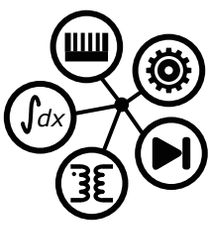


PLECS
DEMO MODEL

Buck Converter with Controls in Co-Simulation

コシミュレーションの制御を備えた降圧コンバータ

PLECSと他のシミュレーションツールとのコシミュレーション用のテンプレート

Last updated in PLECS 4.7.1

Advancing Automation
ADVAN AUTO MATION
アドバン オートメーション株式会社
info-advan@adv-auto.co.jp www.adv-auto.co.jp

1 概要

このデモでは、単純な降圧コンバータ用のデジタルコントローラの実装を紹介します。Controllerブロックは、外部ツールによるコシミュレーションとZドメインコントローラによる離散化デジタル制御(Discrete Digital Control)間で切り替えることができる構成可能なサブシステムを使用します。後者の実装は、demosライブラリの"Buck Converter with Digital Controls"を参照してください。Controllerブロックの実装には、サブシステムのモデル表示(Ctrl+U)でアクセスできます。

Note このモデルには、次からアクセスできるモデル初期化コマンドが含まれています。

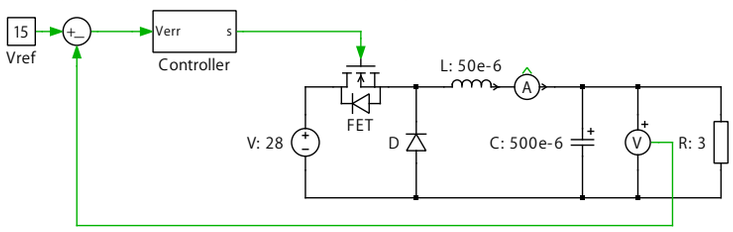
PLECS Standalone: **シミュレーション**メニュー + **シミュレーション・パラメータ...** → **初期化**

PLECS Blockset: **Simulinkモデルウィンドウ**で右クリック → **モデル プロパティ** → **コールバック** → **InitFcn***

2 モデル

図1に、MOSFETを使用した降圧コンバータの回路図を示します。回路は100kHzの固定周波数で動作します。コンバータの出力電圧は、比例/積分/微分(PID)コントローラによって電圧リファレンスに調整されます。コントローラは、離散化デジタルコントローラとして、またはPythonとのコシミュレーションとして構成されます。コシミュレーションの構成については、次のセクションで説明します。

図1: 構成可能なデジタル制御を備えた降圧コンバータ



2.1 離散化デジタル制御

この離散PID制御方法では、Zドメインコントローラを使用します。コントローラは10秒の固定サンプリング時間で動作しますが、PWMブロックは可変時間ステップソルバで評価します。

2.2 コシミュレーション

このコシミュレーションの実装は、PLECSのC言語入力ブロックが実行されます。この場合、コシミュレーションは、コントローラを実行する外部Pythonスクリプト`buck_converter_with_co_simulation.py`によって実行されます。

100*kHz*の各クロックで、PLECSは電圧誤差信号V_{err}をファイル`OutputPLECS.txt`に書き込み、ファイル `OutputPyhton.txt`を削除して、PLECSが完了したことをPythonに知らせます。Pythonは値を読み取り、PIDコントローラのパラメータを更新し、7.5*ns*の時間分解能でPWMパターンを書き込みます。これは、FPGAにとって現実的な解像度かもしれません。Pythonスクリプトは、`outputPLECS.txt`ファイルを削除して、Pythonが完了したことをPLECSに伝えます。PLECSはファイル`outputPython.txt`を読み取り、次のクロック周期までシミュレーションを実行します。シミュレーションステップを表す整数が各ファイルの末尾に追加されます。

