Advanced PLD 98 目次

Advanced PLD 98 目次 **1**

Advanced PLD 98 **3**

Advanced PLD - **統合** PLD **開発環境** 3
データーフロー 5
システムの概要 6

Advanced PLD の紹介 **8**

Advanced PLD **の設定と使用法** 8
Advanced PLD **コンパイラーの設定** 9
Advanced PLD **によるコンパイル** 15
Advanced PLD **によるシミュレーション** 15
Waveform **エディター** 17

Advanced PLD による設計作業の基礎知識 **20**

ステップ 20
コンパイラーソースファイルの作成 20
簡単な論理設計の例 33
簡単なゲートの設計例 35

設計のサンプル **41**

Step 1 - **設計作業の確認** 41
Step 2 - **コンパイラーソースファイルの作成** 43
Step 3 - **式の公式化** 44
Step 4 - **ターゲットデバイスの選択** 46
Step 5 - **ピン割り付け** 47
Step 6 - PLD **ソースファイルのコンパイル** 48
Step 7 - **シミュレーションテストベクタファイルの作成** 54
Step 8 - **デバイスのシミュレーション** 57
Step 9 - **シミュレーション波形の表示** 59
概要 59

設計例 **60**

例 1 - **簡単なゲート** 61
例 2 - TTL **設計の** PLD **への変換** 63
例 3 - 2 **ビットカウンタ** 67
例 4 - **デカードアップ/ダウンカウンタ** 69
例 5 - 7 **セグメントのディスプレイデコーダ** 75
例 6 - **ロードリセット機能付きの** 4 **ビットカウンタ** 77

PLD のシミュレーション **79**

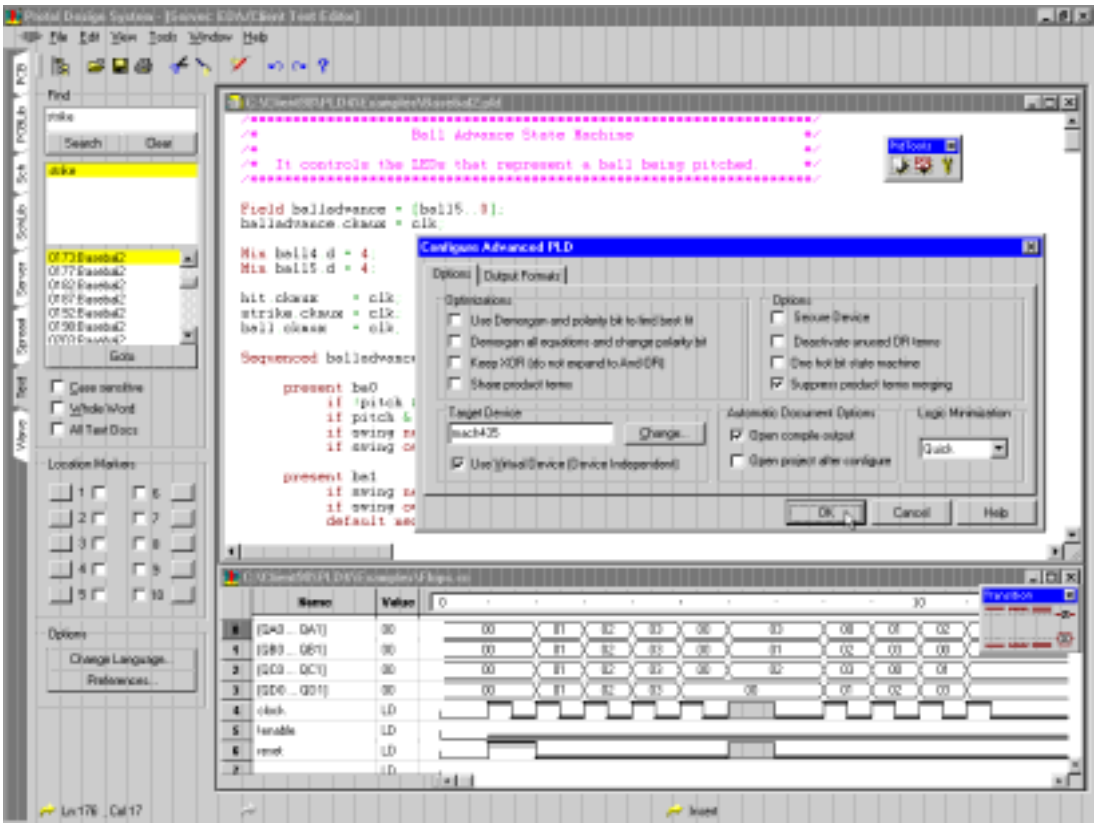
シミュレーターへの入力 79
シミュレーターからの出力 79
シミュレーションテストベクタファイルの作成 79
ヘッダー情報 80
デバイスの指定 80
コメント 81

ステートメント 81
変数宣言 (VAR) 89
シミュレーターディレクティブ 89
アドバンストシンタクス 91
繰り返し命令 93
仮想シミュレーション 99
欠陥のシミュレーション 99

ファイルフォーマット 100

ダウンロードフォーマット 100
ドキュメンテーションファイルフォーマット 103
バークレイ PLA ファイルフォーマット 108

Advanced PLD 98



Protel Designer’s Handbook guide のこのセクションでは、Advanced PLD を使用したプログラマブルロジックデバイスの作成について説明します。このセクションでは、PLD 設計の詳細が記載されています。設計のサンプルと例を使用して、PLD 設計プロセスを理解してください。Advanced PLD で使用する CUPL ハードウェア記述言語についての詳細はオンラインヘルプを参照してください。

➔ Schematic から PLD を作成する方法に関しては、オンラインヘルプを参照してください。

Advanced PLD - 統合 PLD 開発環境

Advanced PLD により、PLD 設計に統合化の新しい段階がもたらされます。EDA/Client に Advanced PLD をインストールすれば、ハードウェア記述言語 CUPL による論理記述ができるようになります。

コンパイルのボタンを押し、Advanced PLD の統合化されたエラーレポーティングを使用してエラーを修正して下さい。コンパイラーは、業界標準の JEDEC プログラムファイルを作成し、デバイスプログラマーにダウンロードできます。

Advanced PLD により、今日使用できる最大のデバイスライブラリーの 1 つが提供されます。このライブラリーは、主な製造元からのプログラマブルロジックがサポートされています。Advanced PLD の CUPL 言語は、製造とは独立しておりエンジニアは設計とパッケージングを自由に行なうことができます。

Advanced PLD シミュレーターを使用して、設計を実装する前にシミュレーションを実行して下さい。そして、シミュレーションの結果を Advanced PLD に付属の Waveform エディタを使用して検証して下さい。

Advanced PLD のパッケージには、PLD 設計用に特別に仕立てられている 3 種類の EDA/クライアントサーバーがあります。TextExpert：文法チェック用のテキストエディタ、PLD：

設計をコンパイルしたりシミュレーションを実行します。Wave：シミュレーションの波形を表示します。

テキストエディター

ロジックやシミュレーションのリストファイルを記述するために、Advanced PLD の文法チェック用のテキストエディタの TextExpert を使用して下さい。Advanced PLD には、統合化されたエラーレポート機能が組み込まれています。コンパイルやシミュレーション中にエラーが見つかると、自動的にソースファイルの中でその時のエラーの状態に応じてハイライト表示されます。

カットやコピー、貼り付け、検索、再配置など通常のテキスト編集機能に加えて、テキストエディターには、Syntax Highlighting と呼ばれる機能があります。Syntax Highlighting 機能により文法に基づいて異なるワードタイプやシンボル、識別子をそれぞれの色で強調表示します。この機能によりドキュメントの編集が便利になります。特に PLD 論理記述ファイルなどで繰り返し作業を行なう時などに大変便利です。

Advanced PLD コンパイラー

Advanced PLD コンパイラーには、4 段階の最小化で、プログラマブル論理回路を最小化する高速で強力なミニマイザがあります。コンパイラーは、分配属性やドモルガンの定理を用いてブーリアン表現で簡潔に表現されます。

コンパイラーは、業界標準の JEDEC ファイルを作成します。このファイルは、JEDEC フォーマットをサポートする論理プログラマーと互換性のあるダウンロードファイルフォーマットです。Advanced PLD は、主な製造元からのプログラマブルロジックをサポートしており、設計とパッケージングを自由に行なうことができます。

Advanced PLD シミュレーター

Advanced PLD シミュレーターを使用して、論理回路が実装される前にシミュレーションを実行して下さい。入力と出力から PLD の機能を予測した記述をシミュレーションファイルに作成します。シミュレーターは、予想される値とコンパイラーの動作中に計算される実際の値とを比較します。

シミュレーターを使用すると、PLD をプログラムする前にロジックを試すことができます。この機能は、デバイスの損傷を防止したりシステムレベルの問題をデバッグする時に役立ちます。シミュレーターにより確認されたテストベクタは、ロジックプログラマーにダウンロードされます。

シミュレーションの波形の表示

シミュレーションの結果は、Advanced PLD の Waveform エディターにより検証することができます。このエディターは、シミュレーションリスティングファイル(ファイル名.SO)を読み込み、スプレッドシートスタイルで波形をエディタウィンドウに表示します。

CUPL 高水準言語

Advanced PLD は、CUPL ハードウェア記述言語を使用します。このガイドの中では、この言語は、CUPL または CUPL 言語と表します。

設計時間を節約するために、CUPL 言語には、リストやアドレス範囲、ビットフィールドのための式や表記が用意されています。Truth テーブルシンタックスにより、ある種の論理記述を簡潔に表現することができます。

CUPL 言語のリファレンスのセクションは、このマニュアルにあります。

広範囲のデバイスサポート

Advanced PLD は、プログラマブルロジックの主な製造元のデバイスをサポートしています。また、Advanced PLD は 2 つの優位性を持っています。1 つは、1 つの開発環境と 1 つの言語を習得するだけで使用できることです。Advanced PLD を使用すると PAL16L8 を用いて簡単なアドレスデコーダを設計したり、現在お手持ちの設計やパッケージを Xilinx5000 シリーズのデバイスを用いて開発したりできます。2 つめは、同じ機能のロジックを違う部品にパッケージすることができということです。これにより、デバイスの製造元の選択に自由度が増します。

レジスタエミュレーション

この機能により、目的のアーキテクチャーを持ったレジスタによらず、あらゆる種類のレジスタを使用できます。例えば、このような機能を利用して、JK レジスタで設計して、ターゲットデバイスには D レジスタしかないデバイスを使用することが可能になります。

ドモルガンの最適化

プログラマブル極性を持つデバイスでは、式にドモルガンの定理を適用して回路の効率を上げることができます。設計者は、コンパイラーがドモルガンの定理を適用するのに最適であるとしたところに、定理を適用させることができます。

D/T 最適化法

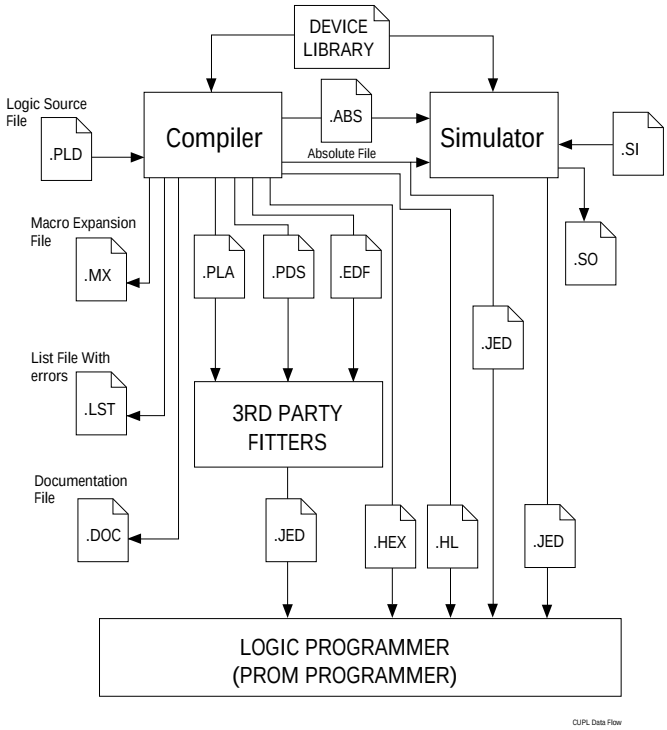
D レジスタと T レジスタの両方を持つデバイスでは、あるレジスタタイプを他のレジスタへ変更することで使用されるプロダクトタームの数を減らすことができます。コンパイラーは、どのレジスタを使用すればより最適かを決定し自動的に配置を実行します。

データーフロー

このセクションでは、Advanced PLD のデータフローについて説明します。

最初に、CUPL 言語を使用して論理記述のソースファイルを作成します。このファイルには、プログラマブルロジックデバイスに割り付けられるロジックが記述されます。

Advanced PLD によりソースファイルをコンパイルしデバイスプログラマーへダウンロードするヒューズマップファイルを作成します。コンパイラーは、Configure Advanced PLD ダイアログボックスの Absolute ABS オプションを On にすることでシミュレーターで使用される.ABS ファイルを作成します。



そしてAdvanced PLDシミュレーターにより、デバイスのシミュレーションを実行します。シミュレーターは、テスト仕様ファイル(.SI)を必要とし論理回路を検証します。シミュレーターにより、テスト仕様ファイルで予測される値とコンパイラーにより作成されたアブソリュートファイルの実際の値

とを比較します。シミュレーションにエラーがなく完了すると、確認済みのテストベクタが、コンパイラーにより生成されたダウンロードファイルに追加されます。

この時点で、確認済みのヒューズマップファイルがデバイスプログラマに渡されます。

システムの概要

Advanced PLD コンパイラーとシミュレーターでは以下のモジュールが使用されます。

言語プロセッサー

CUPLX モジュールは.PLD(入力)ファイルを検索し、$DEFINE などのプリプロセッサディレクティブを処理します。このモジュールはすべてのプリプロセッサディレクティブが展開された中間ファイルを作成します。

パーサーとステートマシントランスレーター

CUPLA モジュールは CUPLX により生成された中間ファイルを検索し、シンボルテーブルを作成して式を展開します。また、このモジュールは、ステートマシンや真理値表、ユーザ定義関数をブーリアン式に展開します。そして、レンジ命令を処理中に、ロジックを簡潔にする作業を実行します。

デバイス互換チェッカーとドモルガナイザー

CUPLB モジュールは、デバイスライブラリー(.DL)ファイルの使用ピンアウトに対して、PLD ソースファイルで選択されたピンアウトを検証します。また、デバイスのアーキテクチャーに基づいて、ソースファイル中の変数が正しく使用されているかを確認します。CUPLB は通常、DeMorganizer として参照されます。すなわち、デバイスのピンの極性と変数の極性とが合っていない場合、自動的にドモルガンの定理が実行されます。これによりアクティブローのデバイスでアクティブハイを実現できます。その逆も同様です。また、コンパイルのこの段階で、最小化の最初の段階が実行されます。最小化により 0 や 1、冗長なターム、他のプロダクトタームに含まれるプロダクトタームが削除されます。CUPLB は、デバイスモデルのリンクや論理設計を表わすビットマップを含む最終的なシンボルテーブルを作成します。

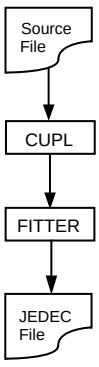
論理最小化

CUPLM モジュールは CUPLB で生成されたロジックのビットマップ表現に論理最小化アルゴリズムを実行します。このモジュールは、最小化が必要な式にだけ実行されます。

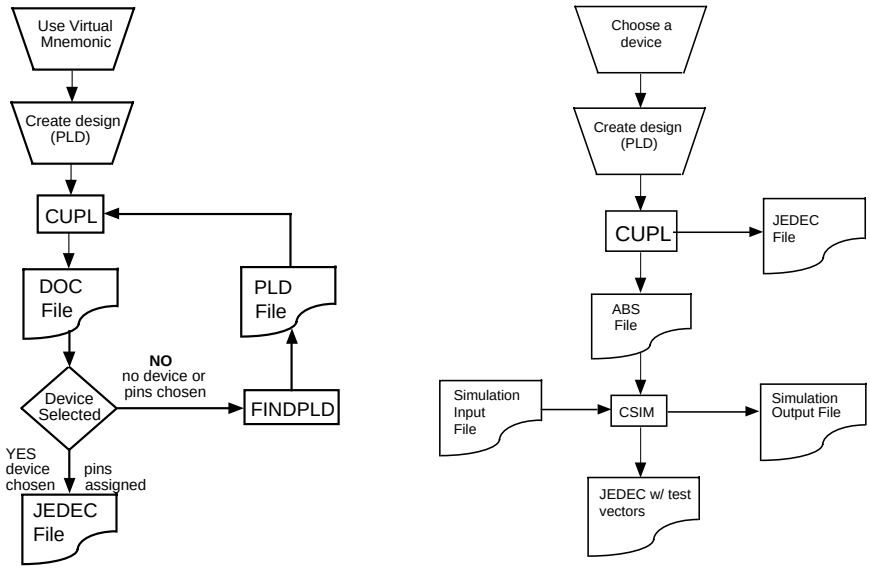
ミニマイザは、ソースファイルの式で使用できる最小化アルゴリズムを実行します。最小化アルゴリズムには Quick、Quine-McClusky、Presto、Expresso があります。PAL アーキテクチャーに最適のアルゴリズムは Quine-McClusky です。そして、PLA アーキテクチャーに最適のアルゴリズムは、Expresso(プロダクトタームの分配に強い)です。一般に、Expresso は処理速度が速く Quine-McClusky とほとんど同じ結果になります。

フィッターと出力ファイルジェネレーター

CUPLC モジュールは、論理回路をデバイスに合わせるモジュールです。このモジュールはビルトインアルゴリズム、またはサードパーティーのフィッターを使用してデバイスのフィットを実行します。CUPLC は、複雑なデバイスのマイクロセルやマルチプレクサを操作してブーリアンロジックをデバイスへフィットさせます。回路がデバイスにフィットできない場合、CUPLC は、回路をフィットできない理由が示されたエラーメッセージを表示します。また、このモジュールは Configure Advanced PLD ダイアログボックスで指定されるオプションで示されるファイルをすべて作成します。



CUPLC により、回路がターゲットデバイスアーキテクチャーにフィットするかやヒューズマップを作成するかどうかが決定されます。ヒューズマップやシンボルテーブルは、ドキュメントや JEDEC ファイルを作成するために使用されます。



デバイスに独立した設計フロー　　*デバイスを特定した設計フロー*

現在使用できるフィッタープログラムを以下に示します。

CUPLC	Generic device fitter
FITR	MACH device fitter
FITMAPL	MAPL device fitter

シミュレーター

シミュレーターは、実際のデバイスで回路がどのように動作するかに影響を与えるユニットの遅れをシミュレーションします。あるデバイスの特異性や内部遅れはシミュレーションされません。しかし、このシミュレーターは、デバイスの特徴を表現し、実装に関して何も気にしないで実装できます。

CUPL.DL

デバイスアーキテクチャー情報はすべてデバイスライブラリー(CUPL.DL)にあります。このライブラリーは、ソースファイルのコンパイル中にコンパイラーにより使用されます。デバイスライブラリーの情報には、拡張できるデバイスアーキテクチャー情報やシミュレーションに必要な情報が含まれます。

Advanced PLD で使用するファイル拡張子

拡張子	ファイルタイプ
.PLD	論理記述設計ファイル
.SI	シミュレーターリスティングファイル(入力)
.SO	エラーの記述されたシミュレーター出力ファイル
.HL	HL ダウンロードファイル
.HEX	Hex ダウンロードファイル
.JED	テストベクタの記述されていない JEDEC ファイル テストベクタの記述された JEDEC ファイル

Advanced PLD の紹介

このセクションでは、Advanced PLD のプログラマブルロジック設計のプロセスを紹介します。まず、Advanced PLD の各コンポーネントを紹介し、コンパイラーをどのように配置しているか、どのように論理記述をコンパイルするか、また、どのようにデバイスのシミュレーションを行ないシミュレーション結果を検証するかを簡単に説明します。

Advanced PLD のコンポーネントには以下のものがあります。

- Advanced PLD ロジックコンパイラー
- Advanced PLD デバイスシミュレーター
- 波形編集用 Waveform エディター

PLD 開発サイクルの第一段階は、論理記述ソースファイルを作成することです。次にソースファイルは、コンパイルされ実際のデバイスをプログラムするために使用される JEDEC ファイルが作成されます。

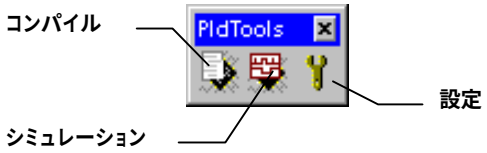
コンパイラーが処理を始めると、必要な出力ファイルやターゲットデバイスなどの情報が必要になります。このような情報は、Configure Advanced PLD ダイアログボックスで入力されます。

Advanced PLD の設定と使用法

プログラマブルロジックの設計の第一段階は、論理記述言語 CUPL で書かれた論理記述ソースファイルを作成することです。EDA クライアントには、文法チェック機能の付いた ASCII テキストエディターがあります。テキストエディターを使用して論理記述ソースファイルを作成し、PldTools ツールバーからコンパイラーとシミュレーターを起動して下さい。

PLD ツールバー

PLD ツールバーを使用してコンパイラーの設定や起動、シミュレーターの起動を行なって下さい。PldTools ツールバーには以下のボタンがあります。



Compile(コンパイル)	Advanced PLD コンパイラーを起動します。これにより、CUPL 言語の論理記述ソースファイルがコンパイルされます。
Simulate(シミュレーション)	Advanced PLD シミュレーターを起動します。このシミュレーターは、ユーザが作成したテスト仕様ソースファイルの予想されるテスト値をコンパイル中の論理式から作成される実際の値とを比較します。
Configure(設定)	Configure Advanced PLD ダイアログボックスをオープンします。このダイアログボックスでコンパイラーの動作や出力の設定を行います。

➔ テキストエディターの中でツールバーが使用できない場合、EDA クライアントのオンラインヘルプを参照して下さい。PLD のコマンドは、テキストエディターの PLD メニューからも実行できます。

Advanced PLD コンパイラーの設定

このセクションでは、コンパイラーの入力と Advanced PLD の設定方法について説明します。

コンパイラーの入力

論理記述ソースファイル(ファイル名.PLD)はコンパイラーへの入力です。このファイルには、ターゲットデバイスで実行したい論理関数を記述します。ソースファイルは、テキストエディターを使用して作成します。

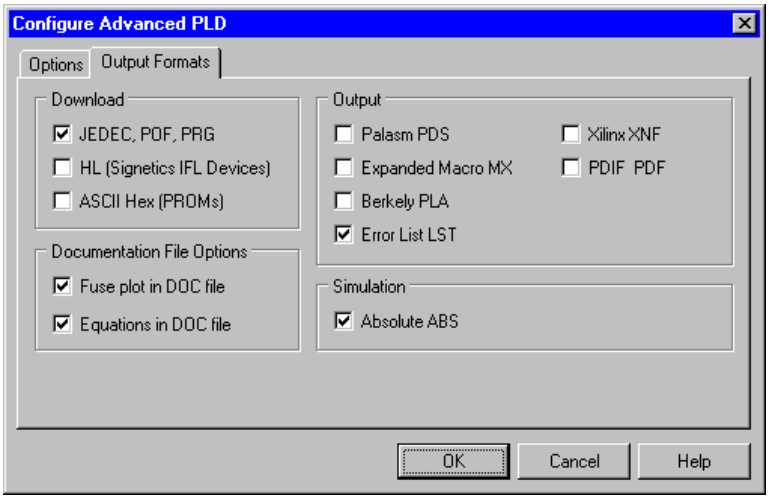
➔ 論理記述ソースファイルの作成に関する詳細は、このガイドの PLD 設計プロセスにおける PLD ソースファイルの作成のセクションを参照して下さい。

出力フォーマット

Advanced PLD は 4 種類の出力ファイルを生成することができます。その 4 種類を以下に示します。

- ダウンロードフォーマット、プログラマへダウンロードするフォーマットです。
- 出力フォーマット、サードパーティのフィッタやエラーリスティングのインタフェース用のフォーマットです。
- ドキュメンテーションフォーマット、ヒューズプロットや式があります。
- シミュレーションフォーマット、仮想デバイスのプログラムです。シミュレーターによりテスト仕様ソースファイル(ファイル名.SI)からテストベクタがこの仮想デバイスへ適用されシミュレーターリスティングファイル(ファイル名.SO)に結果が記録されます。

コンパイラー出力フォーマットを設定するには、Configure Advanced PLD ダイアログボックスの Output Formats タブのチェックボックスを選択して下さい。



Output Formats タブ

表 2.1 個々の出力フォーマットについて

出力フォーマット	説明
JEDEC, POF, PRG	JEDEC 互換の ASCII ダウンロードファイル(ファイル名.JED)を作成します。ファイル名は、コンパイラーへの論理記述入力ファイルと異なる名前でも問題ありません。論理記述ファイルのヘッダー情報セクションの NAME ステートメントで、ダウンロードファイルの名前を決定します。
HL	HL ダウンロードファイル(ファイル名.HL)を作成します。このフォ

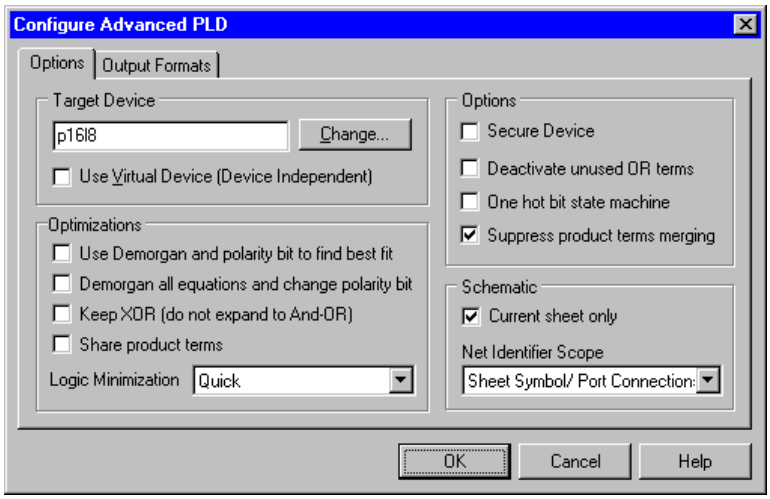
	ーマットは、Signetics IFL デバイスで使用されます。ファイル名は、コンパイラーへの論理記述入力ファイルと異なる名前でも問題ありません。論理記述ファイルのヘッダー情報セクションの NAME ステートメントで、ダウンロードファイルの名前を決定します。
ASCII Hex	ASCII-Hex のダウンロードファイル(ファイル名.HEX)を作成します。このフォーマットは、PROMs で使用されます。ファイル名は、コンパイラーへの論理記述入力ファイルと異なる名前でも問題ありません。論理記述ファイルのヘッダー情報セクションの NAME ステートメントで、ダウンロードファイルの名前を決定します。
PALASM PDS	PAL ハンドブック(第 3 版)の Monolithic Memories による標準設定で、PALASM フォーマットファイル(ファイル名.PDS)を作成します。
Expanded Macro MX	ソースファイルで使用されるマクロがすべて記述されている拡張マクロ定義ファイル(ファイル名.MX)を作成します。REPEAT コマンドを使用する拡張表現も含まれます。
Berkely PLA	PLEASURE やその他の Berkely PLA フォーマットを使用する PLA レイアウトツールなどの Berkely PLA ツールにより使用される Berkely PLA ファイル(ファイル名.PLA)を作成します。
Error list LST	エラーリスティングファイル(ファイル名.LST)を作成します。元のソースファイルの各行には番号が付けられます。エラーメッセージはファイルの最後に一覧表示され、参照のために行番号が使用されます。
QDIF	QuickLogic デバイス用の QDIF フォーマットファイルです。QuickLogic オプティマイザに対応します。QuickLogic の sPDE ツールを使用して最適化された回路の配置や配線を行います。
XNF Xilinx	他の論理設計ツールや XILINX の PDS2XNF などのゲートアレイフィッタ用の入力ファイルを作成します。
PDIF PDF	PDIFIN プログラムにより、PC-CAPS(P-CAD Schematic Capture)プログラムのシンボルに変換される PDIF(P-CAD Database Interchange Format)ファイル(ファイル名.PDF)を作成します。作成されたシンボルには、PLD のパッケージング情報が含まれます。
EDIF	EDIF フォーマットファイルを作成します。
Equations in DOC File	Sum-of-products フォーマットの論理タームの拡張リストやソースファイルで使用されるすべての変数のシンボルテーブルを含むドキュメンテーションファイル(ファイル名.DOC)を作成します。プロダクトタームの総数や各出力で使用できる値が含まれます。
Fuse Plot in DOC File	ドキュメンテーションファイルの中のヒューズプロットを作成します。PAL デバイスでは、各出力ピンは一覧表示され、関連付けられたプロダクトターム列が JEDEC ヒューズ番号を先頭にして表示されます。現在あるヒューズはｘで表わされます。切れたヒューズは、-で表わされます。IFL デバイスでは、HL ダウンロードフォーマットが使用され、H、L、0、-で表わされる入力タームで JEDEC ヒューズ番号が表わされます。
Absolute ABS	Advanced PLD 論理シミュレーターで使用するアブソリュートファイル(ファイル名.ABS)を作成します。

表 2.1 コンパイラー出力フォーマット 一覧

コンパイラーの設定

Configure Advanced PLD ダイアログボックスの Option Tab により以下の設定を行なうことができます。

- ターゲットデバイスの選択
- 最適化オプションの設定
- コンパイルオプションの設定
- ロジック最小化法の設定



Compiler Options タブ

表 2.2 コンパイラーの設定について

設定項目	説明
Use DeMorgan and polarity bit	ピンやピンノード変数のプロダクトタームの使用を最適化します。DEMORGAN ステートメントがソースファイルにある場合、オーバーライドされます。
DeMorgan all equations	ピンやピンノード変数のすべてにドモルガンの定理を適用します。DEMORGAN ステートメントがソースファイルにある場合、オーバーライドされます。
Keep XOR	XOR を AND-OR の式に展開しません。デバイスに独立の設計やフィッタが XOR ゲートをサポートしているデバイスの設計に使用します。
Share Product Terms	最小化を実行中にプロダクトタームを強制的にシェアリングします。これによりグループの削減として参照されます。
Secure Device	Secure device の設定では、JEDEC ファイルにセキュリティーヒューズのコードを追加します。これにより、消去不可のデバイスが作成されます。このオプションはすべてのプログラマーにサポートされていません。
Deactivate unused OR terms	IFL デバイスでは、OR ゲート出力アレイは AND ゲートプロダクトタームで駆動されます。通常、使用されていない OR ゲートの入力は、新しいタームが追加されても良いようにプロダクトタームアレイと接続されたままになっています。しかし、このオプションを使用すると、使用されていない OR ゲートの入力は、プロダクトタームアレイから取り除かれます。その結果、入力から出力への伝達遅れが減少します。
Suppress product terms merging	IFL デバイスでは、AND ゲートからの各プロダクトタームは、

