据所需的相移配置 CC3。例如，如果需要 500 个 TIMCLK 周期的相移，则配置 CC3= 加载值 – 500 = 9500。
- 根据所需的相移配置 CC4。例如，如果需要 1000 个 TIMCLK 周期的相移，则配置 CC4= 加载值 – 1000 = 9000。
- 根据所需的相移配置 CC5。例如，如果需要 2000 个 TIMCLK 周期的相移，则配置 CC5= 加载值 – 2000 = 8000。
- 根据所需的占空比配置 CC0、CC1、CC2，分别以 CC3、CC4 和 CC5 为基准。
- CC3、CC4 和 CC5 值将分别用作 CC0、CC1 和 CC2 的伪加载值。
- 将 CC0、CC1 和 CC2 的 CCACT.CC2DACT 配置为 0x1，以在计数器分别达到 CC3、CC4 和 CC5 值时将输出设置为高电平。
- 将 CC0、CC1 和 CC2 的 CCACT.CDACT 配置为 0x2，以在计数器分别达到 CC0、CC1 和 CC2 值时将输出置为低电平。
- 通过设置 TIMA.CTRCTL.EN = 1 来启用计数器。

```
/* TIMA0 Configuration for Generating Phase Shifted PWMs Using Secondary CC events */
static const DL_TimerA_ClockConfig gPWM_0ClockConfig = {
    .clockSel = DL_TIMER_CLOCK_BUSCLK,
    .divideRatio = DL_TIMER_CLOCK_DIVIDE_1,
    .prescale = 255U
};

static const DL_TimerA_PWMConfig gPWM_0Config = {
    .pwmMode = DL_TIMER_PWM_MODE_EDGE_ALIGN,
    .period = 10000,
    .isTimerWithFourCC = true,
    .startTimer = DL_TIMER_STOP,
};

SYSCONFIG_WEAK void SYSCFG_DL_PWM_0_init(void) {

    DL_TimerA_setClockConfig(
        PWM_0_INST, (DL_TimerA_ClockConfig *) &gPWM_0ClockConfig);

    DL_TimerA_initPWMMode(
        PWM_0_INST, (DL_TimerA_PWMConfig *) &gPWM_0Config);

    // Set Counter control to the smallest CC index being used

DL_TimerA_setCounterControl(PWM_0_INST,DL_TIMER_CZC_CCCTL0_ZCOND,DL_TIMER_CAC_CCCTL0_ACOND,DL_TIMER_
CLC_CCCTL0_LCOND);

    DL_TimerA_setCaptureCompareOutCtl(PWM_0_INST, DL_TIMER_CC_OCTL_INIT_VAL_LOW,
        DL_TIMER_CC_OCTL_INV_OUT_DISABLED, DL_TIMER_CC_OCTL_SRC_FUNCVAL,
        DL_TIMERA_CAPTURE_COMPARE_0_INDEX);

    DL_TimerA_setCaptCompUpdateMethod(PWM_0_INST, DL_TIMER_CC_UPDATE_METHOD_IMMEDIATE,
DL_TIMERA_CAPTURE_COMPARE_0_INDEX);

    DL_TimerA_setCaptureCompareOutCtl(PWM_0_INST, DL_TIMER_CC_OCTL_INIT_VAL_LOW,
        DL_TIMER_CC_OCTL_INV_OUT_DISABLED, DL_TIMER_CC_OCTL_SRC_FUNCVAL,
        DL_TIMERA_CAPTURE_COMPARE_1_INDEX);

    DL_TimerA_setCaptCompUpdateMethod(PWM_0_INST, DL_TIMER_CC_UPDATE_METHOD_IMMEDIATE,
DL_TIMERA_CAPTURE_COMPARE_1_INDEX);
    PWM_0_INST->COUNTERREGS.CCCTL_01[0]=0x60000000;//Configuring CC3 as the secondary event for CC0
    PWM_0_INST->COUNTERREGS.CCCTL_01[1]=0x80000000;//Configuring CC4 as the secondary event for CC1
    PWM_0_INST->COUNTERREGS.CCCTL_23[0]=0xA0000000;//Configuring CC5 as the secondary event for CC2

    PWM_0_INST->COUNTERREGS.CCACT_01[0]=0x00001080;//Configuring CC2DACT to drive PWM High and
CDACT to drive PWM Low
```


Copyright © 2026 Texas Instruments Incorporated

```
    PWM_0_INST->COUNTERREGS.CCACT_01[1]=0x00001080;//Configuring CC2DACT to drive PWM High and
CDACT to drive PWM Low
    PWM_0_INST->COUNTERREGS.CCACT_23[0]=0x00001080;//Configuring CC2DACT to drive PWM High and
CDACT to drive PWM Low

    DL_TimerA_setCaptureCompareValue(PWM_0_INST, 5000, DL_TIMER_CC_0_INDEX);
    DL_TimerA_setCaptureCompareValue(PWM_0_INST, 4000, DL_TIMER_CC_1_INDEX);
    DL_TimerA_setCaptureCompareValue(PWM_0_INST, 3000, DL_TIMER_CC_2_INDEX);
    DL_TimerA_setCaptureCompareValue(PWM_0_INST, 9500, DL_TIMER_CC_3_INDEX);
    DL_TimerA_setCaptureCompareValue(PWM_0_INST, 9000, DL_TIMER_CC_4_INDEX);
    DL_TimerA_setCaptureCompareValue(PWM_0_INST, 8000, DL_TIMER_CC_5_INDEX);

    DL_TimerA_enableClock(PWM_0_INST);
    DL_TimerA_setCCPDirection(PWM_0_INST , DL_TIMER_CC0_OUTPUT | DL_TIMER_CC1_OUTPUT |
DL_TIMER_CC2_OUTPUT);
}
```

图 3-3 显示了移位的 PWM 输出波形。要动态更改周期/占空比，请参阅节 8 。

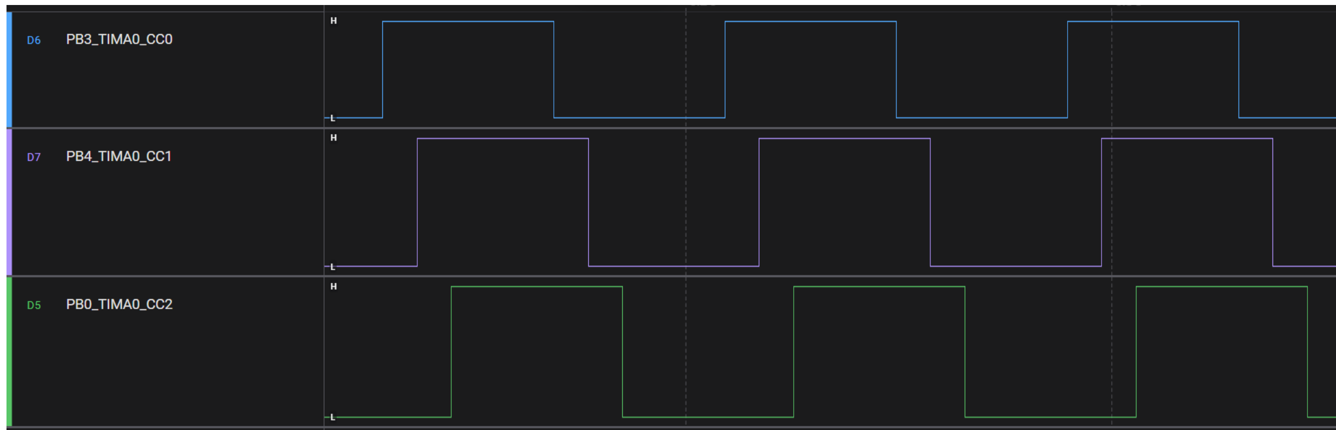

图 3-3. 使用 TIMA0 辅助事件生成相移 PWM

Copyright © 2026 Texas Instruments Incorporated

4 位操作 (Bit-Banging) 仿真：基于软件的通信协议实现

TIMA 可用于对通用异步接收器/发送器 (UART) 等标准通信协议进行仿真。TIMA 还允许实施需要高安全加密的非标准或自定义协议。借助 TIMA，开发人员可以精确调度时钟或数据线路的输出切换，捕获读取操作的输入边沿，并比较事件以生成一致的波特率或时钟周期。