これにより、間違ったファイルが読み取られることがなくなり、`wait`ステートメントが回避されます。詳細についてはセクション4.2および5で説明します。

Pythonコードは、PLECS Coderを使用して離散化デジタル制御構成から生成された汎用Cコードから派生します。

3 シミュレーション

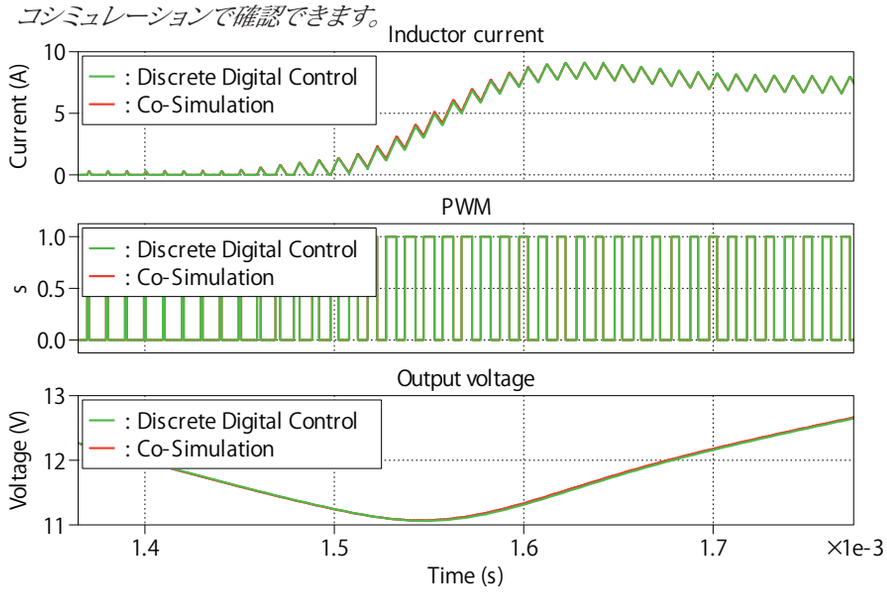
離散化デジタル制御のシミュレーションは他のすべてのPLECSシミュレーションと同様に実行できますが、コシミュレーションはシミュレーションスクリプト(**シミュレーション → シミュレーション・スクリプト...**)を通じてのみアクティブ化できます。

Note これは、オペレーティングシステムがコンソールからPythonスクリプトを直接実行できる場合にのみ機能します。9行目の`wait`ステートメントがPythonスクリプトを開始するのに十分な長さであることを確認してください。

あるいは、Pythonスクリプト`buck_converter_with_co_simulation.py`を適切なランタイム環境で開始することもできます。これを行うには、まず Python スクリプトを開始し、次に9行目を非アクティブにしてシミュレーションスクリプトを開始します。

シミュレーション結果は同様の動作を示しますが、PWMブロックの評価のみが異なります。Python実装は7.5*ns*の固定時間ステップで実行されますが、PLECS PWMブロックは可変時間ステップで実行されます。この小さな違いは、図2に示すように、過渡現象が発生したときに顕著に現れます。

図2: シミュレーションでの負荷の変化。離散化デジタル制御ではPWM発生器に可変タイムステップが使用されるため、固定タイムステップによるPWM生成とのわずかな不一致をコシミュレーションで確認できます。



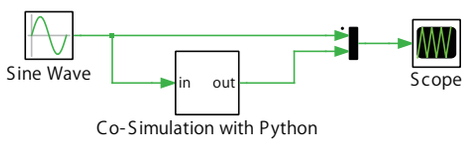
4 テンプレートの概要

以下のセクションでは、PLECSとのコシミュレーション用の簡単なテンプレートを提供します。合計3つのダミーの例を示さします。2つはPythonで、1つはModelSimです。この概念は、FPGAやFEMシミュレータなど、ファイルにデータを書き込むことができるあらゆるソフトウェアに適用できます。

4.1 モデル

PLECSモデルは主に他のシミュレーションツールとの通信を処理するC言語入力で構成しています。この通信チャネルは、`.txt`または`.csv`ファイルを記述することで実現します。この非常にシンプルな通信方法は、さまざまなシミュレーションツールに簡単に適応できます。ただし、シミュレーション間隔を長くする場合、この通信方式は非常に困難であることを考慮する必要があります。