	数個の OR ゲート出力に分配されます。このオプションによりこの機能が働かなくなり、必要な時にそれぞれのプロダクトタームは、それぞれの出力 OR アレイで作成されます。その結果、入力から出力の伝達遅れが減少します。Quine-McClusky 最小化法か Expresso 最小化法が選択されると、このオプションによりそれぞれの個別の出力毎に(出力すべてに対して最小化が実行されるのと反対に)最小化が実行されます。
One-hot-bit State Machines	FPGA を設計する人向けのオプションです。このオプションを使用するとコンパイラーはステートマシンイクエーションを one-hot-bit で作成します。これにより、XILINX デバイスなどのレジスターリッチなアーキテクチャーでは明確な優位性が確認できます。ファニングが減少し、配線作業が簡単になります。そして、レジスタからレジスタへのフィードバックパスの長さの違いにより発生するタイミング問題が無くなります。このオプションをオン/オフにしてコンパイルを実行し違いを確認して下さい。現在のところ、オプションが使用されると、コンパイラーは回路中のステートマシンをすべて one-hot-bit として扱います。

表 2.2 コンパイラーの設定

ロジック最小化法

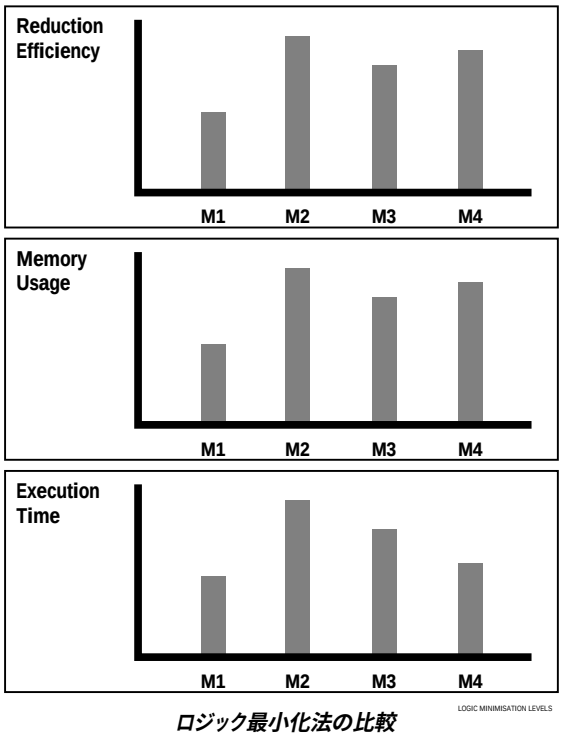
最小化法はコンパイラーにより使用されるロジックを小さくするアルゴリズムです。4 つの最小化法を使用できます。

- Quick
- Quine-McClusky
- Presto
- Expresso

Quine-McClusky 法や Presto 法は IFL デバイスでは、複数の出力の最小化を実行します。これにより、この種のデバイスではプロダクトタームのシェアリングを最大にすることができます。

最小化法を None に設定すると、コンパイル中の回路の最小化は実行されません。これは PROMs を使用する場合、その中のプロダクトタームが削除されるのを防ぐのに便利です。

以下に 4 種類の最小化のメモリ消費や速度、効率の比較を示します。



ロジック最小化法の比較

Flag	Minimization Description
M1	Quick Minimization
M2	Quine-McCluskey Minimization
M3	Presto Minimization
M4	Expresso Minimization

ロジック最小化法の比較

ブーリアン論理

以下は展開された式から不要なプロダクトタームを削除するためのブーリアン論理規則を示します。

表現		結果
!0	=	1
!1	=	0
A & 0	=	0
A & 1	=	A
A & A	=	A
A & !A	=	0
A # 0	=	A
A # 1	=	1
A # A	=	A
A # !A	=	1
A & (A # B)	=	A
A # (A & B)	=	A

表 2.3 ブーリアン論理規則

デバイスオプション

Advanced PLD コンパイラーは他のコンパイラーと同様に、ソースファイルを処理しターゲットアーキテクチャーに基づいて出力を作成します。但し、このコンパイラーは、数百種類のターゲットアーキテクチャーを持っています。これにより、ソースファイルを変更しなくても、別のアーキテクチャーで設計した論理回路を実行することができます。デバイス情報は Device Library に保存されています。

Advanced PLD にデバイスを指示する方法は 2 つあります。

- Target Device ダイアログボックスで指定
- PLD ソースファイルで指定

デバイスをコンパイラーソースファイルで指定するには

ターゲットデバイスをコンパイラーソースファイルで指定することができます。PLD ファイルのヘッダーセクションの DEVICE フィールドで指定して下さい。PLD ファイルのヘッダーセクションには数種類のフィールドがあります。各フィールドは、キーワードとそれに続く情報から構成されます。

DEVICE フィールドのシンタックスの例を以下に示します。

```
DEVICE P16L8;
```

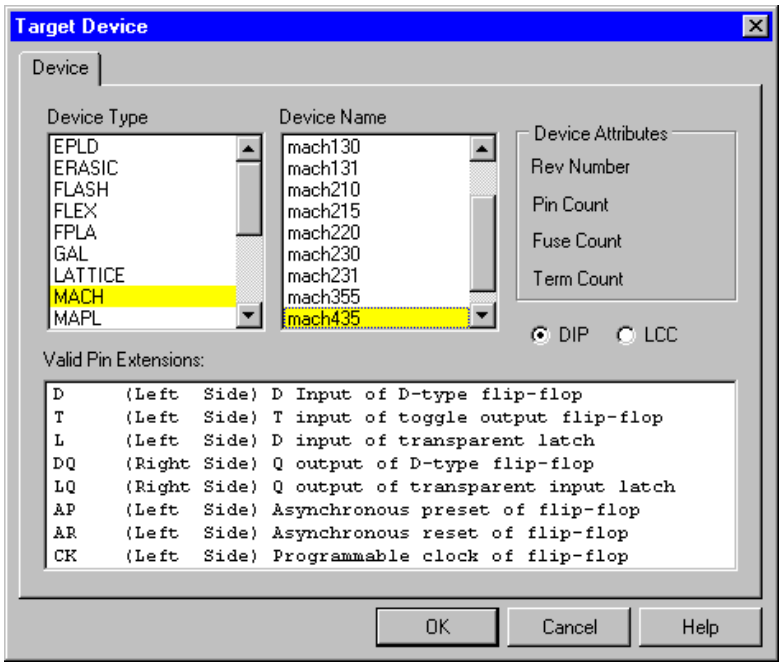
➔ ターゲットデバイスがソースファイルで指定される場合、Target Device ダイアログボックスで自動的にそのデバイスが選択されます。

Target Device ダイアログボックスでデバイスを指定する

Target Device ダイアログボックスにアクセスするには、Configure Advanced PLD ダイアログボックスの Change ボタンを押して下さい。Target Device ダイアログボックスにより Advanced PLD の現在のバージョンでサポートされているターゲットデバイスが表示されます。

ターゲットデバイスを変更するには、Device Type と Device Name を選択し、OK ボタンを押して下さい。

➔ Target Device ダイアログボックスでデバイスが指定されると、Advanced PLD は PLD ソースファイルの Device 仕様を自動的に更新します。



デバイスライブラリー

Target Device ダイアログボックスで表示される情報は、デバイスライブラリーファイル(CUPL.DL)に保存されています。このファイルには、コンパイラーによりサポートされているターゲットデバイスの記述が格納されています。ライブラリーには、各デバイスの内部アーキテクチャーやピンの数、有効な入出力ピン等の物理特性や、登録ピンと非登録ピンやプロダクトタームの数、ヒューズマップ情報、ダウンロードフォーマット情報等の論理特性が記述されています。

デバイスニーモニック(デバイス名)

Advanced PLD は、デバイスニーモニックと呼ばれるデバイスアーキテクチャーのネーミングシステムを使用します。このネーミングシステムはアーキテクチャーに対するものであり、デバイスそのものに対するものではないことに注意して下さい。例えば、AlteraEP312 と Intel5AC312 は両方とも同じアーキテクチャーです。従って、これらのデバイスのどちらかを使用して回路を設計した場合、コンパイラーにはターゲットアーキテクチャーは EP312 であると伝えられます。

➔ デバイスニーモニックは、複数の製造元を参照することができます。Devices.txt の中のニーモニック－製造元クロスリファレンスを参照して下さい。

ニーモニックはデバイスファミリープリフィックスと業界標準のパーツ番号の差フィックスで構成されます。

シンボル	意味
EP	Erasable Programmable Logic Device (EPLD)
G	Generic Array Logic (GAL)
F	Field Programmable Logic Array (FPLA)
F	Field Programmable Gate Array (FPGA)
F	Field Programmable Logic Sequencer (FPLS)
F	Field Programmable Sequence Generator (FPSG)
P	Programmable Logic Array (PAL)
P	Programmable Logic Device (PLD)
P	Programmable Electrically Erasable Logic (PEEL)
PLD	Pseudo Logical Device
RA	Bipolar Programmable Read Only Memory (PROM)

Table 2.4 デバイスニーモニックプリフィックスの一覧

例えば、PAL10L8 のデバイスニーモニックは P10L8 です。また、82S100 のデバイスニーモニックは F100 です。バイポーラ PROM ではサフィックスはアレイの大きさを表します。例えば、1024x8 バイポーラ PROM のデバイスニーモニックは RA10P8 です。すなわち、アドレスピンが 10 本とデータ出力ピンが 8 本です。

LCC と PLCC デバイス

コンパイラーが使用するデフォルトのパッケージタイプは DIP パッケージです。デバイスの中には LCC または PLCC のバージョンを持つものがあります。これらは、Surface Mount Technology または略して SMT と呼ばれることもあります。SMT デバイスのアーキテクチャーは DIP のものと同じですが、ピンアウトは異なります。そのために、Advanced PLD は、デバイスアーキテクチャーの SMT バージョンには異なるニーモニックを使用します。これには、DIP ニーモニックにサフィックスの lcc が付けられます。P16L8 の SMT バージョンは P16L8LCC です。

仮想デバイス (Virtual Device)

仮想デバイスオプションにより、ターゲットアーキテクチャーに関係なくプログラマブルロジックのデジタル設計を行なうことができます。仮想デバイスはデバイスではありません。コンパイラーにより制限が取除かれたデバイスで、プロダクトタームやピンを無制限に使用して回路を設計することができます。仮想デバイスは、設計する回路に必要なリソースを決めるのに有効です。

仮想デバイスを使用すると、コンパイラーはピンの定義の中の極性を無視します。ピン番号はそのままです。

仮想デバイスをイネーブルにするには、Configure Advanced PLD ダイアログボックスの Option Tab の Use Virtual Device チェックボックスを選択するか、論理記述ファイル(ファイル名.PLD)のヘッダーセクションのデバイスに VIRTUAL を指定して下さい。

Advanced PLD によるコンパイル

コンパイラーを起動する前に、Advanced PLD はトピックに記述されているように設定されます。Advanced PLD コンパイラーを設定して.PLD ソースファイルをアクティブドキュメントにします。

➔ PLD ファイルをコンパイルの前に保存します。

コンパイルを実行するには、PldTools ツールバーの Compile ボタンを押すか Tools-Compile メニューアイテムを選択して下さい。Advanced PLD-Compiling ダイアログボックスがポップアップ表示されコンパイルが実行されます。

コンパイラーは Configure Advanced PLD ダイアログボックスで選択された出力ファイルを作成します。View Result オプションをイネーブルにすると出力ファイルが自動的に開かれます。

Advanced PLD によるシミュレーション

このセクションでは Advanced PLD のシミュレーションの概要を説明します。

入力

テスト仕様ソースファイル(ファイル名.SI)はシミュレーターの入力ファイルです。このファイルには回路の中のデバイスに必要な機能の記述を記載します。ソースファイルはテキストエディターを使用して作成することができます。

シミュレーターソースファイルで入力される入力ピンの刺激や出力ピンのテスト値はコンパイラーソースファイルで論理式から計算された実際の値と比較されます。これらの計算

された値は、アブソリュートファイル(ファイル名.ABS)に書き込まれます。このファイルは Configure Advanced PLD ダイアログボックスで Absolute ABS オプションをイネーブルにするとコンパイル中に作成されます。

このガイドのサンプルデザインセッションのシミュレーションテストベクタの作成を参照して下さい。

➔ シミュレーションを実行する場合、コンパイラーにより作成される JEDEC ダウンロードファイルが必要です。

出力

シミュレーションの結果はシミュレーションリスティングファイル(ファイル名 SO)に書き込まれます。また、シミュレーターはテストベクタを JEDEC ダウンローダブルヒューズリンクファイルに追加します。

ヘッダー情報はすべてヘッダーエラーと一緒にリスティングファイルに表示されます。コンプリートベクタには、番号が割り付けられます。失敗した出力テストには、印が付けられ実際の(シミュレーターが決めた)出力値と一緒に表示されます。各変数は、予想される(ユーザが決めた)値と一緒に一覧表示されます。無効なテスト値は、適当なエラーメッセージと一緒に一覧表示されます。

➔ シミュレーターはコンパイラーのように複数のデバイスファイルをサポートしていません。シミュレーターは複数のデバイスファイルの最初のデバイスだけをシミュレートします。

シミュレーターの起動

シミュレーションを実行すると、.PLD ファイルはアクティブドキュメントになります。カレントディレクトリに.SI ファイルが必要です。EDA/クライアントのアクティブドキュメントとして.SI ファイルを用いてシミュレーションを実行しないで下さい。

シミュレーションを実行するには、PldTools ツールバーの Simulate ボタンを押して下さい。Advaned PLD-Simulate ダイアログボックスがポップアップ表示されシミュレーションが実行されます。

シミュレーターはシミュレーションリスティングファイル(ファイル名.SO)を作成します。各変数の入力と出力の値は、リストにされます。エラーメッセージは各ベクタに続いてエラーディスプレイの信号名と一緒に示されます。シミュレーターにより作成された.SO ファイルはシミュレーションが完了すると EDA/クライアントのカレントドキュメントになります。

View Results チェックボックスをイネーブルにすると Waveform エディタにシミュレーションリスティングファイルが自動的に表示されます。

Waveform エディター

Waveform エディターを使用してシミュレーション結果を波形として表示することができます。シミュレーションリスティングファイルには、シミュレーションの結果が保存されています。このファイルは、テキストエディターや Waveform エディターで開くことができる ASCII ファイルです。

シミュレーション結果の表示

シミュレーション結果の表示は Waveform エディターで行います。以下に波形表示の方法を説明します。

- メニューから File-Open を選択して下さい。
- Open Document ダイアログボックスで Editor を Wave に設定して下さい。Wave が一覧の中に無い場合、Wave Server がインストールされていません。サーバーのインストールに関する説明についてはインストレーションセクションを参照して下さい。
- ファイルの種類を PLD Simulation files (*.SO)に設定して下さい。
- シミュレーションリスティングファイルを選択して"開く"ボタンをクリックしファイルを開いて下さい。

シミュレーション結果が一連の波形としてスプレッドシート形式の波形ウィンドウに表示されます。

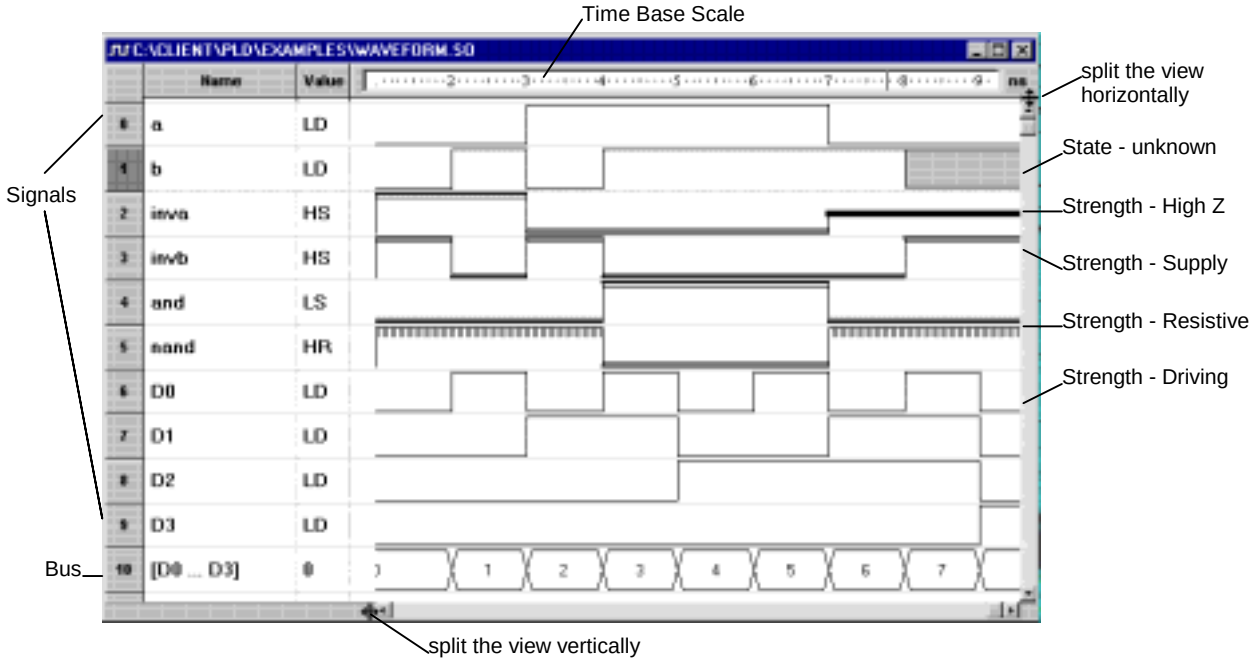


図 2.1 Waveform エディターのウィンドウ

Time Base

Time Base は波形ウィンドウの一番上のスケールに表示されます。デフォルトでは、シミュレーションリスティングファイルのイベントの変化の波形では、1ns の時間になります。

仮想マーカが Time Base Scale に表示され、現在のカーソル位置を示します。正確な時間は Status Line の左端に表示されます。

Signal Name

信号名はシミュレーションリスティングファイルから抽出されます。名前を編集したりフォントを変更したりできます。編集するには名前をダブルクリックして下さい。

Signal Value

Value 列は現在の時間位置での各信号の状態を表します。現在の時間位置は、Time Base Scale の左端です。波形をスクロールすると Value 列の内容が変化することが分かります。

それぞれの値には 2 つの特性があります。以下にその意味を示します。

L　信号の状態は Low
H　信号の状態は High
D　信号の強さは Driving
S　信号の強さは Supply
R　信号の強さは Resistive
xx　バスの現在の値を示すバス信号(2 進数または、10 進数、16 進数)

信号はなぜ違って見えるか?

信号はその強さに応じて違って表わされます。それぞれの強さがどのように表現されるかについて図 3-7 を参照して下さい。View-Legend メニューアイテムを選択して各信号の強さの例を確認して下さい。

表示される信号の強さはシミュレーションリスティングファイルの信号タイプで決まります。

タイミングマーク

Waveform エディターにはタイミングマーク 10 個あり任意の位置に置くことができます。Edit-Set Timing Marks サブメニューを選択しタイミングマークを適当な位置に設定して下さい。Edit-Jump メニューを使用して、設定したタイミングマークへジャンプできます。(ショートカット: J に続いてタイミングマーク番号)

バスの作成

近接する信号の任意の組みからバスを作成することができます。バスを作成する方法を説明します。

- バスに組み込む信号をクリック＆ドラッグして選択して下さい。(それらの信号は黒でハイライト表示されます。)
- Insert-Bus コマンドを選択して下さい。

バスは、選択した信号の最後に挿入されます。Value 列に現在の時間位置のバスの値が表示されます。バスに含まれる各信号はバスの 1 ビットを表し、最上位に選択された信号が最下位ビットを表します。

信号の編集

- トランジションをダブルクリックして編集して下さい。
- 信号のトランジションをクリック＆ドラッグすると、開始時間をグラフィック画面上で変更できます。

表示位置の変更

Waveform エディターには、波形を検証するためのいろいろな機能があります。

垂直方向のスクロールと水平方向のスクロール

スクロールバーを使用して波形ウィンドウを後ろや前、上、下へスクロールして下さい。矢印キーを使用してもできます。大きくスクロールアップやスクロールダウンしたい場合、PAGE UP や PAGE DOWN キーを使用して下さい。

ズームインとズームアウト

View-Zoom In(ショートカット;+)または View-Zoom Out(ショートカット;-)メニューアイテムを選択して、Time Base Scale を変更して下さい。Zoom In や Zoom Out プロセスは現在のカーソル位置で行われます。ショートカットキーを押す前のカーソル位置です。

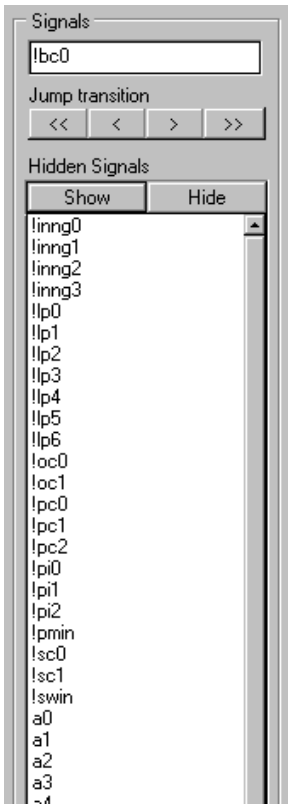
パンニング

HOME キーを押すと波形の表示を左右にふることができます。このショートカットキーは、現在のカーソル位置で行われます。ショートカットキーを押す前のカーソル位置です。

マルチ表示にする方法

Waveform エディターはマルチ表示をサポートしており、波形の別の位置を同時に表示することができます。表示は縦に分割(違う時間位置を同時に表示する)したり横に分割(同時に見る事ができない別の波形を同時に表示する)したりできます。表示を分割して使用した時のカーソルの位置は図 3-7 を参照して下さい。図のようにカーソルを置いて、クリック＆ドラッグすると画面が分割されます。

Waveform エディターパネル



View-Panel メニューアイテムを選択して Waveform エディターパネルの表示と非表示を切り替えてください。Panel を使用してアクティブ信号のトランジションを進めたり信号の表示/非表示を設定して下さい。

アクティブ信号

パネルの一番上のボックスは、アクティブ信号の名前を表示します。Jump Transition ボタンにより(アクティブ信号の)ジャンプできる位置を以下に示します。

最初のトランジション(ショートカット;J,F)
前のトランジション(ショートカット;J,P)
次のトランジション(ショートカット;J,N)
最後のトランジション(ショートカット;J,L)

信号の表示と非表示

Waveform エディターにより、見たくない信号のサブセットを表示しないようにできます。クリック＆ドラッグにより表示したくない信号を選択し、パネル上の Hide ボタンを押して下さい。

➔ メインツールバーの Select ボタンを使用して信号をすべて選択し、Hide ボタンを押してそれらを表示しないようにして下さい。そして、見たい信号を選択して表示して下さい。パネルの選択を表示するようにするには、Shift と Ctrl キーを押しながら信号名をクリックして下さい。

Advanced PLD による設計作業の基礎知識

ステップ

設計作業の検討

設計作業を注意深く見てみると、ステートマシンやブーリアン式、真理値表が設計に使用されます。どのタイプのシンタクスが設計作業に一番合うかを決めて下さい。

コンパイラーソースファイルの作成

与えられたテンプレートファイルを使用し、不要なセクションを取除いて下さい。作成される新しいファイルに影響するヘッダーを編集することを忘れないようにして下さい。

式の創出

式は、CUPL 言語で必要な論理を記述して下さい。これによりブーリアンやステートマシン、真理値表を記述することができます。

ターゲットデバイスの選択

入力ピンの数が充分であることを確認して下さい。

レジスタードまたはノンレジスタードの出力ピンの数が設計に対して充分あるか確認して下さい。

必要に応じて、デバイスが three-state 出力コントロールができるかを確認して下さい。

デバイスが必要なプロダクトタームの数を適当に操作できるかを確認して下さい。

➔ Configure Advanced PLD ダイアログボックスでターゲットデバイスを選択して下さい。

ピンの割り付け

設計の入力と出力をデバイスのピンに割り付けて下さい。製造元の取り扱い説明書に従ってデバイスが適切に使用されることを確認して下さい。

コンパイルの準備

ダウンロードやシミュレーションにどのファイルフォーマットが必要かを決めて下さい。4 種類の最小化法を選択できます。設計した論理回路をコンパイルする準備が完了しました。

コンパイラーソースファイルの作成

コンパイラーソースファイルは、CUPL ハードウェア記述言語で PLD の設計を記述した論理記述ファイルです。PLD 論理記述ファイルには、設計をコンパイルしたりデバイスプログラマに適当なダウンロードファイルを作成するコンパイラーへの入力が保存されています。

このセクションでは、PLD ソースファイルを CUPL で作成する方法について説明します。

CUPL ソースファイルシンタクスの概要

コンパイラーソースファイルの作成することは、テンプレートと呼ばれる一般的な輪郭を作成することに付随します。テンプレートファイル(TMPL.PLD)は、ガイドに与えられます。ヘッダー情報は必ずソースファイルの始めにあり、それに続いてピン/ノードの定義を記述します。テンプレートのその他のエリアは、必要に応じて使用され、順番は特に決められていません。以下のセクションでは、このソースファイルのフォーマットについて説明します。

ヘッダー情報

ヘッダーには、設計に関する基本的な情報を記述します。記述する情報の種類には、ファイル名やデバイスのパーツナンバー、開始日、設計の改訂番号、会社名、設計者名、デバイスのアッセンブリ、アッセンブリでのデバイスの場所があります。また、ヘッダーは、設計に使用されるデバイスを指定したりコンパイラーで作成される出力の型を指定できます。ヘッダー情報の記述順序は任意です。また、情報は name フィールド以外はすべて任意です。ヘッダーアイテムが無い場合、ワーニングメッセージが表示されますがコンパイルは実行されます。

```
Name         XXXXXX;
Partno       XXXXXX;
Date         XX/XX/XX;
Revision     XX;
Designer     XXXXXX;
Company      XXXXXX;
Assembly     XXXXXX;
Location     XXXXXX;
Format       XXXXXX;
Device       XXXXXX;
```

ヘッダー情報のセクション

タイトルブロック

タイトルブロックは、設計者に与えられたコメントエリアでタイトルを記入したり設計についての記述をしたりします。実際のデバイス名や製造元をここに書くこともできます。

注意: Advanced PLD では、PLD ソースファイルのコメント等に、日本語(2 バイト文字)を使用することはできません。

```
/*****************************************************************/
/*                                                               */
/*                                                               */
/*                                                               */
/*****************************************************************/
/*  Allowable Target Device Types:                               */
/*****************************************************************/
```

タイトルブロックセクション

ピン/ノードの定義

ピン/ノードセクションには、セクションのコメントラベルを記述したり、設計者が自分の設計のための番号やラベルを記述するピン宣言ステートメントを記述します。ピン宣言の順番は任意です。ただし、使用に応じてピンをグループ分けしたときに理解しやすいようにして下さい。空のコメントは各ピン宣言が終わった後、設計の中でのピンを役目をよく表した表現が与えられます。

```
/**  Inputs  **/
Pin          =           ;      /*                              */
Pin          =           ;      /*                              */
Pin          =           ;      /*                              */

/**  Outputs  **/
Pin          =           ;      /*                              */
Pin          =           ;      /*                              */
Pin          =           ;      /*                              */

Pinnode      =           ;      /*                              */
Pinnode      =           ;      /*                              */
```

ピン/ノード宣言セクション

中間変数

中間変数は、ソースファイルのピン宣言セクションでは定義されません。中間変数名は、論理式や追加の中間変数を生成する他の表現中で使用されます。このようなトップダウン形式で論理式を記述することで論理記述ファイルは読みやすく、そして理解しやすくなります。

```
/** Declarations and Intermediate Variable Definitions **/
```

中間変数セクション

論理式

このセクションは手紙の本文のようなところです。ステートマシンや真理値表、ブーリアン式などの設計の重要な部分はすべてここに記述されます。

```
/**  Logic Equations  **/
```

論理式セクション

PLD ソースファイルのコンパイル

このセクションでは、コンパイラーがソースファイルに実行することやコンパイラーが作成する出力ファイルのタイプを簡単に説明します。

コンパイラーについて

コンパイラーは、命令の記述されたテキストファイルを処理し、ハードウェア用のプリミティブな情報が含まれたファイルを作成します。この情報は、論理関数をプログラマブルロジックデバイスにプログラムするデバイスプログラマー、または設計した回路のシミュレーションを実行するシミュレーターにより使用されます。

コンパイラーへの入力

ソースファイルは、コンパイラーディレクティブやコマンド、コメントで構成されます。コンパイラーディレクティブは、コンパイル中に実行されるコンパイラーへの命令です。コマンドは、設計そのものを構成する高級命令です。コメントは、設計情報のためだけに設計に書き込まれる注釈で出力に影響しません。

コンパイラーからの出力

コンパイラーは、3 種類の標準的なダウンロードフォーマット;JEDEC、ASCII Hex、HL を作成することができます。JEDEC 出力は、ダウンローディングヒューズマップ用のデバイスプログラマーに必要なフォーマットです。ASCII hex ファイルは PROM デバイスヒューズマップ用のプログラマーに必要なフォーマットです。HL フォーマットは Signetics シーケンサデバイスが選択された場合の通常の出力フォーマットです。

コンパイラーオプション

コンパイラーオプションは、ソースファイルに組み込まれないコンパイラー命令です。ただし、この命令は、Configure Advanced PLD ダイアログボックスからコンパイラーに与えられます。ダイアログボックスの命令は、ソースファイルにあるコンパイラー命令のすべてをオーバーライドします。また、これらの命令により、コンパイラーに作成する出力ファイルを支持します。

ソースファイルディレクティブ

ほとんどのソースファイルのディレクティブはダラーマーク($)で始まります。これらのディレクティブは通常プレプロセッサ命令として参照されます。使用される場合、それらの命令は、ソースファイルのある列から始まります。大文字と小文字の任意の組み合わせを

使用してこれらのコマンドを入力することができます。ソースファイルディレクティブのすべてはソースファイルのどの位置に記述してもかまいません。

シンボル定数の定義と定義の解除

シンボル定数を定義するには$define コマンドを使用します。このコマンドは、チャラクタ文字列を他の指定された演算子や数字、シンボルに置き換えます。置き換えは、コンパイラーにより処理される前に入力ファイルに行われます。シンボル定数の定義を解除するには$undef コマンドを使用して下さい。このコマンドにより$define コマンドはキャンセルされます。

他のファイルのインクルード

他のファイルをソースファイルにインクルードするには$include コマンドを使用して下さい。ファイルをインクルードすることでよく使う CUPL コードの部分を標準化することができます。また、多くのソースファイルで使用する定数の定義を別のファイルに分離することができます。インクルードされるファイルの中にも$include コマンドを使用でき、インクルードファイルをネストすることができます。

