

# エンベデッド システム ツール リファレンス マニュアル

## エンベデッド開発キット EDK 11.1

UG111 (EDK 11.1)

本資料は英語版 (v11.1) を翻訳したものです。英語の更新バージョンがリリースされている場合には、最新の英語版を必ずご参照ください。





© Copyright 2002 – 2009 Xilinx, Inc. All Rights Reserved.

XILINX, the Xilinx logo, the Brand Window and other designated brands included herein are trademarks of Xilinx, Inc.

The PowerPC® name and logo are registered trademarks of IBM Corp., and used under license. All other trademarks are the property of their respective owners.

Disclaimer:

Xilinx is disclosing this user guide, manual, release note, and/or specification (the “Documentation”) to you solely for use in the development of designs to operate with Xilinx hardware devices. You may not reproduce, distribute, republish, download, display, post, or transmit the Documentation in any form or by any means including, but not limited to, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of Xilinx. Xilinx expressly disclaims any liability arising out of your use of the Documentation. Xilinx reserves the right, at its sole discretion, to change the Documentation without notice at any time. Xilinx assumes no obligation to correct any errors contained in the Documentation, or to advise you of any corrections or updates. Xilinx expressly disclaims any liability in connection with technical support or assistance that may be provided to you in connection with the Information.

THE DOCUMENTATION IS DISCLOSED TO YOU “AS-IS” WITH NO WARRANTY OF ANY KIND. XILINX MAKES NO OTHER WARRANTIES, WHETHER EXPRESS, IMPLIED, OR STATUTORY, REGARDING THE DOCUMENTATION, INCLUDING ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT OF THIRD-PARTY RIGHTS. IN NO EVENT WILL XILINX BE LIABLE FOR ANY CONSEQUENTIAL, INDIRECT, EXEMPLARY, SPECIAL, OR INCIDENTAL DAMAGES, INCLUDING ANY LOSS OF DATA OR LOST PROFITS, ARISING FROM YOUR USE OF THE DOCUMENTATION.

# このマニュアルについて

---

エンベデッド開発キット (EDK) は、MicroBlaze™ ソフト プロセッサおよび PowerPC® ハード プロセッサを使用してエンベデッド プロセッサ システムを構築するために必要な、設計ツールおよびペリフェラルを含む開発キットです。

このマニュアルには、EDK に含まれるエンベデッド システム ツールに関する情報が掲載されています。これらのツールは、プロセッサ プラットフォーム設計ユーティリティ、ソフトウェア アプリケーション開発ツール、デバッグ ツール、デバイス ドライバおよびライブラリで構成されており、MicroBlaze および PowerPC プロセッサとそのペリフェラルの機能を最大限に活用した設計が可能です。

## マニュアルの内容

このマニュアルは、次の章から構成されています。

- 第 1 章 「エンベデッド システムとツールの概要」
- 第 2 章 「Platform Generator (Platgen)」
- 第 3 章 「Simulation Model Generator (Simgen)」
- 第 4 章 「Library Generator (Libgen)」
- 第 5 章 「Platform Specification Utility (PsfUtil)」
- 第 6 章 「バージョン管理ツール (revup)」
- 第 8 章 「Bitstream Initializer (BitInit)」
- 第 7 章 「フラッシュ メモリのプログラム」
- 第 9 章 「GNU コンパイラ ツール」
- 第 10 章 「GNU デバッガ (GDB)」
- 第 11 章 「Xilinx Microprocessor Debugger (XMD)」
- 第 12 章 「System ACE ファイル ジェネレータ (GenACE)」
- 第 13 章 「コマンド ライン モード」
- 第 14 章 「ザイリンクス Bash シェル」
- 付録 A 「GNU ユーティリティ」
- 付録 B 「割り込み制御」
- 付録 C 「EDK Tcl インターフェイス」
- 付録 D 「用語集」

## その他のリソース

- ザイリンクスの Web サイト  
<http://japan.xilinx.com/>
- ザイリンクス アンサー ブラウザおよびテクニカル サポート ウェブケース  
<http://japan.xilinx.com/support>
- XPS (Xilinx Platform Studio) と EDK の Web サイト  
[http://japan.xilinx.com/ise/embedded\\_design\\_prod/platform\\_studio.htm](http://japan.xilinx.com/ise/embedded_design_prod/platform_studio.htm)
- XPS と EDK の資料  
[http://japan.xilinx.com/ise/embedded/edk\\_docs.htm](http://japan.xilinx.com/ise/embedded/edk_docs.htm)
- XPS/EDK でサポートされる IP  
[http://japan.xilinx.com/ise/embedded/edk\\_ip.htm](http://japan.xilinx.com/ise/embedded/edk_ip.htm)
- EDK のサンプル デザイン  
[http://japan.xilinx.com/ise/embedded/edk\\_examples.htm](http://japan.xilinx.com/ise/embedded/edk_examples.htm)
- チュートリアル  
<http://japan.xilinx.com/support/techsup/tutorials/index.htm>
- データシート  
[http://japan.xilinx.com/support/documentation/data\\_sheets.htm](http://japan.xilinx.com/support/documentation/data_sheets.htm)
- プロブレム ソルバー  
<http://japan.xilinx.com/support/troubleshoot/psolvers.htm>
- ISE® マニュアル  
[http://japan.xilinx.com/support/software\\_manuels.htm](http://japan.xilinx.com/support/software_manuels.htm)
- その他のマニュアル  
<http://japan.xilinx.com/support/documentation/>
- GNU マニュアル  
<http://www.gnu.org/manual>

## 表記規則

このマニュアルでは、次の表記規則を使用しています。各規則について、例を挙げて説明します。

### 書体

次の規則は、すべてのマニュアルで使用されています。

表記規則	使用箇所	例
Courier フォント	システムが表示するメッセージ、プロンプト、プログラム ファイルを表示します。	<code>speed grade: - 100</code>
<b>Courier</b> フォント (太字)	構文内で入力するコマンドを示します。	<b>ngdbuild</b> <i>design_name</i>
イタリック フォント	ユーザーが値を入力する必要のある構文内の変数に使用します。	<i>ngdbuild design_name</i>
二重/一重かぎカッコ『』、『』、『』	『』はマニュアル名を、『』はセクション名を示します。	詳細は、『コマンド ライン ツール ユーザー ガイド』の「PAR」を参照してください。
角カッコ [ ]	オプションの入力またはパラメータを示しますが、 <b>bus[7:0]</b> のようなバス仕様では必ず使用します。また、GUI 表記にも使用します。	<b>ngdbuild</b> [ <i>option_name</i> ] <i>design_name</i> [File] → [Open] をクリックします。
中カッコ { }	1 つ以上の項目を選択するためのリストを示します。	<b>lowpwr</b> = {on off}
縦棒	選択するリストの項目を分離します。	<b>lowpwr</b> = {on off}
縦の省略記号 . . .	繰り返し項目が省略されていることを示します。	IOB #1: Name = QOUT' IOB #2: Name = CLKIN' . . .
横の省略記号 ...	繰り返し項目が省略されていることを示します。	<b>allow block</b> <i>block_name loc1 loc2 ... locn;</i>

## オンライン マニュアル

このマニュアルでは、次の規則が使用されています。

表記規則	使用箇所	例
青色の文字	マニュアル内の相互参照を示します。	詳細は、「 <a href="#">その他のリソース</a> 」を参照してください。 詳細は、第 1 章の「 <a href="#">タイトルフォーマット</a> 」を参照してください。
赤色の文字	ほかのマニュアルへの相互参照を示します。	詳細は、『Virtex-II Platform FPGA ユーザー ガイド』の <a href="#">図 2-5</a> を参照してください。
<a href="#">青色の下線付き文字</a>	Web サイト (URL) へのハイパーリンクです。	最新のスピード ファイルは、 <a href="http://japan.xilinx.com">http://japan.xilinx.com</a> から入手できます。

# 目次

---

## このマニュアルについて

マニュアルの内容 .....	3
その他のリソース .....	4
表記規則 .....	5
書体 .....	5
オンライン マニュアル .....	6

## 第 1 章：エンベデッド システムとツールの概要

EDK について .....	19
関連リソース .....	20
設計プロセスの概要 .....	20
ハードウェア開発 .....	21
ソフトウェア開発 .....	21
検証 .....	21
シミュレーションを使用したハードウェアの検証 .....	21
デバッグによるソフトウェアの検証 .....	21
デバイスのコンフィギュレーション .....	21
EDK の概要 .....	22
EDK ツールおよびユーティリティ .....	23
Xilinx Platform Studio (XPS) .....	24
XPS コマンド ライン モード .....	25
Base System Builder (BSB) ウィザード .....	25
Create and Import Peripheral (CIP) Wizard .....	25
コプロセッサ ウィザード .....	26
Platform Generator (Platgen) .....	26
Debug Configuration Wizard .....	26
Simulation Model Generator (Simgen) .....	27
バス ファンクション モデル (BFM) .....	27
Platform Specification Utility (PsfUtility) .....	27
ソフトウェア開発キット (SDK) .....	27
Library Generator (Libgen) .....	28
GNU コンパイラ ツール (GCC) .....	28
Xilinx Microprocessor Debugger (XMD) .....	29
GNU デバッガ (GDB) .....	29
シミュレーション ライブラリ コンパイラ (CompLib) .....	29
Bitstream Initializer (BitInit) .....	29
System ACE ファイル ジェネレータ (GenACE) .....	29
フラッシュ メモリ プログラマ .....	30
フォーマット リビジョン ツールおよび Version Management Wizard .....	30

## 第 2 章 : Platform Generator (Platgen)

機能 .....	31
関連リソース .....	32
ツール要件 .....	32
Platgen の使用法 .....	32
Platgen のコマンド オプション .....	32
ディレクトリ構造 .....	33
出力ファイル .....	34
hdl ディレクトリ .....	34
implementation ディレクトリ .....	34
synthesis ディレクトリ .....	34
BMM フロー .....	34
合成ネットリスト キャッシュ .....	35

## 第 3 章 : Simulation Model Generator (Simgen)

Simgen の概要 .....	37
関連リソース .....	38
シミュレーション ライブラリ .....	38
ザイリンクス ISE ライブラリ .....	38
UNISIM ライブラリ .....	38
SIMPRIM ライブラリ .....	38
XilinxCoreLib ライブラリ .....	39
ザイリンクス EDK ライブラリ .....	39
EDK ライブラリの検索順 .....	39
Compplib ユーティリティ .....	39
シミュレーション モデル .....	40
ビヘイビア モデル .....	40
構造モデル .....	41
タイミング モデル .....	41
単一言語モデルと混合言語モデル .....	42
XPS のバッチ モードを使用したシミュレーション モデルの作成 .....	42
Simgen の構文 .....	43
要件 .....	43
Simgen のコマンド オプション .....	43
出力ファイル .....	45
メモリの初期化 .....	46
VHDL .....	46
Verilog .....	46
テストベンチ .....	46
VHDL テストベンチの例 .....	47
Verilog テストベンチの例 .....	48
デザインのシミュレーション .....	49
制限 .....	50



## 第 4 章 : Library Generator (Libgen)

概要 .....	51
関連リソース .....	52
Libgen の使用法 .....	52
Libgen のコマンド オプション .....	52
ディレクトリ構造 .....	54
PC のディレクトリ .....	54
追加ディレクトリの指定 .....	54
ディレクトリの検索順 .....	54
出力ファイル .....	55
include ディレクトリ .....	55
lib ディレクトリ .....	56
libsrc ディレクトリ .....	56
code ディレクトリ .....	56
ライブラリおよびドライバの生成 .....	56
概要 .....	56
MDD、MLD、および Tcl .....	57
MSS ファイルのパラメータ .....	57
ドライバ .....	57
ライブラリ .....	58
OS ブロック .....	59

## 第 5 章 : Platform Specification Utility (PsfUtil)

PsfUtil のコマンド オプション .....	62
MPD ファイル作成プロセスの概要 .....	63
MPD ファイル自動生成の使用法 .....	63
バス インターフェイスを 1 つ使用するペリフェラル .....	63
信号の命名規則 .....	64
PsfUtil の実行 .....	64
複数のバス インターフェイスを使用するペリフェラル .....	64
非排他的および排他的バス インターフェイス .....	64
ポイントツーポイント接続を含むペリフェラル .....	65
PsfUtil での DRC チェック .....	65
HDL ソース エラー .....	65
バス インターフェイス チェック .....	65
HDL ペリフェラルの定義 .....	66
バス インターフェイスの命名規則 .....	66
VHDL ジェネリックの命名規則 .....	66
予約済みパラメータ .....	68
バス インターフェイス信号の命名規則 .....	69
グローバル ポート .....	70
スレーブ DCR ポート .....	70
DCR スレーブ出力 .....	71
DCR スレーブ入力 .....	71
スレーブ FSL ポート .....	71
FSL スレーブ出力 .....	71

FSL スレーブ入力 .....	72
マスタ FSL ポート .....	72
FSL マスタ出力 .....	72
FSL マスタ入力 .....	72
スレーブ LMB ポート .....	73
LMB スレーブ出力 .....	73
LMB スレーブ入力 .....	73
マスタ OPB ポート .....	74
OPB マスタ出力 .....	74
OPB マスタ入力 .....	74
スレーブ OPB ポート .....	75
OPB スレーブ出力 .....	75
OPB スレーブ入力 .....	76
マスタ/スレーブ OPB ポート .....	76
OPB マスタ/スレーブ出力 .....	77
OPB マスタ/スレーブ入力 .....	77
マスタ PLB ポート .....	78
PLB マスタ出力 .....	78
PLB マスタ入力 .....	79
スレーブ PLB ポート .....	79
PLB スレーブ出力 .....	80
PLB スレーブ入力 .....	80
マスタ PLBV46 ポート .....	81
PLBV46 マスタ出力 .....	81
PLBV46 マスタ入力 .....	81
スレーブ PLBV46 ポート .....	82
PLBV46 スレーブ出力 .....	82
PLBV46 スレーブ入力 .....	83

## 第 6 章 : バージョン管理ツール (revup)

概要 .....	85
フォーマット リビジョン ツールによるバックアップおよびアップデート .....	86
11.1 での変更 .....	86
10.1 での変更 .....	86
9.2i での変更 .....	86
9.1i での変更点 .....	87
8.2i での変更点 .....	87
8.1i での変更点 .....	87
7.1i での変更点 .....	87
6.3i での変更点 .....	87
6.2i での変更点 .....	88
フォーマット リビジョン ツールのコマンド オプション .....	88
Version Management Wizard .....	88

## 第 7 章 : フラッシュ メモリのプログラム

概要 .....	89
----------	----

XPS および SDK を使用したフラッシュ デバイスのプログラム .....	90
サポートされるフラッシュ ハードウェア .....	90
フラッシュ プログラムのパフォーマンス .....	91
フラッシュのプログラム設定のカスタマイズ .....	92
ブートローダ アプリケーション用に ELF ファイルを SREC に手動で 変換する方法 .....	94
操作上の注意点と回避策 .....	94
競合するセクタ レイアウトでのフラッシュ デバイスの処理 .....	94
AMD/富士通コマンド セットのデータ ポーリング アルゴリズム .....	95

## 第 8 章 : Bitstream Initializer (BitInit)

概要 .....	97
BitInit の使用法 .....	97
BitInit のコマンド オプション .....	98

## 第 9 章 : GNU コンパイラ ツール

概要 .....	99
関連リソース .....	100
コンパイラのフレームワーク .....	101
コンパイラの使用法とオプション .....	102
構文 .....	102
入力ファイル .....	102
出力ファイル .....	103
ファイル タイプとその拡張子 .....	103
ライブラリ .....	103
言語タイプ .....	104
よく使用されるコンパイラ オプションの一覧 .....	105
一般オプション .....	105
ライブラリ検索オプション .....	108
ヘッダ ファイル検索オプション .....	108
デフォルトの検索パス .....	108
リンカ オプション .....	109
メモリのレイアウト .....	109
予約済みメモリ .....	110
I/O メモリ .....	110
ユーザーおよびプログラム メモリ .....	110
オブジェクト ファイルのセクション .....	111
リンカ スクリプト .....	113
MicroBlaze コンパイラの使用法とオプション .....	115
MicroBlaze コンパイラ .....	115
MicroBlaze コンパイラ オプションの一覧 .....	115
プロセッサ機能選択オプション .....	115
一般プログラム オプション .....	118
アプリケーション実行モード .....	119
位置独立コード (PIC) .....	120
MicroBlaze アプリケーション バイナリ インターフェイス .....	120

MicroBlaze アセンブラ .....	120
MicroBlaze リンカ オプション .....	122
MicroBlaze リンカ スクリプトで割り当てられるセクション .....	123
リンカ スクリプトを記述またはカスタマイズする際のヒント .....	123
スタートアップ ファイル .....	124
第 1 段階の初期化ファイル .....	125
第 2 段階の初期化ファイル .....	125
その他のファイル .....	126
スタートアップ ファイルの変更 .....	127
C プログラムのスタートアップ コード サイズの削減 .....	127
コンパイラ ライブラリ .....	128
スレッド セーフ .....	128
コマンド ライン引数 .....	129
割り込みハンドラ .....	129
interrupt_handler 属性 .....	129
save_volatiles 属性 .....	129
PowerPC コンパイラの使用法とオプション .....	130
PowerPC コンパイラ オプションの一覧 .....	130
PowerPC コンパイラ オプション .....	130
PowerPC プロセッサ リンカ オプション .....	132
PowerPC プロセッサ リンカ スクリプトで割り当てられるセクション .....	132
リンカ スクリプトを記述またはカスタマイズする際のヒント .....	133
スタートアップ ファイル .....	134
初期化ファイル .....	134
スタートアップ ファイルの説明 .....	135
その他のファイル .....	135
スタートアップ ファイルの変更 .....	136
C プログラムのスタートアップ コード サイズの削減 .....	136
アプリケーションをブートローダーで読み込む場合のスタートアップ ファイルの変更 .....	137
コンパイラ ライブラリ .....	137
スレッド セーフ .....	137
コマンド ライン引数 .....	137
その他のメモ .....	138
C++ コードのサイズ .....	138
C++ 標準ライブラリ .....	138
位置独立コード (PIC) .....	138
その他のオプションおよび機能 .....	138

## 第 10 章 : GNU デバッガ (GDB)

概要 .....	139
GDB の使用法 .....	139
GDB のコマンド オプション .....	139
GDB を使用したデバッグ フロー .....	140
関連リソース .....	140
MicroBlaze GDB のターゲット .....	140

シミュレータ .....	140
ハードウェア .....	140
MicroBlaze をデバッグするためのコンパイル .....	141
PowerPC 405 のデバッグ .....	141
PowerPC 440 のデバッグ .....	141
コンソール モード .....	142
GDB コマンドに関するリファレンス情報 .....	143

## 第 11 章 : Xilinx Microprocessor Debugger (XMD)

関連リソース .....	146
XMD の使用法 .....	147
XMD コンソール .....	148
XMD コマンド .....	148
XMD ユーザー コマンドの一覧 .....	148
XMD ユーザー コマンド .....	149
特殊用途レジスタ名 .....	155
MicroBlaze の特殊用途レジスタ名 .....	155
PowerPC 405 プロセッサの特殊用途レジスタ名 .....	156
PowerPC 440 プロセッサの特殊用途レジスタ名 .....	156
XMD のリセット シーケンス .....	157
PowerPC 405 プロセッサ .....	157
PowerPC 440 プロセッサ .....	158
MicroBlaze .....	158
推奨される XMD フロー .....	158
1 つのプログラムのデバッグ .....	158
複数プロセッサ環境でのプログラムのデバッグ .....	159
デバッグ セッションでのプログラムの実行 .....	159
自動例外トラップでのセーフモードの使用 .....	159
プロセッサのデフォルト例外設定 .....	160
例外設定の上書き .....	161
セーフモード設定の表示 .....	161
connect コマンドのオプション .....	162
構文 .....	162
PowerPC プロセッサ .....	162
PowerPC プロセッサ ハードウェアの接続 .....	162
PowerPC プロセッサのデバッグにおける要件 .....	165
デバッグ セッションの例 .....	166
シミュレータを使用した PowerPC プロセッサのデバッグ .....	170
PowerPC プロセッサ ISS の実行 .....	170
ISS を使用した PowerPC プロセッサのデバッグ セッション例 .....	172
DCR、TLB、およびキャッシュのアドレス空間とそのアクセス .....	172
PowerPC プロセッサのデバッグ ヒント (アドバンス) .....	174
MicroBlaze プロセッサのデバッグ .....	175
MDM を使用した MicroBlaze のデバッグ .....	175
MDM を使用した MicroBlaze のデバッグにおける要件 .....	176
デバッグ セッションの例 .....	176

XMDStub を使用した MicroBlaze のデバッグ .....	178
XMDStub/JTAG を使用した MicroBlaze のデバッグ用のコマンド オプション ....	178
XMDStub/シリアル インターフェイスを使用した MicroBlaze のデバッグ用の コマンド オプション .....	179
XMDStub を使用したデバッグにおける要件 .....	180
シミュレータを使用した MicroBlaze のデバッグ .....	181
シミュレータを使用したデバッグにおける要件 .....	181
MDM ペリフェラルおよび UART を使用したデバッグ .....	182
デバッグ セッションの設定 .....	182
マルチプロセッシング システムでのリセットの設定 .....	184
XMD 内部 Tcl コマンド .....	185
プログラム初期化オプション .....	185
レジスタ/メモリのオプション .....	186
プログラム制御オプション .....	187
プログラム トレースおよびプロファイル オプション .....	189
その他のコマンド .....	189

## 第 12 章 : System ACE ファイル ジェネレータ (GenACE)

必要条件 .....	191
ツール要件 .....	192
GenACE の機能 .....	192
GenACE モデル .....	192
genace.tcl スクリプト .....	193
構文 .....	193
構文 .....	196
Genace.tcl スクリプトでサポートされるターゲット ボード .....	196
ACE ファイルの作成 .....	197
カスタム ボード .....	197
1 つの FPGA デバイス .....	197
ハードウェアとソフトウェアのコンフィギュレーション .....	197
ハードウェアとソフトウェアのパーシャル リコンフィギュレーション .....	197
ハードウェアのみのコンフィギュレーション .....	197
ハードウェアのみのパーシャル リコンフィギュレーション .....	197
ソフトウェアのみのコンフィギュレーション .....	198
複数プロセッサ システムの 1 つのプロセッサの ACE 作成 .....	198
複数のプロセッサを含むシステムのコンフィギュレーション .....	198
複数の FPGA デバイス .....	199
関連情報 .....	201
CF デバイス フォーマット .....	201

## 第 13 章 : コマンド ライン モード

XPS コマンド ライン モードの起動 .....	203
新規プロジェクトの作成 (空のプロジェクト) .....	204
新規プロジェクトの作成 (既存の MHS ファイルを指定) .....	204
既存のプロジェクトを開く .....	204
MSS ファイルの読み込み .....	204

プロジェクト ファイルの保存 .....	205
プロジェクト オプションの設定 .....	205
フロー コマンドの実行 .....	206
<b>MHS</b> ファイルの再読み込み .....	207
ソフトウェア アプリケーションの追加 .....	207
ソフトウェア アプリケーションの削除 .....	208
ソフトウェア アプリケーションへのプログラム ファイルの追加 .....	208
ソフトウェア アプリケーションからのプログラム ファイルの削除 .....	208
ソフトウェア アプリケーションのオプションの設定 .....	208
特殊なソフトウェア アプリケーションの設定 .....	209
制限 .....	210
<b>MSS</b> ファイルの変更 .....	210
<b>XMP</b> ファイルの変更 .....	210

## 第 14 章 : ザイリンクス Bash シェル

概要 .....	211
EDK でインストールされる <b>Cygwin</b> 環境 .....	211
既存の <b>Cygwin</b> 環境を使用するための要件 .....	211
ザイリンクス Bash シェル .....	211
<b>xbash</b> の使用 .....	212
<b>-override</b> オプションと <b>-undo</b> オプション .....	212
Windows Vista での <b>Cygwin</b> .....	212

## 付録 A : GNU ユーティリティ

MicroBlaze および PowerPC の汎用ユーティリティ .....	213
<b>cpp</b> .....	213
<b>gcov</b> .....	213
MicroBlaze および PowerPC 専用ユーティリティ .....	213
<b>mb-addr2line</b> .....	213
<b>mb-ar</b> .....	213
<b>mb-as</b> .....	213
<b>mb-c++</b> .....	214
<b>mb-c++filt</b> .....	214
<b>mb-g++</b> .....	214
<b>mb-gasp</b> .....	214
<b>mb-gcc</b> .....	214
<b>mb-gdb</b> .....	214
<b>mb-gprof</b> .....	214
<b>mb-ld</b> .....	214
<b>mb-nm</b> .....	214
<b>mb-objcopy</b> .....	214
<b>mb-objdump</b> .....	214
<b>mb-ranlib</b> .....	215
<b>mb-readelf</b> .....	215
<b>mb-size</b> .....	215
<b>mb-strings</b> .....	215

mb-strip .....	215
その他のプログラムおよびファイル .....	215

## 付録 B : 割り込み制御

関連リソース .....	217
ハードウェアの設定 .....	218
ソフトウェアの設定と割り込みフロー .....	219
MicroBlaze システムの割り込みフロー .....	219
PowerPC システムの割り込みフロー .....	221
ソフトウェア API .....	223
割り込みコントローラ ドライバ .....	223
API の説明 .....	224
MicroBlaze 用のスタンドアロン ソフトウェア プラットフォーム API .....	226
MicroBlaze の割り込み設定例 .....	227
PowerPC 405 および 440 プロセッサ用のスタンドアロン ソフトウェア API .....	229
PowerPC プロセッサの割り込み設定例 .....	231

## 付録 C : EDK Tcl インターフェイス

はじめに .....	235
その他のリソース .....	235
ハンドル .....	236
データ構造の作成 .....	236
Tcl コマンドの使用法 .....	237
一般的な規則 .....	237
Tcl コマンド使用前の確認事項 .....	237
EDK ハードウェア Tcl コマンド .....	238
概要 .....	238
ハードウェア読み出しアクセス API .....	239
ハードウェア読み出しアクセス API の一覧 .....	239
ハードウェア読み出しアクセス API の説明 .....	239
ハードウェア API を使用する Tcl の例 .....	247
例 1 .....	247
例 2 .....	247
アドバンス書き込みアクセス API .....	248
ハードウェア アドバンス書き込みアクセス ハードウェア API の一覧 .....	248
ハードウェア アドバンス書き込みアクセス API の説明 .....	249
ソフトウェア Tcl コマンド .....	254
ソフトウェア API で使用される用語 .....	254
ソフトウェア読み出しアクセス API .....	255
ソフトウェア読み出しアクセス API の一覧 .....	255
ソフトウェア読み出しアクセス API の説明 .....	256
ハードウェア プラットフォーム生成の Tcl フロー .....	264
入力ファイル .....	264
ハードウェア プラットフォーム生成中に呼び出される Tcl プロシージャ .....	264
統合ハードウェア データ構造の追加のキーワード .....	270
ソフトウェア プラットフォーム生成の Tcl フロー .....	271



入力ファイル .....	271
Libgen からの Tcl プロシージャ呼び出し .....	271

## 付録 D : 用語集



# エンベデッド システムとツールの概要

---

この章では、ザイリンクスのエンベデッド開発キット (EDK) で PowerPC® (405 と 440) プロセッサおよび MicroBlaze™ エンベデッド プロセッサを使用したシステムを開発する際に使用するエンベデッド システム ツールと開発フローについて説明します。この章には、次のセクションが含まれています。

- [EDK について](#)
- [関連リソース](#)
- [設計プロセスの概要](#)
- [EDK の概要](#)

## EDK について

ザイリンクス エンベデッド開発キット (EDK) システム ツールでは、ザイリンクス FPGA デバイスにインプリメントするエンベデッド プロセッサ システムを設計できます。

EDK には、次のものが含まれます。

- Xilinx® Platform Studio (XPS) システム ツール：エンベデッド プロセッサのハードウェアを設計するために使用します。
- ソフトウェア開発キット (SDK)：Eclipse オープン ソース フレームワークに基づくツールで、エンベデッド ソフトウェア アプリケーションの開発に使用できます。SDK は、スタンドアロン プログラムとしても提供されています。
- プロセッサおよびペリフェラルなどのエンベデッド プロセッサ IP コア

EDK は、ザイリンクス プログラマブル ロジック デバイスにデザインをインプリメントするのに必要な開発システムである ISE® (Integrated Software Environment) のコンポーネントです。ISE のインストールで EDK を選択するとインストールされます。SDK は、ISE または EDK をインストールせずにスタンドアロンとしてもインストールできます。

EDK 環境でもデザインの作成およびインプリメンテーションはサポートされますが、まず ISE プロジェクトを作成し、その ISE プロジェクトにエンベデッド プロセッサ ソースを追加するデザイン フローをお勧めします。EDK は、マイクロプロセッサ ハードウェア デザインを合成し、FPGA ターゲット デバイスにマップして、ビットストリームを生成およびダウンロードするのに ISE コンポーネントを使用します。

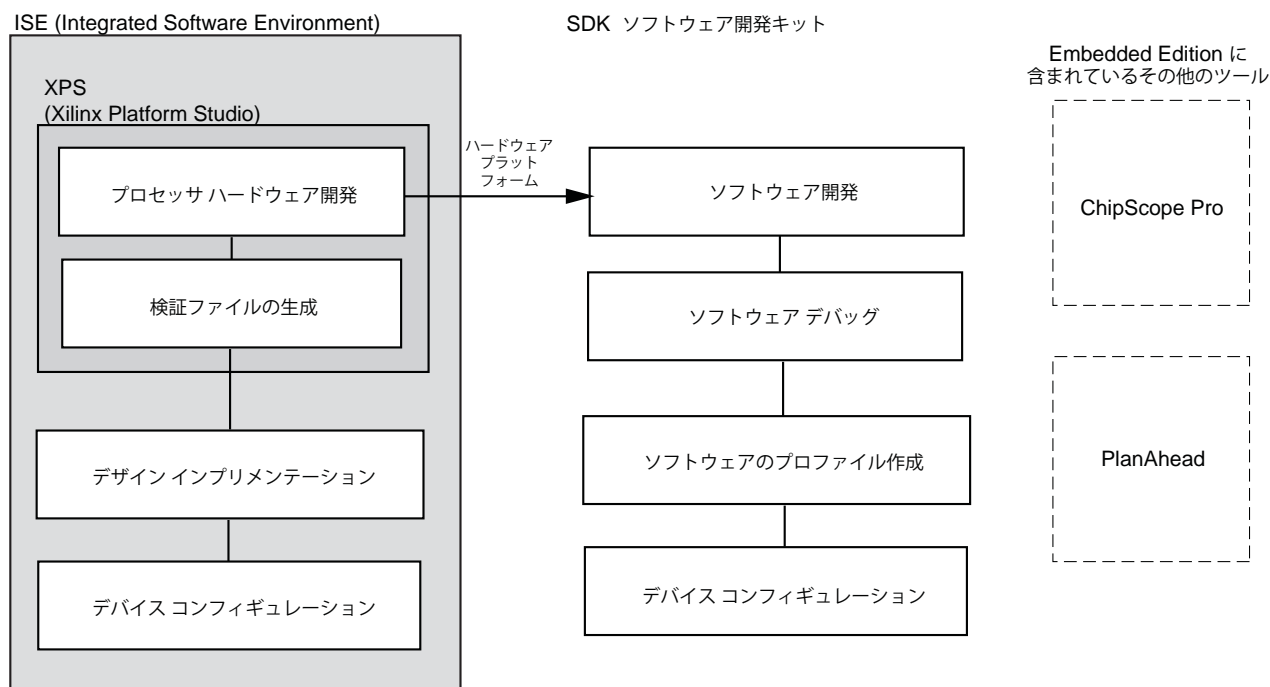
ISE の詳細は、ISE ソフトウェアのマニュアルおよびヘルプを参照してください。ISE ソフトウェア マニュアルおよびその他の情報へのリンクは、[4 ページの「その他のリソース」](#)にあります。

## 関連リソース

- 『Platform Specification Format Reference Manual』  
『OS and Libraries Document Collection』  
『BFM Simulation in Platform Studio』  
『EDK コンセプト、ツール、テクニック』  
[http://japan.xilinx.com/ise/embedded/edk\\_docs.htm](http://japan.xilinx.com/ise/embedded/edk_docs.htm)
- PowerPC (405 および 440) プロセッサ ブロック リファレンス ガイド  
[http://japan.xilinx.com/support/documentation/user\\_guides/ug018.pdf](http://japan.xilinx.com/support/documentation/user_guides/ug018.pdf)  
[http://japan.xilinx.com/support/documentation/user\\_guides/ug011.pdf](http://japan.xilinx.com/support/documentation/user_guides/ug011.pdf)
- 『MicroBlaze プロセッサ リファレンス ガイド』  
[http://japan.xilinx.com/support/documentation/sw\\_manuals/j\\_mb\\_ref\\_guide.pdf](http://japan.xilinx.com/support/documentation/sw_manuals/j_mb_ref_guide.pdf)

## 設計プロセスの概要

EDK に含まれるツールを使用すると、図 1-1 に示すように、エンベデッド システムの設計フローを最初から最後まで実行できます。



X11124

図 1-1：基本的なエンベデッド システム設計プロセスのフロー

## ハードウェア開発

ザイリンクスの **FPGA** 技術を使用すると、プロセッサ サブシステムのハードウェア ロジックをカスタマイズ可能です。標準的なマイクロプロセッサ チップまたはコントローラ チップでは、このようなカスタマイズは不可能です。

ハードウェア プラットフォームは、ザイリンクスの技術を使用して作成する柔軟性の高いエンベデッド プロセッサ サブシステムです。

ハードウェア プラットフォームは、プロセッサ バスに接続された 1 つまたは複数のプロセッサおよびペリフェラルで構成されています。ハードウェア プラットフォームは、**MHS (Microprocessor Hardware Specification)** ファイルで定義されます。

**MHS** ファイルはハードウェア プラットフォーム記述を含む **ASCII** テキスト ファイルで、エンベデッド システムのハードウェア コンポーネントを記述した主要なソース ファイルです。

## ソフトウェア開発

ソフトウェア プラットフォームは、アプリケーションを構築するソフトウェア ドライバおよび **OS** をまとめたものです。作成されるソフトウェア イメージには、ザイリンクス ライブラリのうちエンベデッド デザインで使用するもののみが含まれます。1 つのソフトウェア プラットフォーム上で実行する複数のアプリケーションを作成できます。

## 検証

**EDK** には、ハードウェアとソフトウェア両方の検証ツールが含まれています。使用可能な検証ツールは、次のとおりです。

### シミュレーションを使用したハードウェアの検証

ハードウェア プラットフォームの機能を検証するには、シミュレーション モデルを作成して **HDL** シミュレータ上で実行します。システムをシミュレーションすると、ソフトウェア プログラムがプロセッサで実行されます。ビヘイビア、構造、またはタイミング シミュレーション モデルを作成できます。

### デバッグによるソフトウェアの検証

ソフトウェアの検証には、次の方法があります。

- デザインをサポートされる開発ボードに読み込み、デバッグ ツールでターゲット プロセッサを制御します。
- 命令セット シミュレータ (**ISS**) をホスト コンピュータで実行し、コードをデバッグします。
- コード実行のプロファイルを作成して、システムのパフォーマンスを評価します。

## デバイスのコンフィギュレーション

ハードウェア プラットフォームおよびソフトウェア プラットフォームが完成したら、**FPGA** デバイス用にコンフィギュレーション ビットストリームを作成します。

- プロトタイプを作成では、ホスト コンピュータに接続した状態で、ビットストリームをエンベデッド プラットフォームで実行するソフトウェアと共にダウンロードします。
- 製品システムでは、コンフィギュレーション ビットストリームとソフトウェアを **FPGA** に接続した不揮発性メモリに保存します。

## EDK の概要

エンベデッド ハードウェア プラットフォームは、プロセッサ バスで接続された 1 つまたは複数のプロセッサ、さまざまなペリフェラル、およびメモリ ブロックで構成されています。また、デバイスの外部に接続するポートもあります。各プロセッサ コア (pcore またはプロセッサ IP と呼ばれる) には、その動作をカスタマイズするためのパラメータがあります。これらのパラメータで、ペリフェラルおよびメモリのアドレス マップも定義します。XPS を使用するとさまざまなオプションの機能を選択できるので、FPGA には作成するアプリケーションに必要な機能のサブセットをインプリメントするだけで済みます。

次の図に、エンベデッド システムの作成で EDK に含まれるツールがどのように使用されるかを示します。

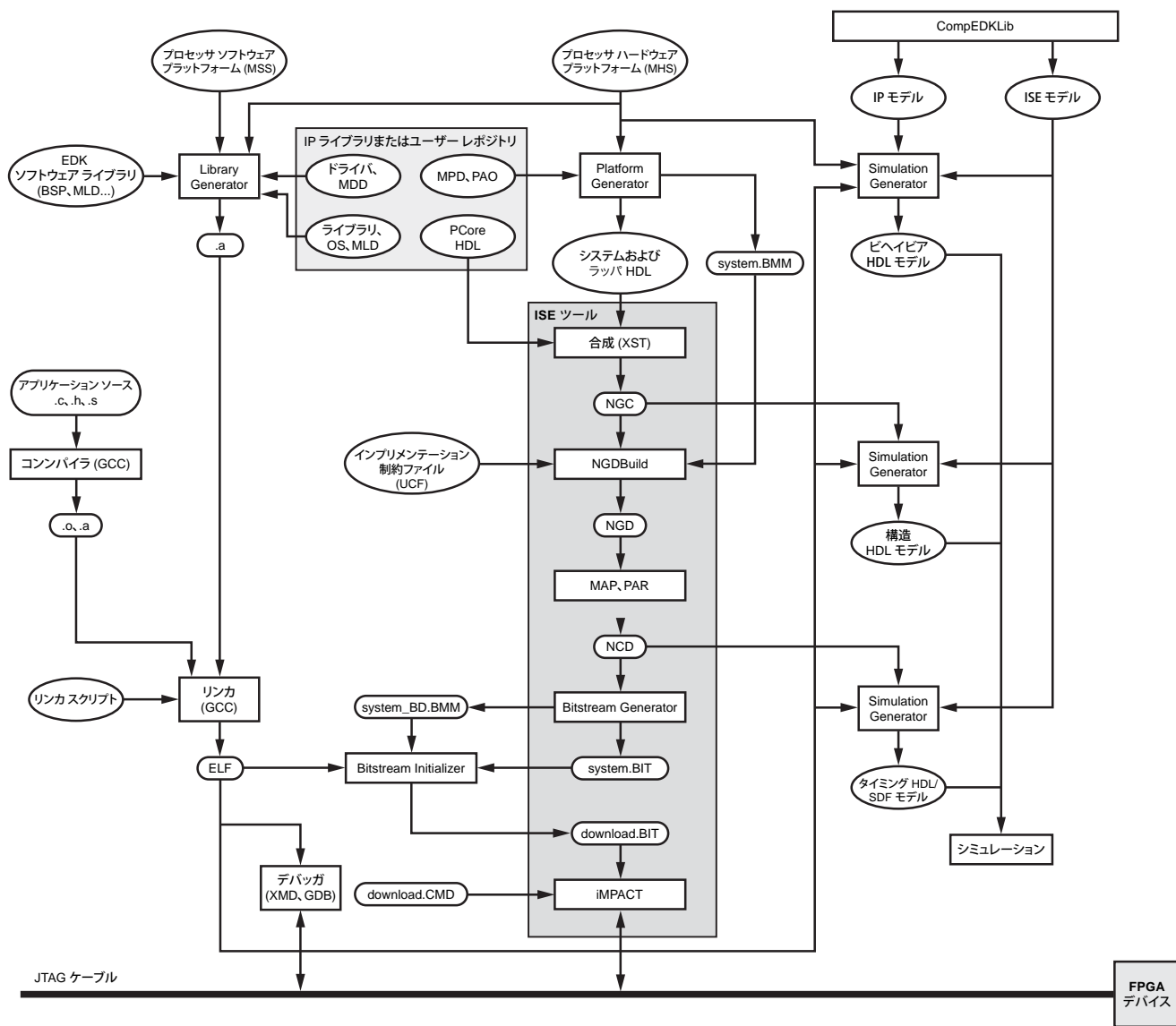


図 1-2：エンベデッド開発キット (EDK) ツールの構成

## EDK ツールおよびユーティリティ

次の表に EDK に含まれるツールおよびユーティリティを示し、その後各ツールの概要と詳細情報が記載されている章を示します。

表 1-1 : EDK ツールおよびユーティリティ

ハードウェア開発および検証	
Xilinx Platform Studio (XPS)	エンベデッド ハードウェア デザインを開発する統合開発環境 (GUI) です。
Base System Builder (BSB) ウィザード	サポートされている開発ボードの機能および一般的なエンベデッド システムの基本的な機能を使用して、エンベデッド デザインを短時間に構築します。プロジェクトの初期作成には、BSB ウィザードを使用することをお勧めします。
Create and Import Peripheral (CIP) Wizard	デザインに独自のペリフェラルを追加する際に使用します。XPS で必要な関連ディレクトリおよびデータ ファイルも作成されます。
コプロセッサ ウィザード	CPU にコプロセッサを追加する際に使用できます。MicroBlaze ベースのデザインでのみ使用可能です。
Platform Generator (Platgen)	チップ上のプログラマブル システムを HDL およびインプリメンテーション ネットリスト ファイルの形で生成します。
XPS コマンド ライン モード	エンベデッド デザイン フローの実行およびツール オプションの変更を、コマンド ラインから実行します。
バス ファンクション モデル (BFM)	実際のエンベデッド システムの代わりに使用するバス環境モデルを作成することにより、カスタム ペリフェラルの検証プロセスを簡略化します。
Simulation Model Generator (Simgen)	システムのハードウェア シミュレーション モデルおよびコンパイル スクリプト ファイルを生成します。
シミュレーション ライブラリ コンパイラ (CompLib)	デザインのビヘイビア シミュレーションを開始する前に、使用するシミュレータ用に EDK シミュレーション ライブラリをコンパイルします。
ソフトウェア開発および検証	
ソフトウェア開発キット (SDK)	ソフトウェア アプリケーション プロジェクトの開発に使用する統合開発環境 (GUI) です。
Library Generator (Libgen)	カスタマイズされたソフトウェア ライブラリ、ドライバ、および OS を使用してソフトウェア プラットフォームを作成します。
GNU コンパイラ ツール (GCC)	Libgen で作成されたプラットフォームに基づき、ソフトウェア アプリケーションを作成します。
Xilinx Microprocessor Debugger (XMD)	ソフトウェアをダウンロードおよびデバッグします。GNU デバッガがデバイスにアクセスするチャネルとしても機能します。
GNU デバッガ (GDB)	シミュレーション モデルまたはターゲット デバイスでソフトウェアをデバッグするための GUI を提供します。

表 1-1：EDK ツールおよびユーティリティ (続き)

Bitstream Initializer (BitInit)	オンチップの命令メモリがソフトウェア実行ファイルで初期化されるよう、FPGA コンフィギュレーション ビットストリームをアップデートします。
Debug Configuration Wizard	多くのデザインに共通するハードウェアおよびソフトウェア プラットフォームのデバッグ コンフィギュレーション タスクを自動化します。
System ACE ファイル ジェネレータ (GenACE)	製品システムのコンパクト フラッシュに保存される FPGA コンフィギュレーション ビットストリームおよびソフトウェア実行ファイルに基づいて、ザイリンクス System ACE™ コンフィギュレーション ファイルを生成します。
フラッシュ メモリ プログラム	ターゲット プロセッサを使用して、共通フラッシュ インターフェイス (CFI) に準拠したボード上のパラレル フラッシュ デバイスにソフトウェアおよびデータをプログラムします。
フォーマット リビジョン ツールおよび Version Management Wizard	プロジェクト ファイルを最新のフォーマットにアップデートします。Version Management Wizard は、以前のバージョンの EDK で作成した IP およびドライバを最新のバージョンに移行します。

## Xilinx Platform Studio (XPS)

XPS は、MicroBlaze および PowerPC プロセッサを使用したエンベデッド プロセッサシステムを開発するための統合環境です。ソース コードを作成/編集するためのエディタ、プロジェクト管理インターフェイスも備えています。XPS では、ツール フローの設定オプションをカスタマイズしたり、グラフィカル システム エディタを使用してプロセッサ、ペリフェラル、バスを接続することが可能です。XPS をバッチ モードで実行することも可能です。

XPS から、ハードウェア システム コンポーネントを処理するために必要なすべてのエンベデッド システム ツールを実行できます。また、システムの検証も XPS 環境内で実行できます。

XPS には、次の機能があります。

- プロセッサおよびペリフェラル コアの追加、コア パラメータの変更、バスおよび信号の接続を行い、MHS ファイルを作成
- MSS ファイルを生成および編集
- 23 ページの表 1-1 に示すツールをすべてサポート
- システムのブロック図およびデザイン レポートを生成および表示
- プロジェクト管理をサポート
- プロセスおよびツール フローの依存関係を管理
- SDK にインポートするためのハードウェア仕様ファイルをエクスポート

ファイルおよびそのフォーマットに関する詳細は、『Platform Specification Format Reference Manual』を参照してください。20 ページの「関連リソース」に、このマニュアルへのリンクがあります。

XPS の使用法の詳細は、XPS ヘルプを参照してください。次に、XPS のコンポーネントであるツールおよびユーティリティについて説明します。



## XPS コマンド ライン モード

OS のコマンド ラインから XPS を実行できます。XPS コマンド ライン モードの詳細は、[第 13 章「コマンド ライン モード」](#)を参照してください。

## Base System Builder (BSB) ウィザード

Base System Builder (BSB) ウィザードを使用すると、システムを短時間で構築できます。BSB ウィザードのみで完成できるエンベデッド デザイン プロジェクトもあります。より複雑なプロジェクトの場合は、BSB で基本システムを作成し、これをカスタマイズしてエンベデッド デザインを完成させることができます。BSB ウィザードを使用すると、サポートされるすべてのプロセッサ タイプで 1 つのプロセッサを含むデザイン、MicroBlaze で 2 つのプロセッサを含むデザインを作成できます。プロジェクトを効率的に作成するため、どの場合でもまず BSB ウィザードを使用することをお勧めします。

選択するボードに基づき、プロセッサ タイプ、デバッグ インターフェイス、キャッシュ設定、メモリのタイプとサイズ、ペリフェラルなどの基本システム要素を選択し、設定できます。各オプションに対してデフォルト値が選択されており、これを必要に応じて変更できます。

ターゲット開発ボードが BSB ウィザードでサポートされていない場合は、カスタム ボード オプションを使用します。このオプションを使用した場合、カスタム ボードに含まれるハードウェア デバイスを手動で指定できます。カスタム ボードで生成したシステムを実行するには、ユーザー制約 ファイル (UCF) に FPGA ピンのロケーション制約を入力します。サポートされているターゲット ボードを選択した場合は、BSB ウィザードによりこれらの制約が UCF に自動的に記述されます。

BSB ウィザードでは、1 つまたは複数のオプションのソフトウェア プロジェクトを生成できます。各プロジェクトには、コンパイルして開発ボード上のハードウェアで実行可能な、サンプル アプリケーションおよびリンカ スクリプトが含まれます。このアプリケーションは、システムが起動していることを検証し、単純なアプリケーションのテストを実行できるよう構成されています。このテスト アプリケーションの内容は、システムに含まれるコンポーネントによって異なります。

BSB ウィザードの機能の詳細な使用方法は、XPS ヘルプを参照してください。

## Create and Import Peripheral (CIP) Wizard

CIP ウィザードを使用すると、独自のペリフェラルを作成し、XPS 準拠のレポジトリまたは XPS プロジェクトにインポートできます。

Create モードではテンプレートが作成され、バス プロトコル、命名規則、XPS で必要なインターフェイス ファイルのフォーマットなどの詳細を理解していなくても、ペリフェラルを簡単にインプリメントできます。テンプレート ファイルの例を参照し、ウィザードで出力されるさまざまな補助デザイン サポート ファイルを使用することにより、カスタム ロジックを短時間で設計できます。

Import モードでは、XPS のさまざまなツールでペリフェラルを処理するために必要なインターフェイス ファイルとディレクトリ構造が作成されます。

Import モードでは、XPS の命名規則に従っていることを前提としています。インポートが終了すると、作成したペリフェラルが XPS ペリフェラル ライブラリに追加されます。

ペリフェラルを作成またはインポートすると、MPD (Microprocessor Peripheral Definition) および PAO (Peripheral Analyze Order) ファイルが自動的に生成されます。

- MPD ファイル : ペリフェラルのインターフェイスを定義します。
- PAO ファイル : Platgen および Simgen に対して、ペリフェラルのコンパイル (合成またはシミュレーション用) に必要な HDL ファイルとその解析順を指定します。

MPD および PAO ファイルの詳細は、『Platform Specification Format Reference Manual』を参照してください。20 ページの「関連リソース」に、このマニュアルへのリンクがあります。CIP ウィザードの機能の詳細な使用方法是、XPS ヘルプを参照してください。

## コプロセッサ ウィザード

このウィザードは、CPU にコプロセッサを追加する際に使用できます。コプロセッサは、FPGA にユーザー定義のファンクションをインプリメントするハードウェア モジュールで、高速シンプレックス リンク (FSL) インターフェイスを使用してプロセッサに接続します。FSL チャネルは、FIFO を使用してインプリメントされた専用のポイント ツー ポイント通信インターフェイスです。このウィザードは、MicroBlaze プロセッサ デザインでのみ使用可能です。

高速シンプレックス リンク (FSL) の詳細は、『MicroBlaze プロセッサ リファレンス ガイド』および PowerPC (405 および 440) プロセッサのガイドを参照してください。また、FSL バスのデータシートも参考になる場合があります。20 ページの「関連リソース」に、これらの資料へのリンクがあります。

コプロセッサ ウィザードの使用方法是、XPS ヘルプを参照してください。

## Platform Generator (Platgen)

Platform Generator (Platgen) は、エンベデッド プロセッサ システムの抽象度の高い記述を、FPGA デバイスにインプリメントできる HDL ネットリストにコンパイルします。

Platgen には、次の機能があります。

- デザイン入力として MHS ファイルを読み込みます。
- XPS プロジェクトおよびユーザー IP レポジトリから、プロセッサ コア (pcore) のハードウェア記述ファイル (MPD、PAO) を読み込みます。
- システムに含まれる pcore のすべてのインスタンスを統合したエンベデッド システムの最上位デザイン ファイルを生成します。このファイルの生成過程で、MHS ファイルに含まれる最上位バス接続を、プロセッサ、ペリフェラル、およびオンチップ メモリを接続するために必要な信号に変換します。Platgen で生成されたシステム レベルの HDL ネットリストは、FPGA のインプリメンテーション プロセスで使用されます。
- インスタンシエートされた各 pcore を合成するため、XST (Xilinx Synthesis Technology) を起動します。
- オンチップ ブロック RAM のアドレスおよびコンフィギュレーションを含む BMM (ブロック RAM メモリ マップ) ファイルを生成します。このファイルは、ブロック RAM をソフトウェアで初期化する際に使用されます。

Platgen の詳細は、第 2 章「Platform Generator (Platgen)」を参照してください。

## Debug Configuration Wizard

多くのデザインに共通するハードウェアおよびソフトウェア プラットフォームのデバッグ コンフィギュレーション タスクを自動化します。

ChipScope™ コアをインスタンシエートすると、プロセッサ ローカル バス (PLB) またはその他のシステム レベル信号を監視できます。また、既存の ChipScope コアのパラメータをハードウェア デバッグ用にも設定できます。JTAG ベースの仮想入力および出力を共有することも可能です。

デバッグ用にソフトウェアを設定するには、プロセッサ デバッグ パラメータを設定します。ChipScope コアの協調デバッグをイネーブルにした場合は、ソフトウェア デバッガ信号とハードウェア信号の協調トリガを設定できます。JTAG インターフェイスは、UART 信号を XMD (Xilinx Microprocessor Debugger) に送信するよう設定できます。

Debug Configuration Wizard の使用方法は、XPS ヘルプを参照してください。

## Simulation Model Generator (Simgen)

Simulation Model Generator (Simgen) は、ハードウェア用のさまざまなシミュレーション モデルを生成するツールです。ビヘイビア モデルを生成する場合は入力として MHS ファイルが読み込まれ、構造モデルを生成する場合は合成後のデザイン データベース、タイミング モデルを生成する場合は配置配線済みのデザイン データベースが読み込まれます。また、オンチップ メモリを初期化するために各プロセッサのエンベデッド アプリケーション実行ファイル (ELF) も読み込まれ、シミュレーション中にモデル化されたプロセッサでソフトウェア コードが実行されるようにします。

詳細は、第 3 章「[Simulation Model Generator \(Simgen\)](#)」を参照してください。

## バス ファンクション モデル (BFM)

バス ファンクション モデル シミュレーションは、バスに接続されたハードウェア コンポーネントの検証を簡略化します。バス ファンクション モデルの詳細は、『[BFM Simulation in Platform Studio](#)』を参照してください。20 ページの「[関連リソース](#)」に、このマニュアルへのリンクがあります。

## Platform Specification Utility (PsfUtility)

EDK で認識される IP コアを作成するために必要な MPD (Microprocessor Peripheral Definition) ファイルを自動生成します。このツールの機能は、CIP ウィザードで使用されます。

詳細は、第 5 章「[Platform Specification Utility \(PsfUtil\)](#)」を参照してください。

## ソフトウェア開発キット (SDK)

SDK は、Eclipse オープン ソース標準に基づくソフトウェア アプリケーション プロジェクトの開発環境です。SDK には、次の機能があります。

- ISE および XPS をインストールせずに少ない容量でインストール可能
- 1 つまたは複数のプロセッサ システムのソフトウェア アプリケーション開発をサポート
- XPS で生成したプラットフォーム定義をインポート
- チーム環境でのソフトウェア アプリケーションの開発をサポート
- CVS に基づくソース コード リビジョンをサポート
- サードパーティ OS 用にソフトウェア プラットフォームおよびボード サポート パッケージ (BSP) を作成および設定可能
- ハードウェアとソフトウェアの機能をテストする既製のサンプル ソフトウェア プロジェクトを提供
- 使いやすい GUI インターフェイスでソフトウェア アプリケーション用のリンカ スクリプトの作成、FPGA デバイスのプログラム、パラレル フラッシュ メモリのプログラムを実行
- 高度な C/C++ コード エディタおよびコンパイル環境
- プロジェクト管理機能

- アプリケーション構築コンフィギュレーションおよび makefile の自動生成
- エラー ナビゲーション
- エンベデッド ターゲットのデバッグおよびプロファイル作成をスムーズに行う統合環境

SDK の詳細は、SDK ヘルプを参照してください。

## Library Generator (Libgen)

Libgen は、エンベデッド プロセッサ システム用にライブラリ、デバイスドライバ、ファイルシステム、および割り込みハンドラを設定し、ソフトウェア プラットフォームを作成します。ソフトウェア プラットフォームは、各プロセッサに対し、ハードウェア プラットフォームに含まれるペリフェラルのドライバ、ライブラリ、標準入力および出力デバイス、割り込みハンドラ ルーチン、その他のソフトウェア機能を定義します。SDK プロジェクトでは、さらに各プロセッサ上で実行されるソフトウェア アプリケーションも定義します。これらのソフトウェア アプリケーションは、ソフトウェア プラットフォームに基づいています。

SDK では、インストールされるライブラリとドライバ、およびカスタム ペリフェラル用のライブラリとドライバを使用して、ライブラリとドライバを含むソフトウェア アプリケーションがプロセッサ ハードウェア プラットフォームで実行できる ELF (Executable Linked Format) ファイルにコンパイルされます。

Libgen は、EDK ライブラリおよびユーザー IP レポジトリから、選択されたライブラリおよびプロセッサ コア (pcore) のソフトウェア記述ファイル (MDD (Microprocessor Driver Definition) およびドライバコード) を読み込みます。

詳細は、第 4 章「[Library Generator \(Libgen\)](#)」および XPS ヘルプを参照してください。ライブラリおよびドライバの詳細は、『OS and Libraries Document Collection』の各ソフトウェア コンポーネントに関する説明を参照してください。20 ページの「[関連リソース](#)」に、このマニュアルへのリンクがあります。

## GNU コンパイラ ツール (GCC)

システムに含まれる各プロセッサのアプリケーション実行ファイルをコンパイルおよびリンクするため、GNU コンパイラ ツールが呼び出されます。プロセッサによって、次のコンパイラが使用されます。