図3: PLECSモデルの主要な回路図



Note このモデルには、次からアクセスできるモデル初期化コマンドが含まれています。

PLECS Standalone: **シミュレーション**メニュー + **シミュレーション・パラメータ...** → **初期化**

PLECS Blockset: **Simulinkモデルウィンドウ**で右クリック → **モデル プロパティ** → **コールバック** → **InitFcn***

4.2 シミュレーション: Pythonで2倍

このコシミュレーションは、PythonおよびPLECSとの同期に焦点を当てています。PLECSからの信号はPythonに読み込まれ、2倍してPLECSに送り返します。コシミュレーションはシミュレーションスクリプト(**シミュレーション** → **シミュレーション・スクリプト...**)を使用するか、demosフォルダの`co_simulation.py`を実行して手動で開始し、PLECSシミュレーションを開始します。

Pythonスクリプトは次の手順で実行します:

1 Pythonスクリプトはダミーファイル`OutputPython_0.txt`を作成します。
2 Pythonは、ファイル`OutputPython_0.txt`がPLECSによって削除されるまで待機します。
3 Pythonは、`outputPLECS_1.txt`を読み取ります。
4 Pythonは値を2倍します
5 Pythonは結果を`outputPython_1.txt`に書き込みます。
6 Pythonは、`outputPLECS_1.txt`ファイルを削除します。これにより、PLECSは、`outputPython_1.txt`を読み取れることがわかります。
7 Goto 2(オーバーフローを避けるために、インデックス番号は2と3の間で切り替わります)

並行して、PLECSは次の手順を実行します:

1 PLECSは、`outputPython_0.txt`が存在するかどうかを確認します。
2 PLECS は1つのタイムステップを実行します。
3 PLECSは、正弦波信号ブロックからの信号を含むファイル`outputPLECS_1.txt`を作成します。

4 PLECSはファイル`outputPython_0.txt`を削除し、Pythonが`outputPLECS_1.txt`の読み取りが許可されていることを認識できるようにします。
5 PLECSは、Pythonがファイル`outputPLECS_1.txt`を削除するまで待機します。
6 PLECSはファイル`outputPython_1.txt`を読み取り、シミュレーションを続行します。
7 Goto 2(オーバーフローを避けるために、インデックス番号は2と3の間で切り替わります)

図4: PythonとPLECS間のコシミュレーションのフロー図。

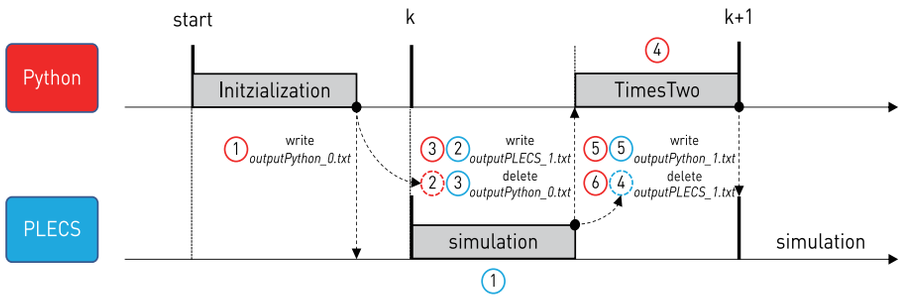
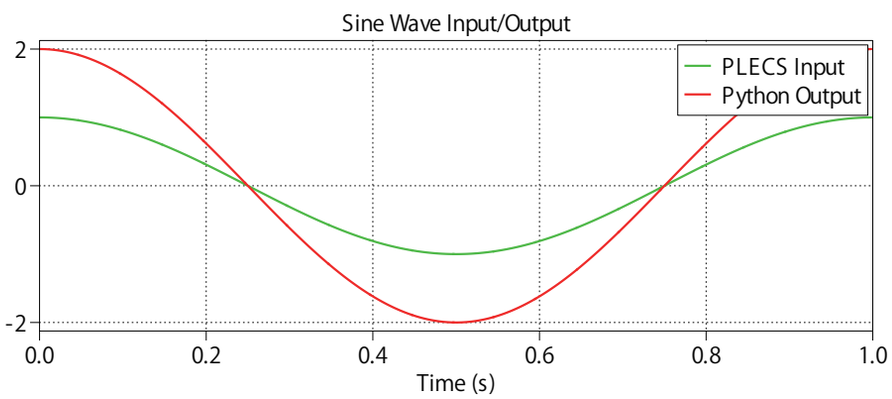