条件付きコンパイル

条件付きコンパイルにより、シンボル定数が存在するか否かに応じてソースコードのセクションをコンパイルすることができます。これらのコマンドは$ifdef や$ifndef、$else、$endif です。シンボル定数が定義された場合、$ifdef コマンドに続くソースステートメントは、$else または$endif コマンドが現れるまでコンパイルされます。定義がまだされていない場合、$ifdef コマンドに続くソースステートメントは無視されます。$ifndef コマンド$ifdef コマンドと反対の様式で動作します。$ifdef や$ifndef コマンドの条件が False の場合、$else や$endif コマンドの間のソースステートメントがコンパイルされます。その他の場合は無視されます。$endif コマンドは$ifdef や$ifndef コマンドから始まった条件コンパイルの終わりを表すために使用されます。条件コンパイルはネストすることができます。$ifdef や$ifndef コマンドのネストの各レベルはそれぞれ$endif で終わる必要があります。

```
$DEFINE Prototype X   /* define Prototype*/
$IFDEF Prototype
pin 1 = memreq;       /* memory request on pin 1 of prototype*/
pin 2 = ioreq;        /* I/O request on pin 2 of prototype*/
$ELSE
pin 1 = ioreq;        /* I/O request on pin 1 of PCB*/
pin 2 = memreq;       /* memory request on pin 2 of PCB*/
$ENDIF
```

条件付きコンパイルの使用例

繰り返し命令

一連の CUPL 言語ステートメントを繰り返すには、$repeat と$repend コマンドを使用して下さい。このコマンドはＣ言語の FOR 命令や Fortran 言語の DO 命令に似ています。$repeat と$repend の間のステートメントは、$repeat ステートメントが許す回数だけ繰り返されます。これにより、カウンタに基づくステートマシンの定義が簡単にできます。

プリプロセッサ後の CUPL ソースコードの結果

```
FIELD sel = [in2..0]  FIELD sel = [in2..0];
$REPEAT i = [0..7]  !out0 = sel:'h'0 & enable;
         !out{i} = sel:'h'{i} & enable;
         !out1 = sel:'h'1 & enable;
$REPEND  !out2 = sel:'h'2 & enable;
         !out3 = sel:'h'3 & enable;
         !out4 = sel:'h'4 & enable;
         !out5 = sel:'h'5 & enable;
```

```
        !out6 = sel:'h'6 & enable;
        !out7 = sel:'h'7 & enable;
```

$repeat と$repend コマンドの使用例

マクロの使用

マクロはユーザ定義のコマンドで一連のコマンドを一語で置き換えることができます。マクロは$macro と$mend コマンドを使って使用します。$macro と$mend コマンドの間のステートメントは、マクロ名が呼ばれるまでコンパイルされません。マクロはソースファイルにマクロ名を記述することで呼び出され、パラメータをマクロに渡します。マクロはデコーダやカウンタなどのライブラリーを作成するために使用されます。マクロ展開ファイル(.MX)を作成できるようになるとプリプロセッサがマクロ定義をどのように処理するかがわかります。

```
$MACRO decoder bits  MY_X  MY_Y MY_enable;
   FIELD select = [MY_Y{bits-1}..0];
   $REPEAT i = [0..{2**(bits-1)}]
      !MY_X{i} = select:'h'{i} & MY_enable;
   $REPEND
$MEND
_/* Other CUPL statements */
decoder(3, out, in, enable);  /*macro function call*/
```

$macro コマンドと$mend コマンドの使用例

出力の最小化

MIN 命令を使用するとピン毎に出力を最小化することができます。MIN 命令の最小化レベルは 0 から 4 までで、それらは Configure Advanced PLD ダイアログボックスのオプションに対応します。この命令により同じ回路内の異なる出力に異なる最小化レベルを指定することができます。

表 3.1 にステートメント番号と Configure Advanced PLD ダイアログボックスでそれらが対応するラベルを示します。

番号	対応ラベル
0	None
1	Quick
2	Quine-McClusky
3	Presto
4	Expresso

表 3.1 ステートメント番号とダイアログボックスでのラベル

```
MIN async_out     = 0;    /* 最小化しない       */
MIN [outa, outb]  = 2;    /* 最小化レベル 2     */
MIN count.d       = 4;    /* 最小化レベル 4     */
```

個別の最小化の例

ヒューズの断線技術

デバイスの中には、デバイスの特性を変えることができるヒューズを持ったものがあります。現在、これらのヒューズは MISER ビットや TURBO ビットとして参照されます。デバイスの特性に応じて、これらのヒューズを断線したり断線しなかったりして必要なデバイス特性を実現します。このような動作は FUSE ステートメントにより行ないます。ヒューズ番号と断線値を記述するとコンパイラーにより指定された値にフューズが設定されます。このステートメントは、正しく使用しないと予測できない結果を招きますので充分注意して使用する必要があります。

```
FUSE(101,1);    /* Turbo ビットを断線します          */
FUSE(102,0);    /* Miser ビットを断線しません        */
```

FUSE ステートメントの使用例

CUPL 言語による設計

回路を設計する場合、トップ-ダウンアプローチを使用して設計されます。トップ－ダウン設計は最初設計の全体定義から始めてメインの各要素の定義プロセスなどを繰り返し、最終的にプロジェクト全体を定義する方法です。CUPL はこの種の設計を実行するのに都合の良い多くの機能を持っています。このセクションでは CUPL が持つ設計実行するための各種の命令について説明します。

言語要素

このセクションでは、CUPL 論理記述言語に含まれる要素について説明します。

ピン/ノードの定義

ピンの定義はソースファイルの始めで宣言する必要があるので、通常それらの定義から設計作業が開始されます。埋もれたレジスタを定義するために使用するノードやピンノードはソースファイルの最初で定義する必要があります。使用するデバイスがすでに決まっている場合、ピン割り付けが必要です。しかし、仮想設計を行なう場合、設定する必要があるのは後でピンに割り付けられる変数名だけです。通常ピン番号が入るエリアは空白のままでもかまいません。

中間変数の定義

中間変数は式に割り付けられる変数です。ただし、ピンやノードには割り付けられません。これらの変数は多くの変数により使用される式を定義するために使用されたり、設計をわかりやすくするために使用されます。

インデックス付き変数の使用

0 から 31 までの十進数で終わる変数名はインデックス付き変数として参照されます。この変数はアドレスラインやデータライン、その他一連の番号が付けられたアイテムのグループを表わすために使用されます。インデックス付き変数がビットフィールドで使用されると、インデックス番号 0 が付けられた変数は常に最下位ビットを表わします。

基数の使用

コンパイラーで数値を含むすべての操作は 32 ビットの精度で実行されます。したがって、数値は 0 から 2^{32}-1 の値を持ちます。数値は 4 種類の基数で表わすことができます。すなわち、2 進数、8 進数、10 進数、16 進数です。デフォルトの基数は 16 です。ただし、デバイスのピン番号やインデックス付き変数には 10 進数が使用されます。2 進数や 8 進数、16 進数の数値は、数値と混合される dont care(X)バリューを持ちます。

数値	基数	10 進数値
‘b’0	2 進数	0
‘B’1101	2 進数	13
‘O’663	8 進数	435
‘D’92	10 進数	92
‘h’BA	16 進数	186
‘O’[300..477]	8 進数(範囲)	192..314
‘H’7FXX	16 進数(範囲)	32512..32767

表 3.2 基数の使用法

リスト表記の使用

リストは変数のグループを定義する簡単な方法です。ピンやピンノード宣言、ビットフィールドの宣言、論理式、演算の組みで使用する場合に通常使用されます。かぎ括弧を使用してリストを区切ります。

ビットフィールドの使用

ビットフィールド宣言はビットのグループに変数名を割り付けるために使用されます。FIELD キーワードを使用してビットフィールドの割り付けを行うと、その名前は、表現の中で使用されます。すなわち、演算はグループの各ビットに適用されます。FIELD ステートメントが使用されると、コンパイラーは内部に 1 つの 32 ビットフィールドを作成します。これを使用して、ビットフィールドに変数を表現します。各ビットはビットフィールドの 1 つを表わします。ビットフィールドを表すビット番号は、インデックス付き変数が使用されている場合、インデックス番号と同じです。すなわち、A0 はビットフィールドのビット

0 を常に使用します。これは主に、アドレスやデータバスを定義したり扱うために使われます。

```
FIELD ADDRESS = [A7, A6, A5, A4, A3, A2, A1, A0];
   or
FIELD ADDRESS = [A7..0];

FIELD Mode = [Up, Down, Hold];
```

FIELD ***ステートメントの使用例***

言語シンタックス

このセクションでは、CUPL 言語を使用して設計を行なうために必要な論理演算子や算術演算子、算術関数について説明します。

論理演算子

NOT、AND、OR、XOP の 4 種類の基本的な論理演算子を使用することができます。以下の表は、演算子と優先順位の一覧です。優先順位が高いほうから低いほうに表示されています。

演算子	例	説明	優先順位
!	!A	NOT	1
&	A & B	AND	2
#	A # B	OR	3
$	A $ B	XOR	4

表 3.3 論理演算子とそれらの優先順位

算術演算子と算術関数

$repeat や$macro コマンドで使用できる基本的な演算子が 6 種類使用できます。以下の表はこれらの演算子と優先順位の一覧です。優先順位が高いほうから低いほうに表示されています。

演算子	例	説明	優先順位
**	2**3	べき乗	1
*	2*i	掛け算	2
/	4/2	割り算	2
%	9%8	余り	2
+	2+4	足し算	3
-	4-i	引き算	3

表 3.4 算術演算子とそれらの優先順位

$repeat$macro コマンドで使用される表現には 1 つの算術関数を使用することができます。以下の表は、算術関すとその基数を示します。

関数	基数
LOG2	2 進数
LOG8	8 進数
LOG16	16 進数
LOG	10 進数

表 3.5 算術関数

LOG 関数は整数を返します。

```
LOG2(32) = 5 <==> 2**5 = 32
LOG2(33) = ceil(5.0444) = 6 <==> 2**6 = 64
```

ceil(x)は x より小さくない最小の整数を返します。

変数の拡張子

変数名に拡張子を追加しプログラマブルデバイスの中でメジャーノードに関連付けられた特定の関数を表わすことができます。これらの関数にはフリップフロップやプログラマブルトリステートなどが含まれます。コンパイラーは拡張子の使用をチェックし、指定されたデバイスで有効かどうかや他の拡張子と競合していないかを判断します。コンパイラーはこれらの拡張子を使用してデバイスのマクロセスを配置します。したがって、設計者は、マイクロセルの中のどのヒューズがなにをコントロールするかを考える必要がありません。

図は拡張子の使用例を示します。この図の回路は実際の回路ではないことに注意して下さい。拡張子を使用して回路内の異なる関数の式を記述する方法を説明しています。

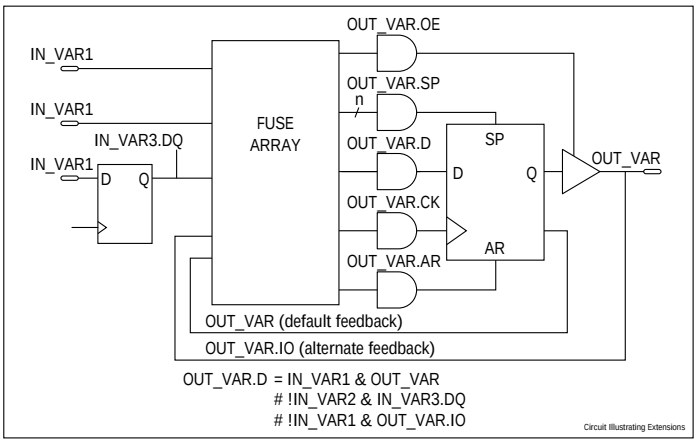


図 3.1 拡張子を図示した回路

論理式

論理式とは、CUPL 言語のビルディングブロックです。論理式の形式を以下に示します。

```
[!] var [.ext] = exp ;
```

ここで

var は変数またはリスト表記のルールに従って定義されたインデックス付きあるいはインデックスなしの変数のリストです。変数リストが使用される場合、式はリスト内の変数それぞれに適用されます。

また、.ext は省略可能な変数拡張子でこの拡張子を利用してプログラマブルデバイス内のメジャーノードに関数が割り付けられます。

exp は式です。すなわち、変数と演算子の組み合わせです。

=は割り付け演算子です。式の値を変数または変数の組みに割り付けます。

!は補足演算子です。

標準の論理式では、通常ひとつの式が変数に割り付けられます。APPEND 命令により複数の式を 1 つの変数に割り付けることができます。APPEND された論理式はその変数の元の式に論理的に OR されます。APPEND 命令の書式は、論理式の定義とは独立です。ただし、論理式が始まる前にキーワード APPEND が現れたときは別です。

テンプレートファイルで与えられるソースファイルの論理式セクションに論理式を記述して下さい。

セット演算の使用

入力ピンや 1 つのレジスタ、出力ピンなどの 1 ビットの情報を扱う演算子はすべて、グループになった複数のビットに適用することができます。セット演算は、ある組みと変数または式、あるいは 2 つの組みの間で実行することができます。

ある組みと 1 つの変数(または式)の間の演算の結果は、演算が組みの各要素と変数(または式)の間で行われ新しい組みになります。

演算が 2 つの組みの間で実行される場合、その組みは同じ大きさである必要があります(すなわち、要素の数が同じ)。2 つの組みの間の演算結果は、各組みの要素の間で実行され新しい組みになります。

数値がセット演算で使用される場合、それらは 2 進数値の組みとして扱われます。8 進数の数値は 3 つの 2 進数値の組みとして扱われます。また、10 進数や 16 進数の数値は 4 つの 2 進数値の組みとして扱われます。

等価演算

他のセット演算と違い、等価演算はひとつのブーリアン式を評価します。変数の組みと定数とをビット単位でチェックします。 定数のビット位置は、その組みの対応する位置の値と比較されます。ビット位置が 2 進数の 1 の場合、組みの要素は変更されません。ビット位置が 2 進数の 0 の場合、組みの要素は、否定されます。ビット位置が 2 進数の X の場合、組みの要素は削除されます。演算後の各要素は互いに AND され 1 つの式を作ります。

等価演算は、変数セットの識別子としても使用することができます。

```
[A3,A2,A1,A0]:&
[B3,B2,B1,B0]:#
[C3,C2,C1,C0]:$
```

上記の式は下記の式と同等です。

```
A3 & A2 & A1 & A0
B3 # B2 # B1 # B0
C3 $ C2 $ C1 $ C0
```

レンジ演算

レンジ演算は、等価演算と似ていますただし、定数フィールドが 1 つの値ではなくて値の範囲になっています。ビットの比較は範囲内の各定数値により行われます。

最初に、アドレスバスを以下のように定義して下さい。

```
FIELD address = [A3..A0]
```

次に RANGE 式を記述して下さい。

```
select = address:[C..F] ;
```

これは以下の式と等価です。

```
select = address:C # address:D # address:E # address:F;
```

真理値表

論理表現を明確に表現するために情報のテーブルを使用することがあります。CUPL は TABLE キーワードにより情報テーブルを作成することができます。まず、関連した入力と出力の変数リストを定義します。次に、入力と出力の変数リストの値の間を一対一で割り付けて下さい。入力値として dont-care 値を使用できます。ただし、出力には使用できません。

入力値のリストには、1 つの命令で複数の割り付けを行なうことができます。以下のブロックは簡単な hex-to-BCD コードコンバータを表わします。

```
FIELD  input = [in3..0] ;
FIELD  output = [out4..0] ;
TABLE  input => output {
   0=>00;   1=>01;   2=>02;   3=>03;
   4=>04;   5=>05;   6=>06;   7=>07;
   8=>08;   9=>09;   A=>10;   B=>11;
   C=>12;   D=>13;   E=>14;   F=>15;
}
```

ステートマシン

AMD/MMI によるとステートマシンは順番にあらかじめ決められた状態を繰り返すディジタルデバイスです。同期ステートマシンはフリップフロップを利用した論理回路です。出力を自分自信または他のフリップフロップの入力へフィードバックすることができるので、フリップフロップの入力値は、自分自信の出力または他のフリップフロップの出力に影響されます。従って、最終的な出力は自信の前の値と他のフリップフロップの前の値に影響されます。

以下に示される CUPL ステートマシンモデルは 6 個の部品すなわち入力と組み合わせ論理、ストレージレジスタ、ステートビット、レジスタード出力、ノンレジスタード出力を使用しています。

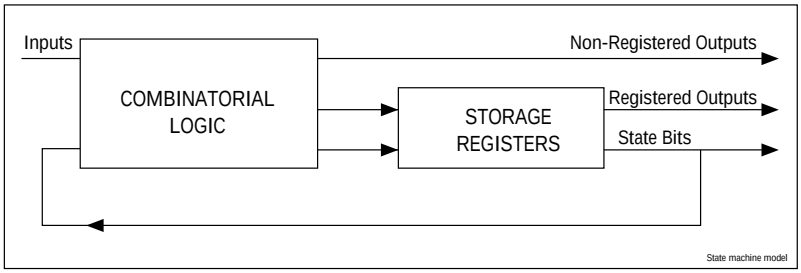


図 3.2 ステートマシンモデル

入力

他のデバイスからの信号をデバイスへ入力します。

組み合わせ論理

出力信号を作る任意の組み合わせ論理(通常 AND-OR)です。出力信号は、これらのゲートを変更して Tpd(伝播遅れ時間)nsec 後に現れます。Tpd は、入力またはフィードバックイベントの始めとノンレジスタード出力の変化の間の時間です。

ステートビット

組み合わせ論理を駆動するためにフィードバックされるストレージレジスタの出力です。これらには、現在の状態に関する情報が含まれています。

ストレージレジスタ

フリップフロップは、ステートマシン組み合わせ論理から入力を受け取ります。レジスタの中には、ステートビットに使用されるものもあります。その他はレジスタード出力に使用されます。レジスタード出力は、クロックパルスから Tco(クロックから出力までの時間)nsec 後に現れます。Tco はクロック信号の始めとフリップフロップ出力が現れるまでの間の遅れ時間です。

ステートマシンを実行するために、CUPL は、ステートマシンにどんな関数でも記述できるようなシンタクスを持っています。SEQUENCE キーワードによりステートマシンの出力が識別され、このキーワードに続いてステートマシンの関数の定義の命令が記述されます。SEQUENCE キーワードによりストレージレジスタやターゲットデバイスのデフォルトの出力タイプが設定されます。SEQUENCE キーワードの他に、 SEQUENCED キーワードや SEQUENCEJK キーワード、SEQUENCERS キーワード、SEQUENCET キーワードがあります。それぞれのキーワードは、D タイプや J-K タイプ、S-R タイプ、T タイプのフリップフロップを設定します。SEQUENCE シンタクスのフォーマットを以下に示します。

```
SEQUENCE state_var_list {
   PRESENT  state_n0
      IF  (condition1)   NEXT  state_n1;
      IF  (condition2)   NEXT  state_n2 OUT  out_n0;
      DEFAULT            NEXT  state_n0;
   PRESENT  state_n1
      NEXT  state_n2;
   .
   .
   .
   PRESENT  state_nn statements ;
}
```

ここで

state_var_list は、ステートマシンブロックで使用されるステートビット変数のリストです。変数リストは、フィールド変数により表現することができます。

state_n は、ステート数で、state_variable_list をデコードした値です。また、state_n は、各 PRESENT 命令に固有の値である必要があります。

条件命令や next 命令、出力命令はこのセクションの以下のサブセクションで説明します。

マルチステートマシン

CUPL シンタクスでは、同じ PLD 設計の中に複数のステートマシンを定義することができます。複数のステートマシンが定義されると、設計者は互いのステートマシン間で通信を行いたい場合があります。例えば、あるステートマシンがある状態になったら、他のステートマシンを起動するような回路です。ステートマシンの通信を実現するには 2 つの方法があります。すなわち、ステートビットのセット演算を使用する方法か、ステートマシン間でアクセス可能なグローバルレジスタを定義する方法です。

1 つのステートマシンに、条件命令にステート番号やステート番号レンジに続いて他のステートマシン名を記述できます。他のステートマシンが特定の状態になると、条件命令が TRUE になります。複数のステートマシンでアクセスするレジスタを使用する場合、同じことが起こると TRUE です。しかし、この方法はレジスタやデバイス出力を 1 つ消費してしまいます。状態に応じて、他のステートマシンから情報を受け取り異なる状態になるグローバルレジスタを組み合わせて使用することができます。

条件命令

CONDITION シンタクスにより、組み合わせ論理の標準ブーリアン論理式を記述して設計するよりわかりやすく論理関数を設計することができます。フォーマットを以下に示します。

```
CONDITION  {
   IF expr0 OUT  var ;
        .
        .
   IF exprn OUT  var ;
   DEFAULT  OUT  var ;
}
```

CONDITON シンタクスは、ステートマシンシンタクスの同期条件出力命令と等価です。ただし、特定の状態を参照することは行われません。式や DEFAULT 状態に適合すると、変数が論理的に宣言されます。

関数の定義

FUCTION キーワードにより、あるロジックに名前を付けてカプセルに入れて関数化し、個人的なキーワードを作成することができます。この名前は、論理式内で使用することができる関数を表わします。ユーザ定義関数のフォーマットを以下に示します。

```
FUNCTION name ([parameter0,....,parametern])
{              body               }
```

body の中の命令が、関数に割り付けられます。

省略可能なパラメータを使用する場合、関数定義または参照内のパラメータの名前は、一致する必要があります。関数の本体で定義されるパラメータは、論理式内で参照されるパラメータで置き換えられます。関数を呼び出す変数は関数本体により式に割り当てられます。命令本体で割り付けが行われない場合、関数を呼び出す変数は ho の値に割り付けられます。

簡単な論理設計の例

地下鉄の回転改札口のコントローラは最も簡単なステートマシン設計です。このコントローラは、コインが入れらた信号を待って、状態が変化すると、ロックからオープンに代わります。オープン状態では、だれかが回転改札口を通貨するのを待ちます。それから、オープンからロックに変わります。この 2 つの状態設計は、入力としてコインが入れられたことによる信号と人が通過したことによる信号とのあいだで繰り返されます。以下の図は、2 つの状態とデバイスをある状態から次の状態へ変更するパルスを示します。

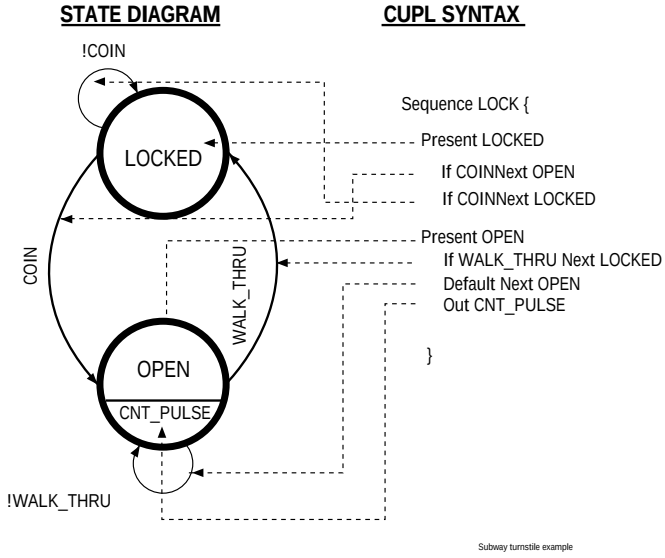


図 3.2 地下鉄の回転改札口の例

コードと設計概念の関係がわかりやすく理解できるように、CUPL ステートマシンコードに相当するものが右に表示されます。

TURNSTIL.PLD

```
Name        Turnstil;
Partno      FL00001;
Date        03/06/89;
Designer    R. Teixeira;
Company     LDI;
Location    D21;
Assembly    Example

/*****************************************************/
/*                                                   */
/* This is an example to demonstrate how CUPL        */
/* compiles a subway turnstile controller            */
/*                                                   */
/*****************************************************/
/*   Target Devices: P16R4                           */
/*****************************************************/

/* Inputs:   define inputs to Turnstile controller   */
Pin 1 = CLK;
Pin 2 = WALK_THRU;
Pin 3 = COIN;

/* Outputs:   define outputs as active HI levels     */
Pin 14 = CNT_PULSE;
```

```
Pin 15 = LOCK;

/* Logic:  Subway Turnstile example expressed in CUPL */
DEFINE LOCKED ‘b’0
DEFINE OPEN  ‘b’1

Sequence  LOCK{
   Present LOCKED
      if COIN     Next OPEN;
      if !COIN    Next LOCKED;
   Present OPEN
      if WALK_THRU Next LOCKED;
      Default      Next OPEN;
      Out CNT_PULSE;
}
```

TURNSTIL.PLD

簡単なゲートの設計例

このセクションでは、PLD の簡単なゲートプログラムの作成について詳しく説明します。以下のインプリメントされる設計を示します。

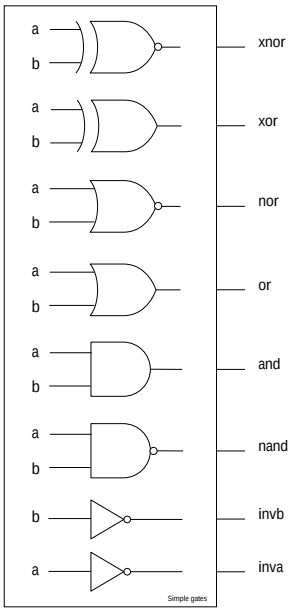


図 3.3 簡単なゲート

この設計は、簡単なゲートの出力を作成するロジックです。出力は、ゲートの関数を表わすラベルが付けられています。例えば、AND ゲートは出力に AND とラベルが付けられます。

以下に、この設計を記述する CUPL ソースファイル(Advanced PLD パッケージの中の GATES.PLD)を示します。

```
Name            Gates;
Partno          CA0001;
Date            07/16/87;
Designer        G Woolheiser;
Company         ATI;
Location        San Jose, CA.;
Assembly        Example

/*************************************************/
/*                                               */
/* This is an example to demonstrate how Advanced PLD */
/* compiles simple gates                         */
/*                                               */
/*************************************************/
/*   Target Devices: P16L8, P16P8, EP300, and 82S153  */
/*************************************************/

/* Inputs:   define inputs to build simple gates     */
Pin 1 = a;
Pin 2 = b;

/* Outputs:                                          */
/* define outputs as active HI levels For PAL16L8 and */
/* PAL16LD8, De Morgan's Theorem is applied to invert */
/* all outputs due to fixed inverting                */
```

```
/* buffer in the device.                                    */

Pin 12 = inva;
Pin 13 = invb;
Pin 14 = and;
Pin 15 = nand;
Pin 16 = or;
Pin 17 = nor;
Pin 18 = xor;
Pin 19 = xnor;

/* Logic:   examples of simple gates expressed in CUPL */

inva = !a;              /* inverters          */
invb = !b;
and  = a & b;           /* and gate           */
nand = !(a & b);        /* nand gate          */
or   = a # b;           /* or gate            */
nor = !(a # b);         /* nor gate           */
xor = a $ b;            /* exclusive or gate  */
xnor  = !(a $ b);       /* exclusive nor gate */
```

簡単なゲートのソースファイル*(GATES.PLD)*

ファイルの最初の部分は、互換性のある PLD や設計されている関数の説明などの記述です。まず、コンパイラーにより出力ファイルの名前として使用される名前の行があります。Partno は会社の固有の部品番号を表わします。部品番号はターゲット PLD のタイプではありません。Data はコンパイル時のデータを示します。データは、現在のものに常に更新されます。Designer は、設計者の名前または設計チームの名前です。Assembly はアッセンブリ名または PLD が使用される PC 基盤の名前を表わすために使用されます。必要に応じて、省略形の ASSY を使用して下さい。Location は、PLD が配置される PC 基盤上の座標を表わすために使用されます。省略形の LOC も使用することができます。

ピン宣言は設計ダイアグラムの入力と出力に対応して行われます。この例でのゲートには、ゲートに接続される 2 つの入力が必要です。a と b は入力ピンの名前です。次に出力ピンに名前が割り付けられます。付けられた名前は関数の動作を表わしています。このように関数の動作を表わす名前を付けることを推奨します。これにより、ファイルのデバッグ時や更新時に内容が解りやすくなります。

ファイルの Logic セクションでは、式によりゲートが記述されます。ブーリアンシンタクスを使用して各出力ピンが入力ピン a と b の関数の出力として指定されます。

固定の反転バッファを持つ PAL16L8 と PAL16LD8 デバイスでは、ピンリストの中でアクティブハイで宣言されているので、コンパイラーはドモルガンの定理を適用して出力をすべて反転します。例えば、アクティブハイとして宣言されている出力ピンの OR ゲートの以下のような式はコンパイラーにより反転され、

```
or = a # b ;
```

以下のような 1 つの展開されたプロダクトタームになります。(ドキュメンテーションファイルで示されるように)

```
or => !a & !b
```

前に示された今回使用されるデバイスは、デバイスを決める基準に合致するために使用されます。すなわち、入力と出力のピン数が適当であり、スリーステートコントロールを持ち、レジスタードとノンレジスタードピンやデバイスが扱うことのできるプロダクトタームの数が適当であるということです。

ソースファイルのコンパイル

このセクションでは、論理記述ファイル(GATES.DOC)がコンパイルされ、シミュレーターにより使用されるドキュメンテーションファイルやデバイスプログラマへダウンロードされるダウンロードファイルが作成されます。

GATES.PLD をコンパイルするには:

- Configure Advanced PLD ダイアログボックスで、以下のオプションをイネーブルして下さい。

Equations in Doc File	このオプションにより、コンパイラーにより作成されたリストを見ることができるように、展開された論理式や変数シンボルの表、プロダクトタームの使用状況が記述されたドキュメンテーションファイル(GATES.PLD)が作成されます。
Fuse Plot in Doc File	このオプションにより、.DOC ファイルにフューズマップ情報が追加されます。
Absolute ABS.	このオプションにより、シミュレーターにより使用される GATES.ABS ファイルが作成されます。
Jedec.	このオプションにより、デバイスプログラマへの入力ファイルやシミュレーターへの入力ファイルとして使用される GATES.JED が作成されます。

- Change ボタンを押してターゲットデバイスを指定して下さい。Target Device ダイアログボックスでは、Device Type 19 と選択されたデバイス P16L8 がハイライト表示されます。
- OK ボタンを押して、ダイアログボックスを閉じて下さい。
- .PLD ファイル(GATES.PLD)をカレントドキュメントにして下さい。
- PldTools ツールバーの Compile ボタンを押してコンパイルプロセスを開始して下さい。

ターゲットデバイスは PAL16L8 でソースファイルは GATES.PLD です。出力ファイルは、GATES.DOC、GATES.ABS、GATES.JEC です。

コンパイルが終了すると、GATES.DOC を開いて下さい。GATES.DOC には作成されたリストが記述されています。これにより、コンパイラーが論理式をどのように展開するかが解ります。

コンパイラーレポートエラーを見る方法

- GATES.PLD を編集し、割付命令のどれかの終わりに記述されているセミコロンを削除して下さい。
- ファイルを保存して下さい。
- エラーリスティングファイルを作成するには、Configure Advanced PLD ダイアログボックスの Error list LST format をイネーブルして、コンパイラーを実行して下さい。

コンパイラーが終了したら、GATES.LST ファイルを調べて下さい。エラーに続いてエラー行が挿入されていることと、各行の先頭に行番号が付けられていることに注意して下さい。

設計のシミュレーション

このセクションでは、GATES.PLD 論理設計が Advanced PLD シミュレーターによりシミュレートされます。コンパイル中に作成された GATES.JED ファイルにテストベクタが付けられます。

