参考資料

AN4776
アプリケーション・ノート
汎用タイマ詳細説明書

はじめに

タイマペリフェラルは、すべての STM32 マイクロコントローラに組み込まれているペリフェラルの基本セットの一部です。タイマペリフェラルの数とそれぞれの機能は、STM32 マイクロコントローラファミリごとに異なりますが、一部の一般的な機能と動作モードについては、すべてのタイマペリフェラルが共有しています。

STM32 タイマペリフェラルは、モータ制御アプリケーションから周期的イベント生成アプリケーションまで、多数のアプリケーションで要となるペリフェラルであるとみなされています。すべての STM32 リファレンスマニュアルに記載されているタイマペリフェラルの仕様は、その多用性により、非常に多岐にわたっています。

このアプリケーションノートの目的は、STM32 汎用タイマペリフェラルの基本的な機能と動作モードについて簡単にわかりやすく説明することです。本書は、STM32 のリファレンスマニュアルに記載されているタイマペリフェラルの仕様を補完するものです。

本書は、次の 2 つの主要部分に分かれています。

- 最初のセクションでは、STM32 タイマの基本機能について簡単に説明し、さらにタイマペリフェラルベースのアプリケーションで共通して使用されるいくつかの特定機能について説明します。
- それ以降のセクションでは、STM32 タイマペリフェラルの個別の使用例について説明します。これらのセクションでは、アプリケーション例を構築するために使用する STM32 タイマの主要機能について詳しく説明します。また、使用するソースコードのアーキテクチャも説明します。

このアプリケーションノートの目標は、STM32 タイマペリフェラルのいくつかの使用例について汎用的かつ簡単な方法で説明することです。モータ制御アプリケーションのような使用例は複雑なので、ここでは説明しません。

表 1. 対象とする製品

タイプ	製品シリーズ
マイクロコントローラ	STM32F0 シリーズ、STM32F1 シリーズ、STM32F2 シリーズ、STM32F3 シリーズ、STM32F4 シリーズ、STM32F7 シリーズ、STM32L0 シリーズ、STM32L1 シリーズ、STM32L1W シリーズ、STM32L4 シリーズ

参考資料

目次

1 STM32 の汎用タイマの基本的な動作モード 6

1.1 はじめに 6

1.2 STM32 のタイマペリフェラルの詳細 6

1.2.1 マスタ / スレーブコントローラユニット 8

1.2.2 タイムベースユニット 8

1.2.3 タイマチャネルユニット 9

1.2.4 ブレーク機能ユニット 11

1.3 STM32 のタイマペリフェラルの基本的な動作モード 11

1.3.1 タイマのタイムベース設定 11

1.3.2 入力モードでのタイマチャネルの設定 12

1.3.3 出力モードでのタイマチャネルの設定 12

1.4 STM32 のタイマペリフェラルの高度な機能 13

1.4.1 フィルタリングステージ 13

1.4.2 タイマレジスタのプリロード機能 14

2 外部クロックソースによるタイマへのクロック供給 17

2.1 概要 17

2.2 同期ブロック 18

2.3 外部クロックソースモード 1 19

2.4 外部クロックソースモード 2 21

2.5 外部クロックソースのモード 1 とモード 2 23

2.6 アプリケーション：ETR タイマ入力で外部クロックソースを使用したタイマへのクロック供給 24

2.7 ファームウェアの概要 32

3 ワンパルスモードによる N パルス波形生成 33

3.1 概要 33

3.2 アプリケーション：ワンパルスモードによる N パルス波形生成 34

3.3 ファームウェアの概要 36

参考資料

4 ブレーク入力によるサイクルごとのレギュレーション 37
4.1 概要 37
4.2 ブレーク入力と OCxRef クリアの利用 37
4.3 アプリケーション：ブレーク機能によるサイクルごとのレギュレーション 39
4.4 ファームウェアの概要 43

5 タイマ DMA バースト機能による任意の波形の生成 44
5.1 STM32 の DMA バースト機能の概要 44
5.2 タイマ DMA バースト機能 44
5.3 アプリケーション例：タイマ DMA バースト機能による任意の波形の生成 48

6 タイマ同期による N パルス波形生成 57
6.1 タイマ同期の概要 57
6.2 N パルス波形生成アプリケーション例 - パート 1 59
6.3 N パルス波形生成アプリケーション例 - パート 2 64
6.3.1 クロック設定 67

7 改版履歴 72

参考資料

表の一覧

表 1. 対象とする製品 1
表 2. 文書改版履歴 72
表 3. 日本語版文書改版履歴 72

参考資料

図の一覧

図 1. TIM1 タイマペリフェラルのブロック図 7
図 2. 出力として設定されているタイマチャネル関連のブロック図 9
図 3. 入力として設定されているタイマチャネル関連のブロック図 11
図 4. 入力信号フィルタリング（ETF [3:0]= 0100）: FSAMPLING = FDTS/2、N=6. 14
図 5. タイマチャネルレジスタのプリロードメカニズム - 無効な場合 15
図 6. タイマチャネルレジスタのプリロードメカニズム - 有効な場合 15
図 7. 外部クロック信号による STM32 のタイマの同期 17
図 8. 外部クロックソースモードのクロックパス 18
図 9. 同期ブロック 19
図 10. タイマカウンタのインクリメント（外部クロックモード 1） 21
図 11. タイマカウンタのインクリメント（外部クロックモード 2） 22
図 12. 周波数メータの概要図. 24
図 13. 内部 HSI オシレータからクロック供給される周波数メータのアーキテクチャ 26
図 14. 外部クロックソースモード 2 でクロック供給される周波数メータのアーキテクチャ 27
図 15. 入力キャプチャのタイミング図 28
図 16. 内部ソースクロックの PPM 30
図 17. 外部ソースのクロックの PPM 31
図 18. プロジェクト構成 32
図 19. ワンパルスモードの例. 34
図 20. アーキテクチャの例 35
図 21. ファームウェアのソースコード構成 36
図 22. TIMx OCxREF クリアのタイミング 38
図 23. ブレーク機能のタイミング 38
図 24. サイクルごとのレギュレーションのタイミング 39
図 25. サイクルごとのレギュレーションのアーキテクチャ 40
図 26. 得られた波形を示すオシロスコープのスクリーンショット 41
図 27. プロジェクト構成 43
図 28. タイマ DMA バースト転送シーケンスの設定 46
図 29. DMA バーストによる任意波形生成の概要図. 48
図 30. 任意波形ジェネレータアプリケーション：ターゲットの波形 49
図 31. マイクロコントローラのメモリに格納される波形生成データパターン 51
図 32. 任意波形生成例のブロック図 52
図 33. TIM1 タイマのチャネル 1 での任意の信号生成 53
図 34. 周期的 N パルス生成のブロック図 59
図 35. 周期的 N パルス生成例の出力波形のターゲット 60
図 36. 周期的 N パルス生成の概要図. 60
図 37. 周期的 N パルス生成例のタイミング図. 61
図 38. 2 つの相補 N パルス波形生成例の概要図 65
図 39. 2 つの相補 N パルス波形生成例の出力 65
図 40. 同様の最終状態を持つ相補 N パルス波形生成のタイミング図. 68

参考資料

1 STM32 の汎用タイマの基本的な動作モード

1.1 はじめに

すべての STM32 マイクロコントローラは少なくとも 1 つのタイマペリフェラルを内蔵しており、一部は複数タイプのタイマペリフェラルを内蔵しています。本書では、汎用タイマについて説明します。汎用タイマは、その名前で STM32 の他のタイプのタイマペリフェラルと区別できます。

STM32 マイクロコントローラのドキュメントでは、汎用タイマペリフェラルは常に「TIMx タイマ」という名前で呼ばれます。「x」には任意の数を使用でき、特定のマイクロコントローラに内蔵されているタイマペリフェラルの数とは関係ありません。たとえば、STM32F100 マイクロコントローラには TIM17 という名前のタイマペリフェラルが内蔵されていますが、このマイクロコントローラに内蔵されているタイマペリフェラルの総数は 17 未満です。

一般に、STM32 マイクロコントローラファミリ全体で、同じ名前のタイマペリフェラルは同じ機能セットを備えていますが、いくつかの例外があります。たとえば、TIM1 タイマペリフェラルは、STM32F1 シリーズ、STM32F2 シリーズ、および STM32F4 シリーズで共有されていますが、STM32F30x マイクロコントローラファミリの特定のケースでは、TIM1 タイマペリフェラルは、他のファミリの TIM1 よりも多少豊富な機能セットを備えています。

STM32 マイクロコントローラに内蔵されている汎用タイマは、同じ基本構造を共有しており、個々のタイマペリフェラルに組み込まれている機能のレベルのみが異なります。個々のタイマペリフェラルの機能集積レベルは、ターゲットとなるアプリケーション分野によって決まります。

タイマペリフェラルは、次のように分類できます。

- TIM1 や TIM8 のような高機能タイマ。
- TIM2 や TIM3 のような汎用タイマ。
- TIM9、TIM10、TIM12、および TIM16 のような簡易タイマ。
- TIM6 や TIM7 のような基本タイマ。

アプリケーションノートSTM32 cross-series timer overview（AN4013）では、さまざまな STM32 マイクロコントローラファミリで使用されている STM32 のタイマペリフェラル全般について詳しく説明しています。

1.2 STM32 のタイマペリフェラルの詳細

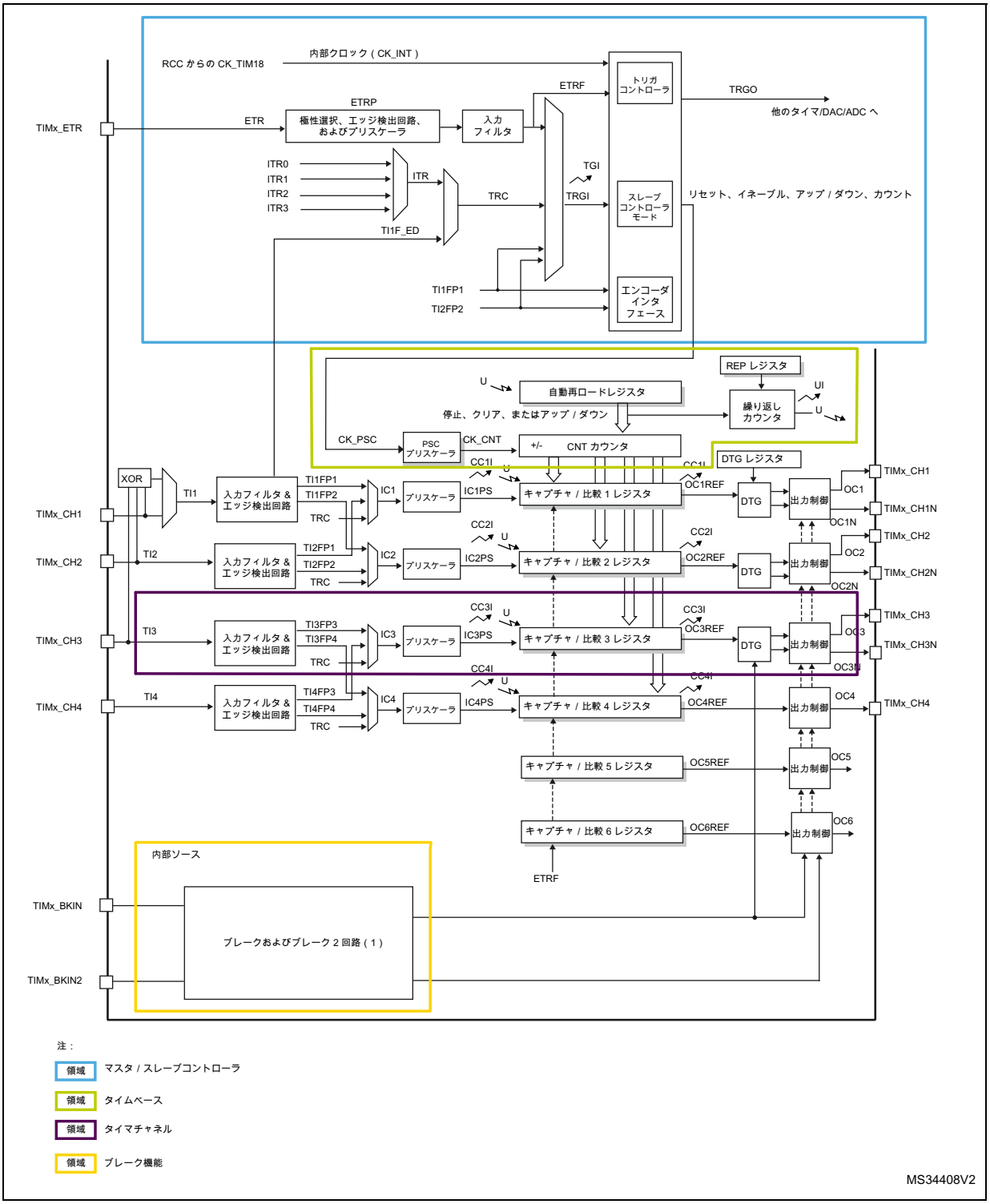
STM32 のすべての汎用タイマペリフェラルは、同じ基本構造を共有しています。このセクションでは、高機能設定 TIM1 タイマペリフェラルについて詳しく説明します。これは、最も多くの機能を備えているタイマペリフェラルです。

図 1 に、TIM1 タイマペリフェラルのブロック図を示します。STM32 のタイマペリフェラルは、次の 4 つのユニットで構成されています。

1. マスタ/スレーブコントローラユニット
2. タイムベースユニット
3. タイマチャネルユニット
4. ブレーク機能ユニット

参考資料

図 1. TIM1 タイマペリフェラルのブロック図



参考資料

1.2.1 マスタ/スレーブコントローラユニット

マスタ/スレーブユニットは、タイムベースユニットにカウントクロック信号（CK_PSC 信号など）およびカウント方向制御信号を提供します。このユニットは、主にタイムベースユニットに制御信号を提供します。

マスタ/スレーブコントローラは、タイマのマスタ/スレーブ設定に基づいて、タイムベースユニットの適切なカウント設定を決定します。また、実際のカウントステータスも決定します。

たとえば、TIMx_SMCR タイマレジスタの SMS 制御ビットフィールドに適切な値を書き込むことによってタイマがいずれかのエンコーダモードに設定されている場合、カウントクロック信号とカウント方向制御信号は、TI1FP1 と TI2FP2 の各入力信号の状態に基づいて計算されます。

マスタ/スレーブコントローラユニットは、タイマ間同期を処理します。このユニットは、特定のタイマ内部イベントに続いて同期信号（TRGO 信号）を出力するように設定できます。また、外部イベント（他のタイマの内部イベント、外部信号など）の機能でタイムベースカウンタを制御するように設定することもできます。

タイマ更新イベントなどのマスタタイマイベントに基づいて、スレーブタイマの 1 つがそのカウンタをインクリメントするように設定できます。この例では、マスタタイマイベントは、マスタ/タイマのマスタ/スレーブコントローラユニットによって通知されます。この制御ユニットは、マスタタイマ出力 TRGO 信号を使用します。マスタタイマ出力 TRGO 信号は、スレーブタイマ TRGI 入力信号に接続されます。スレーブタイマのマスタ/スレーブコントローラユニットは、TRGI 入力信号をクロックソースとして使用してスレーブタイマカウンタをインクリメントするように設定されます。

STM32 のすべてのタイマペリフェラルが同じマスタ/スレーブコントローラ機能を備えているわけではありません。例として挙げた TIM1 タイマペリフェラルはマスタ/スレーブ機能をフルに備えていますが、TIM6 と TIM7 の基本タイマには最も単純なマスタ/スレーブコントローラが内蔵されています。TIM6 と TIM7 の各タイマペリフェラル内部のマスタ/スレーブコントローラには、制御ビットフィールドは存在しません。

TIM6 と TIM7 の各タイマペリフェラルの場合、タイムベースカウンタは、外部イベントに続いてその内容をリセットする手段がなく、常にアップカウントします。それらのカウンタへのクロック供給元として、別のクロックソースまたはタイマペリフェラルの内部クロックを使用することはできません。

1.2.2 タイムベースユニット

タイムベースユニットは、プリスケーラステージと繰り返しカウンタ、さらにタイマカウンタから構成されます。タイムベースユニットに供給されたクロック信号は、タイムベースカウンタに到達する前に、まずプリスケーリングステージを通過します。

カウント信号の周波数は、TIMx_PSC タイマプリスケーラレジスタの内容に応じて、カウンタステージに到達する前に低減することもできます。プリスケーリングステージの出力信号は、タイマカウンタステージのクロックカウント信号になります。

タイマカウンタは、次の 2 つのタイマレジスタによって制御されます。

- TIMx_CNT タイマレジスタは、タイマカウンタの内容の読み書きに使用します。
- TIMx_ARR タイマレジスタには、タイマカウンタの再ロード値が格納されます。
  - タイマカウンタがアップカウントしていて、タイマ自動再ロードレジスタ（TIMx_ARR）の内容に達した場合、タイマカウンタが自身をリセットして、新しいカウントサイクルをリスタートします。
  - タイマカウンタがダウンカウントしていて、ゼロ値に達した場合、タイマカウンタ値はタイマ自動再ロードレジスタ（TIMx_ARR）の内容にセットされ、新しいカウントサイクルがリスタートされます。

繰り返しカウンタの内容が null である限り、新しいカウントサイクルがリスタートされるたびにタイマの「更新イベント」がトリガされます。繰り返しカウンタの内容が null ではない場合、「更新イベント」はトリガされません。ただし、新しいカウントサイクルがリスタートされ、繰り返しカウンタの内容が 1 ずつ減ります。各「更新イベント」に続いて、繰り返しカウンタの内容は TIMx_RCR タイマレジスタに格納されている値にセットされます。

繰り返しカウンタは、STM32 のすべてのタイマペリフェラルに内蔵されているわけではありません。繰り返しカウンタが内蔵されていない場合、タイマペリフェラルは、繰り返しカウンタは内蔵されているがその内容が null であるかのように動作します。

1.2.3 タイマチャネルユニット

タイマチャネルはタイマの動作素子であり、これによりタイマペリフェラルは外部環境とやり取りします。タイマチャネルは、一般には STM32 マイクロコントローラのピンに割り当てられますが、STM32F30x マイクロコントローラファミリの TIM1 タイマペリフェラルのタイマチャネル 5 と 6 のような例外もいくつかあります。STM32 マイクロコントローラのピンに割り当てられているタイマチャネルは、入力または出力のどちらにも使用できます。

タイマチャネルが出力として設定されている場合

出力として設定されているタイマチャネルを使用して、考えられ得る一連の波形を生成できます。チャネルが出力モードで設定されている限り、TIMx_CCRy チャネルレジスタの内容がタイマカウンタの内容と比較されます。

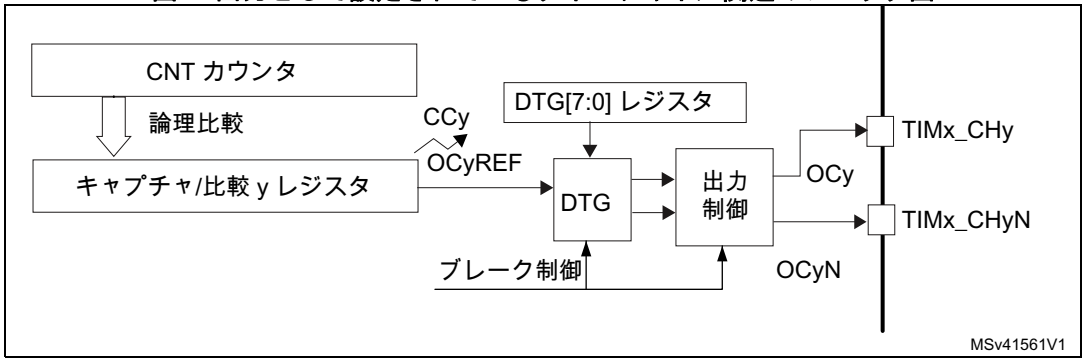
この連続的な論理比較の結果および設定されている出力サブモード（PWM1 モード、インアクティブモードなど）に基づいて、タイマチャネルの内部出力 OCyREF がセットまたはリセットされます。

その後、タイマチャネルの内部出力 OCyREF は、チャネル出力ステージに供給されます。チャネル出力ステージは、設定されている一連のパラメータ（特にチャネル極性設定またはデッドタイム生成）に基づいて、OCyREF 信号に対して一連の調整動作を適用します。

チャネル出力ステージの出力信号は、オルタネート機能としてマイクロコントローラのピンに割り当てられます。タイマチャネルの一部の出力ステージは、図 2 に示す出力ステージのように、2 つの相補信号を出力できます。

そのようなチャネル出力ステージでは、制御ビットフィールドを使用して、各出力信号（出力信号の有効化/無効化、極性など）を個別に設定できます。

図 2. 出力として設定されているタイマチャネル関連のブロック図



タイマチャネルが入力として設定されている場合

タイマチャネルが入力として設定されている場合、外部信号の立ち上がりエッジや立ち下がりエッジのタイムスタンプに使用できます。この機能を処理するため、チャネル入力は、マイクロコントローラのピンの 1 つに割り当てられます。

一部のタイマチャネル入力は、較正目的のオシレータ出力など、一部のオンチップ信号にも割り当てられます。TIy タイマチャネル入力は、図 3 に示すように、チャネル入力調整回路に供給されます。調整回路には、フィルタリングステージとエッジ検出回路が含まれます。フィルタステージでは、設定されているパルス期間より短いパルスは受け付けられません。エッジ検出回路は、該当するタイマ入力で、フィルタリング後にアクティブエッジが発生したかどうかを検出します。

アクティブエッジ設定は、TIMx_CCER タイマレジスタのチャネル極性制御ビットフィールドに書き込むことによってセットします。調整回路は、次の 2 つの信号を出力します。

- TIyFPy：フィルタリングされた TIy タイマ入力信号。ここでタイマチャネル「y」の極性に応じてアクティブエッジが検出されます。
- TIyFPz：これもフィルタリングされた TIy タイマ入力信号ですが、タイマチャネル「z」の極性に応じてアクティブエッジを検出します。

図 3 に示すように、TIyFPz 信号はチャネル「z」のプリスケーラ入力にリダイレクトされ、TIzFPy 信号がチャネル「y」のプリスケーラ入力にリダイレクトされます。フィルタリング後の入力信号をこのように交差させてスワップすることは、入力信号の立ち上がりエッジと立ち下がりエッジの両方をタイムスタンプするのに非常に便利です。このことは、PWM（パルス幅変調）入力のアプリケーションで非常に役に立ちます。

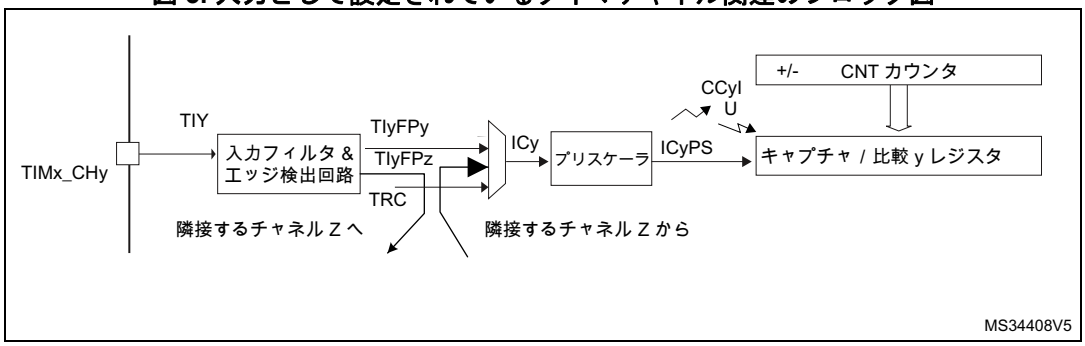
各タイマチャネルは、3 つの入力モードのいずれかに設定できます。各入力モードは、タイマチャネルプリスケーラに接続されているプリスケーラマルチプレクサの 3 つの入力の 1 つに対応します。CCyS 制御ビットフィールドは、タイマチャネルを出力モードに設定（例：CCyS[1:0] を ‘00’ に設定）するか、または入力モードの 1 つに設定（例：CCyS[1:0] を ‘00’ 以外に設定）するかを制御します。

同じ TIMx_CCMRn タイマレジスタ（‘n’ には任意の数を指定可能、ただし通常は 1 または 2 のどちらか）を使用して、タイマチャネルを入力または出力のどちらかとして設定します。TIMx_CCMRn タイマレジスタの一部の制御ビットフィールドは、チャネル設定（入力モードまたは出力モード）に応じて、解釈が異なります。

タイマチャネルプリスケーラは、タイマ入力 TIy で検出されるアクティブエッジの周波数を低減するように設定できます。チャネルプリスケーラの出力でアクティブエッジが検出されると、TIMx_CCRy タイマチャネルの「y」レジスタへのタイマカウンタの内容の転送がトリガされます。

TIMx_CCRy タイマチャネルの「y」レジスタの内容は、チャネル「y」のプリスケーラの出力で最後に検出されたアクティブエッジのタイムスタンプになります。チャネル「y」のプリスケーラが入力信号の周波数を低減しないように設定されている場合（例：プリスケーラ比 = 1、すなわちチャネルプリスケーラがバイパスされている場合）、これは TIy タイマ入力で最後に検出されたアクティブエッジのタイムスタンプになります。

図 3. 入力として設定されているタイマチャネル関連のブロック図



1.2.4 ブレーク機能ユニット

ブレーク機能ユニットを内蔵するのは、相補出力を備えているタイマペリフェラルだけです。言い換えると、2 つの相補出力を備えたチャネルが少なくとも 1 つ存在するタイマペリフェラルだけがブレーク機能を内蔵しています。

ブレーク機能は、出力モードで設定されているタイマチャネルの出力ステージで作用します。ブレーク入力でアクティブエッジが検出されるとすぐに、出力モードで設定されているタイマチャネルの出力がオフになるか、または強制的に事前定義されている安全な状態になります。

ブレーク機能は、通常は異常時の電力インバータの安全停止機能を実装する場合に使用します。

アプリケーションノートUsing STM32 device PWM shut-down features for motor control and digital power conversion（AN4277）では、STM32 マイクロコントローラファミリ全体のブレーク機能について詳しく説明しています。

1.3 STM32 のタイマペリフェラルの基本的な動作モード

このセクションでは、STM32 タイマペリフェラルの一連の基本設定のためのいくつかのソースコードスニペットを示します。これらのスニペットは C プログラミング言語で開発されています。

1.3.1 タイマのタイムベース設定

```
#define ANY_DELAY_RQUIRED        0x0FFF
/* TIM6 タイマペリフェラルによるハードウェア精度の遅延ループの実装です。この機能は STM32 の他のタイマでも実現できますが、集積レベルが低いという理由で TIM6 が選択されました。他のタイマペリフェラルは、もっと複雑なタスクのためにとっておくことができます。*/
/* 更新イベントフラグをクリアします */
TIM6->SR = 0
/* 必要な遅延をセットします */
/* タイマプリスケーラのリセット値は 0 です。遅延を長くする必要がある場合は、プリスケーラレジスタを TIM6->PSC = 0 に */
/* 設定できます。*/
TIM6->ARR = ANY_DELAY_RQUIRED
/* タイマカウンタを開始します */
TIM6->CR1 |= TIM_CR1_CEN
/* 更新イベントフラグがセットされるまでループします */
```

```
while (!(TIM6->SR & TIM_SR_UIF));
/* 必要な遅延時間が経過しました */
/* ユーザコードを実行できます */
```

1.3.2 入力モードでのタイマチャネルの設定

```
/* 最後に検出されたアクティブエッジのタイムスタンプを格納する変数 */
uint32_t TimeStamp;
/* TIM3 タイマの ARR レジスタのリセット値は 0x0000FFFF です。したがって、このスニペットでは問題ないはずです。*/
/* 変更したい場合は、次の行のコメントを解除してください */
/* TIM3->ARR = ANY_VALUE_YOU_WANT */
/* TIM3 タイマチャネル 1 を入力としてセットします */
/* CC1S ビットはチャネル 1 がオフの場合のみ書込み可能です */
/* リセット後のタイマチャネルはすべてオフになります */
TIM3->CCMR1 |= TIM_CCMR1_CC1S_0;
/* TIM3 チャネル 1 を有効にして、チャネル極性のデフォルト設定（リセット後の状態）をそのまま使用します */
TIM3->CCER |= TIM_CCER_CC1E;
/* タイマカウンタを開始します */
TIM3->CR1 |= TIM_CR1_CEN;
/* チャネル 1 のキャプチャイベントフラグをクリアします */
TIM3->SR = ~TIM_SR_CC1IF;
/* キャプチャイベントフラグがセットされるまでループします */
while (!(TIM3->SR & TIM_SR_CC1IF));
/* アクティブエッジが検出されたので、タイムスタンプを格納します */
TimeStamp = TIM3->CCR1;
```