図5: コシミュレーションの結果。C言語入力で選択したサンプル時間0.005秒は、下のプロットにはっきりと表示されます。



4.3 シミュレーション: ModelSimで2倍

このコシミュレーションは、ModelSimおよびPLECSとの同期に焦点を当てています。ModelSimは、VHDLやVerilogなどのハードウェア記述言語をテストするためのシミュレーションツールです。このツールは、無料でダウンロードできるLattice Diamondソフトウェアパッケージに含まれています。上記のコシミュレーションと同様に、信号はPLECSからModelSimに読み取られ、2倍してPLECSに送り返します。混乱を避けるため、ModelSimによって書き込まれたファイルを`modelsim.csv`、PLECSによって書き込まれたファイルを`plecs.csv`と呼びます。

シミュレーションの設定と実行は、次の手順で行うことができます:
1 Lattice Diamondを開く
2 File + **New** + **Project.....**

- Next
- 名前とパスを設定
- Next
- Next(これは単なる汎用テストベンチであるため、FPGAの種類は関係ありません)
- `Synplify Pro`に変更し、Nextをクリック
- 終了

3 File + **Add** + **Existing File...**

- デモモデルのソースフォルダから`tv.sv`と`top.sv`を選択
- プロジェクトを保存してLattice Diamondを閉じる

4 `co_simulation_modelSim.plecs`を上で定義したプロジェクトフォルダにコピー

5 Lattice Diamondでプロジェクトを再度開く

6 Simulation Wizardをクリック

- Next
- プロジェクト名として"tb"(tb := testbench)と入力し、NextとYesをクリック
- Next
- Next
- Next
- 終了

