Application Note

MSPM0 の高度タイマ技術

Gaurang Gupta, Sal Ye, Baibhav Tripathy, Luke Ledbetter

概要

MSPM0 デバイスの高度タイマ (TIMA) ペリフェラルは、最新の組み込みアプリケーションの要求を満たすように設計されています。このアプリケーション ノートでは、高度タイマ モジュールの多角的な機能について説明します。ソフトウェアベースのビットバンギング通信プロトコルから、高度なフィードバック制御 PWM システムまで、さまざまなシナリオでアプリケーションを詳しく説明します。開発者は、入力キャプチャ、PWM 出力、アナログ ペリフェラル (ADC やコンパレータなど) との同期などといった機能を活用することで、応答性が高く効率的な制御ループを実装できます。本書では、高度タイマ アーキテクチャを活用して、高度な組み込みソリューション、適応型電力レギュレーション、リアルタイム信号処理を容易に開発できる方法について、詳細な例とユースケース分析を通じて説明します。記載されている詳細情報は、エンジニアがタイマ ペリフェラルの潜在能力を最大限に活用して革新的なアプリケーション設計を実現するのに役立てることを目的としています。このアプリケーション ノートで詳細に説明されているプロジェクト資料は、「MSPM0 の高度タイマ技術」からダウンロードできます。

目次

商標

1 概要

本アプリケーション ノートでは、MSPM0 マイコンの高度なタイマ (TIMA) モジュールの多様な機能について説明し、最新の組み込みアプリケーションに不可欠な高度なタイミング機能と制御機能を提示します。本書では、モーター制御から通信プロトコルのエミュレーションに至るまで、タイマ ベースのさまざまなソリューションに対する包括的な実装方法を説明します。

このガイドで詳しく説明している主なアプリケーションには、Idle-Low 状態を維持するモーター重視の PWM 設定や、高精度の位相制御による同期出力を特徴とする洗練された PWM 生成技術が含まれています。このノートは、位相シフト PWM を実現するための 2 種類のアプローチを紹介します。具体的には、位相負荷機能の活用と、セカンダリ キャプチャ比較チャネルの活用です。カスタムの通信プロトコルを必要とするアプリケーション向けに、ビットバンギングの実装が検討されており、特に TIMA リソースを使用する UART トランスミッタとレシーバ エミュレーションに重点を置いています。

さらに、本書では、複数のタイマ インスタンス間の調整済み遅延開始、閉ループ システム向けのフィードバックベースの PWM 生成、ハードウェア トリガによるタイマ制御など、タイミング制御に関連する重要な要素について説明しています。最後に、デューティ サイクルと周期の値をリアルタイムで調整できるようにする PWM パラメータの動的な変更方法を説明しています。各実装には、MSPM0 デバイスに特化した実例、タイミング図、最適化された構成設定が含まれています。

2 Idle-Low 状態:PWM 出力チャネル Low 状態構成

TIMA は、すべてのキャプチャ/比較 (CC) チャネルに補完チャネルがあるメカニズムを備えています。この補完チャネルは、メインの CC チャネルの反転である出力を生成できます。モーター アプリケーションでは、最初に両方の出力チャネルを low にする必要があります。

CTRCTL.EN ビットが 0 である場合、CC チャネル出力は OCTL.CCPIV に依存します。CC チャネルと補完チャネルの両方が low で維持される場合、解決策は、補完チャネルの IOMUX を無効のままにしておき、以下の場合に有効にすることです。

- エッジアライン モードでは、CTRCTL.EN が有効化されるまで、補完 CC チャネルに対して IOMUX を無効のままにしておきます。
- センターアラインモードでは、CTRCTL.EN が有効化され、CC チャネルが最初の CC イベントに達するまで、補完 CC チャネルに対して IOMUX を無効のままにしておきます。

```
/* 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();
    }
}
```

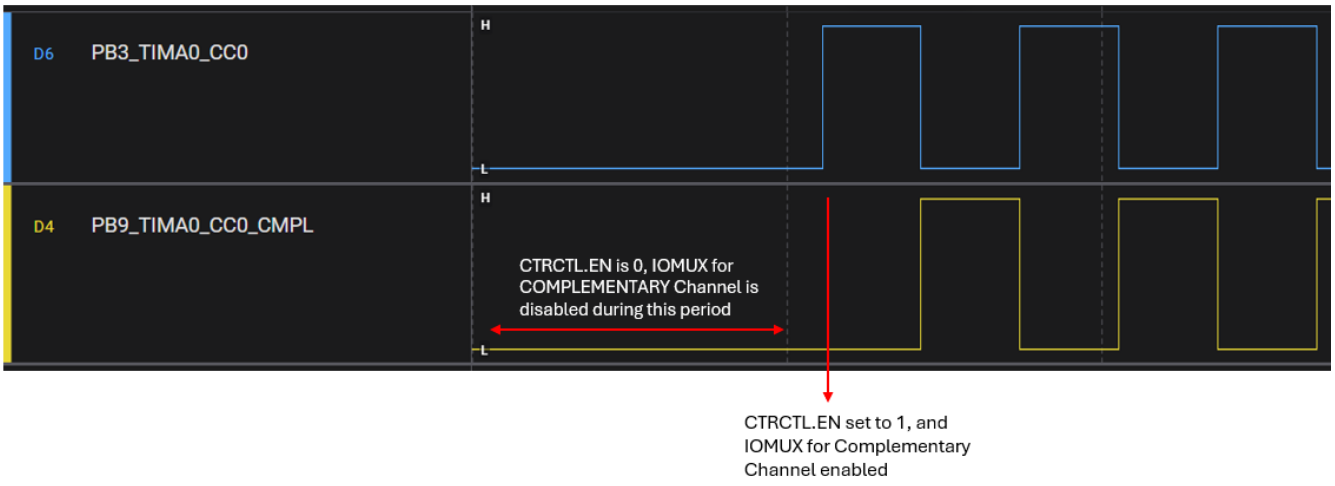

図 2-1. CC 出力を Idle-Low に設定したときの PWM 波形

また、TIMA は、ソフトウェアベースの設定を使用して出力をオーバーライドする機能もサポートしています。CCACT の SWFRCACT_CMPL (補完チャネルのソフトウェア強制) フィールドを使用すれば、CC 出力を LOW にオーバーライドできます。アプリケーションに従って SWFRCACT_CMPL フィールドをクリアするには、外部イベントまたは割り込みサービス ルーチンを使用します。

注

SWFRCACT_CMPL は、CTRCTL.EN が設定された後で有効になります。

3 非対称型 PWM：位相シフト制御によるデュアル同期 PWM 生成

TIMA は、制御された位相シフトにより、2 つの同期されたセンターアライン PWM 信号を生成して、以下を実現できます。

- モーター制御によるトルク リップルの最小化および低速動作での円滑性の向上
- デジタル電源 DC/DC コンバータによるスイッチング損失と EMI の低減
- 位相角度ベースの高精度電力制御を可能にする AC 位相制御回路

3.1 位相負荷機能の使用

非対称型の PWM 信号を生成する場合は TIMA の位相負荷機能を使用できます。この方法では 2 つのタイマ インスタンスを使用します。そのうちの 1 つは TIMA でなければなりません (位相負荷機能をサポートしているのは TIMA のみ)。両方の TIMER インスタンスは、クロストリガ機能を使用して同期されます。タイマがクロストリガされたとき、位相負荷機能が有効化されている場合には位相負荷が発動して、タイマが位相負荷値からカウントを開始します。この設定可能な位相負荷値により、2 つの PWM 間で制御された位相シフトが可能になります。設定については以下のセクションで説明します。

3.1.1 プライマリ タイマ (メイン タイマ) の設定

プライマリ タイマは、クロストリガ信号を生成して他のタイマを同期させるものです。

- TIMx.CTTRIGCTL.CTEN ビットを設定することで、クロストリガ出力機能を有効にします。
- 適切なトリガ ソースを選択することで、クロストリガ生成イベントを設定します。
  - ソフトウェアベースのクロストリガの場合、TIMx.CTTRIG.TRIG ビットを設定セットします。
  - ハードウェアベースのクロストリガの場合、TIMx.CTTRIGCTL.EVTCTTRIGSEL を使ってトリガのソースを選択し、TIMx.CTTRIGCTL.EVTCTEN を設定してハードウェア トリガを有効化します。
- タイマはセルフトリガ機能も備えています。

```
/* 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);

    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 セカンダリ タイマの設定

セカンダリ タイマはクロストリガ信号を受信し、その動作をプライマリ タイマと同期します。

- クロストリガ マッピング (デバイス固有のデータ シートを参照) と、使用するプライマリ タイマとセカンダリ タイマのインスタンスに従って、TIMx.TSEL.ETSEL フィールド設定をします。
- TIMA.TSEL.TE ビットを 1 に設定することで、入力トリガ機能を有効にします。
- TIMAx.IFCTL_01.ISEL ビットを 3 に設定して、クロストリガを入力ソースとして選択します。
- カウンタがダウンカウント モードでカウントしている場合、立ち上がりエッジを使用してロード イベントをトリガするために CCCTL.LCOND を 1 に設定します。これにより、クロストリガの受信時にカウンタ値がロード値として設定され、カウンタはカウント ダウンを開始します。アップカウント モードの場合も同様に CCCTL.ZCOND を 1 に設定します。
- セルフクロストリガ シナリオとアップダウン カウント モードの場合については、たとえば TIMx.CTTRIGCTL.EVTCTTRIGSEL が 0 に設定されている場合は、イベント CCCTL.ZCOND を 1 に設定する必要があります。
- TIMx.CTRCTL.EN ビットは、LCOND 条件または ZCOND 条件が満たされた結果として設定され、カウンタ値はそれぞれ、ロード値またはゼロ値に変化します。

注

クロストリガ モードで動作するタイマの周波数は、同じ TIMCLK である必要があります。生成されたクロストリガ パルス幅が、トリガを受信するタイマの 1 TIMCLK サイクル未満である場合、クロストリガによってカウンタが有効化されません。

