CIRCUIT DESIGN / MACOS APPLE SILICON

# Netlist Studio

## 利用ガイド

共通ガイドの基準版 v0.42.0 / revision 3
分圧、RC、Buck を題材に、回路作成・解析・制御を順に学びます。

### 目次


01 利用ガイド 2

02 はじめに 4

03 回路エディタ 8

04 SPICE解析 12

05 Graph解析 19

06 制御回路 23

07 FirmwareとFeedback 31

08 機能一覧 36

09 技術モジュール 38

10 トラブルシューティング 41


公開 Web ガイドをもとにしたオフライン閲覧版です。

CHAPTER 01

# 利用ガイド

分圧、RC、Buckを題材に、導入から回路作成、解析、制御までの学習順を確認します。

## このガイドでできること

Netlist Studio `v0.42.0`をmacOS Apple Siliconへ導入し、回路の作成、接続確認、保存、SPICE入力の書き出し、ngspice解析、Graph解析、制御Projectまでを段階的に学ぶガイドです。電子回路や制御を初めて学ぶ場合も、各章の「完了チェック」を順に確認すれば、同じ回路を自分で作り直せます。

このガイドでは、次の3つの題材を使います。

| 題材 | 学ぶこと | 完成の目安 |
| --- | --- | --- |
| 10 Vの電圧分割回路 | 部品配置、配線、保存、OP解析、SPICE入力 | 中点電圧が約5 V |
| 1 kΩ／100 nFのRCローパス | AC／TRAN解析、2点測定、時定数、遮断周波数 | 約1.59 kHzで約-3 dB |
| 50 kHzのBuck制御 | 測定点、PWM、PID、実機I/O、Feedback | 指令、適用Duty、出力、電流を同じ時間軸で確認 |

> ⚠ 回路・制御の編集内容は`.netgrid.json`へ保存してください。SPICE入力、Firmware Bundle、WASM、解析結果は、保存したProjectから再生成する成果物として扱います。

Control、AI、独自ブロック、Firmware authoring、Feedbackは`experimental`です。重要な設計や実機制御へ使う前に、生成内容、定格、保護、I/O割当、結果を利用者自身で検証してください。

## おすすめの学習順

1. はじめにで導入、初回起動、保存と再読込を確認します。
2. 回路エディタで電圧分割回路を作り、接続とSPICE入力を確認します。
3. SPICE解析でRCローパスを作り、OP、AC、TRANを実行します。
4. Graph解析で波形、2点測定、区間統計、FFT、演算波形を読みます。
5. 制御回路でPWMからBuckの閉ループ制御へ進みます。
6. FirmwareとFeedbackで生成C++、実機I/O、8ファイルのBundle、結果の有効範囲を確認します。
7. 必要に応じて機能一覧、技術モジュール、トラブルシューティングを参照します。

## 始める前に

- 正式配布対象はmacOS Apple Siliconです。
- 回路解析にはngspice 46を使用します。
- `.netgrid.json`を保存する作業用フォルダーと、書き出し成果物用の別フォルダーを用意します。
- 実機へ接続せず、最初はシミュレーションだけで全章を進めます。
- 画面の文言や機能が一致しない場合は、アプリとこのガイドの製品versionが一致しているか確認します。

## 学習完了チェック

- Projectを保存し、再度開いて同じ回路と制御グラフを復元できる。
- 回路の基準GND、測定点、電源、接続ネットを説明できる。
- OP、AC、TRANを目的に応じて選べる。
- RCの遮断周波数と時定数を、計算値とGraphの両方で確認できる。
- PWMのDuty指令と実際に適用されたDutyを区別できる。
- Project正本、Firmware Bundle、Feedback結果の違いを説明できる。
- シミュレーション結果だけで実機の安全性を判断しない。

CHAPTER 02

# はじめに

インストール、初回起動、画面構成、プロジェクト保存を完了します。

## インストール前の確認

| 確認項目 | 必要な状態 |
|---|---|
| Mac | Apple Silicon搭載のmacOS |
| Netlist Studio | 正式配布版`v0.42.0` |
| 回路解析 | ngspice 46 |
| Project保存先 | 読み書きできる作業用フォルダー |
| 実機 | このガイドのシミュレーション完了までは接続しない |

配布サイトでは、version、対応環境、注意事項、成果物のhashを確認してからDMGを取得します。別のサイトや再配布ファイルを使う場合は、このガイドの前提と一致しない可能性があります。

## インストールと初回起動

1. 配布元からNetlist Studio `v0.42.0`のDMGを取得します。
2. DMGを開き、Netlist Studioをアプリケーションへドラッグします。
3. DMGを取り出します。
4. アプリケーションからNetlist Studioを起動します。
5. macOSの初回確認が表示された場合は、アプリ名と開発元を確認して続行します。

上部に回路と制御の切替が表示され、回路パレット、盤面、インスペクターを操作できれば準備完了です。解析にngspiceを使う場合は、ngspice 46を別途インストールしてください。

## 画面の構成




図2-1　回路ワークスペースの3領域

1. **部品パレット**: 配線や部品を選びます。選択中のツール名もここで確認します。
2. **盤面**: 部品を配置し、配線と接続関係を作ります。選択したセルは枠で示されます。
3. **インスペクター**: 選択した部品の値、RefID、netを確認・編集します。

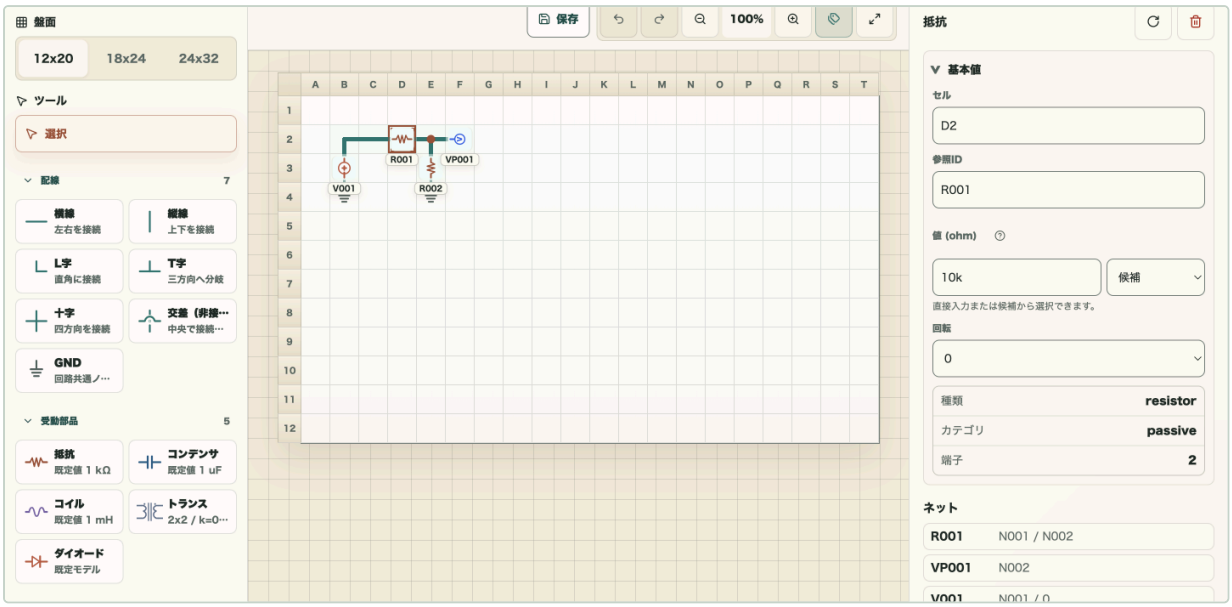



図2-2　パレット・盤面・インスペクターの位置関係

| 領域 | 主な役割 |
|---|---|
| 上部バー | 回路／制御切替、文書状態、主要操作 |
| 左ペイン | 部品または制御ブロックのパレット |
| 中央 | 回路盤面または制御キャンバス |
| 右ペイン | 選択対象、警告、SPICE入力、解析条件、結果 |
| 下部ステータス | 座標、選択、実行状態、診断 |

左右ペインは折り畳めます。狭い画面では必要なペインだけを開き、上部タブは横スクロールして移動します。キーボード操作では、現在のfocus、選択状態、エラー説明を確認してから次へ進みます。色だけでなく、label、icon、診断文も状態判断に使います。

## メニューとショートカット

macOSでは、上部に`Netlist Studio`、ファイル、編集、表示、ウインドウ、ヘルプの6メニューを表示します。

| メニュー | 主な用途 |
|---|---|
| Netlist Studio | macOS標準のアプリ情報、サービス、表示／非表示、終了 |
| ファイル | 新しい盤面、開く、保存、名前を付けて保存、SPICE入力や独自ブロックの入出力 |
| 編集 | Undo／Redo、Cut／Copy／Paste、複製、削除、全選択、移動／回転／中央配置 |
| 表示 | 回路／制御の切替、左右ペイン、参照ID、現在のワークスペースのzoom／全体表示、fullscreen |
| ウインドウ | 最小化、前面表示、Main／Graph解析ウインドウの切替 |
| ヘルプ | 利用ガイド、対応環境でのアップデート確認 |

メニューの有効状態とチェック状態は、現在のワークスペース、選択、履歴、入力フォーカスに合わせて変わります。デスクトップ版のCmd／Ctrl付きショートカットはnative menuが一度だけ処理します。

## 最初のProjectを保存する

1. ファイル ＞ 新しい盤面で空の盤面を用意します。
2. Projectの目的が分かる名前を付けます。最初は電圧分割の練習で構いません。
3. 未確定の入力欄がある場合は、Enterまたはfocus移動で確定します。
4. 保存を実行し、作業用フォルダーへvoltage-divider.netgrid.jsonとして保存します。
5. 文書状態が保存済みになることを確認します。

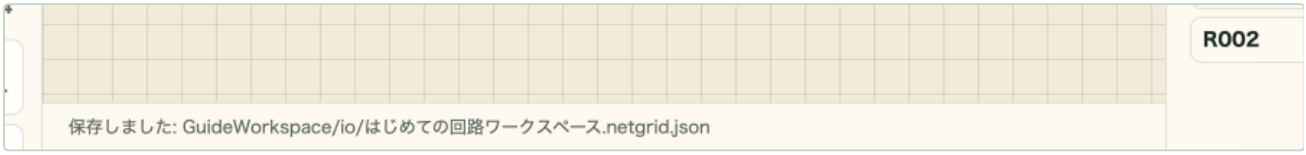



図2-3　保存完了を示すステータス

保存しましたという完了文とProject名が表示されれば、その保存操作は完了です。画面例のGuideWorkspaceは撮影用に匿名化した保存先であり、実際には自分で選んだ作業用フォルダーを確認します。

.netgrid.jsonが回路と制御Projectの正本です。SPICE入力、PDF、画像、Firmware Bundle、WASM、解析結果の保存先とは分けてください。