1.3.3 出力モードでのタイマチャネルの設定

```
/* TIM3 タイマの ARR レジスタのリセット値は 0x0000FFFF です。したがって、このスニペットでは問題ないはずです。変更したい場合は、次の行のコメントを解除してください */
/* TIM3->ARR = ANY_VALUE_YOU_WANT */
/* リセット後は TIM3 タイマチャネル 1 は出力として設定されます */
/* TIM3->CC1S のリセット値は 0 です */
/* PWM2 出力モードを選択するには、OC1M 制御ビットフィールドを 111 にセットします */
TIM3->CCMR1 |= TIM_CCMR1_OC1M_0 | TIM_CCMR1_OC1M_1 |
TIM_CCMR1_OC1M_2;
/* デューティサイクルを 50% にセットします */
TIM3->CCR1 = TIM3->ARR / 2;
/* デフォルトでは、リセット後、チャネル 1 のプリロードはオフになります */
/* これを変更するには、次の行のコメントを解除してください */
/* TIM3->CCMR1 |= TIM_CCMR1_OC1PE;
```

```
/* TIM3 チャネル 1 を有効にして、チャネル極性のデフォルト設定（リセット後の状態）をそのまま使用します */
TIM3->CCER |= TIM_CCER_CC1E;
/* タイマカウンタを開始します */
TIM3->CR1 |= TIM_CR1_CEN
```

1.4 STM32 のタイマペリフェラルの高度な機能

このセクションでは、本書内で後で詳しく説明するアプリケーション例で使用する、STM32 のタイマのいくつかの共通機能について詳しく説明します。

1.4.1 フィルタリングステージ

タイマ入力（ETR 入力、チャネル入力など）にはフィルタリングステージがあり、これを有効にして閾値より短い外部信号パルスをフィルタで除外することができます。

フィルタするパルスの最長期間は、次の 2 つのパラメータに依存します。

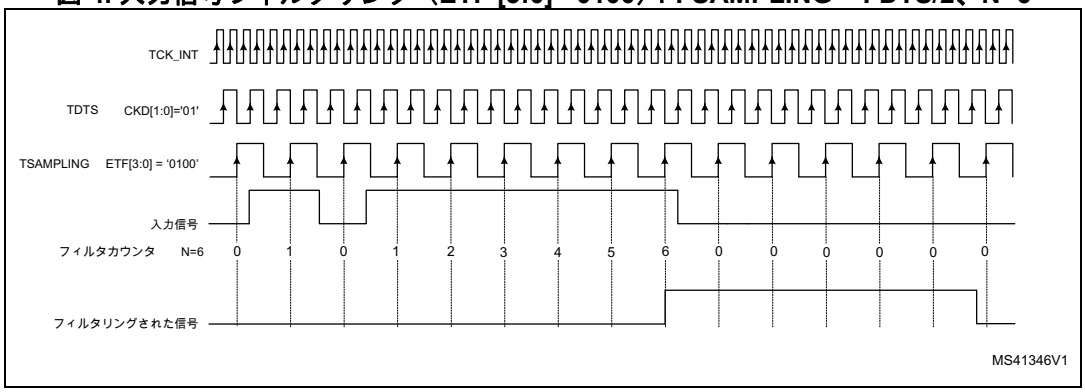
- 特定のタイマ入力に対するフィルタリングステージの設定。たとえば、ETR 入力フィルタリングステージは、TIMx_SMCR レジスタの ETF[3:0] 制御ビットフィールドを使用して設定します。入力フィルタリングステージを設定するとは、サンプリングクロックソースの選択、サンプリングクロック周波数の設定、および有効なパルスの最短期間（すでに設定済みのサンプリングクロックのクロックサイクル単位）の設定を行うことを意味します。
- サンプリングクロックソースとして FDTS クロックソースを使用する場合、フィルタリングステージの有効なパルスの最短期間は、TIMx_CR1 レジスタの CKD[1:0] 制御ビットフィールドに書き込まれている値の影響を受けます。FDTS クロック信号はタイマクロック信号から生成されます。これら 2 つの信号の比率を CKD[1:0] 制御ビットフィールドでセットします。

サンプリングのクロックソースとして、タイマクロック信号 FCK_INT または FDTS クロックソースのどちらか一方を使用できます。

図 4 は、タイマ入力 ETR のフィルタリングステージを有効にした実例を示します。この具体例では、次のように設定されています。

- タイマクロック周波数 FCK_INT = 1MHz。
- CKD [1:0] = ‘01’。FDTS クロック信号周波数は、タイマクロック信号周波数の 1/2 であることを意味します。Fdts=Fck_int/2=500KHz。
- ETF [3:0] = ‘0100’。これは、フィルタのサンプリングクロックとして FDTS クロック信号が選択され、その周波数が 1/2 に分周されることを意味します。また、ETR 入力の有効なパルスは、サンプリングクロック 6 サイクル長以上であることも意味します。
- この例の場合、6xTsampling=6x1/250kHz=24µs より短いタイマ ETR 入力のパルス信号は受け付られません。

図 4. 入力信号フィルタリング（ETF [3:0]= 0100）：FSAMPLING = FDTS/2、N=6



1.4.2 タイマレジスタのプリロード機能

STM32 のタイマペリフェラルにおけるプリロード機能とは、いくつかのタイマレジスタまたはいくつかのタイマ制御ビットフィールドを複製することを指します。タイマのいくつかのレジスタといくつかの制御ビットフィールドの内容は、タイマチャネルによって出力される波形に直接影響を及ぼすので、これらのレジスタと制御ビットフィールドの内容の更新は、新しいカウントサイクルの開始に続いてトリガされるタイマ「更新イベント」と完全に同期する必要があります。このような正確な同期は、これらのレジスタと制御ビットフィールドがプリロードされない限り実現不可能です。

プリロード機能を備えているタイマレジスタには、次の 2 つの物理的なインスタンスが存在します。

- アクティブレジスタインスタンス（別名シャドウレジスタインスタンス）：その内容は、タイマペリフェラルのロジックでタイマチャネル出力波形を生成するために使用されます。
- プリロードレジスタインスタンス：該当するレジスタのプリロード機能が有効な場合にソフトウェアがアクセスするレジスタインスタンスです。

図 5 に示すようにプリロード機能が無効な場合は、主に次の 2 つの特徴があります。

- プリロードレジスタインスタンスは存在しないように見えます。
- ソフトウェアによる該当するレジスタへの書込みアクセスは、アクティブレジスタインスタンスで実行されます。

図 6 に示すようにプリロード機能が有効な場合は、主に次の 2 つの特徴があります。

- 該当するレジスタへのアクセスはプリロードレジスタインスタンスで実行され、アクティブレジスタインスタンスの内容は変更されません。
- タイマペリフェラル内部で「更新イベント」が生成されるとすぐに、プリロードレジスタインスタンスの内容がアクティブレジスタインスタンスに転送されます。プリロードレジスタの内容は、「更新イベント」と完全に同期して転送されます。言い換えると、プリロードレジスタの内容は、出力波形の新しいサイクルに対応する新しいカウントサイクルと完全に同期しています。

参考資料

図 5. タイマチャネルレジスタのプリロードメカニズム - 無効な場合

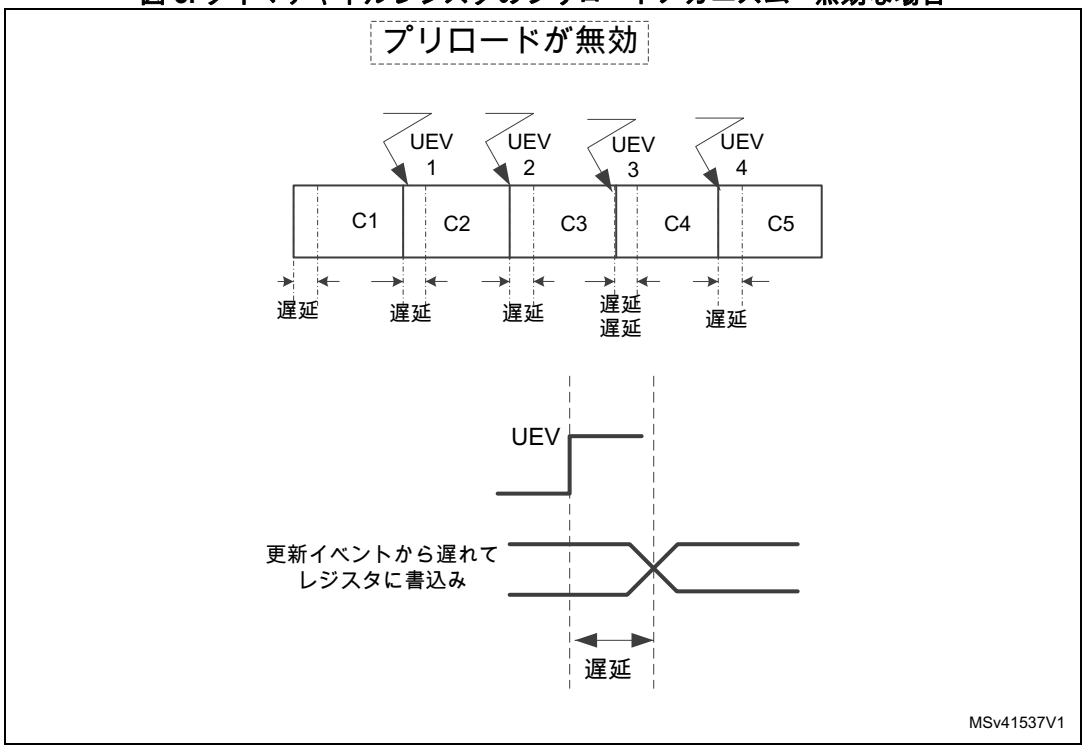
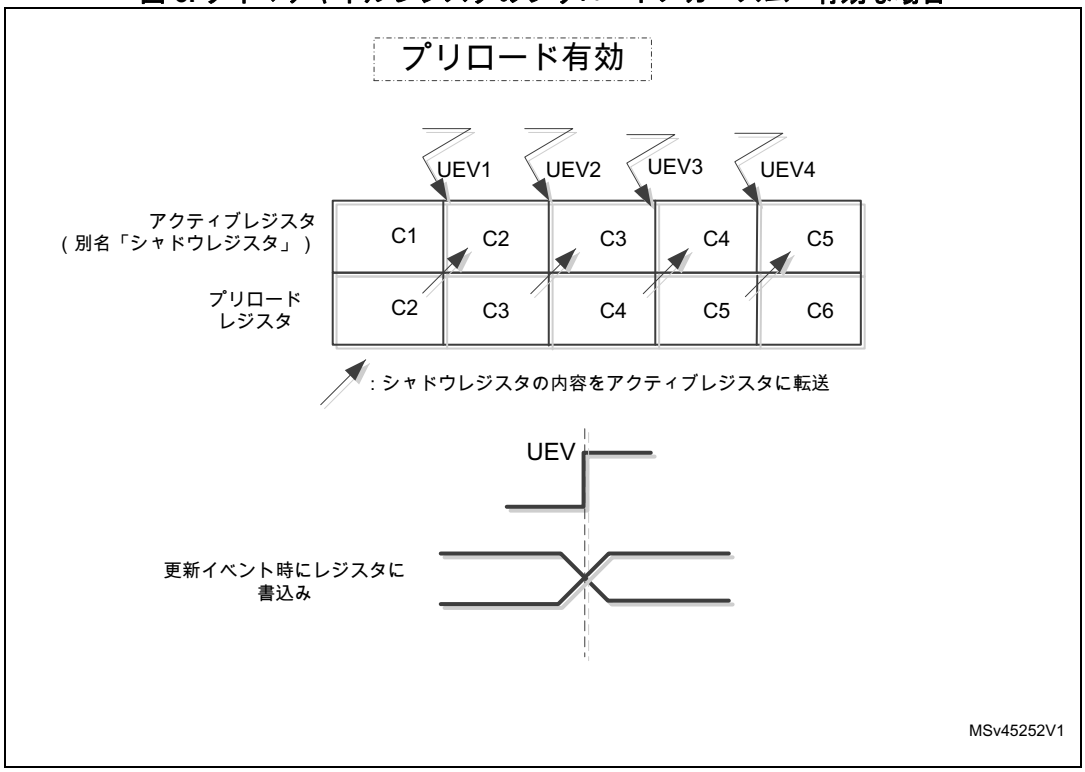


図 6. タイマチャネルレジスタのプリロードメカニズム - 有効な場合



プリロード機能は、以下で使用できます。

- 自動再ロードタイマレジスタ（TIMx_ARR）
- タイマプリスケーラレジスタ（TIMx_PSC）（無効にはできません）
- タイマチャネルレジスタ（TIMx_CCRy）
- TIMx_CCER タイマレジスタの CCxE および CCxNE 制御ビットフィールド
- TIMx_CCMRn タイマ/レジスタの OCxM 制御ビットフィールド

特定のタイマチャネルで PWM 信号を出力する場合、プリロード機能は非常に関心の持てる機能です。タイマカウンタ値と TIMx_CCRy タイマチャネルレジスタ値の連続的な比較に応じてチャネル出力レベルが決まります。タイマチャネルレジスタが変化すると、すぐにタイマチャネル出力レベルの変化が誘導されるのはこのためです。

したがって、PWM 期間の途中でチャネルレジスタに直接書き込むと、誤った波形が生成される可能性があります。この問題を克服するには、該当するタイマチャネルレジスタでプリロード機能を有効にする必要があります。プリロード機能が有効な場合、あらゆる書込みアクセスはチャネルのプリロードレジスタインスタンスに対して行われ、チャネルのアクティブレジスタインスタンスの内容は変わりません。進行中の PWM 期間に対する動揺はありません。

タイマ内部で「更新イベント」が生成されるとすぐに、プリロードレジスタインスタンスの内容がアクティブレジスタインスタンスに転送されます。アクティブレジスタインスタンスは、次の PWM 期間中に比較演算を実行するため、および新しいデューティサイクル比を定義するために使用されます。

要約すると、プリロード機能は、タイマチャネル出力に直接影響を及ぼすタイマのレジスタおよび制御ビットフィールドにアクセスする間、特に波形出力サイクルの途中に、誤った波形が生成されないことを保証します。

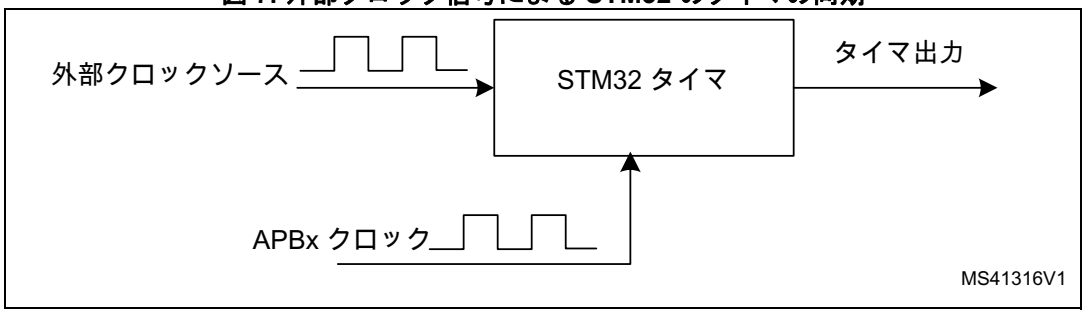
2 外部クロックソースによるタイマへのクロック供給

2.1 概要

STM32 のタイマペリフェラルには、外部ソースのクロックからクロック供給できますが、これは APB（アドバンストペリフェラルバス）クロックが不要であることを意味するのではありません。STM32 のタイマペリフェラルは、外部クロック信号をそれ自身のコアクロック（APB クロック）と同期させます。生成される同期クロック信号はタイマのプリスケーラに供給され、プリスケーラからさらにタイマのカウンタに供給されます。

STM32 のタイマペリフェラルは、そのタイムベースを常時更新し続けるために、2 つのクロックソースを必要とします（図 7 を参照）。外部クロックソースの周期は、タイマペリフェラルのタイムベースの更新に使用する時間単位です。

図 7. 外部クロック信号による STM32 のタイマの同期

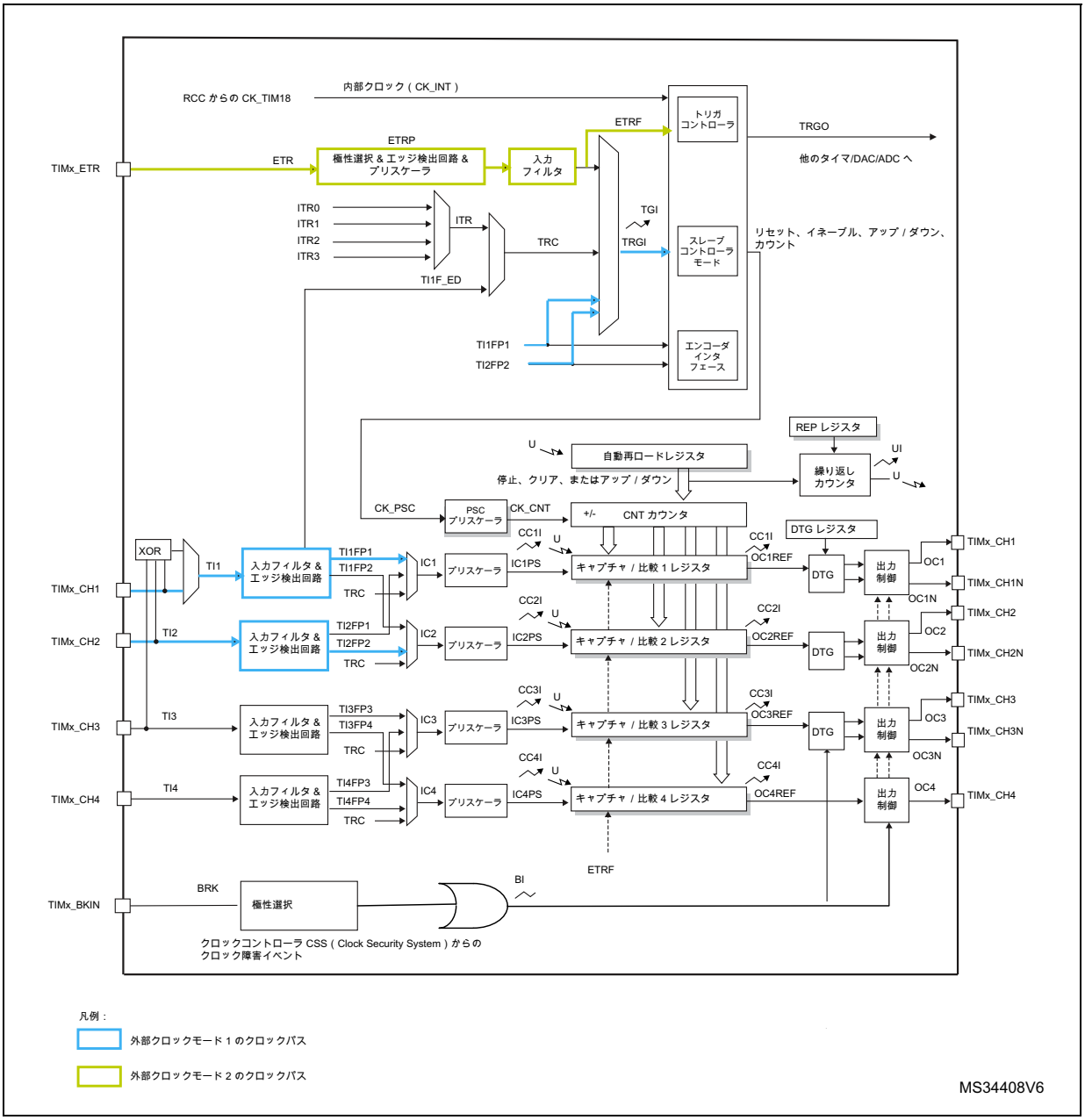


STM32 のタイマを同期（または外部からクロック供給）させるには、次の 2 つの方法があります。

- 外部クロックソースモード 1 : 外部クロック信号をいずれかのタイマチャネル入力 TIx 入力に供給します。
- 外部クロックソースモード 2 : 外部クロック信号をタイマ ETR 入力（実装され、使用可能な場合）に供給します。

次の図 8 は、両方の外部クロックソースモードのクロックパスを示します。

図 8. 外部クロックソースモードのクロックパス



2.2 同期ブロック

外部クロックソースによるタイマペリフェラルへのクロック供給のモードについて説明する前に、STM32 のタイマペリフェラルに実装されている、外部信号処理時の同期メカニズムについて説明することが重要です。外部信号は、タイマペリフェラルの外部からの信号です。外部信号は、タイマペリフェラルで使用されているクロック信号とは異なるクロック信号と同期していたり、完全に非同期な場合があります。

タイマペリフェラルは、完全に非同期な可能性がある外部信号を処理して、その出力状態や波形を必要に応じて調整する必要があります。タイマペリフェラルはさらに、特定のタイマ入力の信号の状態

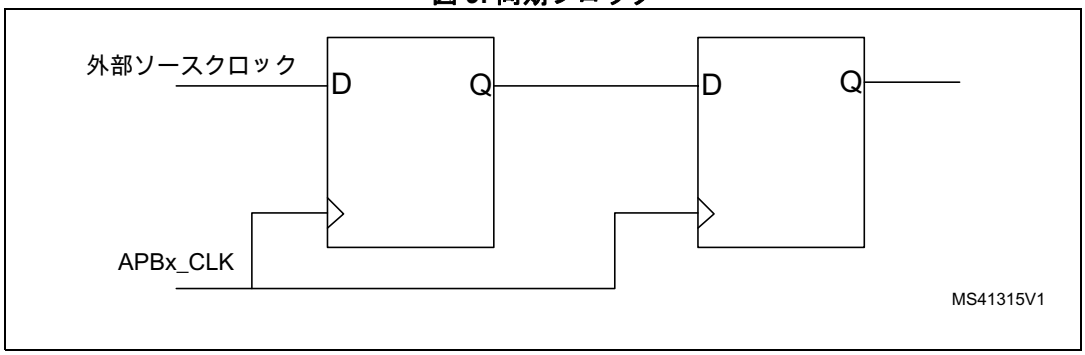
が変化した（信号がトグルした場合など）時点のタイムスタンプをソフトウェアに通知することが可能である必要があります。

タイマペリフェラルは、完全に非同期な信号を処理する場合、まずその信号を自身のクロック信号に再同期させる必要があります。たとえば、まずタイマのコアロジックのクロック信号に再同期させてから、生成される信号をタイマペリフェラルの他のサブブロックに供給する必要がある場合があります。このアクションにより、タイマのコアロジックでメタスタビリティの問題が発生するのを防ぎます。

図 9 は、タイマ入力に供給される外部信号の同期に使用する同期回路の概要図を示します。同期回路は主に、カスケード接続された 2 つの D フリップフロップで構成されます。これらは、タイマのコアクロック信号からクロック供給されます。外部信号は 1 段目の D フリップフロップ入力に供給され、2 段目の D フリップフロップの出力から同期した信号が得られます。この同期回路で、少なくともタイマのコアクロック 2 サイクル分、最大で 3 サイクル分の遅延が発生します。

「まず同期する」方法は、タイマの ETR 以外のすべての入力に組み込まれています。ETR では、まずプリスケーラがあり、その後で再同期します。

図 9. 同期ブロック



次の式に示すように、入力信号の周波数は、タイマのコアクロック信号の周波数の 1/3 より低くなければなりません。

$$Freq_{TIMCLK} \geq 3 \times Freq_{inputsignal}$$

2.3 外部クロックソースモード 1

外部クロックソースモード 1 が有効な場合、TRGI 内部タイマ信号に送ることができる信号は、タイマカウンタへのクロック供給にも使用できます。図 8 は、タイマカウンタに使用できるクロックソースを示します。

- ETRF 入力信号：プリスケール、同期、およびフィルタリングを適用した後の ETR 信号。
- タイマペリフェラル間の同期信号（ITR 入力）
- タイマチャネル 1 の出力である TI1FD 信号。ただし、信号の両エッジを検知します（タイマ入力 1 が遷移するたびにパルスが生成されます）。
- TI1FP1 と TI2FP2 の各入力信号は、それぞれ TI1 と TI2 の各タイマ入力に同期、フィルタリング、およびプリスケールを適用した信号です。

このセクションでは、外部ソースのクロックがいずれかのタイマ入力経由でタイマに供給されている場合、言い換えると ETR、TI1、または TI2 のタイマ入力経由でタイマにクロックが供給されている場合の、外部クロックモード 1 についてのみ説明します。

ETR 入力によるタイマ供給は外部クロックソースモード 2 の代替設定なので、これについてはこのセクションでは詳しく説明しません。対応するリファレンスマニュアルには、この代替クロック供給を有効にする適切な設定が記載されています。

クロックソースとしてのタイマ入力 T1 および T2

外部クロックモード 1 が有効な場合、タイマカウンタへのクロック信号供給に使用できるのは、TI1 入力と TI2 入力だけです。タイマにすでに 4 つのチャネルが内蔵されている場合、このモードでは、TI3 入力と TI4 入力はタイマにクロックを供給できません。

TI1 または TI2 のタイマペリフェラル入力に供給されるクロック信号は、タイマカウンタに到達する前に、まず調整されます。入力調整は、外部信号同期ステージおよびプリスケーリングとフィルタリングのステージで行われます。

入力フィルタとプリスケーラは設定可能です。対象タイマ入力に関連付けられているチャネルは、入力チャネルとして設定する必要があります。次に、入力チャネルの設定に使用するのと同じ制御ビットフィールドを使用して、外部クロックソースの入力を設定します。

外部クロック信号の入力として TI1 入力または TI2 入力を設定する通常のシーケンスを次に示します。

1. 外部クロック信号を供給するタイマ入力に関連付けられているチャネルを入力チャネルとして設定します。キャプチャ /比較選択（CCxS）制御ビットフィールドに 00 以外の任意の値を書き込んでもタイマチャネルは入力モードにセットされますが、設定する正しい値は CCxS = 01 です。

注： **CCxS のビットフィールドが書込み可能になるのは、関連付けられているタイマチャネルが無効な場合のみです（たとえば、TIMx_CCER レジスタでそのチャネルに対応する有効/無効制御ビットフィールドがリセットされている（CCxE = '0'）場合）。**