- mb-gcc コンパイラ (MicroBlaze プロセッサ用)
- powerpc-eabi-gcc コンパイラ (PowerPC プロセッサ用)

GNU コンパイラ ツールでは、次の操作が実行されます (22 ページの図 1-2 を参照)。

- コンパイラでは、ターゲット プロセッサ用の C ソース ファイルとヘッダ ファイルまたはアセンブラ ソース ファイルが読み込まれます。
- リンカでは、コンパイルされたアプリケーションと選択されたライブラリが統合され、ELF フォーマットの実行ファイルが生成されます。リンカ スクリプトも読み込まれます。ツールで生成されたデフォルトのリンカ スクリプト、またはユーザーが作成したリンカ スクリプトが使用されます。

GNU コンパイラおよびそのユーティリティの詳細は、第 9 章「[GNU コンパイラ ツール](#)」および付録 A「[GNU ユーティリティ](#)」を参照してください。

## Xilinx Microprocessor Debugger (XMD)

プログラムのデバッグは、ソフトウェア上で命令セット シミュレータを (ISS) 使用するか、ハードウェア ビットストリームを読み込んだザイリンクス FPGA を搭載するボード上で行います。22 ページの図 1-2 に示すように、デバッガ ユーティリティ (XMD) はアプリケーションの実行ファイル (ELF) を読み込みます。FPGA 上でのデバッグでは、FPGA をビットストリームでコンフィギュレーションする際に使用するのと同じダウンロード ケーブルを使用します。詳細は、第 11 章「Xilinx Microprocessor Debugger (XMD)」を参照してください。

## GNU デバッガ (GDB)

GNU デバッガ (GDB) は、さまざまな開発段階で MicroBlaze および PowerPC システムをデバッグ/検証するためのインターフェイスを提供する、高性能で柔軟性の高いツールです。

GDB は、プロセッサとの通信に XMD (Xilinx® Microprocessor Debugger) を使用します。

詳細は、第 10 章「GNU デバッガ (GDB)」を参照してください。

## シミュレーション ライブラリ コンパイラ (CompLib)

CompLib は、さまざまなシミュレータ ベンダーのツールを使用して、EDK HDL に基づくシミュレーション ライブラリをコンパイルします。このユーティリティは、GUI およびバッチ モードの両方で実行できます。GUI モードでは、EDK のライブラリを使用して、ISE インストール ディレクトリにあるザイリンクス ライブラリもコンパイルできます。

CompLib の詳細は、第 3 章の「シミュレーション モデル」を参照してください。シミュレーション ライブラリのコンパイル方法は、XPS ヘルプを参照してください。

## Bitstream Initializer (BitInit)

BitInit は、プロセッサに接続されているオンチップ ブラック RAM メモリをソフトウェア情報で初期化します。このユーティリティは、ISE ツールで生成されたハードウェアのみのビットストリーム (system.bit) を読み込み、各プロセッサのエンベデッド アプリケーション実行ファイル (ELF) を含むビットストリーム (download.bit) を生成します。Platgen で生成され、各 BRAM ブロックの物理的な配置情報を使用して ISE ツールでアップデートされた BMM ファイルが使用されます。BitInit は、Data2MEM ユーティリティを使用してビットストリーム ファイルをアップデートします。

BitInit が EDK を使用した設計フローのどこで使用されるかについては、22 ページの図 1-2 を参照してください。詳細は、第 8 章「Bitstream Initializer (BitInit)」を参照してください。

## System ACE ファイル ジェネレータ (GenACE)

XPS は、FPGA のビットストリーム、ELF ファイル、およびデータ ファイルから、System ACE コンフィギュレーション ファイルを生成します。生成された System ACE ファイルは、FPGA のコンフィギュレーション、ブロック RAM の初期化、有効なプログラムまたはデータを使用した外部メモリの初期化、および製品システムでのプロセッサの起動に使用できます。EDK には、XMD コマンドを使用して ACE ファイルを作成する Tcl スクリプト genace.tcl が含まれています。ACE ファイルは、MDM (Microprocessor Debug Module) システムを使用して PowerPC プロセッサおよび MicroBlaze プロセッサ用に生成できます。

詳細は、第 12 章「System ACE ファイル ジェネレータ (GenACE)」を参照してください。

## フラッシュ メモリ プログラマ

フラッシュ メモリ プログラマは、さまざまなフラッシュ ソフトウェアおよびレイアウトに対応するよう設計されています。第 7 章「フラッシュ メモリのプログラム」を参照してください。

## フォーマット リビジョン ツールおよび Version Management Wizard

フォーマット リビジョン ツール (revup) は、既存の EDK プロジェクトを現在のバージョンのフォーマットにアップデートします。revup では、フォーマットのみが変更され、デザインはアップデートされません。フォーマットを変更する前に、プロジェクト ファイル (XMP)、MHS、MSS など既存のファイルのバックアップが作成されます。

以前のバージョンのプロジェクトをそれより新しいバージョンの EDK で開くと (EDK 10.1 で作成したプロジェクトを EDK 11.1 で開くなど)、Version Management Wizard が起動します。

Version Management Wizard は、フォーマットが変更された後に起動します。このウィザードには、デザインで使用されているザイリンクス プロセッサ IP の変更に関する情報が表示されます。新しい互換性のある IP が入手可能な場合は、新しいバージョンにアップデートするようメッセージが表示されます。

Version Management Wizard の使用方法是、第 6 章「バージョン管理ツール (revup)」および XPS ヘルプを参照してください。

# Platform Generator (Platgen)

---

Platform Generator (Platgen) は、エンベデッド プロセッサ システムをハードウェア ネットリスト (HDL) および EDIF (Electronic Data Interchange Format) ファイルの形でカスタマイズし、生成するツールです。

デフォルトでは、エンベデッド ハードウェア デザインに含まれるプロセッサの IP コア インスタンスが、XST を使用して合成されます。Platgen は、すべての IP コアを相互接続するシステム レベルの HDL ファイルを生成し、ISE® のインプリメンテーション フローで合成できるようにします。

この章には、次のセクションが含まれています。

- [機能](#)
- [関連リソース](#)
- [ツール要件](#)
- [Platgen の使用法](#)
- [Platgen のコマンド オプション](#)
- [ディレクトリ構造](#)
- [出力ファイル](#)
- [合成ネットリスト キャッシュ](#)

## 機能

Platgen の機能は、次のとおりです。

- チップ上のプログラマブル システムをハードウェア ネットリスト (HDL およびインプリメンテーション ネットリスト ファイル) の形で生成します。
- MHS (Microprocessor Hardware Specification) ファイルを入力として使用し、ハードウェア プラットフォームを作成します。
- さまざまなフォーマットのネットリスト ファイル (NGC、EDIF など) を作成します。
- この後の段階で使用するツールのサポート ファイル、ハードウェア プラットフォームにコンポーネントを追加できるようにする最上位 HDL ラップ ファイルを作成します。

Platgen を実行した後、XPS からインプリメンテーション ツールを実行する ISE のユーザー インターフェイスを表示し、FPGA インプリメンテーション ツールでハードウェアのインプリメンテーションを実行します。インプリメンテーションの最後に、FPGA をコンフィギュレーションするビットストリームが生成されます。このビットストリームには、FPGA チップ上にある BRAM メモリの初期化情報が含まれます。スタートアップ時にこれらのメモリにユーザー コードまたはデータが必要な場合は、Data2MEM を使用して、ソフトウェア アプリケーションの生成および検証で生成される実行ファイルのコードおよびデータで、ビットストリームをアップデートします。



## 関連リソース

『Platform Specification Format Reference Manual』  
[http://japan.xilinx.com/ise/embedded/edk\\_docs.htm](http://japan.xilinx.com/ise/embedded/edk_docs.htm)

## ツール要件

ザイリンクス開発システムを使用するようシステムを設定する必要があります。システムが正しく設定されていることを確認してください。詳細は、『インストール、ライセンス、およびリリースノート』を参照してください。

## Platgen の使用法

Platgen を実行するには、次の構文を使用します。

```
platgen -p <partname> system.mhs
```

**platgen** : ツールの実行ファイル名

**-p** : デバイス パーツを指定するオプション

**<partname>** : パーツ名

**system.mhs** : 出力ファイル名

## Platgen のコマンド オプション

表 2-1 に、Platgen のコマンド オプションを示します。

表 2-1 : Platgen のコマンド オプション

オプション	コマンド	説明
ヘルプの表示	<b>-h</b> 、 <b>-help</b>	コマンドの使用方法を表示します。
バージョン情報の表示	<b>-v</b>	Platgen のバージョン番号を表示します。
オプション ファイルの指定	<b>-f</b> <filename>	コマンド ライン引数およびオプションの情報を、指定したファイルから読み込みます。
統合スタイル	<b>-intstyle</b> {ise default}	フローまたはプロジェクト環境内でザイリンクス アプリケーションを起動する際のコンテキスト情報を指定します。
HDL 言語の指定	<b>-lang</b> {verilog vhdl}	出力の HDL 言語を指定します。 デフォルト : vhdl
ログ ファイルの指定	<b>-log</b> <logfile[.log]>	ログ ファイルを指定します。 デフォルト : platgen.log
ユーザー ペリフェラルおよびドライバレポジトリのライブラリ検索パスの指定	<b>-lp</b> <library_path>	IP 検索ディレクトリに <library_path> を追加します。ライブラリはレポジトリ領域の集合です。
出力ディレクトリの指定	<b>-od</b> <output_dir>	出力ディレクトリ パスを指定します。 デフォルト : 現在のディレクトリ



表 2-1 : Platgen のコマンド オプション (続き)

オプション	コマンド	説明
デバイス名の指定	<b>-p</b> <partname>	デザインのインプリメンテーションで使用するデバイスを指定します。
インスタンス名の指定	<b>-ti</b> <instname>	最上位インスタンス名を指定します。
最上位モジュール名の指定	<b>-tm</b> <top_module>	最上位モジュール名を指定します。
最上位レベル	<b>-toplevel</b> {yes no}	入力デザインでデザイン全体を表すか、階層のあるレベルを表すかを指定します。 デフォルト : yes

## ディレクトリ構造

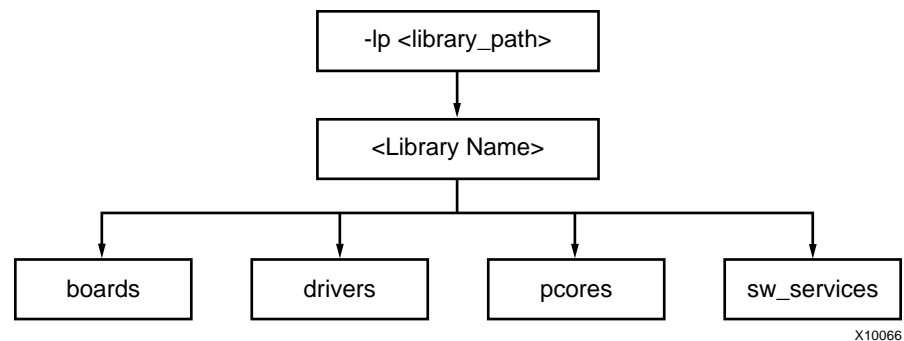
ペリフェラルのディレクトリ構造を、図 2-1 に示します。

追加のディレクトリを指定するには、次のいずれかの方法を使用します。

- Platgen が起動された現在のディレクトリを使用する。
- EDK ツールの **-lp** オプションを使用する。

Platgen では、次の順にペリフェラルが検索されます。

1. プロジェクト ディレクトリ内の pcores ディレクトリ
2. **-lp** オプションで指定された <library\_path>/<Library Name>/pcores ディレクトリ
3. XILINX\_EDK/hw/<Library Name>/pcores ディレクトリ



X10066

図 2-1 : ペリフェラルのディレクトリ構造

pcores ディレクトリ内では、各ペリフェラルはペリフェラル名 (<peripheral\_name>) のルートディレクトリに含まれます。ルート ディレクトリ内では、次のような構造になっています。

```

data/
hdl/
netlist/
  
```

## 出力ファイル

Platgen では、次のディレクトリとファイルが生成されます。プロジェクト ディレクトリからは、次のような構造になっています。

```
/hdl
/implementation
/synthesis
```

### hdl ディレクトリ

hdl ディレクトリには、次のファイルが含まれます。

- `system.[vhd|v]` : MHS で定義されたエンベデッド プロセッサ システムの HDL ファイル。プロジェクトの最上位ファイルとなります。
- `system_stub.[vhd|v]` : システムのインスタンス化の最上位テンプレート HDL ファイル。このファイルは、最上位 HDL ファイルの開始点として使用できます。
- `<inst>_wrapper.[vhd|v]` : MHS で定義された IP コンポーネントの HDL ラッパ ファイル。

### implementation ディレクトリ

implementation ディレクトリには、ペリフェラルのインプリメンテーション ネットリスト ファイル `peripheral_wrapper.ngc` が含まれます。

### synthesis ディレクトリ

synthesis ディレクトリには、次のファイルが含まれます。

```
system.[prj|scr]
```

## BMM フロー

PlatGen は、`<proj>/implementation` ディレクトリに `<system>.bmm` および `<system>_stub.bmm` を生成します。

- `<system>.bmm` : EDK が最上位システムの場合にインプリメンテーション ツールで使用されます。
- `<system>_stub` : EDK が最上位システムのサブモジュールである場合にインプリメンテーション ツールで使用されます。

Data2MEM を使用した EDK インプリメンテーション ツール フローを次に示します。

```
ngdbuild -bm <system>.bmm <system>.ngc
map
par
bitgen -bd <system>.elf
```

BitGen の出力 `<system>_bd.bmm` には、ブロック RAM の物理的なロケーションが含まれます。

BMM (ブロック RAM メモリ マップ) ファイルには、各ブロック RAM がどのように連続した論理データ空間を構成するかが記述されています。

Data2MEM には、`<system>_bd.bmm` と `<system>.bit` ファイルを入力します。Data2MEM は、連続するデータ片を Virtex シリーズのブロック RAM の初期化レコードに変換します。

## 合成ネットリスト キャッシュ

次のいずれかの変更を加えると、IP が再構築されます。

- インスタンス名の変更
- パラメータ値の変更
- コア バージョンの変更
- コアが MPD の CORE\_STATE=DEVELOPMENT オプションで指定されている
- コア ライセンスの変更



# Simulation Model Generator (Simgen)

---

この章では、HDL シミュレーションの基礎と、Simulation Model Generator (Simgen) および Compplib ユーティリティの使用法について説明します。この章には、次のセクションが含まれています。

- [Simgen の概要](#)
- [関連リソース](#)
- [シミュレーション ライブラリ](#)
- [Compplib ユーティリティ](#)
- [シミュレーション モデル](#)
- [Simgen の構文](#)
- [出力ファイル](#)
- [メモリの初期化](#)
- [テストベンチ](#)
- [デザインのシミュレーション](#)
- [制限](#)

## Simgen の概要

Simgen は、ハードウェア用のさまざまな VHDL および Verilog シミュレーション モデルを生成し、コンフィギュレーションするツールです。このツールでは、ハードウェアのインスタンス化と接続を記述する MHS (Microprocessor Hardware Specification) ファイルを入力として使用します。

また Simgen では、特定のベンダーのシミュレーション ツール用にスクリプトを作成できます。このスクリプトを使用して、生成されたシミュレーション モデルをコンパイルします。

ハードウェア コンポーネントは、MHS ファイルで定義されます。詳細は、『Platform Specification Format Reference Manual』の「Microprocessor Hardware Specification (MHS)」の章を参照してください。38 ページの「[関連リソース](#)」に、このマニュアルへのリンクがあります。シミュレーションの基礎およびビヘイビア、構造、タイミング シミュレーションについては、XPS ヘルプを参照してください。

## 関連リソース

- 『Platform Specification Format Reference Manual』  
[http://japan.xilinx.com/ise/embedded/edk\\_docs.htm](http://japan.xilinx.com/ise/embedded/edk_docs.htm)
- 『コマンド ライン ツール リファレンス ガイド』および『合成/シミュレーション デザイン ガイド』  
[http://japan.xilinx.com/support/software\\_manually.htm](http://japan.xilinx.com/support/software_manually.htm)

## シミュレーション ライブラリ

EDK シミュレーション ネットリストでは、ザイリンクス FPGA で使用可能な下位レベルのハードウェア プリミティブが使用されます。ザイリンクスでは、これらのプリミティブのシミュレーション モデルをこのセクションにリストするライブラリで提供しています。

次に示すライブラリは、ザイリンクスのシミュレーション フローで使用できます。HDL コードでは、正しくコンパイルされたライブラリを参照する必要があります。HDL シミュレータにより、論理ライブラリからコンパイルされたライブラリの物理的なロケーションにマップされます。

### ザイリンクス ISE ライブラリ

ザイリンクス ISE® ライブラリは、Compplib ユーティリティを使用してコンパイルできます。Compplib の詳細は、『コマンド ライン ツール リファレンス ガイド』を参照してください。ザイリンクス ISE シミュレーション ライブラリのコンパイルおよび使用方法は、『合成/シミュレーション デザイン ガイド』の「デザインのシミュレーション」の章を参照してください。38 ページの「関連リソース」に、これらのマニュアルへのリンクがあります。

ISE には、シミュレーション用に次のライブラリが含まれます。

- UNISIM ライブラリ
- SIMPRIM ライブラリ
- XilinxCoreLib ライブラリ

### UNISIM ライブラリ

UNISIM ライブラリは、ビヘイビア シミュレーションおよび構造シミュレーションで使用するファンクション モデルのライブラリです。このライブラリには、主な合成ツールで推論されるザイリンクス ユニファイド ライブラリのコンポーネントがすべて含まれます。また、I/O やメモリ セルなど、よくインスタンス化されるコンポーネントも含まれています。

デザイン (VHDL または Verilog) に UNISIM ライブラリ コンポーネントをインスタンス化し、ビヘイビア シミュレーションを実行します。Simgen で生成される構造シミュレーション モデルにより、UNISIM ライブラリ コンポーネントがインスタンス化されます。

UNISIM ライブラリの非同期コンポーネントは、遅延がゼロです。同期コンポーネントには、レース状態を回避するため、ユニット遅延が含まれます。同期コンポーネントの clock-to-out 遅延は 100ps です。

### SIMPRIM ライブラリ

SIMPRIM ライブラリは、タイミング シミュレーションで使用するライブラリです。このライブラリには、ザイリンクス インプリメンテーション ツールで使われるザイリンクス プリミティブ ライブラリのコンポーネントがすべて含まれます。Simgen で生成されるタイミング シミュレーション モデルにより、SIMPRIM ライブラリ コンポーネントがインスタンス化されます。

## XilinxCoreLib ライブラリ

ザイリンクス CORE Generator™ ソフトウェアは、FIR フィルタ、FIFO、CAM などの高度なモジュール (IP) を作成するためのグラフィカル ツールです。モジュールをカスタマイズおよび最適化することにより、ブロック乗算器、SRL、高速キャリー ロジック、オンチップのシングル/デュアルポート RAM などのザイリンクス FPGA デバイスのアーキテクチャ機能を最大限に活用できます。

CORE Generator ソフトウェアの HDL ライブラリ モデルは、ビヘイビア シミュレーションで使用されます。適切な HDL モデルを選択して、HDL デザインに統合します。これらのモデルでは、グローバル信号にライブラリ コンポーネントは使用されません。

## ザイリンクス EDK ライブラリ

EDK ライブラリは、ビヘイビア シミュレーションで使用されます。ModelSim SE/PE または NC-Sim 用にあらかじめコンパイルされた EDK IP コンポーネントがすべて含まれます。このライブラリを使用すると、プロジェクトごとに EDK コンポーネントをコンパイルする必要がなく、コンパイルにかかる時間を節約できます。EDK IP コンポーネント ライブラリは VHDL でのみ提供されており、暗号化されている場合があります。

Compplib を使用すると、EDK IP コンポーネントのコンパイルされたモデルが共有ロケーションに配置されます。暗号化されていない EDK IP は、Compplib を使用してコンパイルできます。暗号化されているコンポーネントには、コンパイル済みのライブラリが提供されています。

## EDK ライブラリの検索順

コンパイル済みのライブラリは、現在のプロジェクトの simlib ディレクトリから検索されます。

Simgen で現在のプロジェクトに含まれるコンパイル済みライブラリが検索され、使用されるようにするには、ディレクトリ構造を次の例のようにする必要があります。

```
<project directory>/
  simlib/
    mti/
      mycore_v1_00_a
    ncsim/
      mycore_v1_00_a
```

## Compplib ユーティリティ

ザイリンクスでは、ザイリンクスのサポートするシミュレータに対して HDL ライブラリをコンパイルする Compplib ユーティリティを提供しています。このユーティリティを使用すると、シミュレータ ベンダーの提供するツールを使用して、サポート デバイスに対して UNISIM、SIMPRIM、XilinxCoreLib ライブラリをコンパイルできます。Compplib を使用して HDL ライブラリをコンパイルするには、インプリメンテーション ツールをインストールしておく必要があります。

使用可能なオプションとその簡単な説明を表示するには、Compplib を -help オプションを使用して実行します。

```
compplib -help
```

シミュレータによって使用される環境変数が異なるため、Compplib を起動する前に設定しておく必要があります。環境変数が使用するシミュレータ用に設定されているかどうかを確認するには、それぞれのシミュレータのマニュアルを参照してください。

**メモ :** ModelSim の実行ファイルがパスに入っていない場合は、`-p <simulator_path>` を使用してそのディレクトリを指定してください。

次は、ModelSim SE (mti\_se) を使用してザイリンクス ライブラリをコンパイルする場合のコマンド例です。

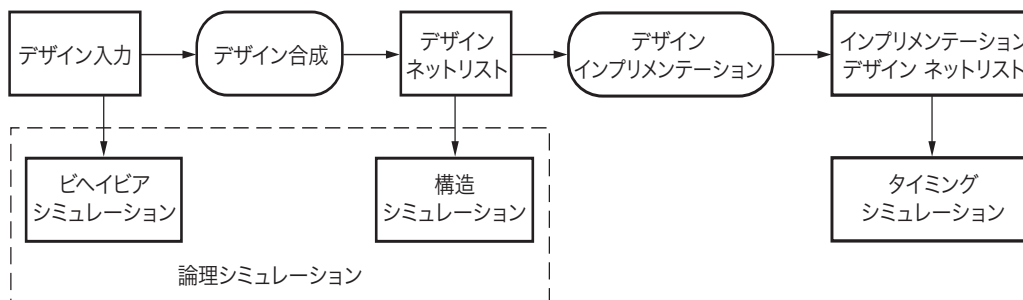
```
compplib -s mti_se -f all -l vhdl -w -o .
```

このコマンドで、必要なザイリンクス ライブラリがすべて現在の作業ディレクトリにコンパイルされます。Compplib の詳細は、『コマンド ライン ツール リファレンス ガイド』を参照してください。ザイリンクス ISE シミュレーション ライブラリのコンパイルおよび使用方法は、『合成/シミュレーション デザイン ガイド』の「デザインのシミュレーション」の章を参照してください。38 ページの「関連リソース」に、このマニュアルへのリンクがあります。

## シミュレーション モデル

このセクションでは、FPGA の 3 種類のシミュレーション モデルをインプリメントする方法と、XPS のバッチ モードを使用してシミュレーション モデルを作成する方法を説明します。図 3-1 に示すように、デザインプロセスの特定の段階で、その段階でのシミュレーションに必要なシミュレーション モデルが作成されます。

次の図に、FPGA デザインのシミュレーション段階を示します。



UG111\_01\_111903

図 3-1 : FPGA デザインのシミュレーション段階

## ビヘイビア モデル

ビヘイビア シミュレーション モデルを生成するには、図 3-2 に示すように、Simgen に MHS ファイルを入力します。Simgen は、デザインの機能をシミュレーションするための HDL ファイルを生成します。また、特定のベンダーのシミュレータ用にコンパイル スクリプトを生成できます。

デザイン内のプロセッサに関連付けられたブロック RAM を初期化するデータを含む HDL ファイルを生成するよう指定することも可能です。この初期化データは、既存の ELF (Executable Linked Format) から取得されます。



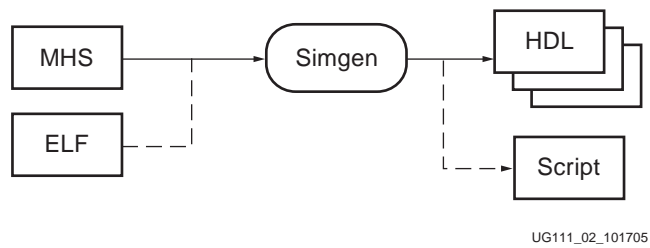


図 3-2 : ビヘイビア シミュレーション モデルの生成

## 構造モデル

構造シミュレーション モデルを生成するには、図 3-3 に示すように、Simgen に MHS ファイルと合成後のネットリスト ファイルを入力します。これらのネットリスト ファイルから、デザインの機能を構造的にシミュレーションするための HDL ファイルが生成されます。

また、特定のベンダーのシミュレータ用にコンパイル スクリプトを生成することも可能です。

Simgen を、デザイン内のプロセッサに関連付けられたブロック RAM を初期化するデータを含む HDL ファイルを生成するようにも指定できます。初期化データは、既存の実行ファイル (ELF) から取得されます。次の図に、構造シミュレーション モデルの生成を示します。

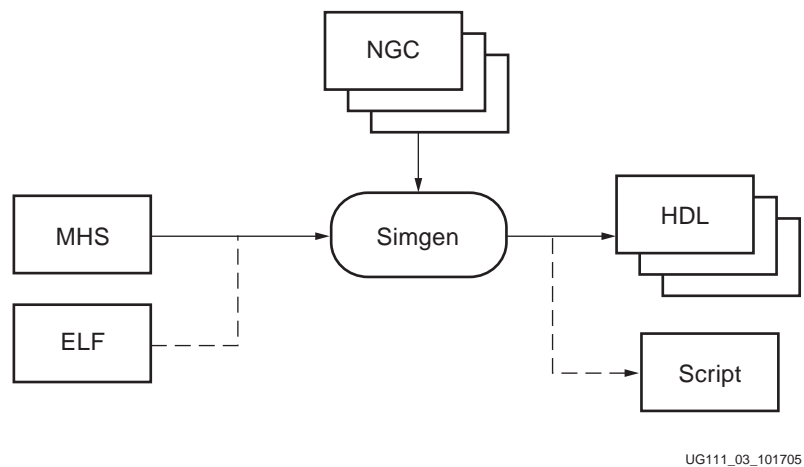
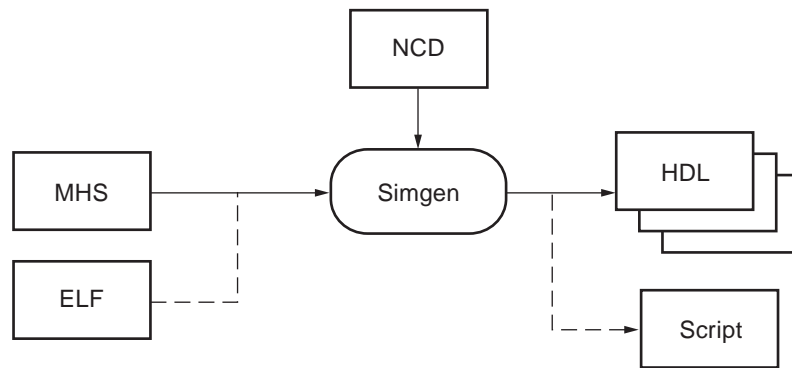


図 3-3 : 構造シミュレーション モデルの生成

**メモ :** EDK 設計フローはモジュール フローです。Simgen では、Platgen で生成されたネットリスト ファイルを使用して、構造シミュレーション モデルを生成します。

## タイミング モデル

タイミング シミュレーション モデルを生成するには、図 3-4 に示すように、Simgen に MHS ファイルとインプリメント後のネットリスト ファイルを入力します。Simgen は、デザインをシミュレーションするための HDL ファイルとタイミング情報を含む標準データ フォーマット (SDF) ファイルを生成します。また、特定のベンダーのシミュレータ用にコンパイル スクリプトを生成するよう指定したり、デザイン内のプロセッサに関連付けられたブロック RAM を初期化するデータを含む HDL ファイルを生成するよう指定することも可能です。初期化データは、既存の実行ファイル (ELF) から取得されます。



UG111\_04\_101705

図 3-4 : タイミング シミュレーション モデルの生成

## 単一言語モデルと混合言語モデル

Simgen では、シミュレーション用のビヘイビア ファイルで混合言語のコンポーネントを使用できます。デフォルトでは、コンポーネントを記述した言語が使用されます。各コンポーネントで言語を混合することはできません。混合言語機能を使用するには、混合言語をサポートするシミュレータが必要です。

ザイリンクスの IP コンポーネントは、VHDL で記述されています。シミュレータが混合言語をサポートしていない場合は、Simgen ですべての HDL ファイルをサポートされている言語に変換し、単一言語のモデルを生成できます。変換された HDL ファイルは、構造ファイルです。

構造シミュレーション モデルおよびタイミング シミュレーション モデルは、常に 1 つの言語で記述されます。

## XPS のバッチ モードを使用したシミュレーション モデルの作成

1. XMP ファイルを読み込み、プロジェクトを開きます。  

```
load xmp <filename>.xmp
```
2. XPS のプロンプトで次のシミュレーション値を設定します。
  - a. 次のコマンドを使用してシミュレータを選択します。  

```
xset simulator [ mti | ncs | none ]
```
  - b. 次のコマンドを使用して、既にコンパイルされているザイリンクスおよび EDK ライブラリへのパスを指定します。  

```
xset sim_x_lib <path>
xset sim_edk_lib <path>
```
  - c. 次のコマンドを使用してシミュレーション モデルを選択します。  

```
xset sim_model [ behavioral | structural | timing ]
```
3. 次のコマンドを入力して、シミュレーション モデルを生成します。  

```
run simmodel
```

プロセスが終了すると、HDL モデルがシミュレーション ディレクトリに作成されます。
4. 次のコマンドを入力して、シミュレータを起動します。  

```
run sim
```

## Simgen の構文

コマンド プロンプトで、MHS ファイルと必要なオプションを指定して、Simgen を実行します。

```
simgen <system_name>.mhs [options]
```

### 要件

システムが ISE ツールを実行できるよう正しく設定されていることを確認してください。詳細は、ソフトウェア パッケージに含まれる『インストール、ライセンス、およびリリース ノート』を参照してください。

## Simgen のコマンド オプション

表 3-1 に、Simgen のコマンド オプションを示します。

表 3-1 : Simgen のコマンド オプション

オプション	コマンド	説明
EDK ライブラリのディレクトリの指定	<b>-E</b> <edklib_dir>	このオプションは廃止予定です。EDK シミュレーション ライブラリのディレクトリは、 <b>-x</b> オプションから判断されます。
ヘルプの表示	<b>-h</b> 、 <b>-help</b>	コマンドの使用方法を表示して終了します。
オプション ファイルの指定	<b>-f</b> <filename>	コマンド ライン引数およびオプションの情報を、指定したファイルから読み込みます。
HDL 言語の指定	<b>-lang</b> {vhdl verilog}	HDL 言語を VHDL または Verilog に指定します。 デフォルト : vhdl
ログ ファイルの指定	<b>-log</b> <logfile.log>	ログ ファイルを指定します。 デフォルト : simgen.log
ライブラリ ディレクトリパスの指定	<b>-lp</b> <library_path>	ライブラリ ディレクトリへのパスを指定します。このオプションを複数回使用して、複数のライブラリ ディレクトリを指定できます。
シミュレーション モデルのタイプの指定	<b>-m</b> {beh str tim}	使用するシミュレーション モデルのタイプを指定します。サポートされているタイプは、ビヘイビア (beh)、構造 (str)、およびタイミング (tim) です。 デフォルト : beh
混合言語の使用	<b>-mixed</b> {yes no}	ビヘイビア ファイルで混合言語を使用できるようにするかどうかを指定します。  yes : ペリフェラルが記述された言語を使用し、混合言語を使用できるようにします。  no : 混合言語を使用不可にし、選択した言語で記述されていないペリフェラルには構造ファイルを使用します。  メモ : <b>-m beh</b> を使用している場合にのみ設定可能です。 デフォルト : yes
出力ディレクトリの指定	<b>-od</b> <output_dir>	プロジェクト ディレクトリのパスを指定します。デフォルトは現在のディレクトリです。

表 3-1 : Simgen のコマンド オプション (続き)

オプション	コマンド	説明
ターゲット デバイス/ ファミリの指定	<b>-p</b> <partname>	ターゲットとするデバイスまたはファミリを指定します。このオプションは必須です。
プロセッサの ELF ファイル の指定	<b>-pe</b> <proc_instance> <b>elf_file</b> <elf_file>	MHS ファイルで定義されているインスタンス名のプロセッサに関連付ける ELF ファイルのリストを指定します。
シミュレータ ベンダーの 指定	<b>-s</b> {mti ncs}	指定したシミュレータ (ModelSim または NC-Sim) 用のコンパイル スクリプトおよびヘルプ スクリプトを生成します。設定可能な値は、次のとおりです。  mti : ModelSim  ncs : NC-Sim
ソース ディレクトリの指定	<b>-sd</b> <source_dir>	ネットリスト ファイルを検索するソース ディレクトリを指定します。
テストベンチ テンプレート の作成	<b>-tb</b>	テストベンチ テンプレート ファイルを作成します。  テスト中のデザインを指定するには <b>-ti</b> 、テストベンチ名を指定するには <b>-tm</b> を使用します。
最上位インスタンス名の 指定	<b>-ti</b> <top_instance>	テストベンチ テンプレートを作成するよう指定した場合に、テスト中のデザインのインスタンス名を指定します。  デザインがサブモジュールである場合は、最上位インスタンス名を指定します。
最上位モジュール名の指定	<b>-tm</b> <top_module>	テストベンチ テンプレートを作成するよう指定した場合に、テストベンチ名を指定します。  デザインがサブモジュールである場合は、最上位エンティティ/モジュール名を指定します。
最上位/サブモジュールの 指定	<b>-toplevel</b> {yes no}	yes : デザインはデザイン全体を表します。 no : デザインは階層の 1 レベル (サブモジュール) を表します。 デフォルト : yes
バージョン情報の表示	<b>-v</b>	バージョン番号を表示して終了します。
ライブラリ ライブラリの ディレクトリの指定	<b>-x</b> <xlib_dir>	ライブラリ シミュレーション ライブラリ (UNISIM、SIMPRIM、XilinxCoreLib) のディレクトリへのパスを指定します。これは、Compplib の出力ディレクトリです。

## 出力ファイル

シミュレーション ファイルは、出力ディレクトリの **simulation** ディレクトリ内にある各シミュレーション モデルのサブディレクトリに保存されます。

`<output_directory>/simulation/<sim_model>`

ここで、`<sim_model>` は `behavioral`、`structural`、`timing` のいずれかです。

Simgen が正常に実行されると、**simulation** ディレクトリに次のファイルが作成されます。

- `<peripheral>_wrapper.[vhd|v]`  
各コンポーネントのモジュール シミュレーション ファイル。タイミング モデルには適用されません。
- `<system_name>.[vhd|v]`  
デザインの最上位 HDL ファイル。
- `<system_name>.sdf`  
配置配線結果から算出されたブロックおよびネット遅延を含む SDF ファイル。タイミング シミュレーションのみに使用されます。
- `xilinxsim.ini`  
ISim 用の初期化ファイル。
- `<system>.prj`  
ISim 用にコンパイルする HDL ソース ファイルおよびライブラリを指定するプロジェクト ファイル。
- `<system_name>_fuse.sh`  
シミュレーション実行ファイルを作成するヘルパ スクリプト (ISim のみ、Simgen でテスト ハーネスを作成しない場合)。
- `<system_name>_setup.[do|sh|tcl]`  
HDL ファイルをコンパイルし、コンパイルされたシミュレーション モデルをシミュレータに読み込むスクリプト ファイル。
- `<test_harness_name>.prj`  
ISim 用にコンパイルする HDL ソース ファイルおよびライブラリを指定するプロジェクト ファイル (Simgen でテスト ハーネスを作成する場合)。
- `<test_harness>_fuse.sh`  
シミュレーション実行ファイルを作成するヘルパ スクリプト (ISim のみ、Simgen でテスト ハーネスを作成する場合)。
- `<test_harness>_setup.[do|sh|tcl]`  
シミュレータを設定し、波形ウィンドウまたはリスト ウィンドウ (ModelSim のみ) に表示する信号を指定するヘルパ スクリプト。
- `<test_harness>_wave.[do|sv|tcl]`  
シミュレーション波形表示を設定するヘルパ スクリプト。
- `<test_harness>_list.do`  
シミュレーションのリスト表示を設定するヘルパ スクリプト (ModelSim のみ)。
- `<instance>_wave.[do|sv|tcl]`  
特定のインスタンスのシミュレーション波形表示を設定するヘルパ スクリプト。
- `<instance>_list.do`  
シミュレーションのリスト表示を設定するヘルパ スクリプト (ModelSim のみ)。

## メモリの初期化

デザインにシステムのメモリ バンクが含まれる場合、対応するメモリ シミュレーション モデルを初期化できます。**-pe** オプションを使用すると、プロセッサのインスタンスに関連付ける ELF ファイルのリストを指定できます。

この実行ファイルは、適切な GNU コンパイラ (gcc) またはアセンブラを使用して、該当する C ソースコードまたはアセンブリ ソースコードから生成できます。

**メモ**：構造シミュレーション モデルのメモリ初期化は、ネットリスト ファイルの階層が保持されている場合にのみサポートされます。

## VHDL

VHDL シミュレーション モデルの場合は、**Simgen** を **-pe** オプションを使用して実行し、VHDL ファイルを生成します。このファイルには、システムのコンフィギュレーションと初期値が含まれます。コマンドの例を次に示します。

```
simgen system.mhs -pe mblaze executable.elf -l vhd1 ...
```

このコマンドを実行すると、VHDL システム コンフィギュレーションが **system\_init.vhd** ファイルに生成されます。このファイルをシステムと共に使用して、メモリを初期化します。プロセッサ **mblaze** に接続された **BRAM** ブロックに、**executable.elf** ファイルのデータが含まれます。

## Verilog

Verilog シミュレーション モデルの場合は、**Simgen** を **-pe** オプションを使用して実行し、Verilog ファイルを生成します。このファイルには、メモリを初期化する **defparam** 文が含まれます。コマンドの例を次に示します。

```
simgen system.mhs -pe mblaze executable.elf -l verilog ...
```

このコマンドを実行すると、Verilog メモリ初期化ファイル **system\_init.v** が生成されます。このファイルをシステムと共に使用して、メモリを初期化します。プロセッサ **mblaze** に接続された **BRAM** ブロックに、**executable.elf** ファイルのデータが含まれます。

## テストベンチ

**Simgen** では、テストベンチ テンプレートを作成できます。**Simgen** を **-tb** オプションを使用して実行すると、最上位デザインがインスタンス化されたテストベンチが生成され、クロック信号およびリセット信号のデフォルト スティミュラスが作成されます。

クロック スティミュラスは、**MHS** ファイルで **SIGIS = CLK** と指定されているグローバル ポートから推論されます。クロックの周波数は **CLK\_FREQ** で指定し、クロックの位相は **CLK\_PHASE** を使用して 0 ~ 360 の値を指定します。

リセット スティミュラスは、**MHS** ファイルで **SIGIS = RST** と指定されているすべてのグローバルポートに対して推論されます。リセット信号の極性は **RST\_POLARITY** で指定し、リセット信号の長さは **RST\_LENGTH** で指定します。

クロックおよびリセットのパラメータの詳細は、**XPS** ヘルプを参照してください。

## VHDL テストベンチの例

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

library UNISIM;
use UNISIM.VCOMPONENTS.ALL;

entity system_tb is
end system_tb;

architecture STRUCTURE of system_tb is

    constant sys_clk_PERIOD : time := 10 ns;
    constant sys_reset_LENGTH : time := 160 ns;
    constant sys_clk_PHASE : time 2.5 ns;

    component system is
        port (
            sys_clk : in std_logic;
            sys_reset : in std_logic;
            rx : in std_logic;
            tx : out std_logic;
            leds : inout std_logic_vector(0 to 3)
        );
    end component;

    -- Internal signals

    signal leds : std_logic_vector(0 to 3);
    signal rx : std_logic;
    signal sys_clk : std_logic;
    signal sys_reset : std_logic;
    signal tx : std_logic;

begin

    dut : system
        port map (
            sys_clk => sys_clk,
            sys_reset => sys_reset,
            rx => rx,
            tx => tx,
            leds => leds
        );

    -- Clock generator for sys_clk

    process
    begin
        sys_clk <= '0';
        wait for (sys_clk_PHASE);
        loop
            wait for (sys_clk_PERIOD/2);
            sys_clk <= not sys_clk;
        end loop;
    end process;

    -- Reset Generator for sys_reset
```

```

process
begin
    sys_reset <= '0';
    wait for (sys_reset_LENGTH);
    sys_reset <= not sys_reset;
    wait;
end process;

-- START USER CODE (Do not remove this line)
-- User: Put your stimulus here. Code in this
--       section will not be overwritten.
-- END USER CODE (Do not remove this line)

end architecture STRUCTURE;

configuration system_tb_conf of system_tb is
for STRUCTURE
    for all : system
        use configuration work.system_conf;
    end for;
end for;
end system_tb_conf;

```

「BEGIN USER CODE」および「END USER CODE」の行の間に独自の VHDL コードを追加できます。シミュレーション ファイルを再作成しても、これらの行の間に追加したコードは保持されます。これ以外の位置に追加したコードは、テストベンチを再作成すると失われます。

## Verilog テストベンチの例

```

`timescale 1 ns/10 ps

`uselib lib=unisims_ver

module system_tb
(
);

    real sys_clk_PERIOD = 10;
    real sys_clk_PHASE = 2.5;
    real sys_reset_LENGTH = 160;

    // Internal signals

    reg [0:3] leds;
    reg rx;
    reg sys_clk;
    reg sys_reset;
    reg tx;

    system
    dut (
        .sys_clk ( sys_clk ),
        .sys_reset ( sys_reset ),
        .rx ( rx ),
        .tx ( tx ),
        .leds ( leds )
    );

```



```
// Data initialization

system_conf
dut_conf();

// Clock generator for sys_clk

initial
begin
    sys_clk = 1'b0;
    #(sys_clk_PHASE); forever
    #(sys_clk_PERIOD/2)
    sys_clk = ~sys_clk;
end

// Reset Generator for sys_reset

initial
begin
    sys_reset = 1'b0;
    #sys_clk_LENGTH sys_reset = ~sys_reset;
end

// START USER CODE (Do not remove this line)
// User: Put your stimulus here. Code in this
//      section will be not be overwritten.
// END USER CODE (Do not remove this line)

endmodule
```

「BEGIN USER CODE」および「END USER CODE」の行の間に独自の Verilog コードを追加できます。シミュレーション ファイルを再作成しても、これらの行の間に追加したコードは保持されます。これ以外の位置に追加したコードは、テストベンチを再作成すると失われます。

## デザインのシミュレーション

デザインをシミュレーションする際は、グローバル リセットやトライステート ネットなど、考慮すべき事項がいくつかあります。ザイリンクスでは、VHDL/Verilog デザインのシミュレーション方法に関する詳細情報を提供しています。詳細は、『合成/シミュレーション デザイン ガイド』の「デザインのシミュレーション」の章を参照してください。38 ページの「関連リソース」に、このマニュアルへのリンクがあります。

テスト ハーネス (テストベンチ) レベルで生成されるヘルパ スクリプトは、シミュレータの setup スクリプトです。setup スクリプトを実行すると、初期化関数が実行され、waveform スクリプトおよび list スクリプトを使用して波形ウィンドウおよびリスト ウィンドウ (ModelSim のみ) を作成する方法が表示されます。最上位スクリプトにより、インスタンス専用のスクリプトが呼び出されます。Simgen で作成していないテスト ハーネスがある場合は、ヘルパ スクリプトの階層パス名を変更する必要があります。

スクリプト内のコマンドは、一部をコメントにすることにより、表示する信号を定義しています。最上位の waveform または list スクリプトを編集すると、あるインスタンスの信号を表示/非表示にでき、インスタンスレベルのスクリプトを編集すると、各ポート信号を表示/非表示にできます。タイミング シミュレーションでは、最上位ポートのみが表示されます。

## 制限

外部メモリのシミュレーション モデルは生成されず、自動的にはサポートされません。外部メモリモデルはインスタンスエートしてシミュレーション テストベンチに接続し、モデルの仕様に応じて初期化する必要があります。

# Library Generator (Libgen)

---

この章では、エンベデッド ソフト プロセッサ用のライブラリおよびドライバを生成する Library Generator (Libgen) について説明し、ペリフェラルとドライバをカスタマイズする方法を示します。この章には、次のセクションが含まれています。

- 概要
- 関連リソース
- Libgen の使用法
- Libgen のコマンド オプション
- ディレクトリ構造
- 出力ファイル
- ライブラリおよびドライバの生成
- MSS ファイルのパラメータ
- ドライバ
- ライブラリ
- OS ブロック

## 概要

Libgen は、ライブラリとデバイスドライバをコンフィギュレーションするために最初にエンベデッド開発キット (EDK) で実行するツールです。Libgen には、XML ファイルまたは MSS (Microprocessor Software Specification) ファイルを入力します。XML ファイルはハードウェアシステムを定義し、MSS ファイルはペリフェラルのドライバ、標準入力および出力デバイス、割り込みハンドラルーチン、その他のソフトウェア機能を定義します。Libgen では、この情報を使用してライブラリおよびドライバをコンフィギュレーションします。

XML ファイルの生成方法は、SDK ヘルプを参照してください。MSS ファイルの詳細は、『Platform Specification Format Reference Manual』の「Microprocessor Software Specification (MSS)」の章を参照してください。52 ページの「関連リソース」に、このマニュアルへのリンクがあります。

**メモ :** EDK には、以前の MSS ファイルフォーマットを新しいフォーマットに変換するツールが含まれています。詳細は、第 6 章「バージョン管理ツール (revup)」を参照してください。

## 関連リソース

- 『Platform Specification Format Reference Manual』  
[http://japan.xilinx.com/ise/embedded/edk\\_docs.htm](http://japan.xilinx.com/ise/embedded/edk_docs.htm)
- 『OS and Libraries Document Collection』  
[http://japan.xilinx.com/ise/embedded/edk\\_docs.htm](http://japan.xilinx.com/ise/embedded/edk_docs.htm)
- 『Device Driver Programmer Guide』  
EDK インストール ディレクトリの /doc/japanese フォルダ内、  
ファイル名は xilinx\_drivers\_guide.pdf

## Libgen の使用法

Libgen を実行するには、次の構文を使用します。

