PLECS
Tutorial

Efficient PWM Generation

効率的なPWM生成

C言語入力ブロックとステートマシンを使用した変調器の実装

Tutorial Version 1.0

KESCO KEISOKU ENGINEERING SYSTEM
計測エンジニアリングシステム株式会社
https://kesco.co.jp

1 はじめに

C言語入力ブロックは、数学関数や制御関数の実装だけでなく、ステートマシンプログラムの実装にも役立ちます。ステートマシンプログラムは、スイッチングパターンの生成やコントローラモードのシーケンス処理に役立ちます。この演習では、C言語入力ブロックを使用してステートマシンプログラムを実装し、スイッチング遷移時にブランキング時間を伴った対称パルス幅変調(PWM)信号を作成します。

可変タイムステップ設定のステートマシンプログラムを使用してPWMを生成すると、PWMステートマシンプログラムはスイッチング遷移中にのみ呼び出されるため、非常に効率的です。この手法は、変調指数を三角搬送波と連続的に比較する標準的な手法よりも効率的です。比較ベースの手法では、比較器出力の正確な遷移時間を決定できるように、各遷移ポイントの周囲で中間シミュレーションステップを実行する必要があります。

始める前に: ファイル`test_inverter.plecs`が作業ディレクトリにあることを確認します。演習の各段階で作成したモデルと、参照モデルとを比較して確認します。

2 背景

PWMは、図1に示すフルブリッジ単相インバータなどのインバータの制御によく使用されます。PWM波形を生成するには通常、変調指数mと呼ばれる正弦波制御信号を、図2に示すような三角形のPWMキャリアと比較します。さまざまなPWMスイッチング手法とスイッチング信号が可能ですが、この演習では、PWM出力はインバータの4つのスイッチすべてを制御する[-1、0、1]のセットのスイッチング信号です。1はスイッチペアS1とS4をオンにし、-1はスイッチペアS2とS3をオンにします。出力が0の場合、すべてのスイッチはオフになります。結果として得られる出力電圧はバイポーラであり、V_{dc}または$-V_{dc}$の値を持つため、これはバイポーラスイッチング手法です。

図1: フルブリッジ単相インバータ

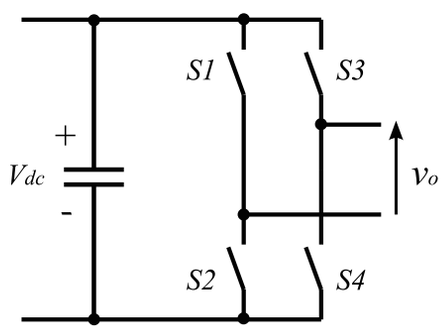
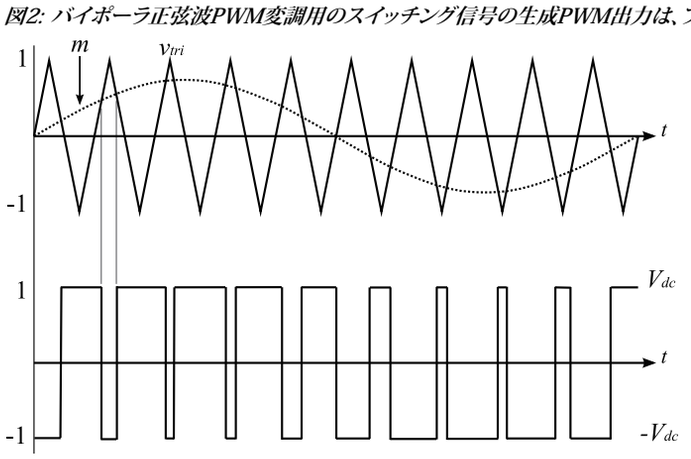


図2: バイポーラ正弦波PWM変調用のスイッチング信号の生成PWM出力は、フルブリッジスイッチペアを制御してバイポーラ電圧出力を生成



実際のインバータでは、スイッチング要素のオンとオフの際の物理的な遅延が重なり合ってDCバスに短絡が発生しないようにするために、スイッチング遷移間にデッドタイムが必要です。デッドタイム制御は、インバータレッグ内の1つのスイッチがオフになるまで一定時間待機してから、レッグの他のスイッチがオンにすることで実現します。