7 シミュレーションが開始され、ModelSimコンソールに“waiting for PLECS to be started...”と書き込まれる。

8 プロジェクトフォルダ内のPLECSモデルを開き、PLECSシミュレーションを開始

9 コシミュレーションが実行され、すべてが完了すると終了

10 Simulation + **Restart...** + **OK**を押してModelSimシミュレーションを再起動し、**Run –All**をクリックします。

このコシミュレーションでは、図5ですでに示したのと同じ結果が生成されます。

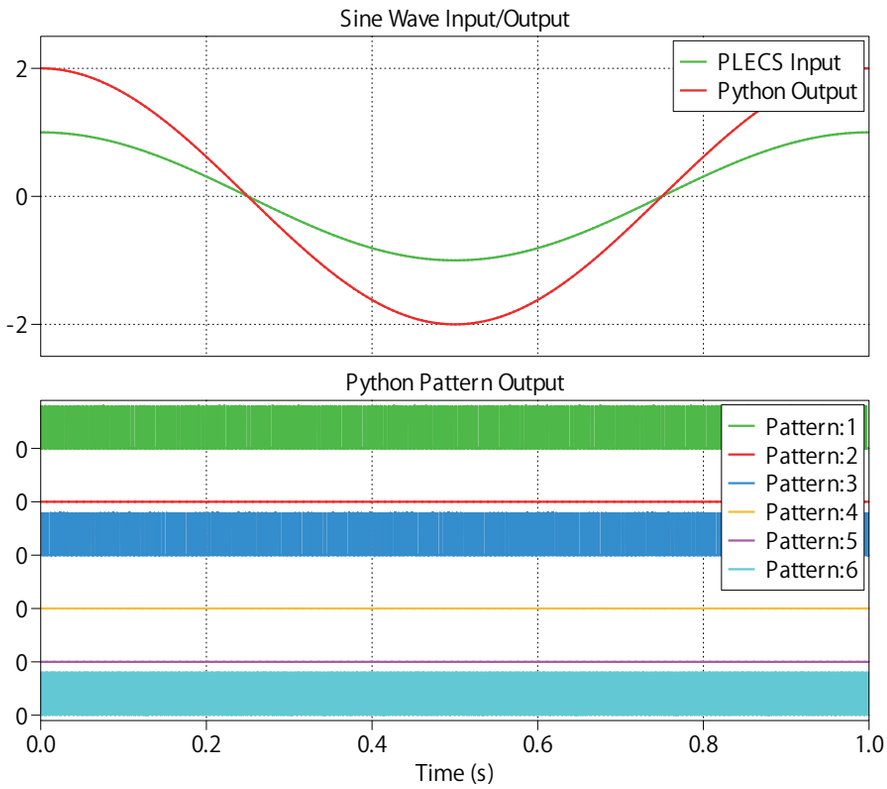
5 シミュレーション: Pythonを使用したパターン

提示された両方のコシミュレーションテンプレートは、各シミュレーションステップの後に通信交換を使用します。ファイルへの書き込みとファイルからの読み取りがボトルネックとなるため、このコシミュレーションのセットアップには時間がかかります。ただし、使用例によっては、このアプローチは必要なく、より大きなステップサイズを選択できる場合があります。例は、スイッチング周期ごとに正確なPWM信号が導出される降圧コンバータの最初の部分に示されています。したがって、パターンはコントローラで計算され、解決のためにパケットとしてPLECSに送信されます。

コシミュレーション`co_simulation_pattern`(および`co_simulation_pattern.py`)はこれを行うことができます。0.005秒のタイムステップでは、パターンが生成され、ファイル`pwm.csv`からロードされます。このパターンはPythonでファイル`OutputPythonPattern.txt`に書き込まれ、このファイルがPLECSにロードされ、コマンド`NextSampleHit`および`IsSampleHit`を使用してこのパターンの長さだけ実行されます。`[2, 0]`[1]のサンプル時間設定。さらに、`outputPython.txt`と`OutputPLECS.txt`の両方のファイルでは、他のコシミュレーションテンプレートと同様に、同じタイムステップに対して同じ信号操作が実行されます。

[1] PLECS *Standalone*のソルバは、`NextSampleHit`マクロを使用して、選択したポイントを正確にヒットできます。残念ながら、Simulinkのs-Functionインタフェースは、ゼロクロッシング法を使用してこの選択された時点を解決することしか提供しません。これは正確にオンではなく、選択した時点に数値的に近いだけであるため、ソルバはサンプルを見逃す可能性があります。この問題を軽減するために、次のタイムステップが選択されます。これにより、最大7:5nsのタイム シフトエラーが発生します。

図6: パターンを使用したコシミュレーションの結果。C言語入力で選択されたサンプル時間0.005秒は、下のプロットにはっきりと表示されます。



6 結論

このデモでは、2つの異なるシミュレーションツールを使用してコシミュレーションをセットアップする方法の簡単なソリューションが示されています。さらに、必要なタイムステップを選択するための2つの異なる可能性が示されています。

改訂履歴:

PLECS 4.7.1　　初版

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

アドバンオートメーションへの連絡方法:

☎ +81 3 5282 7047　Phone
+81 3 5282 0808　Fax
✉ ADVAN AUTOMATION CO.,LTD　Mail
1-9-5 Uchikanda, Chiyoda-ku
Tokyo, 101-0047
Japan
@ info-advan@adv-auto.co.jp　Email
https://adv-auto.co.jp/　Web

PLECS Demo Model