名前を付けて保存は毎回保存先を選び直します。成功した場合だけ、その保存先が次回の通常保存先になります。キャンセルまたは失敗した場合は、元の保存先、文書内容、未保存状態を維持します。

## 保存したProjectを開いて確認する

1. Projectを保存したあと、内容が空でも一度アプリを閉じます。
2. Netlist Studioを再起動し、ファイル ＞ 開くから保存した.netgrid.jsonを選びます。
3. タイトル、盤面サイズ、回路／制御の状態が保存前と一致することを確認します。
4. 古い保存世代を開いたという案内がある場合は、元ファイルを残したまま名前を付けて保存…で新しいcopyを作ります。

Undo／Redoは回路と制御の操作単位で動作します。読込後に意図しない変更がある場合は、そのまま上書きせず、元ファイルと現在の文書状態を確認してください。

## ngspiceを準備する

1. ngspice 46をインストールします。
2. Netlist Studioの設定で自動検出結果を確認します。
3. 検出されない場合だけ、ngspiceの実行ファイルを選択します。
4. SPICE解析の電圧分割OP解析で、実行環境を確認します。

実行ファイルを変更した直後は、古い解析結果を成功証拠として使わず、新しい設定で1回実行します。

## アプリを更新する

正式配布版は起動後に一度だけ更新を確認します。任意の時点で確認する場合は、編集内容を保存してからヘルプ ＞ アップデートを確認…を選びます。新しいversionのbannerからdownloadし、準備完了後に再起動して更新を選びます。

`v0.42.0`では、更新問題の観測またはRelease Ownerの明示要求がなかったため、前版からの実runtime更新journeyは必須gateではなく、未実施です。これは更新成功を実測したという意味ではありません。更新が完了しない場合は操作を繰り返さず、Projectを保存し、配布サイトの注意事項とDMGを確認してください。

## 完了チェック

- 正式版のversionと対応環境を確認した。
- 空のProjectを`.netgrid.json`で保存し、再度開けた。
- 編集正本と生成成果物の保存先を分けた。
- 回路解析に必要なngspice 46を準備した。
- 更新失敗時にProjectを保存して手動DMGへ戻る手順を把握した。

CHAPTER 03

# 回路エディタ

10 Vの分圧回路で部品配置、配線、OP、接続診断を身につけます。

## 部品を置く

1. 上部で回路を選びます。
2. 左パレットでカテゴリを開き、部品を選びます。
3. 中央盤面の空きセルを選びます。
4. 右ペインで値とRefIDを確認します。

盤面に部品が表示され、右ペインに設定が表示されれば完了です。重なり警告が出た場合は、対象を空き位置へ移動します。

## 部品を移動・回転・複製する

移動する部品を選び、移動操作を開始します。プレビューが有効な空き位置へ移動して配置を確定します。端子の向きが合わない場合は、回転してから配線します。複数部品は矩形選択または修飾キーで選択できます。`Cmd+C`、`Cmd+V`、`Cmd+D`はアプリ内の選択内容に対して動作します。

## 配線する

1. パレットから配線を選びます。
2. 接続元の端子に接するセルを選びます。
3. 接続先まで必要な配線セルを配置します。
4. 接続ネットの強調表示を確認します。

T字・十字の接続点はノードとして表示されます。`wire-bridge`は交差しても接続しないため、意図したネットと強調表示が一致することを確認してください。見た目が接していても、端子の向きと配線セルが一致しなければ電気的には接続されません。

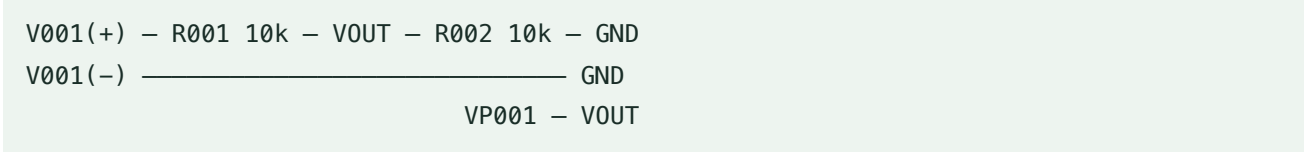
## 値を編集する

対象部品を選び、右ペインの値入力欄へ単位を含む値を入力します。入力を確定し、SPICE入力プレビューを確認します。入力が不正な場合は元の値が維持され、診断が表示されます。

`10k`、`100n`、`470u`のようなSPICE表記を使えます。`M`と`Meg`の解釈を取り違えないよう、確定後の表示とSPICE入力を確認してください。

## 練習1: 10 Vの電圧分割回路を作る

次の回路を作ります。




Vプローブは GND基準の1端子プローブです。`VP001`の端子を`VOUT`へ接続します。




図3-1　完成した10 V電圧分割回路

1. **電圧分割回路**: 電源、2本の抵抗、Vプローブ、GNDが小さな閉じた経路を作っています。
2. **配線と測定点**: 中点へVP001を接続し、上下の抵抗値が同じであることを確認します。
3. **部品設定**: 選択したR001のRefID、値、端子netを右側で照合します。

## 1. 部品を配置する

| 部品 | RefID | 値 | 役割 |
|---|---|---|---|
| DC電源 | V001 | 10 V | 入力電圧 |
| 抵抗 | R001 | 10k Ω | 上側抵抗 |
| 抵抗 | R002 | 10k Ω | 下側抵抗 |
| GND | - | 0 V基準 | 回路の基準node |
| Vプローブ | VP001 | - | 中点電圧の測定 |

盤面上では、電源の+からR001、中点VOUT、R002、GNDへ電流経路が続くように配置します。電源の−も同じGNDへ戻します。部品のRefIDが重複していないことを確認してください。




図3-2　抵抗R001の設定と接続情報

値を入力したら、表示が`10k`として確定し、端子netが意図した2つのnodeへ割り当てられていることを確認します。

## 2. 3つのnetを配線する

1. `VIN`相当のnetとして、`V001(+)`と`R001`の片側をつなぎます。
2. `VOUT`相当のnetとして、`R001`の反対側、`R002`の片側、`VP001`をつなぎます。
3. GND netとして、`R002`の反対側、`V001(−)`、GNDをつなぎます。
4. 各netを選び、強調範囲が別のnetへ漏れていないことを確認します。

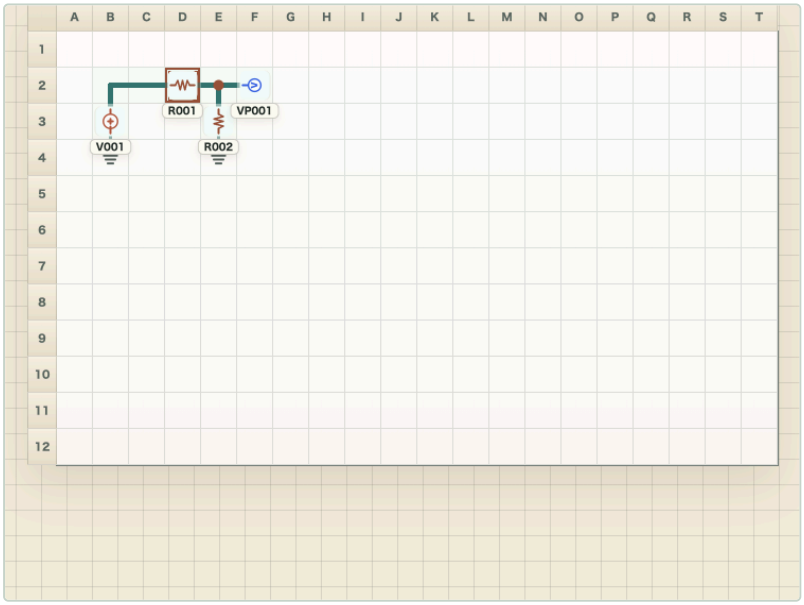



図3-3　配線した3つのnetと測定点

自動生成されるSPICE node名は配置によって変わります。見た目の名前ではなく、どの端子が同じnetへ属するかで確認します。

## 3. 計算値を先に求める

無負荷の分圧電圧は次の式です。

```
VOUT = VIN × R002 / (R001 + R002)
     = 10 × 10k / (10k + 10k)
     = 5 V
```

この5 Vを、後のOP解析で比較する期待値にします。

### 4. 警告とSPICE入力を確認する

1. 右ペインの警告を開きます。
2. 未接続端子、RefID重複、部品重なり、値エラーがないことを確認します。
3. `Netlist`でngspice、解析種別`OP`を選びます。
4. プレビューに`V001`、`R001`、`R002`、`.op`、測定対象が含まれることを確認します。
5. 自動node名そのものではなく、各素子の両端が意図したnetへ割り当てられているかを確認します。

### 5. OP解析で5 Vを確認する

1. シミュレーションで`OP`を選びます。
2. ngspice解析を実行します。
3. `VP001`の結果が約5 Vであることを確認します。
4. 値が一致したらProjectを保存します。

実行の詳細はSPICE解析を参照してください。5 Vから大きく外れる場合は、まず抵抗値とGND、次に`VP001`が接続されたnetを確認します。

## 警告と接続を確認する

1. 右ペインの警告を開きます。
2. 重大度の高い項目から選びます。
3. 未接続端子、短絡、重なり、値エラーを修正します。
4. 警告一覧を再確認します。

警告がなくなっても、回路の電気的妥当性が保証されるわけではありません。電源、GND、モデル、定格、解析条件も確認してください。

## 完了チェック

- `V001=10 V`、`R001=R002=10 kΩ`の分圧回路を自分で配置した。
- 3つのnetとGND基準を説明できる。
- 接続強調とSPICE入力の両方で配線を確認した。
- OP解析で`VP001`が約5 Vになった。
- 結果確認後のProjectを`.netgrid.json`へ保存した。

CHAPTER 04

# SPICE解析

1 kΩ／100 nFのRC回路でOP、AC、TRANを実行します。

## 解析を選ぶ

| 解析 | ディレクティブ | 何を固定／変化させるか | 主な用途 |
|---|---|---|---|
| OP | `.op` | 1つの直流動作点 | 分圧、bias、node電圧 |
| DCスイープ | `.dc` | DC電源を範囲内で変化 | 入出力特性、動作範囲 |
| TRAN | `.tran` | 時間を進める | RC立上り、PWM、過渡応答 |
| AC | `.ac` | 小信号周波数を掃引 | filterの振幅と位相 |

OPで時間波形は得られません。ACは設定した小信号振幅を使い、TRANは電源部品の時間波形を使います。目的と異なる解析を選ぶと、正常な回路でも期待したGraphになりません。

## 電圧分割回路でOPを確認する

1. 回路エディタで保存した分圧Projectを開きます。
2. 解析で`OP`を選びます。
3. SPICE入力プレビューに`.op`が1行だけ含まれることを確認します。
4. ngspice解析を実行します。
5. `VP001`が約5 V、各値がfiniteであることを確認します。