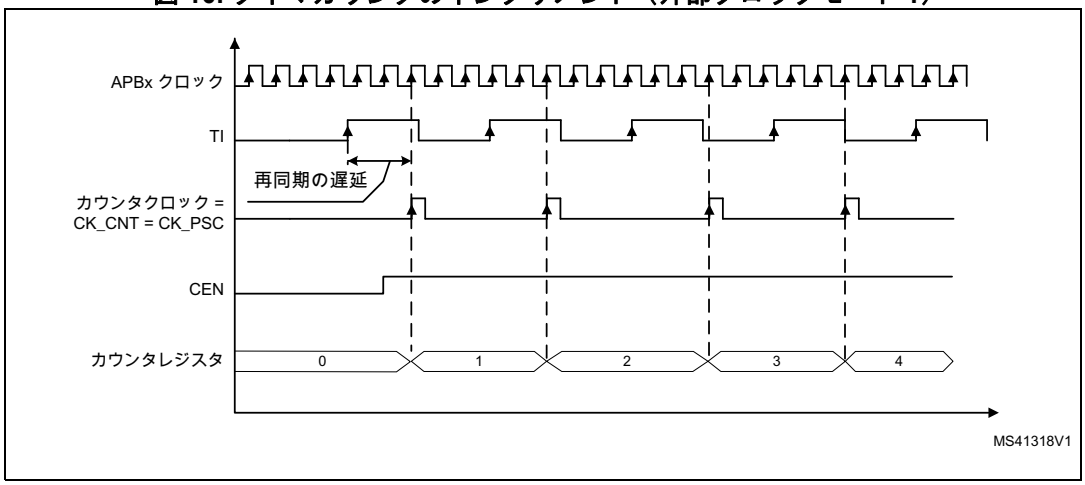
2. アクティブエッジ極性を設定します。一部の STM32 マイクロコントローラ（STM32F2 マイクロコントローラなど）は、立ち下がり、立ち上がり、または両方のエッジを検知する入力を備えたタイマを内蔵しています。他の STM32 ファミリ（STM32F1 マイクロコントローラなど）は、立ち上がりまたは立ち下がりのどちらかのエッジのみを検出する入力を備えたタイマを内蔵しています。各 STM32 マイクロコントローラのリファレンスマニュアルには、簡易タイマ入力極性を設定する極性制御ビットフィールドに書き込む適切な値が指定されています。たとえば、STM32F30x マイクロコントローラのタイマ入力で両エッジを検出させるには、該当するチャネルの極性制御ビットフィールドを CCxP = 1 および CCxNP = 1 と設定する必要があります。
3. 必要な場合、該当するチャネルに関連付けられている入力フィルタを、指定された値より短いパルスの場合、受け付けられないように設定します。入力パルスを拒否するかどうかの閾値は、使用するタイマチャネルに関連付けられている TIMx_CCMRx レジスタの ICxF[3:0] ビットフィールドで設定されます。
4. タイマ入力を必要な設定で明確に設定したら、調整された入力信号をタイマカウンタにリダイレクトします。このステップは、TIMx_SMCR レジスタのトリガ選択制御ビットフィールド TS[2:0] に適切な値を書き込むことによって実行されます。たとえば、外部クロック信号入力として TI1 タイマ入力を使用している場合、適切な設定は TS[2:0]=101 です。
5. 最後に、外部クロックソースモード 1 を有効にします。これは、TIMx_SMCR レジスタのスレーブ/マスタ選択（SMS）ビットフィールドに適切な値を書き込むことによって実行します。このビットフィールドの幅は、STM32 マイクロコントローラのファミリによって異なります。リファレンスマニュアルには、各クロックモードを有効にするために SMS ビットフィールドに書き込む適切な値が指定されています。たとえば、STM32F1 シリーズでは SMS ビットフィールドは 3 ビット幅で、設定する適切な値は SMS[2:0] = 111 ですが、STM32F30x マイクロコントローラでは SMS ビットフィールドは 4 ビット幅で、設定する適切な値は SMS[3:0] = 0111 です。
6. TIMx_CR1 レジスタの CEN 制御ビットフィールドをセットして、タイマカウンタを有効にします。

図 10 は、タイマペリフェラルがアップカウントに設定されている場合のタイマカウンタの典型的なカウントシーケンスを示します。CEN 制御ビットフィールドをセットしたとき、カウンタの内容は null であり、タイマペリフェラルはモード 1 で外部から TI1 入力経由でクロック供給されます。

この例では、設定されているアクティブエッジは立ち上がりエッジであり、その場合、フィルタとプリスケーラの制御ビットフィールドはデフォルト値（リセット後のステータスなど）のまま維持されます。図 10 は、TI1 タイマ入力の再同期ステージの影響も示しています。

TI1 タイマ入力のクロック信号の立ち上がりエッジと、タイマカウンタ CK_CNT に供給されている内部カウント信号の立ち上がりエッジの間で遅延が発生しています。この例では、TIMx_PSC タイマプリスケーラレジスタの内容は null なので、CK_CNT 信号は CK_PSC 信号と同じです（CK_CNT = CK_PSC）。

図 10. タイマカウンタのインクリメント（外部クロックモード 1）



再同期ステージがあるので、次の式に示すように、外部クロックソースの周波数は、タイマの周波数の 2 倍より低くなければなりません。

$$TIMx_{CLK}freq \geq 3 \times ITxfreq$$

2.4 外部クロックソースモード 2

外部クロックソースモードが有効な場合、ETR タイマ入力経由で供給されるクロック信号のアクティブエッジが検出されると、タイマカウンタが更新されます。特定の STM32 ファミリのタイマペリフェラルの一部では、ETR 入力はマイクロコントローラの外部に配線されない場合があります。

一部の ETR では、入力がマイクロコントローラ IO にオルタネート機能として割り当てられていません。他のいくつかのタイマペリフェラルでは、ETR タイマ入力はチャネル 1 タイマ入力と多重化されています。

外部クロックソースモード 2 を使用する場合、モード 1 と比較して、主な利点として、内部タイマのコアクロック（APB バスクロックなど）の周波数と等しいか、またはそれより高い周波数のクロック信号を外部から供給できることがあります。

これは、タイマコアクロックの頻度よりも高い頻度でタイマカウンタを更新できる（例：タイマがアップカウントモードで設定されている場合に頻度を増やす）ことを意味するのではありません。

ETR タイマ入力は、再同期ステージより前にプリスケーラステージを備えている唯一のタイマ入力です。このプリスケーラステージは完全非同期であり、非同期入力信号周波数の 1/8 に分周できます。

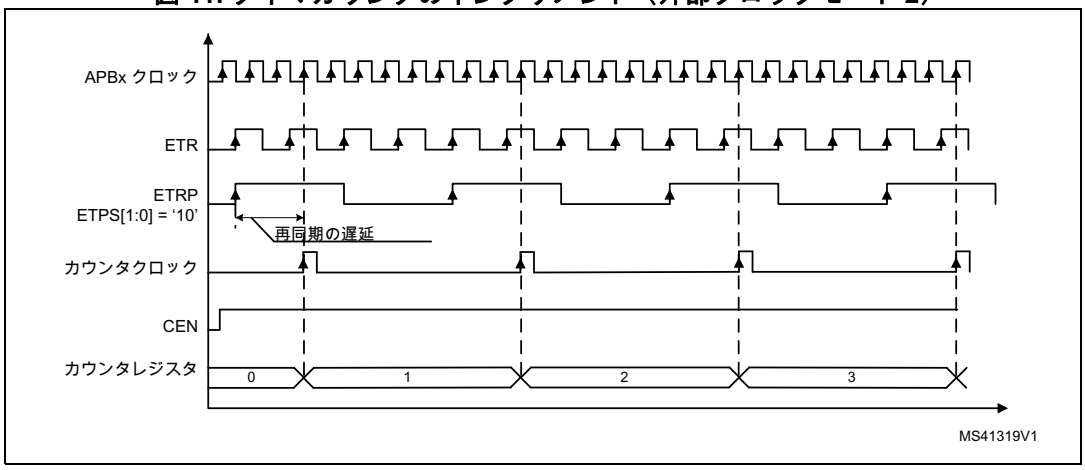
非同期プリスケーラステージの出力は、ETRP 信号と呼ばれ、再同期ステージに送られてからタイマカウンタに供給されます。再同期回路の出力は、ETRF 信号と呼ばれます。ETRP 信号には、タイマに供給される他の非同期信号と同じ制約があります。すなわち、その周波数は、タイマ内部のコアクロックの周波数の 1/3 より低くなければなりません。

外部クロックソースモード 2 のこの機能は、多くの実用的なケースで非常に関心の持てる機能です。たとえば、特定のセンサで生成されるパルスの数をカウントする必要があるものの、そのセンサの出力周波数がタイマ内部のコアクロックの周波数より高い場合があります。その場合、外部信号入力周波数に対する再同期ステージの制約に適合するように指定された比率で、入力信号の周波数を低減するように、プリスケーラを設定する必要があります。カウントしたパルス数を解釈するときは、信号周波数が指定された比率で分周されていることを考慮する必要があります。

図 11 は、タイマペリフェラルが外部クロックソースモード 2 に設定され、タイマにその ETR 入力経由でクロック信号が供給されている例を示しています。この例では、外部クロック信号の周波数は、タイマ内部のコアクロック（APB バスクロック）より低くなっています。ETR 非同期プリスケーラは、TIMx_SMCR レジスタの外部トリガプリスケーラ制御ビットフィールドを ETPS = '10' にセットすることによって、入力信号周波数を 1/4 に分周するように設定されています。

また、図 11 は、プリスケーラの出力（ETRP 信号）と、タイマカウンタの更新に使用するカウンタクロック信号 CK_CNT の間の遅延を示しています。この遅延は、再同期ステージによって挿入されます。

図 11. タイマカウンタのインクリメント（外部クロックモード 2）



タイマペリフェラルを外部クロックソースモード 2 で設定する標準的な推奨シーケンスを次に示します。

1. 入力クロック信号の最高周波数を評価して、非同期プリスケーラ比を選択します。外部クロック信号周波数がタイマ内部コアクロックの 1/3 より低い場合、プリスケーラの使用はオプションであり、使用しなくてもかまいません。非同期プリスケーラを使用しない場合、ETPS 制御ビットフィールドをリセットする必要があります（ETPS ビットフィールドはデフォルトではリセットされています）。
2. 必要であれば、フィルタリングステージを有効にして、特定の閾値より短いクロック信号パルスを受け付けないようにします。STM32 のタイマペリフェラルの外部入力にフィルタ機能をセットアップする方法の詳細については、13 ページのセクション 1.4.1 : フィルタリングステージを参照してください。
3. 外部クロック信号のアクティブエッジを設定します。外部クロックソースモード 2 エッジを検知する ETP 制御ビットをセットします。デフォルトで、およびリセット後は ETP 制御ビットはリセットされており、外部クロック信号の立ち上がりエッジが有効になります（たとえば、外部クロック信号の立ち上がりエッジはタイマカウンタ更新をトリガします）。
4. 外部クロックイネーブル制御ビット ECE = 1 をセットして、外部クロックソースモード 2 を有効にします。
5. 最後に TIMx_CR1 レジスタ内のカウンタイネーブル制御ビットをセットして、タイマカウンタを有効にする必要があります。

注： **ETR 入力は、外部クロックソースモード 1 が設定されている場合に外部クロック信号の入力としても使用できます。外部クロックソースモード 1 と 2 の両方を同時に有効にして、両方のモードで同時に ETR 入力を使用することが可能です。その場合、外部クロックソースモード 2 が優先されて、タイマカウンタへのクロック供給に使用されます。**

2.5 外部クロックソースのモード 1 とモード 2

外部クロックソースモード 1 と 2 はどちらも同じ機能を持つように見えますが、慎重に調べると、いくつかの特定の特性において異なり、このため、それぞれが実用的ケースの異なる範囲に適していることがわかります。

モード 1 と 2 の主な違いを次に示します。

- 外部クロックソースモード 1 を使用して、外部クロック信号の両エッジでタイマカウンタを更新できます。外部クロックソースモード 2 を使用するときは、これは不可能です。
- 外部クロックソースモード 2 を使用することによって、外部クロックソースでタイマペリフェラルにクロックを供給できます。その場合、さらに、同時にタイマペリフェラルを適合するタイマスレーブモードの 1 つに設定することが可能です。たとえば、一定期間内に特定のセンサによって生成されるパルスの数を繰り返しカウントする必要がある場合、次のようにします。
  - 外部クロックソースモード 2 に設定することによって､1 つ目のタイマペリフェラル（TIMy など）に、センサによって生成されるパルスの数をカウントさせます。
  - 2 つ目のタイマペリフェラル（TIMz など）を、一定間隔で繰り返しトリガ出力（TRGO）信号を生成するように設定します。
  - 1 つ目のタイマ（TIMy）をスレーブリセットモードにも設定して、リセットのトリガとして 2 つ目のタイマ（TIMz）の TRGO 出力信号を使用します。

タイマペリフェラルに一定間隔で TRGO 出力信号にパルスを出力させるには、周期的なタイマの「更新イベント」が生成されるように TIMx_ARR タイマレジスタを特定の値にセットする方法があります。

タイマのマスタモードが「値を更新」するようにセットされている（たとえば TIMx_CR2 レジスタの MMS[2:0] ビットフィールドが ‘010’ にセットされている）場合、タイマの「更新イベント」によりタイマの TRGO 信号でパルスをトリガできます。

スレーブモード選択制御ビットフィールドに適切な値をセットする（たとえば TIMx_SMCR レジスタで SMS[2:0] = 100）ことによって、タイマをリセットスレーブモードに設定できます。

リセットスレーブモードの適切なトリガは、トリガ選択制御ビットフィールド TS[2:0] に適切な値を設定することによって確実に選択されます。

TS[2:0] 制御ビットフィールドに書き込む値は、1 つ目のタイマ（TIMy）のどの ITR タイマ入力が 2 つ目のタイマ（TIMz）の TRGO 出力に内部的に接続されているのかによって決まります。各 STM32 マイクロコントローラのリファレンスマニュアルには、タイマペリフェラル間のすべての内部相互接続がリストされています。

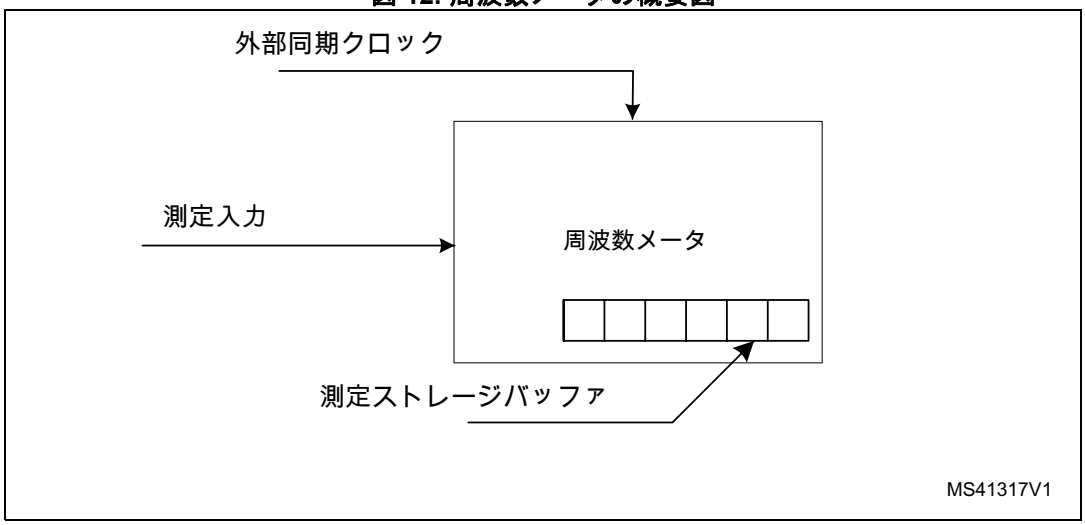
2.6 アプリケーション：ETR タイマ入力で外部クロックソースを使用したタイマへのクロック供給

このアプリケーションでは、外部クロックソースを使用してタイマペリフェラルにクロックを供給する使用例について説明します。このアプリケーションは、外部クロックソースモード 2 で開発されていますが、簡単に外部クロックモード 1 を使用するように作り直すことができます。

このアプリケーションは、周波数範囲を小さくし、精度を下げたカウンタのような周波数カウンタの基本的実装です。このアプリケーションは、次の前提条件の下で開発されました。

- マイクロコントローラは、クロックソースとして、内部のハイスピードオシレータ（HSI オシレータ）を使用しています。
- このアプリケーションは、10 MHz の正確なクロックソースが使用可能な環境で使用されます（たとえば、いくつかの機器が 10 MHz の較正出力を提供する測定ベンチ内）。
- このアプリケーションは、2 つのパートに分けて実証されます。1 つ目のパートでは、周波数メータアプリケーションにクロックを供給するのは内部 HSI オシレータからのみです。2 つ目のパートでは、周波数カウンタアプリケーションは、タイマの ETR 入力に供給される、より正確な 10 MHz クロックソースを利用します。

図 12. 周波数メータの概要図



タイマペリフェラルを外部クロックと同期することが測定精度に及ぼす影響を強調するために、図 12 に示すように、アプリケーション測定入力にリファレンスクロック信号を入れます。

次に、外部クロック信号が供給される場合とされない場合の両方のケースについて、入力信号の周波数の一連の測定を実行する必要があります。幅広く行った測定は、外部クロックソースを使用することで、この基本的な周波数メータアプリケーションの性能が向上するかどうかを示す優れた指標となります。

このアプリケーションを実行するには、次のハードウェアセットアップが必要です。

- STM32F302 Nucleo ボード（NUCLEO-F302R8）
- Nucleo ボードを開発 PC に接続する USB ケーブル（この USB ケーブルでボードの電源も供給します）
- 1 つまたは複数の波形ジェネレータ

このアプリケーションのソースコードを作り直して別の設定をデバッグするには、次の一連のソフトウェアツールが必要です。

– ST-LINK デバッガをサポートする開発ツールチェーン
– 最新バージョンの STM32CubeMX ソフトウェアツール（このアプリケーションの開発時の最新リリースは v4.10.1）

TIM2 タイマは、カウンタ分解能が最も高く（32 ビット）、より多くの PPM サンプルを収集できるため、選択されています。

チャネル 2 は、ETR ピンが TIM2 タイマのチャネル 1 に割り当てられているため、入力キャプチャとして選択されています。

このアプリケーションの考え方は、CCR2 レジスタの連続する 2 つの値の差に TIM2 カウンタのインクリメント周期を乗じた値として受信信号の周期を計算した後、周波数と PPM を推定するというものです。

$$ppm = \frac{df \times 10^6}{f} \qquad with \qquad df = fm - f$$

– fm：測定周波数
– f：公称周波数

設定

システムクロックの初期化

クロックの初期化は、メインルーチンで、HAL ライブラリ初期化（HAL_Init）の直後に SystemClock_Config() 関数を使用して実行されます。

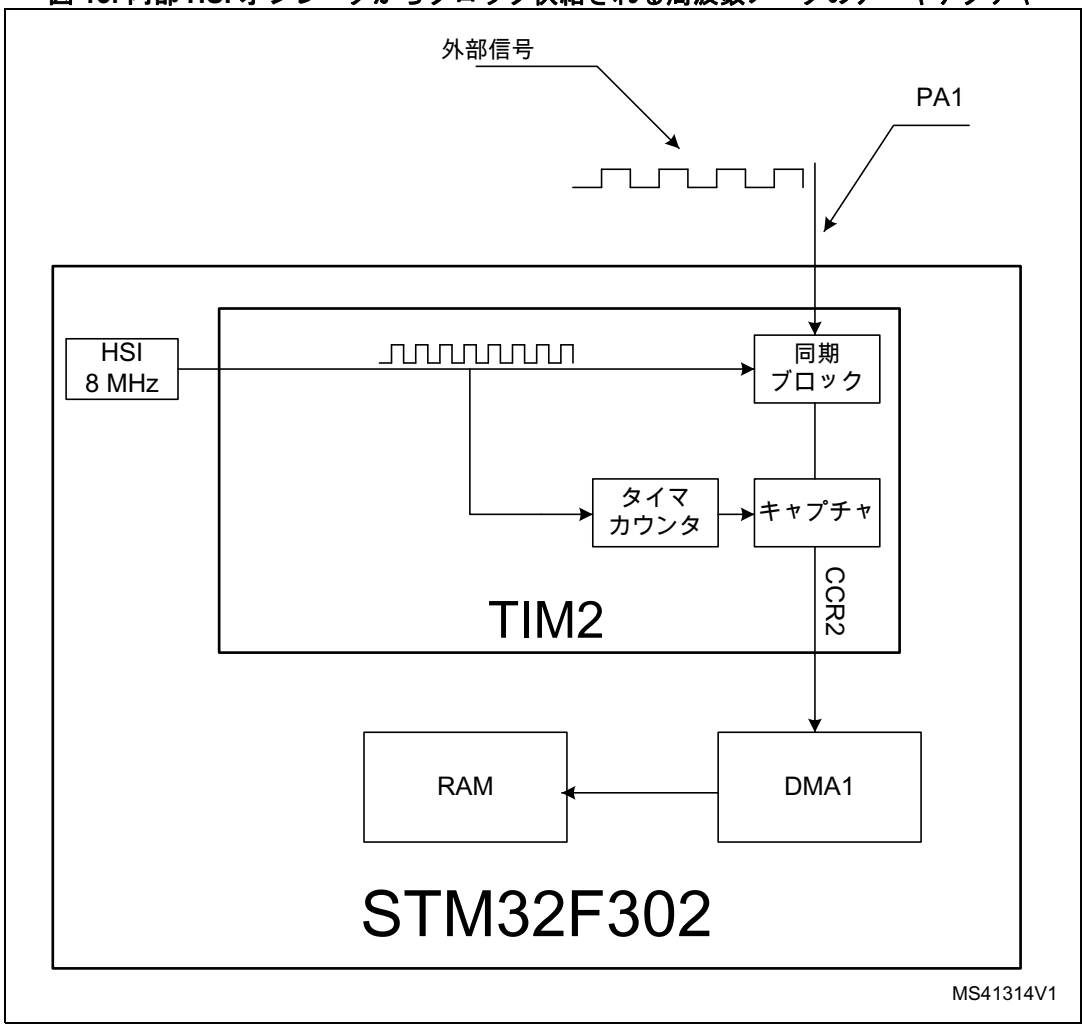
DMA の設定

DMA は、受信信号の周波数を計算するため、CCR2 の値をバッファにコピーする際に使用します。

DMA の設定を次に示します。

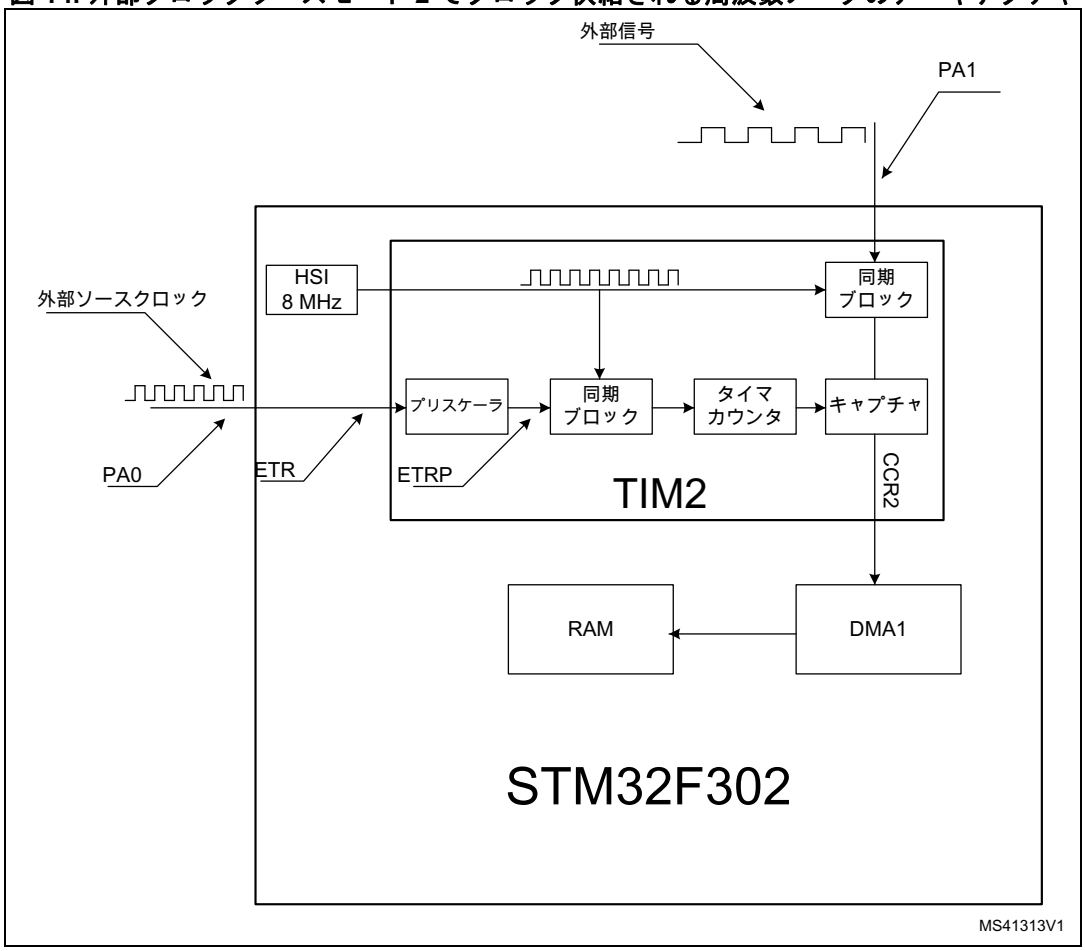
- DMA_Stream = DMA1_Stream6
- DMA_Channel = DMA_Channel_3
- 方向 = ペリフェラルからメモリ
- インクリメントメモリ = 有効
- ペリフェラルのデータの配置 = ワードアライメント
- メモリのデータの配置 = ワードアライメント
- モード = 通常
- FIFO モード = 有効

図 13. 内部 HSI オシレータからクロック供給される周波数メータのアーキテクチャ



この例では、周波数 8 MHz の内部 HSI オシレータからのみタイマにクロック供給されます。測定信号は、最初にタイマのクロック同期ブロックを通過します。次に、各立ち上がりエッジで、タイマはカウンタ値（CCR2 レジスタに格納されている）をキャプチャして、その値を DMA 経由で RAM に保存します。最後に、入力信号の周波数を導出する式を適用します。

図 14. 外部クロックソースモード 2 でクロック供給される周波数メータのアーキテクチャ



この例では、周波数 10 MHz の波形ジェネレータからの較正済み外部クロックソースによってタイマにクロックが供給されます。このクロックソースは非同期プリスケーラを通過して、クロックタイマとの同期条件を検証します。

測定信号は、最初にタイマのクロック同期ブロックを通過します。次に、各立ち上がりエッジで、タイマはカウンタ値を CCR2 レジスタにキャプチャして、その値を DMA 経由で RAM に保存します。最後に、入力信号の周波数を導出する式を適用します。

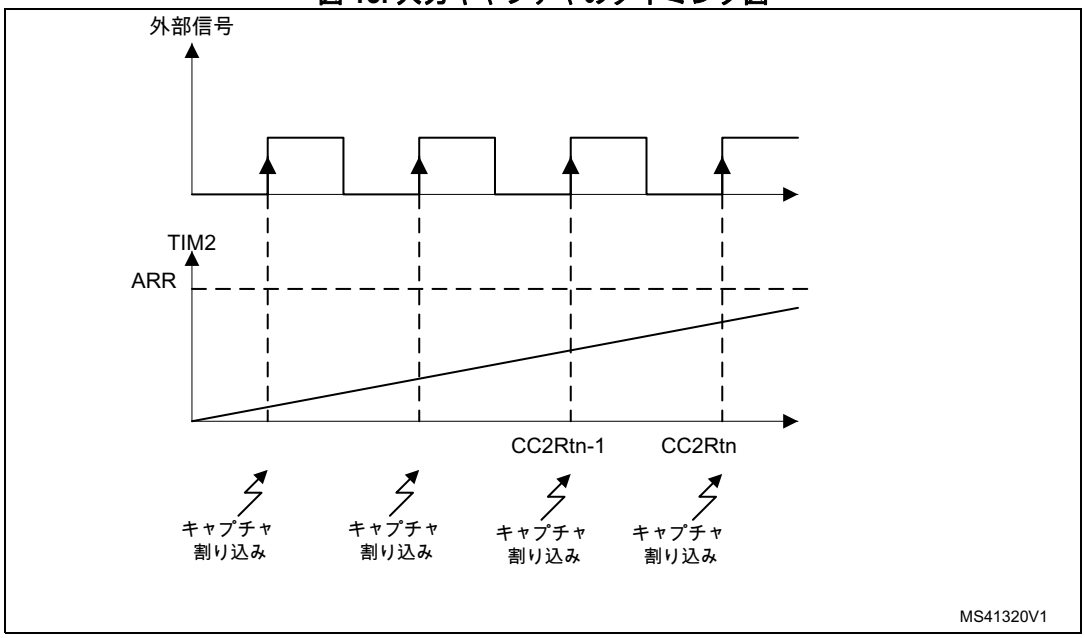
入力キャプチャ

外部信号の周期を測定するには、タイマを入力キャプチャモードで使用します。最大周波数（32 ビットタイマで測定可能）は、TIM2CLK 信号に依存します。

外部信号周波数は、直接測定されるのではなく、タイマでカウントしたクロックパルスの数から計算されます。それには、非常に正確な基準周波数が使用可能である必要があります。

タイマカウンタを有効にした後、基準信号の最初の立ち上がりエッジが発生したときに、タイマカウンタ値がキャプチャされ、CC2R に格納されます。この値は、次の立ち上がりエッジでキャプチャされる後続の値で上書きされないように DMA を使用して、バッファにも格納されます。