テスト仕様ソースファイル(ファイル名.SI)が、シミュレーターへの入力です。この例では、GATES.SI が Advanced PLD パッケージの中にあります。このファイルには、回路内のデバ

イスの関数についての記述があります。テスト仕様ファイルの作成についての詳細は、このマニュアルのシミュレーションテストベクタファイルの作成を参照して下さい。

入力の関数として出力を定義することにより、テストベクタにより予測された PLD の機能が指定されます。プログラムが完了すると、テストベクタはデバイスの機能テストにも使用されます。

シミュレーターにより、入力ピンの信号や GATES.SI ファイルに入力される出力ピンのテスト値が、CUPL ソースファイルの論理式から計算された実際の値とが比較されます。これらの計算された値は、コンパイル中に作成されるアブソリュートファイル GATES.ABS に記録されます。

シミュレーションを実行するには以下のようにして下さい。

- Gates.PLD を表示して下さい。Gates.SI が、同じフォルダに保存されている事を確認して下さい。SI ファイルを表示してシミュレーションを実行しないで下さい。
- PldTools ツールバーの Simulate ボタンを押して下さい。

シミュレーションにより作成されるテストベクタが、コンパイル中に作成される JEDEC ダウンロードファイルに自動的に追加されます。

シミュレーター入力ファイル(GATES.SI)

```
Name            Gates;
Partno          000000;
Revision        03;
Date            9/12/83;
Designer        CUPL Engineering;
Company         Protel International.;
Location        None;
Assembly        None;
/*************************************************************/
/*                                                           */
/*      This is a example to demonstrate how the Compiler    */
/*      compiles simple gates.                               */
/*                                                           */
/*************************************************************/
/*      Target Devices:  P16L8, P16LD8, P16P8, EP300, and 2S153*/
/*************************************************************/

/* Order:  define order, polarity, and output                */
/*         spacing of stimulus and response values           */

Order:  a, %2, b, %4, inva, %3, invb, %5, and, %8,
        nand, %7, or, %8, nor, %7, xor, %8, xnor;

/* Vectors: define stimulus and response values, with header    */
/*          and intermediate messages for the simulator listing */
/* Note:  Don't Care state (X) on inputs is reflected in        */
/*        outputs where appropriate.                           */
Vectors:
$msg "";
$msg "                          Simple Gates Simulation";
$msg "";
$msg "  inverters  and    nand     or     nor      xor    xnor
$msg "a b !a !b   a & b !(a & b)  a # b !(a # b)  a $ b !(a $b)";
$msg  - - ── ──   ─────  ──────   ─────  ──────   ─────  ───── ";

00 HHLHLHLH
01 HLLHHLHL
```

```
10 LHLHHLHL
11 LLHLHLLH
1X LXXXHLXX
X1 XLXXHLXX
0X HXLHXXXX
X0 XHLHXXXX
XX XXXXXXXX
```

*Gates **のシミュレーター入力ファイル**(GATES.SI)*

シミュレーター出力ファイル(GATES.SO) 入力は対応する出力と一緒に一覧表示されます。

```
 1:Name           Gates;
 2:Partno         000000;
 3:Revision       03;
 4:Date           9/12/83;
 5:Designer       CUPL Engineering;
 6:Company        Protel International;
 7:Location       None;
 8:Assembly       None;
 9:
10:/*************************************************************/
11:/*                                                           */
12:/*      This is a example to demonstrate how the Compiler    */
13:/*      compiles simple gates.                               */
14:/*                                                           */
15:/*************************************************************/
16:/* Target Devices:  P16L8, P16LD8, P16P8, EP300, and 82S153 */
17:/*************************************************************/
18:
19:
20:/*
21; * Order:  define order, polarity, and output
22: * spacing of stimulus and response values
23: */
24:
25:Order:  a, %2, b, %4, inva, %3, invb, %5, and, %8,
26:        nand, %7, or, %8, nor, %7, xor, %8, xnor;
27:
28:/*
29: * Vectors: define stimulus and response values, with header
30: *        and intermediate messages for the simulator listing.
31: *
32: * Note: Don't Care state (X) on inputs is reflected in
33: *       outputs where appropriate.
34: */
35:
=================================================================
                        Simulation Results
=================================================================

                Simple Gates Simple Simulation
           inverters  and   nand   or    nor    xor   xnor
      a  a   !a  !b   a&b  !(a&b)  a#b  !(a#b)  a$b  !(a$b)
      -  -   __  __   ___  _____   ___  ______  ___  ______
0001: 0  0    H   H    L      H     L      H     L      H
0002: 0  1    H   L    L      H     H      L     H      L
0003: 1  0    L   H    L      H     H      L     H      L
0004: 1  1    L   L    H      L     H      L     L      H
0005: 1  X    L   X    X      X     H      L     X      X
```

```
0006: X  1    X   L    X      X     H      L     X      X
0007: 0  X    H   X    L      H     X      X     X      X
0008: X  0    X   H    L      H     X      X     X      X
0009: X  X    X   X    X      X     X      X     X      X
```

Gates ***のシミュレーター出力ファイル****(GATES.SO)*

設計のサンプル

このセクションでは、コンパイラーとシミュレーションを使用する設計例を通して段階的に説明をします。

Step 1. 設計作業の確認
Step 2. コンパイラーソースファイルの作成
Step 3. 式の公式化
Step 4. ターゲットデバイスの選択
Step 5. ピン割り付け
Step 6. PLD ソースファイルのコンパイル
Step 7. シミュレーションテストベクタファイルの作成
Step 8. デバイスのシミュレーション
Step 9. シミュレーション波形の表示

Step 1 - 設計作業の確認

このプログラマブルロジックデバイス(PLD)設計例のシステムは、マイクロプロセッサをベースとした ROM と RAM を用いた CPU インターフェースを使用しています。図にシステムダイアグラムを示します。

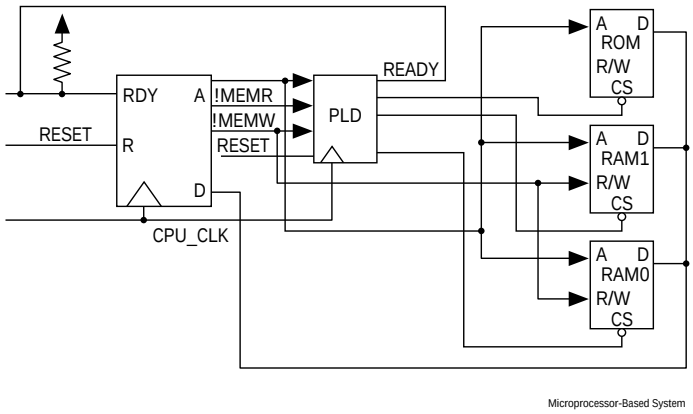


図 4.1 マイクロプロセッサをベースとしたシステム

PLD により、アドレスのデコードやタイミング制御関数による CPU と周辺装置とのフレキシブルなインターフェースを設計できます。ダイアグラムで示されるように、ROM(または PROM)を使用してシステムを制御し、2 個の静的 RAM をスクラッチパッドと補助メモリ関数に使用します。

このサンプルセクションでは、PLD の回路の目的は、メモリのマッピングに使用する CPU のアドレスをデコードしたり、CPU アドレスと CPU データストローブに基づいて ROM と RAM のチップセレクト信号を生成することです。

メモリマップは、CPU のアドレススペースのどこに ROM と RAM の領域が確保されるかを示します。

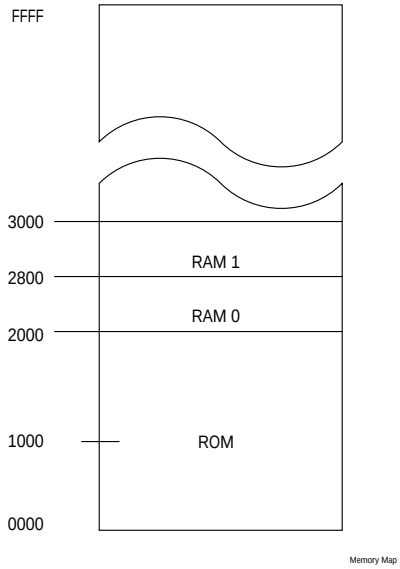


図 4.2 メモリマップ

アドレスは、メモリマップでは 16 進数で表わされます。PLD の論理回路を設計する場合、このメモリマップを使用して下さい。

ROM チップは遅いので、PLD は ROM アクセスに 1CPU クロック以上の待ち状態を生成するように設計する必要があります。

タイミングダイアグラムの矢印は他の信号により影響されたり生成される信号を示します。

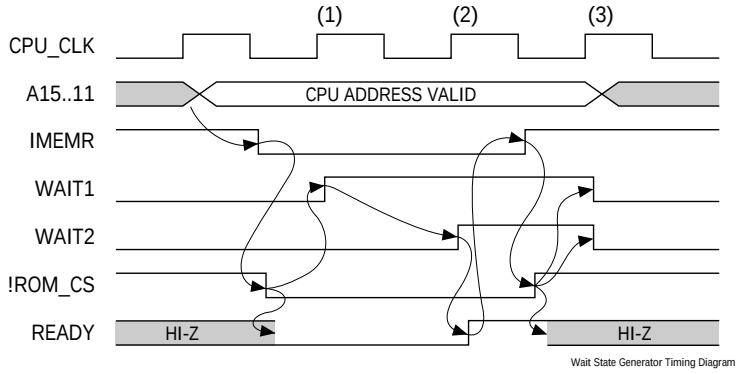


図 4.3 待ち状態ジェネレータタイミングダイアグラム

タイミングダイアグラムの演算の記述が以下に続きます。丸括弧内の番号は CLOCK 信号の立ち上がりエッヂを示します。

待ち状態シーケンスは、メモリの読み取りストローブの前の CPU アドレスが有効になった時に始まります。待ち状態は、ROM のためだけに生成されるので、!MEMR 信号だけを考慮に入れる必要があります。

!MEMR ストローブが ROM に対応したアドレスでアクティブの場合、!ROM_CS 信号が呼び出され、CPU の READY 信号を LO に駆動する(レディー状態か待ち状態かを示す)スリーステートバッファがオンされます。!ROM_CS がアクティブになった後、CPU クロックの次の立ち上がりエッヂ(1)で WAIT1 信号がセットされます。1CPU クロック後に WAIT2 信号が呼び出され(2)ます。すなわち、待ち状態期間(1CPU クロック)が完了し、CPU の READY 信号が HI に駆動されます。この信号により、CPU は読みだしサイクルを続け、!MEMR ストローブが適当な時刻で削除されます。!ROM_CS 信号は反転されレディー信号を駆動する

スリーステイトバッファーはディスエーブルになります。そして、CPU クロックの次の立ち上がりエッヂ(3)により、WAIT1 と WAIT2 がリセットされます。待ち状態ジェネレータは、次の CPU アクセス時間のために初期化されます。

Step 2 - コンパイラーソースファイルの作成

このステップでは、PLD の設計を記述するために論理記述ファイルを作成します。この論理記述ファイルは、CUPL 論理記述言語で記述されます。論理記述ファイルは、デバイスプログラマーへダウンロードする設計をコンパイルするコンパイラーへの入力です。

論理記述ファイルに必要なフォーマットを簡単に設定するために、Advanced PLD にはテンプレートファイル、TMPL.PLD があります。テキストエディターで TMPL.PLD を開き SAMPLE.PLD という名前で保存して下さい。

以下に SAMPLE.PLD にあるテンプレート情報を示します。

TEMPLATE FILE

```
Name            XXXXX;
Partno          XXXXX;
Date            XX/XX/XX;
Revision        XX;
Designer        XXXXX;
Company         XXXXX;
Assembly        XXXXX;
Location        XXXXX;

/*************************************************************/
/*                      Title Block                          */
/*************************************************************/
/*  Allowable Target Device                                  */
/*************************************************************/

/**   Inputs  **/

Pin       =       ;     /*                                   */
Pin       =       ;     /*                                   */
Pin       =       ;     /*                                   */
Pin       =       ;     /*                                   */
Pin       =       ;     /*                                   */
Pin       =       ;     /*                                   */
Pin       =       ;     /*                                   */
Pin       =       ;     /*                                   */
Pin       =       ;     /*                                   */
Pin       =       ;     /*                                   */
Pin       =       ;     /*                                   */
Pin       =       ;     /*                                   */
Pin       =       ;     /*                                   */
Pin       =       ;     /*                                   */


/**  Outputs  **/

Pin       =       ;     /*                                   */
Pin       =       ;     /*                                   */
Pin       =       ;     /*                                   */
```

```
Pin         =         ;      /*                                  */
Pin         =         ;      /*                                  */
Pin         =         ;      /*                                  */
Pin         =         ;      /*                                  */
Pin         =         ;      /*                                  */

/**   Declarations and Intermediate Variable Definitions  **/
```

特定のヘッダーやタイトル情報、入出力ピンの指定、中間変数や論理式の記述のためにファイルは編集することができます。

ヘッダーセクションでは、XXX は会社や設計される PLD を参照する特定の情報と置き換えられます。

ヘッダーセクションの下は、コメント記号(/*と*/)使用するタイトルブロックです。タイトルブロックに設計を説明する情報を記述して下さい。

以下にこの例で入力する情報を示します。

SAMPLE.PLD

```
Name         Sample;
Partno       P9000183;
Date         07/16/87;
Revision     02;
Designer     Osann;
Company      ATI;
Assembly     PC Memory;
Location     U106;

/*****************************************************************/
/* このデバイスは、8Kx8 の ROM と 2Kx8 のスタティック RAM に使用するチップ */
/* セレクト信号を生成します。また、このデバイスは、システムの READY 信号を */
/* 駆動し ROM のアクセス時に 1CPU クロック以上の待ち状態を挿入します。      */
/*****************************************************************/
```

Step 3 - 式の公式化

アドレスのデコードや待ち状態ジェネレータの特定の式の入力を簡単に行なうために、まず、中間変数の式を入力して下さい。中間変数は特定のピンを表しません。従って、中間変数はあまり意味の無い名前です。宣言や中間変数の定義のために SAMPLED.PLD ファイルに確保されたスペースに中間式を入力して下さい。

入力する最初の中間式は、アドレスバスを定義するビットフィールド宣言です。アドレスを表わすために MEMADR(メモリアドレス)という名前を使用し、以下のように式を入力して下さい。

```
FIELD MEMADR = [A15..11] ;
```

図 5-1 のシステムダイアグラムでは、スタティック RAM のチップセレクト信号はアドレスに依存しておらず MEMW または MEMR データストローブのために宣言する必要があることに注意して下さい。

スタティック RAM のチップセレクト信号のための式を簡単にするため、MEMREQ(メモリリクエスト)と呼ばれる信号を作成して下さい。以下のように入力して下さい。

```
MEMREQ = MEMW # MEMR ;
```

MEMREQ が他の式で使用される場合はいつでも、コンパイラーは MEMW#MEMR を置き換えます。

図 4-3 のタイミングダイアグラムでは、ROM に対応するアドレスのデコードがは!MEMR ストローブと組み合わされて ROM のチップセレクト(ROM_CS)を生成し、待ち状態シーケンスを初期化します。

以下のように入力して、!MEMR ストローブを組み合わせて ROM のアドレス空間の特定のアドレスをデコードする SELECT_ROM と呼ばれる中間変数を作成して下さい。

```
SELECT_ROM = MEMR & MEMADR : [0000..1FFF] ;
```

上記の中間式を入力した後、アドレスデコードや待ち状態ジェネレータの式を入力して下さい。

アレイへフィードバックされる ROM_CS 信号が、待ち状態のタイミングを初期化するために使用される場合、PLD にさらに追加遅れが生じます。クロックレートが比較的遅い(4-8MHz)ので、この例では追加遅れは問題になりません。しかし、クロックレートが早い場合、同じロジックを(中間変数 SELECT_ROM を使い)レジスタを使用した論理式に書き換えた方が効果的です。

以下のように入力し、中間変数 SELECT_ROM を使用して ROM のチップセレクト(ROM_CS)を作成して下さい。

```
ROM_CS = SELECT_ROM ;
```

2 個の RAM、RAM_CS0 と RAM_CS1 のチップセレクトは MEMREQ とアドレスマップから取得される 16 進数の範囲内にあるアドレスバスに依存します。デコードする範囲の上限と加減のアドレスを用いて CUPL のレンジ演算を使用して下さい。以下のように入力して下さい。

```
RAM_CS0 = MEMREQ & MEMADR : [2000..27FF];
RAM_CS1 = MEMREQ & MEMADR : [2800..2FFF];
```

次に、待ち状態を生成するための式を作成します。最初にタイミングダイアグラム(図 5-3)に示されるように、CPU クロックの次の立ち上がりエッヂで設定される ROM チップの選択に必要な WAIT1 と呼ばれる信号が必要です。D タイプのフリップフロップの特性から、D 入力での論理レベルはクロック後にQ出力に変換されます。この信号の式を以下のように入力して下さい。ここで WAIT1.D は PLD のフリップフロップの D 入力の信号を表します。

```
WAIT1.D = SELECT_ROM & !RESET ;
```

WAIT1.D の式で、!RESET 信号が AND され式の残りの部分で AND され RESET 信号により同期リセットが実行されることに注意して下さい。

次に、WAIT1 が設定される次のクロックエッヂで WAIT2 信号を WAIT1 に依存する WAIT2.D の式により作成して下さい。WAIT2.D は、ROM の CPU アクセスが終了すると次のクロックエッヂでリセットされる必要があります。以下のように式を入力して下さい。

```
WAIT2.D = SELECT_ROM & WAIT1 ;
```

これにより、タイミングダイアグラム(図 5-3)の信号 SELECT_ROM 信号が作成され、ROM がデコードされている間と MEMR データストローブがアクティブな間、スリーステートバッファーがオンにされることが示されます。従って、以下のように入力してスリーステート出力の式を入力して下さい。

```
READY.OE = SELECT_ROM ;
```

この式が、スリーステートバッファーがその出力を実際に駆動しハイインピーダンス状態に保持するかを決める間、どちらの論理レベルに信号が駆動されるかは決定されません。READY の式により、論理レベルをどちらに駆動するかが決められます。すなわち、1CPU クロックサイクルに相当する待ち状態期間が完了するまで READY の信号はインアクティブのままです。この状態は WAIT2 が設定されるまで起こらないように、READY の式を以下のように入力して下さい。

```
READY = WAIT2;
```

Step 4 - ターゲットデバイスの選択

式が完成すると、次のステップは、使用する PLD を決定します。ターゲットデバイスを選択するときに考慮すべき点を以下に示します。

- 必要な入力ピンの数
- レジスタード出力ピンとノンレジスタード出力ピンの数の比率。
- スリーステート出力制御(必要な場合)
- 各式の論理関数を実行するために必要なプロダクトタームの数

以下に PLD パッケージダイアグラムに、PAL16R4 または 82S155IFL と互換性のあるデバイスのピン配置を示します。

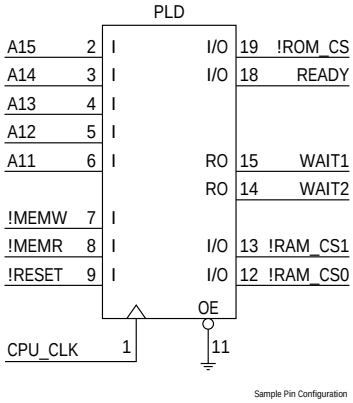


図 4.4 ピン配置のサンプル

図のピン配置では、チップセレクト信号が3本、出力駆動モードのI/Oタイプのピンに割り付けられています。CPUバスに接続されるREADYピンは、切り替え可能なスリーステートモードで使用されます。待ち状態ジェネレータを実行するために必要な2個のフリップフロップは、内部でレジスタに接続される出力ピンに割り付けられます。

READYの論理関数はWAIT2信号の論理関数と同じなので、レジスタード出力の1つを使用してREADY信号を直接駆動することができます。しかし、ターゲットデバイスのピン11に接続されるスリーステート出力イネーブル信号を使用する必要があります。ピン11は内部レジスタに接続される4本のピンすべてのスリーステート出力をコントロールするので、これにより他の2つのレジスタード出力ピンを待ち状態ジェネレータに使用することができます。

設計の発展段階での変更をすべて予測するのは困難なことなので、スリーステートコントロールを使用しないでオプションを開いたままにした方が良いです。したがって、ピン11は、レジスタからのスリーステート出力常にイネーブルになるようにグランドに接続されます。

PAL16R4は、出力に少なくとも7個のプロダクトタームがあります。この数はこのアプリケーションに適当です。このソケット位置のセカンドソースのIFL82S155は、全部で32個のプロダクトタームを使用できます。この数も今回のアプリケーションには適当です。

PAL16R4 デバイスは D タイプのフリップフロップしか持っていません。一方、82S155 デバイスは D または JK タイプのフリップフロップを使用することができます。Step3 で WAIT1 と WAIT2 に記述された式は、拡張子.D が付いているので、コンパイラーは D タイプフリップフロップの配置を自動的に決めます。

Step 5 - ピン割り付け

ピン割り付けを PAL16R4 または 82S155 デバイスのピンに合わせて下さい。まず、SAMPLE.PLD の Allowable Target Device Types とラベルが付けられたコメントスペースに次のように入力して下さい。

```
pal16r4, 82s155
```

ピン割り付けを行なう時は、割り付けられる信号の極性(信号レベル)が論理スケマチックの信号と同じであることを確認して下さい。

以下に示すようにピン割り付けを行なって下さい。

```
                    SAMPLE PIN ASSIGNMENTS
/**   Inputs  **/
Pin 1       = cpu_clk      ;      /* CPU clock                */
Pin [2..6]  = [a15..11]    ;      /* CPU Address Bus          */
Pin [7,8]   = ![memw,memr] ;      /* Memory Data Strobes      */
Pin 9       = reset        ;      /* System Reset             */
Pin 11      = !oe          ;      /* Output Enable            */

/**  Outputs  **/
Pin 19      = !rom_cs       ;     /* ROM Chip select          */
Pin 18      = ready         ;     /* CPU Ready signal         */
Pin 15      = wait1         ;     /* Start wait state         */
Pin 14      = wait2         ;     /* End wait state           */
Pin [13,12] = ![ram_cs1..0] ;     /* RAM Chip selects         */
```

以下に完成した論理記述ファイル、SAMPLE.PLD を示します。

SAMPLE.PLD

```
Name          Sample;
Partno        P9000183;
Date          07/16/87;
Revision      02;
Designer      Osann;
Company       ATI;
Assembly      PC Memory;
Location      U106;

/*****************************************************************/
/* このデバイスは、8Kx8 の ROM と 2Kx8 のスタティック RAM に使用するチップ */
/* セレクト信号を生成します。また、このデバイスは、システムの READY 信号を */
/* 駆動し ROM のアクセス時に 1CPU クロック以上の待ち状態を挿入します。      */
/*****************************************************************/
/** Allowable Target Device Types : PAL16R4, 82S155           **/
/*****************************************************************/

/**   Inputs  **/
Pin 1      = cpu_clk;        /* CPU clock                */
Pin [2..6] = [a15..11];      /* CPU Address Bus          */
Pin [7,8]  = ![memw,memr] ;

/* Memory Data Strobes (active low)*/
Pin 9      = reset;          /* System Reset             */
Pin 11     = !oe;            /* Output Enable (active low) */

/**  Outputs  **/
Pin 19     = !rom_cs;        /* ROM chip select (active low)*/
```

```
Pin 18      = ready;        /* CPU ready          */
Pin 15      = wait1;        /* Wait state 1       */
Pin 14      = wait2;        /* Wait state 2       */
Pin [13,12] = ![ram_cs1..0];
                           /* RAM chip select (active low) */

/* Declarations and Intermediate Variable Definitions      */
Field memadr = [a15..11]  ;     /* Give the address bus    */
                                /* the Name “memadr”       */

memreq = memw # memr ;          /* Create the intermediate */
                                /* variable “memreq”       */
select_rom = memr & memadr:[0000..1FFF] ; /* = rom_cs      */

/**  Logic Equations  **/
rom_cs = select_rom;
ram_cs0 = memreq & memadr:[2000..27FF]  ;
ram_cs1 = memreq & memadr:[2800..2FFF]  ;

/* read as: when select_rom is true and reset is false */
wait1.d = select_rom & !reset  ;

/* read as: when when select_rom is true and wait1 is true  */
/* Synchronous Reset                                        */
wait2.d = select_rom & wait1 ;   /* wait1 delayed           */

ready.oe = select_rom;           /* Turn Buffer off         */
ready = wait2;                   /* end wait                */
```

Step 6 - PLD ソースファイルのコンパイル

このステップでは、論理記述ファイル SAMPLE.PLD を、ターゲットデバイス PAL16R4 で使用できるようにコンパイルします。

ターゲット PLD とコンパイルオプションを Configure Advanced PLD ダイアログボックスで指定して下さい。以下のオプションをイネーブルにして下さい。

- Absolute ABS - SAMPLE.ABS を作成します。後でシミュレーターにより使用されるアブソリュートファイルです。(このファイルは Step7 で必要です。)このファイルには、デバイスの内部へプログラムされる論理関数が記述されています。シミュレーターはこの記述とユーザが作成した入力ファイルのテストベクタとを比較し入力ファイルの応答ベクタが正しい刺激ベクタの応答であるかどうかを確認します。
- Fuse Plot in Doc File and Equations in Doc File - SAMPLE.DOC を作成します。このファイルはドキュメンテーションファイルです。このファイルには、中間変数と出力ピン変数の完全に展開されたプロダクトタームやヒューズプロットとチップダイアグラムが記述されます。
- Error List LST - SAMPLE.LST を作成します。このファイルは、リストファイルです。記述ファイルをサイド作成したものです。ただし、行番号が追加され、コンパイル中に生成されたエラーメッセージがファイルの最後に付け加えられます。
- JEDEC - SAMPLE.JED を作成します。このファイルは、デバイスプログラマーへダウンロードする JEDEC ファイルです。このファイルには、ヒューズパターンが記述されます。テストベクタは、シミュレーション中に JEDEC ファイルに加えられます。

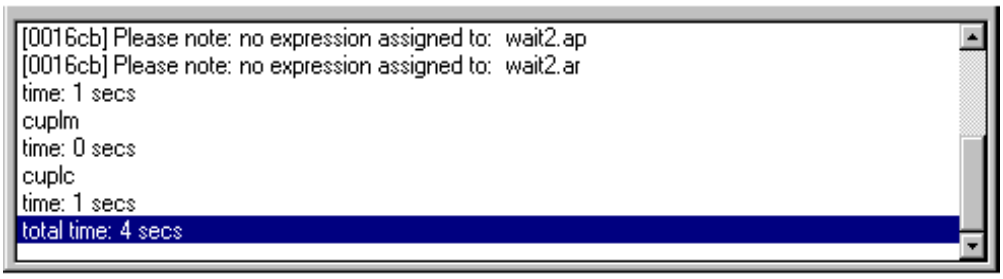
➔ SAMPLE.JED というファイル名は、論理記述ファイルのヘッダー情報セクションの NAME フィールドで決められます。ひとつのデバイスだけがファイルに記述されている場合、同じ名前をファイル名に使用して下さい。(この場合、SAMPLE)

➔ 現在のドキュメントの最新のファイルがコンパイラーのソースファイルとなるように、PLD ソースファイルを保存してから、コンパイラーを実行して下さい。

コンパイラーを設定が完了したら、Configure Advanced PLD ダイアログボックスの OK ボタンをクリックして下さい。

コンパイルするには、SAMPLE.PLD を現在のドキュメントにし PLD ツールバーのコンパイルのボタンを押して下さい。

The Advanced PLD - Compiling ダイアログボックスがポップアップ表示されます。Info ボタンを押して下さい。以下のメッセージがダイアログボックスに表示されます。各コンパイラーモジュールが完了するまでに要した時間が表示されます。実際の時間は使用されるシステムに応じて変わります。



SAMPLE.LST と SAMPLE.DOC ファイルをテキストエディターにオープンして下さい。

リストファイル、SAMPLE.LST は、ソースファイルと同じです。ただし、行番号とエラーメッセージが追加されています。行番号により、エラーが発生した場合、エラーの位置を速く見つけることができます。

SAMPLE.LST

```
CUPL  Version 4.XX  Serial # XX-XXX-XXXX
Copyright (C) 1996 Protel International
CREATED Thur Jan 14 08:42:12 1990
LISTING FOR LOGIC DESCRIPTION FILE: sample.pld;

1:Name                  Sample;
2:Partno                P9000183;
3:Date                  07/16/87;
4:Revision              02;
5:Designer              Osann;
6:Company               ATI;
7:Assembly              PC Memory;
8:Location              U106;
9:
10:/*****************************************************/
11:/* This device generates chip select signals for one  */
12:/* 8Kx8 ROM and two 2Kx8 static RAMs. It also drives  */
13:/* the system READY line to insert a wait-state of at */
14:/* least one cpu clock for ROM accesses               */
15:/*****************************************************/
16:/*****************************************************/
17:/** Allowable Target Device Types : PAL16R4, 82S155  **/
18:/*****************************************************/
19:/**   Inputs  **/
20:
21:Pin 1       = cpu_clk   ;      /*  CPU clock          */
22:Pin [2..6]  = [a15..11] ;      /*  CPU Address Bus    */
23:Pin [7,8]   = ![memw,memr]  ;  /* Memory Data Strobes */
24:Pin 9       = reset     ;      /*  System Reset       */
25:Pin 11      = !oe       ;      /* Output Enable       */
```

```
26:
27:/**  Outputs  **/
28:
29:Pin 19      = !rom_cs    ;      /*                    */
30:Pin 18      = ready      ;      /*                    */
31:Pin 15      = wait1      ;      /*                    */
32:Pin 14      = wait2      ;      /*                    */
33:Pin [13,12] = ![ram_cs1..0]  ;  /*                    */
34:
35:/* Declarations and Intermediate Variable Definitions */
36:
37:Field memadr = [a15..11] ;   /* Give the address bus */
38:                            /* the Name “memadr”    */
39:
40:memreq = memw # memr ;       /* Create the intermediate */
41:                            /* variable “memreq”  */
42:
43:select_rom = memr & memadr:[0000..1FFF] ; /* = rom_cs */
44:
45:/**  Logic Equations  **/
46:
47:rom_cs = select_rom;
48:ram_cs0 = memreq & memadr:[2000..27FF]  ;
49:ram_cs1 = memreq & memadr:[2800..2FFF]  ;
50:wait1.d = select_rom & !reset  ;
51:                                 /* Synchronous Reset */
52:wait2.d = select_rom & wait1 ;   /* wait1 delayed     */
53:ready.oe = select_rom ;          /* Turn Buffer off   */
54:ready = wait2 ;                  /* end wait          */
Jedec Fuse Checksum      (4D50)
Jedec Transmit Checksum  (E88F)
```