```
libgen [options] <filename>.mss
```

## Libgen のコマンド オプション

表 4-1 に、Libgen のコマンド オプションを示します。

表 4-1 : Libgen のコマンド オプション

オプション	コマンド	説明
ヘルプの表示	<b>-h</b> 、 <b>-help</b>	コマンドの使用方法を表示して終了します。
バージョン情報の表示	<b>-v</b>	Libgen のバージョン番号を表示して終了します。
ログ ファイルの指定	<b>-log</b> <logfile.log>	ログ ファイルを指定します。 デフォルト : libgen.log
アーキテクチャファミリの指定	<b>-p</b> <partname>	ターゲット デバイスをアーキテクチャファミリ名またはデバイス名で指定します。-h オプションを使用すると、<partname> に指定可能な値が表示されます。
出力ディレクトリの指定	<b>-od</b> <output_dir>	出力ディレクトリを指定します。デフォルトは現在のディレクトリです。出力ファイルおよびディレクトリは、すべてこのオプションで指定したディレクトリに生成されます。入力ファイル filename.mss は、現在の作業ディレクトリのものが使用されます。このマニュアルでは、出力ディレクトリを OUTPUT_DIR、Libgen を起動するディレクトリを YOUR_PROJECT と記述する場合があります。
ソース ディレクトリの指定	<b>-sd</b> <source_dir>	入力ファイルを検索するソース ディレクトリを指定します。デフォルトは現在の作業ディレクトリです。

表 4-1 : Libgen のコマンド オプション (続き)

オプション	コマンド	説明
ユーザー ペリフェラルおよびドライバレポジトリのライブラリ検索パスの指定	<b>-lp</b> <library_path>	<p>ユーザー ペリフェラル、ドライバ、OS、およびライブラリのレポジトリを検索するライブラリパスを指定します。Libgen では、次が検索されます。</p> <ul style="list-style-type: none"> <li>library_path/sub_dir/drivers/ ディレクトリのドライバ</li> <li>library_path/sub_dir/sw_services/ ディレクトリのライブラリ</li> <li>library_path/sub_dir/bsp/ ディレクトリの OS</li> </ul>
ハードウェア仕様ファイル	<b>-hw</b> <hwspecfile.xml>	Libgen で使用するハードウェア仕様ファイル (XML) を指定します。このファイルは、完全なハードウェア システムを記述します。
MHS ファイルの指定	<b>-mhs</b> <mhsfile.mhs>	<p>Libgen の実行で使用する MHS ファイルを指定します。Libgen では、次の順で mhsfile.mhs が検索されます。</p> <ol style="list-style-type: none"> <li>現在の作業ディレクトリ (YOUR_PROJECT/)。</li> <li>-mhs オプションが使用されていない場合は、MSS ファイルと同じベース名のファイルが使用されます。</li> </ol>
ライブラリのコピー	<b>-lib</b>	このオプションを使用すると、ライブラリおよびドライバはコピーされますが、コンパイルはされません。
Libgen を実行するプロセッサ インスタンスの指定	<b>-pe</b> mblaze_0	指定したプロセッサ インスタンスに対して Libgen を実行します。

## ディレクトリ構造

ドライバ、ライブラリ、OS のディレクトリ構造を、図 4-1 および図 4-2 に示します。

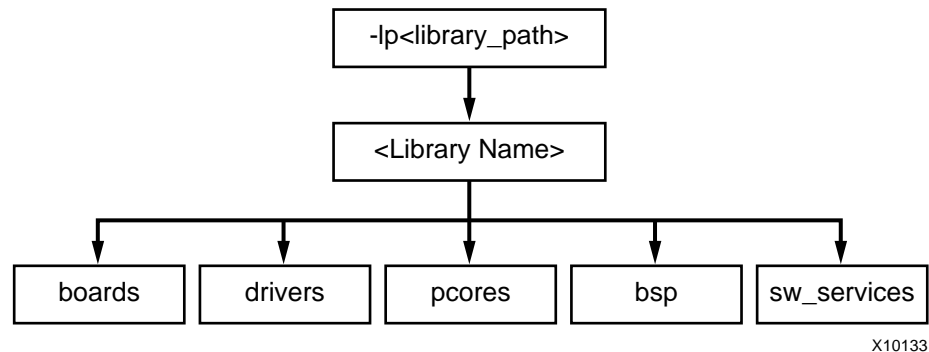


図 4-1 : ペリフェラル、ドライバ、ライブラリ、OS のディレクトリ構造

## PC のディレクトリ

PC では、ドライバ、ライブラリ、OS は次の場所にあります。

- ドライバ : %XILINX\_EDK%\sw\lib\XilinxProcessorIPLib\drivers
- ライブラリ : %XILINX\_EDK%\sw\Library\_Name\sw\_services
- OS : %XILINX\_EDK%\sw\bsp\Library\_Name

## 追加ディレクトリの指定

追加のディレクトリを指定するには、次のいずれかの方法を使用します。

- Libgen を起動した現在の作業ディレクトリを使用する。
- EDK ツールの -lp オプションを使用する。-lp オプションで指定したパスの各サブディレクトリで、ドライバ、OS、およびライブラリが検索されます。

## ディレクトリの検索順

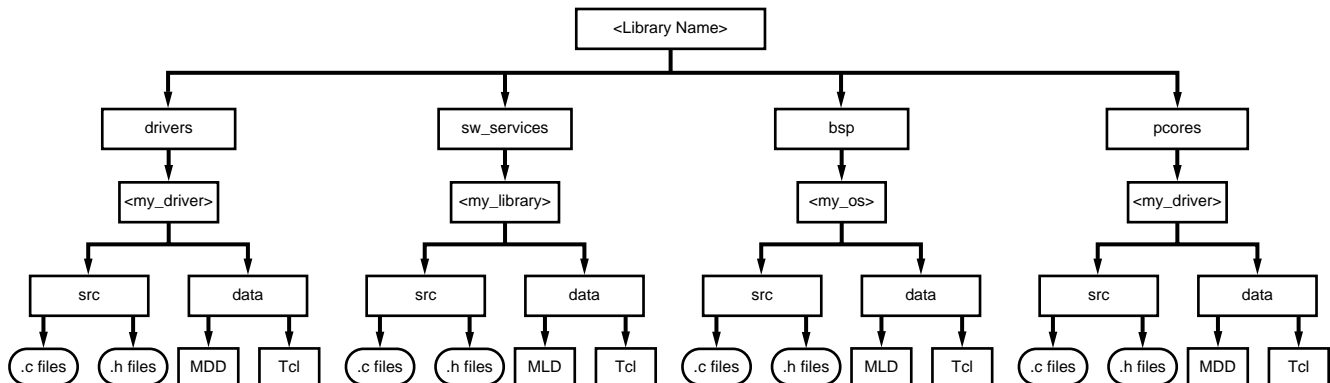
Libgen では、次の順でドライバおよびライブラリが検索されます。

1. 現在の作業ディレクトリ
  - a. ドライバ : Libgen を起動した作業ディレクトリの drivers または pcores ディレクトリ
  - b. ライブラリ : Libgen を起動した作業ディレクトリの sw\_services ディレクトリ
  - c. OS : Libgen を起動した作業ディレクトリの bsp ディレクトリ
2. -lp オプションを使用して指定したライブラリ パス ディレクトリにあるレポジトリ
  - a. ドライバ : 次のいずれかのディレクトリ。
    - library\_path\Library\_Name\drivers
    - library\_path\Library\_Name\pcores
  - b. ライブラリ : 次のディレクトリ。ここで、library\_path は -lp オプションで指定したディレクトリ、Library\_Name はそのサブディレクトリです。
    - library\_path\Library\_Name\sw\_services

- c. OS : 次のディレクトリ。ここで、*library\_path* は `-lp` オプションで指定したディレクトリ、*OS\_Name* はそのサブディレクトリです。
- *library\_path*\OS\_Name\bsp

### 3. EDK のインストール領域

- a. ドライバ : %XILINX\_EDK%\sw\Library\_Name\drivers  
 b. ライブラリ : %XILINX\_EDK%\sw\Library\_Name\sw\_services  
 c. OS : %XILINX\_EDK%\sw\Library\_Name\bsp



X10134

図 4-2 : ドライバ、OS、ライブラリのディレクトリ構造

## 出力ファイル

Libgen では、ディレクトリおよびファイルは YOUR\_PROJECT ディレクトリに生成されます。MSS ファイルの各プロセッサ インスタンスに対してそのインスタンス名のディレクトリが生成され、そのディレクトリ内に次のディレクトリおよびファイルが生成されます。

- [include](#) ディレクトリ
- [lib](#) ディレクトリ
- [libsrc](#) ディレクトリ
- [code](#) ディレクトリ

### include ディレクトリ

include ディレクトリには、ドライバに必要な C ヘッダ ファイルが保存されます。また、`xparameters.h` という include ファイルも、このディレクトリに作成されます。このファイルは、システムのペリフェラルのベース アドレス、ドライバ、OS、ライブラリ、およびユーザー プログラムに必要な `#defines`、関数のプロトタイプを定義します。各ドライバの MDD (Microprocessor Driver Definition) ファイルは、ドライバを使用するペリフェラルに対してカスタマイズする定義を指定します。詳細は、『Platform Specification Format Reference Manual』の「Microprocessor Driver Definition (MDD)」の章を参照してください。各 OS およびライブラリの MLD (Microprocessor Library Definition) ファイルは、カスタマイズする必要がある定義を指定します。詳細は、『Platform Specification Format Reference Manual』の「Microprocessor Library Definition (MLD)」の章を参照してください。

[52 ページの「関連リソース」](#) に、『Platform Specification Format Reference Manual』へのリンクがあります。

## lib ディレクトリ

lib ディレクトリには、libc.a、libm.a、および libxil.a ライブラリが含まれます。libxil ライブラリには、特定のプロセッサがアクセスするドライバ関数が含まれます。ライブラリの詳細は、『OS and Libraries Document Collection』から導入セクションを参照してください。52 ページの「[関連リソース](#)」に、このマニュアルへのリンクがあります。

## libsrc ディレクトリ

libsrc ディレクトリには、OS、ライブラリ、ドライバのコンパイルに必要な中間ファイルおよび makefile が保存されます。このディレクトリには、EDK ディレクトリおよびユーザー ドライバ、OS、ライブラリのディレクトリからコピーされたペリフェラルに固有のドライバファイル、OS 用の BSP ファイル、ライブラリ ファイルが含まれます。詳細は、57 ページの「[ドライバ](#)」、59 ページの「[OS ブロック](#)」、および 58 ページの「[ライブラリ](#)」を参照してください。

## code ディレクトリ

code ディレクトリは、EDK 実行ファイルのレポジトリです。MicroBlaze™ のボード上デバッグ用の xmdstub.elf ファイルは、このディレクトリに作成されます。

メモ：これらのディレクトリはすべて、Libgen を実行するたびに削除されます。ソース ファイル、実行ファイル、その他必要なファイルは、ユーザーが作成した場所に移動してください。

# ライブラリおよびドライバの生成

## 概要

このセクションでは、ライブラリおよびドライバの生成の概要を説明します。

MHS ファイルおよび MSS ファイルでシステムを定義します。システムの各プロセッサに対し、アドレス指定可能ペリフェラルのリストが検索され、固有のドライバおよびライブラリのリストが生成されます。Libgen は、各プロセッサに対して次の処理を実行します。

- 55 ページの「[出力ファイル](#)」に示されているディレクトリ構造を構築します。
- ドライバ、OS、ライブラリに必要なソース ファイルを、プロセッサ インスタンス固有の領域 OUTPUT\_DIR/processor\_instance\_name/libsrc にコピーします。
- プロセッサに関連付けられている各ドライバ、OS、ライブラリに対し、デザイン ルール チェック (DCR) プロシージャ (MDD または MLD ファイルのオプションで定義) を呼び出します。
- プロセッサの各ドライバ、OS、ライブラリに対し、Tcl プロシージャ generate を呼び出します (MDD または MLD ファイルに関連付けられている Tcl ファイルで定義されている場合)。これにより、プロセッサの include ディレクトリにある各ドライバ、OS、ライブラリに対して必要なコンフィギュレーション ファイルが生成されます。
- プロセッサの各ドライバ、OS、ライブラリに対し、Tcl プロシージャ post\_generate を呼び出します (MDD または MLD ファイルに関連付けられている Tcl ファイルで定義されている場合)。
- プロセッサの OS、ドライバ、ライブラリに対して make (ターゲットは include および libs) を実行します。Linux プラットフォームでは gmake ユーティリティ、NT プラットフォームでは make がコンパイルに使用されます。



- プロセッサの各ドライバ、OS、ライブラリに対し、Tcl プロシージャ `execs_generate` を呼び出します (MDD または MLD ファイルに関連付けられている Tcl ファイルで定義されている場合)。

## MDD、MLD、および Tcl

ドライバ、OS、ライブラリには、2 つのデータ ファイルが関連付けられています。

- データ定義ファイル (MDD または MLD) : ドライバ、OS、ライブラリの設定可能なパラメータを定義します。
- データ生成ファイル (Tcl) : MSS ファイルで設定されたパラメータを使用して、ドライバ、OS、ライブラリのデータを生成するスクリプト ファイル。Tcl ファイルにより、ヘッダファイルの生成、C ファイルの生成、ドライバ、OS、ライブラリに対する DRC の実行、実行ファイルの生成などが実行されます。

Tcl ファイルには、**Libgen** の実行中に呼び出されるプロシージャが含まれます。次のようなプロシージャが含まれます。

- ◆ `DRC`  
MDD または MLD ファイルで指定された DRC 名
- ◆ `generate`  
ファイルがコピーされた後に呼び出される **Libgen** で定義されたプロシージャ
- ◆ `post_generate`  
すべてのドライバ、OS、ライブラリに対して `generate` が呼び出された後に呼び出される **Libgen** で定義されたプロシージャ
- ◆ `execs_generate`  
BSP、ライブラリ、ドライバが生成された後に呼び出される **Libgen** で定義されたプロシージャ

**メモ** : ドライバ、OS、ライブラリには、データ生成ファイル (Tcl) は必要ありません。

Tcl プロシージャおよび MDD/MLD ファイルのパラメータの詳細は、『Platform Specification Format Reference Manual』の「Microprocessor Driver Definition (MDD)」および「Microprocessor Library Definition (MLD)」の章を参照してください。52 ページの「[関連リソース](#)」に、このマニュアルへのリンクがあります。

## MSS ファイルのパラメータ

MSS ファイルのフォーマットおよびパラメータの詳細は、『Platform Specification Format Reference Manual』の「Microprocessor Software Specification (MSS)」の章を参照してください。52 ページの「[関連リソース](#)」に、このマニュアルへのリンクがあります。

## ドライバ

ほとんどのペリフェラルには、ソフトウェアドライバが必要です。EDK ペリフェラルには、ドライバ、ライブラリ、BSP が付属しています。ドライバの機能については、『Device Driver Programmer Guide』を参照してください。52 ページの「[関連リソース](#)」に、このマニュアルへのリンクがあります。

MSS ファイルには、各ペリフェラルインスタンスに対するドライバブロックが含まれます。このブロックでは、ドライバを参照するためのドライバ名 (**DRIVER\_NAME** パラメータ) およびドライババージョン (**DRIVER\_VER**) が指定されます。これらのパラメータにデフォルト値はありません。

ドライバには、MDD ファイルと Tcl ファイルが関連付けられています。

- ドライバ MDD ファイルは、ドライバの設定可能なパラメータを指定するデータ定義ファイルです。
- MDD ファイルには、Tcl ファイルが関連付けられています。この Tcl ファイルは、ヘッダ ファイルの生成、C ファイルの生成、ドライバに対する DRC の実行、実行ファイルの生成などを実行するためのスクリプト ファイルです。

独自のドライバも作成できます。その場合、作成したドライバは 54 ページの図 4-1 に示す `<YOUR_PROJECT>\<driver_name>` または `<library_name>\drivers` の下の特定のディレクトリに保存する必要があります。

- ドライバ名 (ドライバ ディレクトリ名) は、`DRIVER_NAME` 属性で指定できます。
- ドライバのソース ファイルおよび `makefile` は、`<driver_name>` ディレクトリ内の `src` サブディレクトリに保存する必要があります。
- `makefile` は、`include` および `libs` をターゲットとする必要があります。
- 各ドライバの MDD ファイルおよび Tcl ファイルは、`data` サブディレクトリに保存します。

既存の EDK ドライバを参照して、ドライバの構造を理解するようにしてください。

MDD ファイルと対応する Tcl ファイルの記述方法は、『Platform Specification Format Reference Manual』の「Microprocessor Driver Definition (MDD)」の章を参照してください。52 ページの「関連リソース」に、このマニュアルへのリンクがあります。

## ライブラリ

MSS ファイルには、各ライブラリに対するライブラリ ブロックが含まれています。このライブラリ ブロックでは、ライブラリを参照するためのライブラリ名 (`LIBRARY_NAME` パラメータ) およびライブラリ バージョン (`LIBRARY_VER`) が指定されます。これらのパラメータにデフォルト値はありません。各ライブラリは、`PROCESSOR_INSTANCE` パラメータで指定されたプロセッサ インスタンスに関連付けられます。ライブラリ ディレクトリには、ライブラリの C ソース ファイル、ヘッダ ファイル、および `makefile` が含まれます。

MLD ファイルは、ライブラリの設定可能なオプションを指定します。MLD ファイルには、Tcl ファイルが関連付けられています。

独自のライブラリも作成できます。その場合、作成したライブラリは 54 ページの図 4-1 に示す `<YOUR_PROJECT>\sw_services` または `<library_name>\sw_services` の下の特定のディレクトリに保存する必要があります。

- ライブラリ名 (ライブラリ ディレクトリ名) は、`LIBRARY_NAME` 属性で指定できます。
- ライブラリのソース ファイルおよび `makefile` は、`<library_name>` ディレクトリ内の `src` サブディレクトリに保存する必要があります。
- `makefile` は、`include` および `libs` をターゲットとする必要があります。
- 各ライブラリの MLD ファイルおよび Tcl ファイルは、`data` サブディレクトリに保存します。

既存の EDK ライブラリを参照して、ライブラリの構造を理解するようにしてください。

MLD ファイルと対応する Tcl ファイルの記述方法は、『Platform Specification Format Reference Manual』の「Microprocessor Library Definition (MLD)」の章を参照してください。52 ページの「関連リソース」に、このマニュアルへのリンクがあります。

## OS ブロック

MSS ファイルには、各プロセッサ インスタンスの OS ブロックが含まれます。この OS ブロックでは、OS を参照するための OS 名 (OS\_NAME パラメータ) および OS バージョン (OS\_VER) が指定されます。これらのパラメータにデフォルト値はありません。bsp ディレクトリには、OS の C ソース ファイル、ヘッダ ファイル、および makefile が含まれます。

MLD ファイルは、OS の設定可能なオプションを指定します。MLD ファイルには、Tcl ファイルが関連付けられています。詳細は、『Platform Specification Format Reference Manual』の「Microprocessor Library Definition (MLD)」および「Microprocessor Software Specification (MSS)」の章を参照してください。52 ページの「関連リソース」に、このマニュアルへのリンクがあります。

独自の OS も作成できます。その場合、作成した OS は 54 ページの図 4-1 に示す <YOUR\_PROJECT>\bsp または <library\_name>/bsp の下の特定のディレクトリに保存する必要があります。

- OS 名 (OS ディレクトリ名) は、OS\_NAME 属性で指定します。
- OS のソース ファイルおよび makefile は、<os\_name> ディレクトリ内の src サブディレクトリに保存する必要があります。
- makefile は、include および libs をターゲットとする必要があります。
- 各 OS の MLD ファイルおよび Tcl ファイルは、data サブディレクトリに保存します。

既存の EDK OS を参照して、OS の構造を理解するようにしてください。MLD ファイルと対応する Tcl ファイルの記述方法は、『Platform Specification Format Reference Manual』の「Microprocessor Library Definition (MLD)」の章を参照してください。52 ページの「関連リソース」に、このマニュアルへのリンクがあります。



# Platform Specification Utility (PsfUtil)

---

この章では、EDK に準拠した IP コアの作成に必要な MPD (Microprocessor Peripheral Definition) ファイルを自動的に生成する Platform Specification Utility (PsfUtil) の機能と使用方法について説明します。MPD ファイルは、EDK に対応する IP ペリフェラルを作成するために必要です。このツールの機能は、Xilinx® Platform Studio™ (XPS) の Create and Import Peripheral (CIP) Wizard で MPD ファイルを作成する際に使用されます。

この章には、次のセクションが含まれています。

- [PsfUtil のコマンド オプション](#)
- [MPD ファイル作成プロセスの概要](#)
- [MPD ファイル自動生成の使用法](#)
- [PsfUtil での DRC チェック](#)
- [HDL ペリフェラルの定義](#)

## PsfUtil のコマンド オプション

表 5-1 : PsfUtil のコマンド オプション

オプション	コマンド	説明
1 つの IP の MHS テンプレート	<b>-deploy_core</b> <corename> <coreversion>	1 つのペリフェラルをインスタンス化して MHS テンプレートを生成します。 サブオプション :  -lp <library path> : IP ライブラリ検索パスを追加します。 -o <outfile> : 出力ファイル名を指定します。デフォルトは stdout です。
ヘルプの表示	<b>-h, -help</b>	コマンドの使用方法を表示します。
HDL ファイルから MPD を生成	<b>-hdl2mpd</b> <hdlfile>	VHDL/Verilog の src/prj ファイルから、MPD ファイルを生成します。 サブオプション :  -lang {ver vhd1} : 言語を指定します。 -top <design> : 最上位エンティティ / モジュール名を指定します。 -bus {opb <sup>(2)</sup>  plb <sup>(2)</sup>  plbv46 dcr lmb fsl m s ms mb <sup>(1)</sup> [<busif_name>]} : ペリフェラルのバスインターフェイスを指定します。 -p2pbus <busif_name> <bus_std> {target initiator} : ペリフェラルのポイントツーポイント接続を指定します。 -o <outfile> : 出力ファイル名を指定します。デフォルトは stdout です。
PAO ファイルから MPD を生成	<b>-pao2mpd</b> <paofile>	PAO (Peripheral Analyze Order) ファイルから MPD ファイルを生成します。 サブオプション :  -lang {ver vhd1} : 言語を指定します。 -top <design> : 最上位エンティティ / モジュール名を指定します。 -bus {opb plb plbv46 dcr lmb fsl m s ms mb <sup>(1)</sup> [<busif_name>]} -p2pbus <busif_name> <bus_std> {target initiator} : ペリフェラルのポイントツーポイント接続を指定します。 -o <outfile> : 出力ファイル名を指定します。デフォルトは stdout です。
バージョン番号の表示	<b>-v</b>	バージョン番号を表示します。

### メモ :

1. バス タイプ mb (バースト トランザクションを生成するマスタ) は、バス規格 plbv46 でしか使用できません。
2. このリリース以降、廃止される予定です。

## MPD ファイル作成プロセスの概要

PsfUtil を使用すると、ペリフェラルの HDL ソースから MPD ファイルを自動的に作成できます。ペリフェラルを作成して EDK で使用するには、次の手順に従います。

1. IP を VHDL または Verilog で記述します。この際、バス信号、クロック信号、リセット信号、および割り込み信号の命名規則に従う必要があります。命名規則は、[66 ページの「HDL ペリフェラルの定義」](#)を参照してください。

メモ：この命名規則に従うことにより、正しい MPD ファイルが生成されます。

2. IP のインプリメントに必要な HDL ソース ファイルをリストした XST (Xilinx Synthesis Technology) プロジェクト ファイルまたは PAO ファイルを作成します。
3. XST プロジェクト ファイルまたは PAO ファイルを入力とし、追加オプションを使用して PsfUtil を実行します。

異なるオプションを使用した PsfUtil の実行については、「[MPD ファイル自動生成の使用法](#)」を参照してください。

## MPD ファイル自動生成の使用法

PsfUtil は、ペリフェラルのバス規格やバス インターフェイスのタイプ、バス インターフェイスの数によって、さまざまな実行方法があります。使用可能なバス規格は、次のとおりです。

- OPB<sup>(1)</sup> (オンチップ ペリフェラル バス) SLAVE
- OPB<sup>(1)</sup> MASTER
- OPB<sup>(1)</sup> MASTER\_SLAVE
- PLB<sup>(1)</sup> (プロセッサ ローカル バス) SLAVE
- PLB<sup>(1)</sup> MASTER
- PLB<sup>(1)</sup> MASTER\_SLAVE
- PLBV46 (プロセッサ ローカル バス バージョン 4.6) SLAVE
- PLBV46 MASTER
- DCR (デザイン コントロール レジスタ) SLAVE
- LMB (ローカル メモリ バス) SLAVE
- FSL (高速シンプレックス リンク) SLAVE
- FSL MASTER
- POINT TO POINT BUS (特殊なケース)

### バス インターフェイスを 1 つ使用するペリフェラル

プロセッサ ペリフェラルのほとんどは、バス インターフェイスを 1 つ使用します。これは、PsfUtil の最も単純な使用法です。このようなペリフェラルの場合、ソース コードに属性を追加せずに MPD を完成させることができます。

---

1. このリリース以降、廃止される予定です。

## 信号の命名規則

信号には、66 ページの「HDL ペリフェラルの定義」に示されている命名規則に従って名前を付ける必要があります。バス インターフェイスが 1 つしかない場合は、バス信号にバス識別子を指定する必要はありません。

## PsfUtil の実行

PsfUtil を実行するには、次のコマンドを入力します。

```
psfutil -hdl2mpd <hdlfile> -lang {vhdl|ver} -top <top_entity>  
-bus <busstd> <bustype> -o <mpdfile>
```

たとえば、UART という PLB スレーブ ペリフェラルの MPD ファイルを作成するには、次のコマンドを使用します。

```
psfutil -hdl2mpd uart.prj -lang vhdl -top uart -bus plb s -o uart.mpd
```

## 複数のバス インターフェイスを使用するペリフェラル

複数のバス インターフェイスを使用するペリフェラルもあります。インターフェイスには、排他的バス インターフェイス、非排他的バス インターフェイス、またはその組み合わせを使用できます。1 つのペリフェラルに同時に接続可能なバス インターフェイスは、排他的です。たとえば、OPB スレーブ バス インターフェイスと DCR スレーブ バス インターフェイスはペリフェラルに同時に接続できるため、排他的バス インターフェイスです。

**メモ：**ペリフェラルに複数の排他的バス インターフェイスが接続されている場合、そのペリフェラルのポートは 1 つの排他的バス インターフェイスにしか接続できません。

非排他的バス インターフェイスは、同時には接続できません。

**メモ：**ペリフェラルに複数の非排他的バス インターフェイスが接続されている場合、ポートを複数の非排他的バス インターフェイスに接続できます。また、これらのインターフェイスのバス規格は同じになります。

## 非排他的および排他的バス インターフェイス

### 信号の命名規則

信号には、66 ページの「HDL ペリフェラルの定義」に示されている命名規則に従って名前を付ける必要があります。

- 非排他的バス インターフェイスでは、バス識別子を指定する必要はありません。
- 排他的バス インターフェイスでは、1 つのペリフェラルに同じバス規格およびタイプのバス インターフェイスが複数ある場合は、バス識別子を指定する必要があります。

### コマンド ラインでバスを指定して PsfUtil を実行する場合

バス信号にバス識別子が付いていない場合は、バスをコマンド ラインで指定できます。次のコマンドを使用します。

```
psfutil -hdl2mpd <hdlfile> -lang {vhdl|ver} -top <top_entity>  
[-bus <busstd> <bustype>] -o <mpdfile>
```



### 非排他的および排他的バス インターフェイスのコマンド ライン例

非排他的バス インターフェイスで PLB スレーブ インターフェイスと PLB マスタ/スレーブ インターフェイスを使用した gemac というペリフェラルの MPD ファイルを作成する場合は、次のコマンドを使用します。

```
psfutil -hdl2mpd gemac.prj -lang vhdl -top gemac -bus plb s -bus plb ms  
-o gemac.mpd
```

排他的バス インターフェイスで PLB スレーブ インターフェイスと DCR スレーブ インターフェイスを使用したペリフェラルの MPD ファイルを作成するには、次のコマンドを使用します。

```
psfutil -hdl2mpd mem.prj -lang vhdl -top mem -bus plb s -bus dcr s  
-o mem.prj
```

## ポイントツーポイント接続を含むペリフェラル

マルチチャネル メモリ コントローラなどのペリフェラルには、ポイントツーポイント接続が含まれるものがあります (BUS\_STD = XIL\_MEMORY\_CHANNEL、BUS\_TYPE = TARGET)。

### 信号の命名規則

ポイントツーポイント接続に関連するすべての信号には、同じバス インターフェイス名接頭辞 (MCH0\_\* など) を付ける必要があります。

### コマンド ラインでポイントツーポイント接続を指定して PsfUtil を実行する場合

ポイントツーポイント接続は、バス インターフェイス名をバス信号名の接頭辞として使用することにより、コマンド ラインで指定できます。次のコマンドを使用します。

```
psfutil -hdl2mpd <hdlfile> -lang {vhdl|ver} -top <top_entity>  
-p2pbus <busif_name> <bus_std> {target|initiator} -o <mpdfile>
```

たとえば、MCH0 接続を含むペリフェラルの MPD ファイルを作成するには、次のコマンドを使用します。

```
psfutil -hdl2mpd mch_mem.prj -lang vhdl -top mch_mem -p2pbus MCH0  
XIL_MEMORY_CHANNEL TARGET -o mch_mem.mpd
```

## PsfUtil での DRC チェック

HDL ソースから正しい MPD ファイルを生成するため、PsfUtil により DRC チェックが実行されます。次に、DRC チェックを実行される順に示します。

### HDL ソース エラー

HDL ソース ファイルにエラーがあると、そのエラーがレポートされます。

### バス インターフェイス チェック

指定された各バス インターフェイスに対し、接続されているペリフェラルに応じて、次のチェックが実行されます。

- バス信号がすべて存在するかどうかを確認し、レポートします。
- バス信号が重複していないかを確認し、レポートします。

すべてのバス インターフェイスに対するチェックが完了すると、MPD ファイルが生成されます。

## HDL ペリフェラルの定義

IP ペリフェラルの最上位 HDL ソース ファイルは、デザインのインターフェイスを定義します。次のような特徴があります。

- バス インターフェイスのポートおよびデフォルトの接続を記述します。
- パラメータ (ジェネリック) およびデフォルト値を記述します。
- HDL ソースのパラメータは、MHS の同等の設定に書き換えられます。

各ペリフェラルのマニュアルに、ソース ファイルのオプションに関する情報が記載されています。

### バス インターフェイスの命名規則

バス インターフェイスは、関連するインターフェイス信号のグループです。自動生成を正しく機能させるためには、バス インターフェイスに関連する信号およびパラメータの命名規則に従う必要があります。命名規則に従うと、表 5-2 に示すインターフェイス タイプが自動的に認識され、MPD ファイルにバス インターフェイスのラベルが含まれます。

表 5-2 : 認識されるバス インターフェイス

説明	MPD のバス ラベル
スレーブ DCR インターフェイス	SDCR
スレーブ LMB インターフェイス	SLMB
マスタ OPB <sup>(a)</sup> インターフェイス	MOPB
マスタ/スレーブ OPB <sup>(a)</sup> インターフェイス	MSOPB
スレーブ OPB <sup>(a)</sup> インターフェイス	SOPB
マスタ PLB <sup>(a)</sup> インターフェイス	MPLB
マスタ/スレーブ PLB <sup>(a)</sup> インターフェイス	MSPLB
スレーブ PLB <sup>(a)</sup> インターフェイス	SPLB
マスタ PLBV46 インターフェイス	MPLB
スレーブ PLBV46 インターフェイス	SPLB
マスタ FSL インターフェイス	MFSL
スレーブ FSL インターフェイス	SFSL

a. このリリース以降、廃止される予定です。

同じタイプのバス インターフェイスが複数使用されているコンポーネントでは、これらのバス インターフェイスがグループ化されるように、命名規則に従って名前を付ける必要があります。

### VHDL ジェネリックの命名規則

ペリフェラルに同じバス インターフェイスが複数含まれる場合は、バス識別子を使用する必要があります。バス識別子は、関連付けられている信号およびジェネリックすべてに付けます。

ジェネリック名は、VHDL に準拠する必要があります。次に、IP ペリフェラルの追加の命名規則を示します。

- ジェネリック名は **C\_** で開始します。
- 1 つのペリフェラルに同じバス インターフェイス タイプのインスタンスが複数ある場合、バス 識別子 **<BI>** を信号名に使用する必要があります。
- ポートに関連付けられている信号にバス識別子が使用されている場合は、そのポートに関連付けられたジェネリックにも **<BI>** を使用できます。
- **<BI>** が使用されていない場合は、バス パラメータに関連付けられているジェネリックはグローバルであると判断されます。たとえば、**C\_DOPB\_DWIDTH** には **D** というバス識別子が付いているので、バス識別子が **D** のバス信号に関連付けられます。**C\_OPB\_DWIDTH** の場合は、ポート信号のバス識別子にかかわらず、すべての **OPB** バスに関連付けられます。

**メモ：PLBV46** バス インターフェイスの場合は、**<BI>** はバス タグ (バス インターフェイス名) として使用されます。たとえば、**C\_SPLB0\_DWIDTH** にはバス識別子 (タグ) **SPLB0** が含まれているので、接頭辞としてバス識別子 **SPLB0** が付いているバス信号に関連付けられます。

- バス インターフェイスが 1 つだけのペリフェラルでは、信号名およびジェネリック名にバス識別子を使用する必要はありません。
- ベース アドレスを指定するジェネリックには最後に **\_BASEADDR** を付け、ハイ アドレスを指定するジェネリックには **\_HIGHADDR** を付けます。さらに、これらのアドレスをバスに関連付けるには、前述のパラメータの命名規則にも従う必要があります。
- 複数のタイプのバス インターフェイスを使用するペリフェラルでは、パラメータ名でバス規格タイプを示す必要があります。たとえば、**PLB** バスのアドレスのパラメータは **C\_PLB\_BASEADDR** および **C\_PLB\_HIGHADDR** とする必要があります。

Platform Generator (Platgen) は、一部の予約済みジェネリックに対して自動的に値を割り当てます。これを正しく機能させるためには、これらのパラメータにバス タグを設定する必要があります。PsfUtil でこの情報を自動的に推論させるには、予約済みジェネリックに対しても命名規則をすべて適用する必要があります。この機能により、ペリフェラルでプラットフォームの生成された情報が必要な場合に、エラーを回避できます。表 5-3 に、予約済みのジェネリック名を示します。

表 5-3：自動的に値が割り当てられる予約済みジェネリック

パラメータ	説明
<b>C_FAMILY</b>	FPGA デバイス ファミリ
<b>C_INSTANCE</b>	コンポーネントのインスタンス名
<b>C_&lt;BI&gt;OPB_NUM_MASTERS</b>	OPB マスタの数
<b>C_&lt;BI&gt;OPB_NUM_SLAVES</b>	OPB スレーブの数
<b>C_&lt;BI&gt;DCR_AWIDTH</b>	DCR アドレス幅
<b>C_&lt;BI&gt;DCR_DWIDTH</b>	DCR データ幅
<b>C_&lt;BI&gt;DCR_NUM_SLAVES</b>	DCR スレーブの数
<b>C_&lt;BI&gt;FSL_DWIDTH</b>	FSL データ幅
<b>C_&lt;BI&gt;LMB_AWIDTH</b>	LMB アドレス幅
<b>C_&lt;BI&gt;LMB_DWIDTH</b>	LMB データ幅
<b>C_&lt;BI&gt;LMB_NUM_SLAVES</b>	LMB スレーブの数
<b>C_&lt;BI&gt;OPB_AWIDTH</b>	OPB アドレス幅
<b>C_&lt;BI&gt;OPB_DWIDTH</b>	OPB データ幅

表 5-3 : 自動的に値が割り当てられる予約済みジェネリック (続き)

パラメータ	説明
C_<BI>PLB_AWIDTH	PLB アドレス幅
C_<BI>PLB_DWIDTH	PLB データ幅
C_<BI>PLB_MID_WIDTH	PLB マスタ ID の幅
C_<BI>PLB_NUM_MASTERS	PLB マスタの数
C_<BI>PLB_NUM_SLAVES	PLB スレーブの数

## 予約済みパラメータ

表 5-4 のパラメータの値は、Platgen により自動的に割り当てられます。

表 5-4 : 予約済みパラメータ

パラメータ	説明
C_BUS_CONFIG	MicroBlaze プロセッサのバス コンフィギュレーション
C_FAMILY	FPGA デバイス ファミリ
C_INSTANCE	コンポーネントのインスタンス名
C_DCR_AWIDTH	DCR のアドレス幅
C_DCR_DWIDTH	DCR のデータ幅
C_DCR_NUM_SLAVES	バス上の DCR スレーブの数
C_LMB_AWIDTH	LMB のアドレス幅
C_LMB_DWIDTH	LMB のデータ幅
C_LMB_NUM_SLAVES	バス上の LMB スレーブの数
C_OPB_AWIDTH	OPB のアドレス幅
C_OPB_DWIDTH	OPB のデータ幅
C_OPB_NUM_MASTERS	バス上の OPB <sup>(a)</sup> マスタの数
C_OPB_NUM_SLAVES	バス上の OPB <sup>(a)</sup> スレーブの数
C_PLB_AWIDTH	PLB <sup>(a)</sup> のアドレス幅
C_PLB_DWIDTH	PLB <sup>(a)</sup> のデータ幅
C_PLB_MID_WIDTH	PLB <sup>(a)</sup> マスタ ID の幅。log2(S) に設定されています。
C_PLB_NUM_MASTERS	バス上の PLB <sup>(a)</sup> マスタの数
C_PLB_NUM_SLAVES	バス上の PLB <sup>(a)</sup> スレーブの数

a. このリリース以降、廃止される予定です。

## バス インターフェイス信号の命名規則

このセクションでは、バス インターフェイス信号の命名規則を示します。この命名規則は、1 つのコンポーネントに複数のバス インターフェイスおよび複数のバス インターフェイス ポートが使用されているエンベデッド プロセッサ システムにも適用できます。デザイン内のペリフェラルに複数のバス インターフェイス ポートがある場合は、バス識別子の使用方法を理解しておく必要があります。前述のとおり、ペリフェラルに同じバス インターフェイスが複数含まれる場合は、バス識別子を使用する必要があります。バス識別子は、関連付けられている信号およびジェネリックすべてに付けます。

信号名は、HDL に準拠する必要があります。次に、追加の命名規則を示します。

- 信号名の最初の文字は、大文字のアルファベットにします。
- 各信号の識別子の既定部分は、この後の該当するセクションに示されているとおりにします。各セクションで、バス インターフェイスのタイプに必要な信号を示します。
- 1 つのペリフェラルに同じバス インターフェイス タイプのインスタンスが複数ある場合、信号名にバス識別子 **<BI>** を含める必要があります。バス識別子は、1 文字でも、最後にアンダースコア ( **\_** ) を付けた説明的な文字列でもかまいません。次の場合には、ポートの信号名に **<BI>** を含める必要があります。
  - ◆ ペリフェラルにスレーブ **PLB<sup>(1)</sup>** ポートが複数ある。
  - ◆ ペリフェラルにマスタ **PLB<sup>(1)</sup>** ポートが複数ある。
  - ◆ ペリフェラルにスレーブ **LMB** ポートが複数ある。
  - ◆ ペリフェラルにスレーブ **DCR** ポートが複数ある。
  - ◆ ペリフェラルにマスタ **DCR** ポートが複数ある。
  - ◆ ペリフェラルにスレーブ **FSL** ポートが複数ある。
  - ◆ ペリフェラルにマスタ **FSL** ポートが複数ある。
  - ◆ ペリフェラルにスレーブ **PLBV46** ポートが複数ある。
  - ◆ ペリフェラルにマスタ **PLBV46** ポートが複数ある。
  - ◆ ペリフェラルにマスタ、スレーブ、またはマスタ/スレーブの **OPB<sup>(1)</sup>** ポートが複数ある。
  - ◆ ペリフェラルにいずれかのタイプのポートが複数あり、**<Mn>** または **<Sln>** の選択によって信号名が不確定になる。たとえば、ペリフェラルにマスタ **OPB<sup>(1)</sup>** ポートとマスタ **PLB<sup>(1)</sup>** ポートがある場合、両方のポートに同じ **<Mn>** を使用するとアドレス バス信号名が不確定になるので、ポートを区別するために **<BI>** が必要です。

バス インターフェイスが 1 つだけのペリフェラルでは、信号名にバス識別子を使用する必要はありません。

---

1. このリリース以降、廃止される予定です。

## グローバル ポート

ペリフェラルのグローバル ポート (クロック、リセットなど) の名前は、標準化されています。その他のグローバル ポート (割り込み信号など) には、任意の名前を使用できます。

### LMB - クロックおよびリセット

```
LMB_Clk
LMB_Rst
```

### OPB<sup>(1)</sup> - クロックおよびリセット

```
OPB_Clk
OPB_Rst
```

### PLB<sup>(1)</sup> - クロックおよびリセット

```
PLB_Clk
PLB_Rst
```

### PLBV46 スレーブ - クロックおよびリセット

```
SPLB_Clk
SPLB_Rst
```

### PLBV46 マスタ - クロックおよびリセット

```
MPLB_Clk
MPLB_Rst
```

## スレーブ DCR ポート

スレーブ DCR ポートは、表 5-5 に示す命名規則に従う必要があります。

表 5-5: スレーブ DCR ポートの命名規則

<Sln>	スレーブ出力を表す意味のある名前または短縮形を使用します。スレーブ出力とバス出力との混乱を避けるため、DCR (大文字、小文字、または大文字/小文字混合) という文字列は含めないでください。
<nDCR>	スレーブ入力を表す意味のある名前または短縮形を使用します。<nDCR> の最後の 3 文字は、DCR (大文字、小文字、または大文字/小文字混合) にする必要があります。
<BI>	バス識別子です。スレーブ DCR ポートが 1 つのみのペリフェラルでは必要ありませんが、複数のスレーブ DCR ポートのあるペリフェラルでは必須です。<BI> には、DCR (大文字、小文字、または大文字/小文字混合) という文字列を含めないでください。複数のスレーブ DCR ポートがあるペリフェラルでは、各バス インターフェイスに固有の <BI> を使用する必要があります。

メモ: <BI> を使用している場合は、<Sln> を使用する必要はありません。

1. このリリース以降、廃止される予定です。

## DCR スレーブ出力

DCR への接続には、すべてのスレーブに次の出力が必要です。

```
<BI><Sln>_dcrDBus : out std_logic_vector(0 to C_<BI>DCR_DWIDTH-1);
<BI><Sln>_dcrAck  : out std_logic;
```

例:

```
Uart_dcrAck      : out std_logic;
Intc_dcrAck      : out std_logic;
Memcon_dcrAck    : out std_logic;
Bus1_timer_dcrAck : out std_logic;
Bus1_timer_dcrDBus : out std_logic_vector(0 to C_<BI>DCR_DWIDTH-1);
Bus2_timer_dcrAck : out std_logic;
Bus2_timer_dcrDBus : out std_logic_vector(0 to C_<BI>DCR_DWIDTH-1);
```

## DCR スレーブ入力

DCR への接続には、すべてのスレーブに次の入力が必要です。

```
<BI><nDCR>_ABus      : in  std_logic_vector(0 to C_<BI>DCR_AWIDTH-1);
<BI><nDCR>_DBus      : in  std_logic_vector(0 to C_<BI>DCR_DWIDTH-1);
<BI><nDCR>_Read      : in  std_logic;
<BI><nDCR>_Write     : in  std_logic;
```

例:

```
DCR_DBus          : in  std_logic_vector(0 to C_<BI>DCR_DWIDTH-1);
Bus1_DCR_DBus     : in  std_logic_vector(0 to C_<BI>DCR_DWIDTH-1);
```

## スレーブ FSL ポート

スレーブ FSL ポートは、表 5-6 に示す命名規則に従う必要があります。

表 5-6: スレーブ FSL ポートの命名規則

<nFSL> または <nFSL_S>	スレーブの入力/出力を表す意味のある名前または短縮形を使用します。 <nFSL_S> の最後の 5 文字は、FSL_S (大文字、小文字、または大文字/小文字混合) にする必要があります。
<BI>	バス識別子です。スレーブ FSL ポートが 1 つのみのペリフェラルでは必要ありませんが、複数のスレーブ FSL ポートのあるペリフェラルでは必須です。 <BI> には、FSL_S (大文字、小文字、または大文字/小文字混合) という文字列を含めないでください。複数のスレーブ FSL ポートがあるペリフェラルでは、各バス インターフェイスに固有の <BI> を使用する必要があります。

## FSL スレーブ出力

FSL への接続には、スレーブに次の出力が必要です。

```
<BI><nFSL_S>_Data      : out std_logic_vector(0 to C_<BI>FSL_DWIDTH-1);
<BI><nFSL_S>_Control  : out std_logic;
<BI><nFSL_S>_Exists   : out std_logic;
```

例 :

```
FSL_S_Control      : out std_logic;
Memcon_FSL_S_Control : out std_logic;
Bus1_timer_FSL_S_Control: out std_logic;
Bus1_timer_FSL_S_Data: out std_logic_vector(0 to C_<BI>FSL_DWIDTH-1);
Bus2_timer_FSL_S_Control: out std_logic;
Bus2_timer_FSL_S_Data: out std_logic_vector(0 to C_<BI>FSL_DWIDTH-1);
```

## FSL スレーブ入力

FSL への接続には、スレーブに次の入力が必要です。

```
<BI><nFSL>_Clk      : in  std_logic;
<BI><nFSL>_Rst       : in  std_logic;
<BI><nFSL_S>_Clk    : in  std_logic;
<BI><nFSL_S>_Read   : in  std_logic;
```

例 :

```
FSL_S_Read : in  std_logic;
Bus1_FSL_S_Read : in  std_logic;
```

## マスタ FSL ポート

マスタ FSL ポートは、表 5-7 に示す命名規則に従う必要があります。

表 5-7 : マスタ FSL ポートの命名規則

<nFSL> または <nFSL_M>	マスタの入力/出力を表す意味のある名前または短縮形を使用します。 <nFSL_M> の最後の 5 文字は、FSL_M (大文字、小文字、または大文字/ 小文字混合) にする必要があります。
<BI>	バス識別子です。マスタ FSL ポートが 1 つのみのペリフェラルでは必要ありませんが、複数のマスタ FSL ポートのあるペリフェラルでは必須です。<BI> には、FSL_M (大文字、小文字、または大文字/小文字混合) という文字列を含めないでください。複数のマスタ FSL ポートがあるペリフェラルでは、各バス インターフェイスに固有の <BI> を使用する必要があります。

## FSL マスタ出力

FSL への接続には、マスタに次の出力が必要です。

```
<BI><nFSL_M>_Full : out std_logic;
```

例 :

```
FSL_M_Full      :out std_logic;
Memcon_FSL_M_Full : out std_logic;
```

## FSL マスタ入力

FSL への接続には、スレーブに次の入力が必要です。

```
<BI><nFSL>_Clk      : in  std_logic;
<BI><nFSL>_Rst       : in  std_logic;
<BI><nFSL_M>_Clk    : in  std_logic;
<BI><nFSL_M>_Data   : in  std_logic_vector(0 to C_<BI>FSL_DWIDTH-1);
<BI><nFSL_M>_Control : in  std_logic;
<BI><nFSL_M>_Write  : in  std_logic;
```



例：

```
FSL_M_Write          : in  std_logic;
Bus1_FSL_M_Write     : in  std_logic;
Bus1_timer_FSL_M_Control: out std_logic;
Bus1_timer_FSL_M_Data: out std_logic_vector(0 to C_<BI>FSL_DWIDTH-1);
Bus2_timer_FSL_M_Control: out std_logic;
Bus2_timer_FSL_M_Data: out std_logic_vector(0 to C_<BI>FSL_DWIDTH-1);
```

## スレーブ LMB ポート

スレーブ LMB ポートは、次の表に示す命名規則に従う必要があります。

表 5-8 : スレーブ LMB ポートの命名規則

<Sln>	スレーブ出力を表す意味のある名前または短縮形を使用します。スレーブ出力とバス出力との混乱を避けるため、 <b>LMB</b> (大文字、小文字、または大文字/小文字混合) という文字列は含めないでください。
<nLMB>	スレーブ入力を表す意味のある名前または短縮形を使用します。<nLMB> の最後の 3 文字は、 <b>LMB</b> (大文字、小文字、または大文字/小文字混合) にする必要があります。
<BI>	バス識別子です。スレーブ <b>LMB</b> ポートが 1 つのみのペリフェラルでは必要ありませんが、複数のスレーブ <b>LMB</b> ポートのあるペリフェラルでは必須です。<BI> には、 <b>LMB</b> (大文字、小文字、または大文字/小文字混合) という文字列を含めないでください。複数のスレーブ <b>LMB</b> ポートがあるペリフェラルでは、各バス インターフェイスに固有の <BI> を使用する必要があります。

メモ：<BI> を使用している場合は、<Sln> を使用する必要はありません。

## LMB スレーブ出力

LMB への接続には、スレーブに次の出力が必要です。

```
<BI><Sln>_DBus      : out std_logic_vector(0 to C_<BI>LMB_DWIDTH-1);
<BI><Sln>_Ready     : out std_logic;
```

例：

```
D_Ready : out std_logic;
I_Ready : out std_logic;
```

## LMB スレーブ入力

LMB への接続には、スレーブに次の入力が必要です。

```
<BI><nLMB>_ABus      : in  std_logic_vector(0 to C_<BI>LMB_AWIDTH-1);
<BI><nLMB>_AddrStrobe: in  std_logic;
<BI><nLMB>_BE        : in  std_logic_vector(0 to C_<BI>LMB_DWIDTH/8-1);
<BI><nLMB>_Clk       : in  std_logic;
<BI><nLMB>_ReadStrobe: in  std_logic;
<BI><nLMB>_Rst       : in  std_logic;
<BI><nLMB>_WriteDBus : in  std_logic_vector(0 to C_<BI>LMB_DWIDTH-1);
<BI><nLMB>_WriteStrobe: in  std_logic;
```

例：

```
LMB_ABUS : in  std_logic_vector(0 to C_LMB_AWIDTH-1);
DLMB_ABUS : in  std_logic_vector(0 to C_DLMB_AWIDTH-1);
```

## マスタ OPB<sup>(1)</sup> ポート

次に示す命名規則は、スレーブ OPB ポートからは独立したマスタ OPB ポートに適用されます。マスタ OPB ポートは、表 5-9 に示す命名規則に従う必要があります。

表 5-9 : マスタ OPB ポートの命名規則

<Mn>	マスタ出力を表す意味のある名前または短縮形を使用します。マスタ出力とバス出力との混乱を避けるため、OPB (大文字、小文字、または大文字/小文字混合) という文字列は含めないでください。
<nOPB>	マスタ入力を表す意味のある名前または短縮形を使用します。<nOPB> の最後の 3 文字は、OPB (大文字、小文字、または大文字/小文字混合) にする必要があります。
<BI>	バス識別子です。OPB ポート (タイプは問わない) が 1 つのみのペリフェラルでは必要ありませんが、複数の OPB ポート (異なるタイプの場合も含む) のあるペリフェラルでは必須です。<BI> には、OPB (大文字、小文字、または大文字/小文字混合) という文字列を含めないでください。複数の OPB ポートがあるペリフェラルでは、各バス インターフェイスに固有の <BI> を使用する必要があります。

メモ : <BI> を使用している場合は、<Mn> を使用する必要はありません。

## OPB マスタ出力

OPB への接続には、マスタに次の出力が必要です。

```

<BI><Mn>_ABus      : out std_logic_vector(0 to C_<BI>OPB_AWIDTH-1);
<BI><Mn>_BE        : out std_logic_vector(0 to C_<BI>OPB_DWIDTH/8-1);
<BI><Mn>_busLock    : out std_logic;
<BI><Mn>_DBus      : out std_logic_vector(0 to C_<BI>OPB_DWIDTH-1);
<BI><Mn>_request    : out std_logic;
<BI><Mn>_RNW       : out std_logic;
<BI><Mn>_select     : out std_logic;
<BI><Mn>_seqAddr    : out std_logic;

```

例 :

```

IM_request      : out std_logic;
Bridge_request  : out std_logic;
O20b_request    : out std_logic;

```

## OPB マスタ入力

OPB への接続には、すべてのマスタに次の入力が必要です。

```

<BI><nOPB>_Clk      : in  std_logic;
<BI><nOPB>_DBus     : in  std_logic_vector(0 to C_<BI>OPB_DWIDTH-1);
<BI><nOPB>_errAck   : in  std_logic;
<BI><nOPB>_MGrant   : in  std_logic;
<BI><nOPB>_retry    : in  std_logic;
<BI><nOPB>_Rst      : in  std_logic;
<BI><nOPB>_timeout  : in  std_logic;
<BI><nOPB>_xferAck  : in  std_logic;

```

1. このリリース以降、廃止される予定です。

例：

```
IOPB_DBus      : in  std_logic_vector(0 to C_IOPB_DWIDTH-1);
OPB_DBus       : in  std_logic_vector(0 to C_OPB_DWIDTH-1);
Bus1_OPB_DBus  : in  std_logic_vector(0 to C_Bus1_OPB_DWIDTH-1);
```

## スレーブ OPB<sup>(1)</sup> ポート

次に示す命名規則は、マスタ OPB ポートからは独立したスレーブ OPB ポートに適用されます。マスタとスレーブを組み合わせたバス インターフェイスを使用するペリフェラルの場合は、[76 ページ](#)の「マスタ/スレーブ OPB ポート」を参照してください。

スレーブ OPB ポートは、[表 5-10](#) に示す命名規則に従う必要があります。

表 5-10：スレーブ OPB ポートの命名規則

<Sln>	スレーブ出力を表す意味のある名前または短縮形を使用します。スレーブ出力とバス出力との混乱を避けるため、OPB (大文字、小文字、または大文字/小文字混合) という文字列は含めないでください。
<nOPB>	スレーブ入力を表す意味のある名前または短縮形を使用します。<nOPB> の最後の 3 文字は、OPB (大文字、小文字、または大文字/小文字混合) にする必要があります。
<BI>	バス識別子です。OPB ポートが 1 つのみのペリフェラルでは必要ありませんが、複数の OPB ポートのあるペリフェラルでは必須です。<BI> には、OPB (大文字、小文字、または大文字/小文字混合) という文字列を含めないでください。複数の OPB ポート (異なるタイプの場合も含む) があるペリフェラルでは、各バス インターフェイスに固有の <BI> を使用する必要があります。

メモ：<BI> を使用している場合は、<Sln> を使用する必要はありません。

## OPB スレーブ出力

OPB への接続には、すべてのスレーブに次の出力が必要です。

```
<BI><Sln>_DBus    : out std_logic_vector(0 to C_<BI>OPB_DWIDTH-1);
<BI><Sln>_errAck   : out std_logic;
<BI><Sln>_retry    : out std_logic;
<BI><Sln>_toutSup  : out std_logic;
<BI><Sln>_xferAck  : out std_logic;
```

例：

```
Tmr_xferAck      : out std_logic;
Uart_xferAck      : out std_logic;
Intc_xferAck      : out std_logic;
```

1. このリリース以降、廃止される予定です。

## OPB スレーブ入力

OPB への接続には、すべてのスレーブに次の入力が必要です。

```
<BI><nOPB>_ABus      : in  std_logic_vector(0 to C_<BI>OPB_AWIDTH-1);
<BI><nOPB>_BE         : in  std_logic_vector(0 to C_<BI>OPB_DWIDTH/8-1);
<BI><nOPB>_Clk        : in  std_logic;
<BI><nOPB>_DBus       : in  std_logic_vector(0 to C_<BI>OPB_DWIDTH-1);
<BI><nOPB>_Rst        : in  std_logic;
<BI><nOPB>_RNW        : in  std_logic;
<BI><nOPB>_select     : in  std_logic;
<BI><nOPB>_seqAddr    : in  std_logic;
```

例：

```
OPB_DBus      : in  std_logic_vector(0 to C_OPB_DWIDTH-1);
IOPB_DBus     : in  std_logic_vector(0 to C_IOPB_DWIDTH-1);
Bus1_OPB_DBus : in  std_logic_vector(0 to C_Bus1_OPB_DWIDTH-1);
```

## マスタ/スレーブ OPB<sup>(1)</sup> ポート

次に示す命名規則は、同じ OPB バスに接続され、同じ入力/出力データバスを共有するマスタおよびスレーブ OPB ポートに適用されます。このタイプのバス インターフェイスは、ペリフェラルにマスタおよびスレーブの両方の機能が備わっている場合、および DMA がペリフェラルに含まれている場合に使用されます。マスタとスレーブで入力/出力データバスを共有するのは有益です。マスタ/スレーブ OPB ポートは、表 5-11 に示す命名規則に従う必要があります。

表 5-11 : マスタ/スレーブ OPB ポートの命名規則

<Mn>	マスタ出力を表す意味のある名前または短縮形を使用します。マスタ出力とバス出力との混乱を避けるため、OPB (大文字、小文字、または大文字/小文字混合) という文字列は含めないでください。
<Sln>	スレーブ出力を表す意味のある名前または短縮形を使用します。スレーブ出力とバス出力との混乱を避けるため、OPB (大文字、小文字、または大文字/小文字混合) という文字列は含めないでください。
<nOPB>	スレーブ入力を表す意味のある名前または短縮形を使用します。<nOPB> の最後の 3 文字は、OPB (大文字、小文字、または大文字/小文字混合) にする必要があります。
<BI>	バス識別子です。OPB ポートが 1 つのみのペリフェラルでは必要ありませんが、複数の OPB ポートのあるペリフェラルでは必須です。<BI> には、OPB (大文字、小文字、または大文字/小文字混合) という文字列を含めないでください。複数の OPB ポート (異なるタイプの場合も含む) があるペリフェラルでは、各バス インターフェイスに固有の <BI> を使用する必要があります。

メモ：<BI> を使用している場合は、<Sln> および <Mn> を使用する必要はありません。

1. このリリース以降、廃止される予定です。

## OPB マスタ/スレーブ出力

OPB への接続には、すべてのマスタおよびスレーブに次の出力が必要です。

```
<BI><Sln>_ABus      : out std_logic_vector(0 to C_<BI>OPB_AWIDTH-1);
<BI><Sln>_BE        : out std_logic_vector(0 to C_<BI>OPB_DWIDTH/8-1);
<BI><Sln>_busLock   : out std_logic;
<BI><Sln>_request    : out std_logic;
<BI><Sln>_RNW       : out std_logic;
<BI><Sln>_select     : out std_logic;
<BI><Sln>_seqAddr    : out std_logic;
<BI><Sln>_DBus      : out std_logic_vector(0 to C_<BI>OPB_DWIDTH-1);
<BI><Sln>_errAck     : out std_logic;
<BI><Sln>_retry      : out std_logic;
<BI><Sln>_toutSup    : out std_logic;
<BI><Sln>_xferAck    : out std_logic;
```

例:

```
IM_request      : out std_logic;
Bridge_request  : out std_logic;
O20b_request    : out std_logic;
```

## OPB マスタ/スレーブ入力

OPB への接続には、マスタおよびスレーブに次の入力が必要です。

```
<BI><nOPB>_ABus      : in  std_logic_vector(0 to C_<BI>OPB_AWIDTH-1);
<BI><nOPB>_BE        : in  std_logic_vector(0 to C_<BI>OPB_DWIDTH/8-1);
<BI><nOPB>_Clk       : in  std_logic;
<BI><nOPB>_DBus      : in  std_logic_vector(0 to C_<BI>OPB_DWIDTH-1);
<BI><nOPB>_errAck     : in  std_logic;
<BI><nOPB>_MGrant     : in  std_logic;
<BI><nOPB>_retry      : in  std_logic;
<BI><nOPB>_RNW       : in  std_logic;
<BI><nOPB>_Rst       : in  std_logic;
<BI><nOPB>_select     : in  std_logic;
<BI><nOPB>_seqAddr    : in  std_logic;
<BI><nOPB>_timeout    : in  std_logic;
<BI><nOPB>_xferAck    : in  std_logic;
```

例:

```
IOPB_DBus      : in  std_logic_vector(0 to C_IOPB_DWIDTH-1);
OPB_DBus       : in  std_logic_vector(0 to C_OPB_DWIDTH-1);
Bus1_OPB_DBus  : in  std_logic_vector(0 to C_Bus1_OPB_DWIDTH-1);
```

## マスタ PLB(1) ポート

マスタ PLB ポートは、表 5-12 の命名規則に従う必要があります。

表 5-12 : マスタ PLB ポートの命名規則

<Mn>	マスタ出力を表す意味のある名前または短縮形を使用します。マスタ出力とバス出力との混乱を避けるため、PLB (大文字、小文字、または大文字/小文字混合) という文字列は含めないでください。
<nPLB>	マスタ入力を表す意味のある名前または短縮形を使用します。<nPLB> の最後の 3 文字は、PLB (大文字、小文字、または大文字/小文字混合) にする必要があります。
<BI>	バス識別子です。マスタ PLB ポートが 1 つのみのペリフェラルでは必要ありませんが、複数のマスタ PLB ポートのあるペリフェラルでは必須です。<BI> には、PLB (大文字、小文字、または大文字/小文字混合) という文字列を含めないでください。複数のマスタ PLB ポートがあるペリフェラルでは、各バスインターフェイスに固有の <BI> を使用する必要があります。

メモ : <BI> を使用している場合は、<Mn> を使用する必要はありません。

## PLB マスタ出力

PLB への接続には、マスタに次の出力が必要です。

```

<BI><Mn>_ABus      : out std_logic_vector(0 to C_<BI>PLB_AWIDTH-1);
<BI><Mn>_BE        : out std_logic_vector(0 to C_<BI>PLB_DWIDTH/8-1);
<BI><Mn>_RNW       : out std_logic;
<BI><Mn>_abort     : out std_logic;
<BI><Mn>_busLock   : out std_logic;
<BI><Mn>_compress  : out std_logic;
<BI><Mn>_guarded   : out std_logic;
<BI><Mn>_lockErr   : out std_logic;
<BI><Mn>_MSize     : out std_logic;
<BI><Mn>_ordered   : out std_logic;
<BI><Mn>_priority  : out std_logic_vector(0 to 1);
<BI><Mn>_rdBurst   : out std_logic;
<BI><Mn>_request   : out std_logic;
<BI><Mn>_size      : out std_logic_vector(0 to 3);
<BI><Mn>_type      : out std_logic_vector(0 to 2);
<BI><Mn>_wrBurst   : out std_logic;
<BI><Mn>_wrDBus    : out std_logic_vector(0 to C_<BI>PLB_DWIDTH-1);

```

例 :

```

IM_request      : out std_logic;
Bridge_request  : out std_logic;
O2Ob_request    : out std_logic;

```

1. このリリース以降、廃止される予定です。

## PLB マスタ入力

PLB への接続には、マスタに次の入力が必要です。

```

<BI><nPLB>_Clk      : in  std_logic;
<BI><nPLB>_Rst      : in  std_logic;
<BI><nPLB>_AddrAck   : in  std_logic;
<BI><nPLB>_Busy      : in  std_logic;
<BI><nPLB>_Err       : in  std_logic;
<BI><nPLB>_RdBTerm   : in  std_logic;
<BI><nPLB>_RdDack    : in  std_logic;
<BI><nPLB>_RdDBus    : in  std_logic_vector(0 to C_<BI>PLB_DWIDTH-1);
<BI><nPLB>_RdWdAddr  : in  std_logic_vector(0 to 3);
<BI><nPLB>_Rearbitrate : in  std_logic;
<BI><nPLB>_SSize     : in  std_logic_vector(0 to 1);
<BI><nPLB>_WrBTerm   : in  std_logic;
<BI><nPLB>_WrDack    : in  std_logic;

```

例：

```

IPLB_MBusy      : in  std_logic;
Bus1_PLB_MBusy  : in  std_logic;

```

## スレーブ PLB<sup>(1)</sup> ポート

スレーブ PLB ポートは、表 5-13 に示す命名規則に従う必要があります。

表 5-13 : スレーブ PLB ポートの命名規則

<Sln>	スレーブ出力を表す意味のある名前または短縮形を使用します。スレーブ出力とバス出力との混乱を避けるため、PLB (大文字、小文字、または大文字/小文字混合) という文字列は含めないでください。
<nPLB>	スレーブ入力を表す意味のある名前または短縮形を使用します。<nPLB> の最後の 3 文字は、PLB (大文字、小文字、または大文字/小文字混合) にする必要があります。
<BI>	バス識別子です。スレーブ PLB ポートが 1 つのみのペリフェラルでは必要ありませんが、複数のスレーブ PLB ポートのあるペリフェラルでは必須です。<BI> には、PLB (大文字、小文字、または大文字/小文字混合) という文字列を含めないでください。複数の PLB ポートがあるペリフェラルでは、各バス インターフェイスに固有の <BI> を使用する必要があります。

メモ：<BI> を使用している場合は、<Sln> を使用する必要はありません。

1. このリリース以降、廃止される予定です。

## PLB スレーブ出力

PLB への接続には、スレーブに次の出力が必要です。

```
<BI><Sln>_addrAck      : out std_logic;
<BI><Sln>_MErr         : out std_logic_vector(0 to C_<BI>PLB_NUM_MASTERS-1);
<BI><Sln>_MBusy        : out std_logic_vector(0 to C_<BI>PLB_NUM_MASTERS-1);
<BI><Sln>_rdBTerm      : out std_logic;
<BI><Sln>_rdComp       : out std_logic;
<BI><Sln>_rdDack       : out std_logic;
<BI><Sln>_rdDBus       : out std_logic_vector(0 to C_<BI>PLB_DWIDTH-1);
<BI><Sln>_rdWdAddr     : out std_logic_vector(0 to 3);
<BI><Sln>_rearbitrate  : out std_logic;
<BI><Sln>_SSize        : out std_logic(0 to 1);
<BI><Sln>_wait         : out std_logic;
<BI><Sln>_wrBTerm      : out std_logic;
<BI><Sln>_wrComp       : out std_logic;
<BI><Sln>_wrDack       : out std_logic;
```

例:

```
Tmr_addrAck  : out std_logic;
Uart_addrAck : out std_logic;
Intc_addrAck : out std_logic;
```

## PLB スレーブ入力

PLB への接続には、スレーブに次の入力が必要です。