図 15. 入力キャプチャのタイミング図



2 つの連続する立ち上がりエッジの間の経過時間 $CC2R_{tn}$ - $CC2R_{tn-1}$ は、基準信号の 1 周期を表します。

タイマはシステムクロック（内部 RC オシレータ HSI）によりクロック供給されているので、内部 RC オシレータによって生成される実際の周波数は、基準信号を使用して次の式で表されます。

測定周波数 = ($CC2R_{tn}$ - $CC2R_{tn-1}$) * 基準周波数

誤差（Hz 単位）は、測定周波数と標準値の差の絶対値として計算されるので、周波数誤差は次の式で表されます。

誤差（Hz）= | 測定周波数 - 標準値 |

各トリミング値の誤差を計算した後で、ppm の推定値は次の式で表されます。

$$ppm = \frac{Error \times 10^6}{typicalvalue}$$

この例では、タイマには 8 MHz が供給されている（TIM2CLK = 8 MHz）ので、タイマカウンタがオーバーフローしないで測定可能な最小周波数は次の式で表されます。

$$f = \frac{TIM2CLK}{ARR} = \frac{8 \times 10^6}{0xFFFFFFFF} = 0.002Hz$$

入力信号は、1 KHz の波形ジェネレータで測定できます。入力キャプチャモードは次のように設定されます。

- 外部信号は TIM2 CH2 ピン（PA01）に接続
- 立ち上がりエッジをアクティブエッジとして使用
- TIM2 CCR2 を使用して周波数値を計算

入力キャプチャモジュールを使用して、入力チャネル 2 で遷移を検出した後にカウンタの値をキャプチャします。外部信号の周期を取得するには、2 つの連続するキャプチャ値が必要です。これらの 2 つの値の差を計算して、周期を求めます。

キャプチャイベントが発生すると、CC2IF (TIM2_SR レジスタ)が 1 にセットされます。DMA 機能が有効な場合、それによって DMA リクエストが生成されます。キャプチャが発生し、CC2IF フラグがセットされている場合、オーバーサンプリングフラグ CC2OF がセットされます。

立ち上がりエッジが発生すると、タイマカウンタの現在の数値（TIM2_CNT）が TIM2_CCR2 に書き込まれます。次の立ち上がりエッジまで待つと、TIM2_CNT の次のカウンタ値が記録されます。2 つのデータ値から、入力データのサイクルを計算できます。オーバーフロータイマは使用できません。

外部クロックソースモード 2 の設定

この例の 2 つ目のパートでは、この設定を維持したままで、タイマ入力クロックとして ETR ピンを使用する外部ソースモード 2 に、クロックタイマのソースを変更します。

外部信号は、次の同期条件を満たす必要があります。

$$TIM2_{Clk} \geq 3 \times f_{ETRP}$$

この例では CLK TIM2 = CLK APB1 = 8 MHz なので、次のようになります。

$$f_{ETR} = \frac{1}{3} \times 8\text{Mhz} = 2.66MHz$$

この場合、周波数 8 MHz の信号波形ジェネレータを実行して、非同期分周器で 1/4 に分周します。

$$f_{ETRP} = 2MHz \leq \frac{1}{3} \times 8\,MHz$$

TIM2 設定

```
/* TIM2 設定 */
/* TIM2 クロックを有効にします */
RCC->APB1ENR |= RCC_APB1ENR_TIM2EN;
/* 8 MHz のカウンタクロックを得られるようにタイマプリスケーラをセットします */
Prescaler = (uint16_t) (SystemCoreClock / 8000000) - 1; TIM2->SMCR =
RESET; /* SMCR レジスタをリセットします */
#ifdef USE_ETR
/* ETR プリスケーラを 4 に設定します */
TIM2->SMCR |= TIM_ETRPRESCALER_DIV4 |
/* 極性を立ち上がりエッジに設定します */
TIM_ETRPOLARITY_NONINVERTED |
/* ETR クロックソースを設定します */
TIM_SMCR_ECE;
#else /* 内部クロックソース */
/* 内部クロックソースを設定します */
TIM2->SMCR &= ~TIM_SMCR_SMS;
#endif /* USE_ETR */
TIM2->CR1 &= ~(TIM_CR1_DIR | TIM_CR1_CMS);
/* アップカウンタモードを選択します */
TIM2->CR1 |= TIM_COUNTERMODE_UP;
```

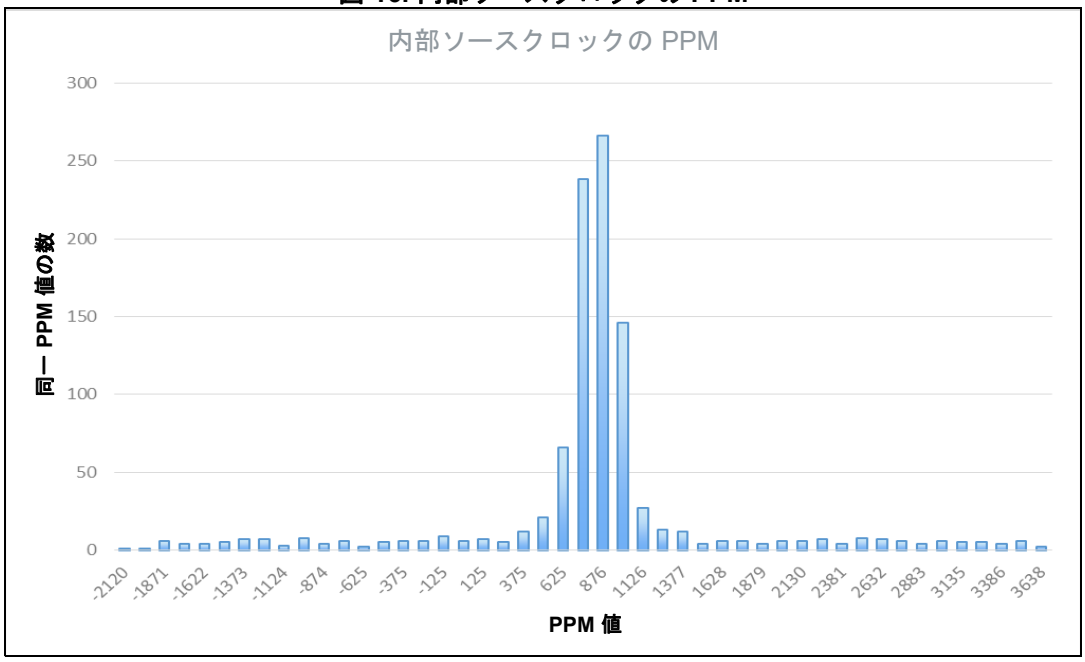
```
TIM2->CR1 &= ~TIM_CR1_CKD;
/* クロック分周を 1 にセットします */
TIM2->CR1 |= TIM_CLOCKDIVISION_DIV1;
/* 自動再ロード値をセットします */
TIM2->ARR = PERIOD;
/* プリスケーラ値をセットします */
TIM2->PSC = (SystemCoreClock / 8000000)-1;
/* 更新イベントを生成して、ただちにプリスケーラ値を再ロードします */
TIM2->EGR = TIM_EGR_UG;
TIM2->CCMR1 &= ~TIM_CCMR1_CC2S;
/* タイマ入力を IC2 に接続します */
TIM2->CCMR1 |= TIM_CCMR1_CC2S_0;
```

入力キャプチャ設定

```
/* DMA1 を有効にします */
DMA1_Channel7->CCR |=  DMA_CCR_EN;
/* TIM キャプチャ / 比較 2 DMA リクエストを有効にします */
TIM2->DIER |= TIM_DMA_CC2; /* TIM キャプチャ / 比較チャネル 2 を有効にします
TIM2->CCER |= TIM_CCER_CC2E;
/* TIM2 を有効にします */
TIM2->CR1 |= TIM_CR1_CEN;
/* 転送が完了するまで待ちます */
while ((DMA1->ISR & DMA_ISR_TCIF7) == RESET) {}
```

1000 個のサンプルを収集して対応する PPM を計算すると、次のグラフが得られます。

図 16. 内部ソースクロックの PPM

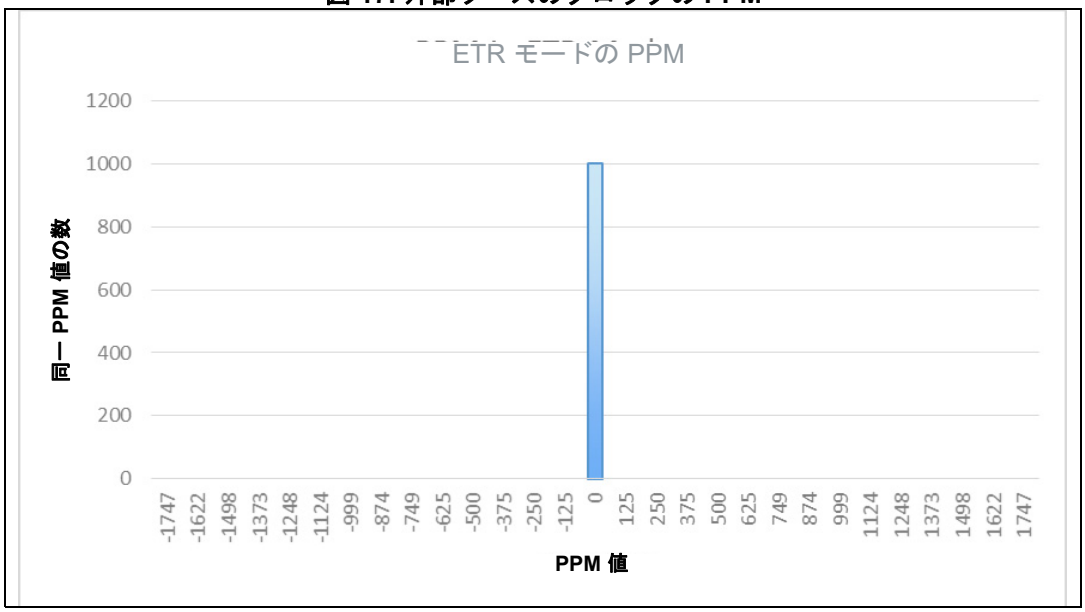


PPM 値の分布は 0 から大きく離れており、静的なオフセットと動的なオフセットがあります。静的なオフセットは、HSI の不安定さ（製造プロセスの変動）によるものです。これは決定論的原因障害であり、マイクロコントローラごとに異なります。動的なオフセットは、温度などの環境条件が原因で生じます。

実際のタイムベースオシレータ周波数と公称周波数の差から生じる誤差はそのまま測定誤差になります。この差は、タイムベースオシレータの個々の誤差すべての累積的影響です。

このアプリケーションの 2 つ目のパートで、タイマが外部クロックソースで同期されている場合、1000 個のサンプルを収集して対応する PPM を計算すると、次のグラフが得られます。

図 17. 外部ソースのクロックの PPM



測定した信号の立ち上がりエッジ間の時間はすべて同じため、PPM 値はすべてゼロになります。この場合、外部クロックソースは正しく較正されているため、測定に誤差が生じないことに注意してください。

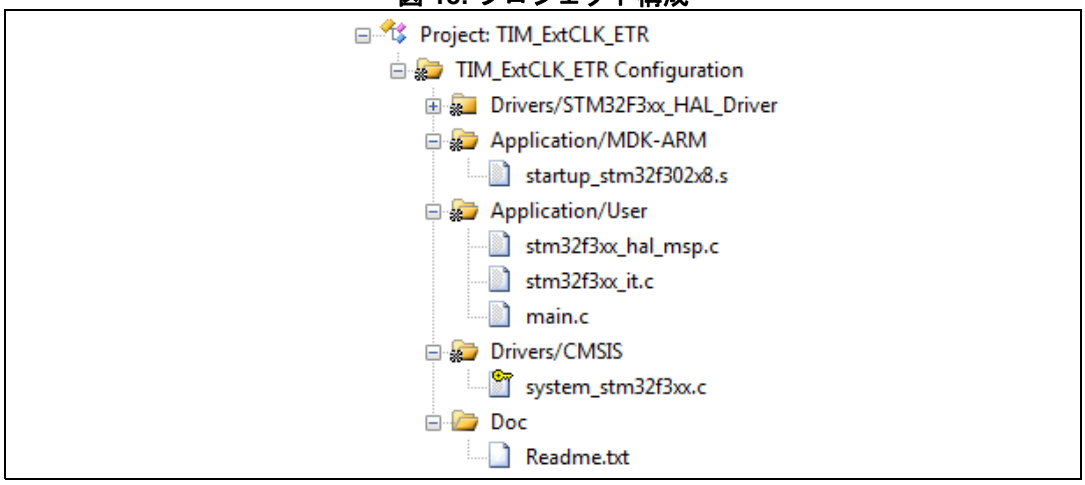
参考資料

2.7 ファームウェアの概要

ファームウェアは、Keil µvision、IAR Embedded workbench、および SYSTEM WORKBENCH で開発しました。

開発したファームウェアは ZIP ファイルで提供され、アプリケーションの中核を構成するサブディレクトリおよび .h と .c の各ソースコードファイルが含まれています。

図 18. プロジェクト構成



ファームウェアにはアプリケーションタスクのすべてのソースファイルと関連ファイルが含まれ、次のプロジェクトフォルダから構成されています。

- STM32 MCU HAL ライブラリ
- スタートアップファイル
- アプリケーションレイヤ

3 ワンパルスモードによる N パルス波形生成

3.1 概要

STM32 タイマペリフェラルのワンパルスモード（OPM）は、出力モードに設定されているタイマチャネルと組み合わせて使用できる機能です。この機能を使用すると、タイマで、PWM1 または PWM2 の出力比較モードに設定されているタイマチャネルに、プログラム可能な遅延の後でプログラム可能な幅のパルスを生成できます。

このモードは、TIMx_CR1 タイマレジスタの OPM ビットをセットすることで有効になります。OPM 制御ビットがセットされている場合、タイマ更新イベントが発生するたびに、タイマカウンタイネーブル制御ビットがリセットされて、カウンタは更新イベント時の値で停止します。アップカウントに設定されている場合、タイマカウンタレジスタは値 0 で停止します。センターアライン設定やダウンカウントモードなど、他のカウント設定の場合については、リファレンスマニュアルを参照して、カウンタが停止する値を判断してください。

タイマ更新イベントが何らかの方法でマスクされている場合は、ワンパルス動作メカニズムで CEN 制御ビットをリセットできないので、タイマカウンタは動作し続けます。タイマの出力は、設定されている波形を出力し続けます。

タイマ更新イベントの影響をマスクする方法の 1 つは、UDIS 制御ビットをセットすることです。この制御ビットの詳細については、リファレンスマニュアルを参照してください。タイマ更新イベントをマスクするもう 1 つの方法は、タイマ繰り返しカウンタレジスタの内容を 0 以外の値にすることです。繰り返しカウンタは、STM32 のすべてのタイマペリフェラルに内蔵されているわけではありません。

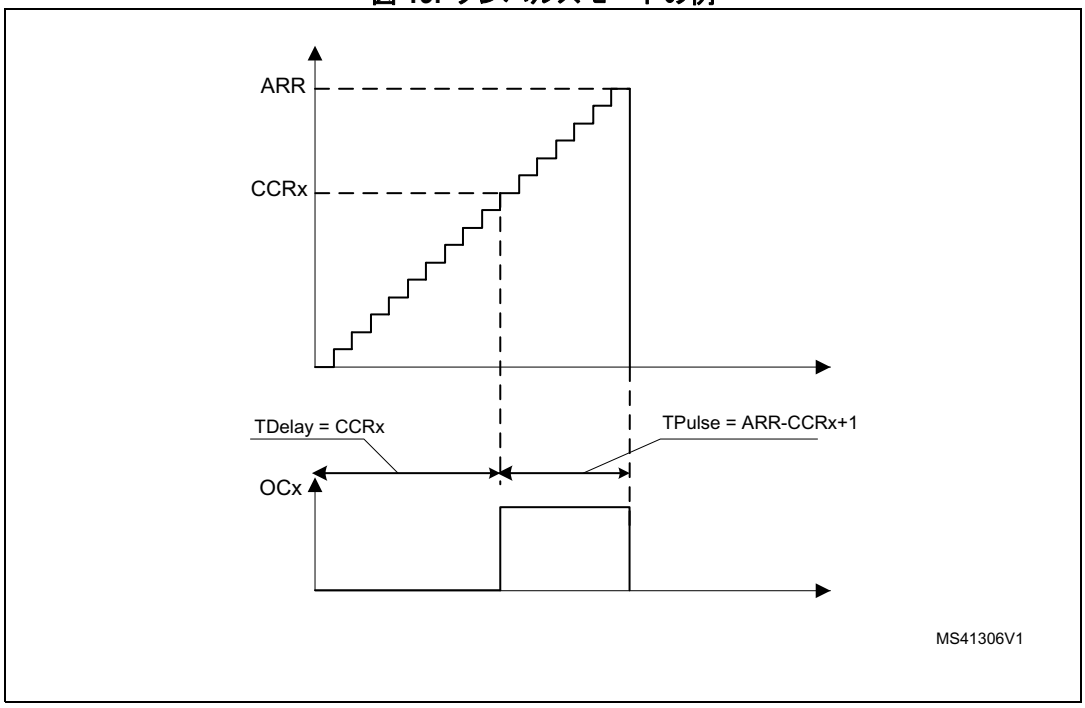
前述の条件で、（効果がマスクされた状態で）更新イベントが発生するたびに繰り返しカウンタの内容がデクリメントされることを考慮すると、この動作を使用して一連のパルスを生成できます。

たとえば、5 個の連続するパルスを生成するには、繰り返しカウンタを 4 にセットする必要があります。最初の 4 回の更新イベントは、繰り返しカウンタの内容が 0 ではないので、マスクされます。これらの最初の 4 回の更新イベントでは、CEN 制御ビットはリセットされませんが、繰り返しカウンタの内容は 0 までデクリメントされます。5 回目の更新イベントで、（繰り返しカウンタがすでに 0 なので）CEN 制御ビットがリセットされます。この方法では、タイマカウンタが無効になる前に 5 回オーバーフローして、必要な 5 個のパルスが生成されます。

デフォルトのチャネル出力極性の場合、PWM2 出力モードでは、一定時間の遅延の後に一定期間のパルスが続く、標準的なパルス波形が生成されます。極性を反転させるには、チャネル出力極性を反転させるか、PWM1 出力モードを使用する必要があります。

次の図 19 では、デフォルトのチャネル出力極性を使用する 1 つ目のシナリオを例として使用しています。出力波形は、パルス幅と遅延時間の 2 つのパラメータで特徴付けられます。

図 19. ワンパルスモードの例



T_{Delay} は、TIM_CCRx タイマキャプチャ /比較レジスタに書き込まれた値によって定義されます。

T_{Pulse} は、タイマ自動再ロードレジスタ（TIMx_ARR）に書き込まれた値と T_{Delay} 値の差によって定義されます。すなわち、TIMx_ARR の値から TIMx_CCRx の値を引いて 1 を足した値となります。

3.2 アプリケーション：ワンパルスモードによる N パルス波形生成

このアプリケーションの目標は、特定のタイマチャネル出力で、定義されている数の一連のパルスで構成される波形を生成することです。これを実現するには、STM32 のタイマペリフェラルのワンパルスモード機能と繰り返しカウンタ機能を使用します。

このアプリケーションでは、TIM1 タイマペリフェラルが必要な要件を満たしているので、これが選択されています。

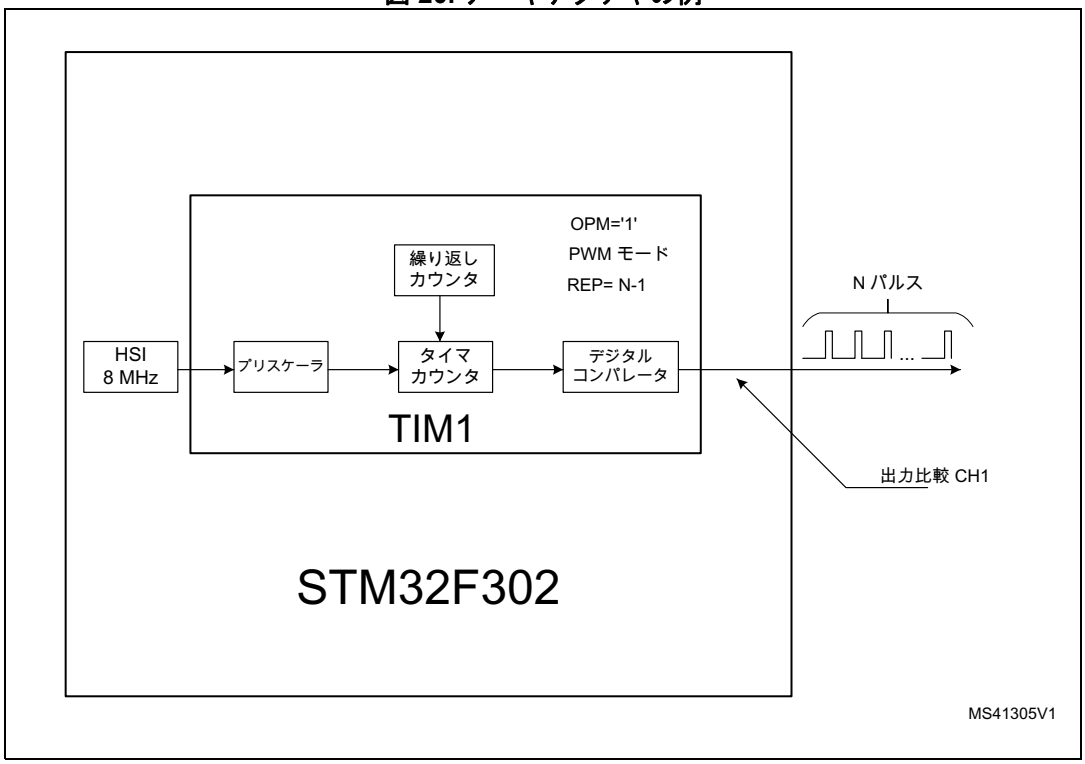
このアプリケーションを実行するには、次のハードウェアセットアップが必要です。

- STM32F302 Nucleo ボード（NUCLEO-F302R8）
- Nucleo ボードを開発 PC に接続する USB ケーブル（この USB ケーブルでボードの電源も供給します）
- 出力信号波形を視覚化するオシロスコープ

このアプリケーションのソースコードを作り直して別の設定をデバッグするには、次の一連のソフトウェアツールが必要です。

- ST-LINK デバッガをサポートする開発ツールチェーン
- STM32CubeMX ソフトウェアツール v4.10.1 以降のバージョン（このアプリケーションの開発時の最新リリースは v4.10.1）

図 20. アーキテクチャの例



このアプリケーションでは、内部 HSI オシレータからタイマにクロック供給されます。TIMx_CR1 タイマレジスタの OPM ビットをセットして、OPM モードを有効にします。出力に N パルスを生成するには、TIMx_RCR レジスタに適切な値（N-1）を設定する必要があります。

システムクロックの初期化

クロックの初期化は、メインルーチンで、HAL ライブラリ初期化（HAL_Init）の直後に SystemClock_Config() 関数を使用して実行します。

設定コードを次に示します。

ワンパルスモード設定による N パルス波形生成

```
RCC->APB2ENR |= RCC_APB2ENR_TIM1EN; /* ペリフェラルクロックを有効にします */
Prescaler = (uint16_t) (SystemCoreClock / 1000000) - 1;
/* 1MHz のカウンタクロックを得られるようにタイマプリスケーラをセットします */
TIM1->CR1 &= ~(TIM_CR1_DIR | TIM_CR1_CMS);  /* アップカウンタモードを選択します */
TIM1->CR1 |= TIM_COUNTERMODE_UP;
TIM1->CR1 &= ~TIM_CR1_CKD
TIM1->CR1 |= TIM_CLOCKDIVISION_DIV1; /* クロック分周を 1 にセットします */
TIM1->ARR = PERIOD; /* 自動再ロード値をセットします */
TIM1->CCR1 = PULSE; /* パルス値をセットします */
TIM1->PSC = Prescaler; /* プリスケーラ値をセットします */
TIM1->RCR = PULSE_NUMBER - 1; /* 繰り返しカウンタ値をセットします */
```

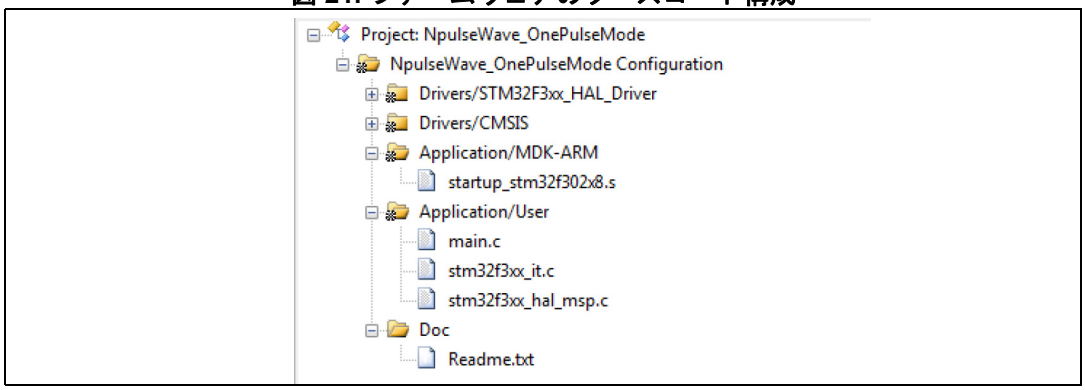
```
TIM1->EGR = TIM_EGR_UG; /* 更新イベントを生成してプリスケーラ値と繰り返しカウンタ値をただちに再ロードします */
TIM1->SMCR =  RESET;  /* 内部クロックソースを設定します */
TIM1->CR1 |= TIM_CR1_OPM; /* OPM モードを選択します */
TIM1->CCMR1 &= (uint16_t)~TIM_CCMR1_OC1M;
TIM1->CCMR1 &= (uint16_t)~TIM_CCMR1_CC1S;
TIM1->CCMR1 |=  TIM_OCMODE_PWM2;
/* チャネル 1 出力比較モードを選択します */
TIM1->CCER &= (uint16_t)~TIM_CCER_CC1P;
/* 出力比較の極性をハイにセットします */
TIM1->CCER |= TIM_OCPOLARITY_HIGH;
TIM1->CCER = TIM_CCER_CC1E; /* 比較出力チャネル 1 を有効にします */
TIM1->BDTR |= TIM_BDTR_MOE; /* TIM メイン出力を有効にします */
TIM1->CR1 |= TIM_CR1_CEN; /* TIM ペリフェラルを有効にします */
```

3.3 ファームウェアの概要

ファームウェアは、Keil µvision、IAR Embedded workbench、および SYSTEM WORKBENCH で開発しました。

開発したファームウェアは ZIP ファイルで提供され、アプリケーションの中核を構成するサブディレクトリおよび .h と .c の各ソースコードファイルが含まれています。

図 21. ファームウェアのソースコード構成



ファームウェアにはアプリケーションタスクのすべてのソースファイルと関連ファイルが含まれ、次のプロジェクトフォルダから構成されています。

- STM32 MCU HAL ライブラリ
- スタートアップファイル
- アプリケーションレイヤ

参考資料

4 ブレーク入力によるサイクルごとのレギュレーション

4.1 概要

ブレーク機能は、TIM1 タイマや TIM8 タイマなどの高機能設定のタイマペリフェラルでも、TIM15 タイマ、TIM16 タイマ、TIM17 タイマなどの簡易設定のタイマペリフェラルでも使用できます。ブレーク機能は、主にタイマ出力によって駆動されるパワーステージを保護するために使用されます。パワーステージ内またはマイクロコントローラ自体に何か異常が発生するとすぐに、ブレーク機能によってタイマ出力は無効になるか、強制的に事前定義されている安全な状態になります。

パワーステージで生成される外部ブレークリクエストを検出するため、ブレーク機能は専用の入力（BKIN や BKIN2 など）と関連付けられており、マイクロコントローラのいくつかの IO にオルタネート機能として割り当てられています。有効になると、ブレーク機能は、クロックが存在しなくても、ブレークイベントが検出されるとすぐに、PWM 出力を無効にするか、または強制的に事前定義されている安全な状態にします。たとえば、出力を非同期に無効にすることが可能です。