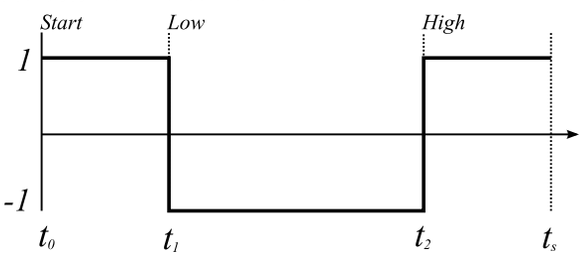
デッドタイムを持つPWM変調器は、スイッチング遷移時にのみ呼び出されるステートマシンを使用してPLECSで効率的にモデリングできます。理想的なPWM変調器の場合、サイクルごとに必要なスイッチング遷移は2つのみで、デッドタイムの場合、追加の遷移は2つ必要になります。

デッドタイムを持つPWM変調器は、スイッチング遷移時にのみ呼び出されるステートマシンを使用してPLECSで効率的にモデリングできます。

3 演習: 理想的なPWM

この最初の演習では、図3のような理想的な対称PWM信号を生成するステートマシンプログラムを実装します。バイポーラスイッチングアプリケーションで使用する場合、入力信号 [-1, 1]の範囲にあり、出力は[-1, 1]のセット内にあります。

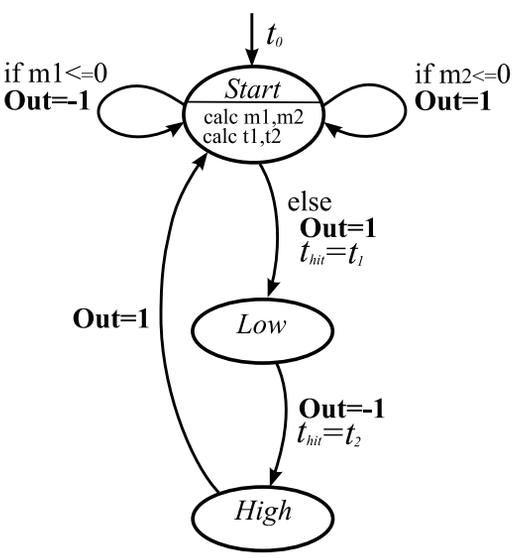
図3: 対称PWMのサイクルタイミング



3.1 PWMステートマシンの動作

ステートマシン プログラムは、図4に示す状態図にまとめられています。状態図は**Start**、*Low*、*High*の3つの状態で構成され、各状態はプログラムが呼び出される時点に対応します。

図4: 対称PWMシーケンス生成の状態図。LowおよびHigh状態への可変遷移時間は、t_{hit}を設定することで定義します。t=0またはHigh状態では、自動固定ステップヒットによってStart状態への遷移が開始されます。



PWMステートマシンは、離散変数ハイブリッド時間設定を使用します。固定時間ステップ設定では、$1/f_s$間隔でC言語入力ブロックへの定期的な呼び出し、つまりヒットを作成します。この設定は、スイッチングサイクルを開始するために使用します。可変時間ステップ設定は、*Low*状態と*High*状態への遷移時間を設定するために使用します。

まず、固定時間ステップのヒットにより、ステートマシンは*Start*状態になります。*Start*状態では、変調指数mが1または0でない場合、スケーリングされた変調指数m_1とm_2が計算されます。スケーリングされた変調指数は、次を使用して範囲[-1, 1]から範囲[0, 1]に変換されます:

$$m_1 = (m + 1) /2 \tag{1}$$

$$m_2 = 1 - m_1 \tag{2}$$

m_1またはm_2が0以下の場合、PWM出力が設定され、ステートマシンプログラムは*Start*状態に戻ります。m_1とm_2が制限内であれば、可変ステップヒット時間t_1とt_2は次のように計算されます:

$$t_1 = t_0 + \frac{m_1}{2} t_s \tag{3}$$

$$t_2 = t_1 + \mathrm{m}_2 t_s \tag{4}$$

対称PWMのヒット時間計算の導出は、付録Aの図7にグラフで示されています。

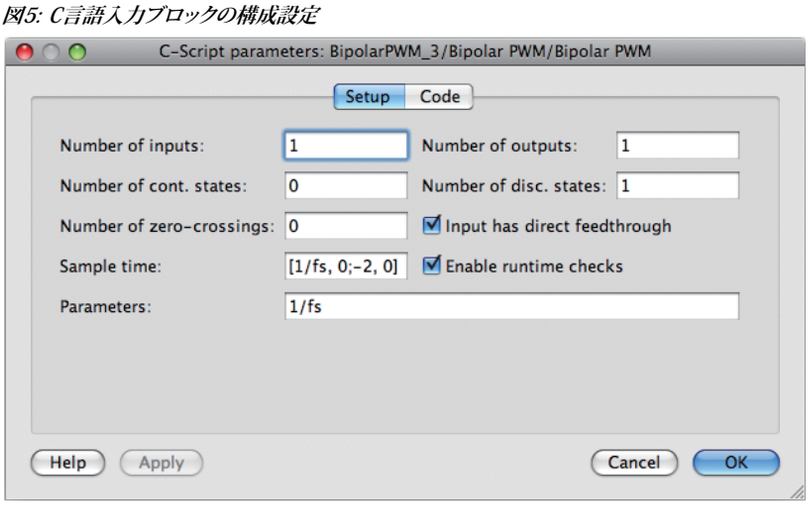
3.2 実装

あなたのタスク:

1 C言語入力ブロックを**サブシステム**ブロック内に配置し、PWMスイッチング周波数を設定するために、**Switching frequency**というラベルの付いた変数f_sを使用してサブシステムパラメータを定義できるようにする必要があります。また、$1/f_s$をカスタムパラメータとしてC言語入力ブロックに渡す必要があります。

2 C言語入力ブロックの構成設定を図5に示します。**サンプリング時間**パラメータは、離散変数ハイブリッド時間設定で構成されていることに注意してください。固定時間設定では、ヒット時間は$1/f_s$の一定間隔で発生し、可変時間設定では、各サイクル中にPWM波形遷移のヒット時間を内部的に定義できます。

図5: C言語入力ブロックの構成設定



3 C言語入力ブロックの構成を完了するには、**コード入力**の**宣言文**で入力を`M`に、出力を`OUT`にマップし、`double`型の変数`TS`を定義します。**Start function**では、マクロ`ParamRealData`を使用して、ユーザパラメータ$1/f_s$を`TS`に割り当てます。

4 ステートマシンをCコードで実装するには、まずstate_typeという名前のデータ型と、そのデータ型に可能な名前を定義する必要があります。これを行うには、次のように **宣言文**に列挙型を作成します:

```
typedef enum {
  Start,
  Low,
  High,
}state_type;
```

5 intやdoubleなどの他の型と同様に、state_type型の変数を作成できるようになりました。次の状態を追跡するには、この型の変数をNEXT_STATEという名前にする必要があります。**宣言文**で定義されている変数はグローバルであるため、永続的であることに注意してください。

6 また、**宣言文**には、double型で表現可能な最大の数DBL_MAXを定義するヘッダファイル*float.h*を組み込み、これを NEVERという名前のマクロに割り当てます。また、最小の許容デューティ比を表すDMINという名前のマクロを作成し、これを任意の小さな値1e-6sに割り当てます。

7 宣言文では、STATEという名前の変数をDiscState(0)にマップする必要もあります。変数STATEは内部的にはdoubleとして表されますが、STATE変数をdoubleではなく状態名と比較できるようにするためにstate_type変数としてキャストできることに注意してください。

8 **コード入力**の**初期化関数**で、STATEを*Start*状態に設定し、内部マクロNextSampleHitをNEVERに初期化します。

9 **コード入力**の**出力関数**で、m_1とm_2を倍精度数値として定義します。また、変数t_1とt_2を静的な倍精度数として定義します。これらの変数は **出力関数**の呼び出し間で値を保持する必要があるため、staticキーワードが必要です。