計算値と一致しない場合は、解析を変える前に回路のGND、抵抗値、probe netを修正します。

## 練習2: RCローパス回路を作る

分圧Projectを名前を付けて保存…で`rc-low-pass.netgrid.json`へ複製してから、次の回路へ作り替えます。




| 部品 | 値／設定 | 役割 |
|---|---|---|
| AC電源`V001` | AC小信号振幅1、位相0° | AC解析の入力 |
| 抵抗`R001` | 1k Ω | series抵抗 |
| コンデンサ`C001` | 100n F | 高周波をGNDへ流す |
| Vプローブ`VP_IN` | 入力net | 入力の基準系列 |
| Vプローブ`VP_OUT` | 出力net | filter出力 |
| GND | 0 V | 共通基準 |

VプローブのRefIDは実際の自動採番と異なる場合があります。学習中はlabelを入力電圧、出力電圧にするとGraphで識別しやすくなります。

この回路の時定数と遮断周波数は次のとおりです。

```
τ = R × C = 1k × 100n = 100 µs
fc = 1 / (2πRC) ≈ 1.59 kHz
```




図4-1　RCローパス回路と解析設定の全体

1. **RC回路**: 入力、1 kΩ、100 nF、入力・出力プローブ、GNDを接続します。
2. **解析種別**: ACまたはTRANを目的に合わせて選びます。
3. **解析条件**: sweep範囲やstep／stop timeを確定してから実行します。

## RCローパスのAC解析

1. 解析種別を`AC`にします。
2. sweepを`dec`、1 decadeあたりの点数を`100`にします。
3. 開始周波数を`10` Hz、終了周波数を`100k` Hzにします。
4. AC電源の小信号振幅が`1`、位相が`0°`であることを再確認します。
5. プレビューの`.ac dec 100 10 100k`相当と、`VP_IN`／`VP_OUT`の保存対象を確認します。
6. ngspice解析を実行し、振幅`dB`と位相を表示します。

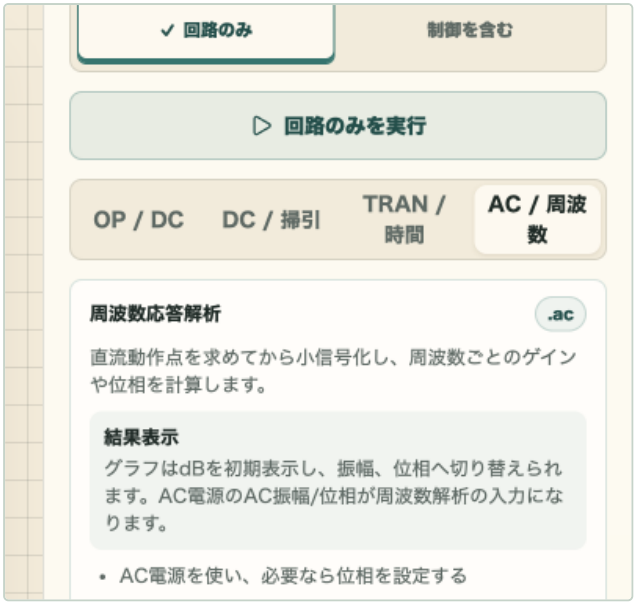



図4-2　AC解析の設定

同じ部品値と理想モデルなら、次を目安にします。

- 低周波のVP_OUT / VP_INは約0 dB。
- 約1.59 kHzで約-3.01 dB、位相は約-45°。
- 遮断周波数より十分高い領域では、おおむね-20 dB/decadeで減衰。
- 位相は0°付近から-90°へ近づく。

値が合わない場合は、AC電源の通常波形値ではなくAC小信号振幅、コンデンサの単位、probe位置を確認します。

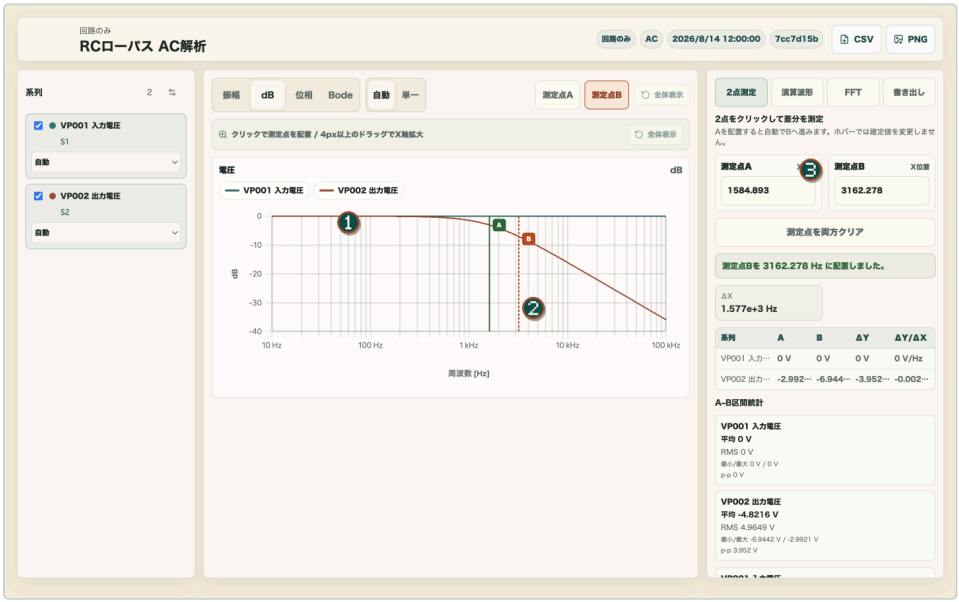



図4-3　RCローパスのAC解析結果

1. **系列**: 入力電圧と出力電圧の両方が表示対象になっていることを確認します。
2. **減衰曲線**: 低周波では約0 dB、遮断周波数以降は低下する形を読みます。
3. **測定パネル**: A/Bの実周波数と差分を数値で確認します。




図4-4　1.59 kHz付近の測定値

画面例ではAが`1584.893 Hz`へsnapされています。理論値`1.59 kHz`との差はsweepの離散点によるもので、表示された実X値を結果記録へ残します。

## RCローパスのTRAN解析

AC解析用Projectをもう一度名前を付けて保存…で複製し、AC電源を矩形波電源へ置き換えます。

| 矩形波設定 | 値 |
|---|---|
| low／high | `0 V／1 V` |
| delay | `100us` |
| rise／fall | `1us／1us` |
| width | `800us` |
| period | `2ms` |

TRANはstep timeを`2us`、stop timeを`1.2ms`にします。`use initial conditions`は最初は無効のままにします。




図4-5　TRAN解析の設定

立上りが100 µsから始まるため、理想RCの出力はおおよそ次の値になります。

| 時刻 | step後の経過 | 出力の目安 |
|---|---|---|
| 200 µs | 1 τ | 約0.632 V |
| 330 µs | 約2.3 τ | 約0.90 V |
| 600 µs | 5 τ | 約0.993 V |

矩形波の有限rise timeと数値刻みがあるため、完全一致ではなく傾向と許容差で確認します。




図4-6　RCローパスのTRAN解析結果

1. **入力系列**: 0 Vから1 Vへ変化する時刻を基準にします。
2. **RC出力**: 指数的に立ち上がり、入力が下がると指数的に減衰します。
3. **2点測定**: A/Bの実時刻、Δt、1/|Δt|を同じパネルで確認します。




図4-7　RC時定数付近の2点測定

画面例のデータ点へsnapしたΔtは`9.671e−5 s`です。理論値100 µsと比較し、B側の出力が約0.62 Vであることも同時に確認します。

## ngspice解析を実行する

1. 右ペインのシミュレーションを開きます。
2. 解析種別と条件を確定します。
3. 推定解析点数と警告を確認します。
4. ngspice実行を開始し、完了表示を待ちます。
5. 波形を開きます。

結果がGraphに表示されれば完了です。大量データの詳細解析は別windowへ遅延読込されます。解析点数が多いという警告がある場合は、必要な時間／周波数範囲とstepを見直します。上限を回避するためだけに意味のない粗いstepへ変更しないでください。

ngspiceが見つからない場合は、設定から実行ファイルを選択します。実行に失敗した場合は、SPICE入力プレビューと最初に表示されたngspice診断を確認してください。

## SPICE入力を書き出す

1. 右ペインの`Netlist`を開きます。
2. `ngspice`または`ltspice`を選びます。
3. OP、DCスイープ、TRAN、ACのいずれかを選びます。
4. 解析条件とプレビューを確認します。
5. `SPICE入力一式を書き出す`を実行し、Projectとは別の保存先を選びます。

出力フォルダーには対象別の`.cir`、LTspice用の補助ファイル、`manifest.json`が含まれます。LTspiceの`.raw`と`.log`はシミュレーター実行時に生成されます。書き出し後にProjectを変更した場合は、古い`.cir`を現在のProjectの成果物として扱わず、再生成します。

## 完了チェック

- OP、AC、TRANの違いを説明できる。
- 分圧回路で約5 VのOPを確認した。

- 1 kΩ／100 nFのRC回路を作り、τ=100 μsとfc≈1.59 kHzを計算した。
- ACで約-3 dB／-45°の点を確認した。
- TRANで1τ、約2.3τ、5τの立上りを確認した。
- ProjectとSPICE書き出し成果物を別に管理した。

CHAPTER 05

# Graph解析

RC波形から遮断周波数、時定数、2点統計、FFT、演算波形を読み取ります。

## 最初に軸と系列を確認する

Graphを開いたら、測定を始める前に次を確認します。

- X軸がOP、sweep値、時間、周波数のどれか。
- Y軸がV、A、%、dB、degreeなど、どの単位か。
- 系列labelとprobe RefIDが、現在のProjectと一致しているか。
- linear表示とstep表示のどちらが信号の意味に合うか。
- 解析条件またはProject変更後の古い結果ではないか。

表示が滑らかでも、解析点数や単位が間違っていれば正しい結果ではありません。




図5-1　Graph解析windowの全体

1. **系列一覧**: 元系列と追加した出力電圧 × 2を区別します。
2. **Graph**: 現在選んだ解析の軸、単位、凡例を確認します。
3. **解析タブ**: 2点測定、演算波形、FFT、書き出しを切り替えます。

## RCのAC結果を読む

1. SPICE解析で作ったRCローパスのAC結果を開きます。
2. `VP_OUT`の振幅をdB表示にします。
3. 低周波で約0 dBであることを確認します。
4. 約1.59 kHz付近で約-3.01 dBの位置を探します。
5. 位相表示に切り替え、同じ周波数付近で約-45°であることを確認します。
6. 10 kHzと100 kHzの差を見て、約-20 dB/decadeの傾向を確認します。

入力と出力を同時に表示できる場合は、`VP_IN`を基準にします。入力振幅が1 Vでない場合、`VP_OUT`単独のdB値と伝達比のdB値を混同しないでください。