```
<BI><nPLB>_Clk         : in  std_logic;
<BI><nPLB>_Rst         : in  std_logic;
<BI><nPLB>_ABus        : in  std_logic_vector(0 to C_<BI>PLB_AWIDTH-1);
<BI><nPLB>_BE          : in  std_logic_vector(0 to C_<BI>PLB_DWIDTH/8-1);
<BI><nPLB>_PValid      : in  std_logic;
<BI><nPLB>_RNW         : in  std_logic;
<BI><nPLB>_abort       : in  std_logic;
<BI><nPLB>_busLock     : in  std_logic;
<BI><nPLB>_compress    : in  std_logic;
<BI><nPLB>_guarded     : in  std_logic;
<BI><nPLB>_lockErr     : in  std_logic;
<BI><nPLB>_masterID    : in  std_logic_vector(0 to C_<BI>PLB_MID_WIDTH-1);
<BI><nPLB>_MSize       : in  std_logic_vector(0 to 1);
<BI><nPLB>_ordered     : in  std_logic;
<BI><nPLB>_pendPri     : in  std_logic_vector(0 to 1);
<BI><nPLB>_pendReq     : in  std_logic;
<BI>_reqpri           : in  std_logic_vector(0 to 1);
<BI><nPLB>_size        : in  std_logic_vector(0 to 3);
<BI><nPLB>_type        : in  std_logic_vector(0 to 2);
<BI><nPLB>_rdPrim      : in  std_logic;
<BI><nPLB>_SAValid     : in  std_logic;
<BI><nPLB>_wrPrim      : in  std_logic;
<BI><nPLB>_wrBurst     : in  std_logic;
<BI><nPLB>_wrDBus      : in  std_logic_vector(0 to C_<BI>PLB_DWIDTH-1);
<BI><nPLB>_rdBurst     : in  std_logic;
```

例:

```
PLB_size  : in  std_logic_vector(0 to 3);
IPLB_size : in  std_logic_vector(0 to 3);
DPLB_size : in  std_logic_vector(0 to 3);
```



## マスタ PLBV46 ポート

マスタ PLBV46 ポートは、表 5-14 の命名規則に従う必要があります。

表 5-14 : マスタ PLBV46 ポートの命名規則

<M>	マスタ出力の接頭辞です。
<PLB_M>	マスタ入力 of 接頭辞です。
<BI>	バス識別子です。マスタ PLBV46 ポートが 1 つのみのペリフェラルでは必要ありませんが、複数のマスタ PLBV46 ポートのあるペリフェラルでは必須です。 複数のマスタ PLBV46 ポートがあるペリフェラルでは、各バス インターフェイスに固有の <BI> を使用する必要があります。<BI> 文字列の最後のアンダースコア ( ) は無視されます。

## PLBV46 マスタ出力

PLBV46 への接続には、マスタに次の出力が必要です。

```

<BI>M_abort           : out std_logic;
<BI>M_ABus             : out std_logic_vector(0 to C_<BI>MPLB>_AWIDTH-1);
<BI>M_UABus           : out std_logic_vector(0 to C_<BI>MPLB>_AWIDTH-1);
<BI>M_BE               : out std_logic_vector(0 to C_<BI>MPLB>_DWIDTH/8-1);
<BI>M_busLock          : out std_logic;
<BI>M_lockErr          : out std_logic;
<BI>M_MSize            : out std_logic;
<BI>M_priority         : out std_logic_vector(0 to 1);
<BI>M_rdBurst          : out std_logic;
<BI>M_request          : out std_logic;
<BI>M_RNW              : out std_logic;
<BI>M_size             : out std_logic_vector(0 to 3);
<BI>M_TAttribute       : out std_logic_vector(0 to 15);
<BI>M_type             : out std_logic_vector(0 to 2);
<BI>M_wrBurst          : out std_logic;
<BI>M_wrDBus           : out std_logic_vector(0 to C_<BI>MPLB>_DWIDTH-1);

```

例 :

```

IPLBM_request         : out std_logic;
Bridge_M_request      : out std_logic;
O20b_M_request        : out std_logic;

```

## PLBV46 マスタ入力

PLBV46 への接続には、マスタに次の入力が必要です。

```

<BI>MPLB_Clk          : in std_logic;
<BI>MPLB_Rst          : in std_logic;
<BI>PLB_MBusy          : in std_logic;
<BI>PLB_MRdErr        : in std_logic;
<BI>PLB_MWrErr        : in std_logic;
<BI>PLB_MIRQ          : in std_logic;
<BI>PLB_MWrBTerm      : in std_logic;
<BI>PLB_MWrDAck       : in std_logic;
<BI>PLB_MAddrAck      : in std_logic;
<BI>PLB_MRdBTerm      : in std_logic;
<BI>PLB_MRdDAck       : in std_logic;
<BI>PLB_MRdDBus       : in std_logic_vector(0 to C_<BI>MPLB>_DWIDTH-1);
<BI>PLB_MRdWdAddr     : in std_logic_vector(0 to 3);

```

```

<BI>PLB_MRearbitrate : in std_logic;
<BI>PLB_MSSize       : in std_logic_vector(0 to 1);
<BI>PLB_MTimeout     : in std_logic;

```

例 :

```

IPLB0_PLB_MBusy      : in std_logic;
Bus1_PLB_MBusy       : in std_logic;

```

## スレーブ PLBV46 ポート

スレーブ PLBV46 ポートは、表 5-15 に示す命名規則に従う必要があります。

表 5-15 : スレーブ PLBV46 ポートの命名規則

<Sl>	スレーブ出力の接頭辞です。
<PLB>	スレーブ入力 of 接頭辞です。
<BI>	バス識別子です。スレーブ PLBV46 ポートが 1 つのみのペリフェラルでは必要ありませんが、複数のスレーブ PLBV46 ポートのあるペリフェラルでは必須です。 複数の PLBV46 ポートがあるペリフェラルでは、各バス インターフェイスに固有の <BI> を使用する必要があります。<BI> 文字列の最後のアンダースコア ( _ ) は無視されます。

## PLBV46 スレーブ出力

PLBV46 への接続には、スレーブに次の出力が必要です。

```

<BI>Sl_addrAck       : out std_logic;
<BI>Sl_MBusy         : out std_logic_vector(0 to C_<BI>/SPLB>_NUM_MASTERS-1);
<BI>Sl_MRdErr        : out std_logic_vector(0 to C_<BI>/SPLB>_NUM_MASTERS-1);
<BI>Sl_MWrErr        : out std_logic_vector(0 to C_<BI>/SPLB>_NUM_MASTERS-1);
<BI>Sl_MIRQ          : out std_logic;
<BI>Sl_rdBTerm       : out std_logic;
<BI>Sl_rdComp        : out std_logic;
<BI>Sl_rdDack        : out std_logic;
<BI>Sl_rddbBus       : out std_logic_vector(0 to C_<BI>/SPLB>_DWIDTH-1);
<BI>Sl_rdWdAddr      : out std_logic_vector(0 to 3);
<BI>Sl_rearbitrate   : out std_logic;
<BI>Sl_SSize         : out std_logic(0 to 1);
<BI>Sl_wait          : out std_logic;
<BI>Sl_wrBTerm       : out std_logic;
<BI>Sl_wrComp        : out std_logic;
<BI>Sl_wrDack        : out std_logic;

```

例 :

```

Tmr_Sl_addrAck       : out std_logic;
Uart_Sl_addrAck      : out std_logic;
IntcSl_addrAck       : out std_logic;

```

## PLBV46 スレーブ入力

PLBV46 への接続には、スレーブに次の入力が必要です。

```

<BI>SPLB_Clk           : in std_logic;
<BI>SPLB_Rst           : in std_logic;
<BI>PLB_ABus           : in std_logic_vector(0 to C_<BI>SPLB>_AWIDTH-1);
<BI>PLB_UABus          : in std_logic_vector(0 to C_<BI>SPLB>_AWIDTH-1);
<BI>PLB_BE             : in std_logic_vector(0 to C_<BI>PLB>_DWIDTH/8-1);
<BI>PLB_busLock        : in std_logic;
<BI>PLB_lockErr        : in std_logic;
<BI>PLB_masterID       : in std_logic_vector(0 to C_<BI>SPLB>_MID_WIDTH-1);
<BI>PLB_PAValid        : in std_logic;
<BI>PLB_rdPendPri      : in std_logic_vector(0 to 1);
<BI>PLB_wrPendPri      : in std_logic_vector(0 to 1);
<BI>PLB_rdPendReq      : in std_logic;
<BI>PLB_wrPendReq      : in std_logic;
<BI>PLB_rdBurst        : in std_logic;
<BI>PLB_rdPrim         : in std_logic;
<BI>PLB_reqPri         : in std_logic_vector(0 to 1);
<BI>PLB_RNW            : in std_logic;
<BI>PLB_SAValid        : in std_logic;
<BI>PLB_MSize          : in std_logic_vector(0 to 1);
<BI>PLB_size           : in std_logic_vector(0 to 3);
<BI>PLB_TAttribute     : in std_logic_vector(0 to 15);
<BI>PLB_type           : in std_logic_vector(0 to 2);
<BI>PLB_wrBurst        : in std_logic;
<BI>PLB_wrDBus         : in std_logic_vector(0 to C_<BI>SPLB>_DWIDTH-1);
<BI>PLB_wrPrim         : in std_logic;

```

例:

```

PLB_size           : in std_logic_vector(0 to 3);
IPLB_size          : in std_logic_vector(0 to 3);
DPORT0_PLB_size    : in std_logic_vector(0 to 3);

```



## バージョン管理ツール (revup)

---

この章では、XPS に含まれるバージョン管理ツールについて説明します。この章には、次のセクションが含まれています。

- 概要
- フォーマット リビジョン ツールによるバックアップおよびアップデート
- フォーマット リビジョン ツールのコマンド オプション
- Version Management Wizard

### 概要

以前のバージョンの EDK で作成したプロジェクトを新しいバージョンの EDK で開くと、フォーマット リビジョン ツールによりプロジェクトが新しいバージョンのフォーマットに自動的にアップデートされます。

フォーマットを変更する前に、XMP (Xilinx® Microprocessor Project)、MHS (Microprocessor Hardware Specification)、MSS (Microprocessor Software Specification) など既存のファイルのバックアップが作成されます。これらのバックアップ ファイルは、プロジェクト ディレクトリの /revup フォルダに保存されます。

IP やドライバのアップデートは、フォーマット リビジョン ツールの後に起動する Version Management Wizard により実行されます。フォーマット リビジョン ツールでは、MHS デザインで使用される IP は変更されません。新しいバージョンのツールでプロジェクトを開くことができるよう、構文がアップデートされるだけです。

## フォーマット リビジョン ツールによるバックアップおよびアップデート

フォーマット リビジョン ツールでは、まず EDK のリリース番号を示す拡張子の付いたバックアップ ファイルが作成されます。たとえば、EDK 10.1 のファイルは、.101 という拡張子で保存されてから、EDK 11.1 用にアップデートされます。

### 11.1 での変更

11.1 を反映してツールをアップデート

- XMP の変更：次のタグが削除されています。
  - ◆ FpgaImpMode : Xplorer と xflow フローを切り替えるために使用されていました。11.1 から、Xplorer は EDK ではサポートされていません。Xplorer フローを使用するには、プロジェクトを ISE® Project Navigator にインスタンシエートしてください。
  - ◆ EnableResetOptimization : タイミングを向上するのにこの設定は必要なくなりました。
  - ◆ InsertNoPads、TopInst、NPL ファイル : これらの設定は XMP から削除されました。
  - ◆ LockAddr、ICacheAddr、DCacheAddr : Address Generator のこれらの設定は、GUI から削除されました。
  - ◆ Simulator、MixLangSim : シミュレータ設定は、すべての XPS プロジェクトに適用されるようになりました。シミュレータ設定は、XPS で [Edit] → [Preferences] をクリックして設定します。
- Simgen の変更：
  - ◆ CompEDKLib が削除され、Compplib が使用されるようになりました。
  - ◆ -E オプションは廃止予定です。
- コマンド ラインの変更：
  - ◆ enable\_reset\_optimization オプションは廃止されています。
- PsfUtility の変更
  - ◆ -tbus サブオプションは廃止されています。
  - ◆ KIND\_OF\_\* 予約ジェネリックは廃止されています。

### 10.1 での変更

10.1 を反映してツールをアップデート。XMP から次のタグが削除されています。

- UseProjNav、PnImportBitFile、PnImportBmmFile : これらの設定は、XMP から削除されています。

### 9.2i での変更

- XMP の変更：XMP タグ EnableResetOptimization が追加され、値が 0 (false) に設定されています。このタグを 1 (true) に設定すると、リセット信号でのタイミングが向上します。
- XMP の変更：XMP タグ EnableParTimingError が追加され、値が 0 (false) に設定されています。このタグを 1 (true) に設定すると、配置配線後にタイミング条件が満たされていない場合にエラー メッセージが表示されます。

## 9.1i での変更点

- **XMP の変更**：プロジェクトからシミュレーション ライブラリ パスが削除されています。シミュレーション ライブラリ パスは、ご使用のマシンのすべての **XPS** プロジェクトに適用されます。
- **XMP の変更**：カスタムリンカ スクリプト のスタック サイズとヒープ サイズが、[Set Compiler Options] ダイアログ ボックスで設定できなくなりました。これらのサイズは、カスタムリンカ スクリプト 内で指定する必要があります。デフォルト のリンカ スクリプト では、[Set Compiler Options] ダイアログ ボックスでスタック サイズとヒープ サイズを表示できます。

## 8.2i での変更点

- **MHS の変更**：サブモジュール デザインの I/O ポートは、個別の **\_I**、**\_O**、および **\_T** ポートに分割されます。これは、**Platgen** の変更 (生成されたスタブ HDL に含まれるバッファはインスタンス化されない、生成された HDL のインターフェイスが **MHS** ファイルのインターフェイスと同じになる) に対応したものです。
- **MHS の変更**：DCMCLK から CLK への最上位ポートの **SIGIS** 値が変更されます。DCMCLK 値は廃止されています。
- プリプロセッサ、アセンブラ、およびリンカ特定のオプションはアドバンス コンパイラ オプションに移動され、個別のオプションは削除されます。
- **XMP の変更**：合成ツール設定は削除されます。

## 8.1i での変更点

- **MSS の変更**：**LIBRARY** ブロックに **PROCINST PARAMETER** が追加されます。このパラメータを使用すると、システムの異なるプロセッサ インスタンスに対してライブラリの設定を変更できます。
- リンカ スクリプトの変更：**CRT** の変更をサポートする新しいベクタ セクションを追加できるよう、**MicroBlaze™** のアプリケーション リンカ スクリプトがアップデートされます。
- リンカ スクリプトの変更：**C++** をサポートする新しいセクションを追加できるよう、**MicroBlaze** のアプリケーション リンカ スクリプトがアップデートされます。
- リンカ スクリプトの変更：**C++** をサポートする新しいセクションを追加できるよう、**PowerPC®** のアプリケーション リンカ スクリプトがアップデートされます。
- プロジェクトの変更なし：**MicroBlaze** アプリケーションで、**xmdstub.elf** のサイズ変更に対応して、プログラムの開始アドレスが **0x0** から **0x50** に変更されます。
- プロジェクトの変更なし：**Spartan®-3** FPGA アーキテクチャを使用するプロジェクトで、**bitgen.ut** が変更されます。

## 7.1i での変更点

リンカ スクリプトの変更：**GCC 3.4.1** をサポートする新しいセクションを追加できるよう、**PowerPC** のアプリケーション リンカ スクリプトがアップデートされます。

## 6.3i での変更点

**MHS の変更**：**MHS** ファイルで、最上位割り込みポートのサブプロパティ **EDGE** および **LEVEL** が **SENSITIVITY** というサブプロパティに統合されます。

## 6.2i での変更点

- プロジェクトの変更なし：ハード乗算器に関連する `mb-gcc` コンパイラ オプションが削除されました (FPGA のみ)。
- MSS の変更：MSS ファイルで、PROCESSOR ブロックが PROCESSOR および OS の 2 つのブロックに分割されます。これに伴い、次の点も変更されます。
  - ◆ Linux および VxWorks の LIBRARY ブロックのステータスが OS ブロックになったため、名前が変更されます。
  - ◆ OS ブロックが導入されたため、Linux および VxWorks で使用されるすべてのペリフェラルが、CONNECT\_TO パラメータの代わりに CONNECTED\_PERIPHS パラメータを使用して指定されます。フォーマット リビジョン ツールが実行されると、以前の CONNECT\_TO ドライバ パラメータがペリフェラルから取得され、OS ブロックの CONNECTED\_PERIPHS パラメータに挿入されます。
  - ◆ MSS ファイルの PROCESSOR ブロックからパラメータ LEVEL、EXECUTABLE、SHIFTER、および DEFAULT\_INIT が削除されます。
  - ◆ PROCESSOR ブロックで、DEBUG\_PERIPHERAL の名前が XMDSTUB\_PERIPHERAL に変更されます。

## フォーマット リビジョン ツールのコマンド オプション

revup を実行するには、コマンド ラインに次のように入力します。

```
revup system.xmp
```

サポートされているオプションは次のとおりです。

-h (ヘルプの表示)：コマンドの使用方法を表示して終了します。

## Version Management Wizard

以前のバージョンの EDK で作成したプロジェクトを新しいバージョンで開くと、フォーマット リビジョン ツールが実行された後に Version Management Wizard が起動します。プロジェクトを最後に処理してからレポジトリで廃止またはアップデートされた IP コアがあると、Version Management Wizard にその変更がリストされ、自動的に最新のバージョンにアップグレードするオプションが示されるか、または最新のコアにアップデートする方法の詳細が表示されます。必要に応じて、ドライバも同様にアップデートできます。MHS および MSS ファイルのバックアップ コピーを作成してから、プロジェクトが変更されます。ウィザードの処理はどの段階でもキャンセルできますが、その場合、現在のバージョンの XPS でプロジェクトを実行することはできません。



# フラッシュ メモリのプログラム

---

この章では、EDK のフラッシュ メモリ プログラム ツールについて説明します。この章は、次の各セクションで構成されています。

- 概要
- サポートされるフラッシュ ハードウェア
- フラッシュ プログラムのパフォーマンス
- フラッシュのプログラム設定のカスタマイズ

## 概要

フラッシュには、次のものをプログラムできます。

- アプリケーションの実行可能イメージ
- FPGA のハードウェア ビットストリーム
- ファイル システム イメージ、サンプル データやアルゴリズム テーブルなどのデータ ファイル

アプリケーションの実行可能イメージをプログラムするのが最も一般的です。デザインのプロセッサのリセットが完了すると、ブロック RAM のプロセッサのリセット位置に保存されている実行コードが開始します。通常ブロック RAM のサイズは数 KB でソフトウェア アプリケーションのイメージ全体を保存するには小さすぎるので、フラッシュ メモリ (MB の単位) を使用します。ブートローダーをブロック RAM に収まるサイズで作成し、リセット時にブートローダーを実行してフラッシュからソフトウェア アプリケーションのイメージを外部メモリにコピーし、ソフトウェア アプリケーションに制御を移行して続行するようにします。

プロジェクトで作成したソフトウェア アプリケーションは、ELF (Executable Linked Format) フォーマットです。フラッシュからソフトウェア アプリケーションをブートロードする場合、ELF イメージをブートロード可能なイメージ フォーマット (Motorola S-record (SREC) など) に変換する必要があります。このようにすると、ブートローダーが単純になり、サイズも小さくなります。EDK では、グラフィカル インターフェイスおよびコマンド ライン オプションを使用して、SREC フォーマットでブートローダを作成できます。フラッシュ ブートローダの作成方法および ELF イメージを SREC フォーマットに変更する方法は、XPS ヘルプを参照してください。

## XPS および SDK を使用したフラッシュ デバイスのプログラム

XPS (Xilinx Platform Studio) とソフトウェア開発キット (SDK) には、外部コンパクト共通フラッシュ インターフェイス (CFI) に準拠したパラレル フラッシュ デバイスをボードにプログラムし、外部メモリ コントローラ (EMC) IP コアを介して接続するためのダイアログ ボックスがあります。フラッシュ メモリ プログラムは、さまざまなフラッシュ ソフトウェアおよびレイアウトに対応するよう設計されています。

フラッシュ メモリのプログラムは、デバッガからプロセッサへの接続を使用して行われます。小型のインシステムフラッシュプログラムスタブが XPS または SDK によりターゲット プロセッサにダウンロードされ、実行されます。インシステム プログラム スタブの動作には、8KB 以上のメモリが必要です。ホスト Tcl スクリプトでコマンドおよびデータを使用してインシステムフラッシュプログラムスタブが制御され、フラッシュがプログラムされます。プログラムされるイメージ ファイルは処理/解釈されることなく、そのままフラッシュ メモリにプログラムされます。ファイルの内容が正しくプログラムされるよう、ソフトウェア アプリケーションおよびハードウェアを設定する必要があります。

## サポートされるフラッシュ ハードウェア

フラッシュ プログラムは、フラッシュ デバイスに対してクエリを送信するのに共通フラッシュ インターフェイス (CFI) を使用するので、フラッシュ デバイスが CFI に準拠していることが必要です。必要な幅のメモリ インターフェイスを形成するため、フラッシュ デバイスのレイアウトも重要です。表 7-1 に、サポートされるフラッシュのレイアウト/コンフィギュレーションを示します。フラッシュのレイアウトが表のコンフィギュレーションと一致しない場合は、フラッシュのプログラム セッションをカスタマイズする必要があります。「[フラッシュのプログラム設定のカスタマイズ](#)」を参照してください。

表 7-1：サポートされるフラッシュ コンフィギュレーション

x8 のみが可能なデバイス (8 ビット データ バスを形成)
x16/x8 が可能なデバイスを x8 モードに設定 (8 ビット データ バスを形成)
x32/x8 が可能なデバイスを x8 モードに設定 (8 ビット データ バスを形成)
x16/x8 が可能なデバイスを x16 モードに設定 (16 ビット データ バスを形成)
x8 のみ可能なデバイスのペア (16 ビット データ バスを形成)
x8 のみ可能なデバイス 4 個 (32 ビット データ バスを形成)
x16 のみ可能なデバイスのペア (32 ビット データ バスを形成)
x32/x8 が可能なデバイスを x32 モードに設定 (32 ビット データ バスを形成)
x32 のみが可能なデバイス (32 ビット データ バスを形成)

物理レイアウトでは、ジオメトリ情報およびコマンド セットなどのその他の論理情報は、CFI を使用して判断されます。フラッシュ プログラムは、表 7-2 にリストされている CFI で定義されたコマンド セットを認識するフラッシュ デバイスのみで使用可能です。

表 7-2 : CFI で定義されたコマンド セット

CFI ベンダー ID	OEM スポンサー	インターフェイス名
1	Intel/シャープ	Intel/シャープ拡張コマンド セット
2	AMD/富士通	AMD/富士通標準コマンド セット
3	Intel	Intel 標準コマンド セット
4	AMD/富士通	AMD/富士通拡張コマンド セット

フラッシュ プログラムは、デフォルトでは、セクタ マップが CFI テーブルに保存されているものと一致するフラッシュ デバイスのみをサポートします。フラッシュ ベンダーによっては、トップ ブートとボトム ブートのフラッシュ デバイスがありますが、両方に同じ CFI テーブルが使用されます。現在のデバイスのブート トポロジを識別するフィールドは CFI 標準には含まれていないので、そのようなフィールドを含むフラッシュ デバイスをフラッシュ プログラムで処理すると問題が発生します。

ブート トポロジを識別するフィールドに関する問題を回避する方法は、[92 ページの「フラッシュのプログラム設定のカスタマイズ」](#)を参照してください。

フラッシュ ハードウェアをプログラムする際は、次の事項が想定されます。

- フラッシュ プログラム スタブによりプログラムが開始する際、フラッシュ ハードウェアがリセット状態になっている。
- フラッシュのすべてのセクタが保護されていない状態である。

フラッシュ ハードウェアがロックされていたり、プログラムできない状態でない場合、フラッシュ プログラム スタブでフラッシュがロック解除されたり初期化されたりすることではなく、エラー メッセージが表示されます。

**メモ：**フラッシュ プログラムでは、各フラッシュ コマンドを DBA (Device Base Address) 値でオフセットする必要のあるデュアル ダイ フラッシュ デバイスは、現在のところサポートされていません。Intel 社の StrataFlash® Embedded Memory (P30) ファミリ フラッシュ メモリの 512Mb デバイスなどがその例です。

## フラッシュ プログラムのパフォーマンス

イメージのプログラム速度には、次の要因が影響します。

- フラッシュ プログラムは JTAG を使用してインシステム プログラム スタブと通信するので、ほとんどの場合、JTAG ケーブルのバンド幅によりフラッシュのプログラム速度が制限されます。
- システムで使用可能な場合、外部メモリをスクラッチ メモリとして使用するのが最適です。このようにすると、デバッガでフラッシュ イメージ データを数回に分けることなくダウンロードできます。
- MicroBlaze™ ソフト プロセッサを使用する場合は、できるだけ高速なコンフィギュレーションをインプリメントするようにします。パレル シフタや乗算器などの機能をオンにしたり、高速ダウンロード機能を使用すると、プログラム速度を向上できます。

## フラッシュのプログラム設定のカスタマイズ

フラッシュのプログラムでは、ハードウェアの違い、フラッシュ コマンド セットの違い、メモリ サイズの制限などを考慮する必要があります。このセクションでは、フラッシュのプログラムのアルゴリズムについて簡単に説明します。アルゴリズムを理解すると、特定の要件に合わせてプログラム設定をカスタマイズできます。

XPS または SDK で [Program Flash] ボタンをクリックすると、次の処理が実行されます。

1. `flash_params.tcl` ファイルが作成され、`etc` フォルダに保存されます。このファイルには、フラッシュのプログラム セッションのパラメータが記述されており、フラッシュ プログラム Tcl で使用されます。
2. XPS または SDK により XMD が起動され、フラッシュ プログラム Tcl スクリプトで `xmd -tcl flashwriter.tcl -nx` などのコマンドが実行されます。このフラッシュ プログラムのホスト Tcl は、インストールディレクトリから取得されます。[Program Flash] ボタンをクリックしたときに独自の Tcl が実行されるようにするには、使用する `flashwriter.tcl` ファイルをプロジェクトのルート ディレクトリにコピーします。ファイルは、まずプロジェクト ディレクトリで検索され、その後インストールディレクトリで検索されます。
3. フラッシュ プログラム Tcl スクリプトにより、フラッシュ プログラム アプリケーションのソース ファイルがインストールディレクトリから `etc/flashwriter` フォルダにコピーされます。このスクリプトは、ダイアログ ボックスで指定したスクラッチ メモリ アドレスからアプリケーションがローカルで実行されるようコンパイルします。独自のフラッシュ ライタソースをコンパイルするには、独自のソース ファイルがコンパイルされるようにローカルにある `flashwriter.tcl` スクリプトを変更します。
4. スクリプトは、フラッシュ プログラムをプロセッサにダウンロードし、メモリのメールボックスを介してフラッシュ プログラムと通信します。フラッシュ プログラムのアドレス空間の変数に対応するメモリ位置にパラメータを書き込み、フラッシュ プログラムを実行させます。
5. フラッシュ プログラムにより 各操作の最後にあるコールバック関数が起動されると、アプリケーションが停止し、コールバック関数の始めにブレークポイントが設定されます。フラッシュ プログラムが停止すると、ホスト Tcl により結果が処理され、必要に応じてコマンドが続けて実行されます。
6. プログラム中、イメージを保存するのに必要なだけのフラッシュ ブロックが消去されます。
7. フラッシュ プログラムにより、使用可能なスクラッチ パッド メモリ の量に応じてストリーミング バッファが割り当てられ、イメージ ファイルがストリーミング手法でプログラムされます。ストリーミング バッファは、フラッシュ プログラム内に割り当てられます。イメージ全体をプログラムするのに十分なスクラッチ メモリ がある場合は、プログラムは短時間で完了します。
8. プログラムが終了すると、フラッシュ プログラム Tcl がフラッシュ ライタに終了コマンドを送信し、XMD セッションを停止します。

次に、カスタム フローの例を示します。

1. `flashwriter.tcl` を `<edk_install>/data/xmd/flashwriter.tcl` から EDK プロジェクト フォルダにコピーします。
2. EDK プロジェクト ディレクトリ内に `sw_services` というディレクトリを作成します。
3. `<edk_install>/data/xmd/flashwriter` ディレクトリ全体を `sw_services` ディレクトリにコピーします。

4. プロジェクト ディレクトリにコピーした flashwriter.tcl で次の行を変更します。

```
set flashwriter_src [file join $xilinx_edk "data" "xmd" "flashwriter"
"src"]
```

変更後：

```
set flashwriter_src [file join "." "sw_services" "flashwriter" "src"]
```

これで、XPS の [Program Flash Memory] ダイアログ ボックスまたは SDK の [Flash Programmer] ダイアログ ボックスで [Program Frash] ボタンをクリックしたときに、sw\_services ディレクトリにコピーしたスクリプトとソースが使用されるようになります。必要に応じて、これらのファイルをカスタマイズします。

etc/flash\_params.tcl ファイルが上書きされないようにするには、コマンド ラインで xmd -tcl flashwriter.tcl コマンドを実行し、etc/flash\_params.tcl ファイルの値が使用されるようにします。

etc/flash\_params.tcl ファイルに含まれるパラメータを表 7-3 に示します。

表 7-3：フラッシュのプログラムのパラメータ

変数	機能
FLASH_FILE	プログラムするファイルへの完全パス
FLASH_BASEADDR	フラッシュ メモリ バンクのベース メモリ
FLASH_PROG_OFFSET	プログラムを実行するフラッシュ メモリ バンクのオフセット
SCRATCH_BASEADDR	プログラム中に使用されるスクラッチ メモリのベース メモリ
SCRATCH_LEN	スクラッチ メモリの長さ (バイト)
XMD_CONNECT	XMD でプロセッサに接続するために使用される接続 コマンド
PROC_INSTANCE	プログラムに使用するプロセッサのインスタンス名
TARGET_TYPE	プログラムに使用するプロセッサ インスタンスのタイプ (MicroBlaze または PowerPC® (405 または 440) プロセッサ)
FLASH_BOOT_CONFIG	<a href="#">「競合するセクタ レイアウトでのフラッシュ デバイスの処理」</a> を参照
EXTRA_COMPILER_FLAGS	MicroBlaze を使用する 場合に、ハードウェア機能のサポート をオンにするコンパイラ オプションを指定します。たとえば、ハードウェア乗算器をイネーブルにした場合は、-mno-xl-soft-mul をここに追加します。PowerPC プロセッサの場合は、この変数は設定しないでください。

## ブートローダ アプリケーション用に ELF ファイルを SREC に手動で変換する方法

XPS または SDK の自動変換機能を使用せずに ELF ファイルを SREC フォーマットに手動で変換する場合は、コマンド ラインを使用できます。たとえば、myexecutable.elf というソフトウェア アプリケーション イメージを変換する場合は、OS のコンソール (Windows では Cygwin) で ELF ファイルを含むフォルダに移動し、次のコマンドを入力します。

```
<platform>-objcopy -O srec myexecutable.elf myexecutable.srec
```

ここで、<platform> はプロセッサが PPC405 または 440 の場合は powerpc-eabi、MicroBlaze の場合は mb です。

これにより、SREC ファイルが生成されます。mb-objcopy および powerpc-eabi-objcopy は、EDK に含まれる GNU バイナリです。

XPS を使用したブート ロードの作成については、XPS ヘルプまたは SDK ヘルプを参照してください。

## 操作上の注意点と回避策

### 競合するセクタ レイアウトでのフラッシュ デバイスの処理

フラッシュ ベンダーによっては、あるセクタ マップを CFI テーブルに、別のセクタ マップをフラッシュ デバイスのブート トポロジに応じてハードウェアに保存するものもあります。ブート トポロジ情報は CFI で規格化されていないので、フラッシュ デバイスで使用されているレイアウトをフラッシュ プログラマで判断する方法はありません。

フラッシュ デバイスのセクタ レイアウトが、そのデバイスの CFI テーブルに保存されているものと異なる場合は、カスタム フラッシュ プログラム フローを作成する必要があります。この場合、フラッシュ デバイスがトップ ブートかボトム ブートかを判断する必要があります。トップ ブート フラッシュ デバイスではフラッシュの最後のセクタが最小のセクタであり、ボトム ブート フラッシュ デバイスでは最初のセクタが最小のセクタです。

フラッシュがトップ ブートかボトム ブートかが判断できたら、ファイルをコピーし、カスタム プログラム フローを作成します。

- ボトム ブート フラッシュの場合は、etc/flash\_params.tcl ファイルに次の行を追加します。

```
set FLASH_BOOT_CONFIG BOTTOM_BOOT_FLASH
```

- トップ ブート フラッシュの場合は、次の行を追加します。

```
set FLASH_BOOT_CONFIG TOP_BOOT_FLASH
```

その後、次のコマンドを使用してフラッシュ プログラムを実行します。

```
xmd -tcl flashwriter.tcl
```

ブート トポロジに応じて、セクタ マップが並べ替えられます。

## AMD/富士通コマンド セットのデータ ポーリング アルゴリズム

ADM/富士通コマンド セットをサポートするフラッシュ デバイスでは、プログラムおよび消去に DQ7 データ ポーリング アルゴリズムが使用されます。

フラッシュ デバイスによっては、データ ポーリング DQ7 ビットの動作を制御するのにコンフィギュレーション レジスタが使用されます。このコンフィギュレーション レジスタを備えているフラッシュ デバイスには、AT49BV322A(T)、AT49BV162A(T)、AT49BV163A(T) などがあります。

DQ7 には、消去中は 0 を出力し、消去が終了したら 1 を出力する必要があります。また、プログラム中は反転したデータを出力し、プログラムが終了したら実際のデータを出力する必要があります。フラッシュ デバイスのコンフィギュレーションが [Program Flash Memory] ダイアログ ボックスで指定したものと異なる場合は、プログラムでエラーが発生する可能性があります。

消去時および消去の終了時に DQ7 から正しい値が出力されるようにコンフィギュレーションをリセットする方法は、フラッシュ デバイスのデータシートを参照してください。





# Bitstream Initializer (BitInit)

---

この章では、Bitstream Initializer (BitInit) ユーティリティについて説明します。次のセクションが含まれています。

- 概要
- BitInit の使用法
- BitInit のコマンド オプション

## 概要

BitInit は、FPGA のブロック RAM に保存されるプロセッサの命令メモリを初期化します。このユーティリティは、MHS (Microprocessor Hardware Specification) ファイルを読み込み、ISE® に含まれる Data2MEM を起動して FPGA のブロック RAM を初期化します。

## BitInit の使用法

BitInit を実行するには、コマンド ラインに次のように入力します。

```
bitinit <mhsfile> [options]
```

メモ：<mhsfile> は、ほかのオプションの前に指定してください。

## BitInit のコマンド オプション

このバージョンの BitInit でサポートされているコマンド オプションは、次のとおりです。

表 8-1 : BitInit のコマンド オプション

オプション	コマンド	説明
入力 BMM ファイルの指定	<b>-bm</b>	アドレス マップおよびプロセッサの命令メモリの場所を含む入力 BMM ファイルを指定します。 デフォルト : implementation/<sysname>_bd.bmm
入力ビットストリームファイルの指定	<b>-bt</b>	メモリ初期化情報を含まない入力ビットストリームファイルを指定します。 デフォルト : implementation/<sysname>.bit
ヘルプの表示	<b>-h</b>	コマンドの使用方法を表示して終了します。
ログ ファイル名の指定	<b>-log</b>	ログ ファイルの名前を指定します。 デフォルト : bitinit.log
ライブラリ パスの指定	<b>-lp</b>	レポジトリ ライブラリのディレクトリへのパスを指定します。このオプションを複数使用して、複数のライブラリを指定できます。
出力ビットストリームファイルの指定	<b>-o</b>	メモリ初期化情報を含む出力ビットストリーム ファイルの名前を指定します。 デフォルト : implementation/download.bit
プロセッサのインスタンス名と ELF ファイルの指定	<b>-pe</b>	命令メモリを構成する ELF ファイルに含まれるプロセッサのインスタンス名を指定します。このオプションは、デザインで使用されているプロセッサの数だけ指定できます。各プロセッサに対し、1 つの ELF ファイルのみをブロック RAM に初期化可能です。
メッセージの非表示	<b>-quiet</b>	ステータス、警告、および情報メッセージを表示せずに実行します。エラー メッセージのみが表示されます。
バージョン番号の表示	<b>-v</b>	BitInit のバージョン番号を表示して終了します。

**メモ :** BitInit を実行すると、Data2MEM により data2mem.dmr というログ ファイルも生成されます。

# GNU コンパイラ ツール

---

## 概要

この章では、GNU コンパイラ ツールについて説明します。次のセクションが含まれています。

- [関連リソース](#)
- [コンパイラのフレームワーク](#)
- [コンパイラの使用法とオプション](#)
- [MicroBlaze コンパイラの使用法とオプション](#)
- [PowerPC コンパイラの使用法とオプション](#)
- [その他のメモ](#)

EDK には、PowerPC® (405 および 440) プロセッサおよび MicroBlaze™ プロセッサ用の GNU コンパイラ ツール (GCC) が含まれています。

- EDK GNU ツールでは、C および C++ 言語の両方がサポートされています。
- MicroBlaze 用の GNU ツールには、mb-gcc および mb-g++ コンパイラ、mb-as アセンブラ、mb-ld リンカがあります。
- PowerPC プロセッサ用の GNU ツールには、powerpc-eabi-gcc および powerpc-eabi-g++ コンパイラ、powerpc-eabi-as アセンブラ、powerpc-eabi-ld リンカがあります。
- C、Math、GCC、および C++ 標準ライブラリも含まれます。
- PowerPC および MicroBlaze プロセッサの GCC ツールは、オープン ソース GCC 4.1.1 ソースを基に作成されています。

コンパイラでは、アセンブラ、リンカ、オブジェクト ダンプなど、一般的なバイナリ ユーティリティも使用されます。PowerPC および MicroBlaze コンパイラでは、バージョン 2.16 の GNU に基づく GNU バイナリ ユーティリティを使用します。言語およびライブラリ サポートの概念、オプション、使用法、例外については、[付録 A「GNU ユーティリティ」](#)を参照してください。

## 関連リソース

### GNU の情報

- GCC4.1.1 リリースの機能の詳細  
<http://gcc.gnu.org/onlinedocs/gcc-4.1.1/gcc/>
- 異なる言語のコンパイラの起動方法  
[http://gcc.gnu.org/onlinedocs/gcc-4.1.1/gcc/Invoking-G\\_002b\\_002b.html#Invoking-G\\_002b\\_002b](http://gcc.gnu.org/onlinedocs/gcc-4.1.1/gcc/Invoking-G_002b_002b.html#Invoking-G_002b_002b)
- GCC オンライン マニュアル  
<http://www.gnu.org/manual/manual.html>
- GNU C++ ライブラリ  
<http://gcc.gnu.org/onlinedocs/libstdc++/manual/spine.html>
- GNU リンカ スクリプト  
<http://www.gnu.org/software/binutils/>

### PowerPC の情報

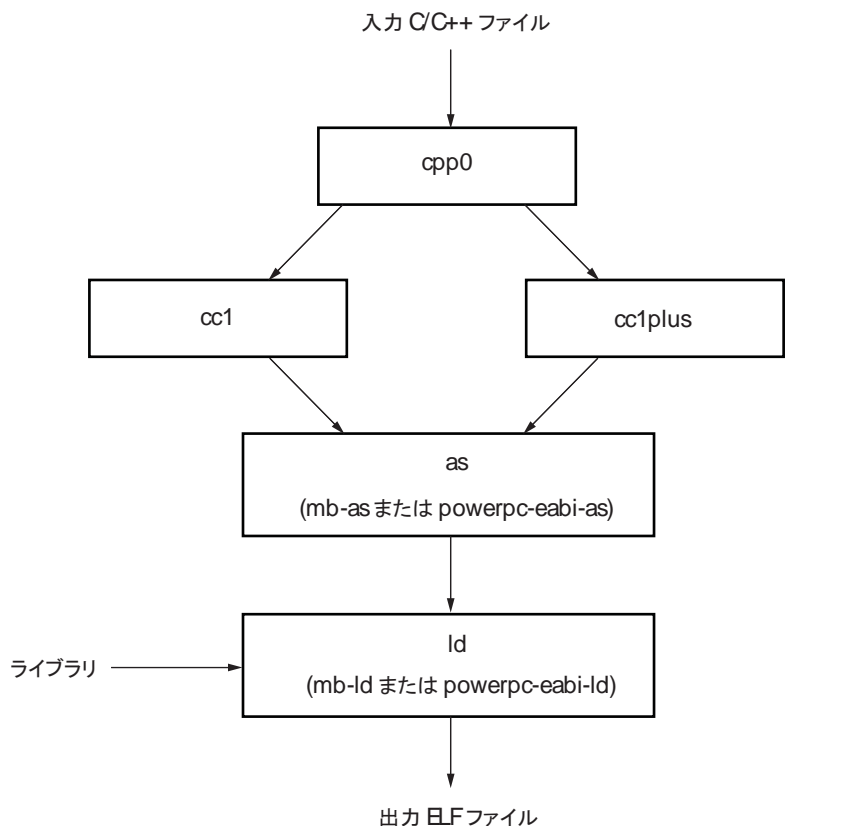
- IBM の Book-E  
<http://www.ibm.com>
- IBM PowerPC パフォーマンス ライブラリ  
<http://sourceforge.net/projects/ppcperflib>
- APU FPU のマニュアル  
[http://japan.xilinx.com/ise/embedded/edk\\_ip.htm](http://japan.xilinx.com/ise/embedded/edk_ip.htm)

### MicroBlaze の情報

- 『MicroBlaze プロセッサ リファレンス ガイド』  
[http://japan.xilinx.com/ise/embedded/edk\\_docs.htm](http://japan.xilinx.com/ise/embedded/edk_docs.htm)

## コンパイラのフレームワーク

このセクションでは、MicroBlaze コンパイラおよび PowerPC プロセッサ コンパイラの主な機能を説明します。図 9-1 に、GNU ツールのフローを示します。



UG111\_05\_101905

図 9-1 : GNU ツール フロー

GNU コンパイラの名前は、MicroBlaze 用は mb-gcc、PowerPC 用は powerpc-eabi-gcc です。GNU コンパイラはラップファイルであり、次の 4 つのツールを呼び出します。

- プリプロセッサ (cpp0)  
コンパイラにより最初に呼び出されるツールで、すべてのマクロをソース ファイルおよびヘッダ ファイルでの定義に置き換えます。
- マシンおよび言語固有コンパイラ  
cpp0 で出力されたプリプロセス済みのコードに対して実行します。次のいずれかのコンパイラが使用されます。
  - ◆ C コンパイラ (cc1)  
入力 C コードを最適化し、アセンブリ コードを生成します。
  - ◆ C++ コンパイラ (cc1plus)  
入力 C++ コードを最適化し、アセンブリ コードを生成します。
- アセンブラ (MicroBlaze プロセッサ用は mb-as、PowerPC プロセッサ用は powerpc-eabi-as)  
アセンブリ コードには、アセンブリ言語のニーモニックが含まれています。アセンブラは、これらのニーモニックをマシン語に変換します。また、コンパイラで生成されたラベルの一部を解決します。アセンブラではオブジェクト ファイルが生成され、これがリンカで処理されます。

- リンカ (MicroBlaze プロセッサ用は `mb-ld`、PowerPC プロセッサ用は `powerpc-eabi-ld`)  
アセンブラで生成されたオブジェクト ファイルをリンクします。コマンド ラインでライブラリが指定されている場合は、アセンブラからの関数をリンクすることにより、コード内の未定義の参照の一部を解決します。

これらのツールのオプションは、105 ページの「よく使用されるコンパイラ オプションの一覧」、109 ページの「リンカ オプション」、115 ページの「MicroBlaze コンパイラ オプションの一覧」、122 ページの「MicroBlaze リンカ オプション」、および 130 ページの「PowerPC コンパイラ オプションの一覧」で説明します。

メモ：この章で GCC と言った場合は、MicroBlaze コンパイラ (`mb-gcc`) および PowerPC プロセッサ コンパイラ (`powerpc-eabi-gcc`) の両方を指します。G++ と言った場合は、MicroBlaze C++ コンパイラ (`mb-g++`) および PowerPC プロセッサ C++ コンパイラ (`powerpc-eabi-g++`) の両方を指します。

## コンパイラの使用法とオプション

### 構文

GNU コンパイラを実行するには、コマンド ラインに次のように入力します。

```
<Compiler_Name> options files...
```

ここで、<Compiler\_Name> は `powerpc-eabi-gcc` または `mb-gcc` です。C++ プログラムをコンパイルするには、`powerpc-eabi-g++` または `mb-g++` コマンドを使用します。

### 入力ファイル

コンパイラには、次のファイルのうち 1 つまたは複数を入力します。

- C ソース ファイル
- C++ ソース ファイル
- アセンブリ ファイル
- オブジェクト ファイル
- リンカ スクリプト

メモ：リンカ スクリプトの指定はオプションです。指定しない場合は、リンカ (`mb-ld` または `powerpc-eabi-ld`) のデフォルト リンカ スクリプトが使用されます。

各ファイルのデフォルトの拡張子については、表 9-1 を参照してください。上記のファイルに加え、コンパイラではライブラリ ファイル `libc.a`、`libgcc.a`、`libm.a`、および `libxil.a` が参照されます。これらのライブラリは、デフォルトでは EDK のインストール ディレクトリにあります。G++ コンパイラを使用した場合は、`libsupc++.a` および `libstdc++.a` ファイルも参照されます。`libsupc++.a` は C++ 言語サポート、`libstdc++.a` は C++ プラットフォーム ライブラリです。

## 出力ファイル

コンパイラでは、次のファイルが出力として生成されます。

- ELF ファイル (デフォルトのファイル名は Solaris では a.out、Windows では a.exe)
- アセンブリ ファイル (-save-temps または -S オプションを使用した場合)
- オブジェクト ファイル (-save-temps または -c オプションを使用した場合)
- プリ プロセッサ出力ファイル (-save-temps オプションを使用した場合、.i または .ii ファイル)

## ファイル タイプとその拡張子

GNU コンパイラでは、ファイルの拡張子からファイルのタイプが判断されます。表 9-1 に有効な拡張子とそのファイル タイプを示します。ファイル タイプに応じて、GCC ラッパ ファイルにより適切なツールが呼び出されます。

表 9-1: ファイルの拡張子

拡張子	ファイル タイプ
.c	C ファイル
.C	C++ ファイル
.cxx	C++ ファイル
.cpp	C++ ファイル
.c++	C++ ファイル
.cc	C++ ファイル
.S	アセンブリ ファイル (プリプロセッサ指示子を含む場合もある)
.s	アセンブリ ファイル (プリプロセッサ指示子は含まない)

## ライブラリ

表 9-2 に、powerpc\_eabi\_gcc および mb\_gcc コンパイラで必要なライブラリを示します。

表 9-2: コンパイラで使用されるライブラリ

ライブラリ	説明
libxil.a	EDK ツール用のドライバ、ソフトウェア サービス (XilMFS など)、初期化ファイルが含まれます。
libc.a	strcmp、strlen などの関数を含む標準 C ライブラリ。
libgcc.a	浮動小数点および 64 ビット演算用のエミュレーション ルーチンを含む GCC の下位ライブラリ。
libm.a	cos、sine などの関数を含む数値計算ライブラリ。
libsupc++.a	例外処理、RTTI などのルーチンを含む C++ サポート ライブラリ。
libstdc++.a	C++ 標準プラットフォーム ライブラリ ストリーム I/O、ファイル I/O、文字列処理などの標準言語クラス。

ライブラリは、両方のコンパイラで自動的にリンクされます。標準ライブラリとは別のライブラリを使用する場合は、使用するライブラリの検索パスを指定する必要があります。libxil.a には、Library Generator (Libgen) を使用してドライバおよびライブラリ ルーチンを追加できます。

## 言語タイプ

GCC コンパイラでは、C および C++ 言語の両方が認識され、対応するコードが生成されます。GCC の規則により、ソース ファイルに GCC または G++ コンパイラを同様に使用することが可能です。使用するコンパイラとソース ファイルの拡張子により、入力ファイルおよび出力ファイルの言語が決定されます。

GCC コンパイラを使用する場合、プログラムの言語は 103 ページの表 9-1 に示すようにファイルの拡張子により決定されます。ファイル拡張子により C++ ソース ファイルと判断された場合は、言語は C++ に設定されます。そのため、たとえば C コードが CC ファイルに含まれていると、GCC コンパイラを使用した場合でも、関数名のマングル処理が行われます。

GCC と G++ の主な違いは、G++ ではデフォルト言語がファイルの拡張子にかかわらず C++ に設定され、C++ サポート ライブラリが読み込まれます。そのため、C ファイルに含まれる C コードを G++ でコンパイルすると、関数名のマングル処理が行われます。

名前マングル処理は、C++ などシンボルのオーバーロードをサポートする言語に特有の概念です。引数によって異なる処理を実行し、異なる戻り値を返すような関数を、オーバーロード (多重定義) された関数と言います。これをサポートするため、C++ コンパイラではその関数名で呼び出される関数のタイプをエンコードして、同じ名前の関数に複数の定義が存在しないようにします。

一部のソース ファイルに C コードが含まれ、その他に C++ コードが含まれる混合コンパイル モードを使用する (一部のファイルのコンパイルに GCC を使用し、その他のファイルのコンパイルに G++ を使用する) 場合は、名前マングル処理に注意する必要があります。C シンボルに対して名前マングル処理が行われないようにするには、シンボル宣言に次の文を使用します。

```
#ifdef __cplusplus
extern "C" {
#endif

int foo();
int morefoo();

#ifdef __cplusplus
}
#endif
```

これらの宣言がヘッダ ファイルで使用されるようにし、ソース ファイルすべてに適用されるようにします。これにより、これらのシンボルの定義および参照をコンパイルする際、C 言語が使用されるようになります。

**メモ :** EDK のすべてのドライバおよびライブラリは、すべてのヘッダ ファイルで上記の規則に従います。G++ を使用してコンパイルする場合は、各ドライバおよびライブラリに記述されているように、必要なヘッダ ファイルを含める必要があります。これにより、コンパイラでライブラリ シンボルが C タイプであることが確実に認識されます。

どちらのコンパイラでコンパイルする場合でも、ファイルを特定の言語に指定するには `-x lang` オプションを使用します。このオプションの詳細は、GNU の Web サイトの GCC マニュアルを参照してください。100 ページの「関連リソース」に、このマニュアルへのリンクがあります。



GCC コンパイラを使用する場合、libstdc++.a および libsupc++.a は自動的にリンクされません。C++ プログラムをコンパイルする場合は、G++ コンパイラを使用して、必要なサポート ライブラリが自動的にリンクされるようにしてください。また、GCC コマンドに -lstdc++ および -lsupc++ を追加することも可能です。

異なる言語に対してコンパイルを起動する方法は、GNU のオンライン マニュアルを参照してください。100 ページの「関連リソース」に、このマニュアルへのリンクがあります。

## よく使用されるコンパイラ オプションの一覧

次に、MicroBlaze および PowerPC の両方のコンパイラに共通のコンパイラ オプションを示します。

**メモ：**これらのオプションでは、大文字と小文字が区別されます。

オプション名をクリックすると、そのオプションの説明にジャンプします。

一般オプション		ライブラリ検索オプション
<a href="#">-E</a>	<a href="#">-Wp,option</a>	<a href="#">-l libraryname</a>
<a href="#">-S</a>	<a href="#">-Wa,option</a>	<a href="#">-L Lib Directory</a>
<a href="#">-c</a>	<a href="#">-Wl,option</a>	
<a href="#">-g</a>	<a href="#">-help</a>	ヘッダ ファイル検索オプション
<a href="#">-gstabs</a>	<a href="#">-B directory</a>	
<a href="#">-On</a>	<a href="#">-L directory</a>	
<a href="#">-v</a>	<a href="#">-I directory</a>	<a href="#">-I Directory Name</a>
<a href="#">-save-temps</a>	<a href="#">-l library</a>	リンカ オプション
<a href="#">-o filename</a>		
		<a href="#">-defsym _STACK_SIZE=value</a>
		<a href="#">-defsym _HEAP_SIZE=value</a>

## 一般オプション

### **-E**

プリプロセスのみを実行し、コンパイル、アセンブリ、リンクは実行しません。プリプロセスの結果は、標準の出力デバイスに表示されます。

### **-S**

コンパイルのみを実行し、アセンブリ、リンクは実行しません。.s ファイルを生成します。

### **-c**

コンパイルおよびアセンブリのみを実行し、リンクは実行しません。.o ファイルを生成します。

### **-g**

出力ファイルに DWARF2 ベースのデバッグ情報を追加します。このデバッグ情報は、GNU デバッガ (mb-gdb または powerpc-eabi-gdb) で使用されます。デバッガでは、ソース レベルまたはアセンブリ レベルでデバッグを実行できます。このオプションは、入力が C または C++ ソース ファイルである場合にのみデバッグ情報を追加します。

**-gstabs**

ソース レベルのアセンブリ ファイル (.s) およびアセンブリ ファイル シンボルに STABS ベースのデバッグ情報を追加します。これはアセンブラ オプションで、GNU アセンブラ (mb-as または powerpc-eabi-as) に直接渡されます。アセンブリ ファイルがコンパイラ (mb-gcc または powerpc-eabi-gcc) を使用してコンパイルされている場合は、**-Wa,** を前に付けてください。

**-On**

GNU コンパイラの最適化レベルを指定します。次の表に示す最適化レベルは、C および C++ ソース ファイルにのみ適用されます。

表 9-3：最適化レベル

<i>n</i>	最適化
0	最適化は実行されません。
1	中レベルの最適化が実行されます。
2	完全な最適化を実行します。
3	完全な最適化を実行します。 サブプログラムをインライン化します。
S	サイズを小さくするよう最適化します。

**メモ：**最適化レベルを 1 以上にすると、コードの構成が変わります。コードのデバッグ中は、最適化レベルを 0 にすることをお勧めします。最適化したプログラムを GDB でデバッグすると、結果が不一致のように見える場合があります。

**-v**

コンパイラおよびコンパイルに関連するすべてのツールを詳細モードで実行します。このオプションを使用すると、ツールで使用されたオプションの詳細が得られるので、各ツールのデフォルト オプションがわかります。

**-save-temps**

コンパイル中に生成された中間ファイルを保存します。次のファイルが保存されます。

- ◆ プリプロセッサ出力 (C コードでは *input\_file\_name.i*、C++ では *input\_file\_name.ii*)
- ◆ アセンブリ フォーマットのコンパイラ (cc1) 出力 (*input\_file\_name.s*)
- ◆ ELF フォーマットのアセンブラ出力 (*input\_file\_name.s*)

デフォルトでは、コンパイル全体が a.out に保存されます。

**-o filename**

コンパイルの出力は、デフォルトでは a.out という ELF ファイルです。このファイル名は、**-o** オプションを使用すると変更できます。出力ファイルは ELF フォーマットで生成されます。

**-Wp,option****-Wa,option****-Wl,option**

コンパイラ (mb-gcc または powerpc-eabi-gcc) はラップ ファイルで、プリプロセッサ、コンパイラ (cc1)、アセンブラ、リンカなどを呼び出します。これらのツールは、コンパイラを介して、または個別に実行できます。

これらのツールで必要なオプションには、最上位のコンパイラでは不要なものもあります。そのようなオプションは、表 9-4 で示すように指定します。

表 9-4 : ツール専用オプションの指定方法

オプション	ツール	例
<b>-Wp,option</b>	プリプロセッサ	<b>mb-gcc -Wp,-D -Wp,MYDEFINE ...</b> プリプロセッサで <b>-D MYDEFINE</b> オプションを使用して、シンボル <b>MYDEFINE</b> が定義されるよう指定します。
<b>-Wa,option</b>	アセンブラ	<b>powerpc-eabi-gcc -Wa,-m405 ...</b> アセンブラで <b>-m405</b> オプションを使用して、PowerPC405 プロセッサがターゲットとして使用されるよう指定します。
<b>-Wl,option</b>	リンカ	<b>mb-gcc -Wl,-M ...</b> リンカで <b>-M</b> オプションを使用して、マップ ファイルが生成されるよう指定します。

**-help**

GNU コンパイラで使用可能なオプションに関する情報を表示します。

これらの情報は、GCC のマニュアルも参照してください。100 ページの「関連リソース」に、このマニュアルへのリンクがあります。

**-B directory**

C のランタイムのライブラリ検索パスに *directory* を加えます。

**-L directory**

ライブラリ検索パスに *directory* を加えます。

**-I directory**

ヘッダ検索パスに *directory* を加えます。

**-l library**

未定義のシンボルを *library* で検索します。

**メモ** : このコマンド ライン オプションで指定したライブラリ名に、**lib** という接頭辞が追加されます。

## ライブラリ検索オプション

### **-l libraryname**

デフォルトでは、libc、libm、libxil などの標準ライブラリのみが検索されます。独自のライブラリも作成でき、このオプションを使用してそのライブラリの名前と機能定義の保存場所を指定できます。このオプションで指定したライブラリ名に lib という接頭辞が追加されます。

コマンドラインでオプションを指定する順序は関係しますが、特に -l オプションでは重要です。このオプションは、ソースファイルの後に使用してください。

たとえば、libproject.a という独自のライブラリを作成している場合は、次のコマンドを使用するとこのライブラリに含まれる関数を使用できます。