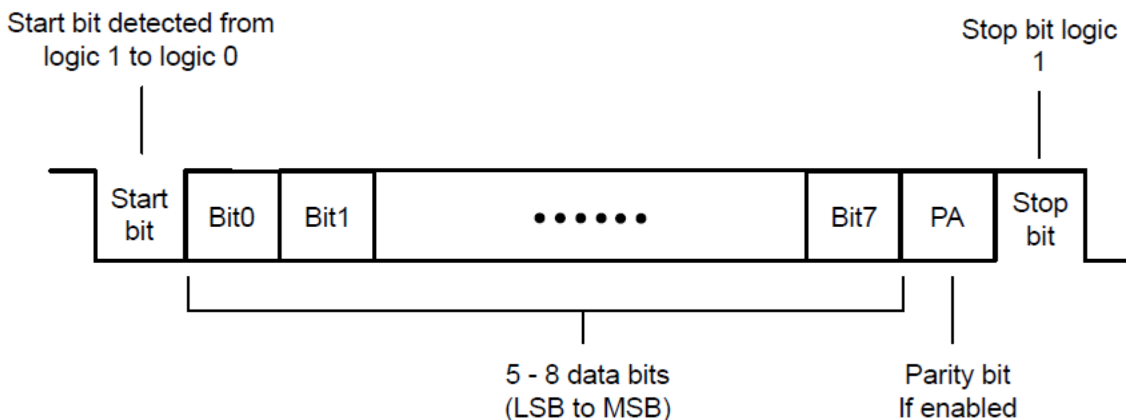

图 4-1. UART 协议标准帧

4.1 使用 TIMA 仿真 UART Rx

TIMA 提供了进行配置以仿真 UART 协议所需的所有功能。图 4-2 展示了使用 TIMA 作为 UART Rx 的流程图。

Copyright © 2026 Texas Instruments Incorporated

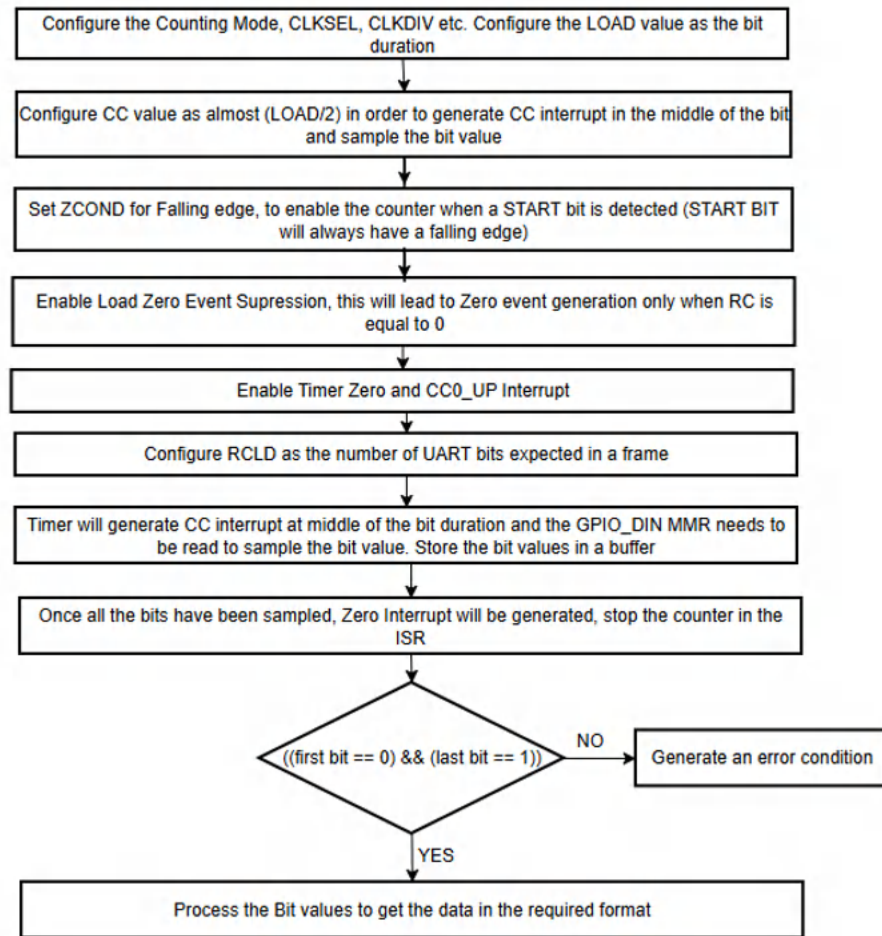

图 4-2. 展示 UART Rx 的流程图

配置序列如下所述：

- 在递增计数模式下配置 TIMA，并将 CCCTL.ZCOND 配置为下降沿检测，这是因为在 UART 中，帧的起始通过下降沿指示。
- 根据波特率，将 TIMER 的加载值配置为位时间。
- 通过配置 RCLD 寄存器，将计时器配置为在重复模式下运行 n 次（其中 n= UART 帧中预期的位数）。将 CC 值配置为（LOAD 值/2 或 LOAD 值/2 +1），以在位中间生成 CC 事件并对相应的位进行采样。
- 启用加载和零事件抑制，这样在 RC 为零之前不会生成零事件。除非所有位均已完成采样，否则不需要零事件。
- 在计时器的中断服务例程中，如果发生 CC 事件，则读取 GPIO_DIN 寄存器，这将提供信号的输入值
- 在计时器的中断服务例程中，如果发生零事件，则停止计数器。

```
/* Timer Configuration for Emulating UART Rx */
//9600 UART baud rates equates to bit duration of 104.167us, hence configuring LOAD Value as 3333
with 32MHz Clock
static const DL_TimerA_ClockConfig gCAPTURE_0ClockConfig = {
    .clockSel     = DL_TIMER_CLOCK_BUSCLK,
    .divideRatio = DL_TIMER_CLOCK_DIVIDE_1,
    .prescale     = 0U
};

static const DL_TimerA_TimerConfig gTimer_Config = {
    .timerMode       = DL_TIMER_TIMER_MODE_PERIODIC_UP,
    .period          = 3332,
    .startTimer      = DL_TIMER_STOP,
    .genIntermInt    = DL_TIMER_INTERM_INT_ENABLED,
    .counterVal      = 1666,
```


English Document: SDAA286
Copyright © 2026 Texas Instruments Incorporated

```
};

SYSCONFIG_WEAK void SYSCFG_DL_CAPTURE_0_init(void) {

    DL_TimerA_setClockConfig(CAPTURE_0_INST,
        (DL_TimerA_ClockConfig *) &gCAPTURE_0ClockConfig);
    DL_TimerA_initTimerMode(CAPTURE_0_INST,
        (DL_TimerA_TimerConfig *) &gTimer_Config);

DL_TimerA_setCounterControl(CAPTURE_0_INST,DL_TIMER_CZC_CCCTL0_ZCOND,DL_TIMER_CAC_CCCTL0_ACOND,DL_TI
MER_CLC_CCCTL0_LCOND);
    DL_TimerA_setCaptCompUpdateMethod(CAPTURE_0_INST, DL_TIMER_CC_UPDATE_METHOD_IMMEDIATE,
DL_TIMERA_CAPTURE_COMPARE_0_INDEX);

    DL_TimerA_setCaptureCompareValue(CAPTURE_0_INST, 1666, DL_TIMER_CC_0_INDEX);//Configuring CC1
as almost half the bit width, to sample the bit in the middle
    uint32_t temp;
    temp = DL_TimerA_getCaptureCompareCtl(CAPTURE_0_INST, DL_TIMER_CC_0_INDEX);
    DL_TimerA_setCaptureCompareCtl(CAPTURE_0_INST, DL_TIMER_CC_MODE_COMPARE, temp | (uint32_t)
DL_TIMER_CC_ZCOND_TRIG_FALL, DL_TIMER_CC_0_INDEX);//Enable Counter on a falling edge, this will
detect the start bit

    DL_TimerA_enableInterrupt(CAPTURE_0_INST , DL_TIMERA_INTERRUPT_CC0_UP_EVENT |
        DL_TIMERA_INTERRUPT_ZERO_EVENT);
    DL_Timer_enableLZEventSuppression(CAPTURE_0_INST);//This will suppress zero events until RC is
equal to 0
    DL_TimerA_enableClock(CAPTURE_0_INST);
}
```

```
/* Application Code for Emulating UART Rx Using Timer */
#define GPIO_CAPTURE_0_C0_PIN_BIT 3
volatile uint32_t gCaptureCnt;
volatile bool gSynced;
volatile bool gCheckCaptures;
uint32_t gLoadValue;
uint32_t uart_data_frame_received[10];
uint32_t number_of_zero_interrupts=0;
uint32_t number_of_CC_interrupts=0;
uint32_t data_bit_value;
uint8_t number_of_bits_in_UART_frame=0;
bool uart_frame_received=0;
uint8_t hex_value=0;
uint8_t uart_data_received=0;
bool uart_frame_erroneous=0;
int main(void)
{
    SYSCFG_DL_init();
    NVIC_EnableIRQ(CAPTURE_0_INST_INT_IRQN);
    number_of_bits_in_UART_frame=sizeof(uart_data_frame_received)/
sizeof(uart_data_frame_received[0]);
    DL_Timer_setRepeatCounter(CAPTURE_0_INST, number_of_bits_in_UART_frame);//Set the repeat
counter= No. of bits to be received in the UART Frame

    while(1){
        while (uart_frame_received==0){};

        if(uart_data_frame_received[0]!=0){
            uart_frame_erroneous=true;//If 0th Bit i.e. the START bit is not 0, raise an ERROR Flag
        }
        if(uart_data_frame_received[9]!=1){
            uart_frame_erroneous=true;//If 9th Bit i.e. the STOP bit is not 1, raise an ERROR Flag
        }
        for(int i=1;i<number_of_bits_in_UART_frame-1;i++){
            hex_value|= uart_data_frame_received[i]<<(i-1);//Convert the values to HEX
        }
        uart_data_received=hex_value;
        uart_frame_received=0;
    }
}
void CAPTURE_0_INST_IRQHandler(void)
{
    switch (DL_TimerA_getPendingInterrupt(CAPTURE_0_INST)) {
        case DL_TIMERA_IIDX_CC0_UP:
            data_bit_value=(DL_GPIO_readPins(GPIO_CAPTURE_0_C0_PORT,
GPIO_CAPTURE_0_C0_PIN))>>GPIO_CAPTURE_0_C0_PIN_BIT;//Read the GPIO value, to sample the input state
and the Bit value
            uart_data_frame_received[number_of_CC_interrupts++]=data_bit_value;//Store the bit
```


Copyright © 2026 Texas Instruments Incorporated

```
value in an array
            break;
        case DL_TIMERG_IIDX_ZERO:
                uart_frame_received=1;
                DL_TimerA_stopCounter(CAPTURE_0_INST);//Zero interrupt will come once all the UART
bits have been sampled, stop the counter when Zero Interrupt comes
                number_of_CC_interrupts=0;
                number_of_zero_interrupts=0;
            break;
        default:
            break;
    }
}
```

GPTIMER 还支持输入滤波机制，此功能可用于防止将不必要的干扰错误地采样为 START 位。

备注

1. 首先在 ISR 中优先读取 GPIO 值，以在正确的时间进行采样
2. 注意将 TIMER ISR 设置为最高优先级中断，否则可能会导致采样时间点出现偏差
3. CLK、CLKDIV 和 PRESCALER 的多种组合可用于实现波特率。较低的计时器功能时钟可能会降低捕获分辨率。

也可以使用 TIMG 代替 TIMA 来实现该应用。但是，需要考虑一个重要的限制因素：TIMG 不支持重复计数器功能。因此，在使用 TIMG 实现时，必须通过 ISR 内的软件手动重新加载计数器，以实现相同的功能。

备注

为优化应用吞吐量以实现更高的波特率，同时降低 CPU 带宽消耗，可以使用 DMA 来实现该应用。在此配置中，DMA 可以配置为在由计时器事件触发时读取 GPIO_DIN MMR，从而无需 CPU 干预。请参阅器件特定数据表，以确认 DMA 对 GPIO_DIN MMR 的可访问性。

4.2 使用 TIMA 仿真 UART Tx

TIMA 通过利用其输出生成功能、影子 CCACT 功能和重复计数器机制，为 UART 发送器仿真提供稳健的解决方案。通过整合 DMA 传输可以显著提升此实现的性能，从而创建优化系统资源的端到端解决方案。通过 DMA 处理数据移动，可以更大限度地减少 CPU 干预，同时确保精确的位时序和可靠的数据传输。

这种集成的计时器 DMA 方法消除了对中断优先级的依赖，并保证一致的时序精度。该解决方案可确保确定性的行为，使其成为位操作仿真应用的理想选择。图 4-3 展示了使用 TIMA 作为 UART Tx 的流程图。