10 以下に示すswitchステートメントを使用して、**出力関数**にステートマシンロジックを実装します。各状態でPWM出力と NextSampleHitマクロを設定し、NEXT_STATEを定義する必要があります。*High*状態では、NextSampleHitマクロをNEVERに設定して、ステートマシンプログラムへの次の呼び出しが固定ステップ設定から行われるようにします。

```
switch ((state_type) STATE) {
  case Start:
  //Calculate m1, m2. Determine NEXT_STATE.
  //Calculate t1, t2. Set PWM output.
  //Set NextSampleHit
  break;

  case Low:
  break;

  case High:
  break;
}
```

11 *Start*状態にいくつかの状態遷移ロジックを実装する必要があります。m_1またはm_2がDMINより小さい場合は、PWM出力を1または-1に設定し、*Start*状態にループバックします。

12 **更新関数**でSTATEをNEXT_STATEに更新することを忘れないでください。

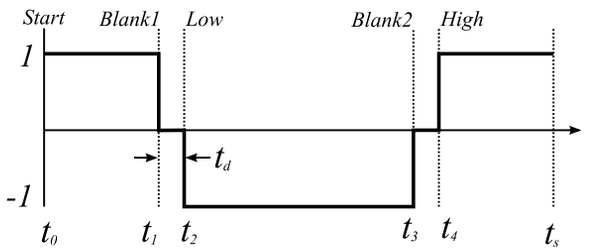
13 PWM変調器をテストするには、振幅1の50Hz正弦波を入力に適用し、Switching frequencyを1000Hzの低い値に設定します。シミュレーションの**終了時間**を20msに設定し、他のすべてのシミュレーションパラメータはデフォルトのままにしておきます。

この段階では、モデルは参照モデルcscript_modulator_1.plecsと同じになっているはずです。

4 演習: ブランキング時間を備えたPWM

この演習では、前の演習のPWMステートマシンを拡張して、スイッチング遷移間のブランキング時間t_dを含めます。また、PWM波形の最小の高時間または低時間をd_{min}に制限します。状態図を変更して、それぞれ*Low*状態と*High*状態の前に、ブランキング状態*Blank1*と*Blank2*を追加する必要があります。状態遷移のヒット時間を図6に示します。

図6: スイッチング遷移間のブランキング時間t_dを伴う単一スイッチングサイクルのタイミング



あなたのタスク:

1 サブシステムに**Dead time ratio**(dr)および`Minimum duty cycle`(d_{min})というラベルの付いた新しいマスクパラメータを作成し、これらをカスタムパラメータとしてC言語入力ブロックに渡します。**宣言文**で`DEADTIME`および`DMIN`という名前の`double`型の変数を定義し、`ParamRealData`マクロを使用してユーザパラメータd_rおよびd_{min}を割り当てます。カスタムパラメータ1/f_sが前と同じように入力され、`TS`にマップされていることに注意してください。

2 新しいオンとオフ比は、式(1)と(2)で計算された値からデッドタイム比d_rを差し引くことにより、デッドタイムに合わせて調整されます:

$$m'_1 = m_1 - d_r \qquad (5)$$

$$m'_2 = m_2 - d_r \qquad (6)$$

3 $m_{1,2} < d$の場合、$m'_{1,2}$の計算値は負になり、ヒット時間の計算は不正確になります。したがって、ヒット時間を計算する前に、m'_1とm'_2にd_{min}の下限を設定し、$1 - 2d_r - d_{min}$の上限を設定します。

4 付録Aでグラフ化されている各状態遷移のヒット時間は、次のように計算されます:

$$t_1 = t_0 + \frac{m'_1}{2} t_s \qquad (7)$$

$$t_2 = t_1 + d_r\, t_s \qquad (8)$$

$$t_3 = t_2 + m'_1\, t_s \qquad (9)$$

$$t_4 = t_3 + d_r\, t_s \qquad (10)$$

5 ステートマシンプログラムを変更して、状態*Blank1*と*Blank2*を含めます。これらの状態をステートマシンモデルに追加するときは、*Start*状態から2つの条件ループも削除する必要があります。デューティ比はゼロより上に制限されるため、これらは不要になりました。

6 最小デューティ比を`0.02`、Deadtime ratioを`0.01`に設定し、変調器の機能を以前のバージョンと比較します。変調器をテストモデル`test_inverter.plecs`に配置します。Switching frequencyを`25e3`Hzに設定し、インバータの出力電圧と負荷電圧を観察します。Deadtime ratioを増やすと、負荷電圧に目に見える変化はありますか?