```
Compiler Source_Files -L${LIBDIR} -l project
```

**注意** -l オプションをソースファイルの前に使用すると、コンパイラでソースファイルにより呼び出される関数を見つけることができません。これは、コンパイラでの検索は一方方向にのみ行われ、ライブラリのリストが保持されないからです。

### **-L Lib Directory**

ライブラリを検索するディレクトリを指定します。デフォルトのライブラリが検索パスとして設定されており、標準ライブラリはここから検索されますが、-L オプションを指定すると、コンパイラの検索パスにライブラリを検索するディレクトリを追加できます。

## ヘッダ ファイル検索オプション

### **-I Directory\_Name**

ヘッダ ファイルを標準パスで検索する前に、Directory\_Name で指定したディレクトリで検索するよう指定します。

## デフォルトの検索パス

コンパイラ (mb-gcc および powerpc-eabi-gcc) では、特定のパスからライブラリおよびヘッダ ファイルが検索されます。プラットフォームによる検索パスを、次に示します。

ライブラリは、次の順に検索されます。

1. -L オプションで指定されたディレクトリ
2. -B オプションで指定されたディレクトリ
3. 次のライブラリ

- a. \${XILINX\_EDK}/gnu/processor/platform/processor-lib/lib
- b. \${XILINX\_EDK}/lib/processor

**メモ** : processor は、PowerPC プロセッサでは powerpc-eabi、MicroBlaze プロセッサでは microblaze です。

ヘッダ ファイルは、次の順に検索されます。

1. -I オプションで指定されたディレクトリ
2. 次のヘッダ ファイル

- a. \${XILINX\_EDK}/gnu/processor/platform/lib/gcc/processor/4.1.1/include
- b. \${XILINX\_EDK}/gnu/processor/platform/processor-lib/include

初期化ファイルは、次の順に検索されます。

1. **-B** オプションで指定されたディレクトリ
2. `${XILINX_EDK}/gnu/processor/platform/processor-lib/lib`
3. 次のライブラリ
  - a. `${XILINX_EDK}/gnu/processor/platform/processor-lib pro/lib`
  - b. `${XILINX_EDK}/lib/processor`
    - ◆ `processor` は、PowerPC では `powerpc-eabi`、MicroBlaze では `microblaze` です。
    - ◆ `processor-lib` は、PowerPC では `powerpc-eabi`、MicroBlaze では `microblaze-xilinx-elf` です。

**メモ**：`platform` は、Solaris では `sol`、Linux では `lin`、Linux 64 ビットでは `lin64`、Windows Cygwin では `nt` です。

## リンカ オプション

リンカ オプションは、次のとおりです。

### **-defsym \_STACK\_SIZE=value**

スタック領域に割り当てられているメモリ容量を変更します。変数 `_STACK_SIZE` は、スタック領域に割り当てられている合計容量を示します。デフォルト値は 100 ワード (400 バイト) です。スタック領域とヒープ領域の合計に 400 バイト以上必要な場合は、このオプションを使用して `_STACK_SIZE` の値を大きくします。値はバイト単位で指定します。

プログラムでスタック領域を大きくする必要がある場合があります。プログラムで必要なスタックサイズが割り当てられているサイズよりも大きい場合は、プログラムの不正な領域に書き込みが行われ、コードが正常に実行されない場合があります。

**メモ**：ザイリンクスが提供する C ランタイム (CRT) ファイルにリンクされたプログラムでは、スタック サイズを 16 バイト (0x0010) 以上にする必要があります。

### **-defsym \_HEAP\_SIZE=value**

ヒープ領域に割り当てられているメモリ容量を変更します。変数 `_HEAP_SIZE` のデフォルト値は 0 です。

ダイナミック メモリ割り当てルーチンは、ヒープ領域を使用します。プログラムでこのようにヒープ領域を使用する場合は、`_HEAP_SIZE` に適切な値を設定する必要があります。

リンカ スクリプトを XPS から直接生成することもできます。

## メモリのレイアウト

MicroBlaze および PowerPC プロセッサは、32 ビットの論理アドレスを使用し、システムのメモリの 0x0 ~ 0xFFFFFFFF のアドレスを指定できます。このアドレス範囲は、次のタイプに分類できます。

- 予約済みメモリ
- I/O メモリ

## 予約済みメモリ

予約済みメモリは、ハードウェアおよびソフトウェアのプログラム環境で専用を使用するため定義される領域で、割り込みベクタおよびシステム レベル ルーチンの場所などが含まれます。[110 ページの表 9-5](#) に、プロセッサ ハードウェアで定義される MicroBlaze および PowerPC の予約メモリの場所を示します。これらのメモリ ロケーションの詳細は、使用するプロセッサのリファレンス マニュアルを参照してください。

**メモ：**ハードウェアだけでなく、ソフトウェア環境でもメモリが予約される場合があります。ソフトウェアで予約されているメモリ ロケーションがあるかどうかは、使用するソフトウェア プラットフォームのマニュアルを参照してください。

表 9-5：ハードウェアで予約されるメモリ ロケーション

プロセッサ	予約済みメモリ	用途	デフォルトのテキスト 開始アドレス
MicroBlaze	0x0 ~ 0x4F	リセット、割り込み、 例外、その他の予約済 みベクタの場所	0x50
PowerPC	0xFFFFFFFFFC ~ 0xFFFFFFFF	リセット ベクタの場所	0xFFFF0000

## I/O メモリ

I/O メモリは、プログラムがプロセッサバス上のメモリ マップされたペリフェラルと通信するために使用されます。これらのアドレスは、ハードウェア プラットフォーム仕様の一部として定義されます。

## ユーザーおよびプログラム メモリ

ユーザーおよびプログラム メモリとは、コンパイルされた実行ファイルの実行に必要なすべてのメモリを指します。命令、読み出し専用データ、読み出し/書き込みデータ、プログラム スタック、プログラム ヒープの保存に使用されます。これらのセクションは、システム内のアドレス指定可能なメモリであればどこにでも保存できます。デフォルトでは、コンパイラで生成されたコードおよびデータは、[表 9-5](#) にリストされているアドレスから開始して、連続するメモリ ロケーションに保存されます。これが最も一般的なプログラムのメモリ レイアウトです。プログラムの開始ロケーションを変更するには、MicroBlaze プロセッサでは `_TEXT_START_ADDR` シンボル、PowerPC プロセッサでは `_START_ADDR` シンボルを定義します。

ELF ファイルを異なるメモリに分割する必要がある場合は、リンカ コマンド言語を使用します。詳細は、[113 ページの「リンカ スクリプト」](#)を参照してください。実行ファイルのメモリ マップを変更するのは、次のような場合です。

- ◆ 長いコードを複数の小型メモリに分割する場合
- ◆ 頻繁に実行されるセクションを高速メモリにマップする場合
- ◆ 読み出し専用のセグメントを不揮発性のフラッシュ メモリにマップする場合

実行ファイルの分割方法に制限はありません。分割は、出力セクション レベルまたは関数レベル、データ レベルで行うことができます。生成される ELF ファイルが不連続となり、メモリ マップにギャップが存在することがあります。予約済みロケーションを使用しないよう注意してください。

または、ツールで提供される予約済みメモリ ロケーションのデフォルト バイナリ データを変更することも可能です。この場合、リンカで提供されるデフォルトのスタートアップ ファイルおよびメモリ マップを置き換える必要があります。

## オブジェクト ファイルのセクション

実行ファイルは、オブジェクト ファイル (.o ファイル) の入力セクションをリンクして作成します。デフォルトでは、コンパイラにより標準的な明確に定義されたセクションからコードが作成されます。各セクションには、その意味および目的に応じて名前が付けられています。オブジェクト ファイルのさまざまな標準セクションを図 9-2 に示します。

これらのセクションに加え、独自のセクションを作成して、メモリに割り当てることもできます。

オブジェクト (実行) ファイルのセクション

.text	テキスト セクション
.rodata	読み出し専用データ セクション
.sdata2	読み出し専用スモール データ セクション
.sbss2	読み出し専用未初期化スモール データ セクション
.data	読み出し/書き込みデータ セクション
.sdata	読み出し/書き込みスモール データ セクション
.sbss	未初期化スモール データ セクション
.bss	未初期化データ セクション
.heap	プログラム ヒープ メモリ セクション
.stack	プログラム スタック メモリ セクション

X11005

図 9-2 : オブジェクト (実行) ファイルのセクション レイアウト

このほかに、.init、.fini、.ctors、.dtors、.got、.got2、.eh\_frame など、通常は変更しない予約済みセクションがあります。

### .text

オブジェクト ファイルのこのセクションには実行可能なプログラム命令が含まれており、**x** (実行コード)、**r** (読み出し専用)、および **i** (初期化) フラグが付けられています。プロセッサ命令バスでアドレス指定可能な初期化済み ROM に割り当てることができます。

### .rodata

このセクションには読み出し専用データが含まれており、**r** (読み出し専用) および **i** (初期化) フラグが付けられています。**.text** セクションと同様、プロセッサ命令バスでアドレス指定可能な初期化済み ROM に割り当てることができます。

### **.sdata2**

このセクションは **.rodata** セクションと同様ですが、8 バイト未満の読み出し専用データが含まれます。このセクションのデータは、すべて読み出し専用のスモール データ アンカーへの参照を使用してアクセスします。これにより、このセクションのすべてのデータに 1 つの命令でアクセスできます。このセクションに配置するデータのサイズは、**-G** オプションで変更できます。このセクションには、**r** (読み出し専用) および **i** (初期化) フラグが付けられています。

### **.data**

このセクションには読み出し/書き込みデータが含まれており、**w** (読み出し/書き込み) および **i** (初期化) フラグが付けられています。初期化済みの **RAM** にマップする必要があります。**ROM** にはマップできません。

### **.sdata**

このセクションには、8 バイトよりも小さい読み出し/書き込み可能なデータが含まれます。このセクションに配置するデータのサイズは、**-G** オプションで変更できます。このセクションのデータは、すべて読み出し/書き込みのスモール データ アンカーへの参照を使用してアクセスします。これにより、このセクションのすべてのデータに 1 つの命令でアクセスできます。このセクションには、**w** (読み出し/書き込み) および **i** (初期化) フラグが付けられており、初期化済み **RAM** にマップする必要があります。

### **.sbss2**

このセクションには、8 バイトよりも小さい読み出し専用の初期化されないデータが含まれます。このセクションに配置するデータのサイズは、**-G** オプションで変更できます。このセクションには、**r** (読み出し) フラグが付けられており、**ROM** にマップする必要があります。

### **.sbss**

このセクションには、8 バイトよりも小さい初期化されないデータが含まれます。このセクションに配置するデータのサイズは、**-G** オプションで変更できます。このセクションには、**w** (読み出し/書き込み) フラグが付けられており、**RAM** にマップする必要があります。

### **.bss**

このセクションには、初期化されていないデータが含まれます。このセクションには、**w** (読み出し/書き込み) フラグが付けられており、**RAM** にマップする必要があります。

### **.heap**

このセクションには、グローバル プログラム ヒープとして使用される初期化されていないデータが含まれます。このセクションのメモリは、ダイナミック メモリ割り当てルーチンにより割り当てられます。**RAM** にマップする必要があります。

### **.stack**

このセクションには、プログラム スタックとして使用される初期化されていないデータが含まれます。**RAM** にマップする必要があります。通常は、**.heap** セクションのすぐ後に配置されます。リンカによっては、**.stack** と **.heap** セクションが統合され、**.bss\_stack** というセクションに配置されます。

### **.init**

このセクションには言語初期化コードが含まれており、**.text** セクションと同じフラグが付けられています。初期化済みの **ROM** にマップする必要があります。



### **.fini**

このセクションには言語クリーンアップ コードが含まれており、**.text** セクションと同じフラグが付けられています。初期化済みの **ROM** にマップする必要があります。

### **.ctors**

このセクションにはプログラムの起動時に呼び出す必要のある関数がリストされ、**.data** と同じフラグが付けられています。初期化済みの **RAM** にマップする必要があります。

### **.dtors**

このセクションにはプログラムの終了時に呼び出す必要のある関数がリストされており、**.data** と同じフラグが付けられています。初期化済みの **RAM** にマップする必要があります。

### **.got2/.got**

このセクションにはプログラム データへのポインタが含まれており、**.data** と同じフラグが付けられています。初期化済みの **RAM** にマップする必要があります。

### **.eh\_frame**

このセクションには例外処理用のフレーム巻き戻し情報が含まれており、**.rodata** と同じフラグが付けられています。初期化済みの **ROM** にマップできます。

### **.tbss**

このセクションにはプログラムのメモリ イメージの一部となるスレッドが初期化されていないローカル データが含まれており、**.bss** と同じフラグが付けられています。**RAM** にマップする必要があります。

### **.tdata**

このセクションには、プログラムのメモリ イメージの一部となるスレッドが初期化済みのローカル データが含まれています。初期化済みの **RAM** にマップする必要があります。

### **.gcc\_except\_table**

このセクションには、言語特定のデータが含まれています。初期化済みの **RAM** にマップする必要があります。

### **.jcr**

このセクションには、コンパイル済みの **Java** クラスを登録するために必要な情報が含まれています。初期化済みの **RAM** にマップする必要があります。

### **.fixup**

このセクションには、フィックスアップ ページ テーブルやフィックスアップ レコード テーブルを実行するために必要な情報が含まれています。初期化済みの **RAM** にマップする必要があります。

## リンカ スクリプト

リンカ ユーティリティでは、リンカ スクリプトで指定したコマンドを使用してユーザー プログラムを複数のメモリ ブロックに分割します。リンカ スクリプトは、すべての入力オブジェクト ファイルのすべてのセクションから実行ファイルへのマップを記述します。出力セクションは、システムのメモリにマップされます。プログラム データを連続するメモリに割り当てるデフォルトを変更しない場合は、リンカ スクリプトは必要ありません。デフォルトのリンカ スクリプトが用意されています。

プログラムの開始ロケーションのみを変更するには、次の例に示すように、MicroBlaze では `_TEXT_START_ADDR` シンボル、PowerPC では `_START_ADDR` シンボルを定義します。

```
mb-gcc <input files and flags> -Wl,-defsym -Wl,_TEXT_START_ADDR=0x100
```

```
powerpc-eabi-gcc <input files and flags> -Wl,-defsym  
-Wl,_TEXT_START_ADDR=0x2000
```

```
mb-ld <.o files> -defsym _TEXT_START_ADDR=0x100
```

`$XILINX_EDK/gnu/<processor_name>/<platform>/<processor_name>/lib/ldscripts` に、次のリンカ スクリプトが含まれています。

- `elf32<procname>.x`: この後に示すオプションが使用されていない場合のデフォルト
- `elf32<procname>.xn`: `-n` オプションを使用した場合
- `elf32<procname>.xbn`: `-N` オプションを使用した場合
- `elf32<procname>.xr`: `-r` オプションを使用した場合
- `elf32<procname>.xu`: `-Ur` オプションを使用した場合

ここで、`<procname>` は `ppc` または `microblaze`、`<processor_name>` は `powerpc-eabi` または `microblaze`、`<platform>` は `lin` または `nt` です。

リンカ スクリプトを使用するには、GCC コマンド ラインで指定します。次のように、コンパイラに `-T <script>` オプションを使用します。

```
compiler -T <linker_script> <Other Options and Input Files>
```

リンカを個別に実行する場合は、リンカ スクリプトを次のように指定します。

```
linker -T <linker_script> <Other Options and Input Files>
```

このコマンドを使用すると、デフォルトのリンカ スクリプトの代わりに指定したリンカ スクリプトが使用されます。プログラム用のリンカ スクリプトは、XPS および SDK から生成できます。

XPS または SDK で、[Software] → [Generate Linker Script] をクリックします。

これにより、リンク スクリプト生成ユーティリティが開きます。セクションからメモリへのマップは、ここでを行います。スタックおよびヒープのサイズとメモリ マップもここで設定できます。リンカ スクリプトが生成されると、XPS または SDK で対応するアプリケーションをコンパイルしたときにそのスクリプトが自動的に GCC に入力されます。

リンカ スクリプトは、メモリに変数または関数を割り当てるために使用できます。これには、C コードのセクション属性を使用します。また、リンカ スクリプトでメモリのセクションにオブジェクト ファイルを割り当てることもできます。これらの機能およびその他の機能については、オンライン `binutils` マニュアルの一部である GNU リンカのマニュアルを参照してください。100 ページの「関連リソース」に、GNU マニュアルへのリンクがあります。MicroBlaze および PowerPC のリンカ スクリプトで割り当てられる入力セクションについては、123 ページの「MicroBlaze リンカ スクリプトで割り当てられるセクション」および 132 ページの「PowerPC プロセッサ リンカ スクリプトで割り当てられるセクション」を参照してください。

## MicroBlaze コンパイラの使用法とオプション

MicroBlaze 用の GNU コンパイラは、標準の GNU ソースに基づいています。MicroBlaze コンパイラに特定の機能およびオプションを、次に説明します。MicroBlaze コンパイラでコンパイルする場合、プリプロセッサで自動的に `__MICROBLAZE__` 定義が使用されます。この定義は、どのような条件コードでも使用できます。

### MicroBlaze コンパイラ

ザイリンクス MicroBlaze ソフト プロセッサ用の `mb-gcc` コンパイラでは、専用のオプションが追加されているだけでなく、GNU コンパイラでサポートされている一部のオプションが変更されています。ここでは、その両方のオプションについて説明します。

### MicroBlaze コンパイラ オプションの一覧

オプション名をクリックすると、そのオプションの説明にジャンプします。

#### プロセッサ機能選択オプション

`-mcpu=vX.YY.Z`  
`-mno-xl-soft-mul`  
`-mxl-multiply-high`  
`-mno-xl-multiply-high`  
`-mxl-soft-mul`  
`-mno-xl-soft-div`  
`-mxl-soft-div`  
`-mxl-barrel-shift`  
`-mno-xl-barrel-shift`  
`-mxl-pattern-compare`  
`-mno-xl-pattern-compare`  
`-mhard-float`  
`-msoft-float`  
`-mxl-float-convert`  
`-mxl-float-sqrt`

#### 一般プログラム オプション

`-msmall-divides`  
`-mxl-gp-opt`  
`-mno-clearbss`  
`-mxl-stack-check`

#### アプリケーション実行モード

`-xl-mode-executable`  
`-xl-mode-xmdstub`  
`-xl-mode-bootstrap`  
`-xl-mode-novectors`

#### MicroBlaze リンカ オプション

`-defsym _TEXT_START_ADDR=value`  
`-relax`  
`-N`

### プロセッサ機能選択オプション

#### `-mcpu=vX.YY.Z`

MicroBlaze ハードウェアのバージョン `vX.YY.Z` に適したコードを生成します。プロセッサ用に最適化された正しいコードを生成するには、このオプションでプロセッサのハードウェア バージョンを指定します。

指定するバージョンによって、処理が異なります。

- **Pr-v3.00.a** : 3 段プロセッサ パイプライン モードを使用します。命令を遅延スロットに移動する例外は禁止されません。
- **v3.00.a** および **v4.00.a** : 3 段プロセッサ パイプライン モデルを使用します。命令を遅延スロットに移動する例外は禁止されます。
- **v5.00.a** 以降 : 5 段プロセッサ パイプライン モードを使用します。命令を遅延スロットに移動する例外は禁止されません。

### **-mno-xl-soft-mul**

32 ビット乗算に対し、ハードウェア乗算命令を使用できるようにします。

MicroBlaze プロセッサには、ハードウェア乗算リソースの使用をオン/オフにするオプションがあります。MicroBlaze でハードウェア乗算オプションがイネーブルになっている場合は、このオプションを使用する必要があります。ハードウェア乗算を使用すると、アプリケーションのパフォーマンスが向上します。このオプションを使用すると、C プリプロセッサ定義 HAVE\_HW\_MUL が自動的に定義され、この機能が使用可能かどうかに基づいて、ハードウェアに適した C またはアセンブリコードが記述されます。MicroBlaze での乗算器オプションの使用については、『MicroBlaze プロセッサ リファレンス ガイド』を参照してください。100 ページの「関連リソース」に、このマニュアルへのリンクがあります。

### **-mxl-multiply-high**

MicroBlaze には、32X32 ビットの乗算の上位 32 ビットを計算する命令をイネーブルにするオプションがあります。このオプションは、コンパイラでこれらの上位ビット乗算命令を使用するよう指示します。このオプションを使用すると、C プリプロセッサ定義 HAVE\_HW\_MUL\_HIGH が自動的に定義され、この機能が使用可能かどうかに基づいて、ハードウェアに適した C またはアセンブリコードが記述されます。MicroBlaze での上位ビット乗算オプションの使用については、『MicroBlaze プロセッサ リファレンス ガイド』を参照してください。100 ページの「関連リソース」に、このマニュアルへのリンクがあります。

### **-mno-xl-multiply-high**

上位ビット乗算命令を使用しないよう指定します。このオプションは、デフォルトで設定されています。

### **-mxl-soft-mul**

MicroBlaze にハードウェア乗算器が含まれていないことを示します。デバイスにハードウェア乗算器がない場合、32 ビット乗算操作はソフトウェア エミュレーション ルーチン \_\_mulsi3 に置換されます。このオプションは、デフォルトで設定されています。

### **-mno-xl-soft-div**

MicroBlaze にハードウェア除算ユニットをインスタンス化できます。除算ユニットがある場合、このオプションを指定すると、コンパイルされるプログラムでハードウェア除算命令が使用できることが示されます。

プログラムに除算処理が多数含まれる場合、このオプションを使用するとパフォーマンスが向上します。このオプションを使用すると、C プリプロセッサ定義 HAVE\_HW\_DIV が自動的に定義され、この機能が使用可能かどうかに基づいて、ハードウェアに適した C またはアセンブリコードが記述されます。MicroBlaze でのハードウェア除算オプションの使用については、『MicroBlaze プロセッサ リファレンス ガイド』を参照してください。100 ページの「関連リソース」に、このマニュアルへのリンクがあります。

### **-mxl-soft-div**

ターゲットの MicroBlaze にハードウェア除算ユニットがないことを示します。

このオプションは、デフォルトで設定されています。このオプションを設定すると、すべての 32 ビット除算が対応するソフトウェア エミュレーション ルーチン (\_\_divsi3、\_\_udivsi3) に置換されます。

### **-mxl-barrel-shift**

MicroBlaze プロセッサは、バレル シフトを組み込むようコンフィギュレーションできます。プロセッサのバレル シフト機能を使用するには、**-mxl-barrel-shift** オプションを使用します。

デフォルトではバレル シフトはないと判断され、オペランドのシフトには加算と乗算が使用されます。バレル シフトをイネーブルにすると、特に浮動小数点ライブラリを使用している場合に、アプリケーションの速度が大幅に向上します。このオプションを使用すると、C プリプロセッサ定義 **HAVE\_HW\_BSHIFT** が自動的に定義され、この機能が使用可能かどうかに基づいて、ハードウェアに適した C またはアセンブリ コードが記述されます。MicroBlaze でのバレル シフト オプションの使用については、『MicroBlaze プロセッサ リファレンス ガイド』を参照してください。100 ページの「[関連リソース](#)」に、このマニュアルへのリンクがあります。

### **-mno-xl-barrel-shift**

ハードウェア バレル シフト命令を使用しないよう指定します。このオプションは、デフォルトで設定されています。

### **-mxl-pattern-compare**

コンパイラでのパターン比較命令の使用をオンにします。

パターン比較命令を使用すると、プログラムのブール演算の速度が向上します。また、パターン比較操作では、**strcpy**、**strlen**、**strcmp** などの文字列処理ルーチンにおいて、バイト長ではなくワード長での操作が可能になります。文字列処理ルーチンを多用するプログラムでは、これにより処理速度が大幅に向上します。このオプションを使用すると、C プリプロセッサ定義 **HAVE\_HW\_PCMP** が自動的に定義され、この機能が使用可能かどうかに基づいて、ハードウェアに適した C またはアセンブリ コードが記述されます。MicroBlaze でのパターン比較オプションの使用については、『MicroBlaze プロセッサ リファレンス ガイド』を参照してください。100 ページの「[関連リソース](#)」に、このマニュアルへのリンクがあります。

### **-mno-xl-pattern-compare**

パターン比較命令を使用しないよう指定します。このオプションは、デフォルトで設定されています。

### **-mhard-float**

コンパイラでの単精度浮動小数点命令 (**fadd**、**frsub**、**fmul**、**fdiv**) の使用をオンにします。

また、**fcmp.p** 命令 (*p* は **le**、**ge**、**lt**、**gt**、**eq**、**ne** などの述語条件) も使用します。これらの命令は、ハードウェアで FPU がイネーブルの場合に、MicroBlaze でデコードおよび実行されます。このオプションを使用すると、C プリプロセッサ定義 **HAVE\_HW\_FPU** が自動的に定義され、この機能が使用可能かどうかに基づいて、ハードウェアに適した C またはアセンブリ コードが記述されます。MicroBlaze でのハードウェア浮動小数点の使用については、『MicroBlaze プロセッサ リファレンス ガイド』を参照してください。100 ページの「[関連リソース](#)」に、このマニュアルへのリンクがあります。

### **-msoft-float**

浮動小数点コードのソフトウェア エミュレーション ライブラリを使用するよう指定します。このオプションは、デフォルトで設定されています。

### **-mxl-float-convert**

コンパイラでの単精度浮動小数点変換命令 (`fint` および `flt`) の使用をオンにします。これらの命令は、ハードウェアで FPU がイネーブルになっており、これらのオプションの命令がイネーブルの場合に、MicroBlaze によりネイティブでデコードされ、実行されます。

MicroBlaze でのハードウェア浮動小数点の使用については、『MicroBlaze プロセッサ リファレンス マニュアル』を参照してください。100 ページの「関連リソース」に、このマニュアルへのリンクがあります。

### **-mxl-float-sqrt**

コンパイラでの単精度浮動小数点平方根命令 (`fsqrt`) の使用をオンにします。この命令は、ハードウェアで FPU がイネーブルになっており、このオプションの命令がイネーブルの場合に、MicroBlaze によりネイティブでデコードされ、実行されます。

MicroBlaze でのハードウェア浮動小数点の使用については、『MicroBlaze プロセッサ リファレンス マニュアル』を参照してください。100 ページの「関連リソース」に、このマニュアルへのリンクがあります。

## 一般プログラム オプション

### **-msmall-divides**

ハードウェア除算器がない場合に、小さい数値の除算に対して最適化されたコードを生成します。分母と分子が 0 ~ 15 の間にあるような符号付き整数の除算では、このオプションを使用するとルックアップ テーブルに基づく高速の除算を生成できます。ハードウェア除算器がイネーブルの場合は、このオプションは無視されます。

### **-mxl-gp-opt**

プログラムに上位 16 ビットに 0 以外の値が含まれるアドレスがある場合、読み込み/格納操作には 2 つの命令が必要です。

MicroBlaze ABI には 2 つのグローバル スモール データ領域があり、それぞれ 64KB までのデータを保存できます。これらのデータ領域にあるメモリ ロケーションには、スモール データ領域アンカーおよび 16 ビットの即値を使用してアクセスできるので、スモール データ領域への読み込みまたは格納操作を 1 つの命令のみで実行できます。この最適化をオンにするには、`-mxl-gp-opt` オプションを使用します。しきい値未満のサイズの変数はこれらの領域に保存され、少ない命令数でアドレス指定可能です。アドレスは、リンク段階で求められます。

**注意** このオプションを使用する場合、プログラムのビルド プロセスのコンパイル コマンドとリンク コマンドの両方で指定する必要があります。どちらか一方のみで使用すると、コンパイル、リンク、またはランタイム エラーが発生する可能性があります。

### **-mno-clearbss**

このオプションは、シミュレーションで使用するプログラムをコンパイルする際に有益です。

C 言語の標準に基づき、初期化されていないグローバル変数は `.bss` セクションに割り当てられ、プログラムの実行が開始したときの値は 0 になります。通常これは、プログラムの実行が開始したときに、C スタートアップ ファイルをループで実行して `.bss` セクションに 0 が記述されるようにすることで達成します。さらにコンパイラを最適化すると、C コードで 0 に割り当てられたグローバル変数が `.bss` セクションに割り当てられます。



シミュレーション環境では、上記の 2 言語機能は余分なオーバーヘッドとなる場合があります。シミュレータによっては、メモリ全体を自動的に 0 にするものもあります。通常的环境でも、グローバル変数が開始時に 0 になっていなくても良いような C コードを記述することも可能です。そのような場合に、このオプションは有益です。このオプションを使用すると、C スタートアップ ファイルにより .bss セクションが 0 に初期化されることはなくなります。また、.bss セクションに 0 に初期化されたグローバル変数は割り当てられず、これらの変数は .data セクションに移動されます。このオプションにより、アプリケーションの起動時間が短縮される場合があります。このオプションは、グローバル変数を 0 に初期化することが必要なコードを使用していない場合、またはシミュレーション プラットフォームで自動的にメモリが 0 に初期化される場合にのみ使用してください。

### **-mxl-stack-check**

プログラムの実行中にスタック オーバーフローが発生しているかどうかをチェックするよう指定します。

このオプションを使用すると、各関数のプロローグ コード内に、スタック ポインタ値と使用可能なメモリを比較するコードが挿入されます。スタック ポインタが使用可能なメモリを超えている場合は、プログラムはサブルーチン `_stack_overflow_exit` にジャンプします。このサブルーチンは、変数 `_stack_overflow_error` の値を 1 に設定します。

ソースコードに関数 `_stack_overflow_exit` を挿入することにより、標準のスタック オーバーフローハンドラの代わりにこの関数をスタック オーバーフロー ハンドラとして使用できます。

## アプリケーション実行モード

### **-xl-mode-executable**

`mb-gcc` でプログラムをコンパイルする際のデフォルト モードです。`mb-gcc` を使用する場合は、指定する必要はありません。このオプションを使用すると、スタートアップ ファイル `crt0.o` が使用されます。

### **-xl-mode-xmdstub**

Xilinx Microprocessor Debugger (XMD) を使用すると、アプリケーションをボード上で XMDStub を使用してデバッグできます。このデバッグ モードは、XMDStub モードと呼ばれます。このオプションを使用すると、XMDStub モードでデバッグが実行されます。この場合、アドレス ロケーション `0x0 ~ 0x800` は XMDStub で使用するために予約されています。`-xl-mode-xmdstub` を使用すると、次のようになります。

- プログラムの開始アドレスは `0x800` に設定されます。この開始アドレスは、リンカ スクリプトで `_TEXT_START_ADDR` の値を指定するか、リンカ オプションを使用すると変更できます。リンカ オプションの詳細は、[109 ページの「リンカ オプション」](#)を参照してください。開始アドレスが `0x800` 未満に指定されると、アドレス オーバーラップ エラーが発生します。
- `crt1.o` が初期化ファイルとして使用されます。プログラムの実行が終了すると、XMDStub に戻ります。

**メモ:** ビットストリームに XMDStub が含まれている場合は、`-xl-mode-xmdstub` オプションを使用してください。このモードは、システムがデバッグなしでコンパイルされている場合、およびハードウェア デバッグがオンの場合には使用しないでください。XMD を使用したデバッグの詳細は、[第 11 章「Xilinx Microprocessor Debugger \(XMD\)」](#)を参照してください。

### **-xl-mode-bootstrap**

ブートローダーを使用して読み込むアプリケーションをコンパイルする際に使用します。通常ブートローダーは、不揮発性メモリに保存され、プロセッサ リセット ベクタにマップされます。通常の実行ファイルがこのブートローダーで読み込まれた場合、アプリケーション リセット ベクタによりブートローダーのリセット ベクタが上書きされます。その場合、プロセッサのリセット時にブートローダーが実行されず (通常は最初に実行されることが必要)、このアプリケーションの再読み込みおよびその他必要な初期化が行われません。

この状況を回避するため、このコンパイラ オプションを使用する必要があります。このコンパイラ オプションを使用すると、プロセッサのリセット時に、アプリケーションではなくブートローダーに到達します。上記とは異なる状況で使用されるアプリケーションでは、このオプションは機能しません。このモードでは、`crt2.o` がスタートアップ ファイルとして使用されます。

### **-xl-mode-novectors**

MicroBlaze ベクタを必要としないアプリケーションをコンパイルする際に使用します。通常は、プロセッサのリセット、割り込み、または例外機能を使用しないスタンドアロン アプリケーションで使用されます。このオプションを使用すると、ベクタの命令が含まれないので、コード サイズが小さくなります。このモードでは、`crt3.o` がスタートアップ ファイルとして使用されます。

**注意** コマンド ラインで、複数の実行モードを指定しないでください。複数のモードを指定すると、複数のシンボル定義が原因でリンク エラーが発生します。

## 位置独立コード (PIC)

MicroBlaze 用の GNU コンパイラでは、`-fPIC` および `-fpic` オプションがサポートされています。これらのオプションを使用すると、コンパイラで位置独立コード (PIC) を生成できます。この機能は、共有ライブラリおよび再配置可能実行ファイルをインプリメントするために **Linux** 上で **MicroBlaze** に対してのみ使用されます。生成されたコードのデータ アクセスをすべて再配置するにはグローバル オフセット テーブル (GOT) が使用され、共有ライブラリへの関数呼び出しにはプロシージャ リンケージ テーブル (PLT) が使用されます。これは、GNU ベースのプラットフォームで再配置可能コードを生成し、共有ライブラリをリンクする際の標準的な方法です。

## MicroBlaze アプリケーション バイナリ インターフェイス

MicroBlaze 用の GNU コンパイラでは、『MicroBlaze プロセッサ リファレンス ガイド』で定義されるアプリケーション バイナリ インターフェイス (ABI) が使用されます。レジスタおよびスタックの使用法に関する規則、コンパイラで使用する標準メモリ モデルの説明は、ABI の資料を参照してください。100 ページの「関連リソース」に、このマニュアルへのリンクがあります。

## MicroBlaze アセンブラ

ザイリンクス MicroBlaze ソフト プロセッサ用の `mb-as` アセンブラでは、標準 GNU コンパイラでサポートされているオプションおよび標準 GNU アセンブラでサポートされているアセンブラ指示子がサポートされています。

`mb-as` アセンブラでは、`imm` 命令以外の MicroBlaze マシン命令セットの `opcode` がサポートされています。`imm` 命令は、大きい即値が使用される場合に生成されます。`imm` 命令を含むコードを記述するためのアセンブリ言語プログラムは必要ありません。MicroBlaze の命令セットの詳細は、『MicroBlaze プロセッサ リファレンス ガイド』を参照してください。100 ページの「関連リソース」に、このマニュアルへのリンクがあります。



mb-as アセンブラでは、即値オペランドを使用するすべての MicroBlaze 命令を定数またはラベルとして指定する必要があります。命令に PC 相対オペランドが必要な場合は、mb-as アセンブラによりそれが算出され、必要に応じて imm 命令に含められます。たとえば、beqi (Branch Immediate if Equal) 命令には PC 相対オペランドが必要です。

アセンブリ プログラムでは、この命令を次のように使用します。

```
beqi r3, mytargetlabel
```

ここで、mytargetlabel は対象となる命令のラベルです。mb-as アセンブラにより、命令の即値が mytargetlabel - PC として算出されます。

即値が 16 ビットより大きい場合は、imm 命令が自動的に挿入されます。コンパイル時に mytargetlabel の値が不明な場合は、常に imm 命令が挿入されます。不要な imm 命令を削除するには、relax オプションを使用してください。

同様に、命令で大きな定数のオペランドが必要な場合は、アセンブリ言語プログラムで imm 命令ではなくオペランドをそのまま使用する必要があります。たとえば次のコードでは、レジスタ r3 の内容に定数 200,000 を追加し、結果をレジスタ r4 に保存します。

```
addi r4, r3, 200000
```

mb-as アセンブラは、このオペランドに imm 命令が必要であると判断し、自動的に挿入します。

mb-as アセンブラでは、アセンブリのプログラムを簡単にするため、標準の MicroBlaze 命令セットに加えていくつかの擬似 opcode がサポートされています。表 9-6 に、サポートされる擬似 opcode を示します。

表 9-6 : GNU アセンブラでサポートされる擬似 Opcode

擬似 Opcode	説明
<b>nop</b>	処理は実行されません。次の命令に置換されます。 <b>or</b> R0, R0, R0
<b>la</b> Rd, Ra, Imm	次の命令に置換されます。 <b>addik</b> Rd, Ra, imm; = Rd = Ra + Imm;
<b>not</b> Rd, Ra	次の命令に置換されます。 <b>xori</b> Rd, Ra, -1
<b>neg</b> Rd, Ra	次の命令に置換されます。 <b>rsub</b> Rd, Ra, R0
<b>sub</b> Rd, Ra, Rb	次の命令に置換されます。 <b>rsub</b> Rd, Rb, Ra

## MicroBlaze リンカ オプション

MicroBlaze ソフト プロセッサ用の `mb-ld` リンカでは、GNU コンパイラでサポートされているオプションに加え、追加のオプションも導入されています。このセクションでは、これらのオプションについて説明します。

### **-defsym \_TEXT\_START\_ADDR=value**

デフォルトでは、出力コードのテキスト セクションはベース アドレス 0x28 (XMDStub モードでは 0x800) から開始しますが、このオプションを使用すると、このデフォルトを変更できます。`mb-gcc` コンパイラの実行時にこのオプションを指定すると、出力コードのテキスト セクションは *value* で指定したアドレスから開始するようになります。

デフォルトの開始アドレスを使用する場合は、`-defsym _TEXT_START_ADDR` を設定する必要はありません。

これはリンカ オプションであり、リンカを個別に実行する場合に使用します。リンカを `mb-gcc` の一部として実行する場合は、次のオプションを使用してください。

```
-Wl,-defsym -Wl,_TEXT_START_ADDR=value
```

### **-relax**

アセンブラで生成された不要な `imm` 命令を削除するリンカ オプションです。アセンブラでは、即値が算出できない命令があると、`imm` 命令が生成されます。

ほとんどの場合、`imm` 命令は必要ありません。`-relax` オプションを使用すると、リンカにより不要な `imm` 命令が削除されます。

このオプションは、リンカを個別に実行した場合にのみ必要です。リンカを `mb-gcc` の一部として実行する場合は、このオプションは自動的に指定されます。

### **-N**

テキストおよびデータ セクションを読み出し/書き込み可能にします。データ セグメントはページ揃えされません。このオプションは、MicroBlaze プログラムにのみ必要です。リンカを GCC コンパイラの一部として実行する場合は、このオプションは自動的に指定されます。リンカを個別に実行する場合は、このオプションを指定する必要があります。

このオプションの詳細は、GNU のマニュアルを参照してください。100 ページの「関連リソース」に、このマニュアルへのリンクがあります。

MicroBlaze リンカでは、リンカ スクリプトを使用して次にリストするセクションをメモリに割り当てます。

## MicroBlaze リンカ スクリプトで割り当てられるセクション

表 9-7 に、MicroBlaze リンカ スクリプトで割り当てられる入力セクションを示します。

表 9-7 : MicroBlaze リンカ スクリプトで割り当てられるセクション

セクション名	説明
<code>.vectors.reset</code>	リセット ベクタ コード
<code>.vectors.sw_exception</code>	ソフトウェア例外ベクタ コード
<code>.vectors.interrupt</code>	ハードウェア割り込みベクタ コード
<code>.vectors.hw_exception</code>	ハードウェア例外ベクタ コード
<code>.text</code>	関数のコードおよびグローバル アセンブリ文からのプログラム命令
<code>.rodata</code>	読み出し専用変数
<code>.sdata2</code>	初期値を持つ読み出し専用の小さいスタティックおよびグローバル変数
<code>.data</code>	初期値を持つスタティックおよびグローバル変数 (ブートコードによりゼロに初期化)
<code>.sdata</code>	初期値を持つ小さいスタティックおよびグローバル変数
<code>.sbss2</code>	初期値のない読み出し専用の小さいスタティックおよびグローバル変数 (ブートコードによりゼロに初期化)
<code>.sbss</code>	初期値のない小さいスタティックおよびグローバル変数 (ブートコードによりゼロに初期化)
<code>.bss</code>	初期値のないスタティックおよびグローバル変数 (ブートコードによりゼロに初期化)
<code>.heap</code>	ヒープ用に定義されたメモリのセクション
<code>.stack</code>	スタック用に定義されたメモリのセクション

## リンカ スクリプトを記述またはカスタマイズする際のヒント

カスタム リンカ スクリプトを記述する場合は、次の事項を考慮する必要があります。

- ベクタ セクションが MicroBlaze ハードウェアで定義された適切なメモリに割り当てられていることを確認します。
- スタックおよびヒープは `.bss` セクションに配置します。`_stack` 変数をこの領域の `_STACK_SIZE` の後に設定し、`_heap_start` 変数を `_STACK_SIZE` の後の次のロケーションに設定します。スタックおよびヒープは、ハードウェアおよびシミュレーションで初期化する必要がないので、`_bss_end` 変数は `.bss` および `COMMON` の後に定義します。ただし、`.bss` セクションの境界にはスタックおよびヒープは含まれません。
- `_SDATA_START__`、`_SDATA_END__`、`SDATA2_START`、`_SDATA2_END__`、`_SBSS2_START__`、`_SBSS2_END__`、`_bss_start`、`_bss_end`、`_sbss_start`、および `_sbss_end` 変数は、それぞれ `.sdata`、`.sdata2`、`.sbss2`、`.bss`、`.sbss` セクションの最初と最後に定義する必要があります。

- ANSI C では、初期化されないメモリはすべてスタートアップに初期化する必要があります (スタックおよびヒープでは不要)。EDK に含まれる標準の CRT では、1 つの .bss セクションが 0 に初期化されると想定されます。複数の .bss セクションがある場合は、この CRT は使用できません。その場合は、すべての .bss セクションを初期化する CRT を作成する必要があります。

## スタートアップ ファイル

コンパイラで実行ファイルを生成する際、最後のリンク コマンドにコンパイル済みのスタートアップ ファイルおよびエンド ファイルが含まれます。スタートアップ ファイルは、アプリケーション コードが実行される前に、言語およびプラットフォーム環境を設定します。スタートアップ ファイルでは、通常次の処理が実行されます。

- 必要に応じて、リセット、割り込み、および例外ベクタを設定します。
- スタック ポインタ、スモール データ アンカー、およびその他のレジスタを設定します。詳細は、124 ページの表 9-8 を参照してください。
- BSS メモリ領域を 0 にクリアします。
- C++ コンストラクタなどの言語初期化関数を呼び出します。
- ハードウェア サブシステムを初期化します。たとえば、プログラムのプロファイルが作成される場合は、プロファイル タイマを初期化します。
- main プロシージャの引数を設定し、呼び出します。

エンド ファイルは、プログラムの終わりに実行する必要のあるコードが含まれます。エンド ファイルでは、通常次の処理が実行されます。

- C++ デストラクタなどの言語クリーンアップ関数を呼び出します。
- ハードウェア サブシステムの初期化を解除します。たとえば、プログラムのプロファイルが作成されている場合は、プロファイル サブシステムをクリーンアップします。

表 9-8：C ランタイム ファイルでのレジスタの初期化

レジスタ	値	説明
r1	_stack-16	スタック ポインタ レジスタ。初期の負のオフセットが 16 バイトのスタック領域の下部を指定するよう初期化されます。この 16 バイトは、引数を渡すのに使用できます。
r2	_SDA2_BASE	読み出し専用のスモール データ領域アンカー アドレス。
r13	_SDA_BASE_	読み出し/書き込み可能なスモール データ領域アンカー アドレス。
その他のレジスタ	未定義	その他のレジスタは、定義されていません。

この後、さまざまなアプリケーション モードで使用される初期化ファイルについて説明します。この情報は、アプリケーションのスタートアップ コードを理解または変更する必要がある場合に使用できます。MicroBlaze では、C ランタイム初期化に 2 つの段階があります。最初の段階では主にベクタが設定され、最後に第 2 段階が呼び出されます。また、アプリケーション モードによっては、exit スタブも提供されます。

## 第 1 段階の初期化ファイル

### **crt0.o**

ブートローダーまたは XMDStub などのデバッグ スタブを使用せずに、スタンドアロンで実行されるプログラムに対して使用します。このファイルにより、リセット、割り込み、例外、およびハードウェア例外ベクタを指定し、第 2 段階のスタートアップ ルーチン `_crtinit` を呼び出します。`_crtinit` から戻ると、`_exit` ラベルで無限ループを実行することによりプログラムを終了します。

### **crt1.o**

アプリケーションを XMDStub を使用してデバッグする際に使用します。ブレークポイントおよびリセット ベクタ以外のすべてのベクタを指定し、第 2 段階のスタートアップ ルーチン `_crtinit` を呼び出します。`_crtinit` から戻ると、XMDStub に戻り、プログラムが終了したことを示す信号がデバッガに送信されます。

### **crt2.o**

実行ファイルがブートローダーで読み込まれる場合に使用します。リセット ベクタ以外のすべてのベクタを指定し、第 2 段階のスタートアップ ルーチン `_crtinit` を呼び出します。`_crtinit` から戻ると、`_exit` ラベルで無限ループを実行することによりプログラムを終了します。リセット ベクタは指定されないため、プロセッサのリセット時には、ブートローダーによりプログラムが再度読み込まれ、開始されます。

### **crt3.o**

実行ファイルでベクタを使用せず、コード サイズを小さくする場合に使用します。リセット ベクタのみを指定し、第 2 段階のスタートアップ ルーチン `_crtinit` を呼び出します。`_crtinit` から戻ると、`_exit` ラベルで無限ループを実行することによりプログラムを終了します。ほかのベクタは指定されないため、リンクの段階で割り込み処理および例外処理に関連するルーチンが含まれることはなく、コード サイズが小さくなります。

## 第 2 段階の初期化ファイル

C 標準仕様に従い、すべてのグローバル変数およびスタティック変数を 0 に初期化する必要があります。これは、上記すべての CRT で必要です。このため、別のルーチン `_crtinit` が呼び出されます。このルーチンは、プログラムの .bss セクションのメモリを初期化します。`_crtinit` ルーチンはラップファイルでもあり、`main` プロシージャも呼び出します。`main` プロシージャを呼び出す前に、ほかの初期化関数が呼び出される場合もあります。`_crtinit` ルーチンは、次のスタートアップ ファイルにより提供されます。

### **crtinit.o**

デフォルトの第 2 段階の C スタートアップ ファイルです。次の処理を実行します。

1. .bss セクションを 0 にクリアします。
2. `_program_init` を呼び出します。
3. コンストラクタ関数 (`_init`) を呼び出します。
4. `main` プロシージャの引数を設定し、呼び出します。
5. デストラクタ関数 (`_fini`) を呼び出します。
6. `_program_clean` を呼び出し、戻ります。

### **pgcrtinit.o**

プロファイル作成時に使用します。次の処理を実行します。

1. .bss セクションを 0 にクリアします。
2. `_program_init` を呼び出します。
3. `_profile_init` を呼び出し、プロファイル ライブラリを初期化します。
4. コンストラクタ関数 (`_init`) を呼び出します。
5. `main` プロシージャの引数を設定し、呼び出します。
6. デストラクタ関数 (`_fini`) を呼び出します。
7. `_profile_clean` を呼び出し、プロファイル ライブラリをクリーンアップします。
8. `_program_clean` を呼び出し、戻ります。

### **sim-crtinit.o**

コンパイラで `-mno-clearbss` オプションが設定されている場合に使用します。次の処理を実行します。

1. `_program_init` を呼び出します。
2. コンストラクタ関数 (`_init`) を呼び出します。
3. `main` プロシージャの引数を設定し、呼び出します。
4. デストラクタ関数 (`_fini`) を呼び出します。
5. `_program_clean` を呼び出し、戻ります。

### **sim-pgcrtinit.o**

プロファイルの作成時に、コンパイラで `-mno-clearbss` オプションが設定されている場合に使用します。次の処理を実行します。

1. `_program_init` を呼び出します。
2. `_profile_init` を呼び出し、プロファイル ライブラリを初期化します。
3. コンストラクタ関数 (`_init`) を呼び出します。
4. `main` プロシージャの引数を設定し、呼び出します。
5. デストラクタ関数 (`_fini`) を呼び出します。
6. `_profile_clean` を呼び出し、プロファイル ライブラリをクリーンアップします。
7. `_program_clean` を呼び出し、戻ります。

## **その他のファイル**

コンパイラは、C++ 言語をサポートするため、特定の標準スタート ファイルおよびエンド ファイルも使用します。`crti.o`、`crtbegin.o`、`crtend.o`、および `crtm.o` がそれに当たります。これらの標準コンパイラ ファイルは、`.init`、`.fini`、`.ctors`、および `.dtors` セクションの内容を指定します。

## スタートアップ ファイルの変更

初期化ファイルは、コンパイル済みのファイルおよびソース ファイルの両方で提供されます。コンパイル済みのオブジェクト ファイルは、コンパイラ ライブラリ ディレクトリに含まれます。MicroBlaze GNU コンパイラ用の初期化ファイルのソース ファイルは、<XILINX\_EDK>/sw/lib/microblaze/src ディレクトリにあります (<XILINX\_EDK> は EDK のインストール ディレクトリ)。

カスタム スタートアップ ファイルを使用するには、ファイルをソース エリアから取り出し、アプリケーション ソースの一部として含める必要があります。また、ファイルを .o ファイルに統合して、共有エリアに配置することも可能です。標準ファイルではなく作成したオブジェクト ファイルを参照する場合は、mb-gcc の実行時に -B オプションを使用します。

デフォルトのスタートアップ ファイルが使用されないようにするには、コンパイルの最後の行に -nostartfiles を追加します。

**メモ：** crt0.o、crtbegin.o などのコンパイラ標準 CRT ファイルは、ソース コードでは提供されないため、インストール ディレクトリに含まれているものをそのまま使用してください。これらのファイルは、最後のリンク コマンドに含める必要がある場合があります。

## C プログラムのスタートアップ コード サイズの削減

C プログラムのコード サイズの制限が厳しい場合は、オーバーヘッドの原因となるあらゆるものを取り除く必要があります。このセクションでは、C プログラムで不要な C++ コンストラクタまたはデストラクタ コードによるオーバーヘッドを削減する方法を示します。次の変更を加えることにより、コードのサイズを約 220 バイト削減できます。

1. 前のセクションで説明したように、インストール領域からスタートアップ ファイルのカスタムコピーを作成します。アプリケーションに適した crt0.s および xcrtinit.s をコピーします。たとえば、アプリケーションがブートローダーを使用して読み込まれ、プロファイルが作成される場合は、インストール領域から crt2.s および pg-crtinit.s をコピーします。
2. pg-crtinit.s から次の行を削除します。

```
brlid    r15, __init
/* Invoke language initialization functions */
nop
```

および

```
brlid    r15, __fini
/* Invoke language cleanup functions */
nop
```

これらの行を削除することにより、コンストラクタおよびデストラクタの処理に使用されるコードが参照されなくなり、コード サイズが小さくなります。

3. これらのファイルを .o ファイルにコンパイルして任意のディレクトリに配置するか、アプリケーション ソースの一部として含めます。
4. コンパイラに -nostartfiles オプションを追加します。特定のフォルダにファイルを統合する場合は、-B directory オプションも使用します。
5. アプリケーションをコンパイルします。



アプリケーションを異なるモードで実行する場合は、124 ページの「スタートアップ ファイル」の説明に従って、適切な CRT ファイルを選択する必要があります。

## コンパイラ ライブラリ

mb-gcc コンパイラには、GNU C 標準ライブラリと GNU 数値計算ライブラリが必要です。EDK には、あらかじめコンパイルされたこれらのライブラリが含まれています。Libgen の実行中に、MicroBlaze のハードウェア コンフィギュレーションに基づき、MicroBlaze の CPU ドライバの適切なバージョンがコピーされます。使用するライブラリのバージョンを選択するには、次のフォルダを確認します。

```
$XILINX_EDK/gnu/microblaze/<platform>/microblaze-xilinx-elf/lib
```

ファイル名は、コンパイラのオプションとライブラリのコンパイルに使用されたコンフィギュレーションに基づいて付けられています。たとえば、libc\_m\_bs.a は、ハードウェア乗算器とバレルシフトをオンにしてコンパイルされた C ライブラリです。

次の表に、使用されているエンコードとそのエンコードで指定されるライブラリのコンフィギュレーションを示します。

表 9-9 : コンパイラ フラグ上のエンコードされたライブラリ ファイル名

エンコード	説明
_bs	バレルシフト用にコンフィギュレーション
_m	ハードウェア乗算器用にコンフィギュレーション
_p	パターン コンパレータ用にコンフィギュレーション
_mh	拡張ハードウェア乗算器用にコンフィギュレーション

注意が必要なのは、数値計算ライブラリ ファイル (libm\*.a) です。C 標準では、一般的な数値計算ライブラリ関数 (sin(), cos() など) で倍精度浮動小数点演算を使用する必要がありますが、倍精度浮動小数点演算では MicroBlaze で使用可能な単精度浮動小数点の機能を最大限に活用できない可能性があります。

Newlib 数値計算ライブラリには、単精度演算を使用してこれらの数値計算関数をインプリメントするバージョンがあります。これらの単精度ライブラリでは MicroBlaze ハードウェア浮動小数点ユニットを直接使用できるので、パフォーマンスが向上する場合があります。アプリケーションで標準精度が必要ではなく、パフォーマンスを向上させる場合は、リンクされたライブラリのバージョンを手動で変更できます。デフォルトでは、CPU ドライバにより倍精度バージョンのライブラリ (libm\*\_fpd.a) が XPS プロジェクトにコピーされます。単精度バージョンを使用する場合は、カスタム CPU ドライバを作成し、libm\*\_fps.a ライブラリ ファイルを libm.a としてプロセッサのライブラリ フォルダ (microblaze\_0/lib など) にコピーします。

使用するライブラリをコピーしたら、アプリケーション ソフトウェア プロジェクトを再構築します。

## スレッド セーフ

EDK に含まれる MicroBlaze の C ライブラリおよび数値演算ライブラリは、マルチスレッド環境で使用するようには構築されていません。printf(), scanf(), malloc(), free() などの共通 C ライブラリ関数はスレッド セーフではなく、システムで回復不可能なランタイム エラーを引き起こすことがあります。マルチスレッド環境で EK ライブラリを使用する場合は、相互排他的なメカニズムを使用してください。



## コマンド ライン引数

MicroBlaze プログラムでは、コマンド ライン引数を使用できません。コマンド ライン引数 `argc` および `argv` は、C ランタイム ルーチンで 0 に初期化されます。

## 割り込みハンドラ

割り込みハンドラは、通常のサブルーチン呼び出しとは別の方法でコンパイルされます。割り込みハンドラでは、不揮発性レジスタだけでなく、使用されている揮発性レジスタも保存する必要があります。また、割り込みが発生した際に、マシン ステータス レジスタ (RMSR) の値も保存する必要があります。

### interrupt\_handler 属性

サブルーチンと割り込みハンドラを識別するため、`mb-gcc` ではコードの宣言部に `interrupt_handler` 属性があるかどうかチェックされます。この属性は、次のように定義されます。

