

PLECS Tutorial

Introduction to the C-Script Block

C言語入力ブロックの紹介

デジタルおよびアナログPIコントローラの実装

Tutorial Version 1.0

1 はじめに

C言語入力ブロックは、カスタムコントローラとコンポーネントの実装に使用できる、PLECSコンポーネントライブラリの多目的ツールです。C言語の高度な機能とC言語入力ブロックの柔軟なサンプル時間設定を組み合わせることで、単純な数学関数から複雑なステートマシンまで、ほぼすべてのカスタムコンポーネントモデルを実装できます。C言語入力ブロックは、コードをDSPで再利用できるため、DSPコントローラ用のCコードを作成する際のワークフローを簡素化することもできます。この演習で習得する主なスキルは次のとおりです：

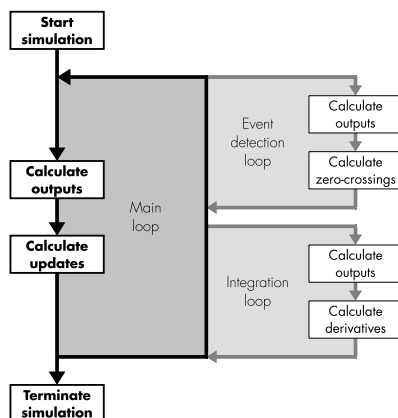
- C言語入力ブロックが関数呼び出しを通じてシミュレーションエンジンとどのように連携するかの理解
- C言語入力ブロックで利用できるさまざまな時間設定の理解
- C言語入力ブロックを使用した数学関数の実装
- C言語入力ブロックを使用した離散および連続系PI制御器の実装

始める前に buck_converter.plecsファイルが作業ディレクトリに配置されていることを確認します。また、演習の各段階で独自のモデルと比較できる参照ファイルも用意する必要があります。

2 関数呼び出しインタフェース

C言語入力ブロックは、いくつかの定義済み関数を呼び出してシミュレーションエンジンと連携します。これらの関数呼び出しを[図1](#)に示します。各関数呼び出しは、プルダウンメニューからアクセスするC言語エディタのコード入力ウィンドウに対応しています。最も一般的に使用するコードウィンドウについては以下で説明します。詳細な説明については、PLECSユーザマニュアル、またはブロックを選択してマウスの右ボタンから**ヘルプ**を選択してヘルプファイルにアクセスします。

図1: C言語入力ブロックの操作中被行われる関数呼び出し。離散状態が定義されるときに更新関数が呼び出され、連続状態が定義されるときに導関数が呼び出されます。



宣言文 関数ウィンドウに加えて、C言語入力ブロック関数で使用するグローバル変数、マクロ、およびヘルパ関数を定義するための**コード宣言**ウィンドウも用意されています。コード宣言コードは本質的にグローバルヘッダファイルです。このウィンドウ内で定義されたすべての変数と関数は、C言語入力ブロック内のすべての関数に対してグローバルに表示されます。

初期化関数 初期化関数ウィンドウは、シミュレーションを初期化するためのものです。たとえば、内部状態変数はここで初期化する必要があります。

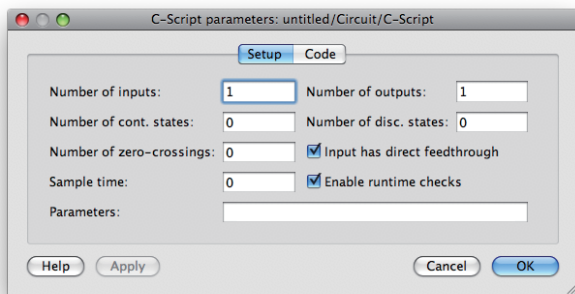
出力関数 出力関数ウィンドウは、C言語入力ブロックの関数コードを格納するように設計されています。各シミュレーション時間ステップ中に、この関数が少なくとも1回呼び出されます。このため、内部状態や固定変数はすべて更新関数で更新する必要があります。

更新関数 モデルに離散的な内部状態が含まれている場合、出力関数が実行された直後に、更新関数ウィンドウのコードが呼び出されます。C言語入力に内部状態が含まれている場合、出力関数ではなくこの関数で内部状態を更新し、タイムステップごとに1回だけ更新されるようにする必要があります。

3 パラメータ

C言語入力ブロックを開くと、[図2](#)に示すように、コードエディタウィンドウの**基本設定**タブが表示されます。このウィンドウでパラメータを構成します。実際には、**コード入力**タブに含まれる関数ウィンドウ内でCコードを記述します。