English Document: SDAA286
Copyright © 2026 Texas Instruments Incorporated

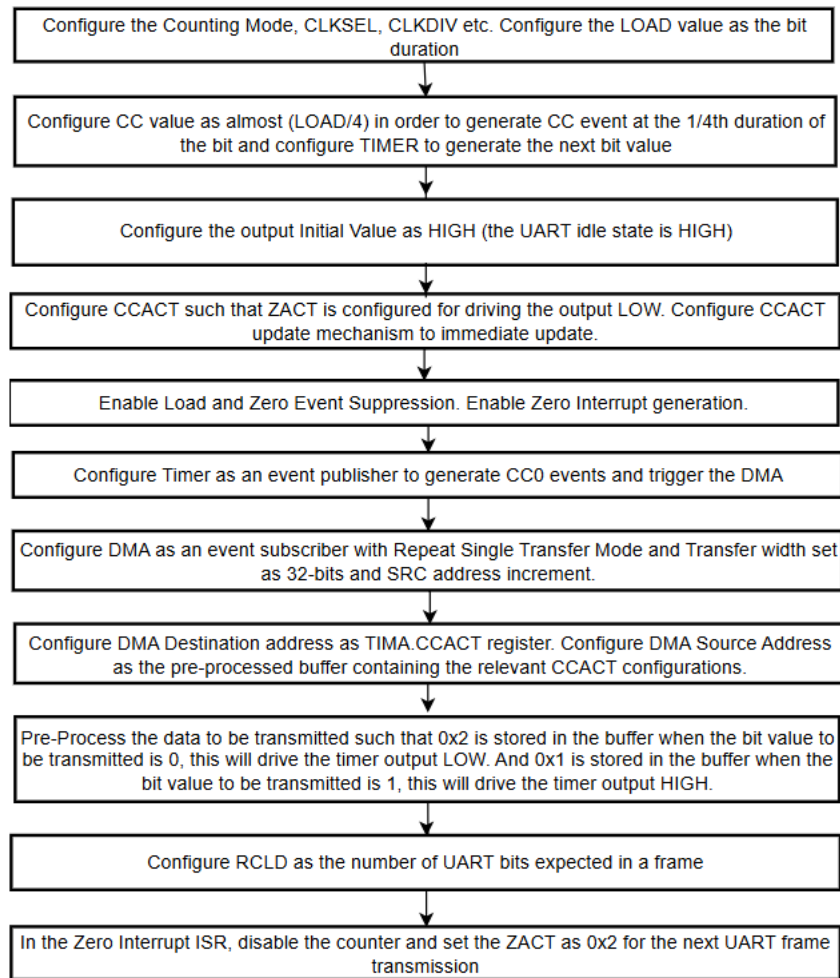

图 4-3. 展示 UART Tx 的流程图

配置序列如下详述：

- 配置计数模式、CLKSEL、CLKDIV 等参数。根据位持续时间配置加载值。
- 将 CC 值配置为“LOAD/4”或“LOAD/4+1”，这将在总位持续时间四分之一处生成 CC 事件，需要此事件以将计时器配置为发送下一个位值。
- 由于 UART 空闲状态为高电平，因此将输出初始值 OCTL.CCPIV 配置为高电平。
- 配置 CCACT 以使 ZACT 配置为将输出驱动为低电平，这是因为在启用计数器后应生成低电平 START 位。
- 将 CCACT 更新机制配置为立即更新。
- 启用重载和零事件抑制功能，这样仅当 RC=0 时才会生成零事件。
- 启用计时器零中断
- TIMA 可以通过事件互联矩阵生成事件并触发 DMA 等其他外设。需要将 TIMA 配置为事件发布者，以在 CC0 事件生成时触发 DMA 来启动数据传输。
- 需要将 DMA 配置为事件订阅者，以使 DMA 可以根据 TIMA 事件执行数据传输。
- 在重复单一传输模式下配置 DMA，宽度为 32 位且源地址递增
- 将 DMA 目标地址配置为 TIMA.CCACT 寄存器。每次发生 CC0 事件时，都需要更新 CCACT 寄存器才能发送下一个位值。此更新机制将由 DMA 执行。
- 预处理要传输的数据，以在要传输的位值为 0 时，将 0x2 存储在缓冲区中，这会将计时器输出驱动为低电平。并且，当要发送的位值为 1 时，将 0x1 存储在缓冲区中，这会将计时器输出驱动为高电平。
- 将 DMA 源地址配置为包含相关 CCACT 配置的预处理缓冲区
- 将 RCLD 配置为帧中预期的 UART 位数
- 要发送数据时启用计数器。

Copyright © 2026 Texas Instruments Incorporated

- 在零中断 ISR 中，禁用计数器，并将 ZACT 设置为 0x2 用于下一次 UART 帧传输。

```
/* Timer Configuration for Emulating UART Tx with DMA */
//For 9600 baud rate, single bit duration is 104.167us, hence configuring LOAD value as 3333, with
32MHz clock
static const DL_TimerA_ClockConfig gPWM_0ClockConfig = {
    .clockSel = DL_TIMER_CLOCK_BUSCLK,
    .divideRatio = DL_TIMER_CLOCK_DIVIDE_1,
    .prescale = 0U
};
static const DL_TimerA_PWMConfig gPWM_0Config = {
    .pwmMode = DL_TIMER_PWM_MODE_EDGE_ALIGN_UP,
    .period = 3333,
    .isTimerWithFourCC = false,
    .startTimer = DL_TIMER_STOP,
};

SYSCONFIG_WEAK void SYSCFG_DL_PWM_0_init(void) {
    DL_TimerA_setClockConfig(
        PWM_0_INST, (DL_TimerA_ClockConfig *) &gPWM_0ClockConfig);
    DL_TimerA_initPWMMode(
        PWM_0_INST, (DL_TimerA_PWMConfig *) &gPWM_0Config);

DL_TimerA_setCounterControl(PWM_0_INST,DL_TIMER_CZC_CCCTL0_ZCOND,DL_TIMER_CAC_CCCTL0_ACOND,DL_TIMER_
CLC_CCCTL0_LCOND);
    DL_TimerA_setCaptureCompareOutCtl(PWM_0_INST, DL_TIMER_CC_OCTL_INIT_VAL_HIGH, //Keeping CC0
initial value as HIGH as UART idle state should be HIGH
        DL_TIMER_CC_OCTL_INV_OUT_DISABLED, DL_TIMER_CC_OCTL_SRC_FUNCVAL,
        DL_TIMERA_CAPTURE_COMPARE_0_INDEX);
    PWM_0_INST->COUNTERREGS.CCACT_01[0]=0x2;//Configuring Zero action as CC Output LOW, as START
bit in UART is always LOW
    DL_TimerA_setCaptCompUpdateMethod(PWM_0_INST, DL_TIMER_CC_UPDATE_METHOD_IMMEDIATE,
DL_TIMERA_CAPTURE_COMPARE_0_INDEX);
    DL_Timer_setCaptCompActUpdateMethod(PWM_0_INST,DL_TIMER_CCACT_UPDATE_METHOD_IMMEDIATE,
DL_TIMERA_CAPTURE_COMPARE_0_INDEX);
    DL_TimerA_setCaptureCompareValue(PWM_0_INST, 834, DL_TIMER_CC_0_INDEX);//Configuring CC0 as
almost 1/4th the bit width
    PWM_0_INST->COUNTERREGS.CCACT_01[0]=0x2;//Configuring Zero action as CC Output LOW, as START
bit in UART is always LOW
    DL_Timer_enableInterrupt(PWM_0_INST, DL_TIMER_INTERRUPT_ZERO_EVENT);
    DL_Timer_enableLZEventSuppression(PWM_0_INST);//This will suppress zero events until RC is
equal to 0

    DL_Timer_enableEvent(PWM_0_INST, DL_TIMER_EVENT_ROUTE_1, DL_TIMER_INTERRUPT_CC0_UP_EVENT);//
CC0_UP Event triggers the DMA CHAN0
    DL_Timer_setPublisherChanID(PWM_0_INST, DL_TIMER_PUBLISHER_INDEX_0, 1);//Timer publishing event
to DMA to carry out CCACT update
    DL_TimerA_enableClock(PWM_0_INST);
    DL_TimerA_setCCPDirection(PWM_0_INST , DL_TIMER_CC0_OUTPUT );
}
// Below is DMA configuration
static const DL_DMA_Config gDMA_CH0Config = {
    .transferMode   = DL_DMA_FULL_CH_REPEAT_SINGLE_TRANSFER_MODE,
    .extendedMode   = DL_DMA_NORMAL_MODE,
    .destIncrement  = DL_DMA_ADDR_UNCHANGED,
    .srcIncrement   = DL_DMA_ADDR_INCREMENT,
    .destWidth      = DL_DMA_WIDTH_WORD,
    .srcWidth       = DL_DMA_WIDTH_WORD,
    .trigger        = DMA_GENERIC_SUB0_TRIG,
    .triggerType    = DL_DMA_TRIGGER_TYPE_EXTERNAL,
};
void SYSCFG_DL_DMA_CH0_init(void)
{
    DL_DMA_initChannel(DMA, DMA_CH0_CHAN_ID , (DL_DMA_Config *) &gDMA_CH0Config);
    DL_DMA_clearInterruptStatus(DMA, DL_DMA_INTERRUPT_CHANNEL0);
        DL_DMA_enableInterrupt(DMA, DL_DMA_INTERRUPT_CHANNEL0);
        DL_DMA_setSubscriberChanID(DMA, DL_DMA_SUBSCRIBER_INDEX_0, 0x01);//DMA subscribing to Timer
event
}

void SYSCFG_DL_DMA_init(void){
    SYSCFG_DL_DMA_CH0_init();
}
```

```
/* Application Code for Emulating UART Tx Using Timer */
uint8_t data_to_transmit[8]={1,1,0,1,0,1,0,0};//this equals to 0x2B
uint8_t uart_frame_to_transmit[9];//8 data bits + STOP Bit
```


English Document: SDAA286
Copyright © 2026 Texas Instruments Incorporated

```
uint8_t number_of_bits_in_each_frame=0;
uint8_t number_of_bits_in_uart_frame=0;
uint32_t DMA_frame_to_transmit[9];
int number_of_CC_interrupts=0;
int number_of_zero_int=0;
//int number_of_load_int=0;
volatile bool gIsTimerExpired;
int main(void)
{
    bool isRetentionError;

    NVIC_EnableIRQ(PWM_0_INST_INT_IRQN);

    SYSCFG_DL_init();
    DL_DMA_setDestAddr(DMA, DMA_CH0_CHAN_ID, (uint32_t)&PWM_0_INST->COUNTERREGS.CCACT_01[0]);//Set
DMA destination address as TIM_CCACT MMR
    DL_DMA_setSrcAddr(DMA, DMA_CH0_CHAN_ID, (uint32_t)&DMA_frame_to_transmit );//Set DMA source
address as the array which contains CCACT config for each bit
    DL_DMA_setTransferSize( DMA, DMA_CH0_CHAN_ID,number_of_bits_in_uart_frame);//Configure DMA SZ
for the number of bits in each frame

    DL_DMA_enableChannel(DMA, DMA_CH0_CHAN_ID);//Enable DMA channel

    number_of_bits_in_each_frame=sizeof(data_to_transmit);
    uart_frame_to_transmit[number_of_bits_in_each_frame+1]=1;//STOP bit is always 1

    for(int i=0;i<number_of_bits_in_each_frame;i++){
        uart_frame_to_transmit[i]=data_to_transmit[i];
        }

    number_of_bits_in_uart_frame=sizeof(uart_frame_to_transmit); //Adding 1 Stop bit Frame

    DL_Timer_setRepeatCounter(PWM_0_INST,number_of_bits_in_uart_frame);//Configuring RCLD as the
number_of_bits_in_uart_frame
/////////PRE-PROCESS THE ARRAY WHICH CONTAINS THE CORRESPONDING CCACT CONFIGURATION FOR EACH
BIT/////////////////////////

for( number_of_CC_interrupts=0;number_of_CC_interrupts<(number_of_bits_in_uart_frame+1);number_of_CC
_interrupts++){
    if(uart_frame_to_transmit[number_of_CC_interrupts]==0){
        DMA_frame_to_transmit[number_of_CC_interrupts]=0x2;//if bit value is 0 configure ZACT as CC
Output LOW
    }
    else if(uart_frame_to_transmit[number_of_CC_interrupts]==1){
        DMA_frame_to_transmit[number_of_CC_interrupts]=0x1;//if bit value is 1 configure ZACT as CC
Output HIGH
    }
    else{
    }
    if(number_of_CC_interrupts==number_of_bits_in_uart_frame-1){
        DMA_frame_to_transmit[number_of_CC_interrupts]=0x1;//Always generate a HIGH stop bit
    }
}
////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////
    DL_TimerA_startCounter(PWM_0_INST);//Start Counter
    while(1);
}

void PWM_0_INST_IRQHandler(void)
{
    switch (DL_TimerA_getPendingInterrupt(PWM_0_INST)) {
        case DL_TIMER_IIDX_ZERO:
            number_of_zero_int++;
            DL_TimerA_stopCounter(PWM_0_INST);//Stop the counter
            PWM_0_INST->COUNTERREGS.CCACT_01[0]=0x2;//Configure ZACT for the next UART Transmission
            delay_cycles(3333);//Adding 1 bit of delay between 2 UART frames
            number_of_CC_interrupts=0;
            DL_TimerA_startCounter(PWM_0_INST);//Start counter for the next UART Transmission
            break;
        default:
            break;
    }
}
```