## Graphで2点間を測定する

2点測定では、確定したA／Bを全plotへ固定表示し、2点間の差と区間統計を確認できます。

1. Graph解析windowで2点測定tabを開きます。
2. グラフ上の測りたい位置をクリックして測定点Aを置きます。
3. もう一つの位置をクリックして測定点Bを置きます。
4. Aの緑実線、Bの橙破線と文字labelが全plotへ表示されたことを確認します。
5. ΔX、時間軸なら1/|Δt|、各系列のΔYとΔY/ΔX、区間のmin／max／平均／RMS／p-pを確認します。

4px以上dragすると従来どおりzoomになり、測定点は移動しません。キーボードでは測定点A／BのX位置を入力できます。有効な値は最寄りのXデータへsnapします。




図5-2　2点測定の入力と結果

番号ではなくA／Bの文字、緑実線／橙破線、数値表を組み合わせて点を識別します。入力欄の値は確定後に最寄りのデータ点へ変わることがあります。

## RCのTRAN結果で時定数を測る

1. 矩形波入力とRC出力を同じ時間軸へ表示します。
2. Aを入力立上りの100 µs付近へ置きます。
3. Bを200 µs付近へ置き、Δt≈100 µsを確認します。
4. Bで出力が最終値のおよそ63.2%になっているか確認します。
5. Bを330 µs付近へ移し、およそ90%になっているか確認します。
6. A–B区間のmin／max／p-pが、選択した系列と時間範囲に対応しているか確認します。

stepの開始点は有限rise timeとdata snapの影響を受けます。期待値との差を見るときは、Graphに実際に表示されたA／BのX値を使います。

## 演算波形を追加する

1. Graph解析windowで演算波形tabを開きます。
2. 差、和、比、倍率、絶対値、dB比から演算を選びます。
3. 系列と、必要な場合は倍率を指定します。
4. previewで表示名、式、出力単位、計算点数、エラーを確認します。

5. エラーがなければ追加します。

RCの伝達比は、入力と出力が同じX点を持つAC結果でdB比を使うと確認できます。和・差・dB比には同じ単位の系列を使用します。電圧を電流で割る比はΩ、電流を電圧で割る比はSとして表示します。任意式はtokenまたは関数buttonから入力し、previewのエラーを解消してから追加します。

演算波形は元データから作る表示上の系列です。Projectやngspiceの生結果を書き換えません。元系列、式、単位を記録してから設計判断へ使います。

## FFTを使う

FFTはTRANの等間隔な時間系列で、周期性や高調波を確認する場合に使います。

1. 起動過渡を除いた定常区間へA／Bを置きます。
2. FFT範囲で測定点A–Bを選びます。
3. A／Bが異なるXへsnapし、十分な点数と複数周期を含むことを確認します。
4. window関数、周波数分解能、sample間隔を確認してから実行します。

A／Bが不足している場合や同じXへsnapした場合はFFTを開始しません。矩形波には基本波以外の奇数次高調波が含まれるため、RC出力側で高周波成分が減る傾向を比較できます。短すぎる区間のpeakを回路固有の周波数と断定しないでください。




図5-3　演算波形を保持したFFT操作

左の系列一覧に演算波形が残り、右のFFT計算完了にsample数と周波数分解能が表示されます。平坦または不自然な結果では、回路を変える前にFFT範囲と元系列を見直します。

## 結果を再現できる状態にする

解析結果を共有するときは、画像だけでなく次を一緒に記録します。

- Projectの製品versionと保存ファイル。
- 解析種別、step／stopまたはsweep条件。
- 表示した系列、単位、linear／step、dB／phase設定。
- A／Bの実X値、演算式、FFT範囲。
- ngspice versionと最初のwarning／error。

## 完了チェック

- 軸、系列、単位、解析種別を確認してから測定した。
- RCの約1.59 kHz、約-3 dB、約-45°をGraphで確認した。
- TRANのA／Bから約100 µsの時定数を確認した。
- 演算波形と元のsimulation系列を区別できる。
- FFTの範囲と分解能が結果へ影響することを説明できる。

CHAPTER 06

# 制御回路

Buckを一から構成し、PWM、PID、閉ループ、dead timeを段階的に確認します。

## この章の境界

この章のControl、独自ブロック、Firmware、Feedbackは`experimental`です。まずシミュレーションだけで操作し、生成されたC++、I/O割当、安全値、回路モデル、解析条件を確認します。

> ⚠ この章の50 V Buckは学習用シミュレーション条件です。この記載だけで実機を製作、通電、接続しないでください。感電、発火、蓄積エネルギー、半導体破壊を防ぐ定格・絶縁・保護設計は別途必要です。

制御Projectでは、次の流れを作ります。

```
回路の測定点 → CircuitProbeInput → PID → PWM → CircuitActuatorOutput → 回路の制御source
                                          └──────────────→ Scope
```

回路のみでは制御電圧／電流sourceは設定した初期値の固定DCとして計算されます。時間ごとに指令を変えるには制御を含むFeedbackを使います。

## 練習3: Buck回路を配置する

最初は50 kHz、目標24 Vの非同期Buckモデルを作ります。盤面の位置ではなく、次の電気的接続を正本にしてください。

| 部品 | 値／model | 接続と役割 |
|---|---|---|
| `V001` DC電源 | `50` V | +をDC bus、−をGND |
| `M001` N-MOSFET | `NMOS_POWER_DEFAULT` | DrainをDC bus、Sourceをswitch node |
| `VCTRL001` 制御電圧出力 | 初期`0` V、範囲`0`〜`10` V | MOSFETのGate–Source間を駆動 |
| `D001` Schottky diode | `D_SCHOTTKY_POWER` | AnodeをGND、Cathodeをswitch node |
| `R002` | `1` Ω | switch nodeから出力側へのseries要素 |
| `IP001` Iプローブ | - | `R002`と`L001`の間へ直列配置 |
| `L001` | `470u` H | switch node側から出力nodeへ直列配置 |
| `C001` | `100u` F、初期電圧`0` | 出力nodeからGND |
| `R001` load | `20` Ω | 出力nodeからGND |
| `VP001` Vプローブ | label出力電圧 | 出力nodeをGND基準で測定 |

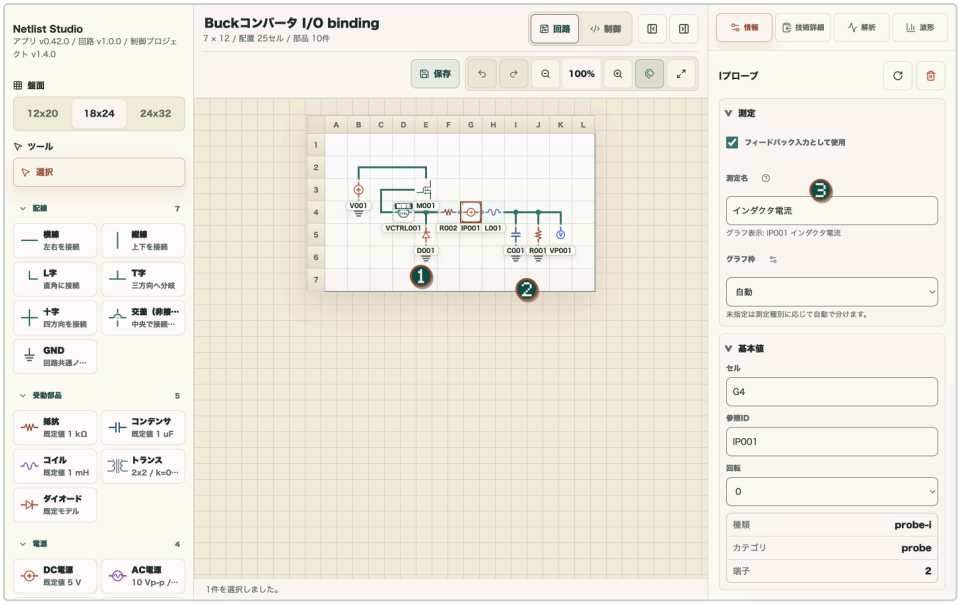



図6-1　Buck回路と制御I/Oの全体

1. **power stage**: 電源、MOSFET、diode、L、C、load、GNDの電力経路を確認します。
2. **probe／actuator**: `IP001`、`VP001`、`VCTRL001`の位置と向きを確認します。
3. **binding設定**: 選択した測定点または制御出力をどのControl blockへ公開するか確認します。

### 接続を検証する

1. DC bus、switch node、output node、GNDの4つを順に強調表示します。
2. `IP001`が電流経路へ直列に入り、別配線で迂回されていないことを確認します。
3. `VP001`がoutput nodeへ接続されていることを確認します。
4. `VCTRL001`の初期値が指令範囲内で、最小値が最大値より小さいことを確認します。
5. SPICE入力で`V001`、`M001`、`VCTRL001`、`D001`、`R002`、`IP001`、`L001`、`C001`、`R001`を確認します。

MOSFETやdiodeの向きが逆の場合は、値を調整する前に回転と端子を修正します。通常TRANで`VCTRL001`を固定DCとして使ってもPWM動作にはならないため、その結果を閉ループ動作と解釈しないでください。

## 回路と制御グラフをbindingする

1. 回路上の`VP001`を選び、Inspectorのフィードバック入力として使用を有効にします。
2. 対応する`CircuitProbeInput`が作成されたことを確認します。
3. `VCTRL001`を選び、Inspectorのbinding状態と`CircuitActuatorOutput`を確認します。計測・制御paletteから新しい専用制御出力を配置した場合は、対応ブロックも同じ操作で作成されます。
4. 上部で制御へ切り替えます。
5. 入力ブロックが`VP001`、制御先ブロックが`VCTRL001`を指していることを確認します。

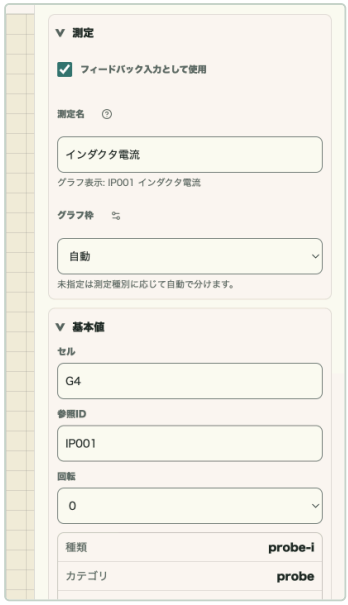



図6-2　probe／actuator bindingの設定

表示中のRefID、物理量、入力／出力方向が回路上の対象と一致していることを確認します。似た名前の別RefIDへ自動で付け替わることはありません。

RefIDを変更、削除、複製した場合はbindingを再確認します。回路の見た目が同じでも、別RefIDへ黙って接続されることはありません。

## PWMだけのopen-loopを作る

閉ループへ進む前に、固定Dutyで経路を1段ずつ確認します。

1. `Constant`、`PWM`、`Scope`を配置します。
2. `Constant.out → PWM.duty`を接続し、Constantを`0.5`にします。
3. `PWM.out → CircuitActuatorOutput.in`を接続します。
4. `Constant.out → Scope.in`も接続し、labelを`Duty指令`、step表示、単位`%`にします。
5. PWMを次の値へ設定します。