図2: パラメータを構成し、コードを記述するためのC言語入力エディタウィンドウ。



3.1 サンプリング時間パラメータ

サンプリング時間の設定は、C言語入力ブロックが呼び出されるタイミングを制御する重要なパラメータです。サンプリング時間は、シミュレーションエンジンから継承することも、C言語入力ブロック自体によって制御することもできます。可能なサンプリング時間の詳細は以下の通りです:

連続 連続の時間設定は、サンプリング時間ダイアログに0を入力することによって選択されます。この設定では、時間ステップはソルバから継承されます。ソルバがステップを実行するたびに、C言語入力ブロックが実行されます。

離散 離散周期の時間設定は、サンプリング時間ダイアログに正の数値を入力します。定義されたC言語入力ブロックは、指定されたサンプリング時間に従って実行されます。

可変 可変の時間設定は、サンプリング時間ダイアログに-2を入力します。離散-変数時間設定では、次のタイムステップは組み込みマクロNextSampleHitを設定することにより、C言語入力ブロック自体によって動的に決定されます。NextSampleHitは、シミュレーションの開始時にCurrentTimeマクロ以上の値に初期化する必要があります。C言語入力ブロック内のマクロの詳細については、以下を参照してください。

3.2 他のパラメータ

他のパラメータについては、C言語入力ブロックのドキュメントで詳しく説明されています。ただし、この段階では次のことに注意してください。静的変数を含むC言語入力ブロックを作成するときに、個別の状態を追加してグローバル静的変数を作成できます。離散状態には、マクロコマンドDiscStateを使用してアクセスします。

3.3 よく使われるマクロのリスト

C言語入力ブロックには、モデルまたはソルバとの対話に使用できる組み込みマクロ関数が多数含まれています。よく使用するマクロの一部を以下に示します:

InputSignal(<i>j</i> , <i>i</i>)	<i>j</i> 番目のC言語入力ブロック入力の <i>i</i> 番目の信号を参照。
OutputSignal(<i>j</i> , <i>i</i>)	<i>j</i> 番目のC言語入力ブロック出力の <i>i</i> 番目の信号を参照。
DiscState(<i>i</i>)	インデックス <i>i</i> の離散状態を参照。
NextSampleHit	C言語入力ブロックの次の呼び出し時間を設定。この変数は可変サンプル時間を設定している場合にのみ使用。
CurrentTime	現在のシミュレーション時間を取得。
SetErrorMessage("msg")	エラーメッセージを表示してシミュレーションを中止。

4 演習: 数学関数の実装

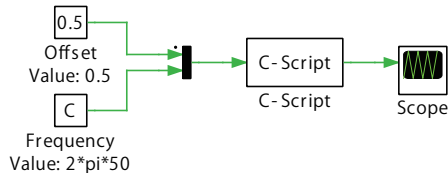
この演習では、C言語入力ブロックを使用して、オフセット値を持つ正弦関数を実装します。



あなたのタスク:

- 新しいシミュレーションモデルを作成し、その中にC言語入力コンポーネントと2つの定数ブロックを配置します。最初の定数ブロックに"Offset"というラベルを付け、その値を0.5に設定します。2番目の定数ブロックに"Frequency"というラベルを付け、その値を $2\pi \cdot 50$ に設定します。マルチプレクサブロックを使用して、2つの定数値をC言語入力ブロックに送ります。PLECSスコープを追加したシミュレーションモデルは図3のようになるはずです。

図3: C言語入力ブロックを使用した関数 $y = 0.5 + \sin(2\pi 50 \cdot t)$ の実装



- 次に、C言語入力ブロックを構成してコードを記述する必要があります。C言語入力ブロックを構成するには、ブロックをダブルクリックして開き、**基本設定**タブで**入力数**を2に、**出力数**を1に設定します。連続または継承されたサンプリング時間を選択するには、**サンプリング時間**設定を0に設定します。正弦関数を記述するには、cmathライブラリ(math.hヘッダ)をインクルードする必要があります。**コード入力**タブの**宣言文**ウィンドウで、次のコードを入力します:

```
#include <math.h>

#define offset InputSignal(0,0)
#define freq InputSignal(0,1)
```

出力関数ウィンドウで、次のコードを入力して正弦関数を作成します:

```
OutputSignal(0,0) = sin(freq*CurrentTime) + offset;
```

- シミュレーションを実行します。**シミュレーションパラメータ**の**ソルバ**タブで次のように設定します:

- 終了時間: 20e-3
- 最大時間刻み幅: 1e-4

シミュレーションを実行すると、周期が20ms、垂直オフセットが0.5Vの正弦波が表示されます。



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

5 演習: デジタルPI制御器の実装

この演習では、降圧コンバータの連続で動作するPI電圧制御器をデジタルPI制御器に置き換えます。降圧コンバータの連続で動作するPI電圧制御器を図4に示します。連続動作のPI制御則は次の関数で記述されます:

$$y(t) = k_p e(t) + k_i \int_0^t e(\tau) d\tau \quad (1)$$

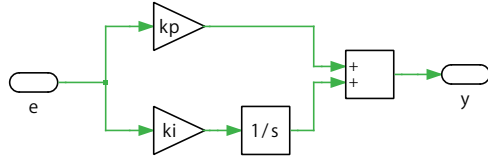
C言語入力ブロックを使用してデジタルPI制御器を実装するには、PI制御則の離散形式を使用する必要があります。PI制御器を離散化する最も簡単な方法は、後退矩形則(Backwards Rectangular rule)を使用して積分項を次のように近似することです:

$$i_k = i_{k-1} + T_s e_k \quad (2)$$

ここで、 i_k はサンプル番号 k の値、 T_s はサンプリング時間です。したがって、デジタルPI制御則は次のようになります:

$$y_k = k_p e_k + k_i i_k \quad (3)$$

図4: 連続動作のPI電圧制御器



5.1 C言語入力ブロックの構成



あなたのタスク: 降圧コンバータ モデルbuck_converter.plecsを開き、連続したPI voltage controllerの実装を確認します。次の手順に進む前に、降圧コンバータモデルのコピーを保存します:

- 1 コンポーネントを右クリックして **サブシステム -> サブシステムのモデル表示**を選択(または**Ctrl + U**)し、PIのマスクの下を確認し、入出力ポートを除くすべてのコンポーネントを削除します。C言語入力ブロックを入力ポートと出力ポートの間に直接配置します。C言語入力ブロック設定の**入力数**と**出力数**が両方とも1に設定されていることを確認します。

- 2 PI voltage controllerのマスクにパラメータ**サンプル周波数(f_s)**追加し、その値を $25\text{e}3\text{Hz}$ に設定します。PI voltage controllerのマスクにパラメータを追加するには、マスクを右クリックして**サブシステム -> マスクの編集...**を選択します(または**Ctrl + M**を使用します)。
- 3 C言語入力のパラメータで、**サンプリング時間**設定を $1/f_s$ に設定します。これにより、C言語入力ブロックは離散周期的、つまり $1/f_s$ の固定サンプリングレートで実行されます。s
- 4 C言語入力のコードでは、パラメータ k_p 、 k_i 、および T_s にアクセスする必要があります。これらを C スクリプト ブロックに直接渡すには、**基本設定**タブにある**パラメータ**ボックスに入力します。**パラメータ**ボックスに変数 k_p 、 k_i 、および $1/f_s$ を入力します。
- 5 **コード入力**タブに切り替えて、**宣言文**で次の変数を定義します。

```
static double kp, ki, Ts;
```

- 6 **初期化関数**で、定義済みの変数に入力パラメータを割り当てます:

```
kp = ParamRealData(0,0);
ki = ParamRealData(1,0);
Ts = ParamRealData(2,0);
```

- 7 **宣言文**では、エラー入力信号を変数 e_k にもマップします:

```
#define ek InputSignal(0,0)
```

5.2 離散状態のデジタル制御則を実装



あなたのタスク: 制御コードは、サンプル期間ごとに1回だけ呼び出されるため、**更新関数**内に記述する必要があります。一方、**出力関数**は通常、サンプル期間ごとに複数回呼び出されます。したがって、出力関数で計算される積分値などの離散状態は不正確になります。

- 1 C言語入力のパラメータで、**離散状態数**の数を1に設定します。これにより、DiscState(0)という名前の静的内部変数が作成され、ソルバはサンプル期間ごとに **更新関数**を1回呼び出すようになります。
- 2 **宣言文**で、制御器出力を表すグローバル変数を定義し、離散状態を前の積分値 i_{k-1} を表す変数にマッピングします。

```
double yk;
#define ik_1 DiscState(0)
```

初期化関数で i_{k-1} を0に初期化します。 y_k は**更新関数**と**出力関数**の両方でアクセスされるため、グローバル変数である必要があることに注意してください。

- 3 **更新関数**で、中間結果を格納するために使用する変数double ikを定義します。次に、式(2)および(3)で定義した制御則を実装します。ikの計算後、 $ik_1 = ik$;を追加することを忘れないでください。
- 4 **出力関数**で、制御法則の計算結果を出力OutputSignal(0,0) = ykに割り当てます。出力は **出力関数**でのみ書き込むことができることに注意してください。