```
/* 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(
        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 クロストリガ機能の実装

表 3-1 に、2 つのクロストリガ生成方式を示します。

表 3-1. クロストリガ

S 番号	ソフトウェア クロストリガ	ハードウェア クロストリガ
1	ソフトウェア クロストリガは、基本的には、アプリケーションコードにより TIMx.CTTRIG.TRIG ビットに 1 が書き込まれると生成されます。	ハードウェア クロストリガは、設定された各イベントにおいて、TIMx.CTTRIGCTL.EVTCTTRIGSEL フィールドで設定したイベントに基づき、(初期設定後における) ソフトウェアの関与なしに、ハードウェアによって生成されます。たとえば、EVTCTTRIGSEL がゼロ イベントに設定されている場合、プライマリ タイマのゼロ イベントごとにハードウェア クロストリガが生成されます。
2	ソフトウェア クロストリガでは、アプリケーションでプライマリ タイマとセカンダリ タイマの同期が必要である場合にのみ、その同期が実行されます。 **注** Load/CC 値が変更された場合、ソフトウェアによる同期が無効になることがあります。	ハードウェア クロストリガ メカニズムでは、プライマリ タイマとセカンダリ タイマの同期が繰り返し行われます。したがって、プライマリ タイマの Load/CC 値が変化した場合でも、ハードウェア クロストリガはすべてのイベントで生成されるため、セカンダリ タイマはプライマリ タイマと同期されます。

ソフトウェアベースのクロストリガ信号を生成するメカニズム

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

ゼロ イベント時にハードウェアベースのクロストリガを生成する設定

```
    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
```

位相シフト PWM 生成

図 3-1 に、位相シフト PWM 生成のテスト波形を示します。

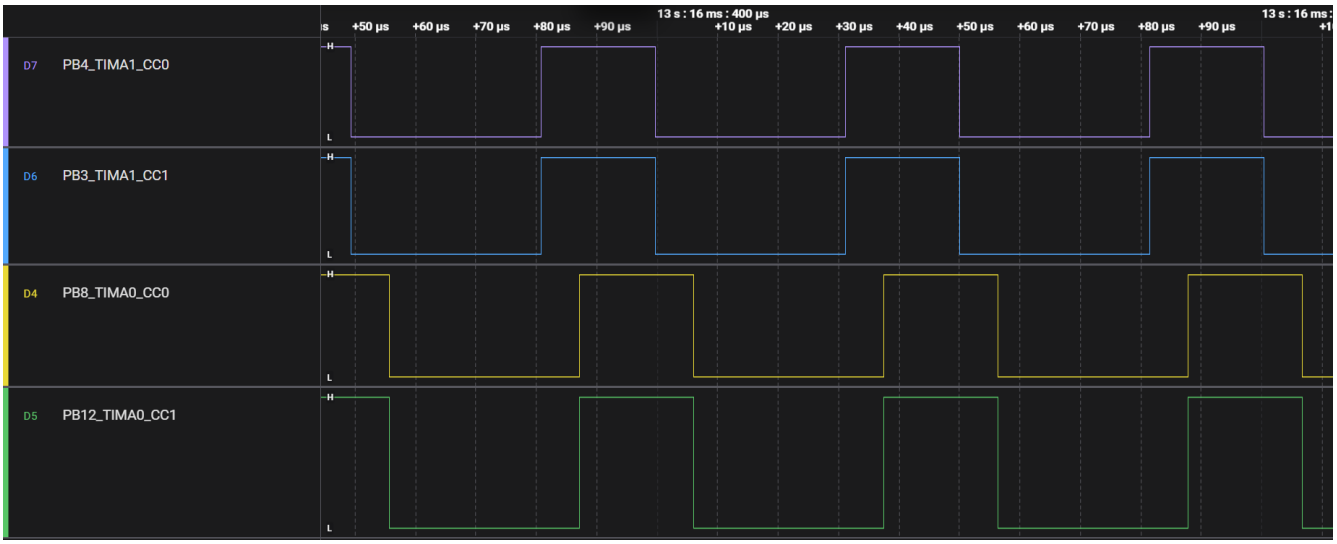

図 3-1. クロストリガと位相負荷を使用した位相シフト PWM 生成

3.2 セカンダリ キャプチャ比較チャネルの使用

TIMA0 と 4 つの外部 CC チャネルおよび 2 つの内部 CC チャネルを組み合わせて、3 相シフト PWM を個別に生成できます。これは、内部 CC チャネルのセカンダリ CC イベントを利用することで実現できます。GPTIMER は、カウント モード (ダウン、アップ、アップダウン) に応じて CCCTL.CC2SELD または CCCTL.CC2SELU を設定することで、セカンダリ CC チャネルを選択する機能を備えています。

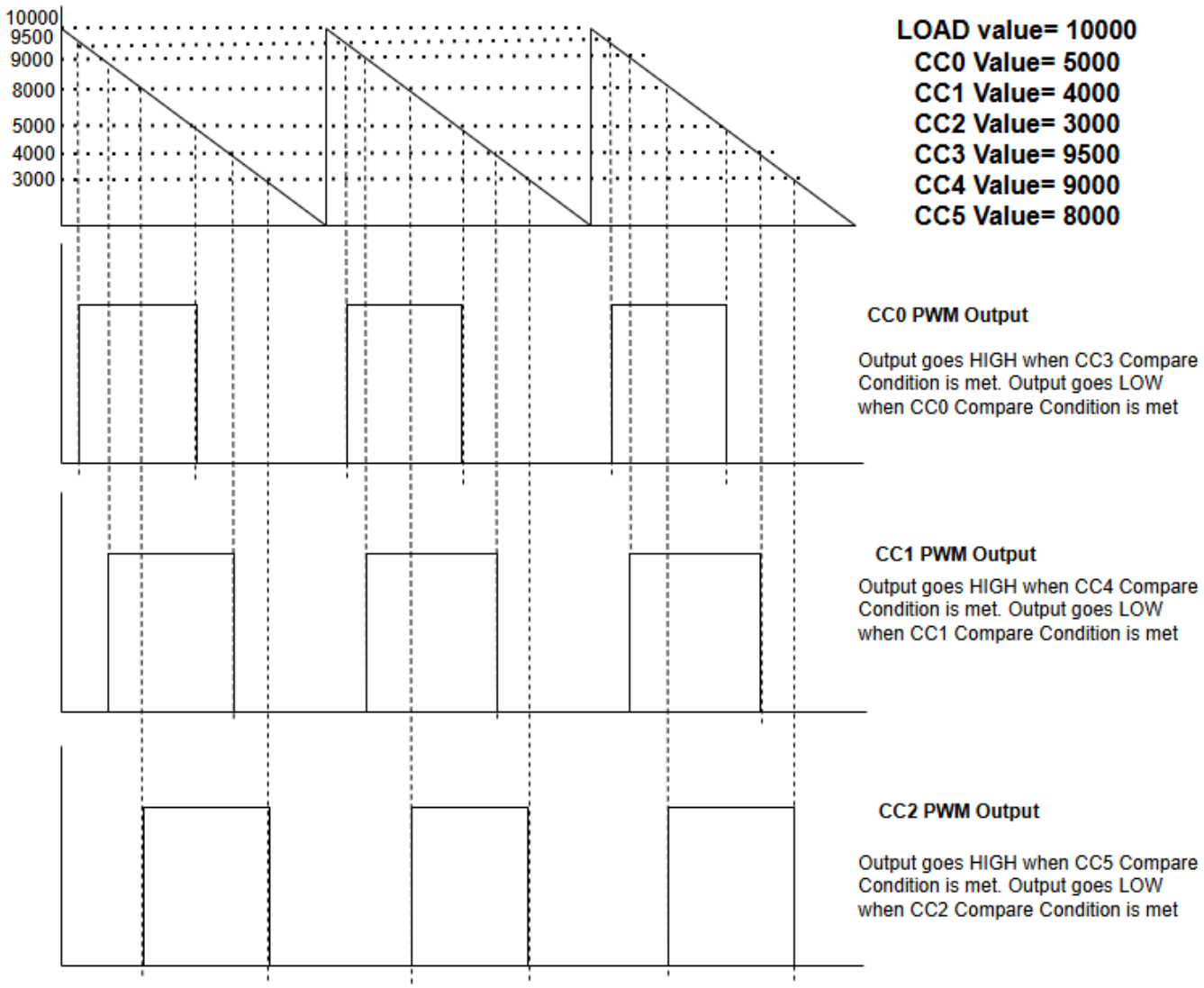

図 3-2. セカンダリ チャネル アプローチと TIMA0 を使用した非対称 PWM 生成の図表示

選択したセカンダリ チャネルに基づいて、カウント モード (ダウン、アップ、ダウン) に応じて CCACT.CC2UACT または CCACT.CC2DACT を設定することで出力を生成できます。