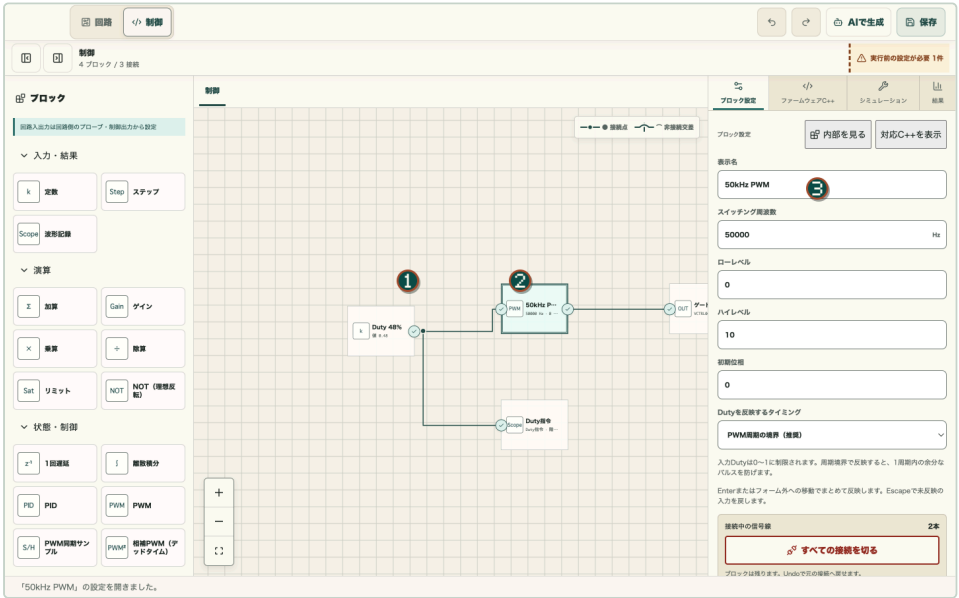



図6-3　固定Dutyのopen-loop PWM

1. **Duty指令**: Constantの`0.5`をPWMへ入力します。

2. **PWM**: 50 kHz、0〜10 VのGate波形へ変換します。
3. **actuator／Scope**: 回路制御出力へ渡す経路と観測経路を分けて確認します。

| PWM設定 | 値 |
|---|---|
| frequency | `50000` Hz |
| low／high | `0` V／`10` V |
| initial phase | `0` |
| Duty update | `period-boundary` |

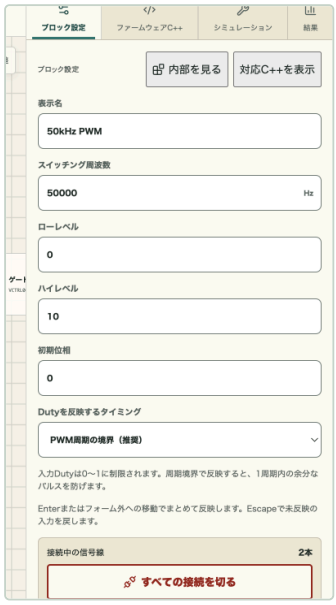


図6-4　open-loop PWMの設定

Feedbackのsample timeを`1us`、durationを`10ms`にします。PWMでは`frequency × sampleTime <= 0.1`が必要です。この例は`50000 × 0.000001 = 0.05`で条件内です。

モデル診断にerrorがなければビルドしてシミュレーションを1回実行し、Gate command、出力電圧、インダクタ電流、Dutyを確認します。理想Buckの`Vout≈Vin×Duty`は最初の目安ですが、起動過渡、model、series抵抗、diode降下、rippleがあるため完全一致は要求しません。

## PIDで閉ループにする

1. `Constant`をもう1つ配置し、labelを目標24V、値を24にします。
2. `PID`を配置します。
3. open-loopの`Constant 0.5 → PWM`を削除します。
4. 次の接続を作ります。

```
目標24V.out ──────────────→ PID.setpoint
VP001のCircuitProbeInput.out → PID.measurement
PID.out ──────────────────→ PWM.duty
PID.out ──────────────────→ Scope.in
PWM.out ──────────────────→ VCTRL001のCircuitActuatorOutput.in
```

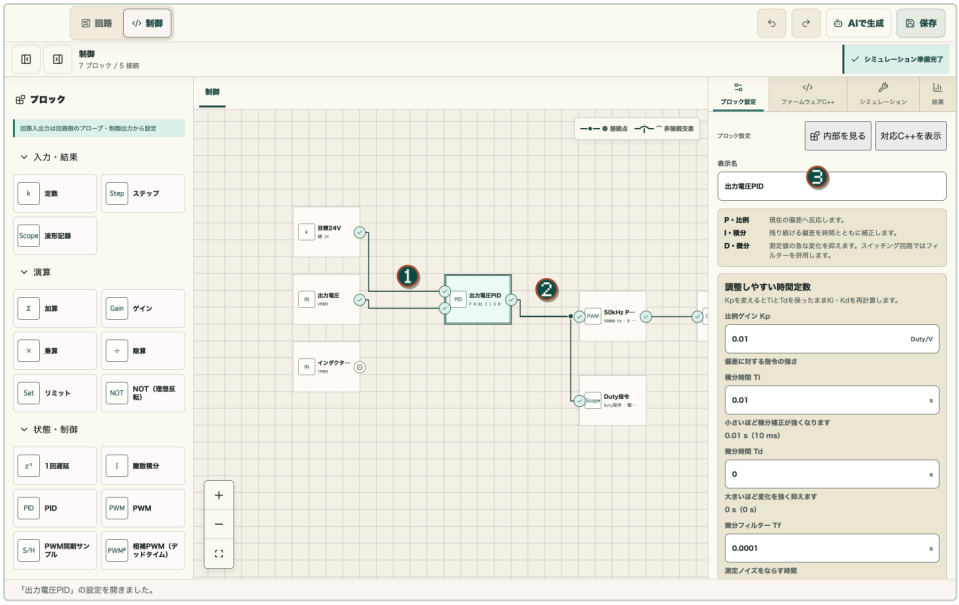



図6-5　PID closed-loopの制御graph

1. **setpointとmeasurement**: 目標24 VとVP001の測定値をPIDへ入れます。
2. **PIDからPWM**: PID出力をDuty指令としてPWMへ渡します。
3. **診断と出力**: Scope、actuator、右側のPID診断を確認します。

最初の学習値として、次を入力します。

| PID設定 | 値 |
|---|---|
| Kp | 0.01 |
| Ti | 10ms |
| Td | 0 |
| Tf | 100us |
| 出力最小／最大 | 0.05／0.95 |
| 積分初期値 | 0.505 |
| derivative | measurement |
| anti-windup | predictive conditional |
| update | every tick |

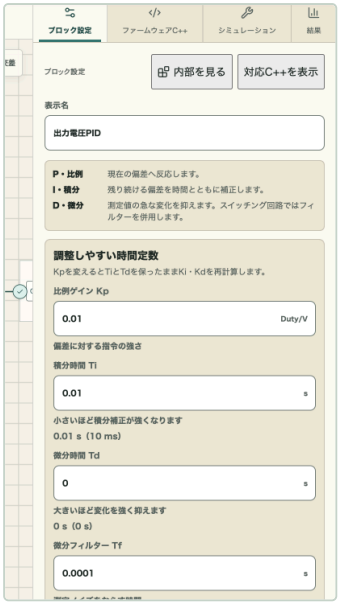


図6-6　PID設定と診断

全fieldを確定し、PID診断を読みます。空欄、入力途中の−や1e、最小値以上の最大値、非finite値がある間はBuild／Runへ進みません。

## 閉ループシミュレーションを実行する

1. 実機I/O設定で数値形式、入力、出力方式、安全値を確認します。
2. Feedbackがsample time `1us`、duration `10ms`であることを確認します。
3. model diagnosticsとfeedback diagnosticsのerrorを0にします。
4. ビルドしてシミュレーションを実行します。
5. 完了後に結果または波形を開きます。

同じmodelと検証用topologyでは、8〜10 msの出力平均23.7〜24.3 V、startup最大40 V未満、出力ripple 0.3 Vp-p以下が回帰目安です。手作業で部品、配線、model、PIDを変えたProjectへ、この数値を合格保証として流用しないでください。

中止した実行の未完成結果は採用されません。回路、制御graph、I/O、timingを変更すると以前のFeedback結果は古い状態になるため、同じ条件で再実行してください。

## 結果を診断する

共通時間軸で、最低限次を表示します。

| 系列 | 見ること |
|---|---|
| 出力電圧 | 目標、startup overshoot、定常偏差、ripple |
| インダクタ電流 | peak、ripple、異常な増加 |
| Gate command | 0／10 Vの遷移、edge数 |
| PID指令Duty | controllerが要求した比率、飽和 |
| PWM適用Duty | 周期境界で実際に適用された比率 |

PID指令と適用Dutyは、`period-boundary`のため瞬間的に一致しない場合があります。指令が上限または下限へ張り付く場合は、極性、measurement、目標、初期値、Kp／Tiを順に確認します。電流が増加し続ける場合は、結果を採用せず回路と制御を停止して見直します。

## PWM同期PIDへ進む

基本の閉ループが理解できた後だけ、`PwmSynchronizedSample`とtriggered PIDを追加します。

1. measurementを`PwmSynchronizedSample.in`へ接続します。
2. PWM Dutyを同期sampleの`duty`へfan-outします。
3. `value → PID.measurement`、`update → PID.update`、`dt → PID.dt`を接続します。
4. sample phaseを`0.75`、PID updateを`triggered`にします。
5. 例としてKp `0.01`、Ti `10ms`、Td `1ms`、Tf `100us`を使います。