以下にコンパイラーにより作成されたドキュメンテーションファイルを示します。

SAMPLE.DOC

```
******************************************************************
                            Sample
******************************************************************
CUPL          4.XX Serial# XX-XXX-XXXX
Device        p16r4 Library DLIB-d-26-11
Created       Mon Aug 20 10:48:32 1990
Name          Sample;
Partno        P9000183;
Date          04/1/90;
Revision      02;
Designer      Osann;
Company       ATI;
Assembly      PC Memory;
Location      U106;
==================================================================
                      Expanded Product Terms
==================================================================
wait1.d  =>
     !memr
     # a15
     # a14
     # a13
     # reset
```

```
select_rom =>
       !a13 & !a14 & !a15 & memr

wait2.d  =>
     !memr
     # a15
     # a14
     # a13
     !wait

memadr  =>
       a15,a14,a13,a12,a11

ready  =>
       !wait2

ready.oe  =>
       !a13 & !a14 & !a15 & memr
rom_cs  =>
       !a13 & !a14 & !a15 & memr
memreq  =>
   memw
   # memr

ram_cs0  =>
   !a11 & !a12 & !a13 & !a14 & !a15 & memw
   # !a11 & !a12 & !a13 & !a14 & !a15 & memr

ram_cs1  =>
   a11 & !a12 & a13 & !a14 & !a15 & memw
   # a11 & !a12 & a13 & !a14 & !a15 % memr

rom_cs.oe  =>
    1

ram_cs0.oe  =>
    1

ram_cs1.oe  =>
    1

================================================================
                      Symbol Table
================================================================
Pin  Variable                          Pterms     Max      Min
Pol   Name          Ext     Pin   Type   Used     Pterms   Level
      wait1                 15     V      -        -        -
      wait1         d       15     X      5        8        1
      all                   6      V      -        -        -
      select_rom            0      I      1        -        -
      wait2                 14     V      -        -        -
      wait2         d       14     X      5        8        1
      a12                   5      V      -        -        -
      a13                   4      V      -        -        -
      a14                   3      V      -        -        -
      a15                   2      V      -        -        -
 !    oe                    11     V      -        -        -
```

```
 !    memr                 8      V       -         -         -
    memadr                0      F       -         -         -
    ready                 18     V       1         7         1
    ready          oe     18     X       1         1         1
 !  memw                  7      V       -         -         -
    cpu_clk               1      V       -         -         -
 !  rom_cs                19     V       1         7         1
    reset                 9      V       -         -         -
    memreq                0      I       2         -         -
 !  ram_cs0               12     V       2         7         1
 !  ram_cs1               13     V       2         7         1
    rom_cs         oe     19     D       1         1         0
    ram_cs0        oe     12     D       1         1         0
    ram_cs1        oe     13     D       1         1         0

LEGEND   F : field     D : default  M : extended node
           N : node   I : Intermediate variable   T : function
           V : variable   X : extended variable   U : undefined
=================================================================
                          Fuse Plot
=================================================================

Pin #19
 0000 --------------------------------
 0032 -x---x---x---------------x------
 0064 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 0096 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 0128 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 0160 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 0192 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 0224 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Pin #18
 0256 -x---x---x---------------x------
 0288 ------------------------x--------
 0320 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 0352 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 0384 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 0416 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 0448 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 0480 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Pin #17
 0512 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 0544 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 0578 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 0608 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 0640 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 0672 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 0704 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 0738 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Pin #16
 0768 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 0800 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 0832 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 0864 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 0896 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

```
 0928 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 0960 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 0992 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Pin #15
 1024 ------------------------x-------
 1056 x-------------------------------
 1088 ----x---------------------------
 1120 --------x-----------------------
 1152 ----------------------------x---
 1184 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 1216 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 1248 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Pin #14
 1280 ------------------------x-------
 1312 x-------------------------------
 1344 ----x---------------------------
 1378 --------x-----------------------
 1408 -------------------x------------
 1440 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 1472 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 1504 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Pin #13
 1536 --------------------------------
 1568 -x---x--x----x--x----x----------
 1600 -x---x--x----x--x--------x------
 1632 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 1664 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 1696 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 1728 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 1760 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Pin #12
 1792 --------------------------------
 1824 -x---x--x----x--x----x----------
 1856 -x---x--x----x--x--------x------
 1888 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 1920 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 1952 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 1984 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 2016 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

=====================================================================
                       Chip Diagram
=====================================================================
                          -------------
                         |   WAITGEN   |
              cpu_clk x--| 1        20 |--x VCC
                         |             |
                  a15 x--| 2        19 |--x !rom_cs
                         |             |
                  a14 x--| 3        18 |--x ready
                         |             |
                  a13 x--| 4        17 |--x
                         |             |
                  a12 x--| 5        16 |--x
```

```
                    |             |
                a11 x—| 6       15 |—x wait1
                    |             |
               !memw x—| 7       14 |—x wait2
                    |             |
               !memr x—| 8       13 |—x !ram_csl
                    |             |
               reset x—| 9       12 |—x !ram_cs0
                    |             |
                GND x—|10       11 |—x !oe
                    |             |
                     -------------
```

特定のデバイスに対してピンリストでアクティブハイと宣言された出力変数がある場合、PAL16R4 の固定反転出力バッファは、コンパイラーによりドモルガンの定理が実行されるので、WAIT1.D と WAIT2.D の展開されたプロダクトタームは、5 つのプロダクトタームで示されます。

Step 7 - シミュレーションテストベクタファイルの作成

このステップでは、PAL16R4 デバイス用にコンパイルされた設計をシミュレーションにより検証します。この段階を踏んでから、ロジックプログラマーへダウンロードするとロジックの間違いが発生する率が減ります。

このセクションでは、ソース仕様ファイル、SAMPLE.SI が作成されます。このファイルには、シミュレーターの入力に使用されるテストベクタが記述されます。シミュレーターはテストベクタ入力と予想される出力と、コンパイラーの演算中に作成される SAMPLE.ABS に記述された実際の値とフラグとの違いを比較します。

以下にソース仕様ファイルのサンプルを示します。

```
Name           Sample;
Partno         P9000183;
Date           07/16/87;
Revision       02;
Designer       Osann;
Company        ATI;
Assembly       PC Memory;
Location       U106;

/*************************************************/
/* This device generates chip select signals for one  */
/* 8Kx8 ROM and two 2Kx8 static RAMs. It also drives  */
/* the system READY line to insert a wait-state of at */
/* least one cpu clock for ROM accesses               */
/*************************************************/

ORDER:
     cpu_clk, %2, a15, %2, a14, %2,
     a13, %2, a12, %2, a11, %2,
     !memw, %2, !memr, %2, reset, %2, !oe,
     %4, !ram_cs1, %2, !ram_cs0, %2, !rom_cs, %2,
     wait1, %2, wait2, %2, ready;

VECTORS:
/* 123456-leave six blanks to allow for numbers in .SO file */
$msg "     Power On Reset                                  ";
           O  X  X  X  X  X  1  1  1  0    H  H  H  *  *  Z
$msg "     Reset Flip Flops                                ";
```

```
           C  X  X  X  X  X  1  1  0  0    H  H  H  L  L  Z
$msg "     Write RAM0                                        ";
           0  0  0  1  0  0  0  1  0  0    H  L  H  L  L  Z
$msg "     Read RAM0                                         ";

           0  0  0  1  0  0  1  0  0  0    H  L  H  L  L  Z
$msg "     Write RAM1                                        ";
           0  0  0  1  0  1  0  1  0  0    L  H  H  L  L  Z
$msg "     Read RAM1                                         ";
           0  0  0  1  0  1  1  0  0  0    L  H  H  L  L  Z
$msg "     Begin ROM read                                    ";
           0  0  0  0  0  0  1  0  0  0    H  H  L  L  L  L
$msg " Two clocks for wait state, Then drive READY High      ";
$repeat2;
            C  0  0  0  0  0  1  0  0  0    H  H  L  *  *  *
$msg "     End ROM Read                                      ";
            0  0  0  0  0  0  1  1  0  0    H  H  H  H  H  Z
$msg "     End ROM Read                                      ";
            C  0  0  0  0  0  1  1  0  0    H  H  H  L  L  Z
```

ソース仕様ファイルには、ヘッダー情報とタイトルブロックと、ORDER 命令、VECTORS 命令の 3 つの主な部分があります。

SAMPLE.SI は、同じ SAMPLE.PLD と同じヘッダー情報を持ち、現在のリビジョンレベルを含む適切なファイルがお互いに比較されること確認する必要があります。従って、テキストエディターでは、SAMPLE.PLD を SAMPLE.SI として保存し SAMPLE.SI のすべてを削除して下さい。ただし、ヘッダーとタイトルブロックは残して下さい。以下に結果を示します。

```
Name          Sample;
Partno        P9000183;
Date          07/16/87;
Revision      02;
Designer      Osann;
Company       ATI;
Assembly      PC Memory;
Location      U106;

/*****************************************************/
/* This device generates chip select signals for one  */
/* 8Kx8 ROM and two 2Kx8 static RAMs. It also drives  */
/* the system READY line to insert a wait-state of a  */
/* least one cpu clock for ROM accesses               */
/*****************************************************/
```

ORDER 命令では、テストベクタにインクルードされる SAMPLE.PLD からの入出力変数が一覧で表示されます。テスト変数で使用される順番に変数が並べられます。すなわち、クロック変数、CPU_CLK が最初に置かれ、続いて他の入力変数が置かれます。出力変数は右に置いて下さい。変数はコンマで区切って下さい。%記号を使用して変数間にスペースを挿入して下さい。すなわち、各変数の間にはスペースを 2 つ置き、一覧の最後の入力変数!OE の間と最初の出力変数!RAM_CS1 の間には、スペースを 4 つ置いて下さい。ORDER 命令を以下のように入力して下さい。

```
ORDER:
   CPU_CLK, %2, A15, %2, A14, %2,
   A13, %2,A12, %2, ALL, %2,
   !MEMW, %2, !MEMR, %2, RESET, %2, !OE,
   %4, !RAM_CS1, %2, !RAM_CSO, %2, !ROM_CS, %2
   WAIT1, %2, READY;
```

ORDER 命令に続いて、11 個のテストベクタがある関数テーブルを作成する VECTORS 命令を入力して下さい。ベクタを簡単に理解するために、テストベクタのヘッダーが作成され、.SO ファイルの ORDER 命令のすぐ後に置かれます。テストベクタは ORDER 命令に現れる信号名で構成され、ベクタセクションが読みやすいように配置されます。

さて、テストベクタを入力して下さい。値を各入力変数に入力し、出力変数に予想される値を入力して下さい。

➔ このガイドのテスト仕様ソースファイルの作成のセクションを参照して下さい。

$MSG ディレクティブを仕様して、関数によりテストされるデバイス関数を記述して下さい。上記の ORDER 命令により、テストベクタを作成する場合、 スペーシングが指定されます。例えば、最初のベクタ、Power On Reset は、以下のように入力して作成して下さい。

```
$msg " Power On Reset     ";
0 X X X X X 1 1 1 0 H H H * * Z
```

デバイスの中には電源の投入Xになったりまたは H や L になるデバイスもあるので、出力値(*)がWAIT1やWAIT2で使用されシミュレーターにレジスタの電源の投入を指示することに注意して下さい。アスタリスクを使用すると統一的なシミュレーションファイルを作成することができます。

テストベクタの残りの部分は、以下に示す様に入力して下さい。

```
/* 123456-leave six blanks to allow for numbers in .SO file */
$msg "     Power On Reset                                    ";
           O  X  X  X  X  X  1  1  1  0    H  H  H  *  *  Z
$msg "     Reset Flip Flops                                  ";
           C  X  X  X  X  X  1  1  0  0    H  H  H  L  L  Z
$msg "     Write RAM0                                        ";
           0  0  0  1  0  0  0  1  0  0    H  L  H  L  L  Z
$msg "     Read RAM0                                         ";
           0  0  0  1  0  0  1  0  0  0    H  L  H  L  L  Z
$msg "     Write RAM1                                        ";
           0  0  0  1  0  1  0  1  0  0    L  H  H  L  L  Z
$msg "     Read RAM1                                         ";
           0  0  0  1  0  1  1  0  0  0    L  H  H  L  L  Z
$msg "     Begin ROM read                                    ";
           0  0  0  0  0  0  1  0  0  0    H  H  L  L  L  L
$msg " Two clocks for wait state, Then drive READY High      ";
$repeat2;
            C  0  0  0  0  0  1  0  0  0    H  H  L  *  *  *
$msg "     End ROM Read                                      ";
            0  0  0  0  0  0  1  1  0  0    H  H  H  H  H  Z
$msg "     End ROM Read                                      ";
            C  0  0  0  0  0  1  1  0  0    H  H  H  L  L  Z
```

テストベクタファイルの$REPEAT ディレクティブにより、8 番目のベクタが 2 回繰返されます。WAIT1 と WAIT2、READY の 8 番目のベクタのアスタリスクにより、シミュレーターは入力を基に出力を計算し出力ファイルに結果を記述します。

クロック変数、CPU_CLK の値は、あるベクタでは 0 でその他では C になります。0 の値になるクロックは作動しません。C の値によりシミュレーターはベクタの入力値を検証しクロック前に内部的にフィードバックされるレジスタード出力の以前のベクタに注目します。それから、クロックが適用されるとシミュレーターは、レジスタード出力やノンレジスタード出力の予想される出力を計算します。

VECTORS 命令の入力が完了したら、ファイルを保存して下さい。次のステップでは、シミュレーターを使用してシミュレーションを実行します。

Step 8 - デバイスのシミュレーション

シミュレーターが実行されると、SAMPLE.SO ファイルが作成されます。このファイルには、シミュレーションの結果が保存されます。エラーメッセージは、エラーの信号名と一緒に各ベクタに続いて一覧で記述されます。

シミュレーターを実行するには、SAMPLE.PLD をカレントドキュメントにし、PLD ツールバーのシミュレーションボタンを押して下さい。

Advanced PLD-Simulate ダイアログボックスで Info ボタンを押すと、シミュレーション情報が表示されます。View Results チェックボックスをイネーブルにするとシミュレーション結果を Waveform Editor へ自動的にロードします。

➔ SAMPLE.SO は ASCII ファイルです。従って、テキストエディターで開くことができます。

以下に SAMPLE.SO の内容を示します。

SAMPLE.SO

```
CSIM:    CUPL Simulation Program
Version 4.XX Serial # XX-XXX-XXXX
copyright (c) 1996 Protel International
CREATED Thur Aug 20 09:34:16 1990
 1: Name                    Sample;
 2: Partno                  P9000183;
 3: Date                    07/16/87;
 4: Revision                02;
 5: Designer                Osann;
 6: Company                 ATI;
 7: Assembly                PC Memory;
 8: Location                U106;
 9:
10: /*****************************************************/
11: /* This device generates chip select signals for one  */
12: /* 8Kx8 ROM and two 2Kx8 static RAMs. It also drives  */
13: /* the system READY line to insert a wait-state of at */
14: /* least one cpu clock for ROM accesses               */
15: /*****************************************************/
16:
17: ORDER:
18:      cpu_clk, %2, a15, %2, a14, %2,
19:      a13, %2, a12, %2, a11, %2,
20:      !memw, %2, !memr, %2, reset, %2, !oe,
21:      %4, !ram_cs1, %2, !ram_cs0, %2, !rom_cs, %2,
22:      wait1, %2, wait2, %2, ready;
23:
=============================================================
                    Simulation Results
=============================================================
                                        !  !
     c                                  r  r  !
     p                                  a  a  r
     u                 !  !  r         m  m  o  w  w  r
     _                 m  m  e         _  _  m  a  a  e
     c  a  a  a  a  a  e  e  s  !       c  c  _  i  i  a
     l  1  1  1  1  1  m  m  e  o       s  s  c  t  t  d
     k  5  4  3  2  1  w  r  t  e       1  0  s  1  2  y
     ________________________________   _______________
    Power On Reset
```

```
0001: O  X  X  X  X  X  1  1  1  0    H  H  H  X  X  Z
     Reset Flip Flops
0002: C  X  X  X  X  X  1  1  0  0    H  H  H  L  L  Z
     Write RAM0
0003: 0  0  0  1  0  0  0  1  0  0    H  L  H  L  L  Z
     Read RAM0
0004: 0  0  0  1  0  0  1  0  0  0    H  L  H  L  L  Z
     Write RAM1
0005: 0  0  0  1  0  1  0  1  0  0    L  H  H  L  L  Z
     Read RAM1
0006: 0  0  0  1  0  1  1  0  0  0    L  H  H  L  L  Z
     Begin ROM read
0007: 0  0  0  0  0  0  1  0  0  0    H  H  L  L  L  L
     Two clocks for wait state, Then drive READY High
0008: C  0  0  0  0  0  1  0  0  0    H  H  L  H  L  L
0009: C  0  0  0  0  0  1  0  0  0    H  H  L  H  H  H
     End ROM Read
0010: 0  0  0  0  0  0  1  1  0  0    H  H  H  H  H  Z
     End ROM Read
0011: C  0  0  0  0  0  1  1  0  0    H  H  H  L  L  Z
```

SAMPLE.SO を、SAMPLE.SI と比較して下さい。$REPEAT ディレクティブの結果としてベクタ 8 と 9 が作成され、シミュレーターにより SAMPLE.SI のアスタリスクが WAIT1 や WAIT2、READY 信号の適切な論理レベル(HまたはL)に置き換えられていることに注意して下さい。

シミュレーションが完了すると、コンパイラーの実行中(STEP6)に作成される JEDEC ファイルにテストベクタが追加されます。Configure Advanced PLD ダイアログボックスでイネーブルにされた JEDEC オプションでシミュレーションを再び実行して下さい。

以下に SAMPLE.JED の内容を示します。この時点で、このファイルにはプログラム情報とテスト情報が記述されています。

SAMPLE.JED

```
CUPL         4.XX   Serial# XX-XXX-XXXX
Device       p16r4  Library DLIB-d-26-11
Created      Thur Aug 20 09:52:02 1990
Name         Sample
Partno       P9000183
Revision     02
Date         12/16/89
Designer     Osann
Company      ATI
Assembly     PC Memory;
Location     U106;
*QP20
*QF2048
*G0
*F0
*L00000 11111111111111111111111111111111
*L00032 10111011101111111111111110111111
*L00256 10111011101111111111111110111111
*L00288 11111111111111111111111011111111
*L01024 11111111111111111111111101111111
*L01056 01111111111111111111111111111111
*L01088 11110111111111111111111111111111
*L01120 11111111011111111111111111111111
*L01152 11111111111111111111111111110111
*L01280 11111111111111111111111101111111
```

```
*L01312 01111111111111111111111111111111
*L01344 11110111111111111111111111111111
*L01376 11111111011111111111111111111111
*L01408 11111111111111111110111111111111
*L01536 11111111111111111111111111111111
*L01568 10111011011110110111101111111111
*L01600 10111011011110110111111110111111
*L01792 11111111111111111111111111111111
*L01824 10111011011110111011101111111111
*L01856 10111011011110111011111110111111
*C4D50
*V0001 0XXXXX111N0HHXXXXZHN
*V0002 CXXXXX110N0HHLLXXZHN
*V0003 000100010N0LHLLXXZHN
*V0004 000100100N0LHLLXXZHN
*V0005 000101010N0HLLLXXZHN
*V0006 000101100N0HLLLXXZHN
*V0007 000000100N0HHLLXXLLN
*V0008 C00000100N0HHLHXXLLN
*V0009 C00000100N0HHHHXXHLN
*V0010 000000110N0HHHHXXZHN
*V0011 C00000110N1HHLLXXZHN
*3152
```

Step 9 - シミュレーション波形の表示

波形の組みとしてシミュレーション結果を観察するには、SAMPLE.SO ファイルを Waveform Editor を用いて開いて下さい。これをいつでも行なうには;

- File-Open を選択して下さい。
- Open Document ダイアログボックスで Editor を Wave に設定して下さい。Wave が一覧の中にない場合、Wave Server がインストールされていないこと意味します。インストレーションセクションのこのサーバーのインストールに関する説明を参照して下さい。
- ファイルの種類を PLD Simulation files (*.SO)に設定して下さい。
- SAMPLE.SO を選択し"開く"のボタンをクリックして開いて下さい。
- シミュレーション結果が波形として、Waveform エディターに表示されます。Waveform エディターの使用法についての簡単な説明は、オンラインヘルプを参照して下さい。

概要

このセクションでは、PLD ソースファイルとシミュレーターテストベクタをを作成しコンパイルする機会を与えます。

- テンプレートファイルを使用して下さい。
- 中間式や論理式を書いて設計を記述して下さい。
- コンパイラーを起動して下さい。
- テスト仕様ファイルを作成しコンパイルして設計を検証して下さい。
- シミュレーターを実行し論理設計のシミュレーションを実行して下さい。
- シミュレーション波形を観察して下さい。

設計例

このセクションでは設計例に沿って、CUPL 言語を使用してどのように設計を記述するかを説明します。設計例を以下に示します。

例 1. 簡単なゲート (GATES.PLD)

例 2. TTL 変換 (WGTTL.PLD)

例 3. 2 ビットカウンタ (FLOPS.PLD)

例 4. ステートマシンシンタクスを使用したデカードアップ/ダウンカウンタ (COUNT10.PLD)

例 5. セグメントのディスプレイデコーダ (HEXDISP.PLD)

例 6. ロード機能とリセット機能のある 4 ビットカウンタ

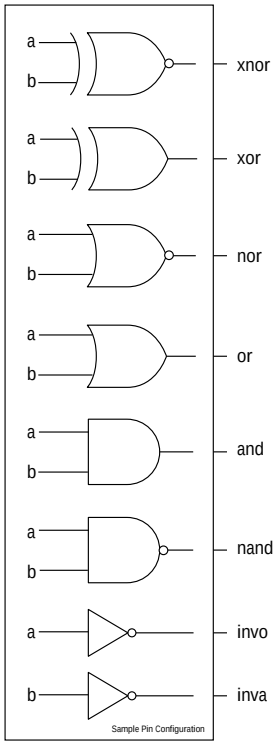
各設計の論理記述ファイルを括弧内に示します。これらのファイルはコンパイラーに入力してドキュメントやダウンロードファイルを作成することができます。

それぞれのテスト仕様ファイルは(ファイル名.SI)は、シミュレーターで設計を確認できるようにそれぞれの論理記述ファイルについてあります。

例 1 - 簡単なゲート

この設計は、簡単なゲートの出力を作成するロジックです。

出力は、ゲートの関数を表わすラベルが付けられています。例えば、AND ゲートは出力に AND とラベルが付けられます。



以下に、この設計を記述する CUPL ソースファイル(Advanced PLD パッケージの中の GATES.PLD)を示します。

GATES.PLD

```
Name          Gates;
Partno        CA0001;
Date          07/16/87;
Designer      G Woolheiser;
Company       ATI;
Location      San Jose, CA.;
Assembly      Example;

/*****************************************************/
/* This is an example to demonstrate how the Compiler */
/* compiles simple gates.                             */
/*****************************************************/
/*   Target Devices: P16L8, P16P8, EP300, and 82S153  */
/*****************************************************/

/* Inputs:   define inputs to build simple gates      */

Pin 1 = a;
Pin 2 = b;

/* Outputs:   define outputs as active HI levels      */

/* For PAL16L8 and PAL16LD8, De Morgan’s Theorem is   */
```

```
/* applied to invert all outputs due to fixed         */
/* inverting buffer in the device.                    */

Pin 12 = inva;
Pin 13 = invb;
Pin 14 = and;
Pin 15 = nand;
Pin 16 = or;
Pin 17 = nor;
Pin 18 = xor;
Pin 19 = xnor;

/* Logic:   examples of simple gates expressed in CUPL */

inva = !a;            /* inverters              */
invb = !b;
and  = a & b;         /* and gate               */
nand = !(a & b);      /* nand gate              */
or   = a # b;         /* or gate                */
xor  = a $ b;         /* xor gate               */
nor  = !(a # b)       /* nor gate               */
xnor = !(a $ b);      /* exclusive nor gate     */
```

ファイルの最初の部分は、互換性のある PLD や設計されている関数の説明などの記述です。まず、コンパイラーにより出力ファイルの名前として使用される名前の行があります。Partno は会社の固有の部品番号を表わします。部品番号はターゲット PLD のタイプではありません。Data はコンパイル時のデータを示します。データは、現在のものに常に更新されます。Designer は、設計者の名前または設計チームの名前です。Assembly はアッセンブリ名または PLD が使用される PC 基盤の名前を表わすために使用されます。必要に応じて、省略形の ASSY を使用して下さい。Location は、PLD が配置される PC 基盤上の座標を表わすために使用されます。省略形の LOC も使用することができます。

ピン宣言は設計ダイアグラムの入力と出力に対応して行われます。この例でのゲートには、ゲートに接続される 2 つの入力が必要です。a と b は入力ピンの名前です。次に出力ピンに名前が割り付けられます。付けられた名前は関数の動作を表わしています。このように関数の動作を表わす名前を付けることを推奨します。これにより、ファイルのデバッグ時や更新時に内容が解りやすくなります。

ファイルの Logic セクションでは、式によりゲートが記述されます。ブーリアンシンタクスを使用して各出力ピンが入力ピン a と b の関数の出力として指定されます。

固定の反転バッファを持つ PAL16L8 と PAL16LD8 デバイスでは、ピンリストの中でアクティブハイで宣言されているので、コンパイラーはドモルガンの定理を適用して出力をすべて反転します。例えば、アクティブハイとして宣言されている出力ピンの OR ゲートの以下のような式はコンパイラーにより反転され、

```
or = a # b ;
```

以下のような 1 つの展開されたプロダクトタームになります。(ドキュメンテーションファイルで示される様に)

```
or => !a & !b
```

例 2 - TTL 設計の PLD への変換

この例では、PLD を使用して既存の TTL 回路を変換する方法を示します。変換には、TTL 論理設計のゲートを PLD でコンパイルできる等価なブーリアン論理式に変換することが必要です。

図 5.1 に、設計する論理システムで使用される TTL ゲート表現とそれぞれのゲートに対応するブーリアン式を示します。

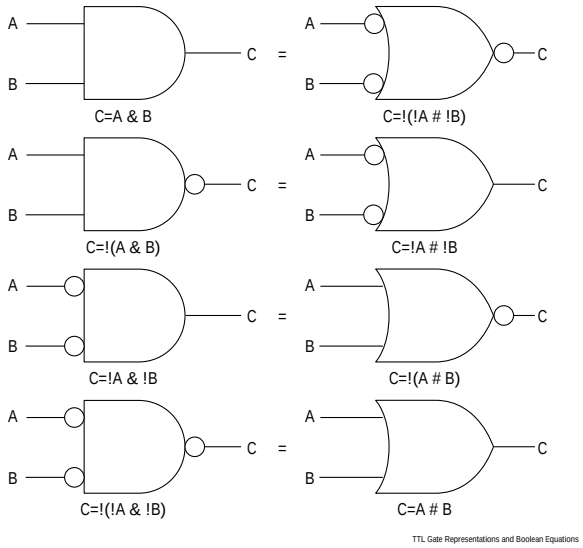


図 5.1 TTL ゲート表現とブーリアン式

図 5.1 で示される基本的な変換ルールを使用すれば、TTL ゲートのシステムの各ゲートの式を記述できます。CUPL は表現の置き換えプロセスを使用して TTL の各ゲートを表現する式からブーリアン式を構築します。式の置き換えは 1 つのゲート毎に行われます。

図 5.2 に例 1 で変換された TTL ロジックの図を示します。

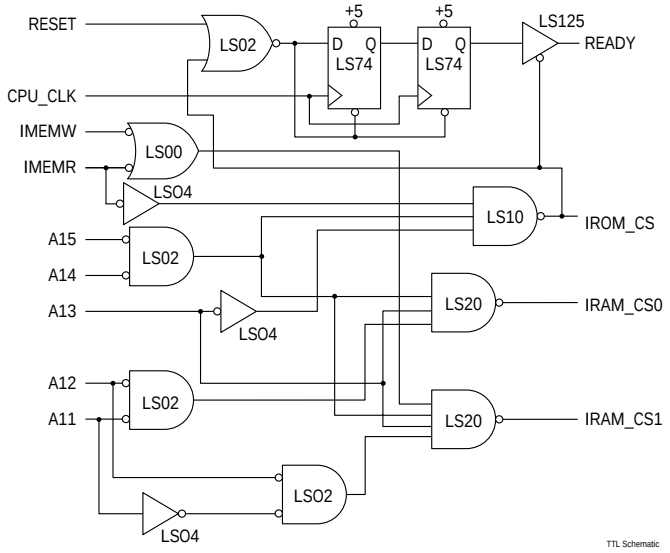


図 5.2 TTL の図

図 5.2 で示される TTL ロジックにより、Sample Design Session で作成された SAMPLED.PLD と同じアドレスデコードや待ち状態を実行することができます。この TTL 回路と等価な

PLD により、デバイスを 1 つ使用し 5〜6 パッケージが置き変わります。変換プロセスの最初のステップは TTL の図から PLD に変換するロジックを決めることです。

図 5.3 に、TTL の図の四角で囲まれたロジックと等価な PLD ダイアグラムと PLD のピン割り付けを示します。

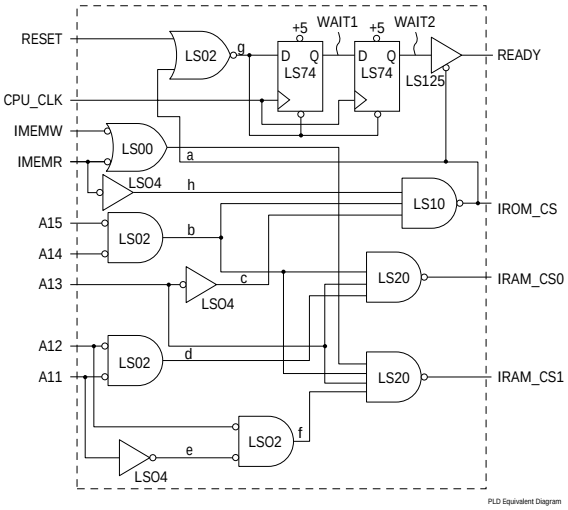


図 5.3 PLD の等価な図

(PLD 出力ピンに接続されていない)内部ゲートの出力に A-H の固有の名前が付けられていることに注意して下さい。これにより論理記述ファイルへの式の入力がやりやすくなります。

TTL 設計を基に待ち状態を生成する回路のため、今回の設計の変換使用される論理記述ファイルは、WGTTL.PLD と名前を付けます。

以下に WGTTL.PLD の内容示します。

```
Name            Sample;
Partno          P9000183;
Date            07/16/87;
Revision        02;
Designer        Osann;
Company         ATI;
Assembly        PC Memory;
Location        U106;

/*****************************************************************/
/* このデバイスは 1 つの 8Kx8ROM と 2 つの 2Kx8 スタティック RAM へチップ      */
/* セレクト信号を発生します。また、その信号はシステムの READY 線を駆動し ROM   */
/* アクセス時に少なくとも 1CPU クロック分の待ち状態を生成します。              */
/*****************************************************************/

/** Inputs **/

PIN 1           = cpu_clk       ;  /* CPU clock          */
PIN [2..6]      = [a15..11]     ;  /* CPU Address Bus    */
PIN [7  8]      = ![memw,memr]  ;  /* Memory Data Strobes*/
PIN 9           = reset         ;  /* System Reset       */
PIN 11          = !oe           ;  /* Output enable      */

/** Outputs **/

PIN 19          = !rom_cs       ;  /* ROM Chip Select    */
PIN 18          = ready         ;  /* CPU ready signal   */
```