Copyright © 2026 Texas Instruments Incorporated

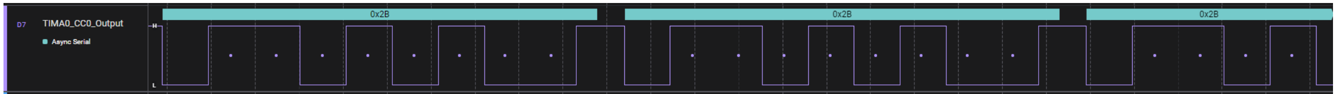

图 4-4. 使用 TIMA 以 UART 格式发送 0x2B

备注

对于上述应用，可以使用 TIMG 而不是 TIMA 作为替代实现。但是，TIMG 不支持重复计数器功能。因此，使用 TIMG 时，需要通过 ISR 内的软件干预手动重新加载计数器。

English Document: SDAA286
Copyright © 2026 Texas Instruments Incorporated

5 基于反馈的 PWM 生成

基于反馈的脉宽调制 (PWM) 信号生成用于根据系统反馈动态调整脉冲宽度，从而在各种应用中实现精确控制和调节。这对于需要持续监控和适应变化条件的应用来说特别有用，例如电机转速控制、电源调节和电池管理系统。

5.1 基于反馈的 PWM 信号复制

生成输入 PWM 信号精确副本的功能在现代电子系统中发挥着重要作用，尤其是在需要安全冗余验证的应用中。该方法在双通道电机控制系统中广泛实现；在此类系统中，信号验证和故障检测对于运行安全至关重要。

在工业和安全关键型应用中，PWM 信号复制可实现强大的冗余控制路径和持续的系统监控。该技术涉及捕获输入 PWM 信号的特性并生成完全相同的输出信号，从而保持精确的时序关系。

此功能在电力电子应用中尤为重要，因为在这些应用中，同步开关控制和多个相位对齐输出至关重要。该实现允许信号缓冲、再生和扇出到多个子系统，同时保持信号完整性和抗噪性。

在电机控制应用中，这种方法可实现冗余的栅极驱动信号和双通道安全系统，对于容错运行至关重要。该系统会持续监控和验证信号，从而提供实时验证和性能监控功能。这在安全性和可靠性至关重要的工业设备中尤其重要。

该实现简单但功能强大，利用计时器捕获功能读取输入 PWM 参数并生成完全相同的输出信号。这可确保精确复制周期值和占空比值，从而在整个系统中保持信号完整性。由此产生的解决方案具有全面的优势，包括信号验证、安全冗余、分布式控制功能和稳健的系统监控功能。

执行以下步骤，可将计时器实例配置为生成基于反馈的 PWM：

- 在递增计数模式下配置计时器。
- 配置 CC0 通道以进行上升沿和下降沿的边沿捕获。
- 在 CC0 通道上输入基准 PWM 信号。
- 在输出模式下配置 CC1，以生成基于反馈的 PWM。
- 将 CC1 通道的 CCCTL.CC2SELU 配置为 CC0，并将 CCACT.CC2UACT 字段配置为 CCP 输出切换。这将根据 CC0 捕获事件切换 CC1 输出。
- CC0 通道上的捕获事件将由输入基准信号生成。

```
/* Configuration Sequence to Generate Feedback-based PWM */
static const DL_TimerA_ClockConfig gCAPTURE_0ClockConfig = {
    .clockSel    = DL_TIMER_CLOCK_BUSCLK,
    .divideRatio = DL_TIMER_CLOCK_DIVIDE_1,
    .prescale = 0U
};

static const DL_TimerA_CaptureConfig gCAPTURE_0CaptureConfig = {
    .captureMode    = DL_TIMER_CAPTURE_MODE_EDGE_TIME_UP,
    .period         = CAPTURE_0_INST_LOAD_VALUE,
    .startTimer     = DL_TIMER_STOP,
    .edgeCaptMode   = DL_TIMER_CAPTURE_EDGE_DETECTION_MODE_EDGE,//Enable Edge Capture on both
rising and falling edges for CC0
    .inputChan      = DL_TIMER_INPUT_CHAN_0,
    .inputInvMode   = DL_TIMER_CC_INPUT_INV_NOINVERT,
};

SYSCONFIG_WEAK void SYSCFG_DL_CAPTURE_0_init(void) {

    DL_TimerA_setClockConfig(CAPTURE_0_INST,
        (DL_TimerA_ClockConfig *) &gCAPTURE_0ClockConfig);

    DL_TimerA_initCaptureMode(CAPTURE_0_INST,
        (DL_TimerA_CaptureConfig *) &gCAPTURE_0CaptureConfig);

DL_TimerA_setCounterControl(CAPTURE_0_INST,DL_TIMER_CZC_CCCTL0_ZCOND,DL_TIMER_CAC_CCCTL0_ACOND,DL_TI
MER_CLC_CCCTL0_LCOND);

    DL_TimerA_enableInterrupt(CAPTURE_0_INST , DL_TIMERA_INTERRUPT_CC0_UP_EVENT);

    CAPTURE_0_INST->COUNTERREGS.CCACT_01[1]=0x00018000;//Enable CC output Toggle for Secondary CC
event. Secondary Event for CC1 will be generated when CC0 captures an edge
    DL_TimerA_setCaptureCompareValue(CAPTURE_0_INST, 0, DL_TIMER_CC_1_INDEX);
    DL_TimerA_enableClock(CAPTURE_0_INST);
```


English Document: SDAA286
Copyright © 2026 Texas Instruments Incorporated

```
    DL_TimerA_setCCPDirection(CAPTURE_0_INST , DL_TIMER_CC1_OUTPUT);
}
```

上述配置适用于基准 PWM 的周期和占空比不断发生变化的情况。

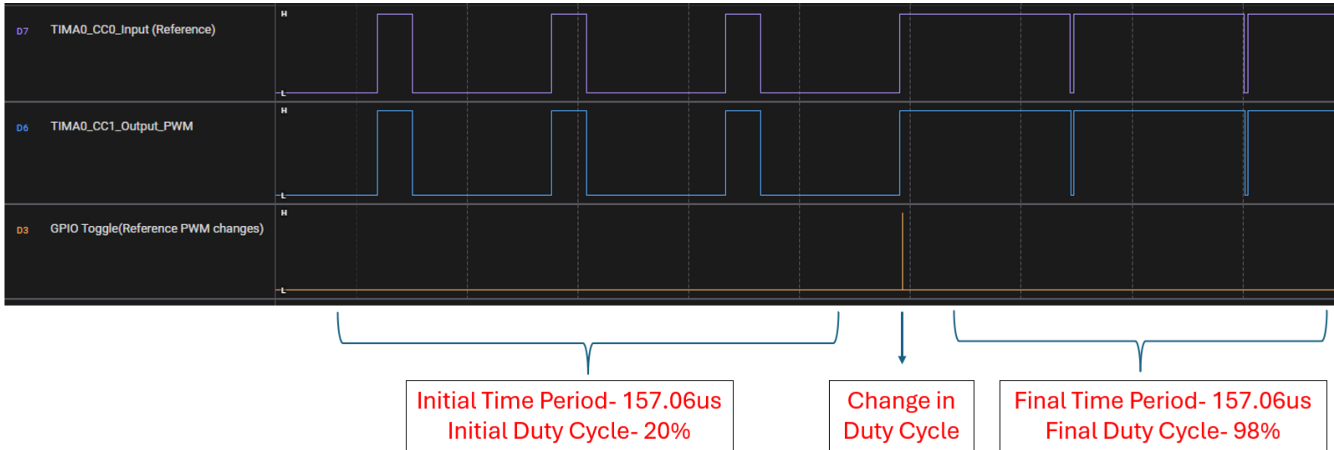

图 **5-1. 仅基准 PWM 的占空比发生变化**

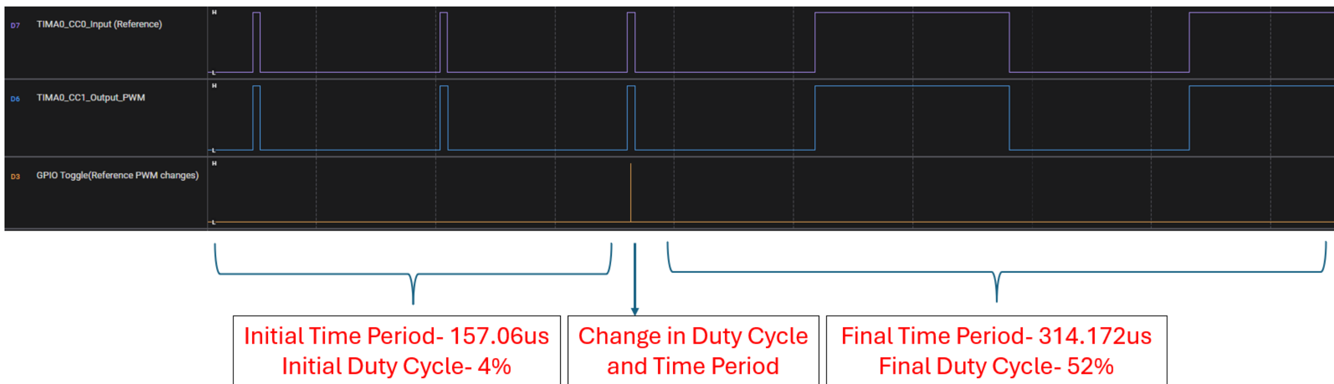

图 **5-2. 基准 PWM 的占空比和时间周期均发生变化**

5.2 使用输入基准生成延迟的 PWM 信号

带受控延迟的反馈式 PWM 生成技术可为多个应用领域提供通用解决方案，尤其是在复杂的电机控制和电力电子系统中。这种技术能够实现精确的相移 PWM 生成，对于双电机同步和互补信号中的死区时间插入至关重要。该实现方案支持精确的时序控制，这在 LED 显示应用和多相电机驱动系统中至关重要。

在电力电子应用中，此方法证明了其对于多相直流/直流转换器和相移电桥转换器不可或缺。引入受控延迟的功能可实现半桥驱动器的高效死区时间插入，并有助于交错电源运行。系统可以精确地生成具有特定时序偏移的输出 PWM 信号。

此功能可扩展到信号处理应用，其中分布式系统中的受控延迟信号中继器和时序补偿至关重要。该技术支持具有相位控制的复杂时钟信号分配，并实现稳健的级联控制系统。具体实现方案包括具有相移的同步降压转换器以及需要精确相位控制的多通道 LED 调光应用。