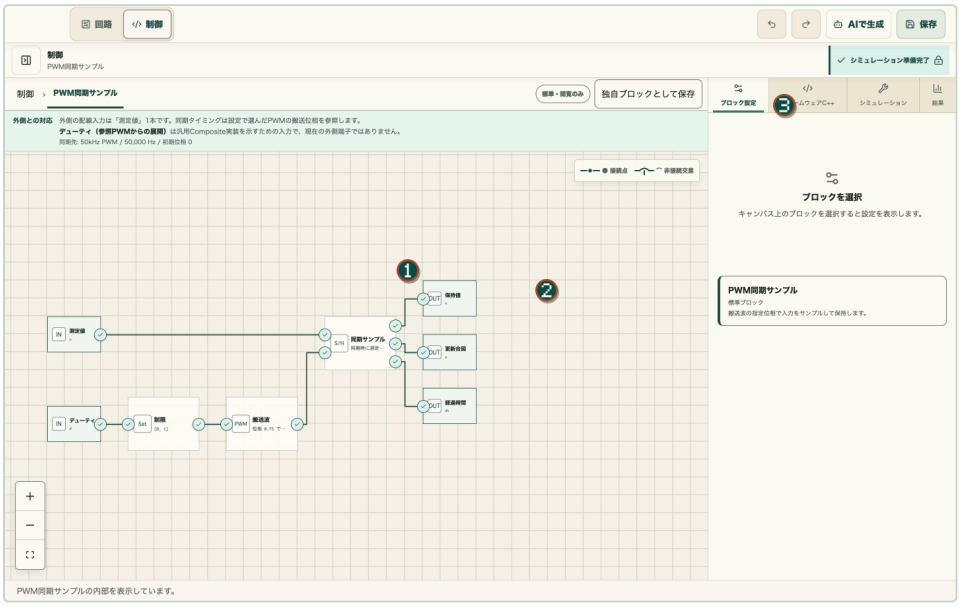



図6-7　PWM同期PIDの階層全体

1. **標準階層**: 関連するblockを階層内へまとめ、breadcrumbで現在位置を確認します。
2. **同期経路**: PWMのDutyとmeasurementから`value`、`update`、`dt`を作ります。
3. **公開port**: 階層外と接続する入出力の名前と方向を確認します。

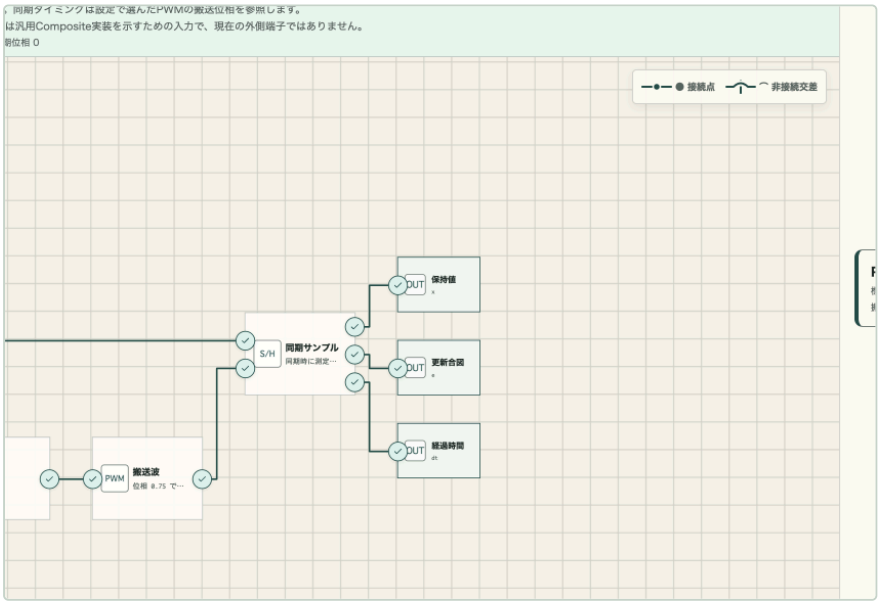



図6-8　階層の公開port

公開portは見た目の位置ではなく、port名、方向、signal typeで外部接続を定義します。階層化後もsetpoint、measurement、Duty、updateの意味が変わっていないことを確認します。

sample phaseは0以上1未満です。50 kHz、base tick 1 µs、duration 10 msではcontroller sampleは10001点、同期updateは500回が目安です。PWM同期はswitching rippleの同じ位相を測るための方法であり、電流制限や実機ADC timingを自動設計する機能ではありません。

## 相補PWMとdead time

half-bridgeへ拡張する場合は`ComplementaryPWM`を使い、上下Gateを別の制御先へ一対一でbindingします。turn-on delayは各PWM周期より短くし、例として上下各`400ns`を設定できます。

- turn-offは即時、反対側のturn-onだけを遅延します。
- 遅延中はboth-offとし、上下同時ONを許可しません。
- safe stateはboth-offです。
- 変換後は出力平均、overlap 0、最小both-off時間、温度と損失を再評価します。

dead timeを設定しただけでshoot-through防止が実機保証されるわけではありません。driver遅延、MOSFETのturn-off、Miller効果、配線寄生を実測してください。

## 完了チェック

- Buckのpower stage、測定点、Gate制御sourceを一から配置した。
- 固定DC sourceとFeedbackで動的に更新されるsourceの違いを説明できる。
- ConstantからPWMへのopen-loopを先に確認した。
- `setpoint → PID ← measurement → PWM → actuator`の閉ループを作った。
- 指令Dutyと適用Duty、出力電圧、インダクタ電流を同じ時間軸で確認した。
- PWM同期、相補PWM、dead timeの制約と実機との差を説明できる。

CHAPTER 07

# FirmwareとFeedback

生成C++、実機I/O、8ファイルのBundle、Feedbackと実機の境界を理解します。

## 正本と生成物を分ける

| 種類 | 役割 | 更新方法 |
|---|---|---|
| `.netgrid.json` | 回路、制御graph、I/O、timingの編集正本 | Netlist Studioで保存 |
| Firmware Bundle | HAL非依存C++17とintegration雛形 | Projectから再生成 |
| Feedback実行artifact | SPICE deck、WASM、manifest、結果 | 実行ごとに一時生成 |
| Graphの表示／演算 | 結果のprojection | 元datasetから再計算 |

Projectを変更したあとに古いBundleや結果を使わないでください。WASMや解析結果はProject本文の正本ではありません。

## 実機I/Oを設定する

実機I/O設定で次を1項目ずつ確認します。

| 設定 | 選択肢／意味 | 必ず確認すること |
|---|---|---|
| numeric profile | `float32`既定、または`float64` | MCU、演算精度、実行時間 |
| input codeName | 測定入力のC++名 | ADC値やsensor値との対応 |
| sampled-value output | 1つの数値を出力 | `safeValue`、単位、範囲 |
| hardware-pwm output | Duty／frequency／polarity | PWM node、`activeHigh`、`safeDuty` |
| complementary PWM | 上下Gateの1組 | dead time、両極性、both-off |

ADC channel、timer、PWM channel、MCU pin、Gate driver、電圧／電流scaleは回路netから自動推定されません。ProjectのRefIDと実機pin名が似ていても、同じものとは扱わず、integration側で明示的に対応付けます。

`safeValue`と`safeDuty`は、停止、未初期化、通信断、watchdog時に採用する設計値です。単に0を入れるのではなく、実回路で安全側になる極性と電力状態を確認します。

# 生成C++を確認する

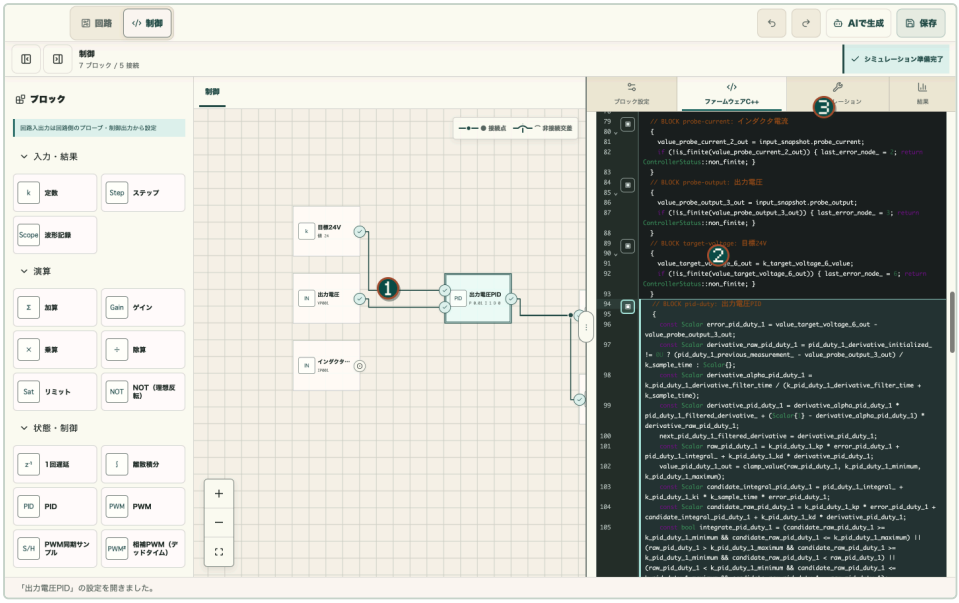



図7-1　Firmware authoring画面の全体

1. **制御graph**: 選択したblockと生成sourceの対応を確認します。
2. **生成C++**: activeな方式、保護領域、診断を確認します。
3. **I/O／Bundle**: numeric profile、mapping、安全値を確定してから書き出します。
4. ファームウェアC++を開きます。
5. キャンバス上のブロックを選び、対応するparameters、演算、state updateのsource範囲を確認します。
6. Blocks方式では、許可された数値parameterまたはCustomCpp保護領域だけを編集します。
7. プレビュー診断を確認し、変更を確定します。
8. C++ ビルドのみでsource policy、C++17、WASM ABIを確認します。

生成器が管理する構造を変更した下書きは適用されません。C++方式へ切り替える場合はC++版として編集を使用します。Blocks版とC++版は別々に保存され、activeな方式だけがBuild／Runの正本です。

Standalone C++では、dynamic allocation、thread、filesystem、network、process、任意host importなどを使えません。回路やpinへ直接アクセスするコードをcoreへ埋め込まず、Platform境界へ分離します。




図7-2　I/O mappingとBundle書き出し状態

画面に表示されたcodeName、入出力方式、safe valueが回路上のRefIDと設計資料に一致していることを確認します。Bundle完了だけでMCU固有driverや安全処理が完成したことにはなりません。

## Firmware Bundleを書き出す

1. 実機I/O設定を確定します。
2. model diagnosticsのerrorと未保存C++ draftを解消します。
3. ファームウェア一式を書き出すを実行し、空の保存先を選びます。
4. 次の8ファイルがあることを確認します。

```
netgrid_controller.hpp
netgrid_controller.cpp
netgrid_platform.hpp
integration_example.cpp
firmware-manifest.json
control-source-map.json
CMakeLists.txt
README.md
```

`firmware-manifest.json` version 4には、C++17、numeric profile、sample time、graph／behavior hash、I/O、安全値、PWM契約、使用block definition、各fileのSHA-256が含まれます。ファイルを編集した後はmanifest hashが一致しなくなるため、元Bundleと変更版を区別します。

## Platformへ接続する

1. `netgrid_platform.hpp`で、アプリが要求するinput／output契約を確認します。
2. `integration_example.cpp`で、sample取得、`Controller::step()`、出力反映の順序を確認します。
3. ADC値を物理単位へscaleし、入力codeNameへ対応付けます。
4. controllerの全出力がfiniteで範囲内であることを確認してから、同一tickの値をatomicに反映します。
5. PWMはshadow／preload相当の周期境界更新を使います。
6. fault時、watchdog時、初期化前にsafe outputへ移る実機処理を別途実装します。