たとえば、カウンタ ロード値が 10000 で、カウンタがダウン カウント モードで CC4 値が 9000 の場合を考えてみましょうです。

CCCTL[1].CC2SELD を 0x4 に設定すると、CC4 がメイン チャネル CC1 のセカンダリ CC チャネルとして選択されます。

CCACT[1].CC2DACT を 0x1 に設定した場合は、カウンタが現在の CC4 値である 9000 に達すると、CC 出力値が HIGH に設定されます。CC4 は、CC1 の**擬似**ロード値として機能します。

CCACT[1].CC2DACT により、カウンタが 9000 に達するたびに CC1 の出力が HIGH になるはずです。カウンタが 4000 (CC1 値) に達すると、PWM 出力は LOW になります。

3 相シフト PWM を生成する際にこの手法を取り入れると便利です。クロストリガ機能を使用して、複数のタイマ インスタンスがある同期シフト PWM 出力チャネルを生成します。

次の**設定シーケンス**に従って、セカンダリ CC チャネルを設定します。

- TIMA.CTRCTL で、以下を目的とするカウンタ制御を適切に設定します。
  - カウント モード (CM) とイネーブル後のカウント値 (CVAE)。
  - CLC、CZC、CAC を使用して、カウンタのゼロ化、前進、またはロードを制御する条件を指定する
- PWM 周期を設定するために、TIMA.LOAD の値を設定する
- 比較モードの場合、TIMA.CCCTL_xy[0/1].COC = 0 に設定する
- CCPD レジスタの各ビットを設定することにより、CCP を CC ブロックの出力として構成します。
- TIMA.OCTL_xy[0/1] で、信号ジェネレータ出力を選択するために CCPO = 0 を設定します。
- 対応するカウンタ n に対して ODIS.C0CCPn を 0 に設定することで、対応する CCP 出力を有効にします。
- CCPOINV ビットを使用して信号の極性を設定し、カウンタが無効化されている時の CCP 出力状態を指定するために CCPIV を設定します。
- CC0 の CCCTL.CC2SELD フィールドを 3 に設定すると、CC3 がセカンダリ CC ブロックとして使用されます。
- CC1 の CCCTL.CC2SELD フィールドを 4 に設定すると、CC4 がセカンダリ CC ブロックとして使用されます。
- CC2 の CCCTL.CC2SELD フィールドを 5 に設定すると、CC5 がセカンダリ CC ブロックとして使用されます。
- 必要な位相シフトに応じて CC3 を設定します。たとえば、500 TIMCLK サイクルにわたって位相シフトが必要な場合は、CC3 = ロード値 - 500 = 9500 を設定します。
- 必要な位相シフトに応じて CC4 を設定します。たとえば、1000 TIMCLK サイクルにわたって位相シフトが必要な場合は、CC4 = ロード値 - 1000 = 9000 を設定します。
- 必要な位相シフトに応じて CC5 を設定します。たとえば、2000 TIMCLK サイクルにわたって位相シフトが必要な場合は、CC5 = ロード値 - 2000 = 8000 を設定します。
- CC3、CC4、CC5 をそれぞれ参照値とし、必要なデューティ サイクルに基づいて CC0、CC1、CC2 を設定します。
- CC3、CC4、CC5 の値は、それぞれ CC0、CC1、CC2 の**疑似**ロード値として機能します。
- カウンタが CC3、CC4、CC5 のそれぞれの値に達するたびに出力が high に設定されるようにするために、CC0、CC1、CC2 の CCACT.CC2DACT を 0x1 に設定します。
- カウンタが CC0、CC1、CC2 のそれぞれの値に達するたびに出力が low にクリア設定されるようにするために、CC0、CC1、CC2 の CCACT.CDACT を 0x2 に設定します。
- TIMA.CTRCTL.EN = 1 に設定してカウンタを有効にします。

```
/* 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
    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);
}
```

図 3-3 に、シフトされた PWM 出力の波形を示します。周期 / デューティ サイクルを動的に変更するには、セクション 8 を参照してください。

図 3-3. TIMA0 セカンダリ イベントを使用した位相シフト PWM の生成

4 ビットバンギング エミュレーション：ソフトウェアベースの通信プロトコルの実装

TIMA は、UART (Universal Asynchronous Receiver Transmitter) などの標準的な通信プロトコルのエミュレートに使用できます。TIMA は、高度なセキュリティ暗号化を必要とする非標準プロトコルやカスタム プロトコルの実装も可能にします。開発者は、TIMA を使用して、クロック ラインやデータ ラインの出力トグルの正確なスケジュール設定、読み取り動作のための入力エッジのキャプチャ、イベントの比較を行い、一貫性のあるボー レートやクロック サイクルを生成できます。

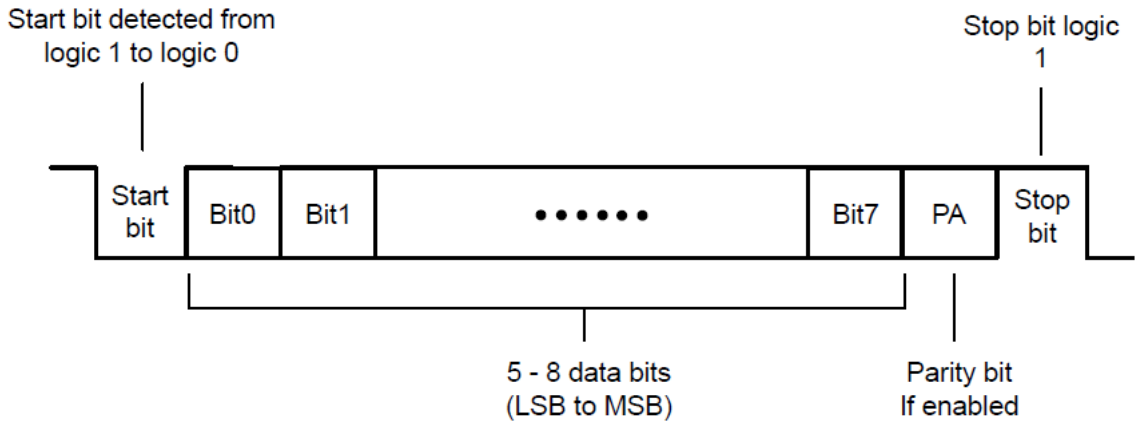

図 4-1. UART プロトコル標準フレーム

4.1 TIMA を使用した UART Rx のエミュレーション

TIMA は、UART プロトコルをエミュレートするために設定すべき必要なすべての機能を搭載しています。図 4-2 に、TIMA を UART Rx として使用する場合のフローチャートを示します。