该实现方案利用计时器捕获功能来测量输入信号时序，并生成具有精确相位关系的延迟输出。这种方法可确保准确的时序延迟，同时在整个系统内保持同步运行。由此产生的解决方案具有显著的优势，包括精确的相位控制、确定性时序延迟以及通过信号重定时增强的抗噪性。

该解决方案在同步功率级控制和需要在多个 PWM 信号之间实现精确相位关系的应用中特别重要。

English Document: SDAA286
Copyright © 2026 Texas Instruments Incorporated

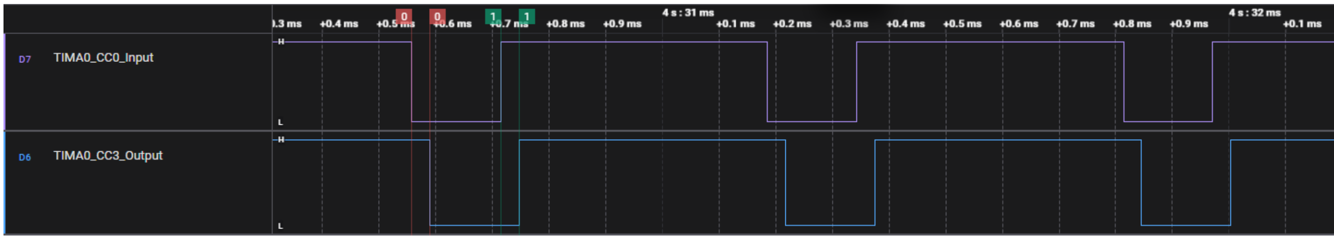

图 5-3. 通过输入基准以固定延迟生成的 PWM 信号

配置序列如下所述

- 将加载值和 CM 配置为递增计数模式。
- 在捕获模式下，将 CC0 配置为上升沿检测，并将 CC1 配置为下降沿检测。配置零条件 (CCCTL.ZCOND) 以使 CC0 上升沿生成零条件。
- 将 CC1 的 IFCTL.ISEL 配置为 0x2，以使 CC0 上的输入也馈送到 CC1 通道。
- 基准输入 PWM 将馈送到 CC0 通道。内部 CC0 输入也馈送到 CC1 通道
- 将 CC4 配置为 CC3 的辅助 CC 通道，这意味着 CC4 比较事件将生成 CC3 的辅助比较事件。
- 启用 CC0_UP 和 CC1_UP 中断。
- 为 CC3 配置 CCACT 寄存器，使得 CUACT 将 CC 输出驱动为高电平，CC2UACT 将 CC 输出驱动为低电平。
- 将 CC3 设置为输出。具有固定延迟的最终输出信号将由 CC3 生成。
- 在中断服务例程中，使用 CC0_UP 中断，根据应用要求的延迟值更新 CC3 的数值。使用 CC1_UP 中断，根据应用要求的延迟值更新 CC4 的数值。

备注

在具有多个 ISR、更高优先级中断或高 PWM 频率的应用中，必须注意提高计时器 ISR 的优先级并优化 ISR 代码，以尽可能快速更新 CC 值。

```
/* Configuration Sequence for Delayed Timer Generation */

static const DL_TimerA_ClockConfig gCAPTURE_0ClockConfig = {
    .clockSel    = DL_TIMER_CLOCK_BUSCLK,
    .divideRatio = DL_TIMER_CLOCK_DIVIDE_1,
    .prescale = 0U
};

static const DL_TimerA_CaptureConfig gCAPTURE_0CaptureConfig = {
    .captureMode    = DL_TIMER_CAPTURE_MODE_EDGE_TIME_UP,
    .period         = 65535,
    .startTimer     = DL_TIMER_STOP,
    .edgeCaptMode   = DL_TIMER_CAPTURE_EDGE_DETECTION_MODE_RISING,//Enable Rising Edge Capture for
CC0
    .inputChan      = DL_TIMER_INPUT_CHAN_0,
    .inputInvMode   = DL_TIMER_CC_INPUT_INV_NOINVERT,
};

SYSCONFIG_WEAK void SYSCFG_DL_CAPTURE_0_init(void) {

    DL_TimerA_setClockConfig(CAPTURE_0_INST,
        (DL_TimerA_ClockConfig *) &gCAPTURE_0ClockConfig);

    DL_TimerA_initCaptureMode(CAPTURE_0_INST,
        (DL_TimerA_CaptureConfig *) &gCAPTURE_0CaptureConfig);

DL_TimerA_setCounterControl(CAPTURE_0_INST,DL_TIMER_CZC_CCCTL0_ZCOND,DL_TIMER_CAC_CCCTL0_ACOND,DL_TI
MER_CLC_CCCTL0_LCOND);

    DL_TimerA_enableInterrupt(CAPTURE_0_INST , DL_TIMERA_INTERRUPT_CC0_UP_EVENT|
DL_TIMERA_INTERRUPT_CC1_UP_EVENT);
    CAPTURE_0_INST->COUNTERREGS.CCCTL_01[0]=0x21001;
    CAPTURE_0_INST->COUNTERREGS.CCCTL_01[1]=0x20002;
    CAPTURE_0_INST->COUNTERREGS.CCCTL_23[1]=0x01000000;
    DL_TimerA_setCaptureCompareInput(CAPTURE_0_INST, DL_TIMER_CC_INPUT_INV_NOINVERT,
```


Copyright © 2026 Texas Instruments Incorporated

```
DL_TIMER_CC_IN_SEL_CCP0, DL_TIMER_CC_1_INDEX);
    CAPTURE_0_INST->COUNTERREGS.CCACT_23[1]=0x10200;
    DL_TimerA_enableClock(CAPTURE_0_INST);
    DL_TimerA_setCCPDirection(CAPTURE_0_INST , DL_TIMER_CC3_OUTPUT);
}
```

```
/* Interrupt Service Routine to Update the CC Values with a Fixed Delay of 1000 Cycles as an
Example */
void CAPTURE_0_INST_IRQHandler(void)
{
    switch (DL_TimerA_getPendingInterrupt(CAPTURE_0_INST)) {
        case DL_TIMERA_IIDX_CC0_UP:
            CC0_value=DL_Timer_getCaptureCompareValue(CAPTURE_0_INST, DL_TIMER_CC_0_INDEX);
            DL_Timer_setCaptureCompareValue(CAPTURE_0_INST, CC0_value+1000, DL_TIMER_CC_3_INDEX);
            CC0_int++;
            break;
        case DL_TIMERA_IIDX_CC1_UP:
            CC1_value=DL_Timer_getCaptureCompareValue(CAPTURE_0_INST, DL_TIMER_CC_1_INDEX);
            DL_Timer_setCaptureCompareValue(CAPTURE_0_INST, CC1_value+1000, DL_TIMER_CC_4_INDEX);
            CC1_int++;
            break;
        default:
            break;
    }
}
```


Copyright © 2026 Texas Instruments Incorporated

6 延迟计时器启动：具有可配置延迟的同步计时器实例启动

具有有意设置的启动时间偏移的计时器可实现受控的相位关系，这些关系在涉及电机控制以及传感器数据轮询或采样的某些应用中对于效率、稳定性和信号完整性至关重要。当使用两个或更多个计时器时，可以通过如下所述的交叉触发机制同步使用这些计时器来实现这一点：

- 按照 节 3 在基于软件的交叉触发机制中配置计时器。
- 根据具体应用，首先启用的计时器可以配置为主要计时器。
- 根据所需的启动时间偏移配置 CC 值。
- 在 CC 事件 ISR 内生成软件交叉触发以启用辅助计时器。

备注

由于延迟限制阻止了在第一个 PWM 输出需要特定占空比时在第二个计时器上设置延迟，因此使用第一个 PWM 的 CC4/5 事件将为配置延迟提供必要的灵活性。

例如，启用 TIMA0 和 TIMA1 之间需要 10ms 的偏移。如果需要先启用 TIMA1，请将 TIMA1 配置为主要计时器，并将 TIMA0 配置为辅助计时器。配置 TIMA1 的 CC 值，以使 CC 事件在启用恰好 10ms 后生成。在此 CC 事件内，可生成软件交叉触发，以便在恰好 10ms 后启用 TIMA0。这意味着，当 CLKSEL 为 LFCLK (32KHz) 时，TIMA1 的 CC 值可以配置为 320，计时器为递增计数模式，这将在约 10ms（320 个周期 × (1/32000) 秒= 0.01 秒 = 10ms）后生成 CC 事件。该 CC 事件将通过基于软件的交叉触发机制启用 TIMA0。

```
/* TIMA Interrupt Service Routine */
void PWM_0_INST_IRQHandler(void)
{
    switch (DL_TimerA_getPendingInterrupt(PWM_0_INST)) {
        case DL_TIMERA_IIDX_CC0_DN:
            if(interrupt_counter==0){
            DL_TimerA_generateCrossTrigger(PWM_0_INST);//Mechanism to generate Software based Cross-
Trigger
            }
            else{
                ;
            }
            interrupt_counter++;
            break;
        default:
            break;
    }
}
```

备注

虽然硬件交叉触发功能也可用于此应用，但务必要认识到此机制将在每个 CC 事件发生时生成交叉触发。因此，如果开发人员打算利用硬件交叉触发功能，则必须在一次交叉触发后显式禁用该功能，此应用才能按预期正常运行。

Copyright © 2026 Texas Instruments Incorporated

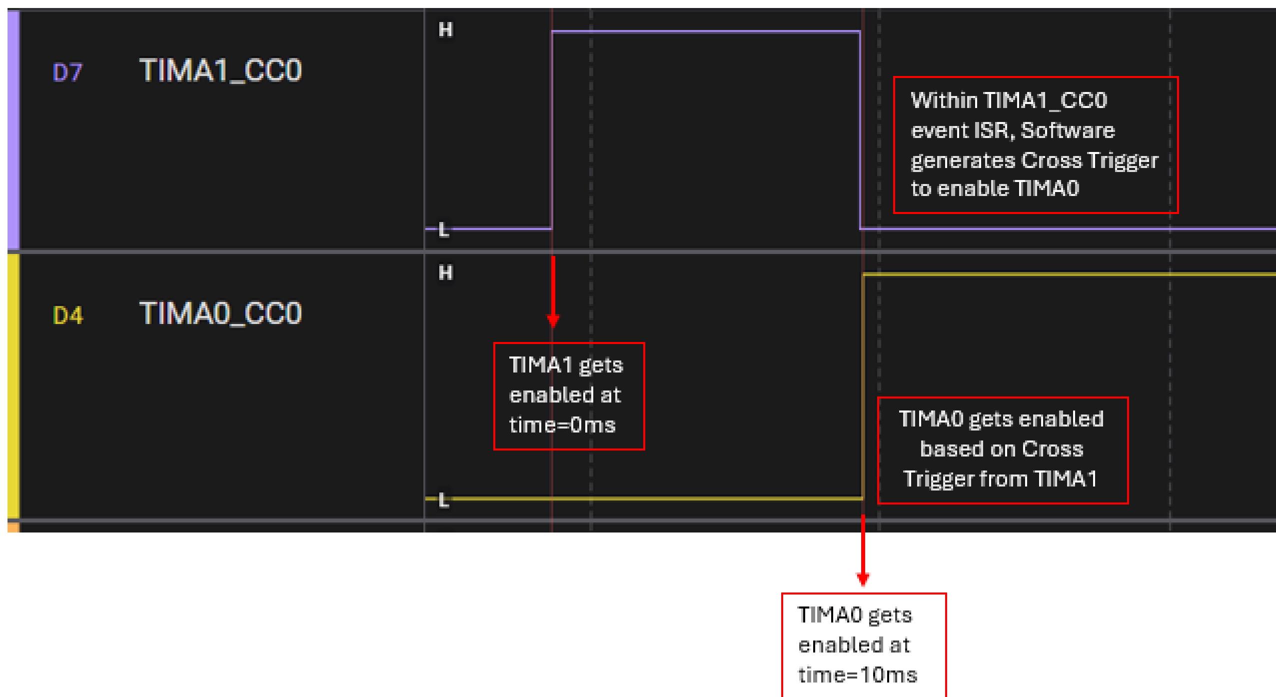

图 **6-1.** 可配置计时器启动时间偏移

Copyright © 2026 Texas Instruments Incorporated

7 基于硬件事件停止正在运行的计时器

使用硬件事件可停止正在运行的计时器，实现对计时功能的低延迟、精确和自主控制。这可以实时测量和响应，在检测、安全和控制系统中至关重要。图 7-1 展示了用于停止正在运行的计时器的流程图。