PWM同期sampleでは、ADC sample取得、controller step、PWM shadow更新の順序とphaseを実機timerへ対応付けます。Netlist StudioはADC channel、timer、pinを決めません。

## Feedbackを実行する

1. activeなcontroller方式と保存済みsourceを確認します。
2. probe／actuator binding、sample time、durationを確認します。
3. model diagnosticsとfeedback diagnosticsのerrorを解消します。
4. ビルドしてシミュレーションを実行します。
5. 完了したdatasetのProject fingerprintと現在のProjectが一致することを確認します。

Feedbackでは、制御sourceの初期値と範囲を再検証し、全出力がfiniteかつ範囲内の場合だけ同一tickのlatchへ反映します。暗黙のclampで不正なcontroller出力を隠しません。

## Feedback結果を読む

結果では、probe、actuator、Scopeを共通時間軸へ統合します。Buck例では次を確認します。

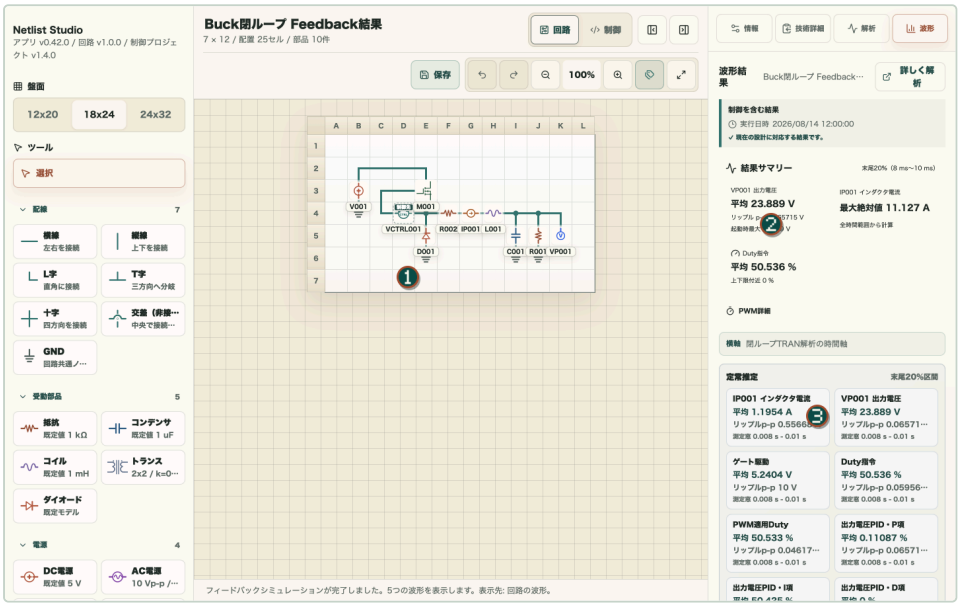



図7-3　Feedback実行完了後の全体

1. **回路**: 結果を生んだpower stage、probe、actuatorを再確認します。
2. **結果summary**: 出力電圧、peak current、Duty指令を確認します。
3. **定常推定**: 測定窓、ripple、適用Duty、PID内部系列を確認します。

- 出力電圧とインダクタ電流。
- Gate commandまたは上下Gate。
- PIDのraw／limited／applied Duty。
- startup最大値、定常平均、偏差、ripple、peak current。
- 相補PWMでは上下同時ON時間と最小both-off時間。



図7-4　Feedback波形の定常summary

画面例では末尾20%の測定窓で出力平均`23.889 V`、Duty指令平均`50.536 %`です。この数値は固定fixtureの結果であり、別の部品値やmodelを使った回路の合格値として流用しません。

PID内部系列はapplicationがaccepted probe系列から再生するprojectionであり、権威あるsimulation artifactそのものではありません。C++単独方式のprivate stateは推測しません。

回路、制御graph、numeric profile、I/O、timing、active sourceを変更した場合は、以前の結果を古い状態として扱います。plotの並べ替えだけではbehaviorを変更しません。

## シミュレーションと実機の差

| シミュレーションで省略／理想化されやすいもの | 実機で必要な確認 |
|---|---|
| switch／diodeの損失と温度依存 | junction温度、heatsink、SOA |
| Gate driver遅延と電源 | UVLO、source／sink電流、Miller耐性 |
| 配線、ESR、ESL、EMI | layout、loop面積、filter、測定方法 |
| ADC／timer量子化とjitter | resolution、trigger、latency、clock drift |
| faultと電源投入順序 | fuse、current limit、interlock、watchdog |
| modelの定格外挙動 | datasheet、derating、破壊mode |

最初の実機bring-upは、回路有識者によるreview、絶縁された計測、電流制限、低エネルギー条件、非常停止、safe Duty確認を完了してから行います。50 Vの学習値をそのまま実機へ適用しません。

## 完了チェック

- Project、Firmware Bundle、Feedback artifactの正本関係を説明できる。
- `float32`／`float64`と実機I/O設定を確認した。
- 8ファイルのBundleとmanifest hashを確認した。
- ADC／timer／pinが自動推定されないことを理解した。
- Feedbackのstale条件とPID projectionの境界を理解した。
- シミュレーション外の定格、保護、bring-up手順を別途設計する必要を理解した。

CHAPTER 08

# 機能一覧

主な機能の導入版、安定度、制約を確認します。

## 安定度の読み方

`stable`は正式版で通常利用する機能、`experimental`は人手検証と結果確認が必要な機能です。導入versionは、現在のGit履歴でその機能を含む最初の正式release tagです。後方互換修正や安全強化が後のversionへ追加されているため、導入versionの古いアプリを推奨する意味ではありません。

## 対応環境

正式配布はmacOS Apple Silicon向けです。ngspice解析、外部toolchain、実機I/Oは別途導入した実行環境に依存します。未公開platformの導入手順や、未検証のpackage変換はこのガイドの対象外です。

## 主な機能

| 機能 | 導入version | 安定度 | 主な前提／制約 |
|---|---|---|---|
| 回路編集、配線、接続解析、SPICE入力生成 | v0.2.0 | stable | GND、model、値、接続は利用者が確認 |
| ngspice直接解析 | v0.11.7 | stable | ngspice 46、対応platform、解析点数上限 |
| Graph解析window | v0.24.0 | stable | 元dataset、軸、単位、解析条件を確認 |
| Control graph、Feedback閉ループ解析 | v0.31.0 | experimental | binding、timing、toolchain、結果の人手検証 |
| AIによるControl提案 | v0.31.0 | experimental | previewと差分を明示確認、自動適用／自動simulationなし |
| Firmware C++、実機I/O、hardware PWM | v0.32.0 | experimental | HAL、ADC、timer、pin、保護は別途実装 |
| 独自Composite block | v0.39.23 | experimental | definition revision、port、依存、展開上限 |
| アプリ内更新確認 | v0.40.0 | stable | releaseごとの注意、対応するstable channel |
| native menu、名前を付けて保存のtransaction | v0.41.0 | stable | desktopのwindow／focus状態に応じて有効化 |
| Graphの2点測定、区間統計 | v0.42.0 | stable | X範囲内へsnap、A／Bは異なる点 |
| Graphの演算波形 | v0.42.0 | stable | 単位整合、式preview、元datasetは不変 |

## 機能別の使い分け

| 目的 | 最初に使う機能 | 次に確認する機能 |
|---|---|---|
| DC回路の電圧確認 | 回路編集＋OP | SPICE入力、Vプローブ |
| filter設計 | AC／TRAN | Graph、2点測定、演算波形 |

| 目的 | 最初に使う機能 | 次に確認する機能 |
| --- | --- | --- |
| switching回路 | TRAN | step表示、電流probe、point数 |
| PWM制御 | Control graph | Scope、指令／適用Duty、Feedback |
| 実機組込み | Firmware Bundle | manifest、Platform実装、safe output |
| 再現可能な共有 | `.netgrid.json` | 解析条件、version、manifest／hash |

## stableでも保証されないもの

`stable`は、入力した回路の安全性、部品定格、SPICE modelの正確さ、実機の規格適合を保証する表示ではありません。次は利用者側で確認します。

- 部品の最大定格、SOA、温度、derating。
- GNDと測定基準、probeの影響。
- modelと実部品の差、寄生成分。
- 解析step、収束、初期条件、結果の許容差。
- 生成物のversionと現在のProjectの一致。

## experimentalを使う条件

- 元Projectを保存し、変更前へ戻せる。
- AI／CustomCpp／C++のpreviewと診断を読める。
- I/Oの物理単位、範囲、極性、安全値を明示できる。
- Feedback結果と実機測定を別のevidenceとして扱える。
- failure時に出力をsafe stateへ移す実装と試験がある。

CHAPTER 09

# 技術モジュール

UI、controller、domain、adapter、workerの責務とデータフローを確認します。

## 公開向けの責務分担

Netlist Studioは、表示、操作調整、純粋なdomain規則、OS境界、重い計算を分離します。

| 層 | 公開sourceの目安 | 入力 | 所有する責務 | 出力 |
|---|---|---|---|---|
| UI | `src/components/`、各windowのApp | 利用者操作、view model | 盤面、制御canvas、Graph、focus、入力draft、診断表示 | intent、確定入力 |
| controller | `src/app/controllers/` | UI intent、現在revision | use caseの順序、stale応答破棄、pending操作 | domain command、adapter request |
| domain | `src/domain/` | versioned data、command | 回路、接続、解析、制御、I/O、診断の決定的規則 | 検証済みmodel／artifact description |
| Electron adapter | `electron/` | allowlist済みrequest | native menu、dialog、file、外部process、window／IPC境界 | bounded result、structured error |
| worker | worker entryと純粋計算module | snapshot、計算条件 | Graph演算、routing、重いprojectionをUI threadから分離 | revision付き計算結果 |

UIはdomain規則を独自に再実装せず、controllerは回路式やPWM semanticsを所有しません。Electron adapterはrendererから任意のfile、process、command、URLを受け取らず、固定または検証済みcontractだけを扱います。

## 回路編集からGraphまでのデータフロー

```
UIの入力中の値
  → controllerが確定操作とdocument revisionを束縛
  → domainが部品値・接続・解析条件を検証
  → Electron adapterが固定ngspice executableでbounded jobを実行
  → parserがresultをversioned datasetへ変換
  → workerがGraph表示・演算・統計を計算
  → UIが同じrequest／revisionの結果だけを表示
```

途中でProjectが変わった場合、古い応答を新しいProjectへ適用しません。外部processのstdoutをそのままUIへ流さず、size、schema、値、単位を検証します。