```
PIN 15         = wait1         ;  /* Start wait state   */
PIN 14         = wait2         ;  /* End wait state     */
PIN [13,12]    = ![ram_cs1..0] ;  /* RAM chip selects   */

/** Declarations and Intermediate Variable Definitions **/

a = !(!memw) # !(!memr) ;
b = !a15 & !a14 ;
c = !a13 ;
d = !a12 & !a11 ;
e = !a11 ;
f = !a12 & !e ;
g = !(!rom_cs # reset) ;
h = !(!memr) ;
/** Logic Equations **/

!rom_cs = !(h & b & c);
!ram_cs0 = !(a & b & a13 & d) ;
!ram_cs1 = !(a & b & a13 & f) ;
wait1.d = g ;
wait2.d = wait1 & g ;
ready.oe = !(!(h & b & c)) ;
ready = wait2 ;
```

機能は同じのため、ヘッダー情報は、SAMPLE.PLD の物と同じです。

内部ゲートの論理式は、Declarations と Intermediate Variable Definitions のセクションに置かれます。このセクションの式は、図 5-3 で図に割り付けられる A-H の出力変数名を使用します。例えば、AND ゲート LS02 は以下の式でで表わされます。

```
d = !a12 & !a11 ;
```

このセクションの式は、簡単化できます。例えば、以下の式のような二重否定をまとめることができます。

```
a = !(!memw) # !(!memr) ;
```

となります。

```
a = memw # memr ;
```

ファイルの Logic Equations のセクションには、PLD の出力信号を記述する式があります。これらの式は、内部ゲートの出力を表わす中間式で記述されます。例えば、AND ゲート LS10 は、!ROM_CS という出力信号があり、信号 H には、B と C という入力があります。したがって、LS10 は以下の式で表わすことができます。

```
!rom_cs = !(h & b & c) ;
```

内部ゲートの定義が違うので WGTTL.PLD と SAMPLE.PLD は厳密には同じではありません。しかし、コンパイルした場合、同じ機能になります。これは、それぞれの論理記述ファイルのシミュレーションを実行することにより確認できます。

TTL 設計を PLD に変換する場合、機能が多少変更される場合があります。LS74 のような TTL フリップフロップにある非同期リセットの機能は通常使用される PLD にはほとんどありません。しかし、同じリセット機能は RESET 変数をすべてのプロダクトタームに取り込みクロックを利用して同期リセットを行なうことができます。

従って、WGTTL.PLD は WAIT1(wait1.d=g;)と WAIT2(wait2.d=wait1.d&g)の式で使用される G の式に!RESET を取り込みます。非同期リセットのタイミングと同期リセットのタイミングに違いがあるものの、同期リセットにしてもデバイスは正しく動作します。

例 2 で示される簡単な方法を使用すると、多くの TTL 設計を変換することができます。特に簡単なゲートで構成された TTL 設計を等価な PLD に変換できます。ただし、元の設計の TTL(ロジック)とのタイミングのづれが多少起こる場合があります。ほとんどの場合、TTL 設計と PLD との間の違いは回路内の伝播遅れだけです。

例 3 - 2 ビットカウンタ

この例では、D タイプのフリップフロップを使用する 2 ビットカウンタのインプリメントを示します。

図 5.4 にカウンタのタイミング図を示します。

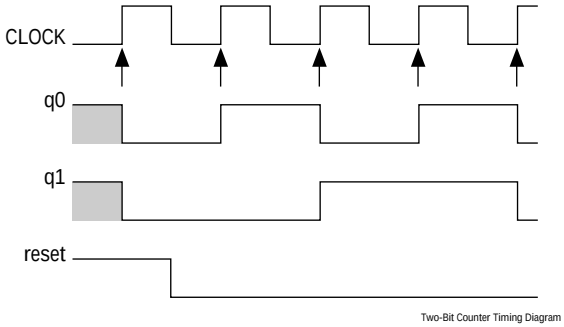


図 5.4 2 ビットカウンタのタイミング図

矢印で示されるように、レジスタはクロック信号の立ち上がりエッヂでクロックされます。

以下に、2 ビットカウンタの(Advanced PLD パッケージにある FLOPS.PLD)CUPL ソースファイルを示します。

FLOPS.PLD

```
Name            Flops;
Partno          CA0002;
Revision        02;
Date            07/16/87;
Designer        G. Woolheiser;
Company         ATI;
Location        None;
Assembly        None;
/*************************************************************/
/*  This example demonstrates the use of D-type flip-flop   */
/*  to implement a two bit counter using the following      */
/*  timing diagram.                                         */
/*                ___     ___     ___     ___     ___       */
/*  clock  |___|   |___|   |___|   |___|   |___|           */
/*          _____           _________       _________       */
/*  q0      /// |_________|         |_________|         |__   */
/*          _____                   _____________________   */
/*  q1      /// |___________________|                     |__ */
/*                                                          */
/*                                                          */
/*          __________                                      */
/*  reset             |_____________________________________ */
/*                                                          */
/*************************************************************/
/*      Target Devices: PAL16R8, PAL16RP8, GAL16V8          */
/*************************************************************/

Pin 1 = clock;
Pin 2 = reset;

/*  Outputs: define outputs and output active levels   */

Pin 17 = q0;
```

```
Pin 16 = q1;

/*  Logic:  two bit counter using expanded exclusive ors */
/*          with d-type flip-flop                        */
qo.d = !reset & (!q0 & !q1
              # !q0 & q1);
q1.d = !reset & (!q0 & q1
              # q0 & !q1);
/* ANDed !reset defines a synchronous register reset */
```

ファイルの最初の部分に、ファイルの管理情報や設計されている機能に関する情報、互換性のある PLD の情報が記述されます。

ピン宣言は、設計ダイアグラムの入力と出力に対応して行われます。

ファイルの Logic セクションでは、カウンタの記述が行われます。q0 の式は、q0 がアサートされる時を定義するために記述されます。すなわち、クロックの立ち上がりエッヂの前の状態を定義します。

!reset タームは q0 と q1 の式で使用され回路を初期化し、同期リセットを実行します。電源の投入時、タイミング図の DONT CARE スラッシュで示されるように、レジスタはハイかローの状態です。リセット信号が始めアサートされます。!reset を各変数の式へ AND することでも、電源の投入の状態ではありません。したがって、レジスタはセットされません。電源投入プロセスが完了してリセット信号が LO(偽)に戻ると、!reset は真になり回路内のレジスタに影響を与えなくなります。

式の中の.d 拡張子は D タイプのフリップフロップを示します。しかし、出力がフィードバックに使用される場合、.d 拡張子は削除されます。例えば、q0 は q1 へフィードバックされる場合、式は以下の様に記述することができます。

```
q1.d = q0 & !reset ;
```

以下のようには記述できません。

```
q1.d = q0.d & !reset ;
```

または

```
q1.d = q0.dq & !reset ;
```

例 4 - デカードアップ/ダウンカウンタ

この例では、同期クリア機能を持つ 4 ビットのアップ／ダウンデカードカウンタを示します。また、このカウンタには、多重カスケードデバイスの非同期リップルキャリー出力があります。カウンタを実行するソースファイルは CUPL ステートマシンシンタクスを使用します。

図 5.5 にカウンタ設計とその図を示します。

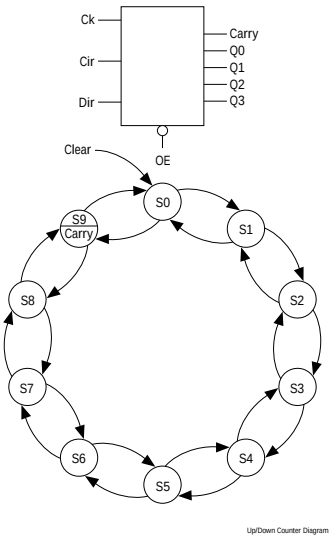


図 5.5 アップ／ダウンカウンタダイアグラム

入力信号 dir により、カウントの方向が決まります。dir がハイの場合、クロック毎に 1 ずつカウントダウンされ、dir がローの場合、クロック毎に 1 ずつカウントアップされます。clr 信号により、同期リセットが行われます。

以下に設計を実行する(Advanced PLD パッケージの COUNT10.PLD)ファイルを示します。

COUNT10.PLD

```
Name            Count10;
Partno          CA0018;
Revision        02;
Date            07/16/87;
Designer        Kahl;
Company         ATI;
Location        None;
Assembly        None;
Device          p16rp4;

/*****************************************************************/
/*                       Decade Counter                          */
/* This is a 4-bit up/down decade counter with synchronous       */
/* clear capability. An asynchronous ripple carry output is      */
/* provided for cascading  multiple devices. CUPL state machine */
/* syntax is used                                                */
/*****************************************************************/
/* Allowable Target Device Types: PAL16RP4, GAL16V8, EP300       */
/*****************************************************************/

/**  Inputs  **/
Pin 1 = clk;                /* counter clock              */
Pin 2 = clr;                /* counter clear input        */
```

```
Pin 3 = dir;                /* counter direction input    */
Pin 11 = !oe;               /* Register output enable     */

/*  Outputs   */
Pin [14..17] = [Q3..0];     /* counter outputs            */
Pin 18 = carry;             /* ripple carry out           */

/*  Declarations and Intermediate Variable Definitions     */
field count = [Q3..0];      /* declare counter bit field  */
$define S0 'b'0000
$define S1 'b'0001
$define S2 'b'0010
$define S3 'b'0011
$define S4 'b'0100
$define S5 'b'0101
$define S6 'b'0110
$define S7 'b'0111
$define S8 'b'1000
$define S9 'b'1001
field node = [clr,dir];     /* declare filed node control */
up = mode:0;                /* define count up mode       */
down = mode:1;              /* define count down mode     */
clear = mode:[2..3];        /* define count clear mode    */

/* Logic Equations */
sequence count {      /*     free running counter     */

present S0          if up          next S1;
                    if down        next S9;
                    if clear       next S0;
present S1          if up          next S2;
                    if down        next S0;
                    if clear       next S0;
present S2          if up          next S3;
                    if down        next S1;
                    if clear       next S0;
present S3          if up          next S4;
                    if down        next S2;
                    if clear       next S0;
present S4          if up          next S5;
                    if down        next S3;
                    if clear       next S0;
present S5          if up          next S6;
                    if down        next S4;
                    if clear       next S0;
present S6          if up          next S7;
                    if down        next S5;
                    if clear       next S0;
present S7          if up          next S8;
                    if down        next S6;
                    if clear       next S0;
present S8          if up          next S9;
                    if down        next S7;
                    if clear       next S0;
present S9          if up          next S0;
                    if down        next S8;
                    if clear       next S0;
out           carry;          /* assert carry output */
```

ファイルの最初の部分にはファイルの管理情報や設計の機能情報、互換性のある PLD の情報が記述されます。

設計ダイアグラムの入力と出力に対応するピン宣言が行われます。

Decralations と Intermediate Variable Definitions セクションには、表記を簡単にするための宣言が記述されます。

名前 count は出力変数 Q3 や Q2、Q1、Q0 に割り付けられます。

$DEFINE 命令を使用して、ステートマシン出力を表わす 10 個の 2 進数ステートに名前が割り付けられます。対応する 2 進数値を表わすためにそのステート名は論理式の中で使用されます。

FIELD キーワードを使用して、clr と dir 入力を mode と呼ばれるセットに結合します。mode は以下の式により定義されます。

```
up = mode:0;
down = mode:1;
clear = mode [2..3];
```

mode は入力 clr と dir を表します。従って、上記の 3 つの式は以下の式と等価です。

```
up = !clr & !dir ;
down = !clr & dir ;
clear = (clr & !dir) # (clr & dir) ;
```

3 つのモードは以下のように定義されます。

up	dir と clr 入力は両方ともアサートされません。
down	dir 入力はアサートされ clr 入力はアサートされません。
clear	clr 入力はアサートされ dir はアサートされるかされないかのどちらかです。

Logic Equations セクションには、カウンタの状態を表わすステートマシンシンタクスが記述されます。最初の行で、SEQUENCE キーワードにより、count(すなわち、Q3 や Q2、Q1、Q0)がステート値を適用する出力とされます。

3 つのモードのそれぞれの現在の状態から次の状態への遷移を指示するために条件ステートメントが記述されます。例えば、現在の状態が S4 の時、モードが up の場合、counter は S5 になり、モードが down の場合、counter は S3 になります。そして、モードが clear の場合、counter は S0 になります。例 4 で示すように、ステートマシンシンタクスの良い点は、設計の動作を分かりやすくに記述できる点です。

例 4 では、clr 信号の結果のステート 0(2 進数値 0000)が定義されます。ステート 0 で別の値にならないように有効な 0000 をすべての設計に設定することを推奨します。例えば、この設計では、電源投入時に 16 進数の A-F のような定義されていない状態になる場合、式に記述された条件がすべて合わないので、ステートはステート 0(hex 値 0000)になります。0000 が有効なステートとして定義されていない場合、カウンターはステート 0 のままです。

以下にこの設計を仮想設計として記述する方法を示します。このファイルは同じファイルですが、仮想設計とデバイスで使用する設計との違いを表わすための変更が必要です。

COUNT10.PLD

```
Name        Count10;
Partno      CA0018;
Revision    02;
Date        07/16/87;
Designer    Kahl;
Company     ATI;
Location    None;
Assembly    None;
Device      VIRTUAL;
```

```
/*************************************************************/
/*                  Decade Counter                           */
/* This is a 4-bit up/down decade counter with synchronous   */
/* clear capability. An asynchronous ripple carry output is  */
/* provided for cascading multiple devices. CUPL state machine */
/* syntax is used                                            */
/*************************************************************/
/* Allowable Target Device Types: PAL16RP4, GAL16V8, EP300   */
/*************************************************************/

/**  Inputs  **/
Pin  = clk;               /* counter clock              */
Pin  = clr;               /* counter clear input        */
Pin  = dir;               /* counter direction input    */
Pin  = !oe;               /* Register output enable     */

/**  Outputs  **/

Pin  = [Q3..0];           /* counter outputs            */
Pin  = carry;             /* ripple carry out           */

/** Declarations and Intermediate Variable Definitions    **/
field count = [Q3..0];     /* declare counter bit field  */
$define S0 'b'0000
$define S1 'b'0001
$define S2 'b'0010
$define S3 'b'0011
$define S4 'b'0100
$define S5 'b'0101
$define S6 'b'0110
$define S7 'b'0111
$define S8 'b'1000
$define S9 'b'1001
field node = [clr,dir];     /* declare filed node control  */
up = mode:0;                /* define count up mode        */
down = mode:1;              /* define count down mode      */
clear = mode:2..3];         /* define count clear mode     */
/* Logic Equations */
sequence count {            /*     free running counter    */

present S0          if up         next S1;
                    if down       next S9;
                    if clear      next S0;
present S1          if up         next S2;
                    if down       next S0;
                    if clear      next S0;
present S2          if up         next S3;
                    if down       next S1;
                    if clear      next S0;
present S3          if up         next S4;
                    if down       next S2;
                    if clear      next S0;
present S4          if up         next S5;
                    if down       next S3;
                    if clear      next S0;
present S5          if up         next S6;
                    if down       next S4;
```

```
                    if clear      next S0;
present S6          if up         next S7;
                    if down       next S5;
                    if clear      next S0;
present S7          if up         next S8;
                    if down       next S6;
                    if clear      next S0;
present S8          if up         next S9;
                    if down       next S7;
                    if clear      next S0;
present S9          if up         next S0;
                    if down       next S8;
                    if clear      next S0;
out           carry;         /* assert carry output */
```

CUPL プリプロセッサの機能を使用するとこの PLD をさらに短くすることもできます。以下にファイルの大きさを小さくする$REPEAT 構造を使用して同じファイルを記述する方法を示します。

```
Name          Count10;
Partno        CA0018;
Revision      02;
Date          07/16/87;
Designer      Kahl;
Company       ATI;
Location      None;
Assembly      None;
Device        VIRTUAL;

/*************************************************************/
/*                  Decade Counter                           */
/* This is a 4-bit up/down decade counter with synchronous */
/* clear capability. An asynchronous ripple carry output   */
/* is provided for cascading  multiple devices.             */
/* CUPL state machine syntax  is used                       */
/*************************************************************/
/* Allowable Target Device Types: PAL16RP4, GAL16V8, EP300 */
/*************************************************************/

/**  Inputs  **/
Pin  = clk;               /* counter clock              */
Pin  = clr;               /* counter clear input        */
Pin  = dir;               /* counter direction input    */
Pin  = !oe;               /* Register output enable     */

/**  Outputs  **/

Pin  = [Q3..0];           /* counter outputs            */
Pin  = carry;             /* ripple carry out           */

/*  Declarations and Intermediate Variable Definitions     */

field count = [Q3..0];       /* declare counter bit field  */
field node = [clr,dir];      /* declare filed node control */
up = mode:0;                 /* define count up mode       */
down = mode:1;               /* define count down mode     */
clear = mode:2..3];          /* define count clear mode    */
```

```
/* state machine description */
sequence count {
present 0
    if up & !clear   next 1;
    if down & !clear next 9;
    if clear         next 0;

$REPEAT i=[1..9]
present i
    if up & !clear   next {(i+1)%10};
    if down & !clear next {(i-1)%10}
    if clear         next 0;
$REPEND
```

このバリエーションでは、代わりに番号をそのままに使用するので$DEFINE 命令を削除しました。最も大きな変更は、ステートを 1 つ 1 つ定義する代わりに、$REPEAT ループを使用してほとんどのステートを定義していることです。このようなことができる理由は、これらのステートが次に進むステート以外は同じだからです。ステート 0 を単独で定義し、それからその他のステートを$REPEAT ループで定義していることに注意して下さい。$REPEAT ループは、コンパイル時に展開され各ステートが定義されます。次のステートで示される命令は繰り返し変数 i から計算されます。ループ内では、i は現在のステートの番号を示します。従って、次のステートは i+1 です。これは、最後のステート以外はすべてのステートで成り立ちます。最後のステートでは、ステートマシンはステート 0 へ進む必要があります。これを実行するために、次のステートを計算する式は(i+1)%10 となります。この式は、i+1 を 10 で割ったあまりを表わします。番号 10 はステートの番号を表わします。従って、ステート 9 の場合、次のステートは(9+1)%10=0 となり次のステートは 0 になります。同じような状態は、以前の状態を計算する場合に起こります。ステート 0 が独立して定義されていることに注意して下さい。これは、$REPEAT 変数は正の値しか扱うことができないためです。$REPEAT ループの中でステート 0 を定義すると、次のステートを-1 とする可能性があり、この場合、コンパイラーがは想できない結果を生成します。

例 5 - 7 セグメントのディスプレイデコーダ

この例では、アノードがコモンの LED を駆動する 16 進数から 7 セグメントへのデコーダを示します。この設計には、数値の先頭のゼロを表示しないようにするリップルブランキング入力と桁の表示を簡単にするリップルブランキング出力が組み込まれます。

図 5.6 にセグメントディスプレイデコーダを示します。

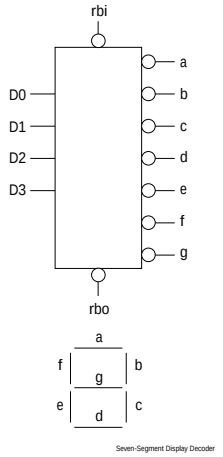


図 5.6　7 セグメントのディスプレイデコーダ

a-g のラベルが付けられた表示のセグメントは図の出力に対応します。

以下にソースファイル(Advanced PLD パッケージの HEXDISP.PLD)を示します。

HEXDISP.PLD

```
Name                Hexdisp;
Partno              CA0007;
Revision            02;
Date                07/16/87;
Designer            T. Kahl;
Company             ATI;
Location            None;
Assembly            None;

/*************************************************************/
/*                                                  a        */
/* This is a hexadecimal-to-seven-segment          -----     */
/* decoder capable of driving  common-anode       |     |    */
/* LEDs. It incorporates both a ripple-          f|     |b   */
/* blanking input (to inhibit displaying          |  g  |    */
/* leading zeroes) and a ripple blanking output    -----     */
/* to allow for easy cascading of digits          |     |    */
/*                                               e|     |c   */
/*                                                |     |    */
/*                                                 -----     */
/*                                                  d        */
/*************************************************************/
/* Allowable Target Device Types: 32 x 8 PROM (82S123 or     */
/* equivalent                                                */
/*************************************************************/

/**  Input group (Note this is only a comment)              **/
pin [10..13] = [D0..3];     /* data input lines to display    */
pin 14 = !rbi;                   /* ripple blanking input     */
```

```
/** Output Group  ( Note this is only a comment)             **/

pin [7..1] = ![a,b,c,d,e,f,g];   /* Segment output lines      */
pin 9      = !rbo;               /* Ripple Blanking output    */

/**  Declarations and Intermediate Variable Definitions       */
field data = [D3..0];            /* hexadecimal input field   */
field segment=[abcdefg];         /* Display segment field     */
$define ON  'b'1           /* segment lit when logically"ON"  */
$define OFF 'b'0         /* segment dark when logically "OFF" */
```

***ディスプレイデコーダのソースファイル**(HEXDISP.PLD) 1/2*

ファイルの最初の部分にはファイルの管理情報や設計の機能情報、互換性のあるPLDの情報が記述されます。

設計ダイアグラムの入力と出力に対応するピン宣言が行われます。

Declarations と Intermediate Variable セクションでは、入力ピンを名前の付けられたデータとして、そして出力ピンを名前の付けられたセグメントとしてグループ化するためにフィールドの割り付けが行われます。ON と OFF はそれぞれ 2 進数の 1 と 0 として定義されます。

```
/**  Logic Equations   **/
     /*       a     b     c     d     e     f     g  */
segment =
/* 0 */     [ ON,   ON,   ON,   ON,   ON,   ON,  OFF] & data:0 & !rbi
/* 1 */  #  [OFF,   ON,   ON,  OFF,  OFF,  OFF,  OFF] & data:1
/* 2 */  #  [ ON,   ON,  OFF,   ON,   ON,  OFF,   ON] & data:2
/* 3 */  #  [ ON,   ON,   ON,   ON,  OFF,  OFF,   ON] & data:3
/* 4 */  #  [OFF,   ON,   ON,  OFF,  OFF,   ON,   ON] & data:4
/* 5 */  #  [ ON,  OFF,   ON,   ON,  OFF,   ON,   ON] & data:5
/* 6 */  #  [ ON,  OFF,   ON,   ON,   ON,   ON,   ON] & data:6
/* 7 */  #  [ ON,   ON,   ON,  OFF,  OFF,  OFF,   ON] & data:7
/* 8 */  #  [ ON,   ON,   ON,   ON,   ON,   ON,  OFF] & data:8
/* 9 */  #  [ ON,   ON,   ON,   ON,  OFF,   ON,   ON] & data:9
/* A */  #  [ ON,   ON,   ON,  OFF,   ON,   ON,   ON] & data:A
/* B */  #  [OFF,  OFF,   ON,   ON,   ON,   ON,   ON] & data:B
/* C */  #  [ ON,  OFF,  OFF,   ON,   ON,   ON,  OFF] & data:C
/* D */  #  [OFF,   ON,   ON,   ON,   ON,  OFF,   ON] & data:D
/* E */  #  [ ON,  OFF,  OFF,   ON,   ON,   ON,   ON] & data:E
/* F */  #  [ ON,  OFF,  OFF,  OFF,   ON,   ON,   ON] & data:F;

rbo = rbi & data:0;
```

***ディスプレイデコーダのソースファイル**(HEXDISP.PLD) 2/2*

論理式は、関数テーブルとして設定され、各入力パターンに応じたセグメントが記述されます。コメントにより関数テーブルのヘッダーが作成され、出力セグメントは表の一番上に表示され、入力番号は表の横に縦方向に表示されます。

表の各行には、デコードされた hex 値と hex 値でオンオフする表示のセグメントが記述されます。例えば、入力値が 4 の行は以下のように記述されます。

```
[OFF, ON, ON, OFF, OFF, ON, ON] & data:4
```

関数テーブルのフォーマットで記述すると、設計の内容が式で記述するよりも解りやすくなります。すなわち、上記の例では、入力値 4 により、セグメント a がオフ、b がオン、c がオンとなることを表わします。

例 6 - ロードリセット機能付きの 4 ビットカウンタ

この例では、ロードとリセットのできるカウンタを示します。設計は仮想デバイス VirtualDevice を使用して行いますが、4 つ以上のレジスタのある PLD ならば簡単にインプリメントすることができます。

```
Name        Counter;
Partno      FL1201;
Revision    01;
Date        08/26/91;
Designer    RGT;
Company     LDI;
Location    None;
Assembly    None;
Device      VIRTUAL;

/*********************************************************/
/*                    4-bit counter                      */
/*********************************************************/

/**   inputs   **/
PIN = clk;            /* clock signal for registers   */
PIN = load;           /* load signal                  */
PIN = !ClrFlag;
PIN = [LoadPin0..3];  /* pins from which to load data */

/**   outputs   **/
PIN = [CountPin0..3];

/**   intermediate variables and fields   **/
field STATE_BITS = [Count0..3];
field LOAD_BUS = [LoadPin0..3];

/**   state machine definition   **/
SequenceD STATE_BITS {

/* build a repeated loop for the states */
$REPEAT i = [0..15]
    Present 'h'{i}

/* go to state 0 if clear signal is true if the load signal */
/* is false go to the next state. Note that the next state  */
/* is (current state + 1) modulo 16.                        */
/* This causes the counter to wrap back to 0 when in the    */
/* last state                                               */

If !load Next 'h'{(i+1)%16};
If !load & ClrFlag Next'b'0;
$REPEND

/* Add the load capability by using the APPEND statement.       */
/* This has the effect of adding more inputs to the OR gate for */
/* this output. This equation states that if the load signal    */
/* is true then the counter registers are loaded with data from */
/* the load pins. This is why 'load' was used in the equations  */
/* for the 'IF' statements in the state definitions.            */
```

```
APPEND STATE_BITS.d = load & LOAD_BUS;
```

これは仮想設計のため、ピンの番号付けは無視されます。使用される信号はピン番号なしで宣言されます。

load 信号が宣言されると、ロードピンの値がステートビットレジスタへ送られます。Clear 信号が宣言されると、ステートマシンは強制的にステート 0 になります。

$REPEAT ループは[0..15]で定義されます。ループの内部は、現在のステートが h{i}で定義されます。これにより番号は hex 番号として評価されます。従って、ステートが 16 個あるステートマシンのは、16 進数で 0-F となります。h が取り除かれると、ステートは 10 進数で 0-15 になります。いずれにしても、コンパイラーはこれらのステートを hex として解釈し、ステート A-F は定義されません。さらに、ステート 10(HEX)は存在しないため、今回扱うステートビットは 4 ビットです。

IF 命令がすべて!load で AND されていることに注意して下さい。これは、ロード機能の優先順位を一番高くしたいためです。これにより、ロード信号とその他の信号が同時に朝ートされた時に起こる競合を取り除くことができます。

次のステートは、(present_state+1)を 16 で割った余りで計算されます。これにより、最後のステートからステート 0 へ戻ることができます。ステートマシンのステートの数が 16 なので 16 の余りを使用します。

この設計の最後の部分で設計にロード機能が追加されます。ここでは、APPEND 命令を使用します。APPEND 命令により指定された変数へ OR された式が与えられます。結局このステートマシン全体は、各ステートビットの組みの式になります。out は入力ピンからの値がロードされるレジスタに他の条件を追加するためのものです。IF 命令のすべてに!load が使用されていることを忘れないで下さい。IF 命令により生成された式が APPEND 命令と矛盾しないことを確認して下さい。

```
CountPin0.d = !load & ???.....
            # !load & ???.....;
```

*APPEND **の前の式***

```
CountPin0.d = !load & ???.....
            # !load & ???.....
            # load & LoadPin0;
```

*APPEND **の後の式***

練習のように、この例にアップダウン機能を追加してみて下さい。また、実際のデバイスにこの設計をインプリメントしてみて下さい。

PLD のシミュレーション

このセクションでは、どのようにテスト仕様ソースファイルを作成しプログラマブルロジックデバイスのシミュレーションを実行するかを説明します。テストベクタにより、出力を入力の関数として定義し PLD の関数の動作を指定します。テストベクタは、プログラム前のデバイスロジックのシミュレーションと論理がプログラムされた後のデバイスのテスト関数とに使用されます。Advanced PLD シミュレーターは、JEDEC 互換のテストベクタを生成します。このテストベクタは、コンパイル中に作成される.JED ファイルに添付されます。

シミュレーターへの入力

テスト仕様ソースファイル(ファイル名.SI)は、シミュレーターへの入力です。このファイルには、回路でのデバイスに機能的に必要な情報が記述されています。

ソースファイルに入力される入力ピンの刺激や出力ピンのテストは、PLD ソースファイルで計算される実際の値と比較されます。計算された値は、アブソリュート ABS フォーマットが指定される場合、コンパイル中に作成されるアブソリュートファイル(ファイル名.ABS)に保存されます。アブソリュートファイルは、シミュレーションを実行する前にコンパイル中に作成する必要があります。

シミュレーターからの出力

シミュレーターの出力は、

- シミュレーションリスティングファイル
- JEDEC フォーマットのダウンロード可能なヒューズリンクファイルに添付されるベクタです。

シミュレーションリスティングファイル(ファイル名.SO)にはシミュレーションの結果が保存されます。ファイル名は、入力のテスト仕様ファイルと同じ名前です。

ヘッダー情報は、適当に印が付けられたヘッダーエラーと一緒にリスティングファイルに記述されます。失敗した出力テストには、実際の出力(シミュレーターが決めた)値と一緒にフラグが付けられます。エラーの各変数は予測された(ユーザが与えた)値と一緒に表示されます。無効または予測されないテスト値は適当なエラーメッセージと一緒に表示されます。

シミュレーションリスティングファイルは ASCII です。したがって、テキストエディターでオープンすることができます。EDA/クライアントの Waveform エディターで.SO ファイルを開くと、シミュレーション結果を、一連の波形として表示することもできます。波形の表示については、Advanced PLD の紹介のセクションを参照して下さい。

シミュレーターは、コンパイル中に作成された既存の JEDEC ヒューズリンクファイル(ファイル名.SI)にテストベクタを追加します。Configure Advanced PLD ダイアログボックスの Format タブの JEDEC オプションをイネーブルにして、このファイルを作成して下さい。

➔ シミュレーターは、複数のデバイスのシミュレーションをサポートしていません。この場合、デバイスファイルの最初のファイルがシミュレーションされます。

シミュレーションテストベクタファイルの作成

テスト仕様ファイル(ファイル名.SI)は ASCII ファイルです。テキストエディターを使用して作成して下さい。ファイル名は、対応する CUPL 論理記述ソースファイルと同じ名前です。ただし、ファイルの拡張子は違います。以下の情報をテスト仕様ファイルに記述して下さい。

- ヘッダー情報
- コメント
- 変数オーダー
- 基本セット
- テストベクタ
- シミュレーターディレクティブ

➔ シミュレーションテストベクタファイルの作成については Advanced PLD セクションを参照して下さい。

ヘッダー情報

入力されるヘッダー情報は、対応する CUPL 論理記述ファイル(.PLD)の情報と一致している必要があります。ヘッダー情報が一致していない場合、ワーニングメッセージが表示され、論理式の状態がテスト仕様ファイルの現在のテストベクタと一致していないことが知らされます。

表 6.1 にヘッダー情報に使用されるキーワードの一覧を示します。

PARTNO
NAME
REVISION
DATE
DESIGNER
COMPANY
ASSEMBLY
LOCATION
DEVICE
FORMAT

表 6.1 シミュレーターソースファイルヘッダー情報

テスト使用ファイルを作成する場合、対応する CUPL ソースファイルの内容をコピーして作成すると間違いの無いヘッダー情報を記述できます。

デバイスの指定

シミュレーターは、デバイスライブラリーファイル(CUPL.DL)でデバイス情報にアクセスします。ライブラリーには、各デバイスの内部アーキテクチャーやピンの数、使用できるレジスタのタイプ、(レジスタードとノンレジスタードピンなどの)論理特性、フィードバック機能、レジスタのパワーオン状態、レジスタ制御機能などの物理特性が記述されています。

デバイスニーモニックを使用してターゲットデバイスを参照して下さい。各ニーモニックは、デバイスファミリプリフィックスと業界標準部品番号プリフィックスで構成されます。表 10-2 にデバイスニーモニックプリフィックスを示します。

プリフィックス	デバイスファミリ
EP	イレーサブルプログラマブルロジックデバイス(EPLD)
G	ジェネリックアレイロジック(GAL)
F	フィールドプログラマブルロジックアレイ(FPLA)
F	フィールドプログラマブルゲートアレイ(FPGA)
F	フィールドプログラマブルロジックシーケンサ(FPLS)
F	フィールドプログラマブルシーケンスジェネレータ(FPSG)
P	プログラマブルロジックアレイ(PAL)

P	プログラマブルロジックデバイス(PLD)
P	プログラマブルエレクトリカリーイレーサブルロジック(PEEL)
PLD	スードロジカルデバイス
RA	バイポーラプログラマブルリードオンリメモリ(PROM)

表 6.2 デバイスニーモニックプリフィックス

PAL10L8 のデバイスニーモニックは、P10L8 です。また、82S100 のデバイスニーモニックは、F100 です。バイポーラ PROM では、サフィックスがアレイサイズを表します。1024X8 のバイポーラ PROM のデバイスニーモニックは、10 本のアドレス入力ピンと 8 本のデータ出力ピンがあるので RA10P8 になります。

ターゲットデバイスは、SI ファイル上で直接記載するか、Configure Advanced PLD ダイアログボックスの Options タブで指定することができます。

コメント

テスト仕様ファイルの任意の位置にコメントを置くことができます。コメントを仕様して仕様ファイルの内容やテストベクタの関数の説明を記述することができます。コメントは、スラッシュ－アスタリスク(/*)で始まり、アスタリスク－スラッシュ(*/)で終わります。複数行に渡りコメントを設定することができます。コメントをネストすることはできません。

ステートメント

シミュレーターには、キーワード ORDER,BASE,VECTORS があり、これらを使用してシミュレーター出力や出力の表示の仕方をソースファイルに記述します。以下のセクションで、CUPL キーワードによりステートメントをどのように記述するかを説明します。

ORDER ステートメント

ORDER キーワードを使用して、シミュレーションテーブルで使用される変数の一覧を表示して下さい。また、それらがどのように表示されるかを定義して下さい。通常、変数名は、CUPL 論理記述ファイルと同じ名前が使用されます。ORDER の後ろにコロンを記述し、リストの変数をコンマで区切って下さい。さらに、リストの最後にセミコロンを記述して下さい。以下に ORDER 命令の使用例を示します。