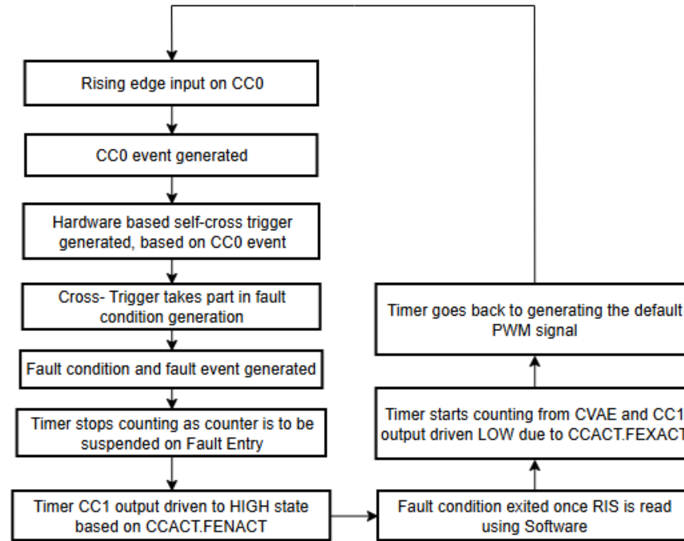

图 7-1. 表示硬件如何停止正在运行的计时器的流程图

可以完成高级计时器 (TIMA) 中的以下配置来实现此要求：

- 配置 TIMA 以在发生 CC 事件时生成自交叉触发。
- 配置 CC0 以进行上升沿捕获。CC0 引脚上的外部上升沿将生成 CC0 事件和自交叉触发。
- 为所使用的 TIMA 实例配置基于硬件的故障检测机制。
- 将 FCTL.TFIM 位配置为“1”，意味着所选触发将参与故障条件生成，这将导致交叉触发生成故障条件以停止计时器。
- 将 FCTL.FL 配置为“1”以启用故障锁存模式，这意味着整体故障条件取决于 CPU_INT.RIS.F 位。
- 配置故障响应为暂停计数器，这将停止正在运行的计数器。

```
/* Configuration Sequence to Stop a Running Timer Using Hardware Events */
static const DL_TimerA_ClockConfig gPWM_0ClockConfig = {
    .clockSel = DL_TIMER_CLOCK_BUSCLK,
    .divideRatio = DL_TIMER_CLOCK_DIVIDE_1,
    .prescale = 0U
};

static const DL_TimerA_PWMConfig gPWM_0Config = {
    .pwmMode = DL_TIMER_PWM_MODE_EDGE_ALIGN_UP,
    .period = 1600,
    .isTimerWithFourCC = false,
    .startTimer = DL_TIMER_STOP,
};
SYSCONFIG_WEAK void SYSCFG_DL_PWM_0_init(void) {

    DL_TimerA_setClockConfig(
        PWM_0_INST, (DL_TimerA_ClockConfig *) &gPWM_0ClockConfig);
    DL_TimerA_initPWMMode(
        PWM_0_INST, (DL_TimerA_PWMConfig *) &gPWM_0Config);
    // Set Counter control to the smallest CC index being used

DL_TimerA_setCounterControl(PWM_0_INST,DL_TIMER_CZC_CCCTL0_ZCOND,DL_TIMER_CAC_CCCTL0_ACOND,DL_TIMER_
CLC_CCCTL0_LCOND);

    DL_TimerA_setCaptCompUpdateMethod(PWM_0_INST, DL_TIMER_CC_UPDATE_METHOD_IMMEDIATE,
DL_TIMERA_CAPTURE_COMPARE_0_INDEX);

    DL_TimerA_setCaptureCompareOutCtl(PWM_0_INST, DL_TIMER_CC_OCTL_INIT_VAL_LOW,
        DL_TIMER_CC_OCTL_INV_OUT_DISABLED, DL_TIMER_CC_OCTL_SRC_FUNCVAL,
        DL_TIMERA_CAPTURE_COMPARE_1_INDEX);
```


English Document: SDAA286
Copyright © 2026 Texas Instruments Incorporated

```
    DL_TimerA_setCaptCompUpdateMethod(PWM_0_INST, DL_TIMER_CC_UPDATE_METHOD_IMMEDIATE,
DL_TIMERA_CAPTURE_COMPARE_1_INDEX);
    DL_TimerA_setCaptureCompareValue(PWM_0_INST, 500, DL_TIMER_CC_1_INDEX);//Set CC1 value, this is
to generate PWM using CC1

//Configure Fault such that Cross Trigger generates a fault condition, which will be latched unless
the Fault RIS is cleared
    DL_TimerA_setFaultConfig(PWM_0_INST, DL_TIMER_FAULT_CONFIG_TFIM_ENABLED|
DL_TIMER_FAULT_CONFIG_FL_LATCH_SW_CLR|
                                         DL_TIMER_FAULT_CONFIG_FI_DEPENDENT|
DL_TIMER_FAULT_CONFIG_FIEN_ENABLED);

    DL_TimerA_configFaultOutputAction(PWM_0_INST,
                                      DL_TIMER_FAULT_ENTRY_CCP_HIGH,
                                      DL_TIMER_FAULT_EXIT_CCP_LOW, DL_TIMER_CC_1_INDEX);
//Generating a Fault condition will stop the Counter if Fault Entry Action is set as Fault Counter
Suspend Counting
    DL_TimerA_configFaultCounter(PWM_0_INST,
                                 DL_TIMERA_FAULT_ENTRY_CTR_SUSP_COUNT,
                                 DL_TIMERA_FAULT_EXIT_CTR_CVAE_ACTION);
    DL_TimerA_setFaultSourceConfig(
            PWM_0_INST, DL_TIMERA_FAULT_SOURCE_EXTERNAL_0_SENSE_HIGH);
    PWM_0_INST->COUNTERREGS.CCCTL_01[0]=0x20001;//CC0 configured for rising edge capture
    DL_TimerA_enableClock(PWM_0_INST);
    DL_TimerA_setCCPDirection(PWM_0_INST ,  DL_TIMER_CC1_OUTPUT );
    DL_TimerA_configCrossTrigger(PWM_0_INST, DL_TIMER_CROSS_TRIG_SRC_CCU0,
                                 DL_TIMER_CROSS_TRIGGER_INPUT_ENABLED,
DL_TIMER_CROSS_TRIGGER_MODE_ENABLED
        );//Configuration to Generate Hardware Based Cross Trigger on CC0 Event
    DL_TimerA_setCaptureCompareInput(PWM_0_INST, DL_TIMER_CC_INPUT_INV_NOINVERT,
DL_TIMER_CC_IN_SEL_TRIG, DL_TIMER_CC_1_INDEX);
    DL_TimerA_setExternalTriggerEvent(PWM_0_INST,DL_TIMER_EXT_TRIG_SEL_TRIG_1);//This is to receive
the self cross trigger generated by TIMA1
    DL_TimerA_enableExternalTrigger(PWM_0_INST);
}
```

备注

上述应用利用 CC0 的上升沿来生成自交叉触发，从而生成故障条件以停止计时器。类似地，TIMA 通过事件结构接收的源自其他外设的事件（例如 GPIO 事件）也可用于生成自交叉触发，从而生成故障条件，以使用硬件停止计时器。

English Document: SDAA286
Copyright © 2026 Texas Instruments Incorporated

8 动态 PWM 更新：占空比和时间周期调整

在运行时修改 PWM 特性的功能可实现对各种应用的精确控制，从而提供灵活的频率和占空比调整。通过动态更新计时器比较 (CC) 和加载值，系统可在保持稳定运行的同时实现实时控制。使用影子寄存器功能实现时，该功能变得特别强大，可确保在预先确定的更新点进行无毛刺转换。

在电机控制系统中，该功能通过逐渐的频率调整来实现平稳的速度斜升，并通过占空比调制来实现精确的扭矩控制。在 LED 和照明应用中，这一机制同样不可或缺，平滑的亮度转换和复杂的调光模式对于获得优质用户体验至关重要。电源应用可受益于自适应电压调节和动态频率调节，从而实现出色的效率；而温度控制系统则通过基于 PWM 的加热器控制和风扇转速调制，利用该功能进行精确的热管理。

8.1 影子加载和影子比较功能

TIMA 提供了用于配置 CC 和 LOAD 值的影子寄存器，以支持动态变化的 PWM 占空比和周期，如图 8-1 所示。影子寄存器机制通过确保参数更新在安全转换点（通常在周期边界处）发生，在保持系统稳定性方面发挥着至关重要的作用。这种系统化方法可防止 PWM 输出毛刺、不规则脉冲宽度和意外频率变化，这些问题可能在直接更新寄存器时发生。因此，对于需要平滑转换和精确时序控制的应用而言，这是一种稳健的解决方案，这在注重可预测行为和稳定运行的系统中尤为宝贵。