```
void function_name () __attribute__((interrupt_handler));
```

**メモ：**割り込みハンドラの属性は、プロトタイプ内でのみ指定し、定義には含めません。

割り込みハンドラで、揮発性レジスタを使用するほかの関数が呼び出される場合があります。揮発性レジスタで正しい値を保持するため、ハンドラが非リーフ関数である場合は、すべての揮発性レジスタが保存されます。

**メモ：**非リーフ関数とは、ほかのサブルーチンを呼び出す関数のことです。

割り込みハンドラは、MHS (Microprocessor Hardware Specification) ファイルおよび MSS (Microprocessor Software Specification) ファイルで定義されます。これらの定義により、割り込みハンドラ関数に属性が追加されます。詳細は、付録 B の「割り込み制御」を参照してください。

割り込みハンドラは、`rtid` 命令を使用して割り込みで中断された関数に戻ります。

### save\_volatiles 属性

`save_volatiles` 属性は、`interrupt_handler` 属性と似ていますが、割り込み処理から戻るのに `rtid` ではなく `rtsd` を使用します。

この属性を使用すると、非リーフ関数の場合はすべての揮発性レジスタが保存され、リーフ関数の場合は使用された揮発性レジスタのみが保存されます。

```
void function_name () __attribute__((save_volatiles));
```

上記 2 つの属性とその機能を表 9-10 に示します。

表 9-10：割り込みハンドラの属性

属性	機能
<code>interrupt_handler</code>	マシン ステータス レジスタ、不揮発性レジスタ、およびすべての揮発性レジスタを保存します。割り込みハンドラから戻るには、 <code>rtid</code> を使用します。割り込みハンドラ関数がリーフ関数の場合は、関数で使用された揮発性レジスタのみが保存されます。
<code>save_volatiles</code>	<code>interrupt_handler</code> と似ていますが、割り込みから戻るのに <code>rtid</code> ではなく <code>rtsd</code> が使用されます。

## PowerPC コンパイラの使用法とオプション

### PowerPC コンパイラ オプションの一覧

#### PowerPC コンパイラ オプション

```
-mcpu=440  
-mfpu={sp_lite, sp_full, dp_lite, dp_full, none}  
-mppcperflib  
-mno-clearbss
```

#### PowerPC リンカ オプション

```
-defsym _START_ADDR=value
```

### PowerPC コンパイラ オプション

PowerPC プロセッサ GNU コンパイラ (powerpc-eabi-gcc) は、GNU より PowerPC プロセッサのポート用に提供されているソースから構築したもので、ザイリンクスのデバイス用に変更が加えられています。EDK に含まれる PowerPC プロセッサ コンパイラに特定の機能およびオプションを、次に説明します。PowerPC プロセッサ コンパイラでコンパイルする場合、プリプロセッサで自動的に `__PPC__` 定義が使用されます。この定義は、どのような条件コードでも使用できます。

#### **-mcpu=440**

440 プロセッサのターゲット コードです。これには、命令スケジュール最適化、命令回避策のインーブル/ディスエーブル、440 プロセッサをターゲットとしたライブラリの使用が含まれます。

#### **-mfpu={sp\_lite, sp\_full, dp\_lite, dp\_full, none}**

ザイリンクス PowerPC プロセッサ APU FPU コプロセッサ ハードウェアで使用するハードウェア浮動小数点命令を生成します。命令およびコード出力は、APU FPU ハードウェア用に変更された部分を除き、PowerPC Book-E の浮動小数点仕様に準拠しています。Book-E は、IBM の Web サイト から参照できます。アーキテクチャの詳細は、FPU ハードウェアのマニュアルを参照してください。100 ページの「関連リソース」に、Book-E および FPU のマニュアルへのリンクがあります。

-mfpu= は、FPU ハードウェアのタイプを指定します。次のタイプがあります。

- **sp\_lite**

単精度 Lite FPU コプロセッサ用のコードを生成します。単精度のハードウェア浮動小数点のみがサポートされ、ハードウェア除算命令および平方根命令は使用されません。このオプションを使用すると、C プリプロセッサ定義 `HAVE_XFPU_SP_LITE` が自動的に定義されます。

- **sp\_full**

単精度 Full FPU コプロセッサ用のコードを生成します。単精度のハードウェア浮動小数点のみがサポートされ、ハードウェア除算命令および平方根命令が使用されます。このオプションを使用すると、C プリプロセッサ定義 `HAVE_XFPU_SP_FULL` が自動的に定義されます。

- **dp\_lite**

倍精度 Lite FPU コプロセッサ用のコードを生成します。単精度と倍精度のハードウェア浮動小数点の両方がサポートされ、ハードウェア除算命令および平方根命令は使用されません。このオプションを使用すると、C プリプロセッサ定義 `HAVE_XFPU_DP_LITE` が自動的に定義されます。

- **dp\_full**

倍精度 Full FPU コプロセッサ用のコードを生成します。単精度と倍精度のハードウェア浮動小数点の両方がサポートされ、ハードウェア除算命令および平方根命令が使用されます。このオプションを使用すると、C プリプロセッサ定義 `HAVE_XFPU_DP_FULL` が自動的に定義されます。

**注意** -mfpu オプションのあるタイプで生成したコードを、-mfpu の別のタイプで (または -mfpu オプションを使用せずに) 生成したコードとリンクしないでください。オブジェクト ファイルをリンクするだけであっても、このオプションを使用する必要があります。このオプションを使用することにより、正しいライブラリが使用され、互換性の問題を回避できます。

- **none**

浮動小数点演算に対してソフトウェア エミュレーションを使用するよう指定します。このオプションがデフォルトです。

ハードウェア浮動小数点コプロセッサを最適に使用するための詳細は、APU FPU のユーザー ガイドを参照してください。100 ページの「[関連リソース](#)」に、このガイドへのリンクがあります。

### **-mppcperflib**

低レベルの整数と浮動小数点のエミュレーション用 PowerPC プロセッサ パフォーマンス ライブラリ、および単純な文字列ルーチンを使用します。これらのライブラリは、GCC で提供されるデフォルトのエミュレーション ルーチンおよび Newlib で提供される単純な文字列ルーチンの代わりに使用されます。パフォーマンス ライブラリを使用すると、これらのルーチンを頻繁に使用するアプリケーションで平均的にスピードが 3 倍になります。詳細は、次の SourceForge プロジェクト Web ページを参照してください。100 ページの「[関連リソース](#)」に、この Web ページへのリンクがあります。

**注意** -mfpu オプションを使用している場合は、パフォーマンス ライブラリは使用できません。

### **-mno-clearbss**

このオプションは、シミュレーションで使用するプログラムをコンパイルする際に有益です。C 言語の標準に基づき、初期化されていないグローバル変数は .bss セクションに割り当てられ、プログラムの実行が開始したときの値は 0 になります。通常これは、プログラムの実行が開始したときに、C スタートアップ ファイルをループで実行して .bss セクションに 0 が記述されるようにすることで達成します。さらにコンパイラを最適化すると、C コードで 0 に割り当てられたグローバル変数が .bss セクションに割り当てられます。

シミュレーション環境では、2 言語機能は余分なオーバーヘッドとなる場合があります。シミュレータによっては、メモリ全体を自動的に 0 にするものもあります。通常的环境でも、グローバル変数が開始時に 0 になっていなくても良いような C コードを記述できます。そのような場合に、このオプションは有益です。このオプションを使用すると、C スタートアップ ファイルにより .bss セクションが 0 に初期化されることはなくなります。また、.bss セクションに 0 に初期化されたグローバル変数は割り当てられず、これらの変数は .data セクションに移動されます。このオプションにより、アプリケーションの起動時間が短縮される場合があります。このオプションは、グローバル変数が 0 に初期化することが必要なコードを使用していない場合、またはシミュレーション プラットフォームでメモリが 0 に初期化される場合にのみ使用してください。

## PowerPC プロセッサ リンカ オプション

PowerPC プロセッサ用の `powerpc-eabi-ld` リンカでは、GNU コンパイラでサポートされているオプションに加え、新しいオプションも導入されています。これらのオプションを次に説明します。

### **-defsym \_START\_ADDR=value**

デフォルトでは、出力コードのテキスト部分はデフォルトのリンカ スクリプトで指定されているベース アドレス `0xffff0000` から開始しますが、上記のオプションを使用するか、リンカ スクリプトで開始アドレスの値を指定することにより変更できます。

コンパイラで設定されるデフォルトの開始アドレスを使用する場合は、`-defsym _START_ADDR` を設定する必要はありません。これはリンカ オプションであり、リンカを個別に実行する場合に使用します。 リンカを `powerpc-eabi-gcc` の一部として実行する場合は、`-Wl,-defsym _START_ADDR=value` オプションを使用してください。

PowerPC リンカでは、リンカ スクリプトを使用して次にリストするセクションをメモリに割り当てます。

## PowerPC プロセッサ リンカ スクリプトで割り当てられるセクション

次の表に、PowerPC プロセッサ リンカ スクリプトで割り当てられる入力セクションを示します。

表 9-11 : PowerPC リンカ スクリプトで割り当てられるセクション

セクション名	説明
<code>.boot</code>	<code>.boot0</code> に初期分岐するプロセッサ リセット ベクタ コード
<code>.boot0</code>	ブート コード
<code>.heap</code>	ヒープ用に定義されたメモリのセクション
<code>.stack</code>	スタック用に定義されたメモリのセクション
<code>.bss</code>	初期値のないスタティックおよびグローバル変数 (ブート コードによりゼロに初期化)
<code>.sbss</code>	初期値のない小さいスタティックおよびグローバル変数 (ブート コードによりゼロに初期化)
<code>.sbss2</code>	初期値を持つ読み出し専用の小さいスタティックおよびグローバル変数 (ブート コードによりゼロに初期化)
<code>.sdata</code>	初期値を持つ小さいスタティックおよびグローバル変数
<code>.data</code>	初期値を持つスタティックおよびグローバル変数 (ブート コードによりゼロに初期化)
<code>.sdata2</code>	初期値を持つ読み出し専用の小さいスタティックおよびグローバル変数
<code>.rodata</code>	読み出し専用変数
<code>.text</code>	関数のコードおよびグローバル アセンブリ文からのプログラム命令
<code>.got2</code>	グローバル オフセット テーブル (GOT)。位置独立コードからグローバル データにアクセスする場所を定義します。

表 9-11 : PowerPC リンカ スクリプトで割り当てられるセクション (続き)

セクション名	説明
.got1	グローバル オフセット テーブル (GOT)。位置独立コードからグローバル データにアクセスする場所を定義します。
.fixup	フィックスアップ情報 (フィックスアップ レコード テーブル など)
.jcr	コンパイラ専用セクション。コンパイラ初期化関数で使用されます。
.gcc_except_table	言語特定データ
.tdata	初期化済みスレッド ローカル データ
.tbss	初期化されていないスレッド ローカル データ

## リンカ スクリプトを記述またはカスタマイズする際のヒント

カスタム リンカ スクリプトを記述する場合は、次の事項を考慮する必要があります。

- PowerPC プロセッサのリンカには、デフォルトのリンカ スクリプトが組み込まれています。このスクリプトでは、アドレス 0xFFFF0000 から連続したメモリ空間に配置されます。Libgen により作成される libxil.a ライブラリに含まれる boot.o が、最初にリンクするファイルとして定義されています。デフォルトの開始アドレス 0xFFFF0000 を変更するには、コマンド ライン オプションを使用するか、リンカ スクリプトを変更します。
- .boot セクションが 0xFFFFFFF0 から開始していることを確認します。PowerPC プロセッサは、電源投入時に 0xFFFFFFF0 から実行されます。
- \_end 変数は .boot0 セクションの後に定義します。このセクションは、.boot0 セクションの開始にジャンプします。ジャンプは 24 ビットに定義されているので、.boot と .boot0 セクションには 25 ビット以上の差があってはなりません。PowerPC 440 プロセッサでは、.boot0 セクションは 0xFFFFFFF0 に固定されています。
- スタックおよびヒープは .bss セクションに配置します。
- \_stack 変数をこの領域の \_STACK\_SIZE の後に設定し、\_heap\_start 変数を \_STACK\_SIZE の後の次のロケーションに設定します。スタックおよびヒープは、ハードウェアおよびシミュレーションで初期化する必要がないので、\_bss\_end 変数は .bss および COMMON の後に定義します。ただし、.bss セクションの境界にはスタックおよびヒープは含まれません。
- \_SDATA\_START\_\_、\_SDATA\_END\_\_、\_SDATA2\_START、\_SDATA2\_END\_\_、\_SBSS2\_START\_\_、\_SBSS2\_END\_\_、\_bss\_start、\_bss\_end、\_sbss\_start、および \_sbss\_end 変数は、それぞれ .sdata、.sdata2、.sbss2、.bss、.sbss セクションの最初と最後に定義する必要があります。
- PowerPC 405 では、.vectors セクションを 64K 境界に揃えます。PowerPC 440 では、特定の位置に揃える必要はありません。このセクションの定義は、プログラムで割り込みまたは例外を使用する場合にのみ含めるようにしてください。
- メモリの各物理領域で、異なるプログラム ヘッダを使用します。メモリの不連続な 2 つの領域に対し、同じプログラム ヘッダを使用することはできません。
- ANSI C では、初期化されないメモリはすべてスタートアップに初期化する必要があります (スタックおよびヒープでは不要)。EDK に含まれる標準の CRT では、1 つの .bss セクションが 0 に初期化されると想定されます。複数の .bss セクションがある場合は、この CRT は使用できません。その場合は、すべての .bss セクションを初期化する CRT を作成する必要があります。

## スタートアップ ファイル

コンパイラで実行ファイルを生成する際、最後のリンク コマンドにコンパイル済みのスタートアップ ファイルおよびエンド ファイルが含まれます。スタートアップ ファイルは、アプリケーション コードが実行される前に、言語およびプラットフォーム環境を設定します。スタートアップ ファイルでは、通常次の処理が実行されます。

- 必要に応じて、リセット、割り込み、および例外ベクタを設定します。
- スタック ポインタ、スモール データ アンカー、およびその他のレジスタを設定します。
- BSS メモリ領域を 0 にクリアします。
- C++ コンストラクタなどの言語初期化関数を呼び出します。
- ハードウェア サブシステムを初期化します。たとえば、プログラムのプロファイルが作成される場合は、プロファイル タイマを初期化します。
- main プロシージャの引数を設定し、呼び出します。

エンド ファイルは、プログラムの終わりに実行する必要があるコードが含まれます。エンド ファイルでは、通常次の処理が実行されます。

- C++ デストラクタなどの言語クリーンアップ関数を呼び出します。
- ハードウェア サブシステムをクリーンアップします。たとえば、プログラムのプロファイルが作成されている場合は、プロファイル サブシステムをクリーンアップします。

表 9-12 : C ランタイム ファイルでのレジスタの初期化

レジスタ	値	説明
r1	_stack-8	スタック ポインタ レジスタ。割り当てられたスタックの下部を初期化します。オフセットは 16 バイトです。この 16 バイトは、引数を渡すのに使用されます。
r2	_SDA2_BASE	読み出し専用のスモール データ領域アンカー アドレス。
r13	_SDA_BASE_	読み出し/書き込み可能なスモール データ領域アンカー アドレス。
その他のレジスタ	未定義	その他のレジスタは、定義されていません。

この後、初期化ファイルについて説明します。この情報は、アプリケーションのスタートアップ コードを理解または変更する必要がある場合に使用できます。

### 初期化ファイル

PowerPC プロセッサ コンパイラでは、xil-crt0.o、xil-pgcrt0.o、xil-sim-crt0.o、および xil-sim-pgcrt0.o の 4 つの CRT ファイルを使用します。これらの CRT ファイルでは、次の処理が実行されます。

1. 関数 `_cpu_init` を呼び出します。この関数は、ボード サポート パッケージ ライブラリに含まれており、プロセッサ アーキテクチャ特定の初期化を実行します。
2. .bss メモリ領域を 0 にクリアします。
3. レジスタを設定します。詳細は、表 9-12 を参照してください。
4. タイマ ベース レジスタを 0 に初期化します。
5. MSR の浮動小数点ユニット ビットを有効にします (オプション)。

6. C++ 言語およびコンストラクタ初期化関数 (`_init`) を呼び出します。
7. `main` を呼び出します。
8. C++ 言語デストラクタ (`_fini`) を呼び出します。
9. `exit` を実行します。

## スタートアップ ファイルの説明

### **xil-crt0.o**

特別な要件なしで、スタンドアロンで実行されるプログラムに対して使用されるデフォルトの初期化ファイルです。前述の共通する処理をすべて実行します。

### **xil-pgcrt0.o**

アプリケーションのプロファイルを作成する際に、プロファイル ソフトウェア コードをプログラムに含める場合に使用します。CRT の共通の処理に加え、`main` を呼び出す前に `_profile_init` ルーチンも呼び出します。これにより、コードが実行される前にソフトウェア プロファイル ライブラリが初期化されます。同様に、`main` が終了したときに、プロファイル ライブラリをクリーンアップする `_profile_clean` ルーチンを呼び出します。

### **xil-sim-crt0.o**

アプリケーションを `-mno-clearbss` を使用してコンパイルする際に使用します。`.bss` セクションを 0 にクリアする以外の CRT の共通処理が実行されます。

### **xil-sim-pgcrt0.o**

アプリケーションを `-mno-clearbss` を使用してコンパイルする際に使用します。`.bss` セクションを 0 にクリアする以外の CRT の共通処理が実行されます。また、`main` を呼び出す前に `_profile_init` ルーチンを呼び出します。これにより、コードが実行される前にソフトウェア プロファイル ライブラリが初期化されます。同様に、`main` が終了したときに、プロファイル ライブラリをクリーンアップする `_profile_clean` ルーチンを呼び出します。

## その他のファイル

コンパイラは、C++ 言語をサポートするため、特定の標準スタート ファイルおよびエンド ファイルも使用します。`ecrti.o`、`crtbegin.o`、`crtend.o`、および `crti.o` がそれに当たります。これらの標準コンパイラ ファイルは、`.init`、`.fini`、`.ctors`、および `.dtors` セクションの内容を指定します。また、PowerPC のデフォルトの生成されたリンカ スクリプトは、`boot.o` をスタートアップ ファイルにします。このファイルは、PowerPC (405 および 440) プロセッサのスタンドアロン パッケージに含まれます。



## スタートアップ ファイルの変更

初期化ファイルは、コンパイル済みのファイルおよびソース ファイルの両方で提供されます。コンパイル済みのオブジェクト ファイルは、コンパイラ ライブラリ ディレクトリに含まれます。PowerPC コンパイラ用の初期化ファイルのソース ファイルは、<XILINX\_EDK>/sw/lib/ppc405/src ディレクトリにあります (<XILINX\_EDK> は EDK のインストール ディレクトリ)。

カスタム スタートアップ ファイルを使用するには、ファイルをソース エリアから取り出し、アプリケーション ソースの一部として含める必要があります。また、ファイルを .o ファイルに統合して、共有エリアに配置することも可能です。標準ファイルではなく作成したオブジェクト ファイルを参照する場合は、powerpc-eabi-gcc の実行時に -B オプションを使用します。デフォルトのスタートアップ ファイルが使用されないようにするには、コンパイルの最後の行に -nostartfiles を追加します。ecrti.o、crtbegin.o などの C++ サポート用のコンパイラ標準 CRT ファイルは、ソース コードでは提供されません。これらのファイルは、インストール ディレクトリに含まれているものをそのまま使用してください。コードでコンストラクタおよびデストラクタを使用する場合は、ファイルを最後のリンク コマンドに含める必要があります。

## C プログラムのスタートアップ コード サイズの削減

C プログラムのコード サイズの制限が厳しい場合は、オーバーヘッドの原因となるあらゆるものを取り除く必要があります。このセクションでは、C プログラムで不要な C++ コンストラクタまたはデストラクタ コードによるオーバーヘッドを削減する方法を示します。次の変更を加えることにより、コードのサイズを約 500 バイト削減できます。

1. 前のセクションで説明するように、インストール領域からスタートアップ ファイルのカスタムコピーを作成します。アプリケーションに適した xil-crt.s をコピーします。たとえば、アプリケーションのプロファイルが作成される場合は、インストール領域から xil-pgcrt0.s をコピーします。

CRT ファイルから次の行を削除します。

```
/* Call _init */
bl    _init

および

/* Invoke the language cleanup functions */
bl    _fini
```

これにより、コンストラクタおよびデストラクタの処理に使用されるコードは参照されなくなり、コード サイズが小さくなります。

2. これらのファイルを .o ファイルにコンパイルして任意のディレクトリに配置するか、アプリケーション ソースの一部として含めます。
3. コンパイラに -nostartfiles オプションを追加します。特定のフォルダにファイルを統合する場合は、-B directory オプションも使用します。
4. アプリケーションをコンパイルします。



## アプリケーションをブートローダーで読み込む場合のスタートアップ ファイルの変更

アプリケーションをブートローダーで読み込む場合、ブートローダーのプロセッサ リセット ベクタをアプリケーションのリセット ベクタで上書きしないようにする方が望ましい場合があります。このようにすると、アプリケーションではなくプロセッサ リセット上のブートローダーが再実行されます。これには、アプリケーションで `boot.o` をスタートアップ ファイルとして呼び出さないようにする必要があります。ほかのコンパイラ スタートアップ ファイルと異なり、`boot.o` はコンパイラにより明示的にリンクされているわけではなく、デフォルトのリンカ スクリプトおよびリンカ スクリプトを生成するツールにより `boot.o` がスタートアップ ファイルとして指定されているので、リンカ スクリプトから `STARTUP` 指示子を削除する必要があります。また、`ENTRY` 指示子を `_boot` から `_start` に変更します。

## コンパイラ ライブラリ

`powerpc-eabi-gcc` コンパイラには、GNU C 標準ライブラリと GNU 数値計算ライブラリが必要です。

EDK には、あらかじめコンパイルされたこれらのライブラリが含まれています。これらのライブラリは、`$XILINX_EDK/gnu/powerpc-eabi/platform/powerpc-eabi/lib` にあります。

この最上位ライブラリ ディレクトリの下さまざまなサブディレクトリには、特定のコンフィギュレーション用のカスタマイズされたライブラリが含まれます。たとえば、`/double` サブディレクトリには倍精度 FPU で使用されるライブラリのバージョンが含まれ、`/440` サブディレクトリには PowerPC 440 プロセッサで使用するのに適したライブラリのバージョンが含まれます。

## スレッド セーフ

EDK に含まれる PowerPC プロセッサの C ライブラリおよび数値演算ライブラリは、マルチスレッド環境で使用するようには構築されていません。`printf()`、`scanf()`、`malloc()`、`free()` などの共通 C ライブラリ関数はスレッド セーフではなく、システムで回復不可能なランタイム エラーを引き起こすことがあります。マルチスレッド環境で EDK ライブラリを使用する場合は、相互排他的なメカニズムを使用してください。

## コマンド ライン引数

PowerPC プロセッサ プログラムでは、コマンド ライン引数を取り込むことはできません。コマンド ライン引数 `argc` および `argv` は、C ランタイム ルーチンにより 0 に初期化されます。

## その他のメモ

### C++ コードのサイズ

最新のオープン ソース C++ 標準ライブラリ (libstdc++-v3) を含む GCC ツールでは、同等の C プログラムに比べて生成されるコードおよびデータ片が大きくなる場合があります。このオーバーヘッドの大部分は、例外処理およびランタイム型情報のコードおよびデータによるものです。C++ アプリケーションによっては、これらの機能は必要ありません。

このオーバーヘッドをなくし、コード サイズを最適化するには、`-fno-exceptions` および `-fno-rtti` オプションを使用します。これらのオプションの使用は、アプリケーションの要件およびこれらの言語の機能に精通している場合のみにすることをお勧めします。使用可能なコンパイラ オプションおよびそれらのオプションによる影響については、GCC のマニュアルを参照してください。

C++ プログラムには、より複雑な言語機能およびライブラリ ルーチンのため、ダイナミック メモリの要件 (スタックおよびヒープ サイズ) が厳しいものがあります。

多くの C++ ライブラリ ルーチンでは、ヒープからのメモリの割り当てが要求されます。C++ プログラムで必要なヒープ サイズおよびスタック サイズが満たされていることを確認してください。

### C++ 標準ライブラリ

C++ 標準ライブラリは、C++ 標準により定義されます。これらのプラットフォーム機能には、デフォルトのザイリンクス EDK ソフトウェア プラットフォームでは使用できないものもあります。たとえば、ファイル I/O は明確に定義された STDIN/STDOUT ストリームでのみサポートされます。また、ロケール関数、スレッド セーフなどの機能もサポートされません。

**メモ：**C++ 標準ライブラリは、マルチスレッド環境で使用するようには構築されていません。`new`、`delete` などの共通 C++ 関数はスレッド セーフではありません。マルチスレッド環境で C++ 標準ライブラリを使用する場合は、注意が必要です。

GNU C++ 標準ライブラリの詳細は、GNU の Web サイトのマニュアルを参照してください。100 ページの「[関連リソース](#)」に、このマニュアルへのリンクがあります。

### 位置独立コード (PIC)

MicroBlaze および PowerPC プロセッサ コンパイラでは、再配置可能な位置独立コードを生成する `-fPIC` オプションがサポートされています。PowerPC コンパイラでは、位置独立コードとは多少異なるコードを生成する `-mrelocatable` オプションもサポートされています。

これらの機能はザイリンクスのコンパイラでサポートされていますが、EDK ではスタンドアロン プラットフォームしか提供されないため、ほかのライブラリおよびツールではサポートされません。位置独立コードは、ローダーまたはデバッガでは認識されず、ランタイムでの再配置は実行されません。これらのコードの機能は、ザイリンクス ライブラリ、スタートアップ ファイル、およびその他のツールでサポートされません。サードパーティ OS ベンダーのディストリビューションおよびツールでは、これらの機能を標準で使用できるものがあります。

### その他のオプションおよび機能

`-fprofile-arcs` など、その他のオプションおよび機能は、ザイリンクスの EDK コンパイラおよびプラットフォームでサポートされていない可能性があります。一部の機能は、オープン ソース GCC で定義されているように試験段階であり、不適切に使用すると、不正なコードが生成されることがあります。特定の機能の詳細は、GCC のマニュアルを参照してください。100 ページの「[関連リソース](#)」に、このマニュアルへのリンクがあります。

# GNU デバッガ (GDB)

---

この章では、MicroBlaze™ プロセッサおよび PowerPC® (405 および 440) プロセッサ用のザイリンクス GNU デバッガ (GDB) の一般的な使用法について説明します。この章には、次のセクションが含まれています。

- [概要](#)
- [関連リソース](#)
- [MicroBlaze GDB のターゲット](#)
- [PowerPC 405 のデバッグ](#)
- [PowerPC 440 のデバッグ](#)
- [コンソール モード](#)
- [GDB コマンドに関するリファレンス情報](#)

## 概要

GDB は、さまざまな開発段階で MicroBlaze および PowerPC (405 および 440) システムをデバッグ/検証するためのインターフェイスを提供する、高性能で柔軟性の高いツールです。Xilinx® Microprocessor Debugger (XMD) を使用して、プロセッサと通信します。

## GDB の使用法

MicroBlaze GDB のコマンド

```
mb-gdb <options> executable-file
```

PowerPC GDB のコマンド

```
powerpc-eabi-gdb <options> executable-file
```

## GDB のコマンド オプション

GNU デバッガがよく使用されるオプションは、次のとおりです。

**-command=FILE**

指定したファイルから GDB コマンドを実行します。バッチおよびスクリプト モードでデバッグを実行する際に使用します。

**-batch**

オプションを処理した後、終了します。バッチおよびスクリプト モードでデバッグを実行する際に使用します。

**-nx**

初期化ファイル `.gdbinit` を読み込みません。XMD への接続で問題が発生した場合は (GDB は XMD ターゲットから接続/接続解除)、このオプションを使用して GDB を起動するか、`.gdbinit` ファイルを削除してください。

**-nw**

GUI インターフェイスを使用しません。

**-w**

GUI インターフェイスを使用します (デフォルト)。

## GDB を使用したデバッグ フロー

1. XPS から XMD を起動します。
2. プロセッサに接続します。この操作により、ターゲットの GDB サーバーが開きます。
3. XPS から GDB を起動します。
4. XMD 上のリモート GDB サーバーに接続します。
5. プログラムおよびデバッグ アプリケーションをダウンロードします。

## 関連リソース

- GNU の Web サイト : <http://www.gnu.org>
- Red Hat Insight の Web サイト : <http://sources.redhat.com/insight>

## MicroBlaze GDB のターゲット

MicroBlaze GNU デバッガおよび XMD ツールでは、リモート ターゲットを使用できます。リモートでのデバッグは、XMD を介して行います。XMD サーバー プログラムは、シミュレータまたはハードウェアを使用して、ホスト コンピュータ上で起動できます。

サイクル精度命令セット シミュレータ (ISS) およびハードウェア インターフェイスは、MicroBlaze システムの完全な検証を実行する高性能のデバッグ ツールです。`mb-gdb` は、TCP/IP ソケット接続上の GDB リモート プロトコルを使用して、XMD に接続します。

### シミュレータ

XMD シミュレータは MicroBlaze システムのサイクル精度 ISS で、シミュレーションされた MicroBlaze システムのステートを GDB に示します。

### ハードウェア

ハードウェア上でのデバッグでは、XMD はマイクロプロセッサ デバッグ モジュール (`mdm`) デバッグ コアまたはハードウェア ボード上で実行される `xmdstub` プログラムとシリアル ケーブルまたは JTAG ケーブルを介して通信し、実行中の MicroBlaze システムのステートを GDB に示します。

XMD の詳細は、第 11 章「Xilinx Microprocessor Debugger (XMD)」を参照してください。

## MicroBlaze をデバッグするためのコンパイル

プログラムをデバッグするには、プログラムをコンパイルする際にデバッグ情報を生成する必要があります。このデバッグ情報はオブジェクト ファイルに保存されます。デバッグ情報には、各変数および関数のデータ型、ソースの行番号に対する実行コードのアドレスなどが含まれています。適切な修飾子を指定すると、ザイリンクス MicroBlaze ソフト プロセッサ用の **mb-gcc** コンパイラによりこの情報が含まれます。

**mb-gcc** で **-g** オプションを使用すると、実行ファイルにデバッグ情報が追加され、ソース レベルでのデバッグを実行できるようになります。ソース、アセンブリ、またはその混合でのデバッグが可能です。

**メモ：**C プログラムの機能を初めて検証する場合は、**-O2** または **-O3** などの **mb-gcc** の最適化オプションを使用しないことをお勧めします。これらのオプションを使用すると、最適化によりコードが大幅に変わる場合があるので、デバッグが困難になります。

**メモ：**XMDStub を使用してハードウェア モードで XMD を使用してデバッグする場合は、**mb-gcc** の **-xl-mode-xmdstub** オプションを指定する必要があります。特定のターゲット用にコンパイルする方法は、第 11 章「[Xilinx Microprocessor Debugger \(XMD\)](#)」を参照してください。

## PowerPC 405 のデバッグ

PowerPC 405 プロセッサのデバッグは、GDB リモート TCP プロトコルを介して **powerpc-eabi-gdb** および XMD を使用して実行します。XMD では、PowerPC 405 ハードウェアとサイクル精度 PowerPC 命令セット シミュレータ (ISS) の 2 つのリモート ターゲットがサポートされます。

PowerPC 405 に接続するには、次の手順に従います。

1. 第 11 章「[Xilinx Microprocessor Debugger \(XMD\)](#)」の説明に従って XMD を起動し、**connect ppc** コマンドを使用してボードに接続します。
2. GDB で [Run] → [Connect to target] をクリックします。
3. [Target Selection] ダイアログ ボックスで、次のように設定します。
  - ◆ [Target] : Remote/TCP
  - ◆ [Hostname] : localhost
  - ◆ [Port] : 1234
4. [OK] をクリックします。

**powerpc-eabi-gdb** により XMD への接続が開始されます。正常に接続されると、XMD を開始したシェル ウィンドウにメッセージが表示されます。

これでデバッガが XMD に接続され、デバッグを制御できるようになります。このユーザー インターフェイスを使用して、プログラムのデバッグおよびメモリとレジスタの読み出し/書き込みを実行します。

## PowerPC 440 のデバッグ

PowerPC 440 プロセッサのデバッグは、GDB リモート TCP プロトコルを介して **powerpc-eabi-gdb** および XMD を使用して実行します。

XMD では、PowerPC 440 ハードウェアとサイクル精度 PowerPC 命令セット シミュレータ (ISS) の 2 つのリモート ターゲットがサポートされます。

PowerPC 440 に接続するには、次の手順に従います。

1. 第 11 章「Xilinx Microprocessor Debugger (XMD)」の説明に従って XMD を起動し、connect ppc コマンドを使用してボードに接続します。
2. GDB で [Run] → [Connect to target] をクリックします。
3. [Target Selection] ダイアログ ボックスで、次のように設定します。
  - ◆ [Target] : Remote/TCP
  - ◆ [Hostname] : localhost
  - ◆ [Port] : 1234
4. [OK] をクリックします。
5. powerpc-eabi-gdb により XMD への接続が開始されます。正常に接続されると、XMD を開始したシェル ウィンドウにメッセージが表示されます。
6. [View] → [Console] をクリックして [Console Window] を開きます。
7. [Console Window] に「set arch powerpc:440」と入力して、アーキテクチャを PowerPC 440 プロセッサに設定します。

これでデバッガが PowerPC 440 モードで XMD に接続され、デバッグを制御できるようになります。このユーザー インターフェイスを使用して、プログラムのデバッグおよびメモリとレジスタの読み出し/書き込みを実行します。

## コンソール モード

powerpc-eabi-gdb をコンソール モードで起動するには、次のように入力します。

```
powerpc-eabi-gdb -nw executable.elf
```

コンソール モードで次の 2 つのコマンドを入力し、XMD を介してボードに接続します。

```
(gdb) target remote localhost:1234
(gdb) load
```

画面に次のテキストが表示されます。

```
Loading section .text, size 0xfcc lma 0xffff8000
Loading section .rodata, size 0x118 lma 0xffff8fd0
Loading section .data, size 0x2f8 lma 0xffff90e8
Loading section .fixup, size 0x14 lma 0xffff93e0
Loading section .got2, size 0x20 lma 0xffff93f4
Loading section .sdata, size 0xc lma 0xffff9414
Loading section .boot0, size 0x10 lma 0xffffa430
Loading section .boot, size 0x4 lma 0xfffffff0
Start address 0xfffffff0, load size 5168
Transfer rate: 41344 bits/sec, 323 bytes/write.
(gdb) c
Continuing
```

コンソール モードでは、この 2 つのコマンドを作業ディレクトリにある GDB スタートアップ ファイル gdb.ini に追加することも可能です。

## GDB コマンドに関するリファレンス情報

mb-gdb の使用方法は、XPS のメイン ウィンドウで [Help] → [Help Topics] をクリック、コンソール モードの場合は「help」と入力してください。

GDB のコンソール ウィンドウを開くには、GDB で [View] → [Console] をクリックします。

GDB の詳細な使用法に関するオンライン マニュアルは、GNU の [Web サイト](#)を参照してください。mb-gdb Insight GUI については、Red Hat Insight の [Web サイト](#)を参照してください。140 ページの「[関連リソース](#)」に、これらのマニュアルへのリンクがあります。

よく使用される mb-gdb コンソール コマンドを表 10-1 に示します。同じ機能を持つ GUI のコマンドは、mb-gdb GUI ウィンドウのアイコンで判別できます。info target、monitor info など一部のコマンドは、コンソール モードでしか使用できません。

表 10-1：よく使用される GDB のコンソール コマンド

コマンド	説明
<b>load</b> <program>	ターゲットにプログラムを読み込みます。
<b>b</b> main	main 関数にブレークポイントを設定します。
<b>c</b>	ブレークポイントの後、プログラムの実行を再開します。 メモ：run コマンドを使用しないでください。
<b>l</b>	現時点でのプログラムのリストを表示します。
<b>n</b>	1 行進めます。関数呼び出しがあっても、停止せずに次の行まで実行します。
<b>s</b>	1 行進めます。関数呼び出しがある場合は、その関数の次の行で停止します。
<b>stepi</b>	アセンブリ 1 行分進めます。
<b>info reg</b>	レジスタ値を表示します。
<b>info target</b>	命令数と実行されたサイクル数を表示します (ビルトイン シミュレータのみ)。
<b>p</b> <xyz>	<xyz> データの値を表示します。
<b>hbreak</b> main	main 関数にハードウェア ブレークポイントを設定します。
<b>watch</b> <gvar1>	グローバル変数 <gvar1> にウォッチポイントを設定します。
<b>rwatch</b> <gvar1>	グローバル変数 <gvar1> に読み出しウォッチポイントを設定します。





# Xilinx Microprocessor Debugger (XMD)

---

Xilinx® Microprocessor Debugger (XMD) は、PowerPC® (405 および 440) プロセッサまたは MicroBlaze™ プロセッサを含むシステムの検証とプログラムのデバッグを行うためのツールです。ハードウェア ボード上またはサイクル精度命令セット シミュレータ (ISS) で実行するプログラムをデバッグできます。

XMD は、Tcl (ツール コマンド 言語) インターフェイスを提供します。このインターフェイスは、ターゲットのコマンド ラインからの制御、ターゲットのデバッグ、および完成したシステム全体をテストする複雑な検証スクリプトの実行に使用します。

XMD では、ターゲットのデバッグを制御するため、GNU デバッガ (GDB) リモート TCP プロトコルがサポートされます。PowerPC プロセッサ GDB および MicroBlaze GDB (powerpc-eabi-gdb および mb-gdb)、Eclipse ベースのソフトウェア ツールであるソフトウェア開発キット (SDK) などのグラフィカル デバッガには、このインターフェイスをデバッグに使用するものがあります。どちらの場合でも、デバッガは同じコンピュータ上またはネットワークを介したリモート コンピュータ上で実行されている XMD に接続されます。

XMD では、プログラムをデバッグするハードウェア システムに関する情報を得るため、XMP (Xilinx Microprocessor Project) システム ファイルを読み込みます。この情報は、メモリ範囲テストを実行したり、高速ダウンロードでの MicroBlaze と MDM (Microprocessor Debug Module) の接続を決定するためなどに使用されます。

この章には、次のセクションが含まれています。

- [関連リソース](#)
- [XMD の使用法](#)
- [XMD コマンド](#)
- [connect コマンドのオプション](#)
- [XMD 内部 Tcl コマンド](#)

146 ページの図 11-1 に、XMD を使用したデバッグを示します。

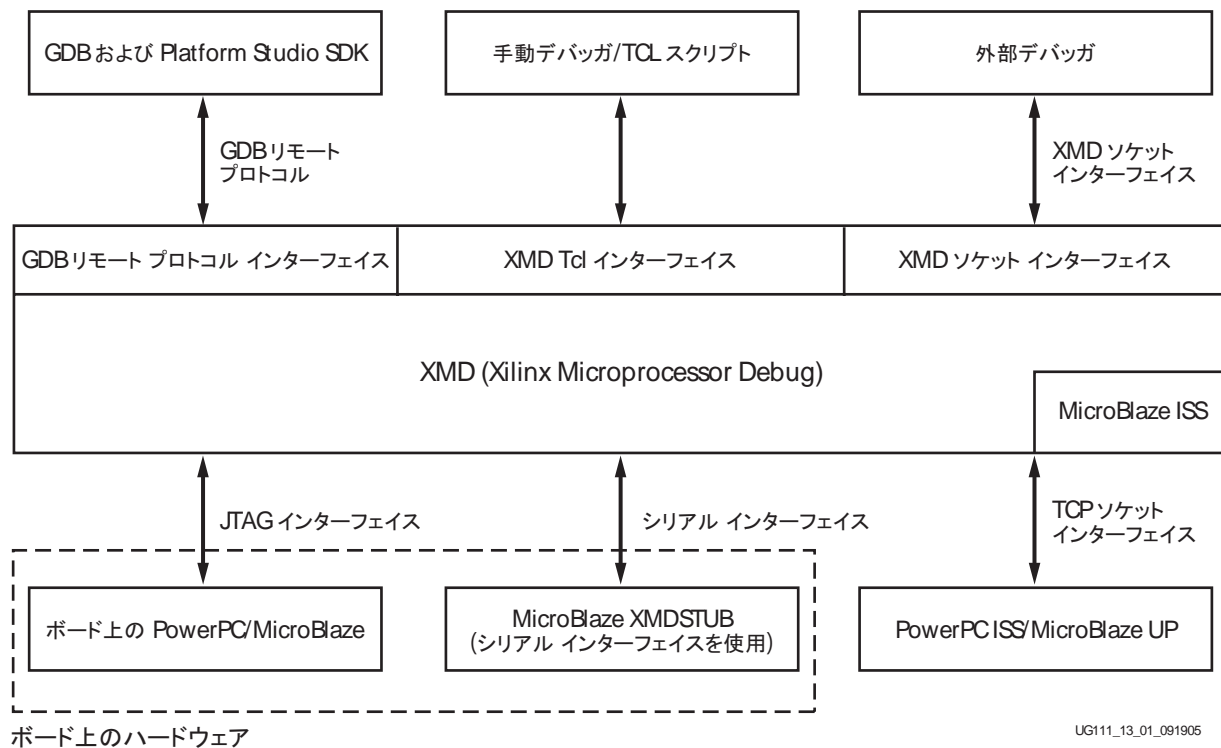


図 11-1 : XMD を使用したデバッグ

## 関連リソース

- PowerPC 405 プロセッサの資料  
PowerPC 404 プロセッサの資料  
『MicroBlaze プロセッサ リファレンス ガイド』  
『IBM Instruction Set Simulator User's Guide』  
[http://japan.xilinx.com/ise/embedded/edk\\_docs.htm](http://japan.xilinx.com/ise/embedded/edk_docs.htm)

## XMD の使用法

```
xmd [-v] [-h] [-help] [-nx] [-ipcport [<portnum>]] [-xmp xmpfile]
[-hw <hardware_specification_file>] [-opt <optfile>]
[-tcl {tcl_file_args}]
```

表 11-1 : XMD のオプション

オプション	コマンド	説明
ハードウェア仕様ファイルの指定	<b>-hw</b> <hw_spec_file>	ハードウェア コンポーネントを説明する XML ファイルを指定します。
ヘルプの表示	<b>-h</b> , <b>-help</b>	コマンドの使用方法を表示して終了します。
初期化ファイルを使用しない	<b>-nx</b>	起動時に xmd.ini ファイルを使用しません。
オプションファイルの指定	<b>-opt</b> <connect_option_file>	ターゲットに接続するのに使用するオプション ファイルを指定します。オプション ファイルには、XMD をターゲットに接続するコマンドが含まれます。
ポート番号	<b>-ipcport</b> [<portnum>]	XMD サーバーをポート番号 <portnum> で開始します。XMD の内部コマンドは、この TCP ポートを使用して発行されます。<portnum> を指定しない場合は、1234 が使用されます。
Tcl ファイルの指定	<b>-tcl</b> <tclfile> <tclarg>	実行する XMD Tcl スクリプトを指定します。  <tclarg> は、Tcl スクリプトへの引数を指定します。この Tcl ファイルは、XMD で使用されます。スクリプトの実行が終了すると、XMD が終了します。  -tcl の後にほかのオプションを指定することはできません。
バージョン情報の表示	<b>-v</b>	バージョン番号を表示して終了します。
XMP ファイルの読み込み	<b>-xmp</b> <xmpfile>	読み込む XMP ファイルを指定します。

XMD を起動すると、次の処理が実行されます。

- XMD Tcl スクリプトが指定されている場合は、そのスクリプトを実行して終了します。
  - XMD Tcl スクリプトが指定されていない場合は、XMD がインタラクティブ モードで実行されます。このモードでは、次の処理が行われます。
1. \${HOME}/.xmdrc ファイルを作成します。このコンフィギュレーション ファイルを使用して、XMD コマンドで使用するカスタム Tcl コマンドを作成できます。
    - ◆ -hw が指定されている場合、XML ファイルを読み込みます。

- ◆ -nx が指定されていない場合、xmd.ini ファイル (現在のディレクトリに存在する場合) を使用します。
  - ◆ -opt が指定されている場合、接続オプション ファイルを使用してプロセッサ ターゲットに接続します。
  - ◆ -ipcport が指定されている場合、XMD ソケット サーバーを開きます。
  - ◆ -xmp が指定されている場合、システムの XMP ファイルを読み込みます。
2. XMD% プロンプトを表示します。XMD Tcl プロンプトから、[148 ページの「XMD コマンド」](#)で説明するように、XMD コマンドを使用してデバッグを実行できます。

## XMD コンソール

XMD コンソールは、Tcl コマンドを実行する標準 Tcl コンソールです。ファイル名やコマンド名の自動入力、コマンド履歴などの入力に便利な機能もあります。

自動入力機能を使用可能な Tcl コマンドは、<EDK\_Install\_Area>/data/xmd/cmdlist ファイルに定義されています。コマンド履歴は \$HOME/.xmcmdhistory に保存されます。自動入力機能を使用可能な Tcl コマンドおよびコマンド履歴に異なるファイルを使用する場合は、環境変数 \$XILINX\_XMD\_CMD\_LIST および \$XILINX\_XMD\_CMD\_HISTORY を使用してデフォルトを上書きします。

## XMD コマンド

### XMD ユーザー コマンドの一覧

次に、XMD コマンドの一覧を示します。オプション名をクリックすると、そのオプションの説明にジャンプします。

<a href="#">bpl</a>	<a href="#">rst</a>
<a href="#">bpr</a>	<a href="#">rwr</a>
<a href="#">bps</a>	<a href="#">run</a>
<a href="#">con</a>	<a href="#">safemode</a>
<a href="#">connect</a>	<a href="#">srrd</a>
<a href="#">cstp</a>	<a href="#">stackcheck</a>
<a href="#">data_verify</a>	<a href="#">state</a>
<a href="#">debugconfig</a>	<a href="#">stats</a>
<a href="#">dis</a>	<a href="#">stop</a>
<a href="#">disconnect</a>	<a href="#">stp</a>
<a href="#">dow</a>	<a href="#">targets</a>
<a href="#">elf_verify</a>	<a href="#">terminal</a>
<a href="#">help</a>	<a href="#">tracestart</a>
<a href="#">mrd</a>	<a href="#">tracestop</a>
<a href="#">mwr</a>	<a href="#">watch</a>