```
ORDER: inputA, inputB, output ;
```

セミコロンの左で実際に使用された変数だけが表示されます。

変数名の極性は、CUPL 論理記述ファイルで宣言されたものと違います。これにより、アクティブハイのシミュレーションベクタを使用してアクティブロー出力のシミュレーションができます。変数名は、任意の順番で入力できます。シミュレーターは、デバイスの JEDEC ダウンロードフォーマット応じてシミュレーション結果の極性や変数の適当な順番を自動的に作成します。インデックス付き変数を ORDER 命令で使用する場合、リスト表記で記述して下さい。しかし、ORDER 命令はすでにリスト形式になっているので、ORDER の組みを囲むかぎ括弧([])は必要ありません。以下に 2 つの等価な ORDER 命令の例を示します。最初の命令は、変数をすべて記述し、2 番目はリスト形式で記述しています。

```
ORDER: A0, A1, A2, A3, SELECT, !OUT0, !OUT1;
ORDER: A0..3, SELECT, !OUT0..1 ;
```

リスト表記フォーマットでは、最初のインデックス付き変数(上記例の!OUT0)の極性により、リスト全体の極性が決まります。CUPL 論理記述ファイルで宣言されるビットフィールドは、1 つの変数名で参照することができます。また、ビットフィールドは、FIELD 宣言命令を使用して、シミュレーターのテスト仕様ファイルで宣言することもできます。 FIELD 命令は、ORDER 命令より先に記述する必要があります。

ORDER 命令を使用して、シミュレーターリスティングファイルのシミュレーション結果ベクタのフォーマットを指定できます。デフォルトでは、変数値はコロンとコロンの間にスペースがありません。ORDER 命令の例を以下に示します。

```
ORDER: clock, input, output ;
```

により出力ファイルに以下のように記述されます。

```
0001: C0H
0002: C1L
```

%記号と 1 から 80 までの 10 進数を使用してコロンの間にスペースを挿入できます。例えば、以下の ORDER 命令により

```
ORDER: clock, %2, input, %2, output ;
```

出力ファイルに以下のように記述されます。

```
 0001: C 0 H
 0002: C 1 L
```

➔ ORDER 命令は、セミコロンで終了する必要があります。

ORDER 命令の中にダブルクオート(",)で囲まれた文字列を記述すると、出力ファイルにテキストを記述できます。例えば、以下の ORDER 命令により

```
 ORDER: "Clock is ", clock,
" and input is ", input,
" output goes ", output ;
```

以下の結果が出力ファイルに記述されます。

```
0001: Clock is C and input is 0 output goes H
0002: Clock is C and input is 1 output goes L
```

多重 ORDER 命令

いろいろな ORDER 命令を SI ファイルに定義することができます。例えば、TEST.SI ファイルに以下の記述がある場合、

```
Name         test;
Partno       XXXXX;
Date         XX/XX/XX;
Revision     XX;
Designer     XXXXX;
Company      XXXXX;
Assembly     XXXXX;
Location     XXXXX;
Device       g16v8;

Order: A, %1, B, %1, X, %1, Y;
Vectors:
 0 0 H L
 0 1 H H
 1 0 H H
 1 1 L L
 0 X H X
 X 0 H X
 1 X X X
 X 1 X X
Order: A, B, X;
Vectors:
 0 0 H
 0 1 H
```

```
 1 0 H
 1 1 L
 0 X H
 X 0 H
 1 X X
 X 1 X
```

TEST.SO ファイルは以下のようになります。

```
CSIM: CUPL Simulation Program
Version 4.2a Serial# ...
Copyright (c) 1996 Protel International
CREATED Wed Dec 04 02:14:12 1991
LISTING FOR SIMULATION FILE: test.si
1: Name      test;
2: Partno    XXXXX;
3: Date      XX/XX/XX;
4: Revision  XX;
5: Designer  XXXXX;
6: Company   XXXXX;
7: Assembly  XXXXX;
8: Location  XXXXX;
9: Device    g16v8;
10:
11: Order: A, %1, B, %1, X, %1, Y;
12:
=====================
      A B X Y
=====================
0001: 0 0 H L
0002: 0 1 H H
0003: 1 0 H H
0004: 1 1 L L
0005: 0 X H X
0006: X 0 H X
0007: 1 X X X
0008: X 1 X X
25: Order: A, B, X; 26:
=====================
        ABX
=====================
0010: 00H
0011: 01H
0012: 10H
0013: 11L
0014: 0XH
0015: X0H
0016: 1XX
0017: X1X
```

BASE ステートメント

多くの場合、(FIELD 変数を除く)ORDER 命令の各変数には、出力ファイルのテストベクタテーブルに記述される 1 つのキャラクタテスト値があります。多重テストベクタ値を引用符で囲まれた番号で表わすことができます。入力値には、シングルクオートを使用し、出力値にはダブルクオートを使用して下さい。BASE ステートメントを入力し、引用符で囲まれた数値をどのように展開するかを指定して下さい。BASE ステートメントのフォーマットを以下に示します。

```
BASE: name;
```

ここで

name は 8 進数、10 進数、16 進数のどれかです。

BASE のうしろにはコロンを記述します。

➔ ベースステートメントは、セミコロンで終了して下さい。

引用符で囲まれたテスト値のデフォルトの基数は 16 進数です。BASE ステートメントは ORDER 命令より前に記述する必要があります。

基数が 10 進数または 16 進数の場合、引用符で囲まれた値は 4 桁に展開されます。基数が 8 進数の場合、3 桁に展開されます。例えば、'7'と入力されたテストベクタは以下のようになります。

```
1 1 1                       基数が 8 進数
```

または

```
0 1 1 1                     基数が 10 進数
```

または

```
0 1 1 1                     基数が 16 進数
```

16 進数や 8 進数の複数の桁が引用符の間に入力されます。例えば、'563'は以下のように展開されます。

```
1 0 1 1 1 0 0 1 1           基数は 8 進数
```

または

```
0 1 0 1 0 1 1 0 0 0 1 1     基数は 10 進数
```

または

```
0 1 0 1 0 1 1 0 0 0 1 1     基数は 16 進数
```

引用符で囲まれた値は、他のテスト値で使用することもできます。例えば、基数が 8 進数に設定されている場合

```
"XX"        X X X X X X に展開されます。
"LL"        L L L L L L に展開されます。
"45"        H L L H L H に展開されます。
```

➔ 引用符で囲まれる値に*は使用できません。

FIELD 変数のテスト値は、個別(例えば、001、HHLL)に記述することもできるし、引用符で囲まれた値(例えば、'1'、"C")で記述することもできます。引用符で囲まれた値が使用される場合、値は自動的にフィールドの値に展開されます。例えば、以下のアドレスフィールド

```
FIELD address = [A0..5] ;
```

に以下のテスト値が記述されている場合、

```
/*
   A     A     A     A     A     A
   5     4     3     2     1     0
  -------------------------------- */
   1     1     1     0     0     1
```

上記の値は、引用符で囲まれた 1 つのテスト値、'39'を使用して記述することができます。

VECTORS 命令

VECTORS キーワードを使用して、ベクタテーブルを前もって設定して下さい。以下のキーワードは､1 つのテストベクタまたは引用符で囲まれたテスト値に付けて使用されます。表 10-3 に使用できるテストベクタの値を一覧表示します。

テスト値	説明
0	入力を LO(0 ボルト)に駆動します。(アクティブハイの入力を否定します。)
1	入力を HI(+5 ボルト)に駆動します。(アクティブハイ入力を宣言します。)
C	(CLOCK)入力を LO、HI、LO に駆動します。
K	(CLOCK)入力を HI、LO、HI に駆動します。
L	テスト出力 LO(0 ボルト)(アクティブハイ出力を否定します。)
H	テスト出力 HI(+5 ボルト)(アクティブハイ出力を宣言します。)
Z	ハイインピーダンスのテスト出力
X	入力が HI または LO、出力が HI または LO
N	テストされない出力
P	プリロード内部レジスタ(値は!Q に適用されます)
R	ランダム入力の生成
*	Outputs only - 出力のみの場合はシミュレーターによりテスト値が決定され、ベクタ(シングルクオート)に入力されます。
"	引用符で囲まれた入力値は、指定された BASE(8 進数、10 進数、16 進数)で展開されます。有効な値は、0-F と X です。
""	この引用符で囲まれた出力値は、指定された BASE(8 進数、10 進数、16 進数)で展開されます。有効な値は、0-F、H、L、Z、X です。

表 6.3 テストベクタ値

以下にテストベクタ表の例を示します。

```
VECTORS:
0 0 1 1 1 'F' Z "H"  /* テスト出力 HI */
0 1 1 0 0 '0' Z "L"  /* テスト出力 LO */
```

ランダム入力の生成

R ベクタにより、0 や 1 をどこでも出現させることができます。ランダム値があると、ランダム値(0 または 1)は、テストベクタの対応する信号で生成されます。

➔ R を使用してランダム入力値を生成することができます。

An example of R in the SI file:

```
$repeat 10;
C   0 RRR 1RRRRRRR  ********
```

上記の記述により、SO ファイルに以下の結果が出力されます。

```
0035: C 0   000 10001011   HLLLHLHH
0036: C 0   000 11100111   HHHLLHHH
0037: C 0   110 10111101   HHHHLHHL
0038: C 0   111 11000100   HLLLHLLH
0039: C 0   101 10001011   LHLHHHLL
```

```
0040: C 0   101 10000110   LLHHLHLL
0041: C 0   010 10000001   LHHLLLLL
0042: C 0   000 10010000   HLLHLLLL
0043: C 0   001 11110100   LHHHHLHL
0044: C 0   001 10011110   LHLLHHHH
```

Don't Care

他のシミュレーターと違い、Advanced PLD シミュレーターは DONT-CARE(X の状態)を扱うことができます。状態 X により、以下の真理値表に示されるルールに応じて、どの入力が出力に影響を与えているかを特定することができます。

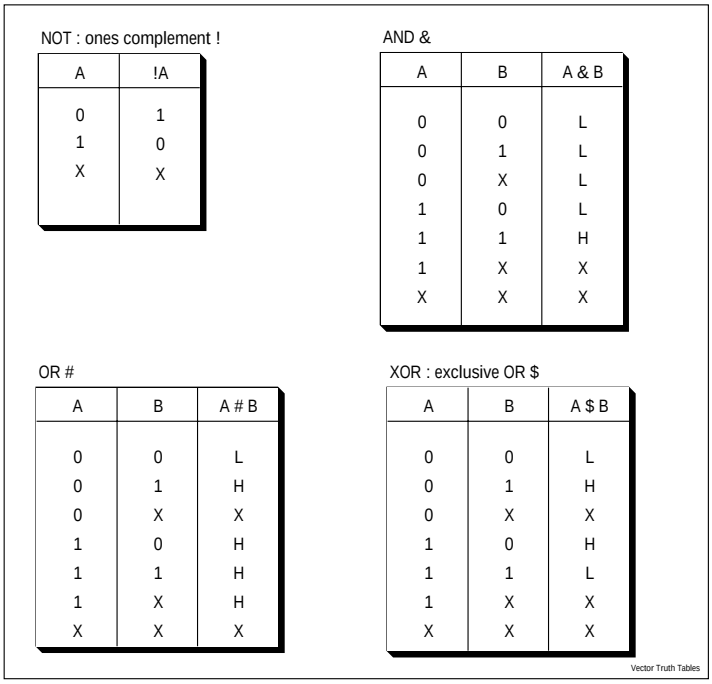


図 6.1 ベクタ真理値表

プリロード

デバイスが TTL レベルのプリロードピンを持っていない場合、レジスタードデバイスのクロックピンテスト値 P を使用して、ステートマシンの内部レジスタやカウンタをの設計をプリロードしたりできます。デバイスプログラマは、管理電圧を使用して実際にレジスタをロードします。デバイスの入力ピンは無視されます。したがって、X で定義されます。レジスタード変数に現れる値は、レジスタの!Q 出力にロードされます。以下に、レジスタ Q 出力とデバイスピンとの間に反転バッファを持つデバイスのアクティブロー出力変数のプリロードシーケンスの例を示します。

```
ORDER: clock, input1, input2 , !output ;
VECTORS:
P X X 1     /* reset flip-flop      */
            /* !Q goes to 1         */
            /* Q goes to 0          */
0 X X H     /* output is HI due  to */
            /* inverting buffer     */
```

➔ シミュレーターでは、プリロード機能を持っていないデバイスでもプリロードテストベクタを作成したりシミュレーションを実行したりできます。しかし、PLD がすべて管理電圧を使用してプリロードを利用できるわけではありません。デバイスの中には、プリロードピンをこの目的のために提供しているものがあります。製造元が違うと特性も違うので、シミュレーターでは、シミュレーション中のデバイスが実際にプリロ

ードできるかどうかは確認できません。プリロード機能を使用する前に、テストされるデバイスが物理的にプリロードできるかどうかを確認する必要があります。

クロック

ほとんどの同期デバイス(出力ピンに接続された共通のクロックを持つレジスタのあるデバイス)では、アクティブハイ(立ち上がりエッヂでトリガーされる)クロックが使用されます。これらのデバイスに対してシミュレーターの動作を適切に行うために、クロックピンには C テスト値(1 または 0 ではなく)を常に使用して下さい。アクティブロー(立ち下がりエッヂでトリガーされる)クロックを使用する同期デバイスでは、K テスト値をクロックピンに使用して下さい。

非同期ベクタ

非同期フィードバックのある回路のテストベクタを記述する場合、一度に 2 つのテストベクタを変更することにより、例外的な結果を生成するスパイク状態を作成することができます。図 6-2 では、3 つの入力[A,B,C]とフィードバックになる 1 つの出力 Y を持つ回路のダイアグラムを示します。

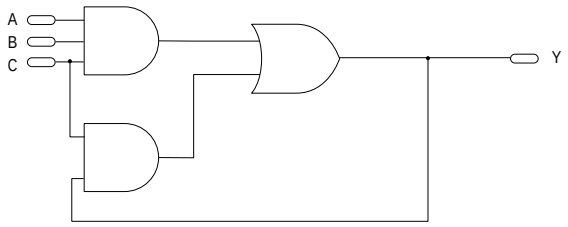


図 6.2 フィードバックを持つ回路

Y での出力の式を以下に示します。

```
Y = A & B & C # C & Y
```

以下のベクタテーブルに、指定された入力値に基づく、予測される出力を示します。

	A	B	C	Y
0001	0	0	0	L
0002	0	1	1	L
0003	1	0	1	L

Vectors Table for Circuit with Feedback

図 6.3 フィードバックのある回路のベクタテーブル

各ベクタで入力の 1 つが 0 であるので、A,B,C で定義される AND ゲートによりロー出力が生成されます。Y 出力からフィードバックされるロー出力は、他の AND ゲートをローに保ちます。したがって、(2 つの AND ゲートの出力により駆動される)OR ゲートと Y の出力は指定されたテストベクタに対してローのままです。

しかし、プログラマーがテストベクタを操作すると、最初のピンから順に値が入力されます。ベクタ間で 2 つのテスト値が変わるので、プログラマーにより(図で"a"のラベルが付けられた)中間結果が作成されます。

	A	B	C	Y
0001	0	0	0	L
0001a	0	1	0	L
0002	0	1	1	L
0002a	1	1	1	H
0003	1	0	1	H

Vectors Table with Intermediate Results

図 6.4 中間結果のあるベクタテーブル

中間結果[0002a]は、出力 Y にハイを生成します。このハイ信号がフィードバックされ、ベクタ[0003]の入力 C に指定される"1"の値に結び付けられ AND ゲートのハイ出力を生成します。そして、OR ゲートや Y 出力もハイになります。このハイ出力は 3 番目のテストベクタで指定される予想されたロー出力と矛盾します。したがって、結果はスパイク状態になります。

テストベクタ間では 1 つの値だけを変更することに注意すれば、上記のようなスパイク状態を避けることができます。また、ソース仕様ファイルで、シミュレーターに中間結果を出力ファイルに出力するように指示する TARCE 値 1,2,3(デフォルトでは 0)を指定することができます。(以下のシミュレーターディレクティブの"TRACE"を参照して下さい。)

I/O ピンシミュレーション

入出力機能やコントロールできる出力イネーブル(OE)をもつ設計のテストベクタを記述する場合、I/O ピンに配置されるテストベクタは、出力イネーブルの値に依存します。出力イネーブルがアクティブの場合、I/O ピンには出力テスト値(L,H,*,...)が必要になります。出力イネーブルがアクティブでなくなると、I/O ピンはハイインピーダンス状態になります。この時、入力テスト値(0,1,...)を I/O ピンに配置してピンを入力ピンとして使用することができます。出力イネーブルが再びアクティブになると、ピンのテスト値により、マクロセルの出力が影響を受けます。

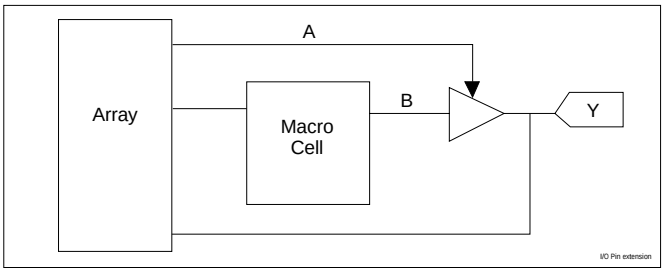


図 6.5 I/O ピンシミュレーション

以下の式は、図 6-5 のブーリアン式を記述したものです。

```
Y = B;
Y.OE = A;
```

A が真の場合、マクロセル(B)の出力がピン(Y)に現れます。A が偽の場合、出力イネーブルはアクティブでなくなりピン(Y)はハイインピーダンス状態になります。出力イネーブルがアクティブでなくなると、入力値をピンに配置することができます。シミュレーションファイルがどのようになるかをここで示します。

```
Order: A, %1, B, %3, Y;
Vectors:
1 0  L    /* OE is ON */
1 1  H
0 0  Z    /* OE is OFF */
0 0  1    /* a valid input value can be  placed on pin Y */
1 0  L    /* OE is ON again */
```

変数宣言 (VAR)

```
Syntax:   VAR <var_name> = <var_list>;
```

<var_name> - 文字までの文字列で文字、数値、アンダースコアを使用できます。数字で終わってはいけません。

<var_list> - (1 つまたは、グループやフィールドの)order 命令からのシンボルリストです。前に宣言された変数でコンマで区切られます。

<var_list> = [!]<field> | [!]<group> | [!]<var> [..[!]<var> | ,<var_list>]

動作:
<var_list>の変数をグループ化します。FIELD 変数と似ています。ただし、この命令は ORDER 命令より前に記述することができません。ORDER 命令と VECTORS 命令の間で使用して下さい。

例:

```
VAR Z = Q7..4;
```

シミュレーターディレクティブ

シミュレーターには、VECTOR 命令の後のファイルの任意の行に記述できる 6 個のディレクティブがあります。ディレクティブ名はすべてダラーマークで始まります。各ディレクティブステートメントはセミコロンで終了して下さい。

$MSG	$SIMOFF
$REPEAT	$SIMON
$TRACE	$EXIT

表 6.4. シミュレーターディレクティブ

$MSG

$MSG ディレクティブを使用して、ドキュメントやフォーマット情報をシミュレーターファイルに記述して下さい。例えば、変数名が一覧表示されるシミュレーター関数テーブルのヘッダーが作成できます。フォーマットを以下に示します。

```
$MSG " 任意の文字列 " ;
```

出力テーブルでは、文字列には引用符(ダブルクオート)はありません。

ブランク行を出力ファイルに挿入できます。例えば、以下のフォーマットによりベクタの間に挿入できます。

```
$MSG "" ;
```

$REPEAT

$REPEAT ディレクティブにより、ベクタを指定された回数だけ繰返すことができます。フォーマットを以下に示します。

```
$REPEAT n ;
```

ここで

n は 1 から 9999 までの 10 進数値です。

$REPEAT ディレクティブに続くベクタが、指定された回数だけ繰返されます。

$REPEAT ディレクティブは、カウンタやステートトランジションに便利です。アスタリスク(*)を使用して、シミュレーションにより与えられる出力テスト値を示します。以下の例

は、CUPL ソースファイルから 2 ビットのカウンタを示し、そのカウンタをテストするために$REPEAT ディレクティブを使用する VECTORS 命令を命令を示します。

コンパイラーから:

```
Q0.d = !Q0 ;
Q1.d = !Q1 & Q0 # Q1 & !Q0 ;
```

シミュレーターで:

```
ORDER: clock, input, Q1, Q0 ;
VECTORS:
0 0 X X       /* power-on condition    */
P X 1 1       /* reset the flip-flops  */
0 0 H H
$REPEAT 4 ;   /* clock 4 times         */
C 0 * *
```

上記のファイルにより以下のテストベクタが作成されます。

```
0 0 X X
P X 1 1
0 0 H H
C 0 L L
C 0 L H
C 0 H L
C 0 H H
```

シミュレーターによりベクタ値の 4 つのセットが与えられます。

$TRACE

$TRACE ディレクティブを使用して、シミュレーターが記述するベクタの情報を設定できます。以下にフォーマットを示します。

```
$TRACE n ;
```

ここで

n は 0 から 4 までの 10 進数値です。

Trace level 0 (デフォルト)追加情報はありません。テストベクタの結果だけが記述されます。ノンレジスタのフィードバックが使用されている場合、フィードバックの出力値はベクタの最初の評価が終わるまで不定です。新しいフィードバック値が出力値を変更する場合、ベクタは再び評価されます。ベクタが安定するまでは、出力はすべて同じである必要があります。

Trace level 1 安定するのに複数の評価パスを必要とすつ任意のベクタ中間結果を出力します。20 以上の評価パスが必要なベクタは、不安定と見なされます。

Trace level 2 レジスタを使用する設計のシミュレーションの 3 フェーズ分を記述します。最初のフェーズは"Before the Clock"で、ノンレジスタードフィードバックを使用する中間ベクタが評価されます。2 番目のフェーズは、"At the Clock"で、レジスタの値がクロックの後すぐに与えられます。3 番目のフェーズは、"After the Clock"で、フィードバックに使用される出力がトレースレベル 1 で評価されます。

Trace level 3 これにより、シミュレーターでできる最高レベルの情報が出力されます。"Before the Clock"、"At Clock"、"After Clock"の各シミュレーションフェーズが出力され、各変数のそれぞれのプロダクトタームが一覧表示されます。AND ゲートの出力値は AND アレイへの入力値と一緒に一覧表示されます。

Trace level 4 これにより、出力バッファの前の論理値を観察することができます。$TRACE 4 を使用して、シミュレーターに出力ピンが真であることをレポートさせたり、入力や埋め

こみノードに"?"を割り付けたりします。組み合わせ出力では、トレースレベル 4 により、OR タームの結果を出力できます。レジスタード出力では、トレースレベル 4 により、レジスタの Q 出力を出力できます。以下の例では、p22v10 を使用しています。

```
pin 1 = CLK;
pin 2 = IN2;
pin 3 = IN3;
....
pin 14 = OUT14;
pin 15 = OUT15;
....
OUT14.D = IN2;
OUT14.AR = IN3;
OUT14.OE = IN4;
```

シミュレーション結果

```
order CLK, IN2, IN3, IN4, OUT14, OUT15;
****** before output buffer ******
???? ..LL...0001:
0011 ..HH........
******before output buffer******
????   HH...0004
C100  ...ZZ.....
```

$EXIT

$EXIT ディレクティブを使用して、任意の場所でシミュレーションを中断することができます。$EXIT ディレクティブ以降のテストベクタは無視されます。このディレクティブは、あるベクタの間違ったトランジションにより起こる、それより後ろのベクタの誤動作をデバッグする時に有効です。

エラーのあるベクタの後に$EXIT コマンドを置くと問題がはっきりします。

$SIMOFF

$SIMOFF シミュレーターディレクティブを使用すると、テストベクタの評価が停止されます。$SIMOFF ディレクティブ以降のテストベクタは、テスト値が正しいかあるいは無効であるかだけが評価されます。このディレクティブは、シミュレーターがレジスタード出力を正しく評価できない非同期クロックの設計をテストする時に有効です。

$SIMON

$SIMON ディレクティブを使用して、$SIMOFF ディレクティブの影響をキャンセルします。$SIMON ディレクティブ以降のテストベクタは、通常どおり評価されます。

アドバンストシンタクス

➔ 以下のコマンドはすべて、SI ファイルの VECTORS キーワード以降のテストベクタセクションで記述して下さい。

割り付け命令($SET)

```
Syntax:  $SET <variable> = <constant>;
```

<variable> = <single_sym> | <field> | <defined_variable>

<constant> = <quoted_val> | <tv_string>

<quoted_val> = シングルクオートで囲まれた数値は、入力を表わし、ダブルクオートで囲まれた数値は出力示します。それらは基数に応じて展開されます。DONT-CARE 値を指定することはできません。

<tv_string> = テストベクタ値の文字列です。値の数値は、それらが割り付けられる変数のビット番号に対応します。

動作:
定数をシンボルやフィールド、変数に割り付けます。テストベクタセクションの任意の位置に記述できます。

例:

```
$set input = '3F';  /* single quotes for inputs       */
$set output = "80"; /* double quotes for outputs      */
$set Z = HHHH;      /* test vector values for a 4-bit */
                    /* output variable                */
```

算術及び論理演算子($COMP)

```
Syntax: $COMP <variable> = <expression>;
```

<variable> = <single_sym> | <field> | <defined_variable>

<expression> = 任意の論理式または算術式で、オペランドには変数や定数を使用できます。

以下の定数は、10 進数値(引用符無し)で、丸括弧を使用できます。

演算子	機能	優先順位
!	NOT	1
&	AND	2
#	OR	3
$	XOR	4

表 6.5. 論理演算子

演算子	機能	優先順位
*	乗算	1
/	割り算	1
+	加算	2
-	減算	2

表 6.6. 算術演算

論理演算子や算術演算子は、式の中で自由に組み合わせて使用できます。通常、論理演算子のほうが、優先順位がたかくなっています。しかし、この規則は、丸括弧を使用すればオーバーライドできます。

動作:
式や計算結果の変数への割り付けを評価します。オペランド(ユーザ値)の現在の値が式の評価に使用されます。変数のユーザ値の使用の時に影響します。ベクタセクションの任意の位置に記述できます。

例:

```
$COMP A = (!B + C) * A + 1;
$COMP X = (Z / 2) # MASK;
```

テストベクタの生成($OUT)

```
Syntax: $OUT;
```

動作:
シンボルの現在の値のシミュレーションを開始しテストベクタを作成します。以前に割り

付けられた値がベクタの評価で評価されるようにできるため、$SET コマンドや$COMP コマンドの後に使用すると有効です。

例:

以下に SI ファイルのコマンドの組みを以下に示します。

```
ORDER: _CLOCK, %3, _OE, %3, shift, %1, input, %2, output;
VECTORS:
0 0 'X' XXXXXXXX LLLLLLLL /* power-on reset state */
$set _CLOCK = C;
$set shift = '0';
$set input = '80';
$set output = "80";
$out;
```

これにより、SO ファイルに以下の結果が出力されます。

```
0001: 0   0   XXX XXXXXXXX  LLLLLLLL
0002: C   0   000 10000000  HLLLLLLL
```

条件シミュレーション($IF)

```
Syntax: $IF <condition> :
           <block_1>
        [ $ELSE :
           <block_2> ]
        $ENDIF;
```

<condition> = <var_list> <logic_operators> <constant>

```
logic operators :
=  equal
#  not equal
>  greater than
<  less than
>= greater than or equal to
<= less than or equal to
```

<constant> = <quoted_val> | <tv_string>

<block_1>,<block_2> = テストベクタを含む任意の命令シーケンス

$ELSE は省略可能です。

動作:
変数の現在のシミュレーション値を使用して条件が評価されます。結果が真の場合、<block_1>が実行されます。その他の場合、$ELSE があれば、<block_2>が実行されます。$ENDIF は、IF ステートメントの終わりを示します。

繰り返し命令

FOR 命令