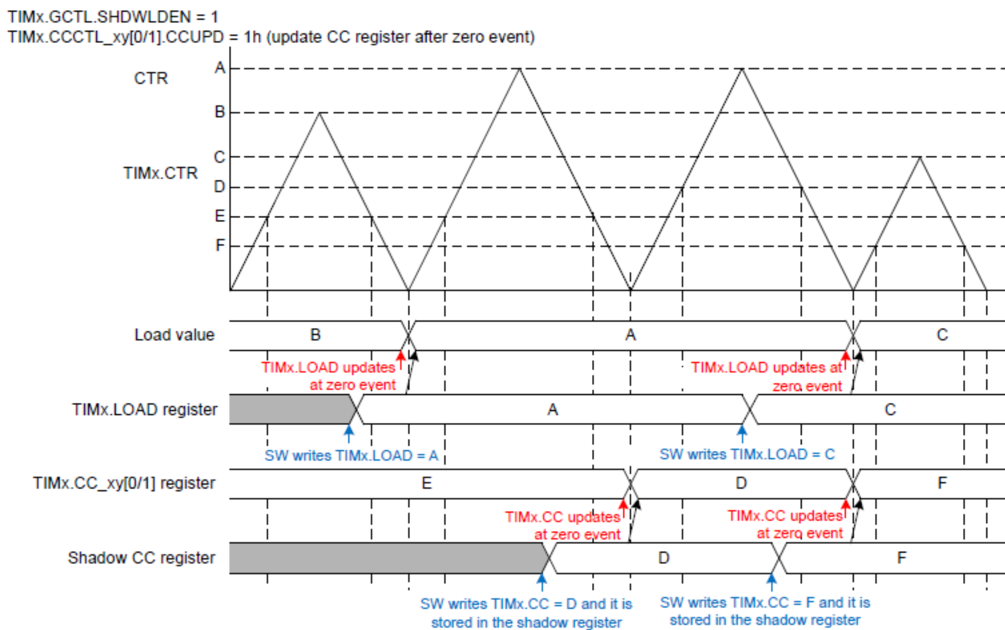

图 8-1. 在递增/递减模式下，影子加载和影子比较在发生归零事件时生效

备注

如果没有影子寄存器，当应用动态更新 PWM 占空比时，可能会随机发生意外的 PWM 输出（0% 或 100% 占空比）。为缓解此问题，请启用该计时器的重载或零事件触发中断，然后在中断服务程序 (ISR) 内执行占空比更新，同时设置合理的最大/最小占空比限制，以抵消中断处理延迟的影响。此外，请考虑使用 DMA 更新 CC 值来减少 CPU 开销和中断处理延迟。

该实现方案在保持系统完整性的同时提供了出色的灵活性，因此非常适合从简单亮度控制到复杂电机驱动系统的各种应用。无论是实现软启动序列、热管理系统还是复杂的 PID 控制环路，这种可在动态修改 PWM 参数的同时确保无毛刺运行的能力，为现代嵌入式系统设计提供了强大的工具。

影子功能可用于在固定基准点更新 CC 和加载值。需要更改加载值时，可以直接写入 TIMx.LOAD 值。在内部，无论何时由软件更新，加载值仅在发生零事件时更新。需要更改 CC 值时，直接写入 TIMx.CC 值。在内部，无论何

English Document: SDAA286
Copyright © 2026 Texas Instruments Incorporated

时由软件更新，CC 值仅对 CC 更新方法中配置的事件更新。通过写入 TIMx.GCTL.SHDWLDEN (=1) 寄存器来启用影子功能。

```
/* Configuration Sequence for Shadow Load and Shadow CC Update */
static const DL_TimerA_ClockConfig gPWM_0ClockConfig = {
    .clockSel = DL_TIMER_CLOCK_BUSCLK,
    .divideRatio = DL_TIMER_CLOCK_DIVIDE_1,
    .prescale = 0U
};

static const DL_TimerA_PWMConfig gPWM_0Config = {
    .pwmMode = DL_TIMER_PWM_MODE_EDGE_ALIGN_UP,
    .period = 5000,
    .isTimerWithFourCC = false,
    .startTimer = DL_TIMER_STOP,
};

SYSCONFIG_WEAK void SYSCFG_DL_PWM_0_init(void) {

    DL_TimerA_setClockConfig(
        PWM_0_INST, (DL_TimerA_ClockConfig *) &gPWM_0ClockConfig);

    DL_TimerA_initPWMMode(
        PWM_0_INST, (DL_TimerA_PWMConfig *) &gPWM_0Config);

    // Set Counter control to the smallest CC index being used

DL_TimerA_setCounterControl(PWM_0_INST,DL_TIMER_CZC_CCCTL0_ZCOND,DL_TIMER_CAC_CCCTL0_ACOND,DL_TIMER_
CLC_CCCTL0_LCOND);

    DL_TimerA_setCaptureCompareOutCtl(PWM_0_INST, DL_TIMER_CC_OCTL_INIT_VAL_LOW,
        DL_TIMER_CC_OCTL_INV_OUT_DISABLED, DL_TIMER_CC_OCTL_SRC_FUNCVAL,
        DL_TIMERA_CAPTURE_COMPARE_0_INDEX);

    DL_TimerA_setCaptureCompareOutCtl(PWM_0_INST, DL_TIMER_CC_OCTL_INIT_VAL_LOW,
        DL_TIMER_CC_OCTL_INV_OUT_DISABLED, DL_TIMER_CC_OCTL_SRC_ZERO,
        DL_TIMERA_CAPTURE_COMPARE_1_INDEX);//CC1 output will go high for 1 TIMCLK cycle when
Counter value is Zero, indicating a Zero Event

    DL_TimerA_setCaptCompUpdateMethod(PWM_0_INST, DL_TIMER_CC_UPDATE_METHOD_ZERO_EVT,
DL_TIMERA_CAPTURE_COMPARE_0_INDEX);//Update the CC value on the subsequent Zero Event

    DL_TimerA_setCaptureCompareValue(PWM_0_INST, 2000, DL_TIMER_CC_0_INDEX);

    DL_TimerA_enableClock(PWM_0_INST);

    DL_TimerA_setCCPDirection(PWM_0_INST , DL_TIMER_CC0_OUTPUT);
}
```

```
/* SDK API Used to Update Load and CC Values in Main Application */
 DL_TimerA_setLoadValue(PWM_0_INST, 10000);
 DL_TimerA_setCaptureCompareValue(PWM_0_INST, 2000, DL_TIMER_CC_0_INDEX);
```

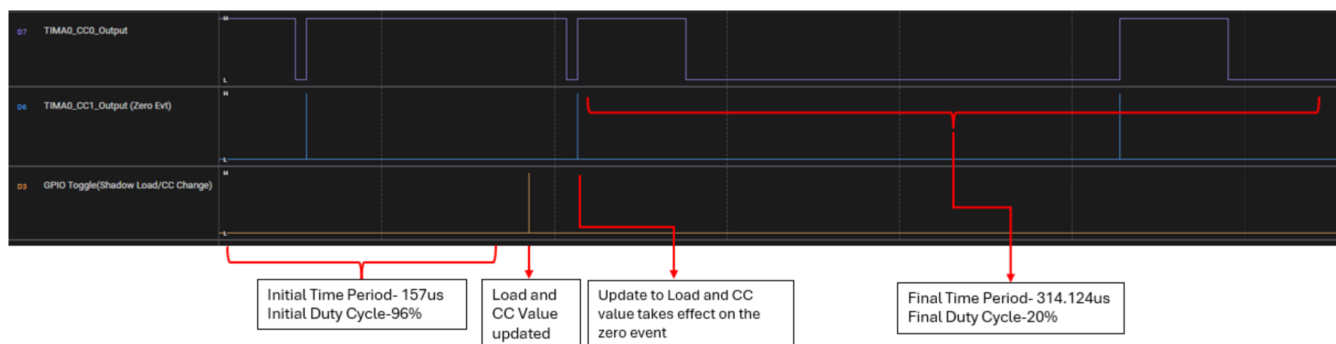

图 8-2. 显示 PWM 周期和占空比变化的波形

用户可根据应用要求配置 CC 和 CCACT 更新事件，如图 8-3 所示。

English Document: SDAA286
Copyright © 2026 Texas Instruments Incorporated

Bit Field	Value	Description/Comment
CCUPD / CCACTUPD	0	The value written to TIMx.CC register take effect immediately.
	1	The value written to the TIMx.CC register is stored in a shadow compare register and gets transferred to the TIMx.CC register in the TIMCLK cycle following a zero event (TIMx.CTR value equals 0).
	2	The value written to the TIMx.CC register is stored in a shadow compare register and gets transferred to the TIMx.CC register in the TIMCLK cycle following a compare (down) event (TIMx.CTR value equals TIMx.CC)
	3	The value written to the TIMx.CC register is stored in a shadow compare register and gets transferred to the TIMx.CC register in the TIMCLK cycle following a compare (up) event (TIMx.CTR value equals TIMx.CC)
	4	The value written to the TIMx.CC register is stored in a shadow compare register and gets transferred to the TIMx.CC register in the TIMCLK cycle following a zero or load event (TIMx.CTR value equals 0 or TIMx.CTR equals TIMx.LOAD). **Note: this update mechanism is defined for use only in up/down counting mode.**
	5	The value written to the TIMx.CC register is stored in a shadow compare register and gets transferred to the TIMx.CC register in the TIMCLK cycle following a zero event and the repeat count equaling zero (TIMx.CTR value equals 0 and TIMx.RC equals 0)
	6	The value written to the TIMx.CC register is stored in a shadow compare register, and gets transferred to the TIMx.CC register in the TIMCLK cycle following a trigger pulse. See Section 27.2.7.

图 8-3. 影子比较和操作更新行为

影子寄存器运行特性

- 默认状态：启用影子功能后，如果没有为 CC 或 LOAD 寄存器指定用户定义的初始值，则影子寄存器默认初始化为 0x0

备注

如果用户在启用影子 CC 功能之前设置 CC 值，且在启用影子 CC 之后未设置第二个 CC 值，则在发生下一个用户定义的事件后，CC 值将恢复为零并保持为零。

- 运行时更新逻辑：启用影子功能后进行的 CC 或 LOAD 值修改仍保持挂起状态，直到下一次发生用户定义的 EVENT 触发

备注

如果用户在启用影子 CC 功能后设置初始 CC 值，则在下一次发生用户定义的 EVENT 之前，CC 值将保持为零。

8.2 使用 DMA 生成任意信号

计时器模块可通过事件互联矩阵生成 DMA 请求。此功能允许用户通过调整计时器外设寄存器内容来动态修改计时器外设的波形输出。例如，它允许用户更新 TIMx.LOAD 寄存器内容以调整波形频率或更新 TIMx.CC 寄存器以调整信号占空比。

标准 PWM 生成通常依赖于分别在计时器的重载寄存器和捕获/比较寄存器中定义的固定频率和占空比参数。虽然这些寄存器可以通过软件干预手动更新，但这种方法会带来显著的 CPU 开销和潜在的时序不确定性。