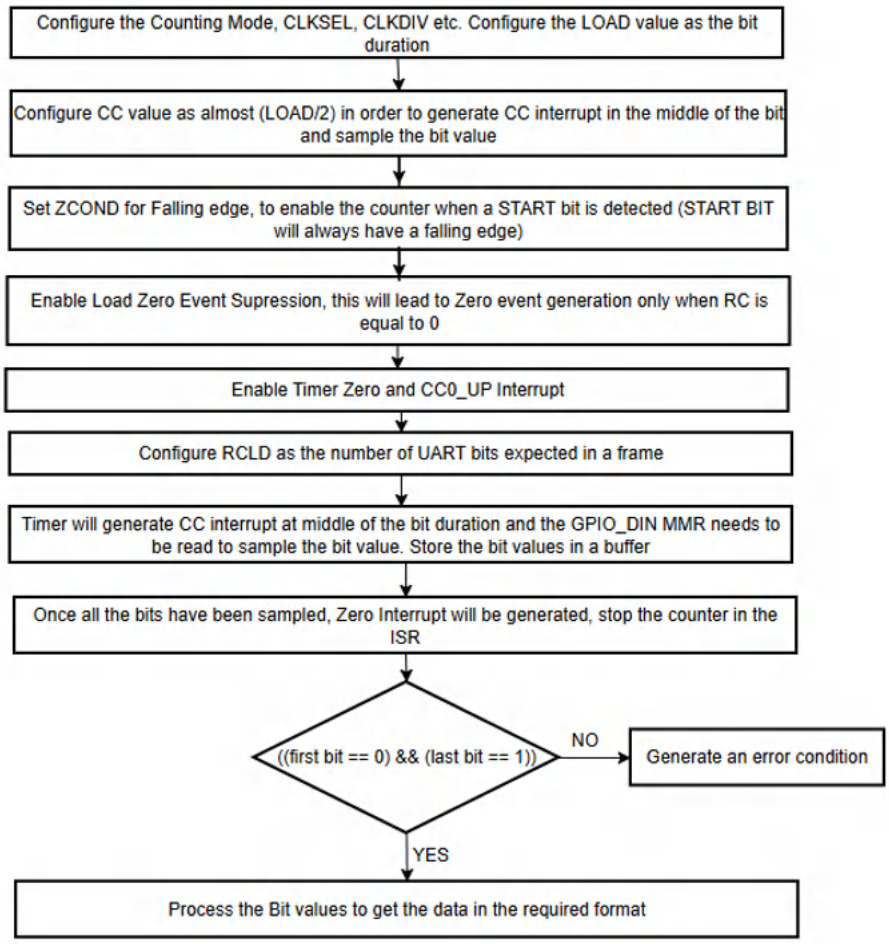

図 4-2. UART Rx を示すフローチャート

設定シーケンスを以下に示します。

- アップカウント モードで TIMA を設定し、立ち下がりエッジ検出に CCCTL.ZCOND を設定します。これは、UART ではフレームの開始が立ち下がりエッジで示されるためです。
- ボー レートに応じて、TIMER の Load 値をビット時間として設定します。
- RCLD レジスタを設定し、n 回 (ここで、n は UART フレームで想定されるビット数) の反復モードで実行するようにタイマを設定します。ビットの中間で CC イベントを生成し、対応するビットをサンプリングするために、CC 値を (LOAD 値 / 2 または LOAD 値 / 2 +1) に設定します。
- ロードとゼロ イベント抑制を有効にします。これで RC がゼロになるまでゼロ イベントは生成されません。すべてのビットがサンプリングされている場合以外、ゼロ イベントは必要ありません。
- タイマの割り込みサービス ルーチンにおいて、CC イベントが GPIO_DIN レジスタを読み取った場合、信号の入力値が出ます。
- タイマの割り込みサービス ルーチンにおいて、ゼロ イベントの場合には、カウンタが停止します。

```
/* 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,
};

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
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 は入力フィルタリング メカニズムもサポートしています。この機能を利用すれば、不要なグリッチが START ビットとして誤ってサンプリングされるのことを防止できます。

注

1. サンプリングが正しい時間に行われるように、まずは ISR で GPIO 値の読み取りを優先させます。
2. TIMER ISR を最優先の割り込みとして設定してください。そうしないと、タイム ポイントのサンプリングが失敗する可能性があります。
3. CLK、CLKDIV、PRESCALER の複数の組み合わせを使用すればボー レートを実現できます。タイマ機能クロックを低くすると、キャプチャ分解能が低下する可能性があります。

このアプリケーションは、TIMA の代わりに TIMG を使用して実装することもできます。ただし、考慮すべき重要な制限事項があります。TIMG はリピート カウンタ機能をサポートしていません。したがって、TIMG を使用して実装する場合、同じ機能を実現するためには、ISR 内のソフトウェアを介してカウンタを手動でリロードする必要があります。

注

DMA を使用してアプリケーションを実装すれば、アプリケーションのスループットを最適化して、高いボー レートと同時に CPU の帯域幅消費の低減を実現できます。この設定では、タイマ イベントによってトリガされたときに GPIO_DIN MMR を読み取るように DMA を設定できるため、CPU の介入は不要です。GPIO_DIN MMR に対する DMA アクセス性を確認するには、デバイス固有のデータ シートを参照してください。

4.2 TIMA を使用した UART Tx のエミュレーション

TIMA は、UART トランスミッタ エミュレーション向けの堅牢なソリューションであり、出力生成機能、シャドウ CCACT 機能、および反復カウンタ メカニズムを活用します。DMA 転送を組み込むことで、この実装のパフォーマンスを大幅に向上させることができ、システム リソースを最適化するエンドツーエンドのソリューションが生まれます。DMA がデータ移動を処理することで、ビット タイミングの高精度とデータ転送の高信頼性を確保しながら、CPU の介入を最小限に抑えます。

この統合タイマ DMA アプローチにより、割り込み優先度への依存性を排除し、タイミング精度の一貫性を確保します。このソリューションは、確定的な動作を保証できるので、ビット バン エミュレーション用途に最適です。図 4-3 に、TIMA を UART Tx として使用する場合のフローチャートを示します。

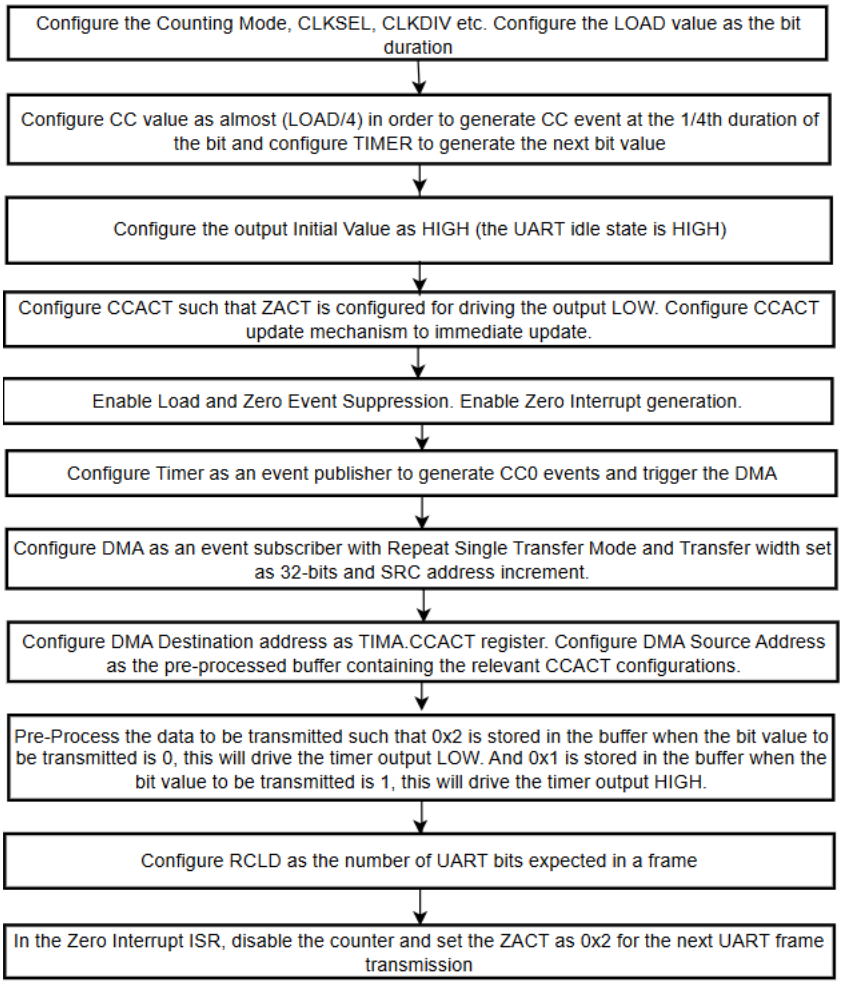

図 4-3. UART Tx を実証する示すフローチャート

設定シーケンスについて以下で詳細に説明します。

- カウント モード、CLKSEL、CLKDIV などを設定します。ビット持続時間に従ってロード値を設定します。
- CC 値を「LOAD/4」または「LOAD/4+1」に設定します。これにより、総ビット持続期間の 1/4 で CC イベントが生成されます。このイベントは、次のビット値が送信されるようにタイマを設定できるようにする場合に必要です。
- UART アイドル状態は HIGH であるため、出力初期値 OCTL.CCPIV を HIGH に設定します。
- 出力が LOW になるように ZACT が設定されるように、CCACT を設定します。これは、カウンタが有効であるときに LOW START ビットが生成されるためです。
- CCACT アップデート メカニズムを即時アップデートに設定します。
- 「ロード」と「ゼロ イベント抑制」を有効にすると、RC = 0 の場合にのみゼロ イベントを生成できます。
- 「タイマ ゼロ割り込み」を有効化します。
- TIMA は、イベントを生成し、イベント ファブリック経由で DMA などの他のペリフェラルをトリガできます。CC0 イベントの生成時に DMA がトリガされてデータ転送が開始されるように、TIMA をイベント パブリッシャとして設定する必要があります。
- DMA が TIMA イベントごとにデータ転送を実行できるように、DMA をイベント サブスクライバとして設定する必要があります。
- リピート シングル転送モードで、DMA の 32 ビット幅と送信元アドレス インクリメントを設定します
- DMA 宛先アドレスを TIMA.CCACT レジスタとして設定します。各 CC0 イベントにおいて、次のビット値を送信するために CCACT レジスタを更新する必要があります。この更新メカニズムは DMA によって実行されます。

- 送信されるビット値が 0 のときに 0x2 がバッファに格納されるように、送信されるデータを前処理します。これにより、タイマ出力が LOW になります。また、送信されるビット値が 1 のときには 0x1 がバッファに格納されます。これにより、タイマ出力が HIGH になります。
- 関連する CCACT 設定を含む前処理済みバッファとして DMA 送信元アドレスを設定します
- フレーム内で予測される UART ビット数として RCLD を設定します
- データ送信時にカウンタを有効にします。
- ゼロ割り込み ISR で、カウンタを無効にし、次回の UART フレーム送信用に ZACT を 0x2 に設定します。