タイマ出力の事前定義されている安全な状態を設定する方法の詳細については、リファレンスマニュアルを参照してください。

有効なブレークイベントが検出されるとすぐに、TIMx_BDTR レジスタの MOE 制御ビットが非同期にクリアされます。これは、出力モードに設定されているタイマチャネルでのみ機能します。タイマチャネル出力の通常の動作状態を再開するには、ソフトウェアで MOE 制御ビットをセットするか、新しい PWM サイクルを開始することによってタイマ出力が通常の動作状態を再開するように AOE 制御ビットをセットする必要があります。

STM32 マイクロコントローラファミリの一部（STM32F303 ファミリなど）は、BKIN と BKIN2 の 2 つの外部ブレーク入力を備えたタイマペリフェラルを内蔵しています。これらのタイマペリフェラルでは、BRK ブレーク入力のほうが BRK2 ブレーク入力より優先されます。BKIN と BKIN2 の各ブレーク入力がタイマチャネル出力を無効にすることができるのは、それらを強制的にハイインピーダンス状態にする場合のみであり、強制的に事前定義されている安全な状態にすることはできません。

4.2 ブレーク入力と OCxRef クリアの利用

前のセクションで説明したように、ブレーク入力の主な用途は、障害状態にある場合にタイマチャネル出力を管理することです。ブレーク入力は、その柔軟な設計により、サイクルごとの電流レギュレーションなど、他の用途にも使用できます。概念上は、サイクルごとの電流レギュレーション機能は、STM32 のタイマペリフェラルの OCxRef クリア機能によって実現されます。

場合によっては、OCxRef クリア機能を使用してサイクルごとの電流レギュレーション機能を管理することができません。これは、この機能で使用される ETR タイマ入力が、別のタイマ機能（タイマの外部同期機能で外部クロックに ETR タイマ入力を使用する場合など）で同時に使用される場合があるためです。そのような状況では、サイクルごとの電流レギュレーション機能の処理にブレーク機能を使用することが有効な場合があります。

OCxRef クリア機能はタイマチャネル出力に作用します（ブレーク機能も同様）が、その一方で OCxRef 機能はチャネルごとに有効にすることができます（たとえば、OCxRef クリア機能は、調整するチャネルに対応する OCxCE 制御ビットをセットすることによってチャネル単位で有効にします）。しかし、ブレーク機能はすべてのタイマチャネル出力に作用します（たとえば、どのチャネルがブレークイベントの影響を受けるかを制御する手段はありません）。

サイクルごとの電流レギュレーションという概念は、調整後の電流の大きさが事前定義されている閾値を超えるとすぐに PWM 信号がローになり、その状態が次の PWM サイクルの開始まで継続するという事実に基づいています。この動作は、OCxREf クリア機能により自然にサポートされます（図 22 を参照）。

ブレーク機能では、この動作は AOE 制御ビットによって制御されます。AOE 制御ビットがそのデフォルト状態（リセット状態）に維持されている場合、調整後の電流が事前定義されている閾値を超えるとすぐに、タイマ出力がローになります。タイマは、ソフトウェアによって MOE ビットがセットされるまでその状態を維持しますが、これはサイクルごとのレギュレーションロジックの標準的な動作ではありません。

ブレーク機能によってレギュレーション機能を有効にする前に AOE 制御ビットをセットすると、新たな PWM サイクルが開始されるたびに自動的に MOE 制御ビットがセットされます。図 23 に、前に説明したブレーク機能と AOE 制御ビットの設定に応じた動作を示します。ここでは、1 つのタイマチャネル出力の波形がプロットされています。

図 22. TIMx OCxREF クリアのタイミング

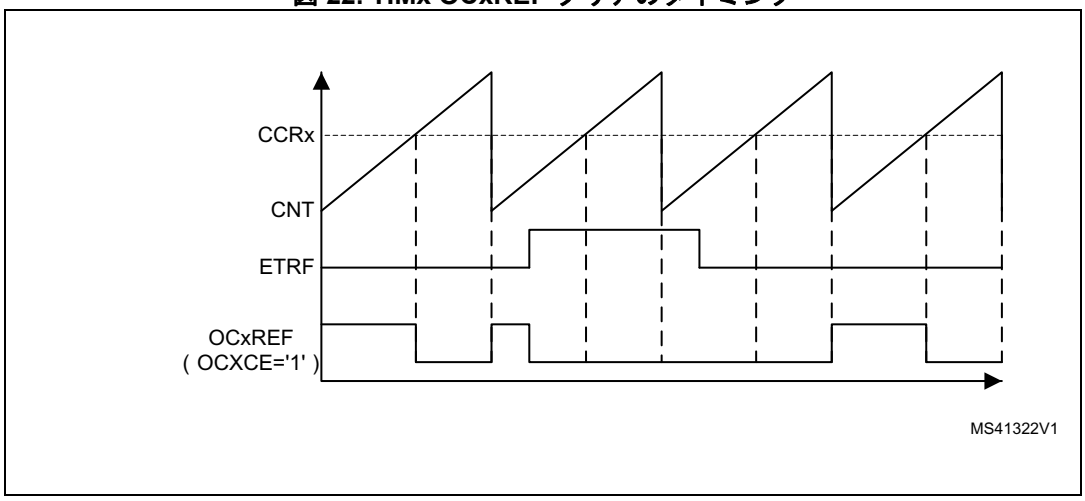


図 23. ブレーク機能のタイミング

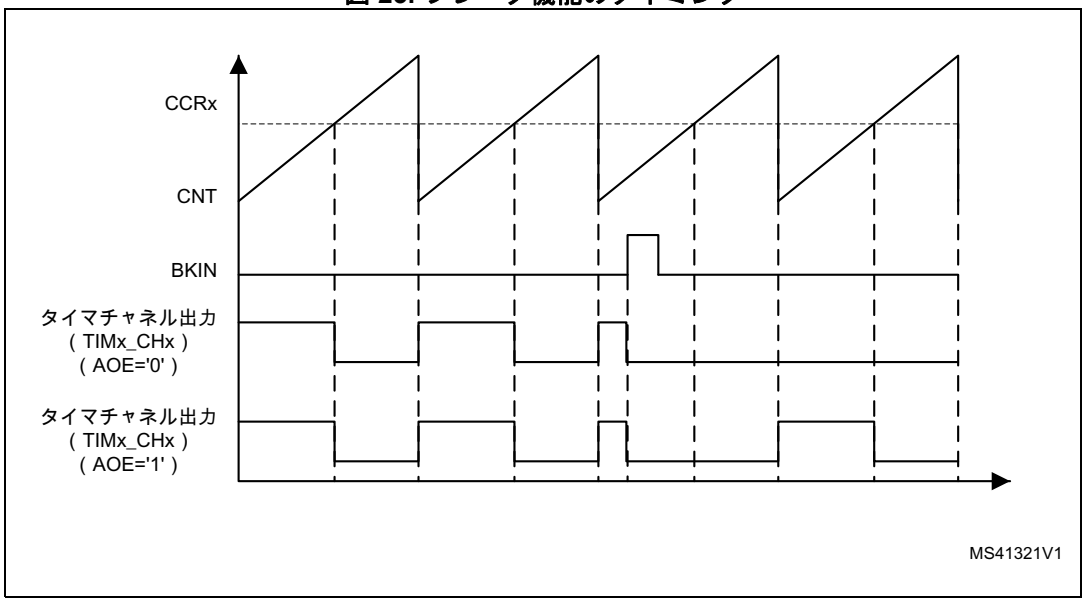


図 23 は、AOE 制御ビットの 2 つの設定でのタイマ出力の異なる動作を示しています。

- AOE = ‘0’ の場合、ブレーク入力がアクティブでなくても、PWM 出力は無効になります。
- AOE = ‘1’ の場合、ブレーク入力がアクティブでなければ、次の更新イベントで PWM 出力は有効になります。

4.3 アプリケーション：ブレーク機能によるサイクルごとのレギュレーション

サイクルごとのレギュレーションは、電力変換ステージレベルで過電流保護を実装するために採用されている手法の 1 つです。この機能は、パワーステージへの電流を監視します。電流が事前定義されている閾値（I_{Ref}）を超えるとすぐに、PWM 信号がローになり、レギュレーションの効果として流れ込む電流を閾値を下回るまで減らします。

PWM 信号は、新しい PWM サイクルの開始まで、設定されているデューティサイクルを再開できません。流れ込む電流がまだ閾値を超えている場合、次の PWM サイクルになるまで PWM 信号はロー状態のままです。

パワーステージへの電流と事前定義されている閾値電流の比較は、コンパレータベースの電子回路によって実現できます。コンパレータ自体をマイクロコントローラに内蔵できます（たとえば、STM32F302 と STM32F303 のマイクロコントローラでは内蔵されています）。コンパレータベースの回路の出力は、ブレーク入力に、および OCxRef クリア機能を使用している場合は ETR 入力に、それぞれ供給する必要があります。

図 24. サイクルごとのレギュレーションのタイミング

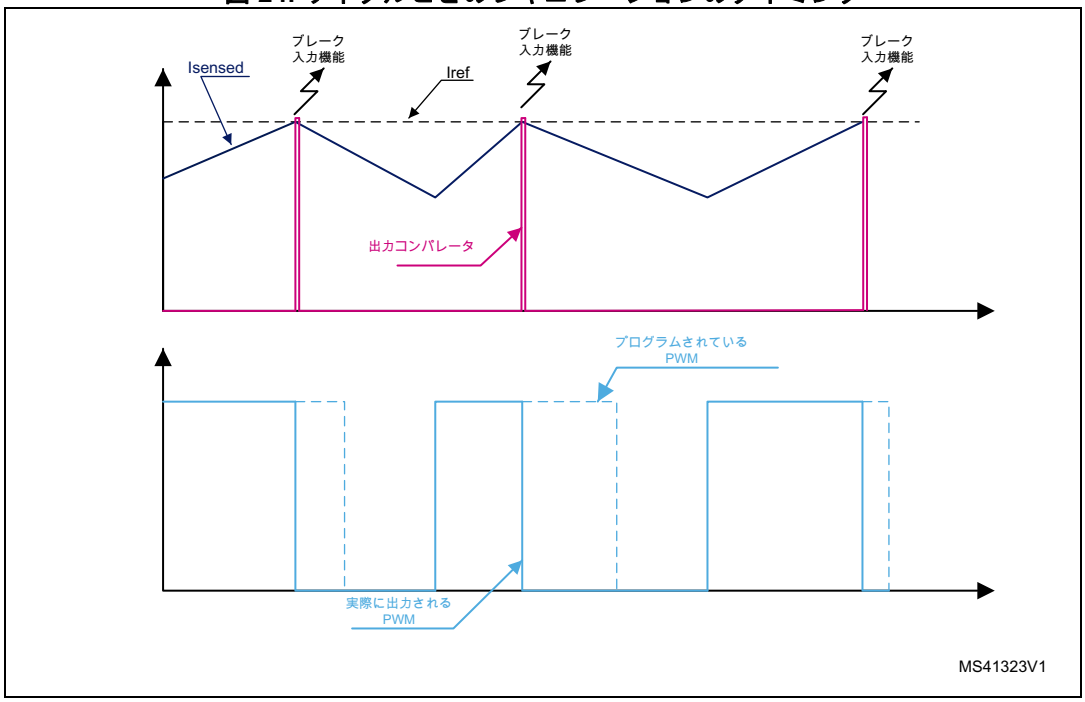


図 24 は、ブレークが有効な場合のブレーク機能の動作を示しています。ここには、実際に出力される PWM （実線）とプログラムされている PWM （点線）の値が示されています。

ブレーク機能は、BRK_ACTH 入力にハイレベルを適用することによって出力チャネルをローに駆動します。AOE ビット制御がセットされているので、次の「更新イベント」までブレーク機能はローのままです。

図 25 は、ブレーク機能によるサイクルごとのレギュレーションの概念のエミュレーションを示しています。実際の調整電力転換システムから出力されるフィードバック信号をエミュレートするために、受動電子部品（抵抗とコンデンサ）で構成される単純な電子回路が使用されています。

図 25. サイクルごとのレギュレーションのアーキテクチャ

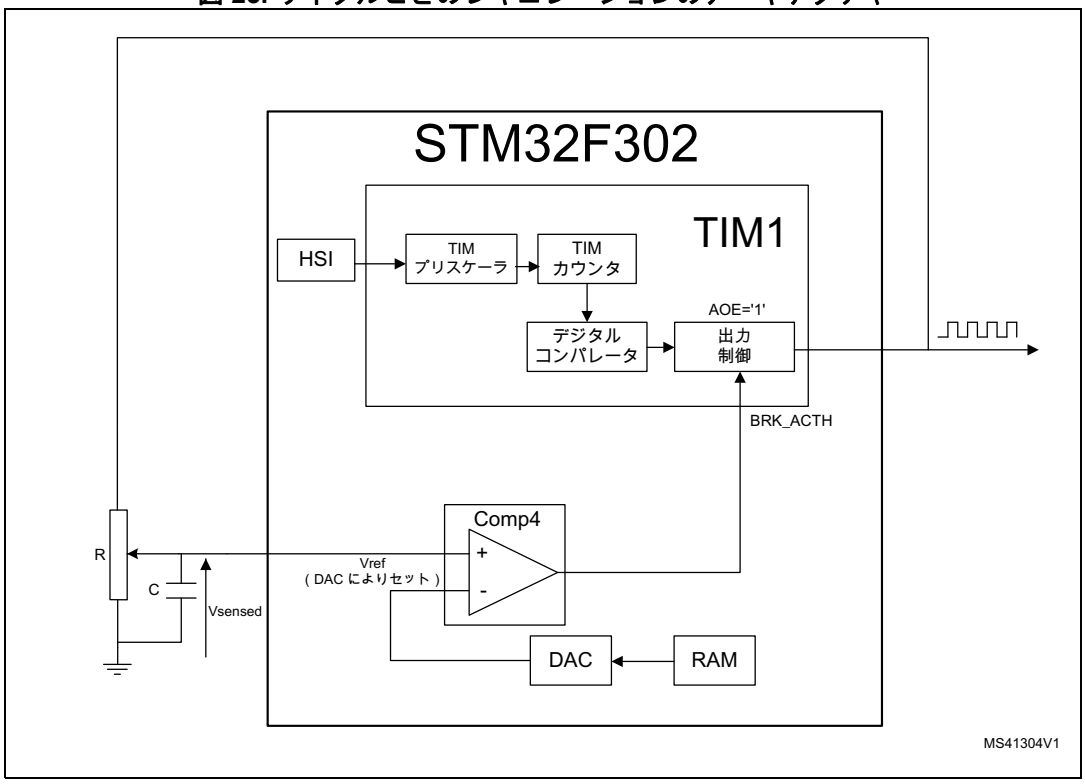
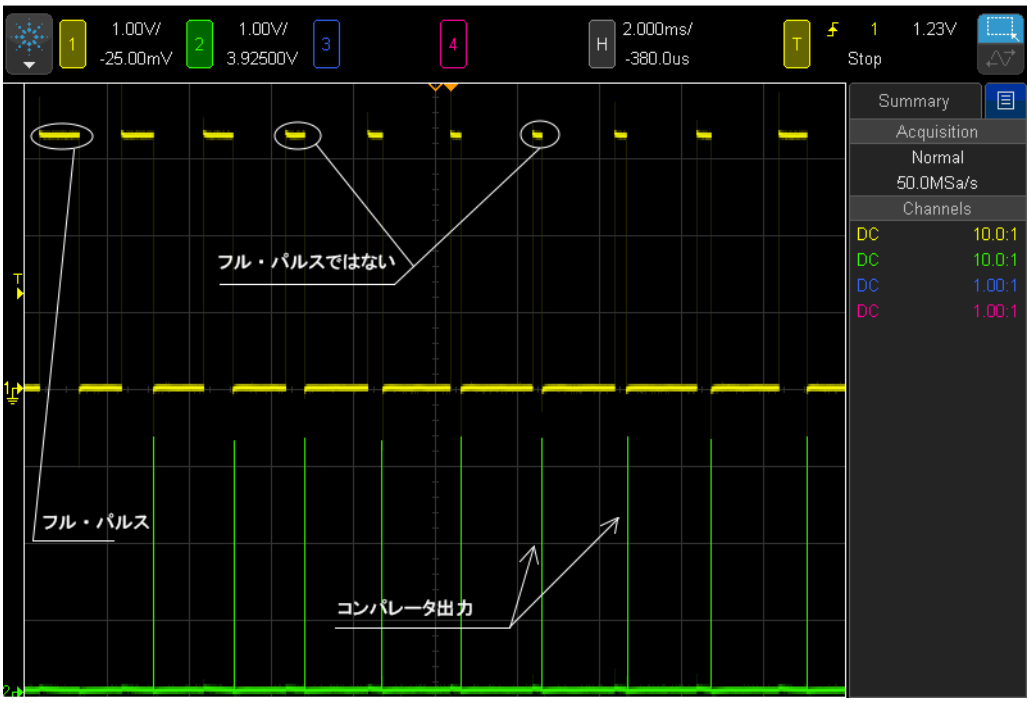


図 25 に示すように、タイマは PWM モードで使用されています。Vref は RAM に格納され、DAC とアナログコンパレータで変換された後、入力電圧と比較されます。コンパレータ出力は内部で、PWM 状態を制御する BRK_ACTH 信号にリダイレクトされます。

図 26. 得られた波形を示すオシロスコープのスクリーンショット



サイクルごとのレギュレーションは、電力変換ステージレベルで過電流保護を実装するために採用されている手法の 1 つです。図 26 は、Vref に対する入力電圧の変動によるデューティサイクル値の変動を時間の関数として示しています。

この図では、最初の PWM パルスの期間中、PWM 入力パルスが終了する前に、電圧が Vref 制限値に達しています。PWM 出力パルスは、そのサイクルの途中で打ち切られます。この後、次のパルスが印加されるまで電圧は低下し、印加されると再び上昇します。2 つ目の PWM パルスの期間中は、上昇する電流が電流制限閾値に達する前にパルスが終了するので、電流制限は発生しません。

このアプリケーションを実行するには、次のハードウェアセットアップが必要です。

- STM32F302 Nucleo ボード（NUCLEO-F302R8）
- Nucleo ボードを開発 PC に接続する USB ケーブル（この USB ケーブルでボードの電源も供給します）
- 出力信号波形を視覚化するオシロスコープ
- ポテンショメータとコンデンサ

このアプリケーションのソースコードを作り直して別の設定をデバッグするには、次の一連のソフトウェアツールが必要です。

- ST-LINK デバッガをサポートする開発ツールチェーン
- STM32CubeMX ソフトウェアツール V4.10.1 以降のバージョン（このアプリケーションの開発時の最新リリースは v4.10.1）

システムクロックの初期化

クロックの初期化は、メインルーチンで、HAL ライブラリ初期化（HAL_Init）の直後に SystemClock_Config() 関数を使用して実行します。

参考資料

ブレーク機能設定

```
RCC->APB2ENR |= RCC_APB2ENR_TIM1EN; /* TIM1 クロックを有効にします */
Prescaler = (uint16_t) (SystemCoreClock / 500000) - 1; /* 500 kHz のカウンタクロックを得られるようにタイマプリスケーラをセットします */
TIM1->CR1 &= ~(TIM_CR1_DIR | TIM_CR1_CMS);  /* アップカウンタモードを選択します */
TIM1->CR1 |= TIM_COUNTERMODE_UP;
TIM1->CR1 &= ~TIM_CR1_CKD;
TIM1->CR1 |= TIM_CLOCKDIVISION_DIV1; /* クロック分周を 1 にセットします */
TIM1->ARR = PERIOD;  /* 自動再ロード値をセットします */
TIM1->CCR1 = PULSE;  /* キャプチャ / 比較レジスタ値をセットします */
TIM1->PSC = Prescaler; /* プリスケーラ値をセットします */
TIM1->EGR = TIM_EGR_UG; /* 更新イベントを生成してプリスケーラ値と繰り返しカウンタ値をただちに再ロードします */
TIM1->SMCR = RESET;  /* 内部クロックソースを設定します */
TIM1->BDTR = RESET; /* BDTR ビットをクリアします */
TIM1->BDTR |= DEAD_TIME; /* デッドタイム値を 0 に設定します */
TIM1->BDTR |= TIM_LOCKLEVEL_OFF;  /* LOCK レベルを無効にします */
TIM1->BDTR |= TIM_OSSI_ENABLE;  /* 出力アイドルモードを有効にします */
TIM1->BDTR |= TIM_OSSR_DISABLE;  /* 出力ランモードを無効にします */
TIM1->BDTR |= TIM_BREAK_ENABLE; /* ブレーク入力を有効にします */
TIM1->BDTR |= TIM_BREAKPOLARITY_HIGH; /* 極性をハイにセットします */
TIM1->BDTR |= TIM_AUTOMATICOUTPUT_ENABLE; /* 自動出力を有効にします */
TIM1->CCMR1 &= ~TIM_CCMR1_OC1M; /* チャネル 1 出力比較モードを選択します */
TIM1->CCMR1 &= ~TIM_CCMR1_CC1S;
TIM1->CCMR1 |= TIM_OCMODE_PWM1;
TIM1->CCER &= ~TIM_CCER_CC1P; /* 出力比較の極性をハイにセットします */
TIM1->CCER |= TIM_OCPOLARITY_HIGH;
TIM1->CCER |= TIM_CCER_CC1E; /* 比較出力チャネル 1 を有効にします */
TIM1->CR1|=TIM_CR1_CEN; /* TIM ペリフェラルを有効にします */
```

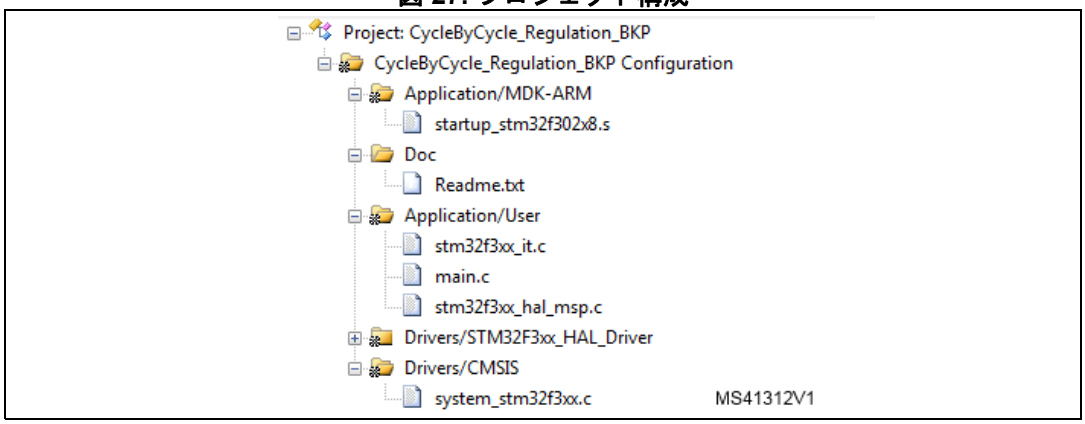
4.4 ファームウェアの概要

ファームウェアは、Keil µvision、IAR Embedded workbench、および

SYSTEM WORKBENCH で開発しました。

開発したファームウェアは ZIP ファイルで提供され、アプリケーションの中核を構成するサブディレクトリおよび .h と .c の各ソースコードファイルが含まれています。

図 27. プロジェクト構成



ファームウェアにはアプリケーションタスクのすべてのソースファイルと関連ファイルが含まれ、次のプロジェクトフォルダから構成されています。

- STM32 MCU HAL ライブラリ
- スタートアップファイル
- アプリケーションレイヤ

5 タイマ DMA バースト機能による任意の波形の生成

5.1 STM32 の DMA バースト機能の概要

DMA（Direct Memory Access）ペリフェラルは、ペリフェラルとメモリ間、およびメモリ間の両方でハイスピードデータ転送を行うために使用します。このアクションで CPU リソースを節約して、他のタスクに使用できます。

各 DMA 転送は、次の 2 つのステージで構成されます。

- 1 つ目のステージでは、転送されるデータが転送元の位置からロードされます。
- 2 つ目のステージでは、取得されたデータが転送先位置に格納されます。

この 2 ステージのデータ転送動作は、DMA ペリフェラル転送インデックスレジスタの更新に関連付けられています。これは、転送待ちデータの残量を追跡するために使用されるレジスタです。

STM32 マイクロコントローラファミリには、次の 2 種類の DMA ペリフェラルがあります。

- DMA バースト転送機能は、STM32F2 製品の DMA ペリフェラルでのみサポートされ、この機能で DMA ペリフェラルは、1 回のデータ転送トリガに続いて、設定可能な数のデータ要素を転送できます。
- STM32F1 製品などで使用されているもう 1 種類の DMA ペリフェラルは、シングル転送のみをサポートします。これは、1 回のデータ転送トリガに続いて転送されるデータ要素は 1 つのみであることを意味します。

上記の段落の目的は、多くの STM32 マイクロコントローラでサポートされている STM32 の DMA バースト機能について詳しく説明することではありません。上記の情報は、この機能に関する誤解や STM32 のタイマバースト機能に関する混乱を緩和することが目的です。STM32 のタイマバースト機能については、この章で重点的に説明します。

STM32 の DMA ペリフェラルの詳細については、STM32 マイクロコントローラのリファレンスマニュアルおよびアプリケーションノート STM32 cross-series timer overview （AN4013）を参照してください。

5.2 タイマ DMA バースト機能

タイマペリフェラルには、1 回のタイマイベントに続いて複数の連続する DMA リクエストを生成する機能があります。この機能の主な用途は、特定のタイマイベントがトリガされるたびにタイマペリフェラルの複数のレジスタの内容を更新することです。これは、タイマの動作モードを動的に再設定する（ある出力モードを別の出力モードに（たとえば PWM2 モードを強制アクティブレベルモードに）切り替える）か、または複数のチャネルで同時にランタイムパラメータを変更する（複数のタイマチャネルのデューティサイクルを同時に変更する）ために行うことができます。

同じ機能を使用して、タイマペリフェラルの多数のレジスタの内容をメモリバッファに転送することもできます。

この機能を使用すると、タイマペリフェラルレジスタの内容を調整することによって、タイマペリフェラル出力に出力される波形を動作中に変更できます。たとえば、TIMx_ARR レジスタの内容を更新して出力される波形の周波数を調整したり、TIMx_CCRx 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できます。

プログラマは、タイマの DMA バースト機能を使用するには、次に示すタイマペリフェラルレジスタを扱う必要があります。

- DMA アドレスレジスタ（TIMx_DMAR）: 読み書きアクセスリダイレクトレジスタ。
- DMA 制御レジスタ（TIMx_DCR）: バースト転送ステートマシン制御レジスタ。
- DMA コントローラのペリフェラルアドレスレジスタ設定

タイマペリフェラル DMA バースト機能のアドレスレジスタ

DMA ペリフェラル内部のペリフェラルアドレスレジスタは、DMA が“メモリからペリフェラル”モードで設定されている場合は、転送先レジスタアドレスを設定するために使用されます。また、DMA が“ペリフェラルからメモリ”モードに設定されている場合は、転送元レジスタアドレスを設定するために使用されます。

DMA ペリフェラルは、事前に定義/計算されている値が格納されているメモリバッファの内容でタイマペリフェラルの一連のレジスタの内容を更新する必要があります。ただし、これは、DMA ペリフェラルアドレスレジスタによってポイントされている転送アドレスを、各データ転送後に DMA コントローラがポストインクリメントするように設定する必要があることを意味するわけではありません。DMA コントローラのペリフェラルアドレスレジスタは、タイマペリフェラルの TIMx_DMAR レジスタをポイントする必要があります。

TIMx_DMAR タイマレジスタは、仮想レジスタです。このレジスタへのアクセスはすべて、タイマペリフェラル DMA バースト制御ロジックにより、タイマペリフェラルの物理レジスタの 1 つにリダイレクトされます。

TIMx_DMAR レジスタへのアクセスには、読出しタイプまたは書込みタイプのどちらもあります。TIMx_DMAR レジスタへの実際のアクセスをタイマペリフェラルの別の物理レジスタにリダイレクトする場合、このリダイレクトは DMA バーストインタフェース設定での TIMx_DCR レジスタの内容に依存します。また、タイマ DMA バーストインタフェースを制御する有限状態機械（FSM）の実際の状態にも依存します。

DMA コントローラのメモリアドレスレジスタ設定

DMA ペリフェラル内部のメモリアドレスレジスタは、DMA が“ペリフェラルからメモリ”モードで設定されている場合は、転送先メモリ位置アドレスを設定するために使用されます。また、DMA が“メモリからペリフェラル”モードに設定されている場合は、転送元メモリ位置アドレスを設定するために使用されます。