シミュレーションを実行すると、出力電圧は連続PI制御器を使用したモデルと同様になるはずです。

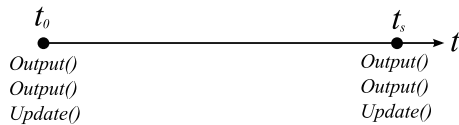


この段階では、モデルは参照モデルcscript_controller_1.plecsと同じになっているはずです。



注意: 図1に示すように、**更新関数**は**出力関数**の後に呼び出されるため、デジタルPI制御器の出力は1サイクル遅延します。したがって、**出力関数**は、前の時間ステップで計算された結果を出力します。このシミュレーションの関数呼び出しの正確なシーケンスを図5に示します。時間ステップごとの**出力関数**の呼び出し回数は、ソルバによって内部的に決定されます。この特定のモデルの場合、**出力関数**は各主要な時間ステップで2回呼び出されます。ただし他のモデルでは、**出力関数**はより頻繁に呼び出される場合があります。

図5: cscript_controller_1.plecsの関数呼び出しのタイミング



1サイクルの遅延を排除



あなたのタスク: サンプル周波数が低すぎる場合、1サイクルの遅延によって不安定になる可能性があります。この効果を観察するには、サンプル周波数を10e3Hzに変更してシミュレーションを再実行します。遅延をなくすには、制御結果が計算と同じ時間ステップで出力されるようにします。

- 1 **宣言**にグローバル変数`double ik`を作成します。
- 2 **更新関数**から、離散状態を更新する行`ik_1 = ik;`を除いて制御コードを削除します。
- 3 制御コードを**出力関数**にシフトします:

```
ik = ik_1 + Ts*ek;
OutputSignal(0,0) = kp*ek + ki*ik;
```

つまり、積分動作は**出力関数**で計算されますが、離散状態によって記録された累計は**更新関数**まで更新されません。



この段階では、モデルは参照モデルcscript_controller_2.plecsと同じになっているはずです。

5.3 連続制御則の実装

C言語入力ブロックの主な機能は複雑な関数と離散制御器を実装することですが、これにより $\dot{x} = f(x)$ という常微分方程式を解くために、連続状態と微分方程式を定義することもできます。シミュレーション エンジン は数値積分を使用して微分方程式を解きます。式(1)は常微分方程式で記述でき、積分は次のようになります:

$$\frac{di}{dt} = e(t) \quad (4)$$

積分動作は、連続状態 $i(t)$ と微分方程式の式(4)を定義することによってC言語入力ブロックでモデル化できます。各タイムステップで、ソルバは $i(t)$ を数値的に計算します。

定義する連続状態ごとに、ContState(i)とContDeriv(i)というマクロが作成されます。これらのマクロは、シミュレーションソルバが微分方程式を解くことを可能にするフックです。必要なのは、**導関数**で方程式を記述することだけです。



あなたのタスク:

1 モデルcscript_controller_2.plecsのコピーを作成し、C言語設定を再構成します。**サンプリング時間**設定を0に設定し、**パラメータ**1/fsを削除します。**離散状態数**を0に、**連続状態数**を1に設定します。連続状態は式(1)の積分項を表すのに使用されます。

2 **宣言文**で、InputSignal(0,0) マッピングを e に変更し、次のように連続状態と導関数を変数名にマッピングします:

```
#define e InputSignal(0,0)
#define I ContState(0)
#define I_deriv ContDeriv(0)
```

3 **導関数**では、積分器を記述する微分方程式を入力します。これは $I = \int e(t)dt$ または $dI/dt = e(t)$ なので、I_deriv = e;と入力します。次に、ソルバは微分方程式を解いて積分値 I を求めます。

4 積分値 $I = 0$ の適切な初期値は、**初期化関数**で設定します。

5 **更新関数**からすべてのコードを削除し、**出力関数**では、次のコードを除くすべてのコードを削除します:

```
OutputSignal(0,0) = kp*e+ki*I;
```

シミュレーションを実行すると、元の連続PI制御器と同じ出力電圧が表示されます。



この段階では、モデルは参照モデルcscript_controller_4.plecsと同じになっているはずです。



注意: C言語入力ブロック内で連続状態を操作すると、微分方程式または状態空間システムに高度な機能を追加できます。C言語入力ブロックではなく、リセット可能な積分器を実装する場合は、外部リセット付きの積分器ブロックを使用します。代数ループの警告を回避するには、フィードバックに積分器ブロックの状態ポートを使用します。