```
/* 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
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;
    }
}
```

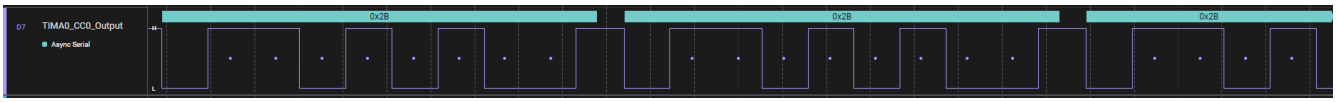

図 4-4. TIMA を使用した 0x2B 送信 (UART フォーマット)

注

上記のアプリケーションでは、TIMA の代わりに TIMG を使用する代替実装が可能です。ただし、TIMG はリピート カウンタ機能をサポートしていません。したがって、TIMG を使用する場合、ISR 内のソフトウェア介入を通じてカウンタを手動で再ロードする必要があります。

5 フィードバックベースの PWM 生成

フィードバックベースのパルス幅変調 (PWM) 信号生成機能を使用すると、システムのフィードバックに基づいてパルス幅を動的に調整することにより、さまざまなアプリケーションで高精度の制御とレギュレーションを実現できます。この機能は、モーター速度制御、電源レギュレーション、バッテリ管理システムなど、変化する条件に対して継続的な監視と適応が必要となるアプリケーションで特に役立ちます。

5.1 フィードバックベースの PWM 信号複製

入力 PWM 信号の正確なコピーを生成する機能は、最新の電子システムにおいて、特に安全上の冗長性検証が必要なアプリケーションにおいて重要な機能の役割を果たします。この手法により、デュアルチャネル モーター制御システムでの包括的な実装が可能になります。動作の安全を確保する上で、信号の検証と障害検出は非常に重要です。

産業用および安全重視のアプリケーションにおいては、PWM 信号複製は、堅牢な冗長制御パスと継続的なシステム監視を可能にします。この技術は、入力 PWM 信号の特性を取り入れ、同一の出力信号を生成して、高精度のタイミング関係を維持するものです。

この機能は、同期型のスイッチ制御と複数の位相アライン出力が不可欠であるパワー エレクトロニクス アプリケーションにおいて特に有益です。この実装により、信号整合性とノイズ耐性を維持しながら、複数のサブシステムへの信号バッファリング、再生、ファンアウトが可能になります。

モータ制御アプリケーションでは、このアプローチにより、耐障害性動作に関して重要である冗長性の高いゲート駆動信号とデュアルチャネル安全システムを実現できます。システムはリアルタイム検証機能とパフォーマンス監視機により、継続的に信号の監視と有効性確認を行います。安全性と信頼性を非常に重視する産業用機器では、この特長は特に重要です。

実装はシンプルでありながら強力で、タイマ キャプチャ機能を利用して入力 PWM パラメータを読み取り、同一の出力信号を生成します。これにより、周期値とデューティ サイクル値の両方を正確に複製できるため、システム全体で信号の整合性が維持されます。その結果、信号検証、安全冗長性、分散制御機能、堅牢なシステム監視機能など、包括的な利点が実現されます。

次の手順を使用して、フィードバックベースの PWM を生成するようにタイマ インスタンスを設定構成できます。

- タイマをアップカウント モードに設定します。
- 立ち上がりエッジと立ち下がりエッジの両方のエッジ キャプチャ用に CC0 チャネルを設定します。
- CC0 チャネルに基準 PWM 信号を入力します。
- フィードバックベースの PWM を生成するように、出力モードで CC1 を設定します。
- CC1 チャネルの CCCTL.CC2SELU を CC0 に、CCACT.CC2UACT フィールドを CCP 出力トグルに設定します。これにより、CC0 キャプチャ イベントに基づいて CC1 出力がトグルされます。
- CC0 チャネルでのキャプチャ イベントは、入力基準信号によって生成されます。

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

上記の設定は、基準 PWM の周期とデューティ サイクルが変化するシナリオに対するものです。

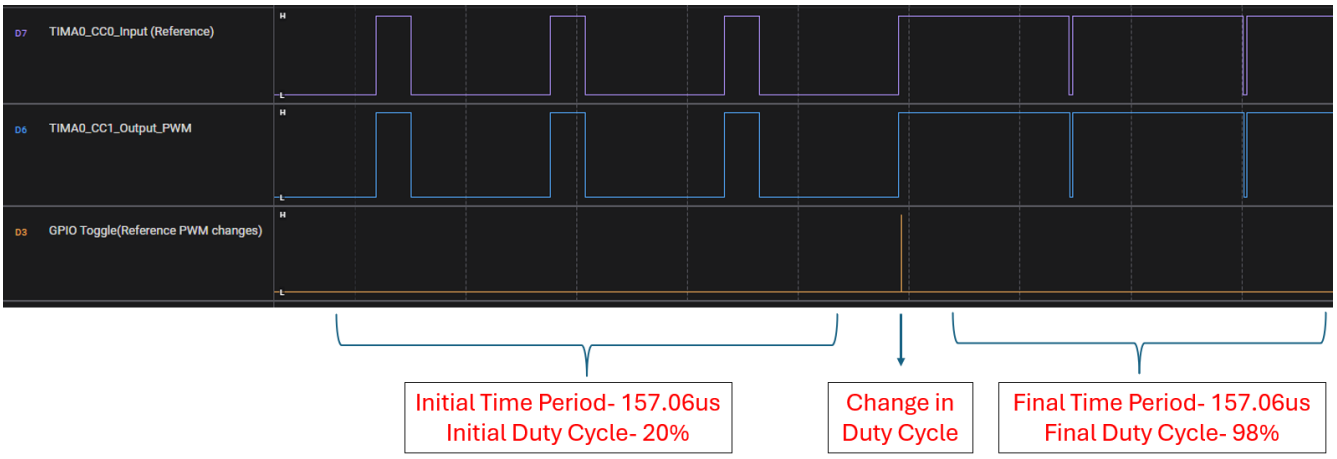

図 5-1. リファレンス PWM のデューティ サイクルのみが変化し

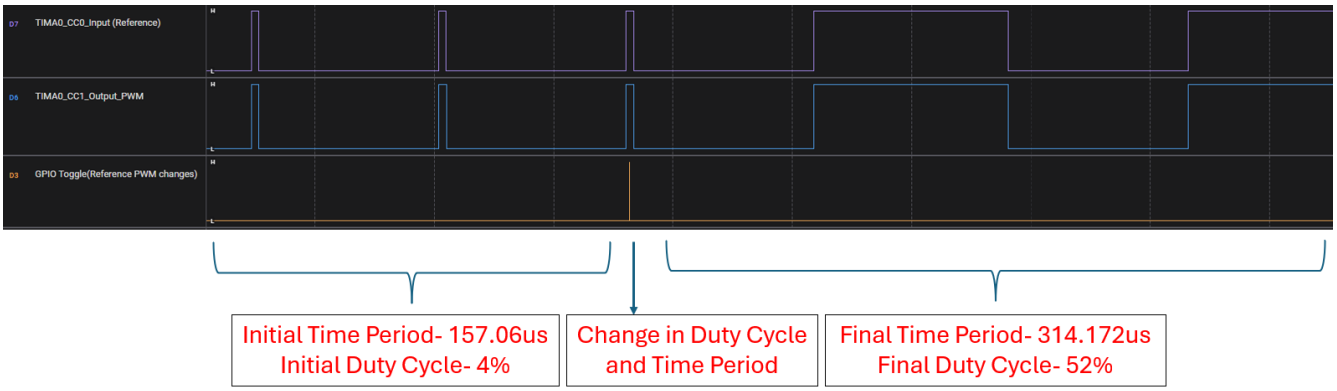

図 5-2. リファレンス PWM のデューティ サイクルと期間の両方が変化

5.2 入力換算を使用した遅延 PWM 信号生成

遅延制御機能を備えたフィードバックベースの PWM 生成により、さまざまなアプリケーション ドメイン、特に高度なモーター制御システムとパワー エレクトロニクス システムに対応した多用途ソリューションを実現できます。この技術により、高精度の位相シフト PWM 生成が可能になります。これはデュアル モーターの同期や相補信号へのデッドタイム挿入に不可欠です。この実装により、LED ディスプレイ アプリケーションや多相モーター ドライブ システムに必須の高精度タイミング制御が可能になります。

パワー エレクトロニクス アプリケーションでは、この手法が多相 DC/DC コンバータや位相シフト ブリッジ コンバータにおいて非常に貴重であることが実証されています。制御遅延を導入することにより、ハーフブリッジ ドライバに効率的なデッドタイム挿入が可能になり、インターリーブ電源動作が容易になりします。このシステムは、特定のタイミング オフセットがある出力 PWM 信号を正確に生成できます。

この機能の用途は、制御遅延信号リピータや分散型システムのタイミング補償が不可欠な信号処理アプリケーションにまで及んでいます。この手法は、位相制御による高度なクロック信号分配をサポートしており、堅牢なカスケード制御システムを実現します。具体的な実装として、位相シフトを備えた同期整流降圧コンバータや、高精度の位相制御を必要とするマルチチャネル LED 調光アプリケーションなどがあります。

この実装では、タイマ キャプチャ機能を利用して入力信号のタイミングを測定し、高精度の位相関係を持つ遅延出力を生成します。このアプローチにより、正確なタイミング遅延を確保すると同時に、システム全体で同期動作を維持します。このソリューションには、高精度の位相制御、確定的なタイミング遅延、信号リタイミングによるノイズ耐性の強化など、大きな利点があります。

このソリューションは、同期された電力段制御と、複数の PWM 信号間で高精度の位相関係を必要とするアプリケーションで特に有益です。

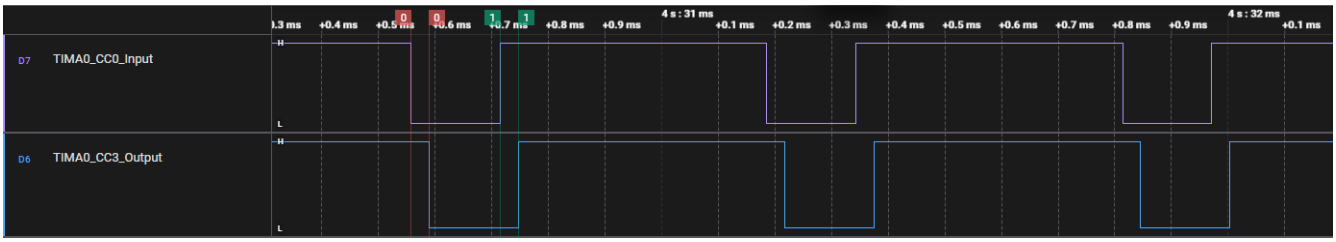

図 5-3. 入力換算により固定遅延を含めて生成された PWM 信号

設定シーケンスを以下に示します。

- ロード値と CM をアップカウント モードに設定します。
- キャプチャ モードでは、立ち上がりエッジ検出に CC0、立ち下がりエッジ検出に CC1 を設定します。CC0 の立ち上がりエッジでゼロ条件が生成されるように、ゼロ条件 (CCCTL.ZCOND) を設定します。
- CC0 の入力も CC1 チャネルに供給されるように、CC1 の IFCTL.ISEL を 0x2 として設定します。
- リファレンス入力 PWM は CC0 チャネルに供給されます。内部については、CC0 入力は CC1 チャネルにも供給されます
- CC4 を CC3 のセカンダリ CC チャネルとして設定すると、CC4 比較イベントによって CC3 のセカンダリ比較イベントが生成されます。
- CC0_UP 割り込みと CC1_UP 割り込みを有効にします。
- CUACT により CC 出力が HIGH に、CC2UACT により CC 出力が LOW になるように、CC3 の CCACT を設定します。
- CC3 を出力として設定します。固定遅延を含む最終出力信号が CC3 によって生成されます。
- 割り込みサービス ルーチンでは、CC0_UP 割り込みを使用して、アプリケーション要件に応じた遅延を含めて CC3 値を更新します。CC1_UP 割り込みを使用して、アプリケーションに応じた遅延を含めて CC4 値を遅更新します。

注

複数の ISR、より優先度の高い割り込み、または高い PWM 周波数を使用するアプリケーションでは、タイマ ISR の優先度を高め、CC 値をできるだけ早く更新できるように ISR コードを最適化する必要があります。