DMA ペリフェラルは、タイマペリフェラルの一連のレジスタの内容をメモリバッファの内容で更新する必要があります。DMA メモリアドレスレジスタによってポイントされる転送アドレスは、各データ転送後に DMA コントローラがポストインクリメントするように設定する必要があります。

タイマペリフェラル DMA バースト機能制御レジスタ

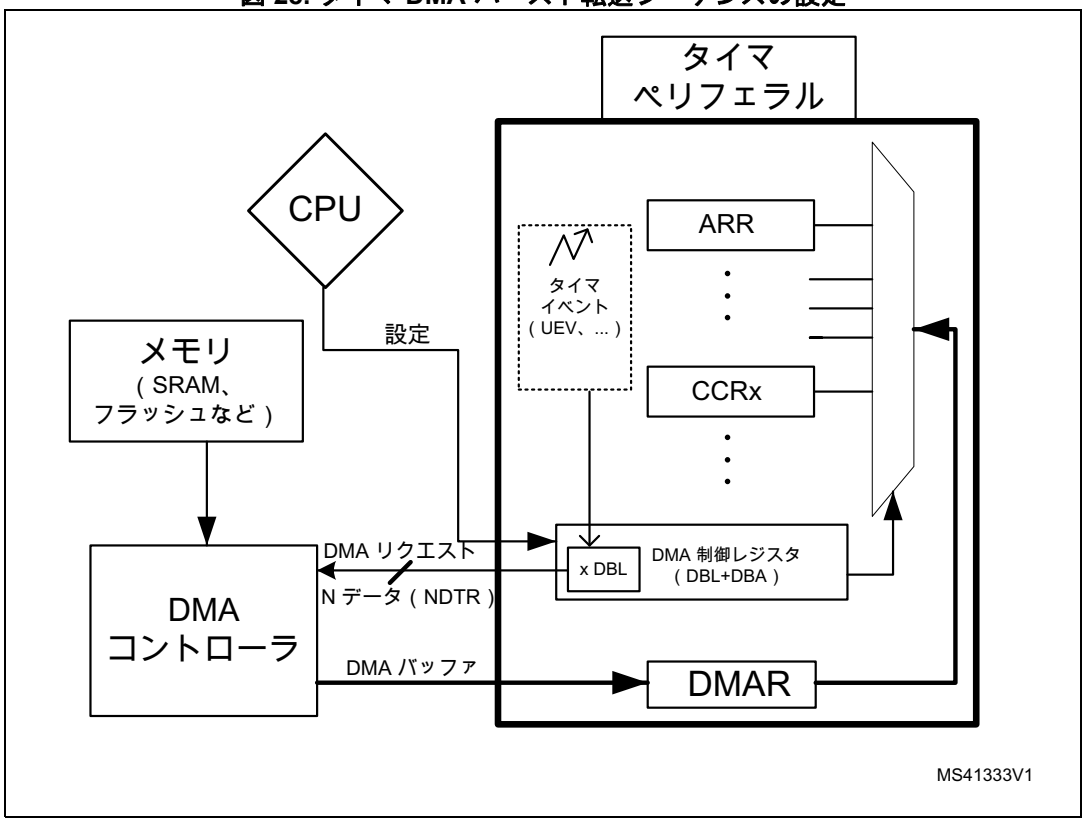
タイマ DMA バースト機能のステートマシン制御レジスタ TIMx_DCR は、1 回のバースト転送中のビート数を設定するために使用されます。また、バースト転送の最初のデータ転送のターゲットタイマレジスタまたは転送元の識別にも使用されます。

DBL[4:0] 制御ビットフィールドは、1 回のバースト転送中のビート数をセットします。この数は、バースト転送に関与（書込みまたは読出し）するタイマレジスタの数と同じである必要があります。DBA[4:0] 制御ビットフィールドの内容は、タイマレジスタの中でバースト転送に関与するスタートレジスタを識別します。

DBA[4:0] 制御ビットフィールドは 5 ビット幅なので、最大 32 個のタイマレジスタを識別できます。タイマレジスタ識別番号は、タイマのレジスタマップ内のレジスタ相対アドレスを 4 で除算して求めます。たとえば、TIMx_CR1 レジスタの相対アドレスが 0x00 である場合、その識別番号は 0 です。

TIMx_CR1 レジスタからバースト転送を開始するには、DBA[4:0] 制御ビットフィールドを 0 にセットする必要があります。TIMx_ARR レジスタからバースト転送を開始するには、DBA[4:0] ビットフィールドを 11 にセットする必要があります。これは 44/4 = 11 を意味します。ここで、44 は TIMx_ARR レジスタの相対アドレス 0x2c を 10 進数にエンコードした値です。

図 28. タイマ DMA バースト転送シーケンスの設定



1 回のタイマ DMA バースト転送シーケンスを完了するには、図 28 に示すように、DMA とタイマペリフェラルが協調する必要があります。タイマレジスタの更新に使用するデータ値は、マイクロコントローラのメモリ内のどこかに格納しておく必要があります。波形生成中にパターンを更新する必要がある場合は、これらのデータ値を SRAM メモリに格納できます。

アプリケーションソフトウェアは、データ転送元としてデータバッファをポイントするように DMA を設定する必要があります。また、データ転送の転送先として、タイマの TIMx_DMAR レジスタをポイントする必要があります。最後に、アプリケーションソフトウェアは、タイマの TIMx_DCR レジスタに適切な設定を書き込むことによって、タイマ DMA バースト機能を設定する必要があります。

これまでの段落では、タイマ DMA バースト機能を設定する方法について詳しく説明しました。適切な設定を行ったらすぐに、データ転送に使用する DMA ストリームまたは DMA チャネルを有効にし、続いてタイマカウンタを有効にする必要があります。

有効になったタイマカウンタは、インクリメントまたはデクリメントすることで周期的に更新されます。しばらくしてから、有効なタイマ DMA リクエストに応じて、タイマペリフェラルが内部的に DMA リクエストを発生させる可能性があります。DMA リクエストは、内部的にタイマ DMA バースト制御ロジックに送られます。

DMA バースト長（TIMx_DCR タイマレジスタの DBL[4:0] ビットフィールド）に設定されている値に基づいて、DMA リクエストがそのまま、または数倍にして、DMA ペリフェラルに送られます。DBL[4:0] の内容が null の場合、DMA リクエストはそのまま送信されます。それ以外の場合、DMA リクエストにDBL の値 + 1の係数 が乗じられます。

DBL[4:0] ビットフィールドの内容が 2 の場合、内部でタイマ DMA リクエストが発生するとすぐに、タイマ DMA バースト制御ロジックが最初の DMA リクエストを DMA ペリフェラルに送信します。DMA ペリフェラルは、DMA レジスタ転送元によってポイントされているメモリ位置の内容を、TIMx_DMAR タイマレジスタに転送します。その後、転送元ポインタをインクリメントして、タイマ DMA リクエストに確認応答します。

タイマ DMA バースト制御ロジックは、最初の DMA 確認応答を受信するとすぐに、2 つ目の DMA リクエストを送信します。この DMA リクエストは再び DMA ペリフェラルによって処理され、再度確認応答がタイマに送信されます。

タイマ DMA バースト制御ロジックは、2 つ目の DMA 確認応答を受信すると、3 つ目の DMA リクエストを送信します。この 3 つ目のリクエストも、DMA ペリフェラルによって処理されます。

タイマが DMA から 3 つ目の確認応答を受信するとすぐに、内部で発生した DMA リクエストによってトリガされた転送シーケンスは正常終了したものとみなされます。これで、タイマ DMA バースト制御ロジックが新しい転送シーケンスを開始する準備が整います。

前述のデータ転送シーケンスでは、DMA ペリフェラルによってポイントされている転送先レジスタは、転送シーケンスを通じてずっと同じです。このレジスタは、TIMx_DMAR タイマレジスタに等しいままです。

タイマ DMA バースト制御ロジックは、TIMx_DMAR タイマレジスタへの書込みアクセスがあるたびに、それを適切な物理タイマレジスタにリダイレクトします。

前述の例で、TIMx_DCR レジスタの DMA ベースアドレスビットフィールド DBA[4:0] が 11 にセットされている場合、次のことに注意してください。

- TIMx_DMAR タイマレジスタへの 1 つ目の DMA 書込みアクセスは、TIMx_ARR タイマレジスタにリダイレクトされます。TIMx_ARR タイマレジスタは、バースト転送のベースアドレスになるように選択されています。
- TIMx_DMAR タイマレジスタへの 2 つ目の DMA アクセスは、TIMx_RCR タイマレジスタにリダイレクトされます（この例で使用しているタイマは TIMx_RCR レジスタを内蔵しているものとします）。
- TIMx_DMAR タイマレジスタへの 3 つ目の DMA アクセスは、TIMx_CCR1 タイマレジスタにリダイレクトされます。
- TIMx_DMAR レジスタへの 3 つ目の DMA アクセスが終了すると、タイマ DMA バースト制御ロジックは、書込みアクセスリダイレクトステートマシンをループバックします。再び、バースト転送で設定されているベースレジスタ（この例では TIMx_ARR タイマレジスタ）がポイントされます。

タイマペリフェラルの内部で新しい DMA リクエストが生成されるたびに、上記のシーケンスが繰り返されます。

タイマ DMA バーストを使用して、定期的にタイマペリフェラルのレジスタの内容を特定のメモリバッファに読み出す場合も、上記のシーケンスは有効です。変更する必要があるのは、DMA 転送方向、転送元アドレス、および転送先アドレスのみです。

5.3 アプリケーション例：タイマ DMA バースト機能による任意の波形の生成

この例では、STM32 のタイマペリフェラルを使用して CPU のオーバーヘッドなしに任意の波形信号を生成する方法を示します。また、この例で要となるタイマの DMA バースト機能についてわかりやすく説明するという意図もあります。

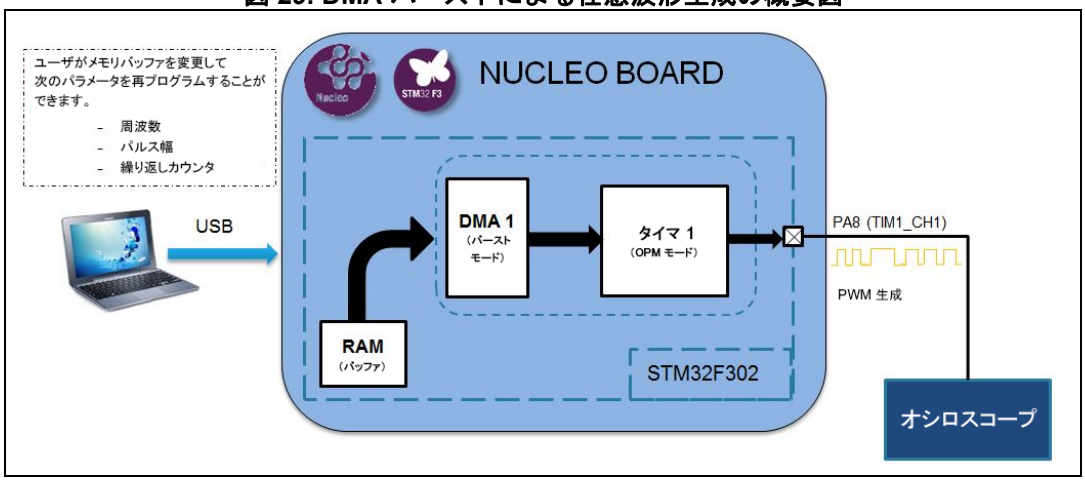
アプリケーションの概要

このアプリケーションは、ボードのメインコントローラとして STM32F302x8 マイクロコントローラを搭載する Nucleo ボード NUCLEO-F302R8 をベースに設計されています。ただし、このアプリケーションは、任意の STM32 ハードウェアプラットフォームに簡単に移植できます。

Nucleo ボードには、独自の ST-Link/V2 デバッガが組み込まれています。アプリケーションのバイナリイメージをマイクロコントローラにダウンロードしたり、アプリケーションをデバッグするため、Nucleo ボードを PC コンピュータに接続する際に必要なのは、USB ケーブルのみです。

図 29 に、このアプリケーション例で示されている任意波形ジェネレータアプリケーションの概要図を示します。生成された信号は、Nucleo ボードのコネクタ CN9 のピン 8 に割り当てられている STM32F302 マイクロコントローラの PA.08 IO に出力されます。

図 29. DMA バーストによる任意波形生成の概要図

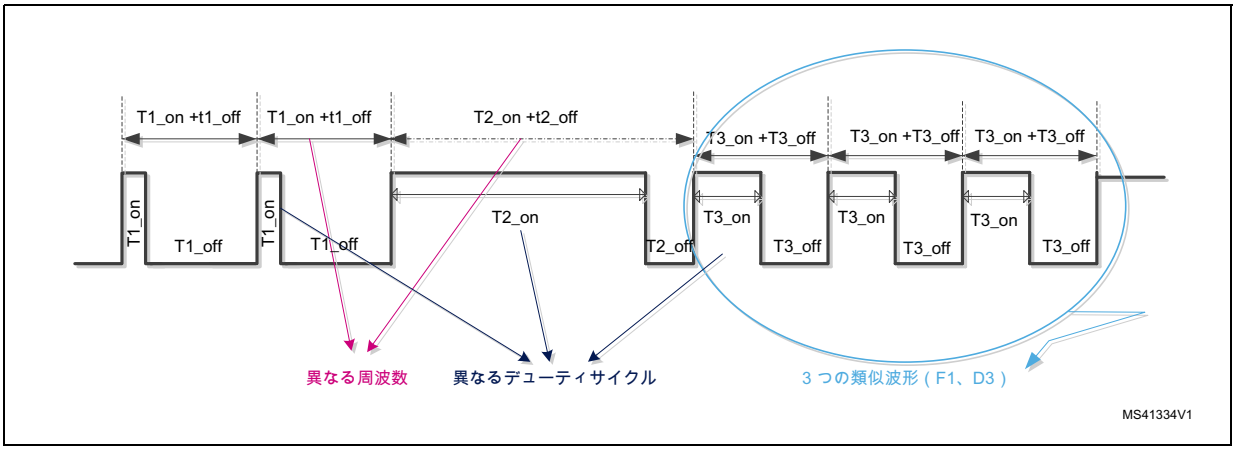


この例で説明している任意波形ジェネレータは、図 30 に示す波形と同様の波形を出力するように設計されています。このアプリケーションのソースコードは、さまざまな波形を出力するように簡単に作り直すことができますが、実証用に 14 ページの図 4 の波形をターゲットにしています。

図 30 でターゲットにしている波形は、次の 3 つの波形部分で構成されています。

- 1 つ目の部分は、特定のオン時間（t1_on）とオフ時間（t1_off）による連続するパルス 2 つで構成されています。
- 2 つ目の部分は、異なるオン時間（t2_on）とオフ時間（t2_off）によるパルス 1 つで構成されています。
- 3 つ目の部分は、さらに異なるオン時間（t3_on）とオフ時間（t3_off）によるパルス 3 つで構成されています。

図 30. 任意波形ジェネレータアプリケーション：ターゲットの波形



STM32 のタイマペリフェラルにこの波形を出力させるには、繰り返しカウンタを搭載しているタイマペリフェラルを使用できます。このタイマペリフェラルは、TIMx_RCR レジスタを内蔵し、少なくとも 1 つのタイマチャネルを備えています。この例では、4 つのタイマチャネルと繰り返しカウンタを備えているという理由で、TIM1 タイマペリフェラルが使用されています。

タイマチャネルは、事前定義されている一定の周波数とデューティサイクルのパラメータで PWM 信号を生成できます。信号周波数パラメータは、TIMx_ARR タイマ自動再ロードレジスタの内容によって制御されます。このレジスタでは、デューティサイクルパラメータは、タイマチャネル 1 レジスタ（TIM1_CCR1）の内容によって制御されます。リクエストされた波形（図 4 を参照）をタイマチャネルに出力させるには、各 PWM サイクルの終了時にこれら 2 つのタイマレジスタの内容を更新する必要があります。

言い換えると、各タイマ「更新イベント」と同期して、これら 2 つのレジスタの内容を更新する必要があります。これを実行する直観的な方法としては、タイマ「更新イベント」が発生するたびにこれら 2 つのレジスタの内容をアプリケーションソフトウェア（CPUユニット）で更新させます。直観的には、この方法では CPU ユニットでオーバーヘッドが発生して、出力される波形は決定論的ではありません。

この後の段落では、いくつかのタイマ機能を DMA ペリフェラルと組み合わせて使用することで、CPU のオーバーヘッドを緩和し、決定論的な波形信号を生成する方法について説明します。

1 回のタイマ「更新イベント」に続いて TIMx_ARR と TIMx_CCR1 のタイマレジスタを同時に更新するには、1 つの DMA チャネルまたは DMA ストリームとともにタイマ DMA バースト機能を使用する必要があります。タイマの DMA バースト機能を設定して使用する方法については、本書のこれまでのセクションで説明しています。

タイマチャネルレジスタの更新によってタイマチャネルで不要なグリッチが生成されないようにするには、STM32 のタイマペリフェラルのプリロード機能を使用します。

タイマチャネルで同じ t_on パラメータと t_off パラメータの繰り返しパルスを出力させ、PWM サイクルのたびにタイマで「更新イベント」が生成されないようにするには、タイマの繰り返しカウンタを使用する必要があります。

図 29 に示す波形を TIM1 タイマペリフェラルのチャネル 1 に出力するには、次に示すように 3 つのレジスタの内容を更新する必要があります。

- **TIM1_ARR**：処理中のパルスの t_on 期間と t_off 期間の合計を格納します。
- **TM1_RCR**：処理中の信号波形部分あたりのパルス数から 1 を引いた値を格納します（TIM1_RCR=Number_of_pulses_per_portion–1）。
- **TIM1_CCR1**：処理中のパルスの t_on 期間の長さを格納します。

図 29 に示す波形を再構成するために使用するタイマレジスタは、次のように設定します。

波形の 1 つ目の部分の適切な設定を次に示します。

- TIM1_ARR = t1_on + t1_off、TIM1_RCR= 1、TIM1_CCR1= t1_on

信号波形の 2 つ目の部分の適切な設定を次に示します。

- TIM1_ARR = t2_on + t2_off、TIM1_RCR= 0、TIM1_CCR1= t2_on

信号波形の 3 つ目の部分の適切な設定を次に示します。

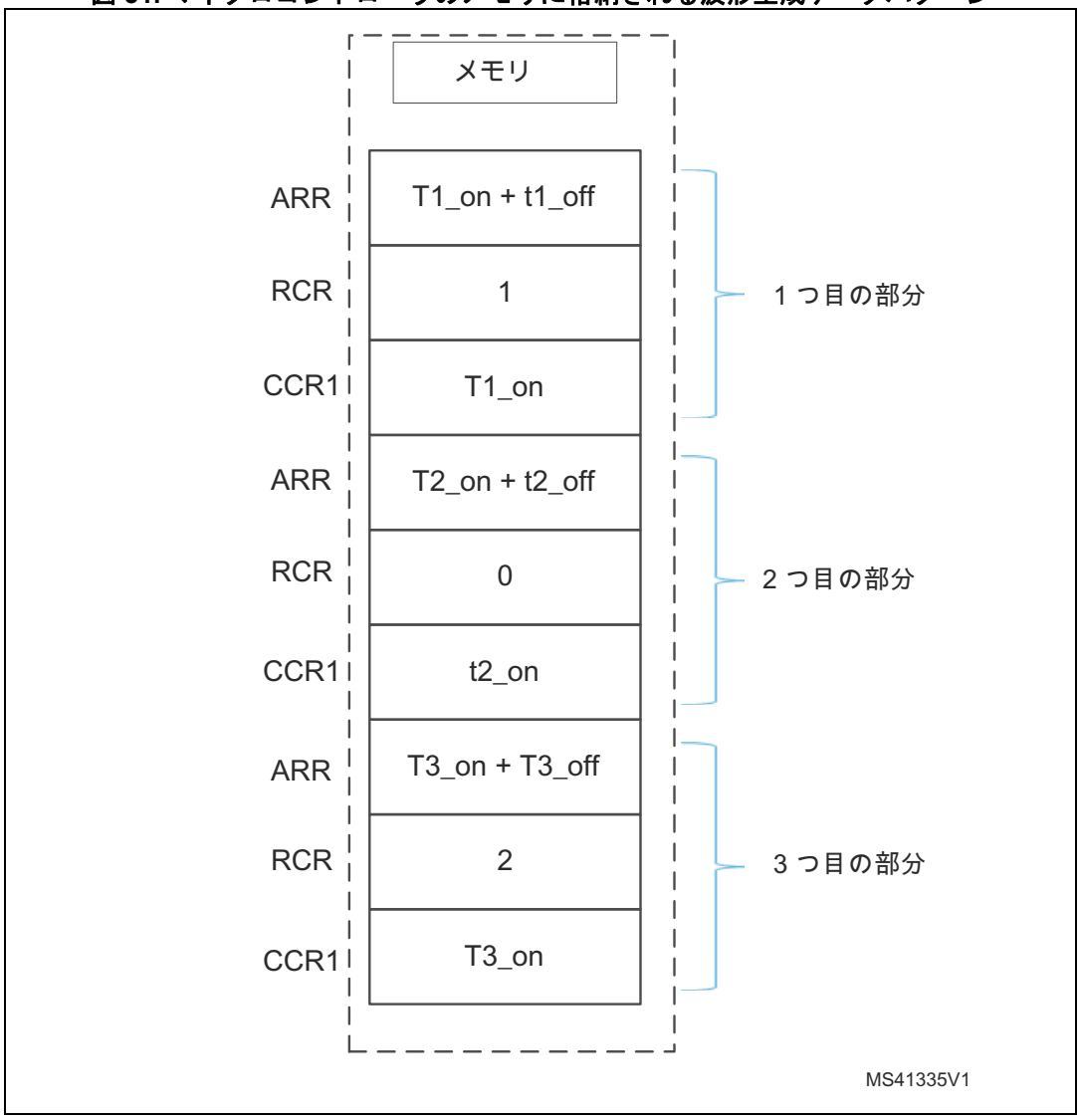
- TIM1_ARR = t3_on + t3_off、TIM1_RCR= 2、TIM1_CCR1= t3_on

メモリバッファの内容：

上に示した、ターゲット波形の各部分の TIM1_ARR、TIM1_RCR、および TIM1_CCR1 タイマレジスタの値を見ながら、図 31 に示すマイクロコントローラのメモリに定義する値テーブルを参照してください。

テーブル内のデータ値の順序は、タイマペリフェラルのレジスタマップ内の各レジスタの順序と同じです。したがって、正しい順序は TIM1_ARR レジスタ、TIM1_RCR レジスタ、および TIM1_CCR1 レジスタになります。

図 31. マイクロコントローラのメモリに格納される波形生成データパターン



上記の値テーブルの C 言語ソースコードを次に示します。Const キーワードは、波形生成中はテーブルの内容が変更されないことを示すために使用されます。

メモリアレイは、フラッシュメモリに割り当てられます。ランタイム時に波形パラメータを変更する必要がある場合は、キーワード Const を使用しないでください。メモリアレイは、フラッシュメモリ領域と SRAM メモリ領域の 2 つのメモリ領域に割り当てられます。

```
uint32_t const aSRC_Buffer[9] = { t1_on+t1_off,1,t1_on,
t2_on+t2_off,0,t2_on, t3_on+t3_off,2,t3_on};
```

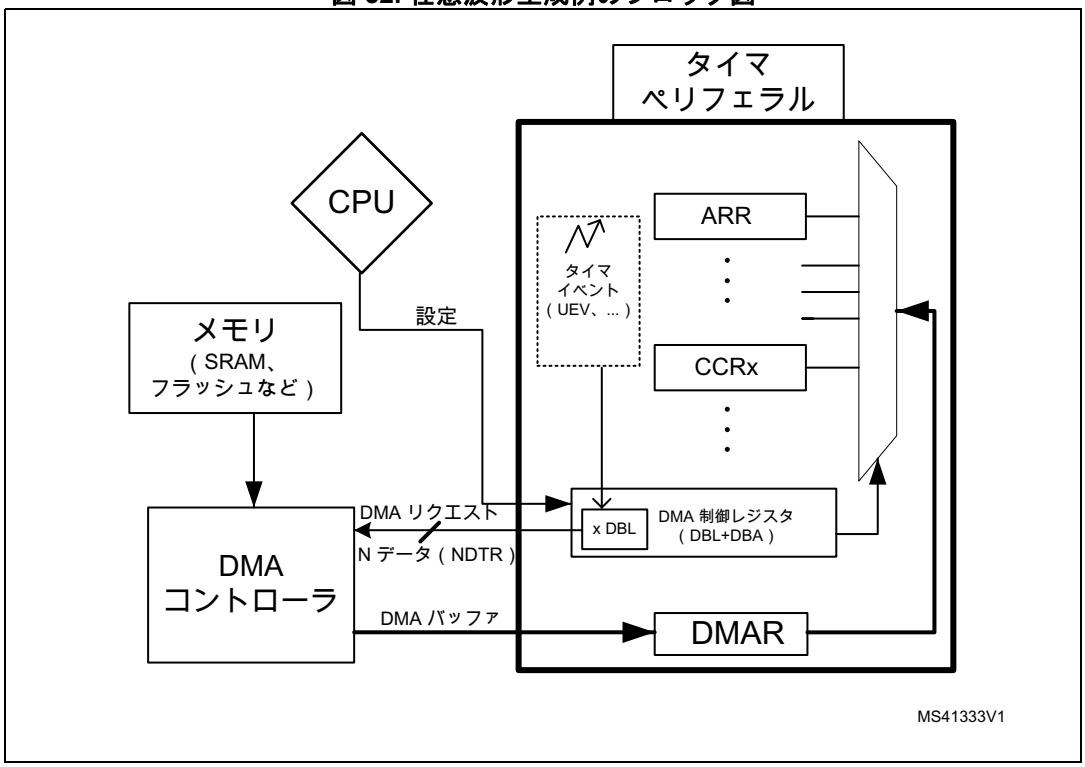
タイマ DMA バースト設定

目的の波形を出力するには、TIM1 のタイマ DMA バーストを次のように設定する必要があります。

- 更新するタイマレジスタが 3 つあるので、バースト転送長は **3** です。TIM1_DCR の DBL[4:0] 制御ビットフィールドは 2 にセットする必要があります。UEV ごとに 3 つのデータが転送され、3 回の更新イベントが発生します。
- 更新するタイマレジスタの中で、TIM1_ARR タイマレジスタは TIM1 タイマのレジスタマップの先頭にあるので、これを転送のベースとして定義します。DBA[4:0] 制御ビットフィールドは、TIM1_ARR レジスタをポイントするようにセット（DBA[4:0] = 11）する必要があります。

図 32 は、これまでに説明した設定のブロック図を示します。

図 32. 任意波形生成例のブロック図



クロック設定

この例では、32 MHz の TIM1 カウンタクロックを得るために、次のように設定します。

- TIM1 入力クロック

TIM1 入力クロック（TIM1CLK）は APB2 クロック（PCLK2）にセットされます。

TIM1CLK = PCLK2 かつ PCLK2 = HCLK なので、TIM1CLK = HCLK = SystemCoreClock となります。

- TIM1 プリスケーラ

32 MHz の TIM1 カウンタクロックを得るために、プリスケーラは次のように計算されます。

プリスケーラ = (TIM1CLK / TIM1 カウンタクロック) - 1

プリスケーラ = (SystemCoreClock / 32 MHz) - 1

周波数とデューティサイクルの計算

この例では、生成される波形のデータパターンは、次のように定義され、

```
uint32_t aSRC_Buffer[9] = { 4000,1,800,10000,0,8500,4000,2,200};
```

次の計算に基づきます。

– TIM1 周波数の計算（F1、F2）:

$$TIM1Frequency(F1) = \frac{TIM1_{counterclock}}{TIM_{ARR}+1} = \frac{32MHz}{4000+1} = 8.0KHz$$

$$TIM1Frequency(F2) = \frac{TIM1_{counterclock}}{TIM_{ARR}} = \frac{32MHz}{10000+1} = 3.2KHz$$