## ControlとFeedbackのデータフロー

```
Board + ControlGraph + Firmware I/O + timing
  → domain validation
  → ControlProgramIRへlowering
  → reference evaluator／C++ generator／WASM adapter
  → main processがSPICE deck・manifest・WASMを再検証
  → bounded cosimulation
  → result parserとProject fingerprint照合
  → 共通Graph datasetとPID projection
```

reference evaluator、生成C++、WASMは同じevaluation order、I/O index、state layoutを共有します。PWMのphase、Duty latch、PID stateはtick境界で一括更新し、direct-feedthroughのalgebraic loopは生成前に拒否します。

## module contractの読み方

各moduleを調べるときは、実装行より先に次の5点を確認します。

1. 入力schemaとversion。
2. normalizationとresource limit。
3. error／warningのdiagnostic code。
4. 成功時に所有権が移るdataと、移らないdata。
5. stale、cancel、timeout、partial failure時の不変条件。

たとえば`analysisSettings`は解析directiveと点数見積りを所有しますが、ngspice processは起動しません。`feedbackSimulationJob`はjob lifecycleを所有しますが、renderer表示や回路semanticsは所有しません。

## 部品を拡張する

新しい回路部品は、次の順で追加します。

1. symbol definitionで端子、向き、既定値、categoryを定義する。
2. sanitizerで値、単位、範囲、legacy入力を正規化する。
3. connectivityとnetlist generatorへ電気的semanticsを追加する。
4. Inspectorへtransactional inputと日本語診断を追加する。
5. ngspice／LTspice出力、保存round-trip、invalid入力をtestする。

見た目のglyphだけ追加しても、接続、保存、SPICE、診断が揃わなければ公開機能にはしません。

## 制御blockを拡張する

新しいControl blockは、port、signal type、direct feedthrough、state、parameter、diagnosticをdomainで定義します。その後、reference semantics、IR lowering、C++ codegen、必要なUI editorを追加します。

- combinational blockはcycle検査へ参加します。
- stateful blockはcurrent stateとnext stateを分けます。
- firmwareへ出すblockは`float32`／`float64`の両方を検証します。
- PWMやI/Oを持つblockはtiming、range、safe stateを明示します。

- Custom／Compositeはnode、edge、source、depthの上限を超えたら部分結果を使わず停止します。

## Graph演算を拡張する

Graph演算は元datasetを変更しない純粋なprojectionとして追加します。

1. 対応するX軸と系列数を定義する。
2. 単位の組合せと出力単位を定義する。
3. zero division、非finite、点数不足を診断する。
4. 追加前previewへ式、label、単位、点数を表示する。
5. worker結果へrequest IDとrevisionを付け、stale結果を破棄する。

## OS／security境界

- Main windowとGraph windowの権限を分離します。
- rendererはNode.js、任意process、任意file、任意navigationへ直接到達しません。
- dialogの取消、保存失敗、stale completionでは元のdocument handleとdirty状態を維持します。
- 外部processは固定実行対象、bounded input／output、timeout、cancel、cleanupを持ちます。
- credential、署名鍵、provider設定、内部運用証跡は公開ガイドへ掲載しません。

## 問題を追う順序

1. UIに表示された最初のdiagnostic codeと対象field／nodeを記録します。
2. controllerがどのuse caseとrevisionを送ったか確認します。
3. domain validatorが同じsnapshotを受理するか確認します。
4. adapter境界へ渡ったversioned requestとbounded resultを確認します。
5. workerまたは外部processの結果が現在revisionと一致するか確認します。

UI表示だけ、外部logだけ、古いartifactだけで原因を断定しないことが、module境界を保った調査の基本です。

CHAPTER 10

# トラブルシューティング

代表的な失敗を切り分け、安全に回復します。

## 切り分けの順序

問題が起きたら、同じ操作を繰り返す前に次を記録します。

1. アプリとガイドのversion。
2. 保存済みProject名と、未保存変更の有無。
3. 実行した解析／Build／Feedbackの条件。
4. 最初のdiagnostic codeとmessage。
5. 期待値と実際値。

最初のerrorを直した後にだけ次のerrorを確認します。Project、生成物、過去結果を混ぜず、どのsnapshotから生じた結果かを固定します。




図10-1　解析を止めるblocking warning

1. **対象回路**: 警告が指すcellと未接続端子を盤面で確認します。
2. **最初の重大警告**: `GND`が見つかりませんのような解析不能条件を先に直します。
3. **回復案内**: 実行buttonが無効な理由と、次に行う修正を読みます。

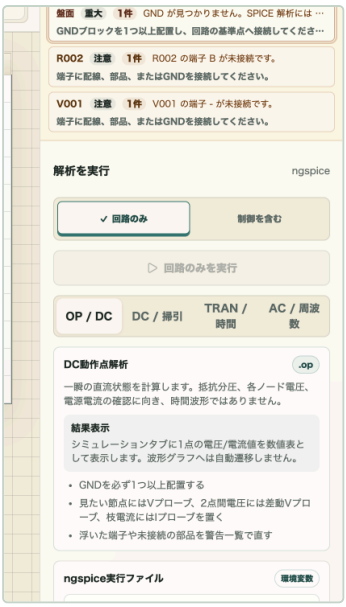



図10-2　警告の原因と修正対象

画面例ではGND不足が重大、R002とV001の未接続が注意として分かれています。重大項目を解消してから残る注意を確認し、実行buttonが有効になったことを確かめます。

## 代表的な失敗と回復

| 状況 | 最初に確認 | 回復方法 |
|---|---|---|
| 部品を配置できない | 占有範囲の重なり、盤面外、回転 | プレビューが有効な空きセルを選ぶ |
| 見た目は接しているのに未接続 | 端子の向き、配線cell、bridge | net強調を使い、端子へ接する配線を置く |
| SPICE入力を生成できない | 未接続、値error、RefID重複、GND | 警告一覧の対象cellを修正する |
| ngspiceを検出できない | ngspice 46、設定した実行file | 設定でexact executableを選ぶ |
| ngspiceが失敗する | 最初の診断、model、収束、解析条件 | SPICE previewと回路を直し、条件変更は根拠を記録する |
| 波形が表示されない | probe、解析種別、実行error | 測定点と保存対象、現在datasetを確認する |
| A／Bを配置できない | 測定点button、X範囲、drag | 配置する点を選び、X範囲内の値を入力する |
| FFTの測定点A–Bを使えない | A／B不足、同じXへのsnap、点数 | 異なるXへ置き、十分な定常区間を選ぶ |
| 演算波形を追加できない | 系列、単位、式、計算点数 | previewの最初のerrorを修正する |
| 制御blockを接続できない | port方向、signal type、cycle | 接続候補の診断を読み、正しいportへ接続する |
| Buildできない | 未確定draft、model diagnostics、C++ source | fieldを確定し、active方式の最初のerrorを直す |
| Feedbackを開始できない | probe／actuator、timing、ngspice、toolchain | feedback diagnosticsを上から解消する |
| 結果が古い | Project／I/O／timing変更後か | 現在snapshotで再実行する |
| 保存操作が止まる | 未確定または無効な入力 | 入力を修正するか、破棄を明示して再開する |

| 状況 | 最初に確認 | 回復方法 |
| --- | --- | --- |
| アップデートを確認できない | 更新server、通信、正式配布版か | Projectを保存し、配布サイトの注意とDMGを確認する |
| 利用ガイドが開かない | 通信、サイト、アプリversion | 接続を確認し、配布サイトから`/guide`を開く |

## 電圧分割が5 Vにならない

1. `V001`が10 Vか確認します。
2. `R001`と`R002`がどちらも10 kΩか確認します。
3. 電源−と`R002`下端が同じGND netか確認します。
4. `VP001`が2つの抵抗の中点へ接続されているか確認します。
5. 解析がOPか確認します。

probeが負荷として分圧を変えるmodelではないこと、別の抵抗や短絡が中点へ接続されていないことも確認します。

## RCの遮断周波数が1.59 kHzにならない

1. `R=1k`、`C=100n`であることを確認します。
2. `100n`を100 nFとして解釈した表示になっているか確認します。
3. AC電源の小信号振幅を1、phaseを0°にします。
4. `VP_OUT`が抵抗とコンデンサの中点か確認します。
5. dBの対象が出力単独か、`VP_OUT / VP_IN`の比か確認します。
6. sweepが10 Hz〜100 kHzを含むか確認します。

TRANではdelayを含む絶対時刻とstep後の経過時間を混同しないでください。delay 100 µsの例では1 τの点は200 µs付近です。

## Buckが目標へ収束しない

1. MOSFET、freewheel diode、L、C、loadの向きとnetを確認します。
2. `VP001`と`VCTRL001`のbinding RefIDを確認します。
3. open-loopのConstant 0.5でPWM経路を先に確認します。
4. setpointとmeasurementがPIDの正しいportへ入っているか確認します。
5. `frequency × sampleTime <= 0.1`、durationがsample timeの整数倍か確認します。
6. PIDの出力min／max、初期積分、極性、飽和診断を確認します。
7. 出力電圧だけでなく、インダクタ電流とDutyも確認します。

異常な電流、非finite、range error、上下Gate overlapがある結果は採用しません。gainを大きくするだけの調整をせず、回路とmeasurementから戻って確認します。

## Firmware Bundleを実機へ組み込めない

| 症状 | 確認 |
| --- | --- |
| compile error | C++17、numeric profile、変更したcore、Platform実装 |
| ADC値が合わない | channel、offset／gain、単位、codeName |
| PWMが逆極性 | `activeHigh`、driver、timer polarity、safe Duty |

| 症状 | 確認 |
| --- | --- |
| Duty更新が不安定 | shadow／preload、sample順序、period boundary |
| manifest hash不一致 | 書き出し後の変更、別Projectのfile混在 |
| 停止時に出力が残る | watchdog、fault handler、safe state、driver enable |

生成された`integration_example.cpp`は接続例であり、対象MCUの完成driverではありません。vendor HALと安全処理を別途実装し、reviewと実機試験を行います。

## 安全に使うための注意

- `experimental`のAI提案は、previewと検証結果を確認してから適用してください。
- 独自block、Firmware authoring、Feedbackは、重要な設計へ使う前に利用者側で検証してください。
- API keyなどのcredentialをProject、guide、screenshot、logへ含めないでください。
- 生成C++は実機固有の保護を自動保証しません。`safeValue`、極性、PWM、timer、pin割当を実機側でも検証してください。
- capacitorを含む回路は電源off後も電荷を保持する可能性があります。
- scopeのGND clipやUSB接続で意図しない短絡を作らないよう、絶縁と基準電位を確認してください。
- Netlist Studioの警告とシミュレーション結果だけで、回路定格、絶縁、安全規格への適合を判断しないでください。

## 問い合わせ用の最小情報

credentialやprivate pathを除き、次を揃えると再現しやすくなります。

- Netlist Studio／macOS／ngspiceのversion。
- 最小化した`.netgrid.json`または再現手順。
- 解析条件、I/O設定、active controller方式。
- 最初のdiagnostic codeとmessage。
- 期待値、実際値、A／B位置または該当時刻。
- Project変更後に再実行したかどうか。