6 上級演習: 計算遅延を備えたデジタルPI制御器の実装

セクション5.2では、計算遅延のないPI制御器を実装しました。実際のシステムでは、コントローラが入力を読み取り、制御計算を実行し、出力に書き込むのに必要な時間のため、有限の遅延が存在します。この遅延により、ある程度システムの安定性が低下する可能性があります。この計算遅延をシミュレートするために、制御結果 y_k がOutputSignal(0,0)に書き込まれる前に遅延時間が導入されます。



あなたのタスク:

- 1 モデル `cscript_controller_2.plecs` のコピーを保存し、voltage controller のマスクに **Calculation delay** というラベルの付いたパラメータを追加します。これを t_d という名前の変数に割り当て、その値を 0.1 に設定します。これは計算遅延時間を $0.1T_s$ に設定するために使用します。

- 2 C 言語入力ブロック設定で、**基本設定** タブの **パラメータ** リストに引数 t_d/f_s を追加し、**宣言文** で変数 t_d を定義します:

```
static double Td;
```

そして、**初期化関数** に値 `Td = ParamRealData(3,0);` を割り当てます。

- 3 計算遅延を実装するには、まずハイブリッド離散変数のサンプリング時間設定を実装する必要があります。固定ステップ設定では、各期間の開始時にサンプルヒットが発生し、可変時間ステップでは計算遅延後にヒットが発生します。ハイブリッド時間設定はマトリックス形式で入力する必要があります。行の最初のエントリはサンプル時間、2 番目のエントリはオフセット時間です。次の **サンプリング時間** を入力します: `[1/fs, 0;-2, 0]`

- 4 最初のヒット時間が固定時間ステップ設定によって生成されるようにするには、**初期化関数** で、可変ステップヒット時間を定義する `NextSampleHit` マクロを大きな数値に初期化する必要があります: `NextSampleHit = NEVER;`

- 5 **宣言文** では、`NEVER` を非常に大きな数値として定義する必要があることに注意してください。ファイル `<float.h>` をインクルードすると、機械で表現可能な最大の浮動小数点数である `DBL_MAX` として `NEVER` を定義できます。

- 6 スイッチングサイクルの開始時に、 i_k と y_k の制御計算を実行する必要があります。計算された制御アクション y_k は、次の **出力関数** の呼び出し (`CurrentTime + Td` の時刻に発生) まで出力されません。**更新関数** に次の行を追加します:

```
if (NextSampleHit == NEVER) //beginning of switching cycle
{
    //Control calculations for ik, ik_1, yk here
    NextSampleHit = CurrentTime + Td;
}
else
    NextSampleHit = NEVER;
```

- 7 **出力関数** では、スイッチングサイクルの開始時に計算された制御アクションを出力するために、 y_k を出力ポートに割り当てます。



この段階では、モデルは参照モデル `cscript_controller_3.plecs` と同じになっているはずです。計算遅延の影響を観察するには、 f_s を 10e3Hz に設定し、計算遅延を 0.1 および 0.9 にしてシミュレーションを実行します。この実装では $t_d \in]0, 1[$ の値のみが許可されることに注意してください。特殊なケース 0 と 1 の処理は、追加の演習としてユーザーに任されています。

7 最後に

この演習では、PLECS C言語入力ブロックを使用して、いくつかの調整を備えたカスタムのデジタルPI制御器を実装する方法を学習しました。これには、ブロック内で事前定義され、シミュレーションエンジンとのインターフェースに使用する関数呼び出しを理解することが含まれます。C言語入力ブロックのもう1つの重要な側面は、ブロック内で構成できるさまざまな時間設定を理解することです。これは、タイミング関数の動的ステップサイズの呼び出しや固定ステップ制御器の再現など、特定の期待した動作を保証するために重要です。PLECS C言語入力 ブロックは非常に汎用性が高く、複雑な制御器やコンポーネントをモデル化するために使用できます。

改訂履歴:

Tutorial Version 1.0 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

@ plecs_adv@adv-auto.co.jp

Email

<https://adv-auto.co.jp/>

Web

PLECS Tutorial

© 2002–2022 by Plexim GmbH

このマニュアルで記載されているソフトウェアPLECSは、ライセンス契約に基づいて提供されています。ソフトウェアは、ライセンス契約の条件の下でのみ使用またはコピーできます。Plexim GmbHの事前の書面による同意なしに、このマニュアルのいかなる部分も、いかなる形式でもコピーまたは複製することはできません。

PLECSはPlexim GmbHの登録商標です。MATLAB、Simulink、およびSimulink Coderは、The MathWorks, Inc.の登録商標です。その他の製品名またはブランド名は、それぞれの所有者の商標または登録商標です。