```
/* 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,
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;
    }
}
```

6 遅延タイマの開始:設定可能な遅延を含む同期タイマ インスタンスの開始

モーター制御やセンサ データのポーリングまたはサンプリングを行う一部のアプリケーションにおいて、開始時間オフセットが意図的に設けられているタイマを使用すると、効率、安定性、信号整合性に不可欠な制御された位相関係を実現できます。これは、以下に示すように、クロストリガ メカニズムによる同期方式で 2 つ以上のタイマを使用することで実現できます。

- セクション 3 に示すように、ソフトウェアベースのクロストリガ メカニズムでタイマを設定します。
- アプリケーションに基づいて、最初に有効にするタイマをプライマリ タイマとして設定できます。
- 必要な開始時間オフセットに応じて CC 値を設定します。
- セカンダリ タイマを有効にするために、CC イベント ISR 内にソフトウェア クロストリガを生成します。

注

遅延制限により、最初の PWM 出力に特定のデューティ サイクルが必要な場合に 2 番目のタイマの遅延を設定できないため、最初の PWM の CC4/5 イベントを使用すると、遅延の設定に必要な柔軟性が得られます。

たとえば、TIMA0 有効後化から TIMA1 有効化前までの間に 10ms のオフセットが必要です。最初に TIMA1 を有効にする必要がある場合は、TIMA1 をプライマリ タイマに、TIMA0 をセカンダリ タイマに設定します。有効にしてから正確に 10ms 後に CC イベントが生成されるように、TIMA1 の CC 値を設定します。この CC イベント内では、正確に 10ms 経過直後に TIMA0 を有効にするようにソフトウェア クロストリガを生成できます。つまり、CLKSEL を LFCLK (32KHz) にすると、アップカウント モードのタイマで TIMA1 の CC 値を 320 に設定でき、約 10ms (320 サイクル × (1/32000) 秒 = 0.01 秒 = 10ms) 後に CC イベントが生成されます。この CC イベントは、ソフトウェアベースのクロストリガ メカニズムを使用して TIMA0 を有効化します。

```
/* 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;
    }
}
```

注

このアプリケーションにはハードウェア クロストリガ機能も利用できますが、このメカニズムによって CC イベントごとにクロストリガが生成されることを認識しておくことが重要です。したがって、開発者がハードウェア クロストリガ機能を利用する場合は、このアプリケーションを想定通りに機能させるためには、1 回のクロストリガの後にこの機能を確実に無効化する必要があります。

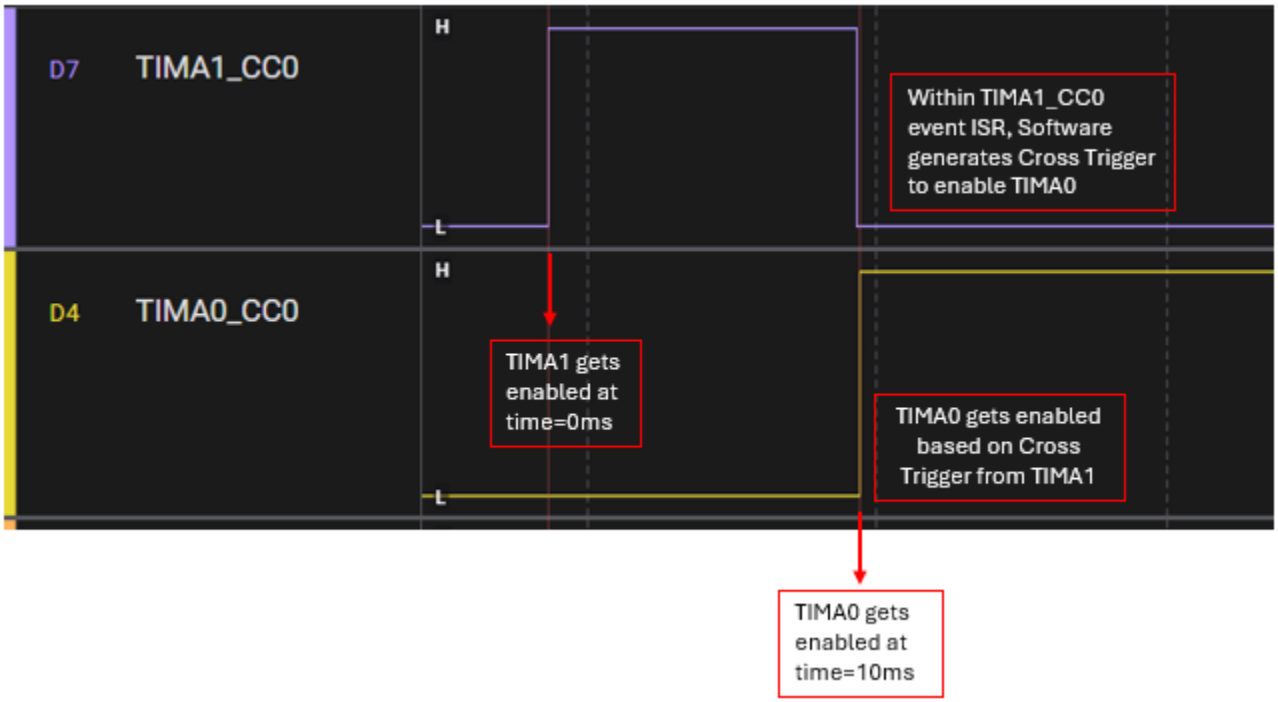

図 6-1. 設定可能なタイマ開始時間オフセット

7 ハードウェア イベントに基づいた作動中タイマーの停止

ハードウェア イベントを使用して作動中のタイマを停止すると、、**低レイテンシ**で高精度かつ自律的な制御を実現できます。その結果、センシング、安全性、制御システムにおいて不可欠なリアルタイムの測定と応答が実現されます。作動中のタイマーを停止するためのフローチャートを図 7-1 に示します。

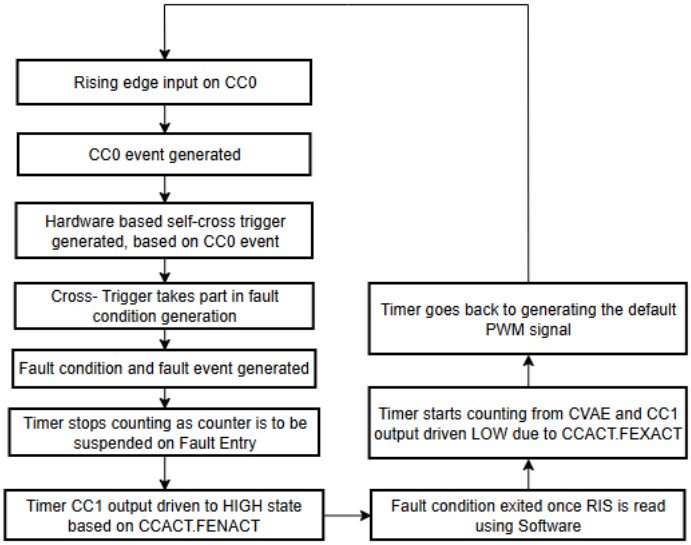

図 7-1. ハードウェアによる作動中タイマの停止方法を示したフローチャート

この要件を満たすために、高度タイマ (TIMA) で次の設定を行うことができます。

- CC イベント時にセルフクロストリガが生成されるように TIMA を設定します。
- 立ち上がりエッジ キャプチャの CC0 を設定します。CC0 ピンの外部立ち上がりエッジによって、CC0 イベントとセルフクロストリガが生成されます。
- 使用している TIMA インスタンスに対してハードウェアベースの異常検出メカニズムを設定します。
- FCTL.TFIM ビットを「1」に設定します。これは、選択されたトリガが異常状態の生成に関与することを意味します。これにより、クロストリガが異常状態を生成してタイマを停止させます。
- フォルト ラッチ モードを有効にするには、FCTL.FL を 1 に設定します。これは、その異常状態全体が CPU_INT.RIS.F ビットに依存することを意味します。
- カウンタを中断するように異常エントリを設定します。これにより作動中のカウンタが停止します。