この段階では、モデルは参照モデル`cscript_modulator_2.plecs`と同じになっているはずです。

5 まとめ

ステートマシンプログラムは、ブランキング時間を備えたPWMスイッチング信号などの出力シーケンスを作成するのに役立ちます。この演習では、PWM遷移時にのみ実行するステートマシンプログラムを使用してPWM信号を生成する方法を学習しました。この実装は、カウンタと比較プログラムに基づく実装よりも効率的です。ステートマシンの概念は、パターン生成に役立つだけでなく、制御システムのシーケンスなどのアプリケーションにおいて、内部イベントではなく外部イベントに応答するように簡単に適応させることができます。

6 付録:サンプリングPWMのヒット時間の導出

図7: 対称PWMのタイミング

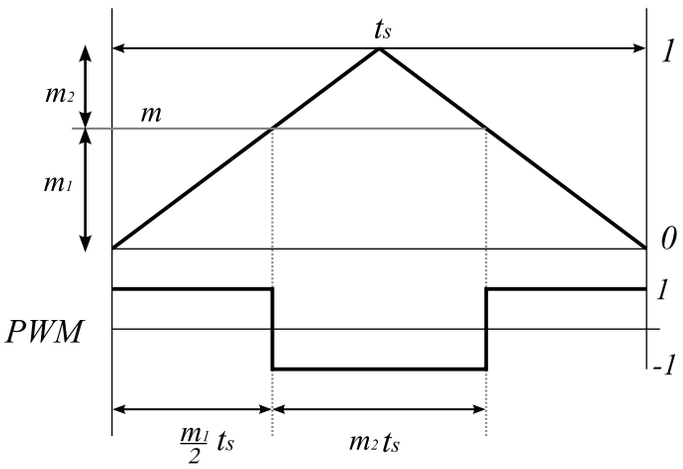
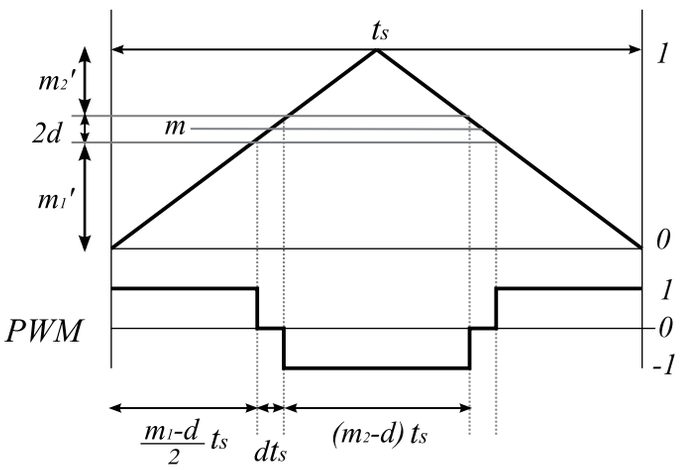


図8: ブランキングを備えた対称PWMのタイミング



改訂履歴:

Tutorial Version 1.0　　初版

Plexim

Pleximへの連絡方法:

☎ +41 44 533 51 00　Phone

+41 44 533 51 01　Fax

✉ Plexim GmbH　Mail

Technoparkstrasse 1

8005 Zurich

Switzerland

@ info@plexim.com　Email

http://www.plexim.com　Web

KESCO KEISOKU ENGINEERING SYSTEM

計測エンジニアリングシステム株式会社

https://kesco.co.jp

PLECS Tutorial