本节演示了一种优化的方法，利用 DMA（直接存储器存取）功能与计时器外设结合以实现：

- 确定性波形生成
- 极少的 CPU 干预
- 精确时序控制
- 自动参数更新

该解决方案利用硬件触发的 DMA 传输自动更新频率和占空比参数，而不是依赖于每次计时器迭代时 CPU 驱动的更新。这种方法可确保：

1. 连续 PWM 周期之间精确的时序关系
2. 可预测且无抖动的波形生成

Copyright © 2026 Texas Instruments Incorporated

3. 高效利用系统资源
4. 在没有影子功能的情况下动态更新 PWM 占空比和周期。

本节详细介绍了实现策略，展示了计时器更新事件如何触发自动 DMA 传输以实现无缝寄存器更新，从而在保持精确波形控制的同时消除 CPU 开销。

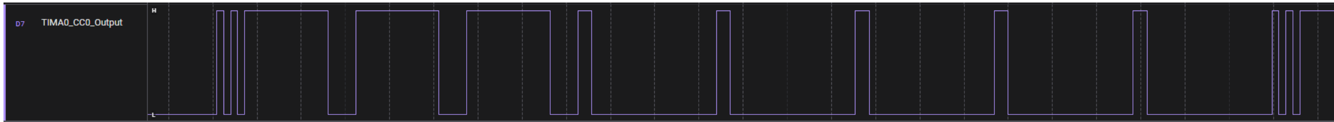

图 8-4. 使用 DMA 生成任意信号

应用流程如下所述：

- 将 TIMER 配置为事件发布者以发布事件来触发 DMA，使用 FPUB0 触发 DMA_CHAN0 以及使用 FPUB1 触发 DMA_CHAN1。
- DMA_CHAN0 将用于写入 LOAD 寄存器，而 DMA_CHAN1 将用于写入 CC 寄存器。
- 更改 LOAD 值将改变周期，而更改 CC 值将改变占空比。
- 将 CC 更新事件配置为零事件，这意味着 CC 值将在发生下一个零事件时更新。
- 在重复单次传输模式下将 DMA 配置为订阅者，传输宽度为 16 位。
- 将 DMA 源地址配置为递增，将目标地址配置为不变。
- 对于 DMA_CHAN0，将源地址配置为包含 LOAD 值的缓冲区，将目标地址配置为 TIMx.LOAD 寄存器地址。
- 对于 DMA_CHAN1，将源地址配置为包含 CC 值的缓冲区，将目标地址配置为 TIMx.CC 寄存器地址。
- 对于 CC1，OCTL.CCPO 已配置为 0x4。因此，当计数器值为零时，CC1 输出信号将在 1 个 TIMCLK 周期内切换并设置为高电平值，之后 CC1 输出将由硬件清除。此功能已用于检查相对于零事件的加载和 CC 更新时序。此功能也可用于各种其他应用中的调试目的。

```
/* Configuration Sequence for Timer */
static const DL_TimerA_ClockConfig gPWM_0ClockConfig = {
    .clockSel = DL_TIMER_CLOCK_BUSCLK,
    .divideRatio = DL_TIMER_CLOCK_DIVIDE_1,
    .prescale = 0U
};

static const DL_TimerA_PWMConfig gPWM_0Config = {
    .pwmMode = DL_TIMER_PWM_MODE_EDGE_ALIGN_UP,
    .period = 5000,
    .isTimerWithFourCC = false,
    .startTimer = DL_TIMER_STOP,
};

SYSCONFIG_WEAK void SYSCFG_DL_PWM_0_init(void) {

    DL_TimerA_setClockConfig(
        PWM_0_INST, (DL_TimerA_ClockConfig *) &gPWM_0ClockConfig);

    DL_TimerA_initPWMMode(
        PWM_0_INST, (DL_TimerA_PWMConfig *) &gPWM_0Config);

    // Set Counter control to the smallest CC index being used

DL_TimerA_setCounterControl(PWM_0_INST,DL_TIMER_CZC_CCCTL0_ZCOND,DL_TIMER_CAC_CCCTL0_ACOND,DL_TIMER_
CLC_CCCTL0_LCOND);

    DL_TimerA_setCaptureCompareOutCtl(PWM_0_INST, DL_TIMER_CC_OCTL_INIT_VAL_LOW,
        DL_TIMER_CC_OCTL_INV_OUT_DISABLED, DL_TIMER_CC_OCTL_SRC_FUNCVAL,
        DL_TIMERA_CAPTURE_COMPARE_0_INDEX);
    DL_TimerA_setCaptureCompareOutCtl(PWM_0_INST, DL_TIMER_CC_OCTL_INIT_VAL_LOW,
        DL_TIMER_CC_OCTL_INV_OUT_DISABLED, DL_TIMER_CC_OCTL_SRC_ZERO,
        DL_TIMERA_CAPTURE_COMPARE_1_INDEX);

    DL_TimerA_setCaptCompUpdateMethod(PWM_0_INST, DL_TIMER_CC_UPDATE_METHOD_ZERO_EVT,
DL_TIMERA_CAPTURE_COMPARE_0_INDEX);//Update CC on Zero Event

    DL_TimerA_setCaptureCompareValue(PWM_0_INST, 2000, DL_TIMER_CC_0_INDEX);
    DL_TimerA_setCaptureCompareValue(PWM_0_INST, 4000, DL_TIMER_CC_1_INDEX);
```


Copyright © 2026 Texas Instruments Incorporated

```
    DL_TimerA_enableClock(PWM_0_INST);

    DL_Timer_enableEvent(PWM_0_INST, DL_TIMER_EVENT_ROUTE_1, DL_TIMERA_INTERRUPT_ZERO_EVENT);//Zero
Event triggers the DMA CHAN0
    DL_Timer_setPublisherChanID(PWM_0_INST, DL_TIMER_PUBLISHER_INDEX_0, 1);//Timer publishing event
to DMA to carry out Load Value update

    DL_Timer_enableEvent(PWM_0_INST, DL_TIMER_EVENT_ROUTE_2, DL_TIMERA_INTERRUPT_ZERO_EVENT);//Zero
Event triggers the DMA CHAN1
    DL_Timer_setPublisherChanID(PWM_0_INST, DL_TIMER_PUBLISHER_INDEX_1, 12);//Timer publishing
event to DMA to carry out CC value update

    DL_TimerA_setCCPDirection(PWM_0_INST , DL_TIMER_CC0_OUTPUT|DL_TIMER_CC1_OUTPUT);
}
```

```
/* Configuration Sequence for DMA */
static const DL_DMA_Config gDMA_CH0Config = {
    .transferMode   = DL_DMA_FULL_CH_REPEAT_SINGLE_TRANSFER_MODE,
    .extendedMode   = DL_DMA_NORMAL_MODE,
    .destIncrement  = DL_DMA_ADDR_UNCHANGED,
    .srcIncrement   = DL_DMA_ADDR_INCREMENT,
    .destWidth      = DL_DMA_WIDTH_HALF_WORD,
    .srcWidth       = DL_DMA_WIDTH_HALF_WORD,
    .trigger        = DMA_GENERIC_SUB0_TRIG,
    .triggerType    = DL_DMA_TRIGGER_TYPE_EXTERNAL,
};
static const DL_DMA_Config gDMA_CH1Config = {
    .transferMode   = DL_DMA_FULL_CH_REPEAT_SINGLE_TRANSFER_MODE,
    .extendedMode   = DL_DMA_NORMAL_MODE,
    .destIncrement  = DL_DMA_ADDR_UNCHANGED,
    .srcIncrement   = DL_DMA_ADDR_INCREMENT,
    .destWidth      = DL_DMA_WIDTH_HALF_WORD,
    .srcWidth       = DL_DMA_WIDTH_HALF_WORD,
    .trigger        = DMA_GENERIC_SUB1_TRIG,
    .triggerType    = DL_DMA_TRIGGER_TYPE_EXTERNAL,
};

void SYSCFG_DL_DMA_CH0_init(void)
{
    DL_DMA_initChannel(DMA, DMA_CH0_CHAN_ID , (DL_DMA_Config *) &gDMA_CH0Config);
    DL_DMA_initChannel(DMA, DMA_CH1_CHAN_ID , (DL_DMA_Config *) &gDMA_CH1Config);
    DL_DMA_clearInterruptStatus(DMA, DL_DMA_INTERRUPT_CHANNEL0);
        DL_DMA_enableInterrupt(DMA, DL_DMA_INTERRUPT_CHANNEL0);
        DL_DMA_setSubscriberChanID(DMA, DL_DMA_SUBSCRIBER_INDEX_0, 0x01);//DMA subscribing to Timer
event
        DL_DMA_setSubscriberChanID(DMA, DL_DMA_SUBSCRIBER_INDEX_1, 12);//DMA subscribing to Timer
event
}

void SYSCFG_DL_DMA_init(void){
    SYSCFG_DL_DMA_CH0_init();
}
```

```
/* Application Code for Timer and DMA */
uint16_t Load_vals[10] = {
    0x3E8,    // 1000
    0x3E8,    // 1000
    0x1F40, // 8000
    0x1F40, // 8000
    0x1F40, // 8000
    0x2710,   // 10000
    0x2710,   // 10000
    0x2710,   // 10000
    0x2710,   // 10000
    0x2710,   // 10000


};
uint16_t CC_vals[10] = { 0x1F4,    // 500
                         0x1F4,    // 500
                         0x1770,   // 6000
                         0x1770,   // 6000
                         0x1770,   // 6000
                         0x3E8,    // 1000
                         0x3E8,    // 1000
                         0x3E8,    // 1000
```


Copyright © 2026 Texas Instruments Incorporated

```
                         0x3E8,    // 1000
                         0x3E8,    // 1000
};
int main(void)
{
    SYSCFG_DL_init();

    DL_DMA_setDestAddr(DMA, DMA_CH0_CHAN_ID, (uint32_t)&PWM_0_INST->COUNTERREGS.LOAD);
    DL_DMA_setSrcAddr(DMA, DMA_CH0_CHAN_ID, (uint32_t)&Load_vals );
    DL_DMA_setTransferSize( DMA, DMA_CH0_CHAN_ID,10);

    DL_DMA_setDestAddr(DMA, DMA_CH1_CHAN_ID, (uint32_t)&PWM_0_INST->COUNTERREGS.CC_01[0]);
    DL_DMA_setSrcAddr(DMA, DMA_CH1_CHAN_ID, (uint32_t)&CC_vals );
    DL_DMA_setTransferSize( DMA, DMA_CH1_CHAN_ID,10);

    DL_DMA_enableChannel(DMA, DMA_CH1_CHAN_ID);
    DL_DMA_enableChannel(DMA, DMA_CH0_CHAN_ID);

    DL_TimerA_startCounter(PWM_0_INST);

    while (1) {
        ;
    }
}
```


Copyright © 2026 Texas Instruments Incorporated

9 总结

本应用手册探讨了 MSPM0 微控制器中的高级计时器 (TIMA) 技术，重点介绍了 TIMA 模块的多用途功能。本文档介绍了 PWM 空闲低电平配置、相移 PWM 生成以及用于 UART 仿真的位操作。本文档演示基于反馈的 PWM 生成、具有可配置延迟的同步计时器启动以及硬件触发的计时器停止控制。主要特性包括用于动态 PWM 更新的影子寄存器和基于 DMA 的任意信号生成。含有代码片段的实际示例说明了现代嵌入式系统的实现策略。

Copyright © 2026 Texas Instruments Incorporated

10 参考资料

- 德州仪器 (TI)，*MSPM0 中的高级计时器技术示例项目*，常见问题解答。
- 德州仪器 (TI)，*MSPM0Gx51x 具有 CAN-FD 接口的混合信号微控制器*，数据表。
- 德州仪器 (TI)，*MSPM0 L 系列 80MHz 微控制器* 技术参考手册。

English Document: SDAA286
Copyright © 2026 Texas Instruments Incorporated

重要通知和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、与某特定用途的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他安全、安保法规或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的相关应用。严禁以其他方式对这些资源进行复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。对于因您对这些资源的使用而对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，您将全额赔偿，TI 对此概不负责。

TI 提供的产品受 TI 销售条款)、TI 通用质量指南 或 ti.com 上其他适用条款或 TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。 除非德州仪器 (TI) 明确将某产品指定为定制产品或客户特定产品，否则其产品均为按确定价格收入目录的标准通用器件。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

版权所有 © 2026，德州仪器 (TI) 公司

最后更新日期：2025 年 10 月