```
/* 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);

    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);
}
```

注

上記のアプリケーションは、CC0 の立ち上がりエッジを利用してセルフクロストリガを生成し、その結果として異常状態が生成されてタイマを停止します。同様に、イベント ファブリック経由で TIMA が受信した他のペリフェラルから発生したイベント (たとえば、GPIO イベント) を利用した場合も、セルフクロストリガを生成し、その結果として異常状態が生成され、ハードウェアを使用してタイマを停止させることができます。

8 動的 PWM 更新：デューティ サイクルと期間の調整

ランタイム中に PWM 特性を変更できるため、多様なアプリケーション間での高精度な制御が可能で、周波数とデューティ サイクルの両方を柔軟に調整できます。タイマ比較 (CC) とロードの値を動的に更新することで、システムは安定した動作を維持しながらリアルタイム制御も実行できます。この機能は、シャドウ レジスタ機能を使用して実装すると特に強力になり、あらかじめ決められた更新ポイントでグリッチのない遷移が確実に行われます。

モーター制御システムでこの機能を活用すると、段階的な周波数調整と、デューティ サイクル変調による高精度トルク制御を通じて、スムーズ スピード ランピングを実現できます。同じメカニズムが非常に重要となるのは LED および照明アプリケーションであり、高品質なユーザー体験を実現するには、スムーズな輝度遷移や複雑な調光パターンが不可欠です。電源アプリケーションでは、適応型電圧レギュレーションと動的な周波数調整を通じて最適な効率を実現できます。一方、温度制御システムでは、この機能を活用して、PWM ベースのヒーター制御とファン速度変調による精密な熱管理を実現しています。

8.1 シャドウ ロード機能とシャドウ コンペア機能

TIMA はシャドウ レジスタを備えており、PWM のデューティと周期の動的変化をサポートするように CC と LOAD 値を設定できます (図 8-1 を参照)。シャドウ レジスタのメカニズムは、通常は周期境界の安全な遷移ポイントでパラメータの更新が確実に行われるようにするというもので、システムの安定性を維持するために重要な役割を果たします。この決定論的アプローチにより、PWM 出力グリッチ、不規則なパルス幅、および、レジスタの直接更新で発生する可能性のある不測の周波数変動を防止できます。その結果、スムーズな遷移と精密なタイミング制御を必要とする用途に適した堅牢なソリューションとなり、予測可能な動作と安定した動作が非常に重要となるシステムでは特に有益です。

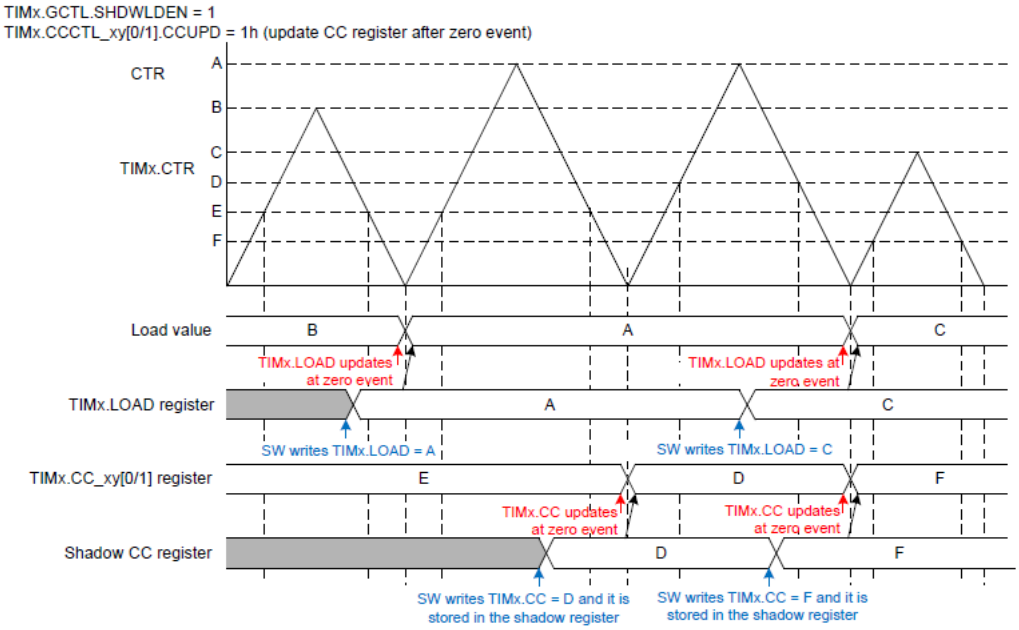

図 8-1. アップ / ダウン モードにおけるゼロ イベントで効果を発揮するシャドウ ロードとシャドウ コンペア

注

シャドウ レジスタなしでは、アプリケーションが PWM デューティ サイクルを動的に更新する際に、不測の PWM 出力 (デューティ サイクルが 0% または 100%) がランダムに発生する可能性があります。この問題を軽減するには、このタイマのロード イベントまたはゼロ イベントでトリガされる割り込みを有効にした上で、割り込み処理遅延を考慮した適切な最大/最小デューティ サイクル制約を実装すると同時に、ISR 内でデューティ サイクルの更新を実行します。また、DMA 更新の CC 値を使用して CPU のオーバーヘッドと割り込み処理遅延を低減することも検討してください。

この実装は、システムの整合性を維持しながら優れた柔軟性を提供するものであり、シンプルな輝度制御から複雑なモーター ドライブ システムまで、幅広い用途に利用できます。ソフトスタート シーケンス、熱管理システム、洗練された PID 制

御ループのいずれを実装する場合でも、PWM パラメータを動的に変更する能力を実現すると同時に、グリッチのない動作を確保できるので、最新の組み込みシステム設計に役立つ強力なツールとして活用できます。

シャドウ機能を使用すれば、固定の基準点で CC 値とロード値を更新できます。ロード値を変更する必要があるときに、TIMx.LOAD 値を直接書き込むことができます。内部については、ロード値は、ソフトウェアによる更新タイミングに関係なく、ゼロ イベント時にのみ更新されます。CC 値を変更する必要がある場合は、TIMx.CC 値に直接書き込んでください。内部については、CC 値は、ソフトウェアによる更新タイミングに関係なく、CC 更新方法で設定されているイベント時にのみ更新されます。シャドウ機能を有効にするには、TIMx.GCTL.SHDWLDEN (= 1) レジスタに書き込んでください。