```

profile          verbose
read_uart        xload
rrd

```

## XMD ユーザー コマンド

表 11-2 に、XMD ユーザー コマンドおよびそのオプションを示します。MicroBlaze および PowerPC プロセッサの特殊なレジスタの名前のリストは、[155 ページの「特殊用途レジスタ名」](#)を参照してください。connect コマンドのオプションは、[162 ページの「connect コマンドのオプション」](#)を参照してください。

表 11-2 : XMD ユーザー コマンド

コマンド [オプション]	使用例	説明
<b>bpl</b>	<b>bpl</b>	ブレイクポイントおよびウォッチポイントをリストします。
<b>bpr</b> <b>bpr</b> {all <bp id> <address> <function>}	<b>bpr 0x400</b> <b>bpr main</b> <b>bpr all</b>	ブレイクポイントおよびウォッチポイントを削除します。
<b>bps</b> <b>bps</b> {<address>  <function name>} {sw hw}	<b>bps 0x400</b> <b>bps main hw</b>	<address> または <function name> の開始部分にソフトウェアブレイクポイントまたはハードウェアブレイクポイントを設定します。<function name> を指定すると、最後にダウンロードされた ELF ファイルから検索されます。デフォルトでは、ソフトウェアブレイクポイントが設定されます (sw)。
<b>con</b> <b>con</b> [<Execute Start Address>] [-block [-timeout <Seconds>]]	<b>con</b> <b>con 0x400</b>	現在の PC または <Execute Start Address> から続行します。 <ul style="list-style-type: none"><li>• -block オプションを使用すると、プロセスがブレイクポイントまたはウォッチポイントで停止した場合にコマンドが戻されます。</li><li>• -timeout で値を指定すると、コマンドにより無限にブロックされるのを回避できます。</li><li>• -block オプションは、スクリプトで有益です。</li></ul>
<b>connect</b> <b>connect</b> <target_type(s)>	<b>connect mb mdm</b> <b>connect ppc</b>	ターゲットに接続します。<target_type(s)> に有効な値は、mb、ppc、および mdm です。詳細は、 <a href="#">162 ページの「connect コマンドのオプション」</a> を参照してください。

表 11-2 : XMD ユーザー コマンド (続き)

コマンド [オプション]	使用例	説明
<b>cstp</b> <b>cstp</b> <number of cycles>	<b>cstp</b> <b>cstp</b> 10	<number of cycles> で指定したサイクル数だけ進めます。  <b>メモ</b> : このコマンドは、ISS ターゲットでのみ使用可能です。
<b>data_verify</b> <b>data_verify</b> <Binary filename> <Load Address>	<b>data_verify</b> system.dat 0x400	<Binary filename> がターゲットの <Load Address> に正しくダウンロードされたかどうかを確認します。
<b>debugconfig</b> <b>debugconfig</b> -step_mode {disable_interrupt   enable_interrupt}  <b>debugconfig</b> -memory_datawidth_matching {disable   enable}  <b>debugconfig</b> -reset_on_run <options>	<b>debugconfig</b> -step_mode enable_interrupt  <b>debugconfig</b> -memory_ datawidth_matching enable	ターゲットのデバッグセッションを設定します。詳細は、 <a href="#">182 ページの「デバッグセッションの設定」</a> を参照してください。
<b>dis</b> <b>dis</b> [<address in hex>] [<num words>]	<b>dis</b> 0x400 10	逆アセンブルを実行します。  <b>メモ</b> : MicroBlaze でサポートされます。
<b>disconnect</b> <b>disconnect</b> <target id>	<b>disconnect</b> 0	現在ターゲットとなっているプロセッサへの接続を解除し、GDB サーバーを閉じて、以前のターゲットであるプロセッサに戻ります。

表 11-2 : XMD ユーザー コマンド (続き)

コマンド [オプション]	使用例	説明
<b>dow</b> <b>dow</b> <filename.elf> <b>dow</b> <PIC filename.elf> <Load Address> <b>dow</b> -data <Binary File Name> <Load Address>	<b>dow executable.elf</b> <b>dow executable.elf 0x400</b> <b>dow -data system.dat 0x400</b>	<p>指定の ELF ファイルまたはデータ ファイル (-data オプションを使用) を現在のターゲット メモリにダウンロードします。</p> <ul style="list-style-type: none"> <li>• アドレスを指定しない場合は、ELF ファイルのヘッダからダウンロードするアドレスが判断されます。</li> <li>• アドレスを指定した場合は (MicroBlaze ターゲットのみ)、ELF ファイルは位置独立コード (PIC コード) として処理されて指定のアドレスにダウンロードされます。</li> </ul> <p>また、PIC コードのセマンティクスに従って、レジスタ R20 が開始アドレスに設定されます。</p> <p>ELF ファイルがダウンロードされるとリセットが実行され、ブレークポイントを使用してリセット ロケーションでプロセッサが停止し、ELF プログラムがメモリに読み込まれます。リセットは、システムを既知の状態にするために実行されます。debugconfig -reset_on_run {system enable   processor enable   disable} コマンドを使用すると、リセットの動作を設定できます。182 ページの「デバッグセッションの設定」を参照してください。</p>
<b>elf_verify</b> <b>elf_verify</b> [<filename.elf>]	<b>elf_verify executable.elf</b>	<p>ELF ファイルがターゲットに正しくダウンロードされたかどうかを確認します。ELF ファイルを指定しない場合は、ターゲットに最後にダウンロードされた ELF ファイルが確認されます。</p>
<b>help</b> <b>help</b> [options]	<b>help</b> <b>help init</b> <b>help connect</b> <b>help connect mb</b>	<p>すべてのコマンドを表示します。</p>
<b>mrd</b> <b>mrd</b> <address> [<num of words   half words   bytes> {w h b}] <b>mrd</b> <Global Variable Name>	<b>mrd 0x400</b> <b>mrd 0x400 10</b> <b>mrd 0x400 10 h</b>	<p>&lt;address&gt; から指定したワード/バイト数のメモリ ロケーションを読み出します。デフォルトでは、1 ワード読み出します (w)。</p> <p>変数名 (&lt;Global Variable Name&gt;) を指定した場合、以前にダウンロードされた ELF ファイルのグローバル変数に対応するメモリを読み出します。</p>

表 11-2 : XMD ユーザー コマンド (続き)

コマンド [オプション]	使用例	説明
<b>mrd_var</b> <b>mrd_var</b> <Global Variable Name> <filename.elf>	<b>mrd global_var1</b> <b>executable.elf</b>	指定した ELF ファイル (<filename.elf>) または以前にダウンロードされた ELF ファイルのグローバル変数に対応するメモリを読み出します。
<b>mwr</b> <b>mwr</b> <address> <values> [<num of words/half words/bytes> {w h b}] <b>mwr</b> <Global Variable Name> <values> [<num of words/half words/bytes> {w h b}]	<b>mwr 0x400 0x12345678</b> <b>mwr 0x400 0x1234 1 h</b> <b>mwr 0x400 {0x12345678 0x87654321} 2</b>	<address> または <Global Variable Name> から指定したワード/バイト数のメモリ ロケーションに書き込みます。デフォルトでは、1 ワード書き込みます (w)。
<b>profile</b> <b>profile</b> [-o <GMON Output filename>]	<b>profile -o gproff.out</b>	<p>プロファイル出力ファイルを生成します。このファイルは、mb-gprof または powerpc-eabi-gprof でプロファイル情報を作成するために読み込まれます。</p> <p>プロファイル コンフィギュレーションのサンプリング周波数 (Hz)、ヒストグラムのビン サイズ、収集したプロファイル データのメモリ アドレスを指定します。</p> <p>EDK を使用したプロファイル作成の詳細は、XPS ヘルプを参照してください。</p>
<b>read_uart</b> <b>read_uart</b> [start stop] [TCL Channel ID]	<b>read_uart start</b> <b>read_uart stop</b> <b>read_uart start</b> <b>\$channel_id</b>	<p><b>read_uart start</b> コマンドは、MDM UART インターフェイスからの出力を Tcl チャンネル (TCL Channel ID) に出力します。</p> <p><b>read_uart stop</b> コマンドは、指定した Tcl チャンネルへの出力を停止します。</p> <p>Tcl チャンネルは、開いているファイルまたはソケット接続を示します。このコマンドを使用する前に、適切なコマンドを使用して Tcl チャンネルを開いておく必要があります。</p>
<b>rrd</b> <b>rrd</b> [<reg_num>]	<b>rrd</b> <b>rrd r1</b> または <b>rrd R1</b> <b>rrd 1</b>	すべてのレジスタまたは <reg_num> レジスタを読み出します。
<b>rst</b> <b>rst</b> [-processor]	<b>rst</b> <b>rst -processor</b>	<p>システムをリセットします。</p> <p>-processor オプションを指定すると、現在のプロセッサ ターゲット がリセットされます。</p> <p>プロセッサが実行中でない場合 (state コマンドで確認)、プロセッサはリセット時にプロセッサ リセット ロケーションで停止します。</p>



表 11-2 : XMD ユーザー コマンド (続き)

コマンド [オプション]	使用例	説明
<b>rwr</b> <b>rwr</b> <reg_num / reg_name> / <Hex_value>	<b>rwr pc 0x400</b>	<reg_num>、<reg_name>、または <Hex_value> からレジスタに書き込みます。
<b>run</b>	<b>run</b>	プログラムの開始アドレスからプログラムを実行します。このコマンドはリセットを実行し、ブレークポイントを使用してプロセッサをリセット ロケーションで停止し、ELF プログラム データ セクションをメモリに読み込みます。ELF プログラム データ セクションを読み込むことにより、スタティク変数が適切に初期化され、リセットを実行することによりシステムが既知の状態になります。  <b>debugconfig -reset_on_run</b> {system enable   processor enable   disable} コマンドを使用すると、リセットの動作を設定できます。 <a href="#">182 ページの「デバッグセッションの設定」</a> を参照してください。
<b>safemode</b> <b>safemode</b> [options] <b>safemode</b> [-config <mode> <exception_mask>] <b>safemode</b> [on off]  <b>safemode</b> [-config <exception_id> <exception_addr>]  <b>safemode</b> [-info] <b>safemode</b> [-elf <elf_file>]	<b>safemode -config &lt;mode&gt; &lt;exception_mask&gt;</b>  <b>safemode on</b> <b>safemode off</b>  <b>safemode -config &lt;exception_id&gt; &lt;exception_addr&gt;</b>  <b>safemode -info</b> <b>safemode -elf &lt;elf_file&gt;</b>	セーフモードで読み出されるファイルをイネーブル、ディスエーブル、設定、または指定します。  現在のセーフモード コンフィギュレーションを変更します。  セーフモードをイネーブルまたはディスエーブルにします。  例外ハンドラのアドレスを変更します。  セーフモードの情報を表示します。 デバッグする ELF ファイルを指定します。
<b>srrd</b> <b>srrd</b> [<reg_name>]	<b>srrd</b> <b>srrd pc</b>	特殊なレジスタまたは <reg_name> レジスタを読み出します。
<b>stackcheck</b>	<b>stackcheck</b>	現在のターゲットで実行中のプログラムのスタック使用情報を表示します。ターゲットに最後にダウンロードされた ELF ファイルのスタックがチェックされます。

表 11-2 : XMD ユーザー コマンド (続き)

コマンド [オプション]	使用例	説明
<b>state</b> <b>state</b> [<target_id>] <b>state</b> -system <system_id>	<b>state</b> <b>state</b> <target id> <b>state</b> -system <system id>	ターゲットを指定しない場合、すべてのターゲットの現在のステートを表示します。 <ul style="list-style-type: none"> <li>• &lt;target_id&gt; を指定した場合、そのターゲットの現在のステートを表示します。</li> <li>• -system &lt;system_id&gt; を指定した場合、そのシステムに含まれるすべてのターゲットの現在のステートを表示します。</li> </ul>
<b>stats</b> <b>stats</b> [<filename>]	<b>stats</b> trace.txt <b>stats</b>	ISS および VP ターゲットに対する実行統計を表示します。<filename> は、トレース情報収集からの出力です。
<b>stop</b>	<b>stop</b>	ターゲットを停止します。MicroBlaze では、プログラムがメモリまたは FSL アクセスでストールした場合は、強制的に停止されます。
<b>stp</b> <b>stp</b> <number of instructions>	<b>stp</b> <b>stp</b> 10	<number of instructions> で指定したステップ数だけ命令を進めます。
<b>targets</b> <b>targets</b> <target_id> <b>targets</b> -system <system_id>	<b>targets</b> <b>targets</b> 0 <b>targets</b> -system 1	現在のターゲットに関する情報をリストしたり、ターゲットを変更します。
<b>terminal</b> <b>terminal</b> [-jtag_uart_server] [<port_no>]	<b>terminal</b> <b>terminal</b> <b>-jtag_uart_server</b> 4321	JTAG ベースのハイパーターミナルを mdm UART インターフェイスと通信させます。mdm で UART インターフェイスをイネーブルにする必要があります。  -jtag_uart_server オプションを指定した場合、TCP サーバーが <port_no> で開きます。任意のハイパーターミナルユーティリティを使用して、TCP ソケットで opb_mdm UART インターフェイスと通信します。<port_no> のデフォルト値は 4321 です。
<b>tracestart</b> <b>tracestart</b> [<pc_trace_filename>] [-function_name <func_trace_filename>]	<b>tracestart</b> pctrace.txt <b>tracestart</b> pctrace.txt - <function_name> <b>fntrace.txt</b> <b>tracestart</b>	トレース情報の収集を開始し、指定したファイルに記述します。 <ul style="list-style-type: none"> <li>• トレース情報の収集は、プログラム実行のどの時点でも停止、開始できます。</li> <li>• ファイル名は、最初に tracestart を使用する場合のみ指定します。</li> <li>• &lt;pc_trace_filename&gt; のデフォルトのファイル名は isstrace.out です。</li> <li>• &lt;func_trace_filename&gt; のデフォルトのファイル名は fntrace.out です。</li> </ul> <b>メモ</b> : このコマンドは、ISS ターゲットでのみ使用可能です。

表 11-2 : XMD ユーザー コマンド (続き)

コマンド [オプション]	使用例	説明
<b>tracestop</b> <b>tracestop [done]</b>	<b>tracestop</b> <b>tracestop done</b>	<p>トレース情報の収集を停止します。done オプションを指定した場合は、トレース情報収集を終了します。</p> <p><b>メモ</b>：このコマンドは、ISS ターゲットでのみ使用可能です。</p>
<b>watch</b> <b>watch {r w} &lt;address&gt;</b> <b>[&lt;data&gt;]</b>	<b>watch r 0x400 0x1234</b> <b>watch r 0x40X 0x12X4</b> <b>watch r 0b01000000XXXX</b> <b>0b00010010XXXX0100</b> <b>watch r 0x40X</b>	<p>&lt;address&gt; に読み出しまたは書き込みのウォッチポイントを設定します。値が &lt;data&gt; と一致すると、プロセッサを停止します。</p> <ul style="list-style-type: none"> <li>• アドレスおよびデータは、16 進数フォーマット (0x) または 2 進数フォーマット (0b) で指定できます。</li> <li>• ドントケアの値は、X で指定します。</li> <li>• アドレスには、連続した範囲しか指定できません。</li> <li>• &lt;data&gt; のデフォルト 値は 0xFFFFFFFF です。この値は、任意の値に一致します。</li> </ul> <p><b>メモ</b>：PowerPC プロセッサでは、絶対値のみがサポートされます。</p>
<b>verbose</b> <b>verbose [level]</b>	<b>verbose</b>	<p>詳細モードのオン/オフを切り替えます。詳細モードを使用すると、XMD にデバッグ情報が表示されます。</p>
<b>xload</b> <b>xload xmp &lt;xmp_filename&gt;</b>	<b>xload xmp system.mhs</b>	<p>XMP システム ファイルを読み込みます。XMD は、プロセッサの命令およびデータ メモリ アドレスを取得するため XMP ファイルを読み込みます。この情報は、プロセッサのメモリにダウンロードされたプログラムおよびデータを確認するために使用されます。</p>

## 特殊用途レジスタ名

### MicroBlaze の特殊用途レジスタ名

MicroBlaze プロセッサで有効な特殊用途レジスタ名は、次のとおりです。

pc	msr	ear	esr	zpr
fsr	btr	pvr0	pvr1	zpr
pvr2	pvr3	pvr4	pvr5	zpr
pvr6	pvr7	pvr8	pvr9	
pvr10	pvr11	edr	pid	

MicroBlaze の特殊用途レジスタ名の詳細、説明、使用法は、『MicroBlaze プロセッサ リファレンス ガイド』の「MicroBlaze アーキテクチャ」の章で「特殊用途レジスタ」を参照してください。  
146 ページの「関連リソース」に、このマニュアルへのリンクがあります。

メモ : MicroBlaze を XMDStub モードでデバッグする場合、アクセスできるレジスタは PC および MSR のみです。

## PowerPC 405 プロセッサの特殊用途レジスタ名

PowerPC 405 プロセッサで有効な特殊用途レジスタ名は、次のとおりです。

表 11-3 : PowerPC 405 プロセッサの特殊用途レジスタ名

ccr0	f0	f11	f22	iac1	pvr	su0r
cr	f1	f12	f23	iac2	sgr	tbl
ctr	f2	f13	f24	iac4	sler	tbu
dac1	f3	f14	f25	iccr	sprg0	tcr
dac2	f4	f15	f26	icdbdr	sprg1	tsr
dbcr0	f5	f16	f27	lr	sprg2	usprg0
dbcr1	f6	f17	f28	msr	sprg3	xer
dbsr	f7	f18	f29	pc	sprg4	zpr
dccr	f8	f19	f30	pid	sprg5	su0r
dcwr	f9	f20		pit	sprg6	tbl
dear	f10	f21		iac1	sprg7	tbu
dvc1				iac2	srr0	
dvc2					srr1	
esr					srr2	
evpr					srr3	

メモ : XMD では、常に 64 ビット表記を使用して浮動小数点レジスタ (f0 ~ f31) を表します。単精度浮動小数点ユニットでは、32 ビット単精度値が 64 ビット値に拡張されます。

PowerPC 405 プロセッサの特殊用途レジスタ名の詳細は、『PowerPC 405 Processor Block Reference Guide』を参照してください。146 ページの「関連リソース」に、このマニュアルへのリンクがあります。

## PowerPC 440 プロセッサの特殊用途レジスタ名

PowerPC 440 プロセッサで有効な特殊用途レジスタ名は、次のとおりです。

表 11-4 : PowerPC 440 プロセッサの特殊用途レジスタ名

pc	msr	cr	lr	ctr	xer
fpscr	pvr	sprg0	sprg1	sprg2 s	prg3
srr0	srr1	tbl	tbu	icdbdr	esr
dear	ivpr	tsr	tcr	dec	csrr0

表 11-4 : PowerPC 440 プロセッサの特殊用途レジスタ名 (続き)

csrr1	dbsr	dbcr0	iac1	iac2	dac1
dac2	pir	rstcfg	mmucr	pid	ccr1
dbdr	ccr0	dbcr1	dvc1	dvc2	iac3
iac4	dbcr2	sprg4	sprg5	sprg6	sprg7
decar	usprg0	ivor0	ivor1	ivor2	ivor3
ivor4	ivor5	ivor6	ivor7	ivor8	ivor9
ivor10	ivor11	ivor12	ivor13	ivor14	ivor15
inv0	inv1	inv2	inv3	itv0	itv1
itv2	itv3	dnv0	dnv1	dnv2	dnv3
dtv0	dtv1	dtv2	dtv3	dvlim	ivlim
dcdctrl	dcdctrlh	icdctrl	icdctrlh	mcsr	mcsrr0
mcsrr1	f0	f1	f2	f3	f4
f5	f6	f7	f8	f9	f10
f11	f12	f13	f14	f15	f16
f17	f18	f19	f20	f21	f22
f23	f24	f25	f26	f27	f28
f29	f30	f31			

メモ : XMD では、常に 64 ビット表記を使用して浮動小数点レジスタ (f0 ~ f31) を表します。単精度浮動小数点ユニットでは、32 ビット単精度値が 64 ビット値に拡張されます。

PowerPC 440 プロセッサの特殊用途レジスタ名の詳細は、『Embedded Processor Block in Virtex-5 FPGAs Reference Guide』を参照してください。146 ページの「関連リソース」に、このマニュアルへのリンクがあります。

## XMD のリセット シーケンス

rst コマンドを実行すると、XMD によりプロセッサまたはシステムがリセットされ、既知の状態に戻されます。次に、プロセッサのタイプ別に rst によるリセット シーケンスを示します。

### PowerPC 405 プロセッサ

1. 仮想アドレス指定をディスエーブルにします。
2. リセット アドレス (0xFFFFF0FC) が書き込み可能で OCM 上ではない場合、リセット ロケーションに自身への分岐命令を書き込みます。
3. DBCR0 を 0x81000000 に設定します。
4. JTAG デバッグ コントロール レジスタ (DCR) を介してリセット信号 (システム リセットまたはプロセッサ リセット) を発行します。プロセッサが動作を開始します。
5. プロセッサを停止します。
6. リセット アドレスに元の命令を復元します。

## PowerPC 440 プロセッサ

1. DBCR0 を 0x81000000 に設定します。
2. レジスタ MMUCR を 0 に設定します。
3. DBCR1 および DBCR2 を 0 に設定します。
4. 仮想アドレスが実際のアドレスと同じになるよう TLB を設定します。
5. シャドウ TLB と同期化します。
6. リセット アドレス (0xFFFFF000) が書き込み可能である場合、リセット ロケーションに自身への分岐命令を書き込みます。
7. JTAG DCR を介してリセット信号 (システム リセットまたはプロセッサ リセット) を発行します。プロセッサが動作を開始します。
8. プロセッサを停止します。
9. リセット アドレスに元の命令を復元します。

## MicroBlaze

1. リセット ロケーション (0x0) にハードウェア ブレークポイントを設定します。
2. リセット信号 (システム リセットまたはプロセッサ リセット) を発行します。プロセッサが動作を開始します。
3. プロセッサがリセット ロケーションが停止したら、ブレークポイントを削除します。

## 推奨される XMD フロー

次に、1 つのプログラムのデバッグ、複数プロセッサ環境でのプログラムのデバッグ、デバッグセッションでのプログラムの実行に推奨される XMD の手順を説明します。

### 1 つのプログラムのデバッグ

1 つのプログラムをデバッグするには、次の手順に従います。

1. プロセッサを接続します。
2. ELF ファイルをダウンロードします。
3. 必要なブレークポイントおよびウォッチポイントを設定します。
4. con コマンドを使用してプロセッサの実行を開始するか、stp コマンドを使用してプログラムの命令を順に実行します。
5. state コマンドを使用してプロセッサのステータスを確認します。
6. 必要に応じて stop コマンドを使用し、プロセッサを停止します。
7. プロセッサが停止したら、レジスタおよびメモリに対して読み出しおよび書き込みを実行します。
8. プログラムを再実行するには、run コマンドを使用します。

## 複数プロセッサ環境でのプログラムのデバッグ

複数プロセッサ環境でプログラムをデバッグするには、次の手順に従います。

1. プロセッサ 1 に接続します。
2. `debugconfig` コマンドを使用してリセット時の動作を設定します。リセット時の動作は、システムアーキテクチャにより異なります。「[デバッグセッションの設定](#)」を参照してください。
3. ELF ファイルをダウンロードします。
4. 必要なブレークポイントおよびウォッチポイントを設定します。
5. `con` コマンドを使用してプロセッサの実行を開始するか、`stp` コマンドを使用してプログラムの命令を順に実行します。
6. プロセッサ 2 に接続します。
7. `debugconfig` コマンドを使用してリセット時の動作を設定します。リセット時の動作は、システムアーキテクチャにより異なります。「[デバッグセッションの設定](#)」を参照してください。
8. ELF ファイルをダウンロードします。
9. 必要なブレークポイントおよびウォッチポイントを設定します。
10. `con` コマンドを使用してプロセッサの実行を開始するか、`stp` コマンドを使用してプログラムの命令を順に実行します。
11. `targets` コマンドを使用してシステムのターゲットをリストします。各ターゲットにはターゲット ID が付けられています。アクティブなターゲットは、アスタリスク (\*) で示されます。
12. `targets <target id>` コマンドを使用してターゲットを切り替えます。
13. `state` コマンドを使用してプロセッサのステータスを確認します。
14. `stop` コマンドを使用して、プロセッサを停止します。
15. プロセッサが停止したら、レジスタおよびメモリに対して読み出しおよび書き込みを実行します。
16. プログラムを再実行するには、`run` コマンドを使用します。

## デバッグセッションでのプログラムの実行

1. プロセッサに接続します。
2. ELF ファイルをダウンロードします。
3. `<exit>` 関数でブレークポイントを設定します。
4. `con` コマンドを使用してプロセッサの実行を開始します。
5. `state` コマンドを使用してプロセッサのステータスを確認します。
6. `stop` コマンドを使用して、プロセッサを停止します。
7. プロセッサが停止したら、レジスタおよびメモリに対して読み出しおよび書き込みを実行します。
8. プログラムを再実行するには、`run` コマンドを使用します。

## 自動例外トラップでのセーフモードの使用

XMD では、エラーが発生した場合にプログラムの例外をトラップできます。エラーには、不正な命令の実行やバスエラーなどがあります。次の手順に従います。

1. プログラムをダウンロードします。

2. `safemode on` コマンドを実行します。
3. `con` コマンドを使用してプログラムを開始します。

例外が発生すると、プログラムが停止します。この機能は、GUI デバッガ (Insight GDB または SDK) で作業する際に有益です。

- SDK を使用する場合、プログラムを実行する前に [Initialization] タブで [Enable Safemode] チェック ボックスをオンにしてください。
- GDB を使用する場合、プログラムを実行する前に、プログラムをダウンロードして XMD コンソールで `safemode on` コマンドを実行してください。

例外が発生するとプログラムが停止し、GUI に例外が発生させたコード行が表示されます。

## プロセッサのデフォルト例外設定

次の表に、例外トラップのデフォルト設定をプロセッサのタイプ別に示します。

表 11-5 : PowerPC プロセッサの例外設定

例外 ID	トラップ	例外名
0	なし	外部クリティカル割り込み例外
1	あり	外部バス エラー例外
2	あり	データ格納例外
3	あり	命令格納例外
4	なし	外部非クリティカル割り込み例外
5	なし	不整列データ アクセス例外
6	あり	不正 op コード例外
7	あり	FPU 使用不可例外
8	なし	システム読み出し命令
9	あり	APU 使用不可例外
10	なし	プログラマブル インターバル タイマのタイムアウト例外
11	なし	固定インターバル タイマのタイムアウト例外
12	なし	ウォッチドッグ タイマのタイムアウト例外
13	なし	データ TLB ミス例外
14	なし	命令 TLB ミス例外
15	なし	デバッグ イベント例外
16	あり	アサート エラー
17	あり	プログラム終了



表 11-6 : MicroBlaze の例外設定

例外 ID	トラップ	例外名
0	あり	高速シンプレックス リンク例外
1	なし	不整列データ アクセス例外
2	あり	不正 op コード例外
3	あり	命令バス エラー例外
4	あり	データ バス エラー例外
5	あり	0 での除算例外
6	あり	浮動小数点例外
7	あり	特権命令例外
8	あり	データ格納例外
9	あり	命令格納例外
10	あり	データ TLB ミス例外
11	あり	命令 TLB ミス例外
12	あり	アサート エラー
13	あり	プログラム終了

## 例外設定の上書き

デフォルトの例外設定を上書きするには、次の 2 つの方法があります。

1. 次のコマンドを使用します。

```
xmdconfig [-mb_trap_mask|-ppc_trap_mask] [MASK]
```

このコマンドでは、現在の XMD セッションのすべてのターゲットにマスクが設定されます。すべての XMD セッションに独自のデフォルト設定を定義するには、そのコマンドをホームディレクトリにある .xmdrc ファイルに記述します。

2. 次のコマンドを使用します。

```
safemode -config mode [MASK]
```

このコマンドでは、現在のターゲットにのみマスクが設定されます。プログラムのデバッグ中、トラップ設定を変更するのに便利な方法です。

メモ：ターゲットへの接続を切断すると、現在のターゲットは破棄されます。

## セーフモード設定の表示

現在のセーフモード設定を表示するには、次のコマンドを入力します。

```
safemode -info
```

セーフモードでは、XMD によりトラップする例外ハンドラにブレークポイントが設定されます。

- MicroBlaze では、すべての例外により PC が 0x20 に移動します。
- PowerPC プロセッサでは、ELF ファイルから例外ハンドラの手番が検出されます。

この検出は、ほとんどのスタンドアロンまたは Xilkernal プロジェクトで機能します。別のソフトウェア プラットフォームを使用している場合、検出がうまく機能しない可能性があります。その場合は、次のコマンドを使用して例外ハンドラのアドレスを設定してください。

```
safemode -config [exception_id] [exception_handler_addr]
```

## connect コマンドのオプション

XMD では、異なるターゲット (プロセッサまたはペリフェラル) でプログラムをデバッグできます。XMD とターゲットの通信を確立するには、XMD をターゲットに接続する必要があります。接続が確立されると、各ターゲットに固有のターゲット ID が割り当てられます。プロセッサに接続すると、GDB サーバーが起動し、GDB または SDK との通信が可能になります。

### 構文

```
connect {mb | ppc | mdm} <Connection_Type> [Options]
```

表 11-7 : connect コマンドのオプション

オプション	説明
ppc	PowerPC プロセッサに接続します。
mb	MicroBlaze プロセッサに接続します。
mdm	MDM ペリフェラルに接続します。
<Connection_Type>	接続方法を指定します。ターゲットによって異なります。
[Options]	接続オプション。

次のセクションで、異なるターゲットの接続オプションについて説明します。

## PowerPC プロセッサ

ザイリンクス Virtex® デバイスには、PowerPC (405 および 440) プロセッサ コアが 1 つまたは 2 つ含まれています。XMD は、ボード上の JTAG 接続を介してこれらの PowerPC プロセッサに接続します。また、TCP ソケット インターフェイスを介して IBM PowerPC プロセッサ命令セットシミュレータ (ISS) にも接続されます。

PowerPC プロセッサに接続するには、connect ppc コマンドを使用して、リモートの GDB サーバーを起動します。XMD が PowerPC プロセッサに接続されたら、XMD を介して powerpc-eabi-gdb または SDK を PowerPC に接続し、デバッグを実行します。

**メモ :** XMD では、仮想アドレス指定はサポートされません。デバッグは、リアル モードで実行されているプログラムでのみ可能です。

### PowerPC プロセッサ ハードウェアの接続

XMD は PowerPC プロセッサ ハードウェアに接続する際、JTAG チェーンと PowerPC プロセッサ タイプおよびシステム内のプロセッサを自動的に検出し、最初のプロセッサに接続します。このデフォルト処理は、次のオプションを使用して変更できます。

## 構文

```
connect ppc hw [-cable <JTAG Cable options>] {[-configdevice <JTAG chain options>]} [-debugdevice <PowerPC options>]
```

### -cable のオプション (JTAG ケーブル オプション)

次のオプションを使用して、ターゲットとの接続に使用する JTAG ケーブルを指定します。

表 11-8 : -cable のオプション (JTAG ケーブル オプション)

オプション	説明
<b>fname</b> <filename.svf>	SVF (Serial Vector Format) ファイルの名前を指定します。
<b>frequency</b> <cable speed in Hz>	ケーブルのクロック スピードを Hz で指定します。 有効なケーブル スピードは次のとおりです。 <ul style="list-style-type: none"> <li>• パラレル ケーブル 4 : 5000000 (デフォルト)、2500000、200000</li> <li>• プラットフォーム USB : 24000000、12000000、6000000 (デフォルト)、3000000、1500000、750000</li> </ul>
<b>port</b> <port name>	ポートを指定します。有効なポート名は lpt1、lpt2、...usb21、usb22... です。
<b>type</b> <cable_type>	ケーブルのタイプを指定します。有効なケーブル タイプは次のとおりです。 <ul style="list-style-type: none"> <li>• xilinx_parallel3</li> <li>• xilinx_parallel4</li> <li>• xilinx_platformusb</li> <li>• xilinx_svffile</li> </ul> xilinx_svffile に指定すると、fname オプションで指定したファイルに JTAG コマンドが記述されます。

### -configdevice のオプション (JTAG チェーン オプション)

次のオプションを使用して、JTAG チェーンに含まれるザイリンクス以外のデバイスに関する情報を指定します。169 ページの「特殊な JTAG チェーン設定でのザイリンクス以外のデバイスのデバッグ セッション例」を参照してください。

表 11-9 : -configdevice のオプション (JTAG チェーン オプション)

オプション	説明
<b>devicenr</b> <device position>	JTAG チェーン内のデバイスの位置を指定します。デバイス位置の番号は、1 から開始します。
<b>irlength</b> <length of the JTAG Instruction Register>	デバイスの IR レジスタの幅を指定します。この情報は、デバイスの BSDL ファイルに記述されています。
<b>idcode</b> <device idcode>	デバイスの JTAG ID コードを指定します。
<b>jtagport</b> <cpu>	PowerPC プロセッサの JTAG ピンを FPGA ユーザー IO ピンに直接接続するかを指定します。
<b>partname</b> <device name>	デバイス名を指定します。

## -debugdevice のオプション (PowerPC プロセッサ オプション)

次のオプションを使用して、デバッグする FPGA デバイスとプロセッサの番号を指定します。ISOCM、キャッシュ、TLB、DCR レジスタなどの PowerPC プロセッサの特殊な機能を未使用のメモリ アドレスに割り当て、デバッガからメモリ アドレスとしてアクセスできるようにすることも可能です。この機能は、GDB または XMD からこれらのレジスタやメモリの書き込みおよび読み出しを行う場合に便利です。

**メモ**：これらのオプションでは、ハードウェアに実際のメモリ マップは作成されません。

表 11-10 : -debugdevice のオプション (PowerPC オプション)

オプション	説明
<b>cpunr</b> <CPU Number>	複数の PowerPC プロセッサを含む Virtex デバイスで、デバッグする PowerPC の番号を指定します。プロセッサの番号は、1 から開始します。
<b>dcachestartadr</b> <D-Cache start address>	データ キャッシュの内容の読み出しまたは書き込みを行う開始アドレスを指定します。
<b>dcrstartadr</b> <DCR start address>	デバイス コントロール レジスタ (DCR) の読み出しまたは書き込みを行う開始アドレスを指定します。  このオプションを使用すると、XMD および GDB からのデバッグにおいて、DCR のアドレス空間全体 (210 個のアドレス) を指定したアドレスから始まるアドレスにマップできます。
<b>devicenr</b> <PowerPC device position>	PowerPC プロセッサを含む Virtex デバイスの JTAG チェーンでの位置を指定します。デバイス位置の番号は、1 から開始します。
<b>dtagstartadr</b> <D-Cache start address>	データ キャッシュ タグの読み出しまたは書き込みを行う開始アドレスを指定します。
<b>fputype</b> [sp dp]	XMD では、PowerPC プロセッサ システムで浮動小数点ユニット (FPU) は自動的に検出されません。FPU が検出されるようにするには、このオプションでシステムに含まれる FPU のタイプを指定します。指定可能なタイプは、次のとおりです。  sp : 単精度 dp : 倍精度
<b>icachestartadr</b> <I-Cache start address>	命令キャッシュの内容の読み出しまたは書き込みを行う開始アドレスを指定します。
<b>isocmdcrstartadr</b> <ISOCM (in Bytes) DCR address>	PowerPC 405 プロセッサ上で C_ISOCM_DCR_BASEADDR パラメータを使用して指定した ISOCM インターフェイスに対応する DCR アドレスを指定します。
<b>isocmstartadr</b> <ISOCM start address>	命令側オンチップ メモリ (ISOCM) の開始アドレスを指定します。PowerPC 405 プロセッサでのみ使用可能です。

表 11-10 : -debugdevice のオプション (PowerPC オプション) (続き)

オプション	説明
<b>isocmsize</b> <ISOCM size in Bytes>	ISOCM インターフェイスに接続された ISBRAM メモリのサイズを指定します。PowerPC 405 プロセッサでのみ使用可能です。
<b>itagstartadr</b> <I-Cache start address>	命令キャッシュ タグの読み出しまたは書き込みを行う開始アドレスを指定します。
<b>romemstartadr</b> <ROM start address>	読み出し専用メモリ (ROM) の開始アドレスを指定します。このオプションは、フラッシュ メモリの範囲を指定するのに使用できます。XMD で、ソフトウェアブレイクポイントではなくハードウェアブレイクポイントが設定されます。
<b>romemsize</b> <ROM Size in Bytes>	読み出し専用メモリ (ROM) のサイズを指定します。
<b>tlbstartadr</b> <TLB start address>	変換ルックアサイド バッファ (TLB) の読み出しまたは書き込みを行う開始アドレスを指定します。

## PowerPC プロセッサのデバッグにおける要件

XMD から JTAG を介して PowerPC プロセッサに接続するには、次の 2 つの方法があります。これら 2 つの方法の要件を次のサブセクションに示します。

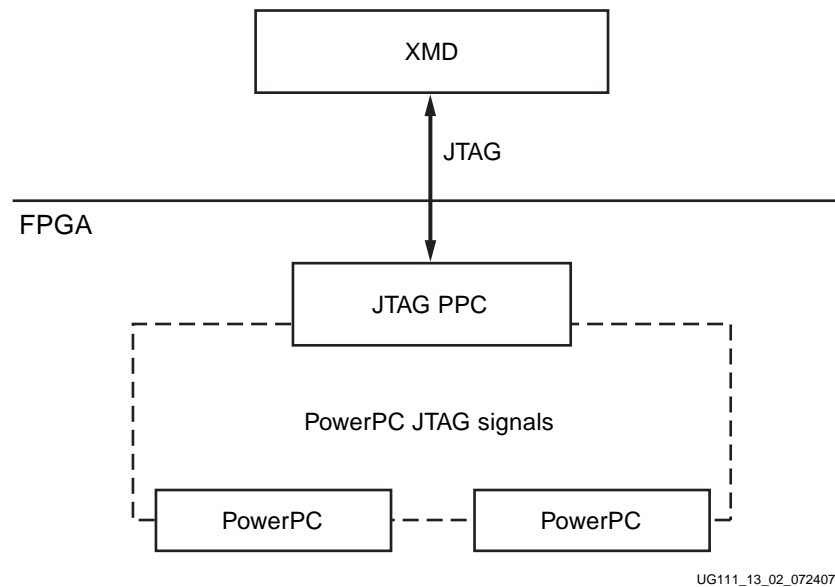
### Virtex FPGA の JTAG ポートを使用して接続する方法

図 11-2 に示すように、PowerPC の JTAG ポートを JTAGPPC プリミティブを使用して FPGA の JTAG ポートに接続している場合は、XMD から FPGA 内のどの PowerPC プロセッサにも接続できます。詳細は、『PowerPC 405 Processor Block Reference Guide』および『Embedded Processor Block in Virtex-5 FPGAs Reference Guide』を参照してください。146 ページの「関連リソース」に、このマニュアルへのリンクがあります。

### ユーザー I/O ピンを使用して PowerPC プロセッサの JTAG ポートに接続する方法

PowerPC の JTAG ポートを FPGA の I/O ピンに接続している場合は、XMD から PowerPC プロセッサに直接接続できます。このモードでは、XMD は 1 つの PowerPC プロセッサとしか通信できません。システムに PowerPC プロセッサが 2 つある場合、それらをチェーン接続することはできず、各プロセッサへの JTAG ポートを FPGA I/O ピンを使用するよう取り出す必要があります。詳細は、『PowerPC 405 Processor Block Reference Guide』および『Embedded Processor Block in Virtex-5 FPGAs Reference Guide』を参照してください。146 ページの「関連リソース」に、このマニュアルへのリンクがあります。

次の図に、PowerPC プロセッサのデバッグを示します。



UG111\_13\_02\_072407

図 11-2 : PowerPC プロセッサのデバッグ

## デバッグ セッションの例

### PowerPC 405 プロセッサのデバッグ例

次に、PowerPC 405 プロセッサをデバッグするセッションの例を示します。connect ppc hw コマンドを使用して PowerPC プロセッサに接続した後、基本的な XMD コマンドを使用します。

セッションの最後に、powerpc-eabi-gdb が GDB リモート ターゲットを使用して XMD に接続します。GDB の XMD への接続については、第 10 章「GNU デバッグ (GDB)」を参照してください。

```

XMD% connect ppc hw
JTAG chain configuration
-----
Device    ID Code          IR Length    Part Name
1         0a001093          8            System_ACE
2         f5059093         16           XCF32P
3         01e58093         10           XC4VFX12
4         49608093          8            xc95144x1

PowerPC405 Processor Configuration
-----
Version.....0x20011430
User ID.....0x00000000
No of PC Breakpoints.....4
No of Read Addr/Data Watchpoints....1
No of Write Addr/Data Watchpoints...1
User Defined Address Map to access Special PowerPC Features using XMD:
  I-Cache (Data).....0x70000000 - 0x70003fff
  I-Cache (TAG).....0x70004000 - 0x70007fff
  D-Cache (Data).....0x78000000 - 0x78003fff
  D-Cache (TAG).....0x78004000 - 0x78007fff
  DCR.....0x78004000 - 0x78004fff
  TLB.....0x70004000 - 0x70007fff
Connected to "ppc" target. id = 0
  
```

Starting GDB server for "ppc" target (id = 0) at TCP port no 1234

XMD% **rrd**

r0: ef0009f8	r8: 51c6832a	r16: 00000804	r24: 32a08800
r1: 00000003	r9: a2c94315	r17: 00000408	r25: 31504400
r2: fe008380	r10: 00000003	r18: f7c7dfcd	r26: 82020922
r3: fd004340	r11: 00000003	r19: fbcbefce	r27: 41010611
r4: 0007a120	r12: 51c6832a	r20: 0040080d	r28: fe0006f0
r5: 000b5210	r13: a2c94315	r21: 0080040e	r29: fd0009f0
r6: 51c6832a	r14: 45401007	r22: c1200004	r30: 00000003
r7: a2c94315	r15: 8a80200b	r23: c2100008	r31: 00000003
pc: ffff0700	msr: 00000000		

XMD% **srrd**

pc: ffff0700	msr: 00000000	cr: 00000000	lr: ef0009f8
ctr: ffffffff	xer: c000007f	pvr: 20010820	sprg0: ffffe204
sprg1: ffffe204	sprg2: ffffe204	sprg3: ffffe204	srr0: ffff0700
srr1: 00000000	tbl: a06ea671	tbu: 00000010	icdbdr: 55000000
esr: 88000000	dear: 00000000	evpr: ffff0000	tsr: fc000000
tcr: 00000000	pit: 00000000	srr2: 00000000	srr3: 00000000
dbsr: 00000300	dbcr0: 81000000	iac1: ffffe204	iac2: ffffe204
dac1: ffffe204	dac2: ffffe204	dccr: 00000000	iccr: 00000000
zpr: 00000000	pid: 00000000	sgr: ffffffff	dcwr: 00000000
ccr0: 00700000	dbcr1: 00000000	dvc1: ffffe204	dvc2: ffffe204
iac3: ffffe204	iac4: ffffe204	sler: 00000000	sprg4: ffffe204
sprg5: ffffe204	sprg6: ffffe204	sprg7: ffffe204	su0r: 00000000
usprg0: ffffe204			

XMD% **rst**

Sending System Reset

Target reset successfully

XMD% **rwr 0 0xAAAAAAAA**

XMD% **rwr 1 0x0**

XMD% **rwr 2 0x0**

XMD% **rrd**

r0: aaaaaaaaa	r8: 51c6832a	r16: 00000804	r24: 32a08800
r1: 00000000	r9: a2c94315	r17: 00000408	r25: 31504400
r2: 00000000	r10: 00000003	r18: f7c7dfcd	r26: 82020922
r3: fd004340	r11: 00000003	r19: fbcbefce	r27: 41010611
r4: 0007a120	r12: 51c6832a	r20: 0040080d	r28: fe0006f0
r5: 000b5210	r13: a2c94315	r21: 0080040e	r29: fd0009f0
r6: 51c6832a	r14: 45401007	r22: c1200004	r30: 00000003
r7: a2c94315	r15: 8a80200b	r23: c2100008	r31: 00000003
pc: ffffffff	msr: 00000000		

XMD% **mrd 0xFFFFF000**

FFFFF000: 4BFFFC74

XMD% **stp**

fffffc70:

XMD% **stp**

fffffc74:

XMD% **mrd 0xFFFFC000 5**

FFFC000: 00000000

FFFC004: 00000000

FFFC008: 00000000

FFFC00C: 00000000

FFFC010: 00000000

XMD% **mwr 0xFFFFC004 0xabcd1234 2**

XMD% **mwr 0xFFFFC010 0xa5a50000**

XMD% **mrd 0xFFFFC000 5**

FFFC000: 00000000

FFFC004: ABCD1234

FFFC008: ABCD1234

```
FFFFC00C: 00000000
FFFFC010: A5A50000
XMD%
XMD%
```

## PowerPC 440 プロセッサのデバッグ例

PowerPC 440 プロセッサに接続するには、`connect ppc hw` コマンドを使用します。

XMD によりプロセッサのタイプが自動的に検出され、PowerPC 440 プロセッサに接続されます。

ソフトウェアをリモートでデバッグするには、`powerpc-eabi-gdb` を使用します。GDB の XMD への接続については、第 10 章「GNU デバッガ (GDB)」を参照してください。

```
XMD% connect ppc hw
JTAG chain configuration
-----
Device   ID Code           IR Length   Part Name
  1      f5059093         16         XCF32P
  2      f5059093         16         XCF32P
  3      59608093          8         xc95144xl
  4      0a001093          8         System_ACE
  5      032c6093         10         XC5VFX70T_U

PowerPC440 Processor Configuration
-----
Version.....0x7ff21910
User ID.....0x00f00000
No of PC Breakpoints.....4
No of Read Addr/Data Watchpoints....1
No of Write Addr/Data Watchpoints...1
User Defined Address Map to access Special PowerPC Features using XMD:
  I-Cache (Data).....0x70000000 - 0x70007fff
  I-Cache (TAG).....0x70008000 - 0x7000ffff
  D-Cache (Data).....0x78000000 - 0x78007fff
  D-Cache (TAG).....0x78008000 - 0x7800ffff
  DCR.....0x78020000 - 0x78020fff
  TLB.....0x70020000 - 0x70023fff

Connected to "ppc" target. id = 0
Starting GDB server for "ppc" target (id = 0) at TCP port no 1234
XMD% targets
-----
System(0) - Hardware System on FPGA(Device 5) Targets:
-----
      Target(0) - PowerPC440(1) Hardware Debug Target*
XMD%
```

## ISOCM メモリで実行され、DCR レジスタにアクセスしているプログラムのデバッグセッション例

次に、PowerPC 405 プロセッサの ISOCM メモリ で実行しているプログラムをデバッグするセッションの例を示します。ISOCM アドレス パラメータは、`connect` コマンドで指定できます。XMP ファイルを読み込むと、MHS ファイルからシステムの ISOCM アドレス パラメータが取得されます。

**メモ :** Virtex-4 デバイスでは、ISOCM メモリは読み出し可能です。このオプションを使用すると、ISOCM メモリから実行されるプログラムのデバッグが向上します。

```
XMD% connect ppc hw -debugdevice \
```



```

isocmstartadr 0xFFFFE000 isocmsize 8192 isocmdcrstartadr 0x15 \
dcrstartadr 0xab000000
JTAG chain configuration
-----
Device      ID Code          IR Length    Part Name
  1         0a001093             8    System_ACE
  2         f5059093            16    XCF32P
  3         01e58093            10    XC4VFX12
  4         49608093             8    xc95144x1

PowerPC405 Processor Configuration
-----
Version.....0x20011430
User ID.....0x00000000
No of PC Breakpoints.....4
No of Read Addr/Data Watchpoints....1
No of Write Addr/Data Watchpoints...1
ISOCM.....0xfffffe000 - 0xffffffff
User Defined Address Map to access Special PowerPC Features using XMD:
    I-Cache (Data).....0x70000000 - 0x70003fff
    I-Cache (TAG).....0x70004000 - 0x70007fff
    D-Cache (Data).....0x78000000 - 0x78003fff
    D-Cache (TAG).....0x78004000 - 0x78007fff
    DCR.....0xab000000 - 0xab000fff
    TLB.....0x70004000 - 0x70007fff

XMD% stp
ffffe21c:
XMD% stp
ffffe220:
XMD% bps 0xFFFFE218
Setting breakpoint at 0xfffffe218
XMD% con
Processor started. Type "stop" to stop processor
RUNNING>
8
Processor stopped at PC: 0xfffffe218
XMD%
XMD% mrd 0xab000060 8
AB000060:  00000000
AB000064:  00000000
AB000068:  FF000000 <--- DCR レジスタ : ISARC
AB00006c:  81000000 <--- DCR レジスタ : ISCNTL
AB000070:  00000000
AB000074:  00000000
AB000078:  FE000000 <--- DCR レジスタ : DSARC
AB00007c:  81000000 <--- DCR レジスタ : DSCNTL
XMD%

```

### 特殊な JTAG チェーン設定でのザイリンクス以外のデバイスのデバッグ セッション例

次に、XMD で JTAG チェーンを自動検出できない場合に `-configdevice` オプションを使用してボード上の JTAG チェーンを指定する例を示します。

ザイリンクス以外のデバイスでは、JTAG の IR 幅が不明な場合に XMD で自動検出がうまく機能しないことがあります。JTAG (バウンダリ スキャン) の IR 幅は、通常バウンダリ スキャン記述言語 (BSDL) ファイルに示されています。IR 幅のみが重要な情報で、デバイス名や ID コードなどのフィールドはオプションです。

ここで示す例は、次のように実行されています。

- ザイリンクス パラレル ケーブル (III または IV) の接続には、LPT1 パラレル ポートを使用します。
- JTAG チェーン内の 2 つのデバイスを明示的に示します。
- PROM の IR 幅、デバイス名、および ID コードを指定します。
- `-debugdevice` オプションは、JTAG チェーンの 2 番目のデバイスをデバッグすることを指定します。Virtex デバイスで PowerPC が複数ある場合は、最初の PowerPC であることも指定します。

```
XMD% connect ppc hw -cable type xilinx_parallel port LPT1 -configdevice
devicenr 1 partname PROM_XC18V04 irlength 8 idcode 0x05026093
-configdevice devicenr 2 partname XC2VP4 irlength 10 idcode 0x0123e093
-debugdevice devicenr 2 cpunr 1
```

### ザイリンクス以外のデバイスの追加

XMD は、`${XILINX_EDK}/data/xmd/devicetable.lst` からデバイス情報を読み出します。

XMD にデバイスのサポートを追加するには、次の手順に従います。

1. `devicetable.lst` に、デバイスの ID コード、命令レジスタの幅、および名前情報を追加します。
2. XMD が開いている場合は、閉じてからもう一度開きます。JTAG チェーン内のデバイスが自動的に認識されます。

### シミュレータを使用した PowerPC プロセッサのデバッグ

XMD では、ソケット接続を使用して複数の PowerPC プロセッサ ISS に接続できます。`connect ppc sim` コマンドを使用してローカル ホスト上の PowerPC プロセッサ ISS を起動し、そのホストに接続して、リモートの GDB サーバーを起動します。

`connect ppc sim` コマンドでは、ローカル ホストまたはほかのマシン上で実行中の PowerPC プロセッサ ISS にも接続できます。

XMD が PowerPC プロセッサに接続されたら、XMD を介して `powerpc-eabi-gdb` を PowerPC プロセッサに接続し、デバッグを実行します。

### PowerPC プロセッサ ISS の実行

XMD は、ISS をデフォルト の設定で起動します。

- ISS の実行ファイルは `${XILINX_EDK}/third_party/bin/<platform>/` ディレクトリにあります。
- PowerPC 405 プロセッサのコンフィギュレーション ファイルは `${XILINX_EDK}/third_party/data/iss405.icf` です。
- PowerPC 440 プロセッサのコンフィギュレーション ファイルは `${XILINX_EDK}/third_party/data/iss440.icf` です。

ISS は異なる設定でも実行できます。詳細は、『IBM Instruction Set Simulator User's Guide』を参照してください。146 ページの「関連リソース」に、このマニュアルへのリンクがあります。

ISS のデフォルト設定は、次のとおりです。

- ローカル メモリ バンク 2 個
- XMD デバッガに接続
- デバッガ ポート : 6470...6490
- データ キャッシュ サイズ : 16K
- 命令キャッシュ サイズ : 16K
- 非決定性乗算サイクル
- プロセッサのクロック周期およびタイマのクロック周期 : 5ns (200MHz)

次の表に、ローカル メモリ バンクを示します。

表 11-11 : ローカル メモリ バンク

名前	開始アドレス	幅	スピード
Mem0	0x0	0x80000	0
Mem1	0xff80000	0x80000	0

図 11-3 に、ISS を使用した PowerPC プロセッサのデバッグを示します。

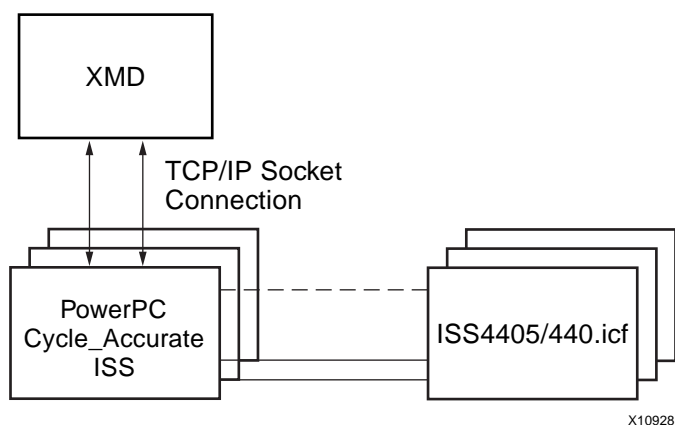


図 11-3 : ISS を使用した PowerPC プロセッサのデバッグ

## 構文

```
connect ppc sim [-debugdevice proctype <ppc440|ppc405>]
[-icf <Configuration File>] [-ipcport IP:<port>]
```

表 11-12 : PowerPC プロセッサ ISS のオプション

オプション	説明
<b>-debugdevice proctype</b> <ppc405 ppc440>	PowerPC プロセッサ タイプを指定します。 ppc405 = PowerPC 405 プロセッサ ppc440 = PowerPC 440 プロセッサ  このオプションを指定しない場合、デフォルトで ppc405 に設定されます。
<b>-icf</b> <Configuration File>	デフォルトのコンフィギュレーション ファイルではなく、指定した ISS コンフィギュレーション ファイルを使用します。キャッシュ サイズ、メモリ アドレス マップ、メモリのレイテンシなどの PowerPC ISS 機能をカスタマイズできます。
<b>-ipcport</b> IP:<port>	起動した PowerPC プロセッサ ISS の IP アドレスとデバッグ ポートを指定します。XMD は ISS を起動しませんが、起動している ISS に接続します。

## ISS を使用した PowerPC プロセッサのデバッグ セッション例

```
XMD% connect ppc sim
Instruction Set Simulator (ISS)
PPC405,
Version 1.9 (1.76)
(c) 1998, 2005 IBM Corporation
Waiting to connect to controlling interface (port=6470,
protocol=tcp)....
[XMD] Connected to PowerPC Sim
Controlling interface connected....
Connected to PowerPC target. id = 0
Starting GDB server for target (id = 0) at TCP port no 1234
XMD% dow dhry2.elf
XMD% bps 0xffff09d0
XMD% con
Processor started. Type "stop" to stop processor

RUNNING>
```

## DCR、TLB、およびキャッシュのアドレス空間とそのアクセス

TLB エントリおよびキャッシュ エントリにアクセスするため、XMD によりアドレス空間が設定されます。これらのアドレスは、connection コマンドのオプションとして tlbstartadr、icachestartadr、および dcachestartadr を使用して指定します。TLB およびキャッシュのアドレス空間が指定されていない場合は、デフォルトの未使用アドレス空間が使用されます。接続すると、次の例に示すように、XMD コンソールにこれらのアドレス空間が表示されます。

```
I-Cache (Data).....0x70000000 - 0x70007fff
I-Cache (TAG).....0x70008000 - 0x7000ffff
D-Cache (Data).....0x78000000 - 0x78007fff
D-Cache (TAG).....0x78008000 - 0x7800ffff
DCR.....0x78020000 - 0x78020fff
TLB.....0x70020000 - 0x70023fff
```

## TLB アクセス

各 TLB エントリは 4 ワードで表されます。次の表に、PPC405 および PPC440 で使用可能な 4 ワード エントリを示します。

表 11-13 : PPC405 および PPC440 の TLB エントリ

ワード	PPC405	PPC440
1	PID	PID
2	TLBHI	TLB Word0 (PID を除く)
3	TLBLO	TLB Word1
4	0 でパディング	TLB Word2

TLB 開始アドレスから 256 ワード、合計 64 個の TLB エントリを読み出しまたは書き込みできます。

## キャッシュ ワード アクセス

キャッシュ エントリは、ウェイごとにアドレス空間にマップされます。前述の例でキャッシュ ライン サイズが 32 バイトで各ウェイが 16 セットで構成される場合、0x70000000 ~ 0x700001FF が命令キャッシュ ウェイ 0 にマップされ、0x70000200 ~ 0x700003FF が命令キャッシュ ウェイ 1 にマップされます。

## キャッシュ タグおよびパリティ アクセス

キャッシュ タグ アドレス空間には、対応するキャッシュ アドレス空間のキャッシュ エントリのタグ、ステータス、およびパリティ情報が含まれています。前述の例では、0x70000100 での命令キャッシュ エントリのタグ情報は 0x70008100 にあり、0x78000600 のデータ キャッシュ エントリのタグ情報は 0x78008600 にあります。

PowerPC 405 プロセッサでは、1 つのキャッシュ ラインのタグおよびステータスを格納するのに 1 ワードが使用され、パリティを格納するのに 1 ワードが使用されます。PowerPC 440 プロセッサでは、1 つのキャッシュ ラインのタグを格納するのに 2 ワード (最初のワードはタグ ロー、2 番目のワードはタグ ハイ) が使用されます。タグ ビットの変換方法については、該当する PowerPC405 または PowerPC440 のユーザー マニュアルの `icread` および `dcread` 命令に関するセクションを参照してください。146 ページの「[関連リソース](#)」に、これらのマニュアルへのリンクがあります。キャッシュ ライン サイズは 32 バイトなので、タグ値は同じキャッシュ ライン内で繰り返されます。

## DCR アドレス空間

DCR バスは PLB バスとは同じアドレス ドメインにありませんが、XMD で DCR バスに PLB アドレス マップを介してアクセスできます。各 DCR アドレスは 1 つの DCR レジスタ (4 バイト) に対応します。DCR レジスタを PLB アドレスにマップするには、4 バイトのアドレス範囲が必要です。172 ページの「[ISS を使用した PowerPC プロセッサのデバッグ セッション例](#)」に示す例では、アドレス マップは次のようになっています。

DCR アドレス	マップされたアドレス
0x0	0x78020000
0x1	0x78020004
0x2	0x78020008
...	...
0x10	0x78020040

## PowerPC プロセッサのデバッグ ヒント (アドバンス)

### ISOCM および ICACHE からのプログラムの実行

PowerPC 405 プロセッサの ISOCM メモリ および 命令キャッシュ (ICACHE) からプログラムをデバッグする際には、ソフトウェア ブレークポイントが使用できないなどの制限があります。このような場合、connect コマンドのオプションとして ISOCM または ICACHE のアドレス範囲が指定されると、XMD によりハードウェア ブレークポイントが自動的に設定されます。ICACHE では、これは ICACHE の内容を固定してプログラムをすべて ICACHE から実行する場合にのみ必要です。

詳細は、XPS ヘルプを参照してください。

PowerPC プロセッサの特殊機能には、XMD コンソール ウィンドウで connect コマンドと該当するオプションを指定するとアクセスできます。

### サードパーティ デバッグ ツールの設定

WindRiver SingleStep や Green Hills Multi などのサードパーティのデバッグ ツールを使用するには、FPGA から PowerPC プロセッサの JTAG 信号 (TCK、TMS、TDI、および TDO) をユーザー I/O として取り出し、ハードウェア ボードの該当するデバッグ コネクタに接続してください。

また、DBGC405DEBUGHALT と C405JTGTDOEN 信号も FPGA から取り出して、ユーザー I/O として接続する必要があります。

PowerPC プロセッサが複数ある場合は、PowerPC プロセッサの JTAG 信号を FPGA 内でチェーン接続することをお勧めします。PowerPC プロセッサ JTAG ポートと FPGA ユーザー I/O の接続の詳細は、『PowerPC 405 Processor Block Reference Guide』および『Embedded Processor Block in Virtex-5 FPGAs Reference Guide』の JTAG ポートのセクションを参照してください。146 ページの「関連リソース」に、このマニュアルへのリンクがあります。

**メモ :** PowerPC プロセッサの JTAG 信号をユーザー I/O として取り出す場合は、JTAGPowerPC モジュールは使用しないでください。

## MicroBlaze プロセッサのデバッグ

XMD では、MDM ペリフェラルを使用して、JTAG を介して複数の MicroBlaze プロセッサに接続できます。XMD は、JTAG またはシリアル インターフェイスを介して、XMDStub などの ROM モニタと通信します。プログラムのデバッグには、ビルトインのサイクル精度 MicroBlaze ISS も使用できます。次のセクションで、これらのターゲットをデバッグする際のオプションについて説明します。

### MDM を使用した MicroBlaze のデバッグ

MDM に接続するには、`connect mb mdm` コマンドを使用し、リモート GDB サーバーを起動します。MDM では、ハードウェア ブレークポイントおよびシングル ステップのデバッグがサポートされており、ROM モニタは必要ありません。

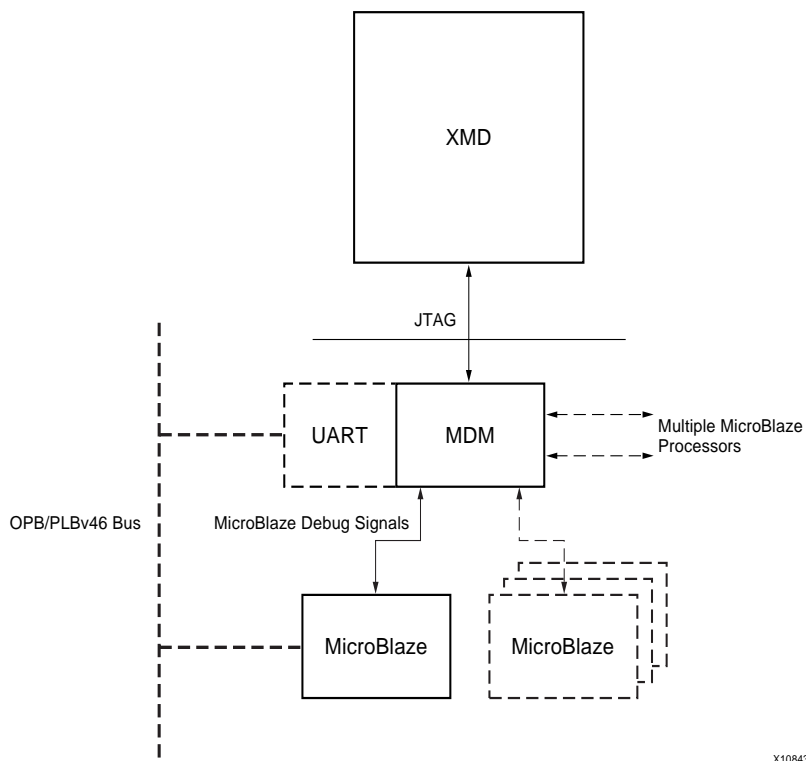


図 11-4 : MDM を使用した MicroBlaze のデバッグ

`connect mb mdm` コマンドでオプションを指定しない場合は、XMD により JTAG ケーブルが自動的に検出され、MicroBlaze と MDM を含む FPGA デバイスがチェーン接続されます。

XMD で JTAG チェーンまたは FPGA デバイスが自動的に検出されない場合は、次のオプションを使用して指定する必要があります。

#### 構文

```
connect mb hw [-cable <JTAG Cable options>] {[-configdevice
<JTAG chain options>]} [-debugdevice <MicroBlaze options>]
```

### -cable および -configdevice のオプション

-cable のオプション (JTAG ケーブル オプション) については [163 ページの表 11-8](#)、  
-configdevice のオプション (JTAG チェーン オプション) については [163 ページの表 11-9](#) を参照してください。

### -debugdevice のオプション (MicroBlaze オプション)

[表 11-14](#) に、-debugdevice のオプション (MicroBlaze オプション) を示します。

表 11-14 : -debugdevice のオプション (MicroBlaze オプション)

オプション	説明
<b>cpunr</b> <CPU Number>	FPGA に MDM に接続された MicroBlaze プロセッサが複数含まれている場合に、デバッグする MicroBlaze プロセッサの番号を指定します。プロセッサの番号は、1 から開始します。
<b>devicenr</b> <MicroBlaze device position>	MicroBlaze プロセッサを含む FPGA デバイスの JTAG チェーン内での位置を指定します。デバイス位置の番号は、1 から開始します。
<b>romemstartadr</b> <ROM start address>	読み出し専用メモリ (ROM) の開始アドレスを指定します。フラッシュ メモリの範囲を指定するには、このオプションを使用します。XMD で、ソフトウェア ブレークポイントではなくハードウェア ブレークポイントが設定されます。
<b>romemsize</b> <ROM Size in Bytes>	読み出し専用メモリ (ROM) のサイズを指定します。
<b>tlbstartadr</b> <TLB start address>	変換ルックアサイド バッファ (TLB) の読み出しおよび書き込みの開始アドレスを指定します。

## MDM を使用した MicroBlaze のデバッグにおける要件

1. ハードウェア ブレークポイント、ステップや停止などの MicroBlaze のハードウェア デバッグ機能を使用するには、ハードウェアのデバッグ ポートを MDM に接続する必要があります。
2. MDM で UART 機能を使用するには、システムに MDM コアをインスタンスエートして、C\_USE\_UART パラメータを設定する必要があります。

**メモ :** MDM を使用する場合は、XMDStub モードではなく executable モードでプログラムをコンパイルする必要があります。XMDStub をコンパイルするための XMDSTUB\_PERIPHERAL を指定する必要はありません。

## デバッグ セッションの例

### MDM を使用した MicroBlaze のデバッグの例

ここでは、MDM を使用した MicroBlaze のデバッグのセッション例を示します。基本的な XMD コマンドは、connect mb mdm コマンドを使用して MDM に接続した後に使用します。セッションの最後に、GDB リモート ターゲットを使用して mb-gdb に接続します。GDB の XMD への接続については、[第 10 章「GNU デバッガ \(GDB\)」](#)を参照してください。



```

XMD% connect mb mdm
JTAG chain configuration
-----
Device      ID Code          IR Length      Part Name
  1         0a001093             8      System_ACE
  2         f5059093            16      XCF32P
  3         01e58093            10      XC4VFX12
  4         49608093             8      xc95144x1