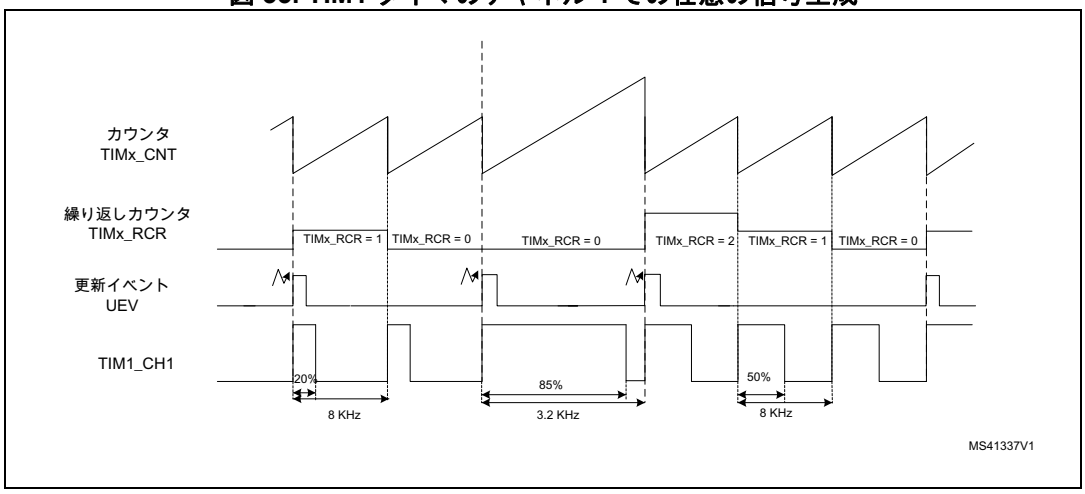
– TIM1 チャネル 1 デューティサイクルの計算（D1、D2、D3）:

$$TIM1dutycycle(D1) = \frac{TIM1_{CCR1}}{TIM_{ARR}} \times 100 = \frac{800}{4000} \times 100 = 20\%$$

$$TIM1dutycycle(D2) = \frac{TIM1_{CCR1}}{TIM_{ARR}} \times 100 = \frac{8500}{10000} \times 100 = 85\%$$

$$TIM1dutycycle(D3) = \frac{TIM1_{CCR1}}{TIM_{ARR}} \times 100 = \frac{2000}{4000} \times 100 = 50\%$$

図 33. TIM1 タイマのチャネル 1 での任意の信号生成



ファームウェアの説明

前のセクションで説明した任意の信号の生成を実現するには、次のステップを適用する必要があります。

- **システムクロック**
  - PLL （システムクロックソース）: 64 MHz
  - HSI （オシレータ。HSE を Nucleo ボードにはんだ付けする必要はなし）
  - AHB 分周 = 1 / APB1 分周 = 2 / APB2 分周 = 1

この例では、DMA1 と TIMER1 を使用します。

- **DMA1 設定**

```
/* DMA1 クロックを有効にします */
RCC->AHBENR |= RCC_AHBENR_DMA1EN;
/* DMA1 チャネル 5 CR レジスタを設定します */
/* DMA1 チャネル 5 制御レジスタをリセットします */
  DMA1_Channel5->CCR = 0;
/* CHSEL ビットを DMA チャネル 5 にセットします */
/* DIR ビットをメモリからペリフェラルの方向にセットします */
/* PINC ビットを DMA ペリフェラルインクリメント無効にセットします */
/* MINC ビットを DMA メモリインクリメント有効にセットします */
/* PSIZE ビットをペリフェラルデータサイズ = ワードにセットします */
/* MSIZE ビットをメモリデータサイズ = ワードにセットします */
/* CIRC ビットをサーキュラモードにセットします */
/* PL ビットを最優先にセットします */
/* MBURST ビットをシングルメモリバーストにセットします */
/* PBURST ビットをシングルペリフェラルバーストにセットします */
  DMA1_Channel5->CCR |= DMA_MEMORY_TO_PERIPH |
                        DMA_PINC_DISABLE| DMA_MINC_ENABLE|
                        DMA_PDATAALIGN_WORD| DMA_MDATAALIGN_WORD |
                        DMA_CIRCULAR | DMA_PRIORITY_HIGH;
/* DMA1 チャネル 5 のデータのレジスタ数を書き込みます */
  DMA1_Channel5->CNDTR = 9;
/* DMA1 チャネル 5 ペリフェラルアドレスレジスタに書き込みます */
  DMA1_Channel5->CPAR = (uint32_t)TIM1_DMAR_ADDRESS;
/* DMA1 チャネル 5 メモリアドレスレジスタに書き込みます */
/* アドレスをメモリバッファ aSRC_Buffer にセットします */
  DMA1_Channel5->CMAR = (uint32_t)aSRC_Buffer;
/* DMA1 チャネル 5 を有効にします */
  DMA1_Channel5->CCR |= (uint32_t)DMA_CCR_EN;
```

- **TIM1 設定**

```
/* タイマプリスケーラをセットします */
  Tim1Prescaler= (uint16_t) (SystemCoreClock / 32000000) - 1;
/* 周期を設定します */
  TIM1->ARR = 0xFFFF;
/* タイマプリスケーラを設定します */
  TIM1->PSC = Tim1Prescaler;
/* パルス幅を設定します */
  TIM1->CCR1 = 0xFFF;
/* クロック分周として 1 を選択します */
/* クロック分周ビットフィールドをリセットします */
  TIM1->CR1 &= ~ TIM_CR1_CKD;
```

```
/* クロック分周として DIV1 を選択します */
  TIM1->CR1 |= TIM_CLOCKDIVISION_DIV1;
/* TIM1 カウンタのアップカウントを選択します */
/* モード選択ビットフィールドをリセットします */
  TIM1->CR1 &= ~( TIM_CR1_DIR | TIM_CR1_CMS);
/* アップカウントモードを選択します */
  TIM1->CR1 |=  TIM_COUNTERMODE_UP;
/* PWM1 モードをセットします */
/* 出力比較モードビットをリセットします */
  TIM1->CCMR1 &= ~TIM_CCMR1_OC1M;
  TIM1->CCMR1 &= ~TIM_CCMR1_CC1S;
/* 出力比較モード 1 を選択します */
  TIM1->CCMR1 |= TIM_OCMODE_PWM1;
/* 出力比較 1 プリロードを有効にします */
  TIM1->CCMR1 |= TIM_CCMR1_OC1PE;
/* 自動再ロードプリロードを有効にします */
  TIM1->CR1 |= TIM_CR1_ARPE;
/* TIM1 DMA 更新を有効にします */
  TIM1->DIER |= TIM_DMA_UPDATE;
/* DMA ベースレジスタと DMA バースト長を設定します */
/* DBA と DBL の各ビットフィールドをリセットします */
  TIM1->DCR &= ~TIM_DCR_DBA;
  TIM1->DCR &= ~TIM_DCR_DBL;
/* DMA ベースレジスタと DMA バースト長を選択します */
  TIM1->DCR = TIM_DMABase_ARR | TIM_DMABurstLength_3Transfers;
/* UG ビットをセットしてバッファデータをプリロードレジスタにロードすることにより UEV を有効にします */
  TIM1->EGR |= TIM_EGR_UG;
/* UG ビットがリセットされるまで待ちます */
  while((TIM1->EGR & TIM_EGR_UG) == SET){}
/* UG ビットをセットしてデータをプリロードレジスタからアクティブレジスタにロードすることにより UEV を有効にします */
  TIM1->EGR |= TIM_EGR_UG;
/* TIM1 メイン出力を有効にします */
  TIM1->BDTR |= TIM_BDTR_MOE;
/* CC1 出力を有効にします */
  TIM1->CCER |= TIM_CCER_CC1E;
/* TIM カウンタを有効にします */
```

```
TIM1->CR1 |= TIM_CR1_CEN;
```

- GPIO：
  - ピン PA8：TIM1_ch1_output
  - モード：プッシュプル
  - プル：プルアップ
  - スピード：ハイ
  - オルタネート機能：GPIO_AF6_TIM1

6 タイマ同期による N パルス波形生成

このアプリケーション例は、類似した 2 つのアプリケーション例について説明する 2 つのパートに分かれています。どちらの例も、タイマ間同期を使用して N パルス波形を生成しますが、次に示す点が多少異なります。

- このセクションのパート 1 で説明する 1 つ目のアプリケーション例では、N パルス波形が、TIM1 タイマチャネル 1 の出力と相補出力で生成されます。N パルス波形生成の終了時に、TIM1 タイマチャネル 1 の各出力はそれらの最後の状態（チャネル 1 の出力はロー、その相補出力はハイ）を維持します。
- このセクションのパート 2 で説明する 2 つ目のアプリケーション例では、N パルス波形が、TIM1 タイマチャネル 1 の出力と相補出力で生成されます。N パルス波形生成の終了時に、TIM1 タイマチャネル 1 の各出力はどちらもローである必要があります。これは、STM32 のタイマに組み込まれている転流機能を使用して達成できます。この機能の詳細については、この章の 2 つ目のパートで説明します。

6.1 タイマ同期の概要

タイマ間の同期や連携した動作のために、STM32 のいくつかのタイマを内部で相互リンクすることができます。タイマ間同期機能を搭載する STM32 のタイマペリフェラルは、TRGO （「Trigger Out」（トリガ出力）の略）という名前の同期出力信号を 1 つだけ備えています。また、ITRx という名前（x の範囲は 1 ～ 4）の 4 つの同期入力が備えられ、STM32 の他のタイマの同期出力に接続されます。タイマ間接続マトリックスについては、STM32 マイクロコントローラのリファレンスマニュアルを参照してください。

タイマ間同期は、マスタ/スレーブコントローラユニットを備えたタイマでのみサポートされます（多少の例外があります）。たとえば、TIM2 タイマペリフェラルはマスタ/スレーブコントローラユニットを備えており、STM32 の他のタイマペリフェラルと同期できます。

TIM2 は、その TRGO 同期出力信号を通じて、他のタイマ内部でイベントをトリガできます。この場合、TIM2 タイマペリフェラルは、マスタタイマとして動作します。また、TIM2 タイマペリフェラルが他のタイマの同期出力信号によってトリガされるように設定することもできます。この場合、TIM2 タイマペリフェラルは、スレーブタイマとして動作します。1 つのタイマペリフェラルを同時にスレーブタイマとしてもマスタタイマとしても動作させることができます。

TIM11 タイマペリフェラルは、マスタ/スレーブコントローラユニットを備えていないタイマペリフェラルの例です。TRGO 出力信号の同期機能は備えていませんが、他のいくつかのタイマペリフェラルのマスタタイマとしては動作できます。これは、同期出力信号として TIM11 タイマチャネル出力を使用することで可能となります。TIM11 タイマチャネル出力は、他のタイマペリフェラルの同期入力に接続されます。

タイマのマスタ設定

タイマペリフェラルがマスタタイマとして設定されている場合、対応する TRGO 同期出力信号は、次に示すいずれかのタイマイベントに続いて同期パルスを出力できます。次に示すリストは全ての場合ではなく、最も一般的なモードのみが示されていることに注意してください。マスタモードのリストは、マイクロコントローラごとに異なる場合があります。

- リセット：EGR レジスタの UG ビットがトリガ出力（TRGO）として使用されます。
- イネーブル：カウンタイネーブル信号がトリガ出力（TRGO）として使用されます。これは、いくつかのタイマを同時に開始するときや、スレーブタイマが有効な時間枠を制御するときに使用されます。
- 更新：更新イベントがトリガ出力（TRGO）として選択されます。たとえば、マスタタイマをスレーブタイマのプリスケーラとして使用できます。
- パルス比較：CC1IF フラグがセットされている場合（すでにハイであった場合も）、キャプチャまたは比較一致が発生するとすぐに、トリガ出力が正のパルスを送信します。
- OC1Ref：OC1REF 信号がトリガ出力（TRGO）として使用されます。
- OC2Ref：OC2REF 信号がトリガ出力（TRGO）として使用されます。
- OC3Ref：OC3REF 信号がトリガ出力（TRGO）として使用されます。
- OC4Ref：OC4REF 信号がトリガ出力（TRGO）として使用されます。

同期出力としてタイマイベントまたは内部信号を使用するように設定するには、TIMx_CR2 タイマ制御レジスタ 2 の MMS （マスタモード選択）制御ビットフィールドに適切な値を書き込む必要があります。

タイマのスレーブ設定

マスタ/スレーブコントローラユニットを内蔵する各タイマペリフェラルは、他のタイマの同期出力にすでに接続されている 4 つの同期入力を備えています。一度にアクティブになることができる同期入力は 1 つだけであり、どの同期入力がアクティブになるのかを選択するには TS （トリガ選択）制御ビットフィールドを使用します。

タイマペリフェラルの同期入力でアクティブエッジを検出することによって、ペリフェラル内部で「更新イベント」、カウンタリセット、カウンタインクリメントなどのタイマイベントを 1 つトリガできます。これは、タイマのスレーブモード制御レジスタ（TIMx_SMCR）の SMS （スレーブモード選択）制御ビットフィールドに設定されている値によって決まります。

使用する同期入力を選択するため、スレーブタイマは入力トリガ経由でマスタタイマに接続されています。各 ITRx は内部的に別のタイマに接続されます。この接続は、各 STM32 製品に固有です。

STM32 のタイマの転流機能

転流機能は、プリロード機能と組み合わせて使用され、タイマの内部入力または外部入力の 1 つを経由して供給される外部イベントと完全に同期してタイマのチャネル設定を変更します。適用できる設定には、チャネル出力モード、チャネルの有効/無効などがあります。内部入力の例として ITRx 入力、外部入力の例として ETR、TI1、TI2 があります。

セクション 1：STM32 の汎用タイマの基本的な動作モードで説明しているように、OCxM、CCxE、および CCxNE の各制御ビットフィールドは、プリロード機能を備えています。これらのビットフィールドでプリロード機能が有効な場合、それらへの書込みアクセスではタイマチャネル動作モードは変化しません。なぜなら、書込み動作は、それらの制御ビットフィールドのプリロードインスタンスに対して実行されるからです。それらのアクティブインスタンス（タイマチャネルの出力モードを実質的に制御します）は、変化しません。

タイマ内部で転流イベントが生成されるとすぐに制御ビットフィールドのプリロードインスタンスの内容がアクティブインスタンスに転送され、結果的にチャネル出力モードを変更できます。

タイマ制御レジスタ 2（TIMx_CR2）の CCUS 制御ビットフィールドの設定に応じて、タイマのトリガ入力信号（TRGI）のアクティブエッジを検出した後に転流イベントを生成できます。特に、タイマペリフェラルのタイマアクティブ同期 ITRx 入力のアクティブエッジを検出した後で生成できます（ITRx 入力は TRGI 信号のソースの一部です）。

1 つのタイマを制御して、もう 1 つのタイマでタイマ間同期によって転流イベントをトリガする必要がある場合に、この機能を使用できます。別の方法として、ソフトウェアで TIMx_EGR タイマレジスタの COMG 制御ビットフィールドをセットすることによって、転流イベントを生成することもできます。

6.2 N パルス波形生成アプリケーション例 - パート 1

この例は、設定可能な各期間に特定の数のパルスを生成するために、STM32 タイマの相互接続機能を使用する方法を示します。

アプリケーションの概要

このアプリケーションは、ボードのメインコントローラとして STM32F302x8 マイクロコントローラを搭載する Nucleo ボード NUCLEO-F302R8 をベースに設計されています。ただし、このアプリケーションは、任意の STM32 ハードウェアプラットフォームに簡単に移植できます。Nucleo ボードには、独自の ST-Link/V2 デバッガが組み込まれています。アプリケーションのバイナリイメージをマイクロコントローラにダウンロードしたり、アプリケーションをデバッグするため、Nucleo ボードを PC コンピュータに接続する際に必要なのは、USB ケーブルのみです。

図 34 に、このアプリケーション例で示されている周期的 N パルス生成アプリケーションの概要図を示します。生成された信号は、コネクタ CN9 のピン 8 に割り当てられている STM32F302 マイクロコントローラの PA.08 IO に出力されます。

次の主な特徴が示されています。

- タイマ 2 は、タイマ 1 をトリガするマスタトリガモードとして設定されています。
- タイマ 1 は、スレーブワンパルスモード（OPM）として設定されています。

図 34. 周期的 N パルス生成のブロック図

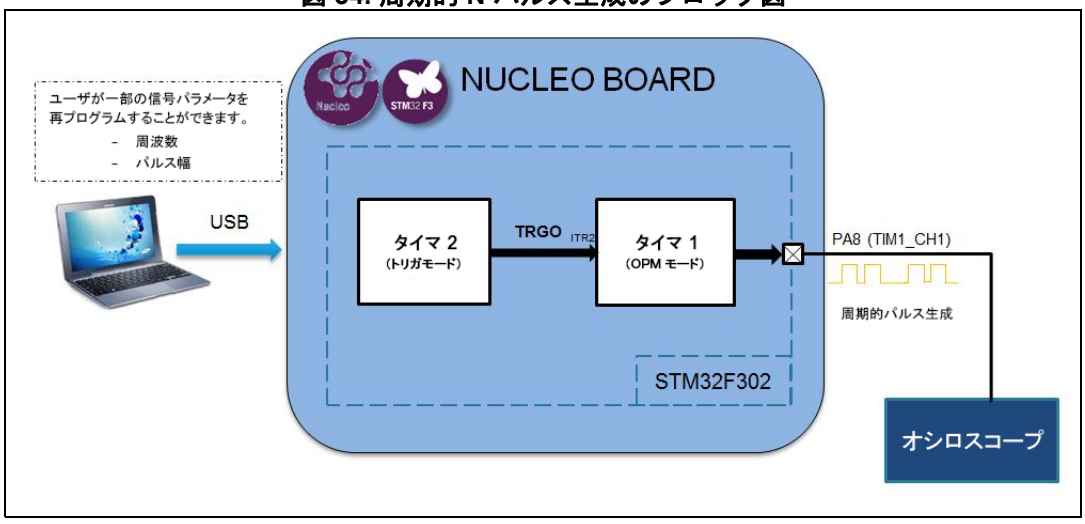
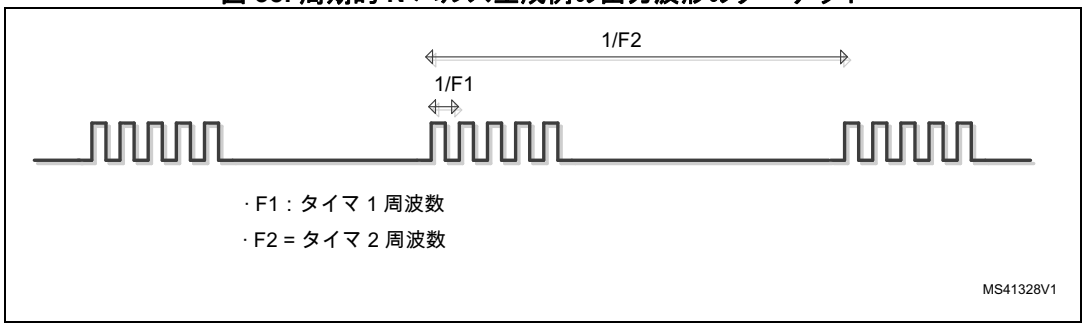


図 35 は、この例で生成しようとしたターゲット波形を示しており、N=5 のパルスを周期的に生成しています。

図 35. 周期的 N パルス生成例の出力波形のターゲット



この例で説明する波形は、図 35 の波形信号のように、PWM サイクルごとに 5 つのパルスを周期的に出力するように設計されています。波形は、プログラムされた 2 つの周期で与えられる 2 つの周波数で構成されています。T2 は TIM2 タイマペリフェラルの周期、T1 は TIM1 タイマペリフェラルの周期です。このアプリケーションのソースコードは、さまざまな波形を出力するように簡単に作り直すことができますが、実証用に図 37 の波形をターゲットにしています。

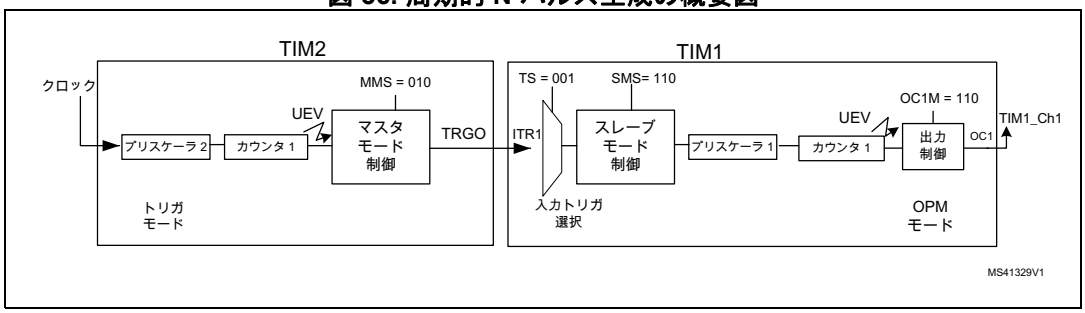
機能詳細

図 35 に関して、ターゲット波形は、周期的 N パルス（この例では 5 つのパルス）による PWM 信号です。パルスは特定のオン時間とオフ時間（t1_on、t1_off、および t2_on）で構成され、合計は T1 に等しくなります。各 T2 周期で複数のパルスが再生成され、PWM サイクルの残りの時間はローレベルを維持します。

STM32 のタイマペリフェラルにこれらの周期的パルスを出力させるには、相互接続されている TIM1 タイマペリフェラルと TIM2 タイマペリフェラルの 2 つのタイマペリフェラルを使用します。TIM1 タイマペリフェラルは、TIM1_CCR1 にプログラムされている T1_on デューティサイクルおよび TIM1_ARR にプログラムされている T1 周期で PWM 信号を生成します。TIM2 タイマペリフェラルでは、生成される各サイクルの周期がその TIM2_ARR レジスタにプログラムされている T2 に確実に等しくなるようにします。

図 36 に示すように、TIM1 ペリフェラルと TIM2 タイマペリフェラルは、マスタとスレーブとして連続的に相互接続されています。

図 36. 周期的 N パルス生成の概要図



TIM2 タイマペリフェラルはマスタトリガモードに設定されており、更新イベントのたびに内部の TRGO 信号経由で TIM1 タイマペリフェラルの ITR1 タイマ入力をトリガします。2 つの「更新イベント」の間の時間が、ターゲット波形の T2 です。

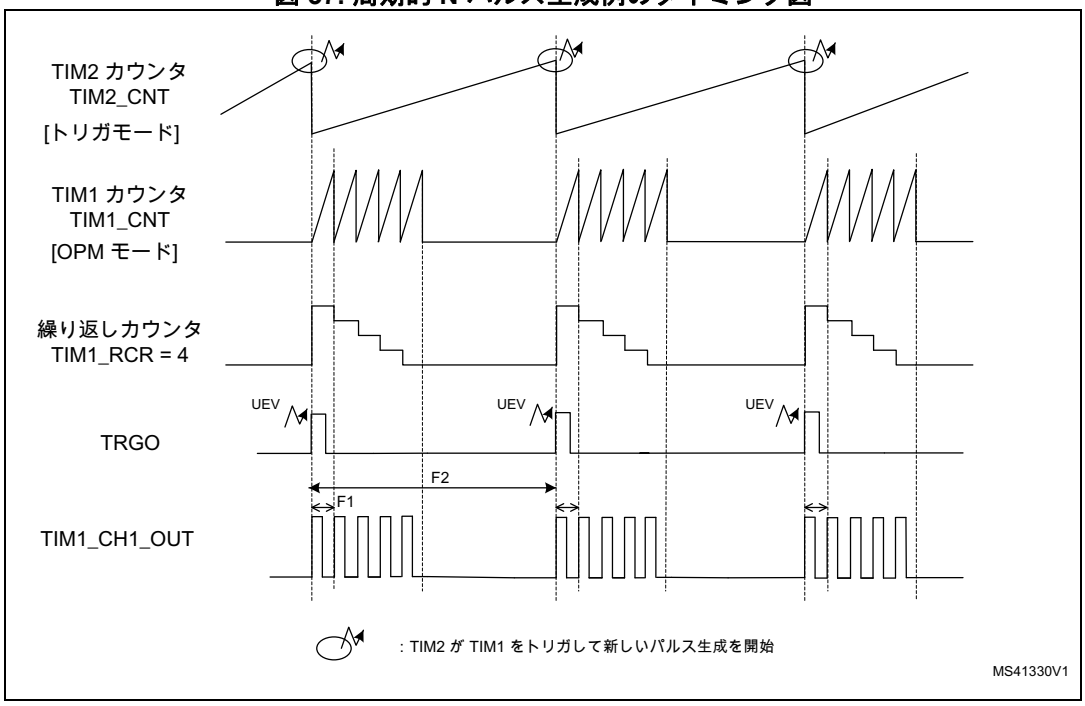
TIM1 タイマペリフェラルはスレーブワンパルスモード（OPM）に設定されており、TIM2 タイマペリフェラルによってトリガされるとパルスを 1 つ生成します。

トリガ入力（ITR1）に立ち上がりエッジが発生すると、カウンタはアップカウントを 1 回だけ開始して、1/F1 に等しい設定周期（TIM1_CCR1）に一致すると、UEV が発生します。

生成されるパルスを繰り返す（アップカウントする）には、TIM1 タイマペリフェラルの**繰り返しカウンタ**を使用します。このカウンタには、処理中の信号波形ごとのパルス数から 1 を引いた数（TIM1_RCR = number_of_pulses_per_portion – 1 = 5-1）を設定する必要があります。UEV は、設定されている回数 TIM1_RCR に 1 を足した値だけアップカウントを繰り返した後にのみ生成されます。

図 37 のタイミング図は、説明した設定の同期したタイマを使用して、どのように目的の波形を得るのかを示しています。

図 37. 周期的 N パルス生成例のタイミング図



ファームウェアの説明

- **システムクロック**
  - PLL （システムクロックソース）: 64 MHz
  - HSI （オシレータ。HSE を Nucleo ボードにはんだ付けする必要はなし）
  - AHB 分周 = 1 / APB1 分周 = 2 / APB2 分周 = 1
- **タイマ 1**

  出力比較モードの TIM1_channel1 を使用して、PWM1 信号を生成します。
  - APB2 プリスケーラ = 64 -1 （タイマクロックとして 1 MHz を得る）
  - 周期（ARR）= 50 µs -> 周波数 F1 = 20 KHz
  - パルス（CCR1）= 周期 / 2 （50%）
  - RCR = 5-1 -> 5 つのパルスを繰り返す
  - ワンパルスモードを選択
  - スレーブモードはトリガを選択
  - 入力トリガ : ITR1
  - PWM モード : モード 1
  - カウントモード : アップカウント
  - 出力比較プリロード : 有効

```
/* 1MHz のカウンタクロックを得られるようにタイマプリスケーラをセットします */
  Tim1Prescaler= (uint16_t) (SystemCoreClock / 1000000) - 1;
/* PWM 周期を初期化して 1 MHz から周波数 20 KHz を取得します */
  Period = 1000000 / 20000;
/* タイマプリスケーラを設定します */
  TIM1->PSC = Tim1Prescaler;
/* 周期を設定します */
  TIM1->ARR = Period-1;
/* 繰り返しカウンタを設定します */
  TIM1->RCR = ((uint32_t) 6) -1;
/* パルス幅を設定します */
  TIM1->CCR1 = Period / 2;
/* クロック分周として 1 を選択します */
/* クロック分周ビットフィールドをリセットします */
  TIM1->CR1 &= ~ TIM_CR1_CKD;
/* クロック分周として DIV1 を選択します */
  TIM1->CR1 |= TIM_CLOCKDIVISION_DIV1;
/* TIM1 カウンタのアップカウントを選択します */
/* モード選択ビットフィールドをリセットします */
   TIM1->CR1 &= ~(TIM_CR1_DIR | TIM_CR1_CMS);
/* アップカウントモードを選択します */
  TIM1->CR1 |= TIM_COUNTERMODE_UP;
/* PWM1 モードをセットします */
/* 出力比較モードビットをリセットします */
```

```
  TIM1->CCMR1 &= ~TIM_CCMR1_OC1M;
  TIM1->CCMR1 &= ~TIM_CCMR1_CC1S;
/* 出力比較モード 1 を選択します */
  TIM1->CCMR1 |= TIM_OCMODE_PWM1;
/************** ワンパルスモード設定 ************/
  /* ワンパルスモードを選択します */
  TIM1->CR1 |= TIM_CR1_OPM;
/*****************************************************/
/******* スレーブモード設定：トリガモード *********/
/* TIM の入力トリガとして TIM_TS_ITR1 信号を選択します */
  TIM1->SMCR &= ~TIM_SMCR_TS;
  TIM1->SMCR |= TIM_TS_ITR1;
/* スレーブモードを選択します */
  TIM1->SMCR &= ~TIM_SMCR_SMS;
  TIM1->SMCR |= TIM_SLAVEMODE_TRIGGER;
/*****************************************************************/
/* 出力比較 1 プリロードを有効にします */
  TIM1->CCMR1 |= TIM_CCMR1_OC1PE;
/* UG ビットをセットして UEV を有効にします */
  TIM1->EGR |= TIM_EGR_UG;
/* TIM1 メイン出力を有効にします */
  TIM1->BDTR |= TIM_BDTR_MOE;
/* 出力極性レベルとしてアクティブローを選択します */
/* 出力極性レベルをリセットします */
  TIM1->CCER &= ~TIM_CCER_CC1P;
/* ロー出力をセットします */
  TIM1->CCER |= TIM_OCPOLARITY_LOW;
/* CC1 出力をハイレベルで有効にします */
  TIM1->CCER |= TIM_CCER_CC1E;
/* TIM カウンタを有効にします */
  TIM1->CR1 |= TIM_CR1_CEN;
```