```
Syntax: $FOR <count> = <n1>..<n2> :
           <block>
        $ENDF;
```

<count> = FOR ループのカウンタです。<n1> と <n2> との間で変化します。

<n1>,<n2> = <count>値の限界を示します。正の 10 進数値を設定して下さい。

<block> =テストベクタを含む任意の命令シーケンスです。

動作:
Step 1. <count>が最初の値<n1>に初期化されます。
Step 2. <block>が実行されます。
Step 3. <count>=<n2>の場合、停止します。
その他の場合、<count>が 1 ずつインクリメント(<n1>が<n2>より小さい場合)されます。また、(<n1>が<n2>より大きい場合、1 ずつデクリメントされます。それから、ステップ 2.と 3.が繰返されます。

WHILE 命令

```
Syntax: $WHILE <condition> :
           <block>
        $ENDW;
```

<condition> = IF 条件と同じです。

<block> =命令の任意のシーケンスです。

動作:
Step 1:条件が比較され、フォールスの場合停止します。
その他の場合、ステップ 2 へ進みます。
Step 2: <block>を実行します。
Step 3: ステップ 1 を繰返します。

DO..UNTIL 命令

```
Syntax:  $DO:
         <block>
         $UNTIL <condition> ;
```

<condition> = IF 条件と同じです。

<block> = 命令の任意のシーケンスです。

動作:
Step 1: <block>を実行します。
Step 2: 条件を比較し、真の場合停止します。
その他の場合、ステップ 1 を続けます。

➔ IF 命令や繰り返し命令はネストすることができます。ただし、最多のネスト数は 10 までです。

MACRO ステートメントと CALL ステートメント

マクロ定義

```
Syntax: $MACRO name(<arg_list>);
           <macro_body>
        $MEND;
```

name = マクロ名です。

<arg_list> = コンマで区切られた引き数名です。

<macro_body> = 任意のステートメントのシーケンスです。ただし、(マクロセルの除く)$MACRO コマンドは使用できません。

マクロ本体で変数名や定数を使用する所に、引き数を使用することができます。演算子や特別な文字、予約語を置き換えることはできません。

マクロコール

```
Syntax: $CALL name(<act_arg_list>);
```

name = 定義されているマクロの名前です。

<act_arg_list> = 引き数のリストです。

CALL ステートメントがマクロ本体に記述される場合、引き数には、変数名や定数、マクロ定義を指定できます。

動作:
引き数に指定された変数を置き換えてマクロの本体を実行します。

➔ マクロコールを確実に実行するために、マクロ本体のシンタクスに引き数が合っている事を確認して下さい。すなわち、置き換えられる引き数がシンタクスエラーの原因にならないように注意して下さい。

例:

```
$MACRO m1(a,b,c);    /* Macro definition */
$set shift = a;
$set shift = b;
$set output = c;
$MEND;

$CALL m1('0','80',********);  /* Macro call */
```

以下のステートメントが実行されます。

```
$set shift = '0';
$set shift = '80';
$set output = ********;
```

以下にこれらのステートメントがどのように動作するかを示し、多くのテストベクタを入力しないでも設計のシミュレーションを実行できることを示します。

これら 2 つの SI ファイルは同じ出力を生成します。

1. 古い方法

```
Name        Barrel22;
Partno      CA0006;
Date        05/11/89;
Revision    02;
Designer    Kahl;
Company     Protel International;
Assembly    None;
Location    None;
Device      g20v8a;

ORDER:  _CLOCK, %3, _OE, %3, shift, %1, input, %2, output;
VECTORS:
0   0 'X' XXXXXXXX  HHHHHHHH    /* power-on reset state */
C   0 '0' 10000000  HLLLLLLL    /* shift 0 */
C   0 '1' 10000000  LHLLLLLL    /* shift 1 */
C   0 '2' 10000000  LLHLLLLL    /* shift 2 */
C   0 '3' 10000000  LLLHLLLL    /* shift 3 */
C   0 '4' 10000000  LLLLHLLL    /* shift 4 */
C   0 '5' 10000000  LLLLLHLL    /* shift 5 */
C   0 '6' 10000000  LLLLLLHL    /* shift 6 */
C   0 '7' 10000000  LLLLLLLH    /* shift 7 */
C   0 '0' 01111111  LHHHHHHH    /* shift 0 */
C   0 '1' 01111111  HLHHHHHH    /* shift 1 */
C   0 '2' 01111111  HHLHHHHH    /* shift 2 */
C   0 '3' 01111111  HHHLHHHH    /* shift 3 */
C   0 '4' 01111111  HHHHLHHH    /* shift 4 */
```

```
C   0 '5' 01111111  HHHHHLHH    /* shift 5 */
C   0 '6' 01111111  HHHHHHLH    /* shift 6 */
C   0 '7' 01111111  HHHHHHHL    /* shift 7 */
```

2. 新しい方法

```
ORDER:  _CLOCK, %3, _OE, %3, shift, %1, input, %2, output;
VECTORS:
0 0 'X' XXXXXXXX LLLLLLLL       /* power-on reset state */
$set _CLOCK = C;
$set shift = '0';
$set input = '80';
$set output = "80";
$for i = 1..16 :
$out;
$if shift = '7':
$set shift = '0';
$set input = '7f';
$set output = "7f";
$else:
$comp shift = shift + 1;
$comp output = output / 2;
$if input = '7f':
$comp output = output # 128;
$endif;
$endif;
$endf;
```

マクロを使用すると

```
ORDER:  _CLOCK, %3, _OE, %3, shift, %1, input, %2, output;
VECTORS:
$macro m1(x,y,z);
$set shift = x;
$set input = y;
$set output = z;
$mend;

$macro m2(a,b,c,d);
$call m1(a,b,c);
$for i = 1..8 :
$out; $comp shift = shift + 1;
$comp output = output / 2 + d;
$endf;
$mend;
0 0 'X' XXXXXXXX LLLLLLLL /* power-on reset state */
$set _CLOCK = C;
$call m2('0','80',"80", 0);
$call m2('0','7f',"7f", 128);
```

3. 出力:

```
CSIM: CUPL Simulation Program Version
4.2a Serial# ...
Copyright (c) 1996 Protel International
CREATED Wed Dec 04 03:00:11 1991
LISTING FOR SIMULATION FILE: barrel22.si
1: Name        Barrel22;
2: Partno      CA0006;
3: Date        05/11/89;
4: Revision    02;
5: Designer    Kahl;
6: Company     Protel International;
7: Assembly    None;
8: Location    None;
9: Device      g20v8a;
10:
11: FIELD input = [D7,D6,D5,D4,D3,D2,D1,D0];
12: FIELD output = [Q7,Q6,Q5,Q4,Q3,Q2,Q1,Q0];
13: FIELD shift = [S2,S1,S0];
14:
15: ORDER: _CLOCK, %3, _OE, %3, shift, %1, input, %2, output;
16:
17: var X = Q7;
18: var Y = Q7..4;
19:
=====================================
      _
      C
      L
      O   _
      C   O   shi
      K   E   ft  input     output
=====================================
0001: 0   0   XXX XXXXXXXX  LLLLLLLL
0002: C   0   000 10000000  HLLLLLLL
0003: C   0   001 10000000  LHLLLLLL
0004: C   0   010 10000000  LLHLLLLL
0005: C   0   011 10000000  LLLHLLLL
0006: C   0   100 10000000  LLLLHLLL
0007: C   0   101 10000000  LLLLLHLL
0008: C   0   110 10000000  LLLLLLHL
0009: C   0   111 10000000  LLLLLLLH
0010: C   0   000 01111111  LHHHHHHH
0011: C   0   001 01111111  HLHHHHHH
0012: C   0   010 01111111  HHLHHHHH
0013: C   0   011 01111111  HHHLHHHH
0014: C   0   100 01111111  HHHHLHHH
0015: C   0   101 01111111  HHHHHLHH
0016: C   0   110 01111111  HHHHHHLH
0017: C   0   111 01111111  HHHHHHHL
```

新しいシンタクスを使用してシミュレーション入力ファイルを作成する場合、ユーザが注意しなければならないことが 1 つあります。

中間に$OUT ステートメントのない条件ステートメント(IF、WHILE、UNTIL)のすぐ前にひとつ以上の$SET コマンドや$COMP コマンドが記述される場合、これらのコマンドで設定

される値(ユーザ値)は条件の値に影響しません。変数の最後のシミュレーション結果を使用して条件は評価されます。

例えば、以下のシミュレーション結果を生成したい場合、

```
ORDER: _CLOCK,clr,dir,!_OE,%2,count,%1,carry;
var mode = clr,dir;
VECTORS:
C 100  LLLL L         /* synchronous clear to state 0 */
C 000  LLLH L         /* count up to state 1          */
C 000  LLHL L         /* count up to state 2          */
C 000  LLHH L         /* count up to state 3          */
C 000  LHLL L         /* count up to state 4          */
C 000  LHLH L         /* count up to state 5          */
C 000  LHHL L         /* count up to state 6          */
C 000  LHHH L         /* count up to state 7          */
C 000  HLLL L         /* count up to state 8          */
C 000  HLLH H         /* count up to state 9 - carry  */
```

以下のシーケンスでは、違う出力が生成されます。

```
ORDER: _CLOCK,clr,dir,!_OE,%2,count,%1,carry;
var mode = clr,dir;
VECTORS:
C 100 LLLL L $set mode = '0';
$for i=1..9 :
$comp count = count + 1;
$if count="9":
$set carry = H;
$endif;
$out;
$endf;
```

that is:

```
0001: C 100  LLLL L
0002: C 000  LLLH L
0003: C 000  LLHL L
0004: C 000  LLHH L
0005: C 000  LHLL L
0006: C 000  LHLH L
0007: C 000  LHHL L
0008: C 000  LHHH L
0009: C 000  HLLL L
0010: C 000  HLLH H
                  ^
[0019sa] user expected (L) for carry
```

これは、ベクタ 10 の IF 条件の評価に使用された count の値は現在のシミュレーション値(ベクタ 9 に表示される値)であり、$COMP コマンドに設定された値ではないためです。

正しいシーケンスを以下に示します。

```
C 100 LLLL L
$set mode = '0';
$for i=1..9 :
$if count="8":
$set carry = H;
$endif;
$comp count = count + 1;
$out;
$endf;
```

仮想シミュレーション

仮想シミュレーションによりターゲットをデバイスを使用しないで作成した設計のシミュレーションができます。従って、ターゲットにするアーキテクチャーが決まる前に設計のシミュレーションを実行できます。これは、部分的な設計のシミュレーションに便利です。

仮想シミュレーションの使用法はわかりやすくなっています。コマンドやシンタクスを新しく習得する必要はありません。コンパイルやシミュレーションで Virtual DeviceVirtualDevice オプションを使用すると仮想シミュレーションを実行できます。

仮想シミュレーションは FPGA の設計のシミュレーションにも使用できます。デバイス内部の特性や内部論理リソースの複雑さのためにアーキテクチャーのシミュレーションを完全に実行できない場合、仮想シミュレーションの使用は非常に有効です。

欠陥のシミュレーション

テストベクタの欠陥の保険の意味で、任意のプロダクトタームの内部欠陥のシミュレーションを実行できます。このオプションのフォーマットを以下に示します。

```
STUCKL n ;
```

または

```
STUCKH n ;
```

ここで

n はプロダクトタームの最初のヒューズのヒューズ番号です。

ドキュメンテーションファイル(ファイル名.DOC)のヒューズマップにデバイスの各プロダクトタームの最初のヒューズのヒューズ番号が示されます。

フォーマット 1 によりプロダクトタームは 0 番にスタックされます。

フォーマット 2 によりプロダクトタームは 1 番にスタックされます。STUCK コマンドは、ORDER ステートメントと VECTORS ステートメントの間に置かれます。

ファイルフォーマット

このセクションでは、以下のフォーマットについて説明します。

ダウンロードフォーマット

このセクションでは、JEDEC と ASCII-hex 標準フォーマットについて説明します。HL フォーマットについての詳細はオンラインヘルプを参照してください。

JEDEC フォーマット

JEDEC JC-42.1 標準は、ASCII Start-of-Text(STX)キャラクタで始まり、トランスミッションに続いて各種の情報のフィールドにより構成されます。情報フィールドには、ASCII End-of-Text(ETX)キャラクタや送信チェックサムがあります。使用できるキャラクタは hex 20 から hex 7E までの ASCII の表示できるキャラクタと表 7-1 で示される 4 つの制御文字です。

STX	Start-of-Text	hex 02
ETX	End-of-Text	hex 03
LF	Line Feed	hex 0A
CR	Carriage Return	hex 0D

表 7.1. 制御文字

コンパイラーとシミュレーターを使用して作成した JEDEC ファイルのサンプルを示します。

```
<STX>
Cupl          3.0  Serial # 0-00000-000
Device        p16r4  Library DLIB-h-24-11
Created       Tue Jul 07 15:22:33 1987
Name          SAMPLE
Partno        P9000183
Revision      02
Date          03/14/85
Designer      Osann
Company       P-CAD
Assembly      PC Memory
Location      U106
*QP20
*QF2048
*G0
*F0
*L00000 10110101110111111100111000110111
*C0307
*QV
*P 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
*V0001 CXXXXX110N0HHLLZXXHN
*<<ETX>6AA1
```

JEDEC ファイルのサンプル

このセクションの残りの部分では、上記のサンプルファイルの各フィールドについて説明します。

設計の仕様はフォーマットの最初のフィールドです。このフィールドには、STX と最初のアスタリスク(*)の間の情報がすべて含まれます。この情報は、ドキュメントに使用され、それ以外には使用されません。さらに、コンパイラーやデバイスライブラリーのバージョンに沿った CUPL ソースファイルからのヘッダー情報で構成されます。

設計仕様フィールドの後の各フィールドは表 7-2 で示される 1 文字の識別子で始まります。

A - *	N - *
B - *	O - *
C - ヒューズチェックサム	P - ピン順序
D - デバイスタイプ	Q - 値
E - *	R - *
F - デフォルトヒューズステイト	S - *
G - セキュリティヒューズ	T - *
H - *	U - *
I - *	V - テストベクタ
J - *	W - *
K - *	X - *
L - ヒューズリンクデータ	Y - *
M - *	Z - *

* - 将来の使用が考えられるために予約されています。

表** 7.2. **フィールド識別子

フィールドは、複数の文字で識別することができます。

例えば、QF はデフォルトヒューズステートの値を示します。

デバイスフィールド(D)はサーポートされていません。

バリューフィールド QP は、デバイスのピン番号を表します。その他のバリューフィールド QF はデバイスの中でプログラムできるヒューズの総数を表わします。値は両方とも 10 進数です。

セキュリティーヒューズフィールド(G)は、このオプションのあるデバイスのセキュリティヒューズのプログラムがディスエーブル(G0)はイネーブル(G1)かをプログラマに指示します。スペースを 1 つ空けて互換性のあるデバイスの番号が示されます。

デフォルトヒューズステートフィールド(F)により、L フィールドよりさらに厳密にヒューズのステートが定義されます。コンパイラーは、一部のヒューズ状態しか変換しない(大規模な設計のではデータ変換の速度をあげるため)ので、このフィールドはデバイスプログラマにより認識されます。

ヒューズリンクフィールド(L)には、実際のデータがあります。各デバイスのヒューズリンクが 10 進数で割り付けられます。この値は 0000 から始まります。番号が付けられたそれぞれのヒューズには、2 つの状態があります。2 進数の 0 は、無傷のヒューズを表し、2 進数の 1 は、ヒューズの断線を表します。

➔ 製造元の中にはプログラミング前のデバイスの AC パラメータテストを実行するためのテストヒューズを指定するところもあります。これらのヒューズはヒューズリンクデータの部品ではありません。

L 識別子によりフィールドが始まり、フィールドで定義される最初のヒューズの番号がこれに続きます。複数の 2 進数値が指定される場合、最初のヒューズから連続で番号が付けられたヒューズに追加の値が割り付けられます。

次のフィールドは、ヒューズチェックサム(C)フィールドです。チェックサムは、デバイスの各ヒューズの指定されたステートで形成される 8 ビットワードを加えることにより計算される 16 ビットの 16 進値です。リンク番号 0 は最下位ビットを表わし、リンク番号 7 はワード 0 の最上位ビットを表わします。最後の 8 ビットの指定されていないビットは、チェックサムが計算される前に 0 がセットされます。最初の 22 のヒューズにより 8 ビットワードは以下のように作成されます。

```
            Msb                         lsb
  word 00    1   0   1   0   1   1   0   1   —>   AD
```

```
word 01    1   1   1   1   1   0   1   1   —>   FB
word 02    0   1   1   1   0   0   1   1   —>   73
word 03    1   1   1   0   1   1   0   0   —>   EC
                                    -------
                                      0307
```

テストベクタフィールドは、シミュレーターで作成します。このフィールドには、各デバイスピンの機能テスト情報が記述されます。QV バリューフィールドにより、ファイルに記述されているテストベクタの数が定義されます。テストベクタには、0001から番号が付けられ、テストされるデバイスに番号順に適用されます。表 11-3 に有効なピンの条件を示します。

0	-	**入力を** LO(0 volts)**へ駆動**
1	-	**入力を** HI(+5 volts)**へ駆動**
C	-	**入力を** LO,HI,LO **に駆動**
K	-	**入力を** HI,LO,HI **に駆動**
L	-	**出力** LO(0 volts)**をテスト**
H	-	**出力** HI(+5 volts)**をテスト**
Z	-	**ハイインピーダンスの出力をテスト**
X	-	**定義されていない入力、テストされない出力**
N	-	**電源ピンとテストされない出力**
P	-	**プリロードレジスタ**
		Value given applied to !Q of register

表** 7.3. **テスト条件

ベクタの中で表わされるテスト条件はピンオーダーフィールド(P)で定義される順番でデバイスピンに適用されます。この例(図 C-1)では、最初の条件はピン 1 に適用され最後の条件は 20 ピンデバイスのピン 20 に適用されます。C や K の駆動信号は他の入力が安定してから与えられます。L や H、Z 条件は入力がすべて安定してからテストされます。

クロックピンの駆動信号 P は、スーパー電圧のプリロードレジスタの機能をもつデバイスでしか使用できません。TTL レベルのプリロードピンを使用するデバイスでは、レジスタにプリロードするためにこれらのピンで C,K 駆動信号を使用する必要があります。

送信は、4 つの ASCII hex 文字の送信チェックサム(サムチェック)に続く、表示されない ASCII ETX 文字で終わります。チェックサムは STX 文字で始まり ETX 文字で終わるトランスミット文字の ASCII 値の 16 ビットの総数です。サンプルファイルでは、送信チェックサムは、各行の最後のキャリッジリターンとラインフィードを考慮して 46C9 になります。

ASCII-Hex フォーマット

ASCII-hex フォーマットは PROM で使用されます。このフォーマットのデータは、実行文字(スペース)で区切られた連続するバイトで構成されます。実行文字のすぐ前の文字がデータバイトと解釈されます。フォーマットは、一桁の 16 進数(x4PROM)か二桁の 8 進数でデータバイトを表わします。

送信は、ASCII STX [Ctrl]-[B]文字で始まります。16 個のデータバイトは$や A、カンマ($A,)に続く 4 桁の 16 進数アドレスから始まります。ASCII ETX [Ctrl]-[C]で送信のデータ部が終わります。この次に 40 個のスペースが入ります。

以下に hex ファイルのサンプルを示します。

```
^B
$A0000,00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
$A0010,10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
$A0020,20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F
$A0030,30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F
^C
$S07E0,
```

ドキュメンテーションファイルフォーマット

このセクションでは、ドキュメンテーションファイル(ファイル名.DOC)のフォーマットについて説明します。このファイルには、ヒューズプロット情報も記述されます。ドキュメンテーションファイルを生成するには、コンパイラーを実行する前に Doc File オプションの Equations をイネーブルして下さい。Doc File オプションの Fuse Plot をイネーブルするとドキュメンテーションファイルにヒューズプロット情報を生成することができます。以下にドキュメンテーションファイルのサンプルを示します。

WAITGEN.DOC

```
*************************************************************************
                              Waitgen
*************************************************************************

ADVANCED PLD    4.0 Serial# MW-67999999
Device          f155  Library DLIB-h-36-14
Created         8 11 15:18:42 1998
Name            Waitgen
Partno          P9000183
Revision        02
Date            03/14/85
Designer        Osann
Company         ATI
Assembly        PC Memory
Location        U106

=========================================================================
                        Expanded Product Terms
=========================================================================

memadr =>
    a15 , a14 , a13 , a12 , a11

memreq =>
    memw
  # memr

ram_cs0 =>
    !a11 & !a12 & a13 & !a14 & !a15 & memw
  # !a11 & !a12 & a13 & !a14 & !a15 & memr

ram_cs1 =>
    a11 & !a12 & a13 & !a14 & !a15 & memw
  # a11 & !a12 & a13 & !a14 & !a15 & memr

ready =>
    wait2

ready.oe  =>
    !a13 & !a14 & !a15 & memr

rom_cs =>
    !a13 & !a14 & !a15 & memr

select_rom =>
    !a13 & !a14 & !a15 & memr
```

```
!wait1.d  =>
    !memr
  # a15
  # a14
  # a13
  # reset

!wait2.d  =>
    !memr
  # a15
  # a14
  # a13
  # !wait1

a11.oe  =>
    0

memr.oe  =>
    0

memw.oe  =>
    0

ram_cs0.oe  =>
    1

ram_cs1.oe  =>
    1

reset.oe  =>
    0

rom_cs.oe  =>
    1

================================================================
                          Symbol Table
================================================================

Pin  Variable                        Pterms    Max       Min
Pol  Name       Ext    Pin    Type    Used    Pterms    Level
---  --------   ---    ---    ----    ----    ------    -----
    a11             6      V       -       -         -
    a12             5      V       -       -         -
    a13             4      V       -       -         -
    a14             3      V       -       -         -
    a15             2      V       -       -         -
    cpu_clk         1      V       -       -         -
    memadr          0      F       -       -         -
 !  memr            8      V       -       -         -
    memreq          0      I       2       -         -
 !  memw            7      V       -       -         -
 !  oe             11      V       -       -         -
 !  ram_cs0        12      V       2       32        0
 !  ram_cs1        13      V       2       32        0
    ready          18      V       1       32        0
    ready       oe 18      X       1       1         1
```

```
    reset               9      V       -       -         -
 !  rom_cs              19     V       1       32        0
    select_rom          0      I       1       -         -
    wait1               15     V       -       -         -
    wait1       d       15     X       5       32        0
    wait2               14     V       -       -         -
    wait2       d       14     X       5       32        0
    a11         oe      6      D       1       1         0
    memr        oe      8      D       1       1         0
    memw        oe      7      D       1       1         0
    ram_cs0     oe      12     D       1       1         0
    ram_cs1     oe      13     D       1       1         0
    reset       oe      9      D       1       1         0
    rom_cs      oe      19     D       1       1         0

LEGEND D : default variable       F : field      G : group
       I : intermediate variable  N : node       M : extended node
       U : undefined          V : variable   X : extended variable
       T : function

Total product terms merged: 4

Total product terms used by OR Arrays: 12

=====================================================================
                          Fuse Plot
=====================================================================

02092 F/F  AAAA   Pol  LHLLLLLL  O/E  AA
00000 LLHL ---- LL------ - ...A.... .... ----
00054 LLHL ---- L-L----- - ...A.... .... ----
00108 LLHL ---- HL------ - ..A..... .... ----
00162 LLHL ---- H-L----- - ..A..... .... ----
00216 ---- L--- -------- - .A...... .... ----
00270 LLL- ---- --L----- - A....... .... ----
00324 ---- ---- --H----- - ........ .... --HH
00378 H--- ---- -------- - ........ .... --HH
00432 -H-- ---- -------- - ........ .... --HH
00486 --H- ---- -------- - ........ .... --HH
00540 ---- ---- ---H---- - ........ .... --H-
00594 ---- -H-- -------- - ........ .... ---H
00648 0000 0000 00000000 0 AAAAAAAA AAAA 0000
00702 0000 0000 00000000 0 AAAAAAAA AAAA 0000
00756 0000 0000 00000000 0 AAAAAAAA AAAA 0000
00810 0000 0000 00000000 0 AAAAAAAA AAAA 0000
00864 0000 0000 00000000 0 AAAAAAAA AAAA 0000
00918 0000 0000 00000000 0 AAAAAAAA AAAA 0000
00972 0000 0000 00000000 0 AAAAAAAA AAAA 0000
01026 0000 0000 00000000 0 AAAAAAAA AAAA 0000
01080 0000 0000 00000000 0 AAAAAAAA AAAA 0000
01134 0000 0000 00000000 0 AAAAAAAA AAAA 0000
01188 0000 0000 00000000 0 AAAAAAAA AAAA 0000
01242 0000 0000 00000000 0 AAAAAAAA AAAA 0000
01296 0000 0000 00000000 0 AAAAAAAA AAAA 0000
01350 0000 0000 00000000 0 AAAAAAAA AAAA 0000
01404 0000 0000 00000000 0 AAAAAAAA AAAA 0000
01458 0000 0000 00000000 0 AAAAAAAA AAAA 0000
01512 0000 0000 00000000 0 AAAAAAAA AAAA 0000
```

```
01566 0000 0000 00000000 0 AAAAAAAA AAAA 0000
01620 0000 0000 00000000 0 AAAAAAAA AAAA 0000
01674 0000 0000 00000000 0 AAAAAAAA AAAA 0000
01728 0000 0000 00000000 .
01761 0000 0000 00000000 .
01794 0000 0000 00000000 .
01827 0000 0000 00000000 .
01860 ---- ---- -------- -
01893 ---- ---- -------- -
01926 LLL- ---- --L----- -
01959 ---- ---- -------- -
01992 0000 0000 00000000 .
02025 0000 0000 00000000 .
02058 0000 0000 00000000 .

LEGEND    single fuse
          L : active level low        H : active level high
          A : active fuse not blown   . : active fuse blown

          double fuse
          0 : neither fuse blown      - : both fuses blown
          L : first fuse blown        H : second fuse blown
          A : generate                . : propagate

================================================================
                          Chip Diagram
================================================================

                        _______________
                       |   Waitgen     |
           cpu_clk x---|1           20|---x Vcc
               a15 x---|2           19|---x !rom_cs
               a14 x---|3           18|---x ready
               a13 x---|4           17|---x
               a12 x---|5           16|---x
               a11 x---|6           15|---x wait1
             !memw x---|7           14|---x wait2
             !memr x---|8           13|---x !ram_cs1
             reset x---|9           12|---x !ram_cs0
               GND x---|10          11|---x !oe
                       |_______________|
```

ファイルの最初の部分には、ファイル管理情報やファイル履歴情報の対応する CUPL ソースファイルと同じヘッダー情報が記述されます。また、デバイスライブラリーやコンパイラープログラムのバージョン情報やファイルが作成された日付や時間の情報が記述されます。

ファイルの次のセクション、Expanded Product Terms には、論理記述ファイルの式からコンパイラーにより生成されたプロダクトタームが記述されます。Advanced PLD パッケージの中の WAITGEN.PLD は、上記のドキュメンテーションファイルのサンプルの元になった論理記述ファイルです。内容を見れば、元の論理式とコンパイラーにより生成されたプロダクトタームを比較することができます。

指定されたデバイスのプロダクタームはコンパイラーにより生成されます。例えば、PAL16L8 などのデバイスの中には、固定された反転バッファを持つものがあり、デバイスに論理を適合させるためにコンパイラーはドモルガンの定理を使用する場合があります。例えば、論理記述ファイルが PAL16L8 向けに作成される場合、ピンリストの中の出力はすべてアクティブ HI で宣言される必要があります。以下の式により OR 関数が指定されます。

```
c = a # b ;
```

しかし、PAL16L8 には固定反転バッファーがあります。反転バッファーを変更することはできないので、コンパイラーは OR 式にドモルガンの定理を適用して以下の式を生成し論理をデバイスに適合させます。

```
c => !a & !b
```

ファイルの次のセクションの Symbol Table には、論理記述ファイルの各変数に関する情報が記述されます。この中の情報には、ピン番号や拡張子、変数のタイプ、使用できるプロダクトタームの番号、使用されているプロダクトタームの番号、コンパイラーにより使用された最小化レベルがあります。

デバイスで使用できる最大のプロダクトタームを越える場合、コンパイラーは、コンパイル時のピンに名前を付ける処理を行なっている時にエラーメッセージを表示します。しかし、メッセージには限界がどのくらいかは表示されません。シンボルテーブルのプロダクトタームに関するこれらの情報により、使用できるプロダクトタームの数が大きく限度を越えているのか、それともわずかに越えているのかが分かります。

バークレイ PLA ファイルフォーマット

このセクションでは、バークレイ PLA ファイル(ファイル名.PLA)のフォーマットについて説明します。バークレイ PLA フォーマットは、バークレイ PLA ツールなどの PLA 論理合成ツールのインターフェースフォーマットとして使用されます。バークレイ PLA-フォーマットファイルは、コンパイル時に Berkely PLA オプションをイネーブルすると生成できます。以下に、バークレイ PLA-フォーマットファイルのサンプルを示します。

ファイルの最初の部分には、ファイルの管理情報や履歴情報が記述されます。#文字によりコメントが示されます。この情報は、対応する CUPL ソースファイルのヘッダー情報と同じです。また、デバイスライブラリーやコンパイラープログラムのバージョン情報やファイルが作成された日付や時間の情報が記述されます。

次のセクションは、(Advanced PLD パッケージにある)論理記述ファイル COUNT10.PLD の式からコンパイラーにより生成される PLA 記述で構成されます。その内容を見て、元の論理記述式をコンパイラーにより生成された PLA 記述と比較することができます。

PLA 記述は、入力.i や出力.o、プロダクトターム.p、プロダクトターム毎に一行に記述される AND や OR プレーンの数を定義するフィールドで構成されます。AND プレーンの接続は、1 が非反転入力で 0 が反転入力として表わされます。接続のない入力は-で表わされます。OR プレーンの接続は 1 で表わされ、接続されていない入力は 0 で表わされます。PLA 記述の最後は.end で示されます。

以下にバークレイ PLA-フォーマットファイルのサンプルを示します。

```
# Berkeley PLA format generated using
# CUPL        3.0 Serial# 9-99999-999
# Device      p16rp4 Library DLIB-g-24-15
# Created     Thu Feb 26 13:45:23 1987
# Name        Count10
# Partno      CA0018
# Revision    01
# Date        07/16/87
# Designer    Kahl
# Company     Assisted Technology
# Assembly    None
# Location    None
# Inputs 1 Q0 Q1 Q2
#        Q3 clr dir
# Outputs Q0.d Q1.d Q2.d Q3.d
#         carry carry.oe
.i 7
.o 6
.p 18
-00010- 100000
-0--00- 100000
-000101 010000
-10-000 010000
-01-000 010000
-11-001 010000
-001001 010000
-000101 001000
-110000 001000
—01000- 001000
-1-1001 001000
-01100- 001000
-100101 000100
-000001 000100
```

```
-111000 000100
-000100 000100
-1001-- 000010
1------ 000001
.end
```