```
/* 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);
```

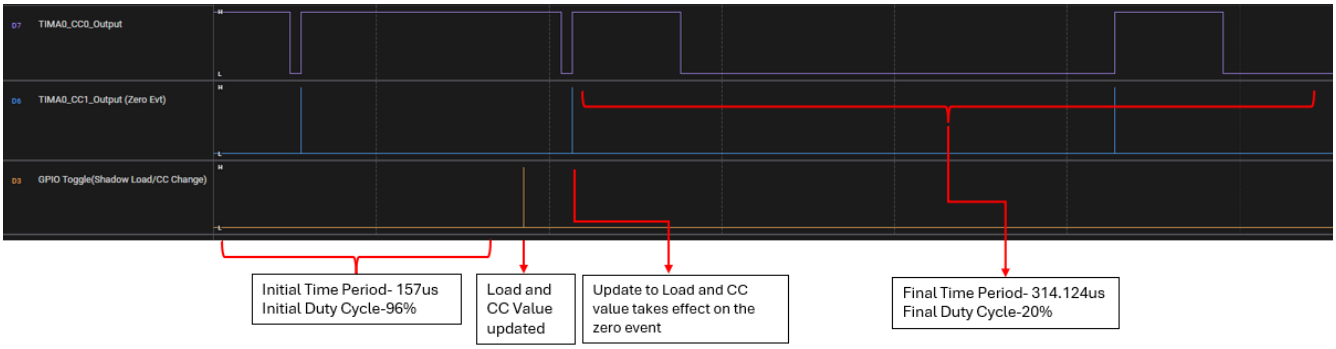

図 8-2. PWM 周期とデューティ サイクルの変化を示す波形

ユーザーは、図 8-3 に示すように、アプリケーション要件に従って CC および CCACT 更新イベントを設定できます。

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.

図 8-3. シャドウ コンペアとアクション更新の動作

シャドウ レジスタの動作特性

- デフォルト状態：シャドウ機能有効化後にユーザー定義の初期値が CC または LOAD レジスタに指定されていない場合、シャドウ レジスタはデフォルトにより 0x0 に初期化される

注

ユーザーがシャドウ CC 機能有効化前に CC 値を設定し、シャドウ CC 有効化後に 2 番目の CC 値を設定しなかった場合、次のユーザー定義イベントが発生した後、CC 値はゼロに戻り、ゼロのまま維持されます。

- ランタイム更新ロジック：シャドウ機能有効化後に行われた CC 値または LOAD 値の変更は、次のユーザー定義イベント トリガが発生するまで保留状態になる

注

シャドウ CC 機能有効化後にユーザーが初期 CC 値を設定した場合、CC 値は次のユーザー定義イベントが発生するまでゼロのまま維持されます。

8.2 DMA による任意信号の生成

タイマ モジュールは、イベント ファブリック経由で DMA 要求を生成できます。この機能により、タイマ ペリフェラル レジスタのコンテンツを調整することで、タイマ ペリフェラルによって波形出力を動作中に変更できます。たとえば、ユーザーは

TIMx.LOAD レジスタの内容を更新して波形周波数を調整したり、TIMx.CC レジスタを更新して信号のデューティ サイクルを調整したりすることができます。

標準的な PWM 生成は通常、タイマのロード レジスタとキャプチャ比較レジスタのそれぞれで定義される固定の周波数とデューティ サイクル パラメータに依存します。これらのレジスタはソフトウェアの介入によって手動で更新できますが、このやり方では CPU のオーバーヘッドとタイミングの大きな不確定性が生じる可能性があります。

このセクションでは、DMA (ダイレクト メモリ アクセス) 機能をタイマ ペリフェラルと組み合わせて利用する最適化されたアプローチについて説明します。このアプローチでは以下を実現できます。

- 決定論的確定的な波形生成
- 最小限の CPU 介入
- 高精度のタイミング制御
- パラメータの自動更新

タイマ反復 1 回ごとに CPU 主導の更新に依存するのではなく、ハードウェア トリガの DMA 転送を利用して周波数とデューティ サイクルの両方のパラメータを自動的に更新します。この手法ではにより以下が確保されます。

1. 連続的な PWM サイクル間の高精度タイミングの関係
2. 予測可能でジッタフリーの波形生成
3. システム リソースの効率的な使用
4. シャドウ機能を使用しない PWM のデューティと周期の動的更新

このセクションでは、このタイマの更新イベントにより自動 DMA 転送が開始されてレジスタがシームレスに更新されるという方法を示した実装戦略について詳しく説明します。これにより高精度の波形制御を維持しながらも CPU のオーバーヘッドを排除できます。

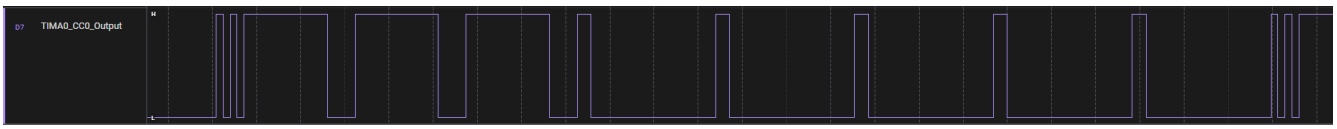

図 8-4. DMA を使用した任意信号の生成

アプリケーション フローを以下に示します。

- TIMER をイベント パブリッシャとして設定し、DMA をトリガするイベントを発行します。FPUB0 を使用して DMA_CHAN0 をトリガし、FPUB1 を使用して DMA_CHAN1 をトリガします。
- DMA_CHAN0 は LOAD レジスタへの書き込みに使用され、DMA_CHAN1 は CC レジスタへの書き込みに使用されます。
- LOAD 値を変更すると周期が変動しますが、CC 値を変更するとデューティ サイクルが変動します。
- CC 更新イベントをゼロ イベントとして設定します。つまり、次回のゼロ イベントで CC 値が更新されます。
- 16 ビットを幅とするリピート シングル転送モードで DMA をサブスクライバに設定します。
- DMA の送信元アドレスをインクリメント増分に設定し、宛先アドレスを未変更にとして設定します。
- DMA_CHAN0 の場合、送信元アドレスは、TIMx.LOAD レジスタ アドレスとして LOAD 値と送信先アドレスを含んでいるバッファとして設定します。
- DMA_CHAN1 の場合、送信元アドレスは、TIMx.CC レジスタ アドレスとして CC 値と送信先アドレスを含んでいるバッファとして設定します。
- CC1 の場合、OCTL.CCPO は 0x4 として設定されています。このため、カウンタ値がゼロの場合、CC1 出力信号はトグルし、1 TIMCLK サイクルの HIGH 値に設定されます。その後、CC1 出力はハードウェアによってクリアされます。これは、ゼロ イベントに対する負荷と CC 更新タイミングをチェックするために使われます。この機能は、他のさまざまなアプリケーションにおいてデバッグを目的として使用することもできます。

```
/* 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);

    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
                         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) {
        ;
    }
}
```

9 まとめ

このアプリケーション ノートでは、TIMA モジュールの多用途機能に注目し、MSPM0 マイコンの高度タイマ (TIMA) 技術について説明します。本書では、PWM idle-low 設定、位相シフト PWM 生成、UART エミュレーション用ビットバンギングを取り上げています。また、フィードバックベースの PWM 生成、設定可能な遅延時間を用いた同期型タイマ開始、およびハードウェアトリガによるタイマ停止制御についても解説しています。主な機能には、動的 PWM 更新用のシャドウ レジスタと、DMA ベースの任意信号の生成が含まれます。コード スニペットを含む実例では、最新の組み込みシステムの実装戦略を示しています。

10 参考資料

- テキサス・インスツルメンツ、『MSPM0 サンプル プロジェクトの高度タイマ技術』、FAQ
- テキサス インスツルメンツ、『CAN-FD インターフェイスを搭載した MSPM0Gx51x 混合信号マイコン』、データシート
- テキサス インスツルメンツ、『MSPM0 G シリーズ 80MHz マイコン テクニカル リファレンス マニュアル』、テクニカル リファレンス マニュアル。

重要なお知らせと免責事項

TI は、技術データと信頼性データ (データシートを含みます)、設計リソース (リファレンス デザインを含みます)、アプリケーションや設計に関する各種アドバイス、Web ツール、安全性情報、その他のリソースを、欠陥が存在する可能性のある「現状のまま」提供しており、商品性および特定目的に対する適合性の黙示保証、第三者の知的財産権の非侵害保証を含むいかなる保証も、明示的または黙示的にかかわらず拒否します。

これらのリソースは、TI 製品を使用する設計の経験を積んだ開発者への提供を意図したものです。(1) お客様のアプリケーションに適した TI 製品の選定、(2) お客様のアプリケーションの設計、検証、試験、(3) お客様のアプリケーションに該当する各種規格や、その他のあらゆる安全性、セキュリティ、規制、または他の要件への確実な適合に関する責任を、お客様のみが単独で負うものとします。

上記の各種リソースは、予告なく変更される可能性があります。これらのリソースは、リソースで説明されている TI 製品を使用するアプリケーションの開発の目的でのみ、TI はその使用をお客様に許諾します。これらのリソースに関して、他の目的で複製することや掲載することは禁止されています。TI や第三者の知的財産権のライセンスが付与されている訳ではありません。お客様は、これらのリソースを自身で使用した結果発生するあらゆる申し立て、損害、費用、損失、責任について、TI およびその代理人を完全に補償するものとし、TI は一切の責任を拒否します。

TI の製品は、 TI の販売条件 、 TI の総合的な品質ガイドライン 、 ti.com または TI 製品などに関連して提供される他の適用条件に従い提供されます。TI がこれらのリソースを提供することは、適用される TI の保証または他の保証の放棄の拡大や変更を意味するものではありません。 TI がカスタム、またはカスタマー仕様として明示的に指定していない限り、TI の製品は標準的なカタログに掲載される汎用機器です。

お客様がいかなる追加条項または代替条項を提案する場合も、TI はそれらに異議を唱え、拒否します。

最終更新日：2025 年 10 月