- **タイマ 2**

出力比較モードの TIM2_channel1 を使用して、PWM1 信号を生成します。

- APB1 プリスケーラ = 64 -1 （タイマクロックとして 1 MHz を得る）
- 周期（ARR）= 20000 µs -> 周波数 F2 = 50 Hz
- パルス（CCR1）= 周期 / 2 （50%）
- 選択するマスタモード：TRGO 更新
- カウントモード：アップカウント

```
/* 1MHz のカウンタクロックを得られるようにタイマプリスケーラをセットします */
  Tim2Prescaler= (uint16_t) ((SystemCoreClock ) / 1000000) - 1;
/* PWM 周期を初期化して 1 MHz から周波数 50 Hz を取得します */
  Period = 1000000 / 50;
```

```
/* 周期を設定します */
  TIM2->ARR = Period-1;
/* タイマプリスケーラを設定します */
  TIM2->PSC = Tim2Prescaler;
/* クロック分周として 1 を選択します */
/* クロック分周ビットフィールドをリセットします */
  TIM2->CR1 &= ~ TIM_CR1_CKD;
/* クロック分周として DIV1 を選択します */
  TIM2->CR1 |= TIM_CLOCKDIVISION_DIV1;
/* TIM1 カウンタのアップカウントを選択します */
/* モード選択ビットフィールドをリセットします */
  TIM2->CR1 &= ~(TIM_CR1_DIR | TIM_CR1_CMS);
/* アップカウントモードを選択します */
  TIM2->CR1 |= TIM_COUNTERMODE_UP;
/****** マスタモード設定：トリガ更新モード *******/
/* TIM2 更新トリガを TIM1 スレーブに送ります */
  TIM1->CR2 &= ~ TIM_CR2_MMS;
  TIM2->CR2 |= TIM_TRGO_UPDATE;
/*************************************************************/
/* TIM カウンタを有効にします */
  TIM2->CR1 |= TIM_CR1_CEN;
```

- GPIO
  - ピン PA8：TIM1_ch1_output
  - モード：プッシュプル
  - プル：プルアップ
  - スピード：ハイ
  - オルタネート機能：GPIO_AF6_TIM1

6.3 N パルス波形生成アプリケーション例 - パート 2

アプリケーションの概要

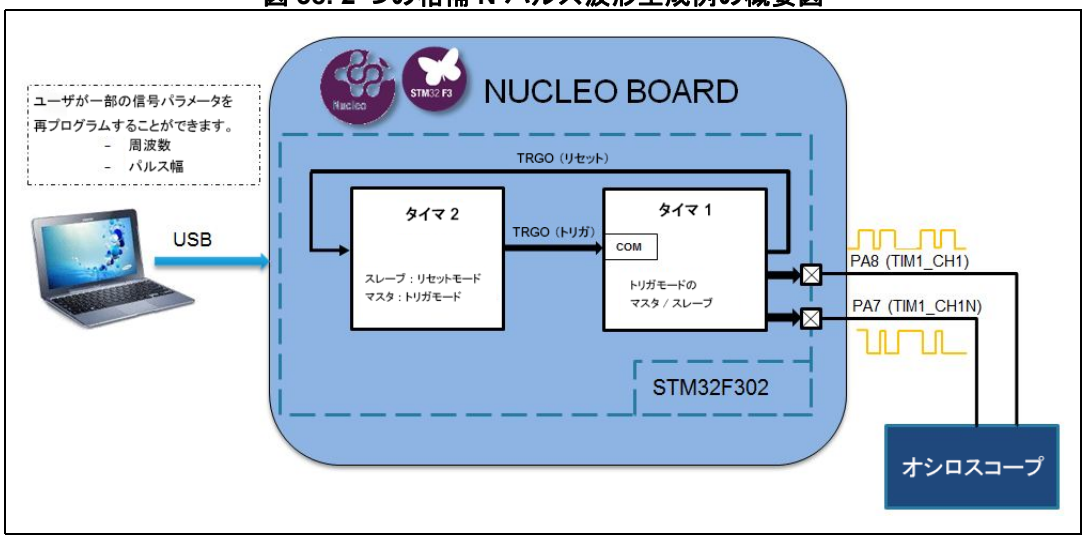
このアプリケーション例は、ボードのメインコントローラとして STM32F302x8 マイクロコントローラを搭載する Nucleo ボード NUCLEO-F302R8 をベースに設計されています。ただし、このアプリケーションは、任意の STM32 ハードウェアプラットフォームに簡単に移植できます。Nucleo ボードには、独自の ST-Link/V2 デバッガが組み込まれています。アプリケーションのバイナリイメージをマイクロコントローラにダウンロードしたり、アプリケーションをデバッグするため、Nucleo ボードを PC コンピュータに接続する際に必要なのは、USB ケーブルのみです。

図 38 に、このアプリケーション例で示されている周期的 N パルス生成アプリケーションの概要図を示します。相補的に生成された両方の信号が STM32F302 マイクロコントローラの PA.08 IO と PA.07 IO に出力されます。これらはそれぞれ CN9 コネクタのピン 8 と CN10 コネクタのピン 26 に割り当てられています。

次の主な特徴が示されています。

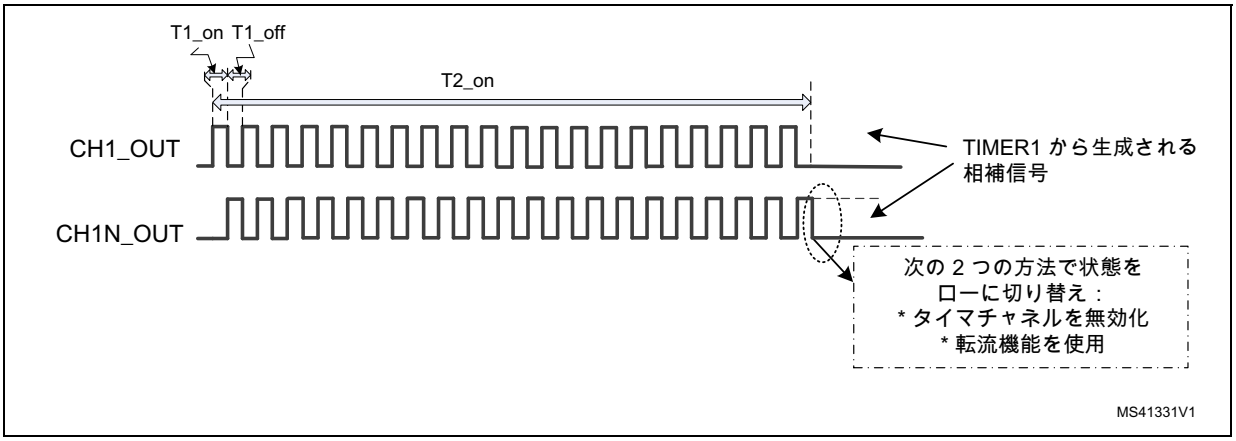
- タイマ 2 はスレーブリセットモードおよびマスタトリガモードとして設定されています。
- タイマ 1 はトリガモードのスレーブとマスタとして設定されています。

図 38. 2 つの相補 N パルス波形生成例の概要図



この例で説明している 2 つの相補 N パルス波形ジェネレータは、それぞれが N パルスで構成され、同じ最終状態を持つ、2 つの相補波形を出力するように設計されています。結果は、図 39 の信号と同様になるはずです。このアプリケーションのソースコードは、さまざまな波形を出力するように簡単に作り直すことができますが、実証用に図 39 の波形をターゲットにしています。

図 39. 2 つの相補 N パルス波形生成例の出力



機能詳細

図 39 の波形のとおり、ターゲット波形は N パルスのみで構成される 2 つの相補 PWM 信号で、このケースでは 20 パルス分です。パルスは特定のオン時間（t1_on）とオフ時間（t1_off）で構成されています。T2_on は、パルスフレームを生成するアクティブ時間です。相補信号は両方とも同じ最終状態で終了する必要があり、図 39 の例では、これはローレベルとなっています。

STM32 のタイマペリフェラルにこれらの PWM を出力させるには、相互接続された 2 つのタイマペリフェラルを使用する必要があります。そのうち 1 つは相補 PWM ジェネレータとして動作し、もう 1 つはパルス生成の最後を制御します。

1 つ目のタイマは、T2_on をカウントして、PWM 出力を停止するためにジェネレータタイマをトリガすることによって、パルス生成の開始と停止のタイミングを制御します。したがって、両方のタイマペリフェラルのカウンタは、開始タイミングを同期させることによって、同時にカウントを開始します。

これを実行する直観的な方法としては、ジェネレータタイマをマスタトリガモードに設定し、もう 1 つのタイマをスレーブリセットモードに設定して、ジェネレータタイマが有効になったら 1 つ目のタイマペリフェラルのカウンタをリセットします。一方で、ジェネレータタイマをスレーブトリガモードとしても設定し、1 つ目のタイマペリフェラルをマスタトリガ更新モードに設定して、T2_on の時間カウントがオーバーフローして PWM 生成を停止するときに「更新イベント」を生成します。

T2_on のカウントが終了して「更新イベント」が生成され、ジェネレータタイマがトリガされたら、割込みルーチンで PWM 生成を停止するため、キャプチャ /比較出力の両方を無効にするか、転流機能を使用して強制的に出力をインアクティブモードにします。

TIM1 タイマペリフェラルで転流更新制御を有効にし、TIM1_CR2 ペリフェラルの CCUS ビットフィールドをセットすることによって TRGI 入力をトリガするように設定する必要があります。タイマ転流割込みソースは、TIM1_DIER レジスタの COMIE ビットをセットすることによって有効になります。COM イベントの次のステップの設定をあらかじめプログラムする必要があります。この設定は、TIM1_CCMR1 レジスタの OC1M [2:0] ビットフィールドを 4 にセットすることによって、強制的にインアクティブレベルになります。

TIM2 が TRGI TIM1 入力をトリガすると、立ち上がりエッジを検出して転流イベント（COM）が発生します。COM 転流イベントで、プリロード OC1M ビットフィールドがシャドウビットに転送されます。最後に、強制的に TIM1 の出力が同時にローになります。

両方のタイマがマスタ/スレーブモードに設定可能である必要があります。また、ジェネレータタイマは、相補出力も備えている必要があります。この例では、TIM1 タイマペリフェラルは、相補信号と転流イベントの両方を出力できる 4 つのタイマチャネルを備えているので、タイマジェネレータの役割で使用する候補として適しています。TIM1 タイマのチャネル 1 とチャネル 1N を使用します。TIM2 はもう一方のタイマとして使用します。

タイマチャネルレジスタの更新によってタイマチャネルで不要なグリッチが生成されないように、STM32 のタイマペリフェラルのキャプチャ /比較プリロード機能を使用します。

無駄な時間サイクルを避けるため、TIM1 タイマペリフェラルでチャネルレジスタを有効にした後で「更新イベント」が生成される前に、TIM1_EGR レジスタの更新生成ビットフィールド（UG）をセットする必要があります。これにより、両方のタイマペリフェラルのカウンタ間の同期タイミングがさらに正確になります。

TIM1 タイマペリフェラルの相補チャネルで図 39 に示す波形を出力するため、TIM1 タイマペリフェラルと TIM2 タイマペリフェラルの両方を次に示すように設定します。

TIM1 タイマペリフェラル：マスタトリガリセットモード/スレーブトリガモード

タイマ 1 をマスタトリガリセットモードに設定して、TIM2 タイマペリフェラルをトリガし、そのカウンタをリセットします。これは、両方のタイマペリフェラルカウンタのカウント開始時間を同期させるために実行します。タイマ 1 は、スレーブトリガモードに設定され、TIM2 が T2_on 期間のカウントを終了したときの更新イベントによるトリガで、信号をローレベルにします。

- **TIM1_ARR**：処理中のパルスの周期（t1_on 期間と t2_off 期間の合計）を格納します。
- **TIM1_CCR1**：パルスの t1_on 期間の長さを格納します。
- **TIM1 タイマペリフェラル**：マスタトリガ更新/スレーブトリガリセットモード

タイマ 2 をマスタトリガモードに設定して、T2_on 期間が終了したときの「更新イベント」によって TRGO 信号を通じてタイマ 1 をトリガします。タイマ 2 をスレーブトリガリセットモードに設定して、TIM1 が有効になったときにリセットされるようにします。

- **TIM2_ARR**：T2_on 期間を格納します。

6.3.1 クロック設定

この例は、1 MHz の TIM1 および TIM2 カウンタクロックを得ることが目的です。

- **TIM1 入力クロック**

TIM1 入力クロック（TIM1CLK）は APB2 クロック（PCLK2）にセットされます。

TIM1CLK = PCLK2 かつ PCLK2 = HCLK なので、TIM1CLK = HCLK = SystemCoreClock = 64 MHz となります。

- **TIM2 入力クロック**

TIM2 入力クロック（TIM2CLK）は APB2 クロック（PCLK2）にセットされます。

TIM2CLK = PCLK2 かつ PCLK2 = HCLK なので、TIM2CLK = HCLK = SystemCoreClock = 64 MHz となります。

- **TIM1 プリスケーラ**

1 MHz の TIM1 カウンタクロックを得るために、プリスケーラは次のように計算されます。

プリスケーラ = (TIM1CLK / TIM1 カウンタクロック) - 1

プリスケーラ = (SystemCoreClock / 1 MHz) - 1

- **TIM2 プリスケーラ**

1 MHz の TIM2 カウンタクロックを得るために、プリスケーラは次のように計算されます。

プリスケーラ = (TIM2CLK / TIM2 カウンタクロック) - 1

プリスケーラ = (SystemCoreClock / 1 MHz) - 1

この例の目標は、TIM1 タイマペリフェラルが、それぞれ周期（t1_on + t1_off）が 500 us、デューティサイクル（t1_on）が 50% のパルスを 20 個出力することです。したがって、タイマペリフェラルを次のようにプログラムする必要があります。

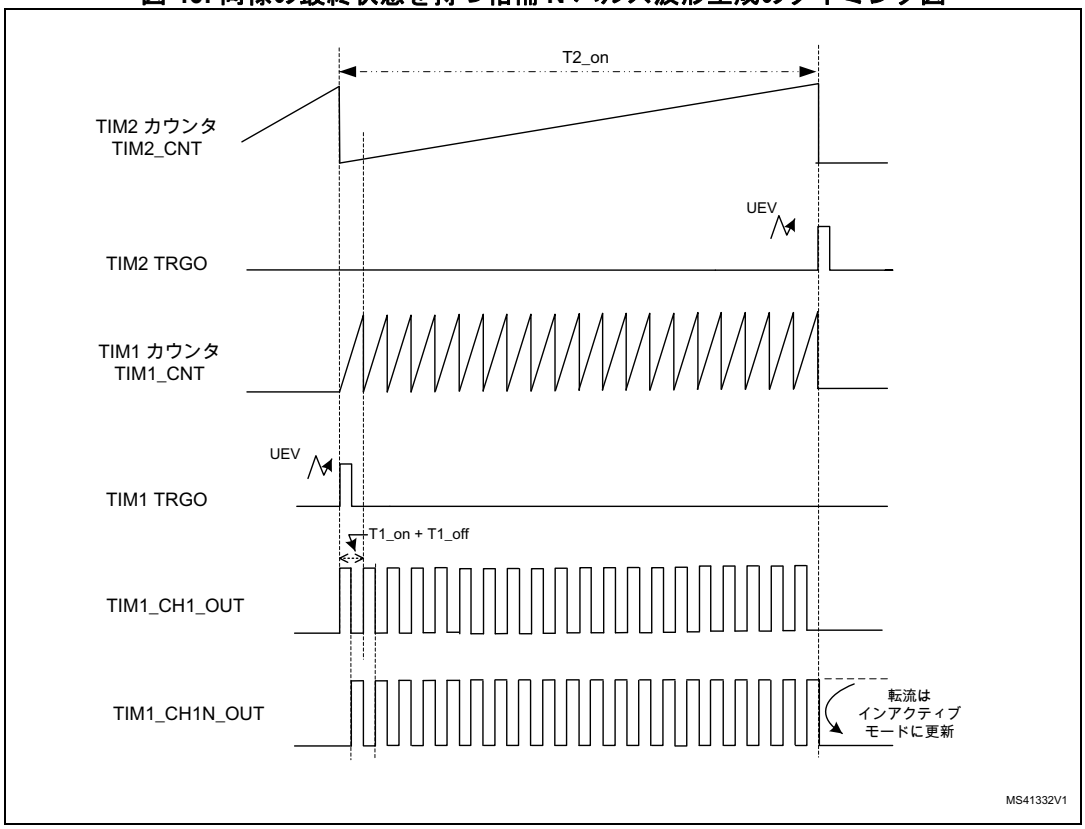
- **TIM1 周期**
  - TIM1_ARR = T1_on+T1_off = 500 µs
- **TIM1 デューティサイクル**
  - TIM1_CCR1 = 250 µs

正確に 20 パルスを得られるようにする、TIM2 タイマペリフェラルに対してプログラムする周期の計算を次に示します。

- **TIM2 周期**
  - TIM2_ARR=T2_ON = 500 x 20 = 10000 µs

図 40 のタイミング図は、説明した設定の同期したタイマを使用して、どのように目的の波形を得るのかを示しています。

図 40. 同様の最終状態を持つ相補 N パルス波形生成のタイミング図



ファームウェアの説明

- システムクロック
  - PLL （システムクロックソース）: 64 MHz
  - HSI （オシレータ。HSE を Nucleo ボードにはんだ付けする必要はなし）
  - AHB 分周 = 1 / APB1 分周 = 2 / APB2 分周 = 1
- **タイマ 1**

TIM1_channel1 を出力比較モードで使用して、PWM1 信号を生成します。

  - APB2 プリスケーラ = 64 -1 （タイマクロックとして 1 MHz を得る）
  - 周期（ARR）= 50
  - 0 us -> 周波数 F1 = 2 KHz
  - パルス（CCR1）= 周期 / 2 （50%）
  - PWM モード : モード 1
  - 出力比較プリロード : 有効
  - 更新生成イベントをセット
  - 転流機能を有効化
  - カウントモード : アップカウント
  - マスタトリガリセットモードを選択
  - 入力トリガ : ITR1
  - スレーブモードのトリガを選択

```
/* 1MHz のカウンタクロックを得られるようにタイマプリスケーラをセットします
(SystemCoreClock = 64 MHz) */
  Tim1Prescaler= (uint16_t) (SystemCoreClock / 1000000) - 1;
/* PWM 周期を初期化して 10000 Hz から周波数 2 KHz を取得します */
Period = 1000000 / 2000;
/* タイマプリスケーラを設定します */
  TIM1->PSC = Tim1Prescaler;
/* 周期を設定します */
  TIM1->ARR = Period-1;
/* パルス幅を設定します */
  TIM1->CCR1 = Period / 2;
/* クロック分周として 1 を選択します */
/* クロック分周ビットフィールドをリセットします */
  TIM1->CR1 &= ~ TIM_CR1_CKD;
/* クロック分周として DIV1 を選択します */
  TIM1->CR1 |= TIM_CLOCKDIVISION_DIV1;
/* TIM1 カウンタのアップカウントを選択します */
/* モード選択ビットフィールドをリセットします */
  TIM1->CR1 &= ~(TIM_CR1_DIR | TIM_CR1_CMS);
/* アップカウントモードを選択します */
  TIM1->CR1 |= TIM_COUNTERMODE_UP;
/* PWM1 モードをセットします */
/* 出力比較モードビットをリセットします */
  TIM1->CCMR1 &= ~TIM_CCMR1_OC1M;
  TIM1->CCMR1 &= ~TIM_CCMR1_CC1S;
/* 出力比較モード 1 を選択します */
  TIM1->CCMR1 |= TIM_OCMODE_PWM1;
/* 出力極性レベルとしてアクティブハイを選択します */
/* 出力極性レベルをリセットします */
  TIM1->CCER &= ~TIM_CCER_CC1P;
/* 出力比較極性をハイにセットします */
  TIM1->CCER |= TIM_OCPOLARITY_HIGH;
/* CC1 出力をハイレベルで有効にします */
  TIM1->CCER |= TIM_CCER_CC1E;
/* 出力相補極性レベルとしてアクティブハイを選択します */
/* 出力 N の状態をリセットします */
  TIM1->CCER &= ~TIM_CCER_CC1NP;
/* 出力 N の極性をハイレベルにセットします */
  TIM1->CCER |= TIM_OCNPOLARITY_HIGH;
/* CC1 出力をハイレベルで有効にします */
  TIM1->CCER |= TIM_CCER_CC1NE;
/****************** COM 制御更新設定 ******************/
```

```
/* キャプチャ / 比較プリロードをセットします */
 TIM1->CR2 |= TIM_CR2_CCPC;
/* CCUS ビットをセットして、トリガ入力に対する COM 制御更新を選択します */
  TIM1->CR2 |= TIM_CR2_CCUS;
/* 転流割込みソースを有効にします */
  TIM1->DIER |= TIM_IT_COM;
/********** マスタモード設定：トリガリセットモード *********/
/* TIM1 のトリガ出力として TIM2 をトリガする更新を設定します */
  TIM1->CR2 &= ~ TIM_CR2_MMS;
  TIM1->CR2 |= TIM_TRGO_RESET;
/************ スレーブモード設定：トリガモード ************/
/* TIM の入力トリガとして TIM_TS_ITR1 信号を選択します */
  TIM1->SMCR &= ~TIM_SMCR_TS;
  TIM1->SMCR |= TIM_TS_ITR1;
/* スレーブモードを選択します */
  TIM1->SMCR &= ~TIM_SMCR_SMS;
  TIM1->SMCR |= TIM_SLAVEMODE_TRIGGER;
/*****************************************************************/
/* UG ビットをセットして UEV を有効にします */
  TIM1->EGR |= TIM_EGR_UG;
/* TIM1 メイン出力を有効にします */
  TIM1->BDTR |= TIM_BDTR_MOE;
/* TIM カウンタを有効にします */
  TIM1->CR1 |= TIM_CR1_CEN;
```

- **タイマ 2**

TIM2_channel1 を出力比較モードで使用して、PWM1 信号を生成します。

- APB1 プリスケーラ = 64 -1 （タイマクロックとして 1 MHz を得る）
- 周期（ARR）= 10000 µs
- カウントモード：アップカウント
- マスタトリガ更新モードを選択：TRGO 更新
- 入力トリガ：ITR0
- スレーブトリガリセットモードを選択

```
/* 1MHz のカウンタクロックを得られるようにタイマプリスケーラをセットします
(SystemCoreClock = 64 MHz) */
  Tim2Prescaler= (uint16_t) ((SystemCoreClock) / 1000000) - 1;
/* PWM 周期を初期化して 1 MHz から周波数 100 Hz を取得します */
  Period = 1000000 / 100;
/* 周期を設定します */
  TIM2->ARR = Period-1;
/* タイマプリスケーラを設定します */
  TIM2->PSC = Tim2Prescaler;
```

```
/* クロック分周として 1 を選択します */
/* クロック分周ビットフィールドをリセットします */
  TIM2->CR1 &= ~ TIM_CR1_CKD;
/* クロック分周として DIV1 を選択します */
  TIM2->CR1 |= TIM_CLOCKDIVISION_DIV1;
/* TIM1 カウンタのアップカウントを選択します */
/* モード選択ビットフィールドをリセットします */
  TIM2->CR1 &= ~(TIM_CR1_DIR | TIM_CR1_CMS);
/* アップカウントモードを選択します */
  TIM2->CR1 |= TIM_COUNTERMODE_UP;
/********** マスタモード設定：トリガ更新 ***********/
/* TIM2 更新トリガを TIM1 スレーブに送ります */
  TIM2->CR2 &= ~TIM_CR2_MMS;
  TIM2->CR2 |= TIM_TRGO_UPDATE;
/********** スレーブモード設定：トリガモード ************/
/* TIM の入力トリガとして TIM_TS_ITR0 信号を選択します */
  TIM2->SMCR &= ~TIM_SMCR_TS;
  TIM2->SMCR |= TIM_TS_ITR0;
/* スレーブモード選択：トリガリセットモード */
  TIM2->SMCR &= ~TIM_SMCR_SMS;
  TIM2->SMCR |= TIM_SLAVEMODE_RESET;
/*************************************************************/
/* TIM1 カウンタを有効にします */
  TIM2->CR1 |= TIM_CR1_CEN;
```

- **GPIO**
  - ピン PA8：TIM1_CH1 出力
  - ピン PA7：TIM1_CH1N 出力
  - モード：プッシュプル
  - プル：プルアップ
  - スピード：ハイ
  - オルタネート機能：GPIO_AF6_TIM1
- **転流更新制御**

```
/* CCMR1 レジスタの OC1M ビットをリセットします */
TIM1->CCMR1 &= TIM_CCMR1_OC2M;
/* CCMRx レジスタの OC1M ビットをインアクティブモードに設定します */
TIM1->CCMR1 |= TIM_OCMODE_FORCED_INACTIVE;
```

参考資料

7 改版履歴

表 2. 文書改版履歴

日付	版	変更内容
2016 年 6 月 23 日	1	初版発行
2017 年 5 月 03 日	2	セクション 1.4.2 : タイマレジスタのプリロード機能および図 6 : タイマチャネルレジスタのプリロードメカニズム - 有効な場合を更新。

表 3. 日本語版文書改版履歴

日付	版	変更内容
2018 年 6 月	1	日本語版　初版発行

重要なお知らせ（よくお読み下さい）

STMicroelectronics NV およびその子会社（以下、ST）は、ST製品及び本書の内容をいつでも予告なく変更、修正、改善、改定及び改良する権利を留保します。購入される方は、発注前にST製品に関する最新の関連情報を必ず入手してください。ST 製品は、注文請書発行時点で有効なST の販売条件に従って販売されます。

ST製品の選択並びに使用については購入される方が全ての責任を負うものとします。購入される方の製品上の操作や設計に関してST は一切の責任を負いません。

明示又は黙示を問わず、STは本書においていかなる知的財産権の実施権も許諾致しません。

本書で説明されている情報とは異なる条件でST 製品が再販された場合、その製品についてSTが与えたいかなる保証も無効となります。

ST およびST ロゴはSTMicroelectronics の商標です。その他の製品またはサービスの名称は、それぞれの所有者に帰属します。

本書の情報は本書の以前のバージョンで提供された全ての情報に優先し、これに代わるものです。

この資料は、STMicroelectronics NV 並びにその子会社(以下ST)が英文で記述した資料（以下、「正規英語版資料」）を、皆様のご理解の一助として頂くためにSTマイクロエレクトロニクス㈱が英文から和文へ翻訳して作成したものです。この資料は現行の正規英語版資料の近時の更新に対応していない場合があります。この資料は、あくまでも正規英語版資料をご理解頂くための補助的参考資料のみにご利用下さい。この資料で説明される製品のご検討及びご採用にあたりましては、必ず最新の正規英語版資料を事前にご確認下さい。ST及びSTマイクロエレクトロニクス㈱は、現行の正規英語版資料の更新により製品に関する最新の情報を提供しているにも関わらず、当該英語版資料に対応した更新がなされていないこの資料の情報に基づいて発生した問題や障害などにつきましては如何なる責任も負いません。