MicroBlaze Processor Configuration:
-----
Version.....7.00.a
Optimisation.....Performance
Interconnect.....PLBv46
No of PC Breakpoints.....3
No of Read Addr/Data Watchpoints...1
No of Write Addr/Data Watchpoints..1
Exceptions Support.....off
FPU Support.....off
Hard Divider Support.....off
Hard Multiplier Support.....on - (Mul32)
Barrel Shifter Support.....off
MSR clr/set Instruction Support....on
Compare Instruction Support.....on
PVR Supported.....on
PVR Configuration Type.....Base

Connected to MDM UART Target
Connected to "mb" target. id = 0
Starting GDB server for "mb" target (id = 0) at TCP port no 1234
XMD% rrd
      r0: 00000000      r8: 00000000      r16: 00000000      r24: 00000000
      r1: 00000510      r9: 00000000      r17: 00000000      r25: 00000000
      r2: 00000140      r10: 00000000      r18: 00000000      r26: 00000000
      r3: a5a5a5a5      r11: 00000000      r19: 00000000      r27: 00000000
      r4: 00000000      r12: 00000000      r20: 00000000      r28: 00000000
      r5: 00000000      r13: 00000140      r21: 00000000      r29: 00000000
      r6: 00000000      r14: 00000000      r22: 00000000      r30: 00000000
      r7: 00000000      r15: 00000064      r23: 00000000      r31: 00000000
      pc: 00000070      msr: 00000004

<--- XMD% コンソールから GDB を起動 --->
XMD% start mb-gdb microblaze_0/code/executable.elf
XMD%

<--- GDB から XMD への接続が確立され、GDB GUI からデバッグを実行 --->
XMD: Accepted a new GDB connection from 127.0.0.1 on port 3791
XMD%
XMD: GDB Closed connection
XMD% stp
BREAKPOINT at
      114: F1440003 sbi      r10, r4, 3
XMD% dis 0x114 10
      114: F1440003 sbi      r10, r4, 3
      118: E0E30004 lbui     r7, r3, 4
      11C: E1030005 lbui     r8, r3, 5
      120: F0E40004 sbi      r7, r4, 4
      124: F1040005 sbi      r8, r4, 5
      128: B800FFCC bri      -52
      12C: B6110000 rtsd     r17, 0
      130: 80000000 Or       r0, r0, r0

```

```

134:  B62E0000  rtid    r14, 0
138:  80000000  Or      r0, r0, r0
XMD% dow microblaze_0/code/executable.elf
XMD% con
Info:Processor started. Type "stop" to stop processor
RUNNING> stop
XMD% Info:User Interrupt, Processor Stopped at 0x0000010c
XMD% con
Info:Processor started. Type "stop" to stop processor
RUNNING> rrd pc
pc : 0x000000f4 <--- MDM では、プログラムの実行中に MicroBlaze の現在の PC を
読み出し可能
RUNNING> rrd pc
pc : 0x00000110 <--- プログラムの実行中は PC が常に変化
RUNNING> stop
Info:Processor started. Type "stop" to stop processor
XMD% rrd
r0: 00000000      r8: 00000065      r16: 00000000      r24: 00000000
r1: 00000548      r9: 0000006c      r17: 00000000      r25: 00000000
r2: 00000190      r10: 0000006c      r18: 00000000      r26: 00000000
r3: 0000014c      r11: 00000000      r19: 00000000      r27: 00000000
r4: 00000500      r12: 00000000      r20: 00000000      r28: 00000000
r5: 24242424      r13: 00000190      r21: 00000000      r29: 00000000
r6: 0000c204      r14: 00000000      r22: 00000000      r30: 00000000
r7: 00000068      r15: 0000005c      r23: 00000000      r31: 00000000
pc: 0000010c      msr: 00000000
XMD% bps 0x100
Setting breakpoint at 0x00000100
XMD% bps 0x11c hw
Setting breakpoint at 0x0000011c
XMD% bpl
SW BP: addr = 0x00000100, instr = 0xe1230002 <-- ソフトウェア ブレークポイント
HW BP: BP_ID 0 : addr = 0x0000011c <-- ハードウェア ブレークポイント
XMD% con
Info:Processor started. Type "stop" to stop processor
RUNNING>
Processor stopped at PC: 0x00000100
Info:Processor stopped. Type "start" to start processor
XMD% con
Info:Processor started. Type "stop" to stop processor
RUNNING>
Info:Processor started. Type "stop" to stop processor

```

## XMDStub を使用した MicroBlaze のデバッグ

MicroBlaze に接続するには、XMDStub (プロセッサ上で実行される ROM モニタ) を使用して GDB サーバーを起動します。XMD は、JTAG またはシリアル インターフェイスを使用して XMDStub に接続します。デフォルト オプションでは、JTAG インターフェイスを使用して接続されます。

## XMDStub/JTAG を使用した MicroBlaze のデバッグ用のコマンド オプション

### 構文

```

connect mb stub -comm jtag [-cable <JTAG Cable options>]
[-configdevice <JTAG chain options>] [-debugdevice
<MicroBlaze options>]

```

### -cable および -configdevice のオプション

-cable のオプション (JTAG ケーブル オプション) については [163 ページの表 11-8](#)、  
-configdevice のオプション (JTAG チェーン オプション) については [163 ページの表 11-9](#) を参照してください。

### -debugdevice のオプション (MicroBlaze オプション)

表 11-15 : -debugdevice のオプション (MicroBlaze オプション)

オプション	説明
<b>devicenr</b> <MicroBlaze device position>	MicroBlaze を含む FPGA デバイスの JTAG チェーン内での位置を指定します。

## XMDStub/シリアル インターフェイスを使用した MicroBlaze のデバッグ用のコマンド オプション

### 構文

```
connect mb stub -comm serial <Serial Communication options>
```

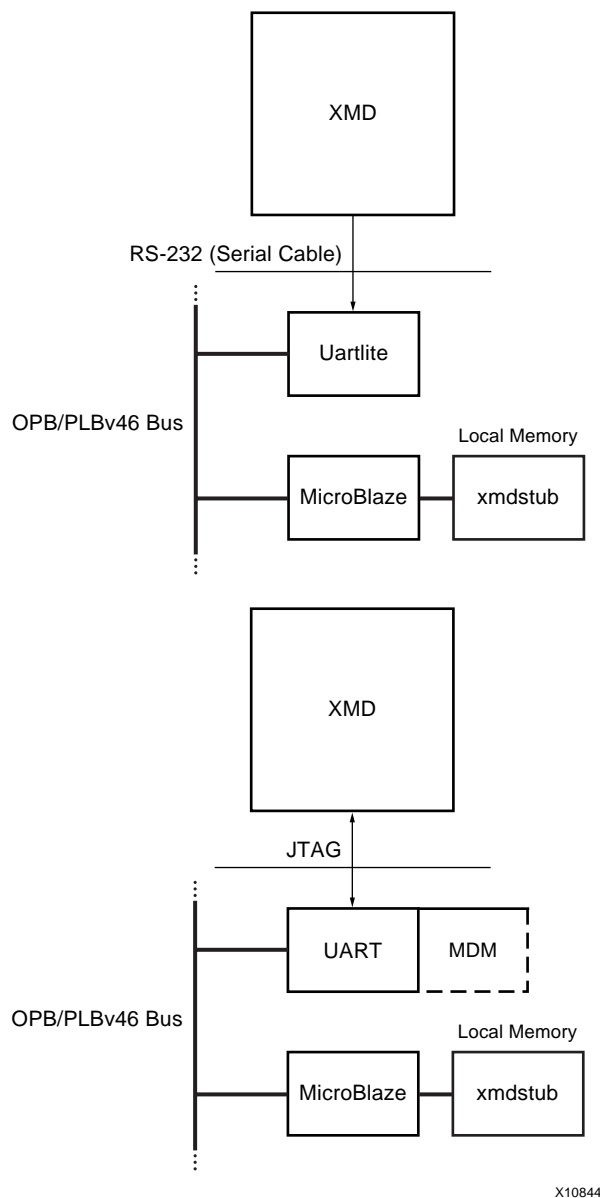
### -comm serial のオプション (シリアル通信オプション)

XMDStub/シリアル インターフェイスを使用した MicroBlaze のデバッグで使用するオプションを、次に示します。

表 11-16 : XMDStub/シリアル インターフェイスを使用した MicroBlaze のデバッグ用のコマンド オプション

オプション	説明
<b>-baud</b> <serial port baud rate>	シリアル ポートのボー レートを bps で指定します。 デフォルト値は 19200bps です。
<b>-port</b> <serial port>	XMD 接続がシリアル ケーブルを介している場合に、 リモート ハードウェアが接続されているシリアル ポートを指定します。  デフォルトのシリアル ポートは、次のとおりです。 <ul style="list-style-type: none"> <li>Linux : /dev/ttyS0</li> <li>Windows : Com1</li> </ul>
<b>-timeout</b> <timeout in secs>	XMD コマンドが XMDStub からの応答を待つ時間を 秒で指定します。

**メモ** : プログラムに UART または MDM UART に出力を書き出す print() または putnum() などの I/O 関数がある場合は、XMD を起動したコンソールまたはターミナルに表示されます。ライブラリおよび I/O 関数については、[第 4 章「Library Generator \(Libgen\)」](#)を参照してください。



X10844

図 11-5 : XMDStub を使用し、JTAG UART および Uartlite を介した MicroBlaze のデバッグ

### XMDStub を使用したデバッグにおける要件

XMD を使用したハードウェア ボード上のプログラムのデバッグは、次のように行われます。

- XMD は、JTAG またはシリアル接続を使用してボード上の XMDStub と通信します。そのため、ターゲットの MicroBlaze システムの MSS ファイルで MDM または UART を XMDSTUB\_PERIPHERAL として指定する必要があります。

Platform Generator では、MHS ファイルで指定すると、MDM または UART を含むシステムを作成できます。XMDStub モードでサポートされる JTAG ケーブルは、次のとおりです。

- ◆ ザイリンクス パラレル ケーブル
- ◆ プラットフォーム USB ケーブル

- ボード上の XMDStub は、MDM または UART を使用してホスト コンピュータと通信するので、MicroBlaze システムで MDM または UART を使用できるよう設定する必要があります。

Library Generator (Libgen) では、XMDStub をシステムの XMDSTUB\_PERIPHERAL を使用するように設定できます。Libgen により XMDStub が XMDSTUB\_PERIPHERAL を使用するように設定され、MSS ファイルの XMDStub 属性で指定されるように code/xmdstub.elf に組み込まれます。詳細は、第 4 章「Library Generator (Libgen)」を参照してください。

- XMDStub の実行ファイルは、システムの起動時に MicroBlaze のローカル メモリに含まれている必要があります。

XMDStub の MicroBlaze メモリへの書き込みは、Data2MEM により行われます。Libgen により、MicroBlaze システムのビットストリームの内容をブロック RAM に書き込むための Data2MEM スクリプトが生成されます。これには、DEFAULT\_INIT で指定されている実行ファイルが使用されます。

- デバッグするためにボードにダウンロードする必要のあるプログラムは、開始アドレスを 0x800 より大きくし、crt1.o のスタートアップ コードとリンクする必要があります。

mb-gcc を -xl-mode-xmdstub オプションを使用して実行すると、これらの条件を満たしてプログラムをコンパイルできます。

**メモ：**ソース レベルのデバッグでは、プログラムのコンパイルに -g オプションも使用する必要があります。C プログラムの機能を初めて検証する場合は、-O2 または -O3 などの mb-gcc の最適化オプションを使用しないことをお勧めします。これらのオプションを使用すると、最適化によりコードが大幅に変わる場合があるので、デバッグが困難になります。

## シミュレータを使用した MicroBlaze のデバッグ

mb-gdb および XMD を使用して、XMD に組み込まれたサイクル精度シミュレータ上でプログラムをデバッグできます。

### 構文

```
connect mb sim [-memsize <size>]
```

### シミュレータを使用した MicroBlaze のデバッグのオプション

表 11-17：シミュレータを使用した MicroBlaze のデバッグのオプション

オプション	説明
-memsize <size>	シミュレータに割り当てるメモリ アドレス バスの幅を指定します。プログラムは、0 ~ (2 <sup>size</sup> ) - 1 のメモリ範囲にアクセスできます。デフォルトのメモリ サイズは 64KB です。

## シミュレータを使用したデバッグにおける要件

XMD を使用してサイクル精度 ISS (命令セット シミュレータ) でプログラムをデバッグするには、プログラムをデバッグ用にコンパイルし、crt0.o のスタートアップ コードにリンクする必要があります。

mb-gcc を -g オプションを使用して実行すると、プログラムがデバッグ用にコンパイルされ、デフォルトですべてのプログラムが crt0.o にリンクされます。

オプションは -xl-mode-executable です。

プログラムのメモリ サイズは 64KB 以下にし、アドレス 0 で開始します。プログラムは、メモリの最初の 64KB に保存する必要があります。

**メモ:** シミュレータを使用したデバッグでは、OPB ペリフェラルのシミュレーションはサポートされていません。

## MDM ペリフェラルおよび UART を使用したデバッグ

MDM ペリフェラルに接続し、UART インターフェイスを使用して、デバッグおよびシステムから情報を収集できます。

### 構文

```
connect mdm -uart
```

### MDM を使用したデバッグにおける要件

MDM で UART 機能を使用するには、システムに MDM をインスタンス化して、C\_USE\_UART パラメータを設定する必要があります。

**xuart w <byte>** コマンドを使用しても、ホストから MicroBlaze 上のプログラムに UART 入力を供給できます。**terminal** コマンドを使用すると、ハイパーターミナルのようなインターフェイスが開き、UART インターフェイスからの読み出しおよび書き込みを実行できます。**read\_uart** コマンドを使用すると、STDIO またはファイルに書き込むインターフェイスが提供されます。

## デバッグ セッションの設定

デバッグ セッションを設定するには、**debugconfig** コマンドを使用します。デバッガの命令の進行方法、メモリのアクセス方法などを設定できます。

### 構文

```
debugconfig [-step_mode {disable_interrupt|enable_interrupt}]
[-memory_datawidth_matching {disable|enable}] [-reset_on_run {system
enable | processor enable | disable}]
```

表 11-18 : debugconfig のオプション

オプション	説明
オプションなし	現在のセッションの現在のデバッグ設定を表示します。
<b>-step_mode</b> {disable_interrupt  enable_interrupt}	<p>命令ステップ モードを指定します。</p> <ul style="list-style-type: none"> <li><b>disable_interrupt</b> : 命令の進行中の割り込みを無効にします。これがデフォルト モードです。</li> <li><b>enable_interrupt</b> : 命令の進行中の割り込みを有効にします。</li> </ul> <p>命令中に割り込みが発生すると、その割り込みはプログラムの割り込みハンドラにより処理されます。</p>

表 11-18 : debugconfig のオプション (続き)

オプション	説明
<b>-memory_datawidth_matching</b> {disable enable}	<p>メモリの読み出しおよび書き込みの処理方法を指定します。デフォルトでは、enable に設定されています。</p> <p>すべてのデータ幅 (バイト、ハーフ ワード、ワード) が、最適な方法で処理されます。このオプションは、データ幅アクセスに厳密に従う必要のあるメモリ コントローラやフラッシュ メモリなどで特に便利です。</p> <p>このオプションを disable に設定にすると、XMD によりワード アクセスなどの最適なデータ幅アクセス方法が選択されます。</p>
<b>-reset_on_run</b> {system enable   processor enable   disable}	<p>プログラム実行時のリセットの処理方法を指定します。リセットを実行すると、システムが既知の一定したステートになり、プログラムが以前の実行の影響を受けずに正しく実行されます。デフォルトでは、プログラムのダウンロードおよびプログラムの実行時にシステム リセットが実行されます。</p> <ul style="list-style-type: none"> <li>異なるリセット タイプを設定するには、次のように入力します。  <pre>debugconfig -reset_on_run processor enable</pre> <pre>debugconfig -reset_on_run system enable</pre> </li> <li>リセットをディスエーブルにするには、次のように入力します。  <pre>debugconfig -reset_on_run disable</pre> </li> </ul>

### 命令ステップ モードの設定

XMD では 2 つの命令ステップ モードがサポートされており、debugconfig コマンドで指定できます。サポートされている 2 つのモードは、次のとおりです。

- 割り込み無効

命令の進行中の割り込みは無効です。これがデフォルト モードです。

- 割り込み有効

命令進行中の割り込みは有効です。次の命令にブレークポイントが設定され、プロセッサが実行されます。

命令中に割り込みが発生すると、その割り込みは割り込みハンドラにより処理されます。プログラムは次の命令で停止します。

**メモ:** プログラムの命令メモリは、プロセッサの D 側インターフェイスに接続する必要があります。

```
XMD% debugconfig
Debug Configuration for Target 0
-----
Step Mode..... Interrupt Disabled
Memory Data Width Matching... Disabled

XMD% debugconfig -step_mode enable_interrupt
XMD% debugconfig
Debug Configuration for Target 0
-----
Step Mode..... Interrupt Enabled
Memory Data Width Matching... Disabled
```

## メモリ アクセスの設定

XMD では、異なるメモリ データ幅へのアクセスがサポートされています。サポートされるデータ幅は、ワード (32 ビット)、ハーフ ワード (16 ビット)、およびバイト (8 ビット) です。デフォルトでは、メモリの読み出しおよび書き込み処理に適切なデータ幅が XMD により選択されます。debugconfig コマンドを使用すると、データ幅がメモリ処理のデータ幅と一致するよう設定できます。この設定は、異なるデータ幅のフラッシュ デバイスにアクセスする場合に必要です。

```
XMD% debugconfig
Debug Configuration for Target 0
-----
Step Mode..... Interrupt Disabled
Memory Data Width Matching... Enabled

XMD% debugconfig -memory_datawidth_matching disable
XMD% debugconfig
Debug Configuration for Target 0
-----
Step Mode..... Interrupt Disabled
Memory Data Width Matching... Disabled
```

## マルチプロセッシング システムでのリセットの設定

デフォルトでは、プロセッサにプログラムがダウンロードされるとシステム リセットが実行されます。これは、プログラムを実行する前にプロセッサのステートを既知のものにするためです。マルチプロセッシング システムでは、プログラムのダウンロードおよび実行が複数のプロセッサで順次発生します。

システム アーキテクチャによっては、ダウンロード時に実行されるシステム リセットにより、別のプロセッサにダウンロードされたプログラムがリセットされてしまうことがあります。この動作が好ましい場合とそうでない場合があるので、debugconfig コマンドを使用してシステム リセットをディセーブルにしたり、特定のプロセッサのみでリセットがイネーブルになるように設定します。

次に例を示します。

### 例 1 : 1 つのマスタ プロセッサと複数のスレーブ プロセッサ

この例では、マスタ プロセッサ上のプログラムが最初にダウンロードされて実行され、その後ほかのプロセッサへのダウンロードが実行されます。この場合、マスタ プロセッサへのダウンロード時のシステム リセットをイネーブルにし、ほかのプロセッサへのダウンロード時にはプロセッサ リセットのみをイネーブルにするか、またはリセットなしにします。



## 例 2 : 2 つの同等のプロセッサ

この例では、ダウンロード シーケンスは任意にでき、両方のプロセッサでプロセッサ リセットのみをイネーブルにするか、またはリセットなしにします。このようにすると、一方のプロセッサへのプログラムのダウンロードがもう一方のプロセッサのシステム ステートに影響を及ぼすことはありません。

このモデルにおけるリセットの接続とシーケンスについては、`proc_sys_reset` IP モジュールの資料を参照してください。

## XMD 内部 Tcl コマンド

Tcl インターフェイス モードでは、XMD コマンドが追加された Tcl シェルが起動します。XMD Tcl コマンドはすべて `x` という文字で開始し、XMD で「`x?`」と入力すると表示されます。

これらの内部コマンドには、146 ページの図 11-1 に示す Tcl ラップアを使用することをお勧めします。Tcl ラップアファイルによりこれらのコマンドほとんどの出力が表示され、より多くのオプションが提供されます。Tcl ラップアは以前のバージョンでも機能しますが、`x<name>` コマンドは将来の EDK リリースでは使用できなくなります。

このセクションには、次の Tcl コマンド サブセクションが含まれます。

- 「プログラム初期化オプション」
- 「レジスタ/メモリのオプション」
- 「プログラム制御オプション」
- 「プログラム トレースおよびプロファイル オプション」
- 「その他のコマンド」

### プログラム初期化オプション

表 11-19 : プログラム初期化オプション

オプション	説明
<b>xconnect</b> <target> {mb ppc mdm} <connect type> [options]	プロセッサまたはペリフェラルに接続します。有効なターゲット タイプは、mb、ppc、および mdm です。  オプションの詳細は、162 ページの「 <a href="#">connect コマンドのオプション</a> 」を参照してください。
<b>xdebugconfig</b> <target id> [-step_mode <Step Type>] [-memory_datawidth_matching {disable enable}] [-reset_on_run {system enable   processor enable   disable}]	ターゲットのデバッグ セッションを設定します。詳細は、182 ページの「 <a href="#">デバッグ セッションの設定</a> 」を参照してください。
<b>xdisconnect</b> <target id>	ターゲットとの接続を解除します。

表 11-19 : プログラム初期化オプション (続き)

オプション	説明
<b>xdownload</b> <target id> <filename> [load address]  <b>xdownload</b> <target id> -data <filename> <load address>	<p>指定した ELF ファイルまたはデータ ファイル (-data オプションを使用した場合) を現在のターゲットのメモリにダウンロードします。</p> <p>ELF ファイルと共にアドレスを指定しない場合、ダウンロードするアドレスは ELF ファイルのヘッダから判断されます。</p> <p>アドレスを指定した場合は、位置独立コード (PIC コード) として処理されて指定のアドレスにダウンロードされ、PIC コードのセマンティクスに従って、レジスタ R20 が開始アドレスに設定されます。</p> <p>XMD では、XMDStub 領域 (アドレス 0x0 ~ 0x800) への書き込みを禁止する目的以外では、バウンド チェックは行われません。</p>
<b>xload_sysfile</b> <XMP System file>	XMP ファイルを読み込みます。
<b>xrut</b> [Session ID]	XMD ソケット インターフェイスを介して通信する場合に、XMD セッションを認証します。まずセッション ID が割り当てられ、続く呼び出しによりセッション ID が返されます。
<b>xtargets</b> -listSysID <b>xtargets</b> -system <system_ID> [-print] [-listTgtID] <b>xtargets</b> -target <target_ID> {-print   -prop}	<p>現在の XMD セッションのシステムおよびターゲット 情報を表示します。</p> <ul style="list-style-type: none"> <li>-listSysID: 既存のシステムをリスト表示します。</li> <li>-system &lt;system_ID&gt;: 指定したシステムの情報を表示します。 <ul style="list-style-type: none"> <li>-print: システムの異なるターゲットを印刷します。</li> <li>-listTgtID: システムの既存のターゲットをリスト表示します。</li> </ul> </li> <li>-target &lt;target_ID&gt;: 指定したターゲットの情報を表示します。 <ul style="list-style-type: none"> <li>-print: ターゲットの情報を印刷します。</li> <li>-prop: ターゲットのプロパティを表示します。</li> </ul> </li> </ul>

## レジスタ/メモリのオプション

表 11-20 : レジスタ/メモリのオプション

オプション	説明
<b>xdata_verify</b> <target id> <Binary filename> <load address>	<Binary filename> が <load address> のメモリに正しくダウンロードされたかどうかを確認します。
<b>xdisassemble</b> <inst>	逆アセンブルを実行し、32 ビット 命令を 1 つ表示します。
<b>xelf_verify</b> <target id> [<filename>.elf]	<p>&lt;filename&gt;.elf がメモリに正しくダウンロードされたかどうかを確認します。</p> <p>&lt;filename&gt;.elf を指定しない場合は、最後にダウンロードされた ELF ファイルが確認されます。</p>
<b>xrmmem</b> <target id> <address> {<num of bytes/half/word>} {b h w} <b>xrmmem</b> <target id> -var <Global Variable Name>	<p>メモリ アドレス &lt;address&gt; から指定のワード/バイト数のメモリ ロケーションを読み出します。デフォルトでは、1 バイト読み出します (b)。データ値のリストが返されます。データ タイプは、メモリ アクセスのデータ幅によって異なります。</p>

表 11-20 : レジスタ/メモリのオプション (続き)

オプション	説明
<b>xwmem</b> <target id> <address> {<num of bytes/half/word>} {b h w} <value list>  <b>xwmem</b> <target id> -var <Global Variable Name> <value list>	メモリ アドレス <address> から指定のワード/バイト 数のメモリ ロケーションにデータ値を書き込みます。デフォルト では、1 バイト 書き込みます (b)。
<b>xrreg</b> <target id> [reg]	すべてのレジスタまたはレジスタ <i>reg</i> のみを読み出します。
<b>xwreg</b> <target id> [reg] [value]	レジスタ <i>reg</i> に <i>value</i> で指定した 32 ビットの値を書き込みます。
<b>xstack_check</b> <target id>	現在のターゲット で実行中のプログラムのスタック使用情報を表示します。ターゲット に最後にダウンロードされた ELF ファイルのスタックがチェックされます。

## プログラム制御オプション

表 11-21 : プログラム制御オプション

オプション	説明
<b>xbreakpoint</b> <target id> {<addr function name>} {sw hw}	指定のアドレスまたは関数の開始部分にブレークポイント を設定します。 <b>メモ :</b> XMDStub ターゲットでは、imm 命令の直後の命令にブレークポイントがあると、不定な結果となる場合があります。
<b>xcontinue</b> <target id> [<Execute Start Address>] [-block]	現在の PC または指定のアドレス <Execute Start Address> から実行を続けます。 <ul style="list-style-type: none"> <li>-block オプションを指定した場合は、プロセッサがブレークポイントまたはウォッチポイントで停止すると、コマンドが戻されます。このオプションは、スクリプトで使用すると有益です。</li> </ul>
<b>xcycle_step</b> <target id> [cycles]	PowerPC プロセッサ ISS の 1 クロック サイクル分進めます。cycles を指定した場合は、そのクロック サイクル数だけ進めます。 <sup>a</sup>
<b>xlist</b> <target id>	すべてのブレークポイントのアドレスを表示します。
<b>xremove</b> <target id> {<addr> <function name> <bp id> all}	1 つまたは複数のブレークポイントまたはウォッチポイントを削除します。
<b>xreset</b> <target id> [reset type]	ターゲットをリセットします。188 ページの表 11-22 に示されている信号など、特定のリセット タイプを設定することも可能です。
<b>xrun</b> <target id>	プログラムを開始アドレスから実行します。
<b>xstate</b> <target id>	プロセッサ ターゲットのステート (実行または停止) を返します。
<b>xstep</b> <target id>	MicroBlaze で 1 命令分進めます。imm 命令では、次の命令も実行されます。この操作では、BIP フラグが設定され、割り込みはディスエーブルになります。デバッグで割り込みをイネーブルにするには、xcontinue をブレークポイントと共に使用してください。

表 11-21 : プログラム制御オプション (続き)

オプション	説明
<b>xstop</b> <target id>	プログラムの実行を停止します。
<b>xwatch</b> <target id> {r w} <address> [<data value>]	指定のアドレス <address> に読み出し/書き込みウォッチポイントを設定し、<data value> をチェックします。<data value> を指定しない場合は、ウォッチポイントは任意の値に一致します。アドレスおよびデータは、16 進数フォーマットまたは 2 進数フォーマットで指定できます。

a. このコマンドは、シミュレータを使用する場合のみ使用可能です。

表 11-22 : ハードウェアを使用した MicroBlaze のデバッグで使用される信号

信号名 (値)	説明
マスク不可ブレーク (0x10)	ブレーク信号と同様に機能しますが、BIP フラグが設定されている場合でも機能します。 BIP フラグの詳細は、『MicroBlaze プロセッサ リファレンス ガイド』を参照してください。146 ページの「関連リソース」に、このマニュアルへのリンクがあります。
プロセッサ ブレーク (0x20)	JTAG UART Ext_Brk 信号を使用して、MicroBlaze の Brk 信号を High にします。MicroBlaze の BIP (Break-in-Progress) フラグをセットし、アドレス 0x18 にジャンプします。
プロセッサ リセット (0x80)	JTAG UART Debug_Rst 信号を使用して MicroBlaze をリセットします。
システム リセット (0x40)	OPB Rst を JTAG UART Debug_SYS_Rst 信号を使用して送信し、システム全体をリセットします。

## プログラム トレースおよびプロファイル オプション

表 11-23 : プログラム トレース/プロファイル オプション

オプション	説明
<b>xprofile</b> <target id> [-o <GMON Output File>] <b>xprofile</b> <target id> -config [sampling_freq_hw <value>] [binsize <value>] [profile_mem <start addr>]	プロファイル出力ファイルを生成します。このファイルは、mb-gprof または powerpc-eabi-gprof で使用されます。 プロファイル コンフィギュレーションのサンプリング周波数 (Hz)、ヒストグラムのビン サイズ、収集したプロファイル データのメモリ アドレスを指定します。
<b>xstats</b> <target id> [options]	現在のセッションのシミュレーション統計を表示します。reset オプションを使用すると、シミュレーション統計をリセットできます。 <sup>a</sup>
<b>xtracestart</b> <target id>	トレース情報の収集を開始します。
<b>xtracestop</b> <target id>	トレース情報の収集を停止します。 <sup>a</sup>

a. このコマンドは ISS ターゲット用です。

## その他のコマンド

表 11-24 : その他のコマンド

コマンド	説明
<b>xhelp</b>	XMD コマンドをリスト表示します。
<b>xuart</b> [r w s] [<data>]	MDM の UART がイネーブルの場合に、3 つの UART 操作のいずれかを実行します。このコマンドは、MDM を使用する場合にのみ使用可能です。 <b>xuart r</b> : MDM UART から 1 バイト読み出します。 <b>xuart w</b> <data> : MDM UART に 1 バイト 書き込みます。 <b>xuart s</b> : MDM UART のステータスを読み出します。
<b>xverbose</b>	詳細モードのオン/オフを切り替えます。XMD からのデバッグ情報を表示します。



# System ACE ファイル ジェネレータ (GenACE)

---

この章では、FPGA のビットストリームと ELF (Executable Linked Format) データ ファイルから Xilinx® System ACE™ テクノロジ コンフィギュレーション ファイルを生成する方法について説明します。生成された System ACE ファイルは、次の目的で使用できます。

- FPGA のコンフィギュレーション
- ブロック RAM の初期化
- 有効なプログラムまたはデータを使用した外部メモリの初期化
- 製品システムでのプロセッサの起動

EDK には、XMD (Xilinx Microprocessor Debug) コマンドを使用して ACE ファイルを作成する Tcl スクリプト `genace.tcl` が含まれています。ACE ファイルは、MDM (マイクロプロセッサ デバッグ モジュール) システムを使用して PowerPC® プロセッサおよび MicroBlaze™ 用に生成できます。

この章には、次のセクションが含まれています。

- [必要条件](#)
- [ツール要件](#)
- [GenACE の機能](#)
- [GenACE モデル](#)
- [genace.tcl スクリプト](#)
- [ACE ファイルの作成](#)
- [関連情報](#)

## 必要条件

この章のフローを実行するには、次の条件を満たしている必要があります。

- XMD を使用したプログラムのデバッグと XMD コマンドに精通している。
- EDK の標準的なハードウェアおよびソフトウェア システム モデルに精通している。
- Tcl スクリプトに対する基礎知識がある。

## ツール要件

ACE ファイルを作成するには、次のツールが必要です。

- genace.tcl
- XMD
- iMPACT (ISE® の一部)

## GenACE の機能

GenACE には、次の機能があります。

- MDM ターゲットを含む PowerPC (405 および 440) プロセッサと MicroBlaze プロセッサをサポート
- ハードウェア (ビットストリーム) およびソフトウェア (ELF およびデータ) ファイルから ACE ファイルを作成
- PowerPC (405 および 440) プロセッサと MicroBlaze システムの外部メモリを初期化
- 複数のプロセッサを含むシステムをサポート
- 単一および複数の FPGA デバイス システムのサポート

## GenACE モデル

System ACE CF は、System ACE CF コントローラおよび記憶媒体 (コンパクトフラッシュ カードまたは 1 インチのマイクロドライブ ディスクのいずれか) を必要とする 2 チップ ソリューションです。System ACE CF は、バウンダリ スキャン (JTAG) 命令とバウンダリ スキャン チェーンを使用してデバイスをコンフィギュレーションします。作成された System ACE ファイルでは、System ACE CF ファミリがサポートされます。System ACE ファイルは、SVF (Serial Vector Format) ファイルから生成されます。SVF ファイルは、JTAG 操作を実行するためのプログラム命令とコンフィギュレーション データの両方を含むテキスト ファイルです。

ソフトウェアおよびハードウェアのシステム ファイルの SVF ファイルを作成するには、それぞれ XMD および iMPACT を使用します。SVF ファイルには、ボードの JTAG チェーンと通信するための JTAG 命令およびデータが含まれています。次のような操作を実行する命令およびデータが含まれます。

- iMPACT を使用した FPGA のコンフィギュレーション
- プロセッサ ターゲットへの接続
- プログラムのダウンロードおよび XMD からプログラムの実行

これらの操作は、SVF ファイル フォーマットに記録されます。この SVF ファイルは ACE ファイルに変換され、記憶媒体に書き込まれます。書き込まれた操作は、System ACE コントローラで実行されます。

次は、シンプルなハードウェアとソフトウェア コンフィギュレーションを iMPACT と XMD を使用して ACE ファイルに変換する手順です。

1. iMPACT を使用してビット ストリームをダウンロードします。このビット ストリーム (download.bit) には、システム コンフィギュレーションとブート ループコードが含まれます。
2. デバイスをリセット状態から解除して DONE ピンを High にします。これでプロセッサ システムが起動します。



3. XMD を使用してプロセッサに接続します。
4. 複数のデータ ファイルをブロック RAM または外部メモリにダウンロードします。
5. 複数の実行ファイルをブロック RAM または外部メモリにダウンロードします。最後にダウンロードした ELF ファイルの開始ロケーションが PC により指定されます。
6. PC の命令アドレスから実行を続けます。

System ACE ファイルを作成するフローは、bit → svf、elf → svf、バイナリ データ → svf、svf → ace ファイルです。

次のセクションで、genace.tcl スクリプトで使用可能なオプションを説明します。

## genace.tcl スクリプト

### 構文

```
xmd -tcl genace.tcl [-opt <genace_options_file>] [-jprog {true|false}]
[-target <target_type> {ppc_hw|mdm}] [-hw <bitstream_file>] [-elf
<elf_files>] [-data <data_files> <load_address>] [-board <board_type>]
[-ace <ACE_file>]
```

表 12-1 : genace.tcl スクリプト コマンド オプション

オプション	デフォルト	説明
<b>-ace</b> <ACE_file>	なし	出力 ACE ファイル。ファイルの接頭辞は、入力ファイル (ビットストリーム、ELF、データ ファイルなど) の接頭辞とは異なるものにする必要があります。
<b>-data</b> <data_file> <load_address>	なし	データ/バイナリ ファイルとそのロード アドレスのリスト。ロード アドレスは、10 進数または 16 進数フォーマット (冒頭に 0x が必要) のいずれかで指定します。SVF ファイルが指定されている場合は、そちらが使用されます。
<b>-elf</b> <list_of_Elf_Files>	なし	ダウンロードする ELF ファイルのリスト。SVF ファイルが指定されている場合は、そちらが使用されます。
<b>-hw</b> <bitstream_file>	なし	システムのビットストリーム ファイル。SVF ファイルが指定されている場合は、そちらが使用されます。
<b>-jprog</b> {true false}	false	既存の FPGA コンフィギュレーションをクリアします。 ランタイム コンフィギュレーションを実行する場合は、このオプションを指定しないでください。
<b>-opt</b> <genace_options_file>	なし	GenACE オプションが <genace_options_file> から読み込まれます。
<b>-target</b> <target_type> {ppc_hw mdm}	ppc_hw	ELF またはデータ ファイルをダウンロードするターゲットを指定します。ターゲット タイプは次のとおりです。 <ul style="list-style-type: none"> <li>• <b>ppc_hw</b> : PowerPC (405 および 440) プロセッサ システムに接続します。</li> <li>• <b>mdm</b> : MicroBlaze プロセッサ システムに接続します。この設定では、mdm がシステムにあると想定されます。</li> </ul>

オプションは、OPT ファイルで指定して GenACE スクリプトに渡すことができます。OPT ファイルの構文を表 12-2 に示します。

表 12-2 : GenACE の OPT ファイルのオプション

オプション	デフォルト	説明
# <Some Text>	なし	# で開始する行は、コメントとして処理されます。
-ace <ACE_file>	なし	出力 ACE ファイル。ファイルの接頭辞は、入力ファイル (ビットストリーム、ELF、データ ファイルなど) の接頭辞とは異なるものにする必要があります。
-board <board_type> {user   supported_board_list}	なし	ボードの JTAG チェーン (デバイス、IR 幅、デバッグ デバイスなど) を識別します。オプションは、System ACE コントローラに対して設定します。スクリプトには、定義済みボードのオプションが含まれます。ボード タイプのオプションは次のとおりです。 <ul style="list-style-type: none"> <li>• user : ユーザー特定のボード。OPT ファイルで -configdevice と -debugdevice オプションを指定する必要があります。詳細は、genace.opt ファイルを参照してください。</li> <li>• サポートされるボード タイプは、「Genace.tcl スクリプトでサポートされるターゲット ボード」を参照してください。</li> </ul>
-configdevice (-user ボード タイプの場合のみ)	なし	JTAG チェーンのデバイスに対するコンフィギュレーション パラメータ。 <ul style="list-style-type: none"> <li>• devicenr : JTAG チェーン上でのデバイスの位置</li> <li>• idcode : ID コード</li> <li>• irlength : 命令レジスタ (IR) の幅</li> <li>• partname : デバイス名</li> </ul> デバイスの位置は、System ACE デバイスを基準としたものです。これらの JTAG デバイスは、ボード上で JTAG チェーンを接続する順序で指定する必要があります。
-data <data_file> <load_address>	なし	データ/バイナリ ファイルとそのロード アドレスのリスト。ロード アドレスは、10 進数または 16 進数フォーマット (0x が前に必要) のいずれかで指定します。SVF ファイルが指定されている場合は、そちらが使用されます。

表 12-2 : GenACE の OPT ファイルのオプション (続き)

オプション	デフォルト	説明
<b>-debugdevice</b> <XMD <i>debug device options</i> > <b>[cpu_version</b> <version> <b>[mdm_version</b> <version>]	MB v7 MDM v1	<p>JTAG チェーンでデバッグまたはコンフィギュレーションする PowerPC (405 または 440) プロセッサまたは MicroBlaze を含むデバイスを指定します。</p> <p>&lt;XMD debug device options&gt; には、次のような XMD デバッグデバイス オプションを指定します。</p> <ul style="list-style-type: none"> <li>• JTAG チェーンでのデバイスの位置 (devicenr)</li> <li>• プロセッサの数 (cpunr)</li> <li>• プロセッサ オプション (OCM、キャッシュ アドレスなど)</li> </ul> <p>MicroBlaze システムでは、MicroBlaze v7 および MDM v1 が使用されていると想定されます。</p> <p>ほかのバージョンの MicroBlaze を指定するには、次のように指定します。</p> <pre>cpu_version {microblaze_v5 microblaze_v6 microblaze_v7}</pre> <p>ほかのバージョンの MDM を指定するには、次のように指定します。</p> <pre>mdm_version {mdm_v2 mdm_v3 mdm_v1}</pre>
<b>-elf</b> <list of Elf or SVF files>	なし	ダウンロードする ELF ファイルのリスト。SVF ファイルが指定されている場合は、そちらが使用されます。
<b>-hw</b> <bitstream file>	なし	システムのビットストリーム ファイル。SVF ファイルが指定されている場合は、そちらが使用されます。
<b>-jprog</b>	false	既存の FPGA コンフィギュレーションをクリアします。ランタイム コンフィギュレーションを実行する場合は、このオプションを指定しないでください。
<b>-start_address</b> <processor run address>	最後の ELF ファイルの開始アドレス (ELF ファイルが指定されている場合)、それ以外はなし	プロセッサの実行を開始するアドレスを指定します。データ ファイルが読み込まれ、プロセッサを読み込みアドレスから開始する必要がある場合に有益です。
<b>-target</b> <target type>	ppc_hw	<p>ELF/データ ファイルをダウンロードするターゲットを指定します。ターゲット タイプは次のとおりです。</p> <ul style="list-style-type: none"> <li>• ppc_hw : PowerPC (405 または 440) プロセッサ システムに接続します。</li> <li>• mdm : MicroBlaze システムに接続します。この設定では、mdm がシステムにあると想定されます。</li> </ul>

## 構文

```
xmd -tcl genace.tcl -jprog -target mdm -hw  
<implementation/download.bit> -elf executable1.elf executable2.svf  
-data image.bin 0xfe000000 -board ml507 -ace system.ace
```

genace.opt ファイルの内容は、次のとおりです。

```
-jprog  
-hw implementation/download.bit  
-ace system.ace  
-board ml507  
-target mdm  
-elf executable1.elf executable2.svf  
-data image.bin 0xfe000000
```

## Genace.tcl スクリプトでサポートされるターゲット ボード

Tcl スクリプトでは、次の 3 つのボードがサポートされます。

- ML401 : ボード タイプは ml401 です。このボードの JTAG チェーンには、XCF32P → XC4VLX25 → XC95144XL が含まれます。
- V4LX25 ES を含む ML401 : ボード タイプは ml401\_es です。このボードの JTAG チェーンには、XCF32P → XC4VLX25-ES → XC95144XL が含まれます。
- ML402 : ボード タイプは ml402 です。このボードの JTAG チェーンには、XCF32P → XC4VSX35 → XC95144XL が含まれます。
- ML403 : ボード タイプは ml403 です。このボードの JTAG チェーンには、XCF32P → XC4VFX12 → XC95144XL が含まれます。
- ML405 : ボード タイプは ml405 です。このボードの JTAG チェーンには、XCF32P → XC4VFX20 → XC95144XL が含まれます。
- ML410 : ボード タイプは ml410 です。このボードの JTAG チェーンには、XC4FX60 が含まれます。
- ML411 : ボード タイプは ml411 です。このボードの JTAG チェーンには、XC4FX100 が含まれます。
- ML501 : ボード タイプは ml501 です。このボードの JTAG チェーンには、XC5VLX50 が含まれます。
- ML505 : ボード タイプは ml505 です。このボードの JTAG チェーンには、XC5VLX50T が含まれます。
- ML506 : ボード タイプは ml506 です。このボードの JTAG チェーンには、XC5VSX50T が含まれます。
- ML507 : ボード タイプは ml507 です。このボードの JTAG チェーンには、XC5VFX70T が含まれます。

カスタム ボードを使用する場合は、-configdevice オプションを使用して JTAG チェーンを指定し、OPT ファイルを使用してください。

## ACE ファイルの作成

System ACE ファイルは、次のサブセクションに示すような場合に作成できます。各状況に対し、OPT ファイルの例を示します。OPT ファイルを使用するには、次のように入力します。

```
xmd -tcl genace.tcl -opt genace.opt
```

### カスタム ボード

使用するボードが 196 ページの「Genace.tcl スクリプトでサポートされるターゲット ボード」にリストされていない場合は、ボードの JTAG チェーン コンフィギュレーションを -configdevice オプションを使用して指定できます。

```
-jprog  
-hw implementation/download.bit  
-ace system.ace  
-board user <= メモ: ボード タイプは user  
-configdevice devicenr 1 idcode 0x1266093 irlength 14 partname XC2VP20  
devicenr 2 idcode 0x1266093 irlength 14 partname XC2VP20 <= メモ: JTAG  
チェーンをここで指定  
-target ppc_hw  
-elf executable.elf
```

### 1 つの FPGA デバイス

#### ハードウェアとソフトウェアのコンフィギュレーション

```
-jprog  
-hw implementation/download.bit  
-ace system.ace  
-board ml501  
-target mdm  
-elf executable1.elf executable2.elf
```

#### ハードウェアとソフトウェアのパーシャル リコンフィギュレーション

```
-hw implementation/download.bit  
-ace system.ace  
-board ml501  
-target mdm  
-elf executable1.elf executable2.elf
```

#### ハードウェアのみのコンフィギュレーション

```
-jprog  
-hw implementation/download.bit  
-ace system.ace  
-board ml401
```

#### ハードウェアのみのパーシャル リコンフィギュレーション

```
-hw implementation/download.bit  
-ace system.ace  
-board ml501
```

## ソフトウェアのみのコンフィギュレーション

```
-jprog  
-ace system.ace  
-board ml501  
-target mdm  
-elf executable1.elf
```

## 複数プロセッサ システムの 1 つのプロセッサの ACE 作成

Virtex® ファミリー デバイスの多くでは、システムに PowerPC (405 および 440) プロセッサが 2 つ、または複数の MicroBlaze プロセッサが含まれます。1 つのプロセッサの ACE ファイルを作成するには、`-debugdevice` オプションを使用します。プロセッサ インスタンスは、`cpunr` で指定します。

PowerPC プロセッサ 2 つを含むコンフィギュレーションで、2 番目のプロセッサの ACE ファイルを生成する場合、このコンフィギュレーションの OPT ファイルは次のようになります。

```
-jprog  
-hw implementation/download.bit  
-ace system.ace  
-board user  
-configdevice devicenr 1 idcode 0x1266093 irlength 14 partname XC2VP20  
-debugdevice devicenr 1 cpunr 2 <= メモ : cpunr は 2  
-target ppc_hw  
-elf executable1.elf executable2.elf
```

## 複数のプロセッサを含むシステムのコンフィギュレーション

PowerPC プロセッサが 2 つと MicroBlaze プロセッサが 1 つあり、それぞれに ELF ファイルがロードされているコンフィギュレーションでは、ボード コンフィギュレーションは次のような OPT ファイルで指定されます。

```
-jprog  
-hw implementation/download.bit  
-ace system.ace  
-board user  
-configdevice devicenr 1 idcode 0x1266093 irlength 14 partname XC2VP20  
# Options for PowerPC Processor 1 - Target Type, ELF files & Data files  
-debugdevice devicenr 1 cpunr 1  
-target ppc_hw  
-elf executable1.elf  
# Options for PowerPC Processor 2 - Target Type, ELF files & Data files  
-debugdevice devicenr 1 cpunr 2  
-target ppc_hw  
-elf executable2.elf  
# Options for MicroBlaze Processor - Target Type, ELF files & Data files  
-debugdevice devicenr 1 cpunr 1  
-target mdm  
-elf executable3.elf
```

メモ : OPT ファイルで複数のプロセッサを指定する場合は、ターゲット タイプ、ELF/データ ファイルなどのプロセッサ特定のオプションを `-debugdevice` オプションの後に指定します。プロセッサの `cpunr` は、`-debugdevice` オプションから判断されます。

## 複数の FPGA デバイス

2 つの FPGA デバイスにそれぞれ 1 つのプロセッサと ELF ファイルが使用されるコンフィギュレーションでは、ボード コンフィギュレーションは次のような OPT ファイルで指定されます。

このコンフィギュレーションの場合、ACE ファイルを作成するには、次のような手順に従う必要があります。

1. 1 つ目の FPGA デバイスに対する SVF ファイルを作成します。OPT ファイルは、次のようになります。

```
-jprog
-target ppc_hw
-hw implementation/download.bit
-elf executable1.elf
-ace fpgal.ace
-board user
-configdevice devicenr 1 idcode 0x123e093 irlength 10 partname XC2VP4
-configdevice devicenr 2 idcode 0x123e093 irlength 10 partname XC2VP4
-debugdevice devicenr 1 cpunr 1
```

これにより、fpgal.svf ファイルが作成されます。

2. 2 つ目の FPGA デバイスに対する SVF ファイルを作成します。OPT ファイルは、次のようになります。

```
-jprog
-target ppc_hw
-hw implementation/download.bit
-elf executable2.elf
-ace fpga2.ace
-board user
-configdevice devicenr 1 idcode 0x123e093 irlength 10 partname XC2VP4
-configdevice devicenr 2 idcode 0x123e093 irlength 10 partname XC2VP4
-debugdevice devicenr 2 cpunr 1 <= メモ：devicenr を変更
```

これにより、fpga2.svf ファイルが作成されます。

3. ファイルを fpgal.svf、fpga2.svf の順で連結して final\_system.svf を作成します。
4. impact -batch svf2ace.scr を実行して ACE ファイルを作成します。次の SCR ファイルを使用します。

```
svf2ace -wtck -d -i final_system.svf -o final_system.ace
quit
```

ML561 などの一部のボードでは、FPGA DONE ピンがすべて 1 つにまとめられて接続されています。これらのボードでは、ボード上の FPGA をハードウェア ビットストリームで同時にコンフィギュレーションし、その後ソフトウェア コンフィギュレーションを実行する必要があります。次に、このようなコンフィギュレーション用の ACE ファイルを生成する方法を示します。ML561 ボードを例として使用しています。

すべての FPGA をハードウェア コンフィギュレーションするための SVF ファイルを生成するには、次の手順に従います。

1. 次の内容の SCR ファイル (impact\_download.scr) を作成し、`impact -batch impact_download.scr` コマンドを実行します。

```
setMode -cf
setPreference -pref KeepSVF:True
addCollection -name Temp
addDesign -version 0 -name config0
addDeviceChain -index 0
setCurrentDeviceChain -index 0
setCurrentCollection -collection Temp
setCurrentDesign -version 0
addDevice -position 1 -file "ML561_FPGA1_Download.bit"
addDevice -position 2 -file "ML561_FPGA2_Download.bit"
addDevice -position 3 -file "ML561_FPGA3_Download.bit"
generate
quit
```

これにより、config0.svf ファイルが作成されます。

2. 1 つ目の FPGA デバイスのソフトウェアに対する SVF ファイルを作成します。OPT ファイルは、次のようになります。

```
-jprog
-ace fpga1_sw.ace
-board user
-configdevice devicenr 1 idcode 0x22a96093 irlength 10 partname
xc5vlx50t
-configdevice devicenr 2 idcode 0x22a96093 irlength 10 partname
xc5vlx50t
-configdevice devicenr 3 idcode 0x22a96093 irlength 10 partname
xc5vlx50t
-debugdevice devicenr 1 cpunr 1
-target mdm
-elf executable1.elf
```

これにより、fpga1\_sw.svf ファイルが作成されます。

3. 2 つ目の FPGA デバイスのソフトウェアに対する SVF ファイルを作成します。OPT ファイルは、次のようになります。

```
-jprog
-ace fpga2_sw.ace
-board user
-configdevice devicenr 1 idcode 0x22a96093 irlength 10
partname xc5vlx50t
-configdevice devicenr 2 idcode 0x22a96093 irlength 10
partname xc5vlx50t
-configdevice devicenr 3 idcode 0x22a96093 irlength 10
partname xc5vlx50t
-debugdevice devicenr 2 cpunr 1
-target mdm
-elf executable2.elf
```

これにより、fpga2\_sw.svf ファイルが作成されます。



4. 3 つ目の FPGA デバイスのソフトウェアに対する SVF ファイルを作成します。OPT ファイルは、次のようになります。

```
-jprog  
-ace fpga3_sw.ace  
-board user  
-configdevice devicenr 1 idcode 0x22a96093 irlength 10  
partname xc5vlx50t  
-configdevice devicenr 2 idcode 0x22a96093 irlength 10  
partname xc5vlx50t  
-configdevice devicenr 3 idcode 0x22a96093 irlength 10  
partname xc5vlx50t  
-debugdevice devicenr 3 cpunr 1  
-target mdm  
-elf executable3.elf
```

これにより、fpga3\_sw.svf ファイルが作成されます。

5. ファイルを config0.svf、fpga1\_sw.svf、fpga2\_sw.svf、fpga3\_sw.svf の順で連結して final\_system.svf を作成します。
6. impact -batch svf2ace.scr を実行して ACE ファイルを作成します。次の SCR ファイルを使用します。

```
svf2ace -wtck -d -i final_system.svf -o final_system.ace  
quit
```

## 関連情報

### CF デバイス フォーマット

System ACE コントローラで CF デバイスを読み出すことができるようにするには、次の手順に従います。

1. CF デバイスを FAT16 としてフォーマットします。
2. ルート ディレクトリで Xilinx.sys ファイルを作成します。このファイルには、ACE コントローラで使用されるディレクトリ構造が含まれます。
3. 作成された ACE ファイルを該当するディレクトリにコピーします。詳細は、iMPACT ヘルプを参照してください。



## コマンド ライン モード

---

この章では、XPS コマンド ライン モードについて説明します。次のセクションが含まれています。

- XPS コマンド ライン モードの起動
- 新規プロジェクトの作成 (空のプロジェクト)
- 新規プロジェクトの作成 (既存の MHS ファイルを指定)
- 既存のプロジェクトを開く
- MHS ファイルの読み込み
- プロジェクト ファイルの保存
- プロジェクト オプションの設定
- フロー コマンドの実行
- MHS ファイルの再読み込み
- ソフトウェア アプリケーションの追加
- ソフトウェア アプリケーションの削除
- ソフトウェア アプリケーションへのプログラム ファイルの追加
- ソフトウェア アプリケーションからのプログラム ファイルの削除
- ソフトウェア アプリケーションのオプションの設定
- 特殊なソフトウェア アプリケーションの設定
- 制限

### XPS コマンド ライン モードの起動

ザイリンクス Bash シェルで「xps -nw」というコマンドを実行すると、XPS をコマンド ライン モード (ユーザー インターフェイスを使用しないモード) で起動できます。これは、Linux ベースのプラットフォーム用に環境変数が設定された、Windows プラットフォーム または Linux シェル用の EDK Cygwin シェルです。XPS で指定の処理が実行され、コマンド プロンプトが表示されます。

コマンド ラインから次の操作を実行できます。

- MSS (Microprocessor Software Specification) ファイルおよび makefile の生成
- プロジェクト フローのバッチ モードでの実行
- XMP プロジェクト ファイルの作成
- XPS GUI で作成した XMP ファイルの読み込み

XPS のバッチ モードでは、EDK デザイン データベースに対してクエリを発行できます。クエリ用の Tcl コマンドがあります。-scr オプションを使用すると、Tcl スクリプトを指定できます。XPS への入力として既存の XMP ファイルも使用できます。

## 新規プロジェクトの作成 (空のプロジェクト)

コンポーネントを含まない新規プロジェクトを作成するには、次のコマンドを使用します。

```
xload new <basename>.xmp
```

空の MHS (Microprocessor Hardware Specification) ファイルと対応する MSS ファイルが作成されます。すべてのファイルのベース名は、XMP ファイルと同じです。ディレクトリに同じベース名のプロジェクト ファイルが存在すると、上書きされます。同じベース名の MHS ファイルまたは MSS ファイルが存在する場合は、新規プロジェクトの一部として読み込まれます。

## 新規プロジェクトの作成 (既存の MHS ファイルを指定)

MHS ファイルを指定して新規プロジェクトを作成するには、次のコマンドを使用します。

```
xload mhs <basename>.mhs
```

MHS ファイルが読み込まれ、新規プロジェクトが作成されます。プロジェクト名は MHS ファイルのベース名と同じになります。生成されるすべてのファイルに MHS と同じベース名が付けられます。MHS ファイルが読み込まれると、デフォルトのドライバが XPS に存在する場合は、各ペリフェラル インスタンスにそのドライバが割り当てられます。

## 既存のプロジェクトを開く

既存の XMP プロジェクト ファイルを読み込むには、次のコマンドを使用します。

```
xload xmp <basename>.xmp
```

XMP ファイルが読み込まれます。MSS ファイルは、XMP ファイルで指定されたものが使用されます。指定されていない場合は、同じベース名のファイルがプロジェクト ディレクトリに存在すれば、その MSS ファイルが読み込まれます。ファイルが存在しない場合は、新しい MSS ファイルが作成されます。

## MSS ファイルの読み込み

MSS ファイルを読み込むには、次のコマンドを使用します。

```
xload mss <filename>
```

<filename> を指定しない場合は、そのプロジェクトに割り当てられた MSS ファイルが読み込まれます。MSS ファイルを読み込むと、それ以前の設定は無効になります。たとえば、あるペリフェラル インスタンスに対して MSS ファイルで新しいドライバが指定されている場合、それまでペリフェラルに指定されていたドライバは使用されなくなります。

## プロジェクト ファイルの保存

プロジェクトの MSS、XMP、makefile を保存するには、次のコマンドを使用します。

```
save [mss|xmp|make|proj]
```

save proj を使用すると、XMP ファイルと MSS ファイルが保存されます。makefile を保存するには、save make を使用する必要があります。

## プロジェクト オプションの設定

XPS のプロジェクトのオプションを設定するには、xset コマンドを使用します。xget コマンドを使用すると、オプションの現在の値を表示できます。また、xget コマンドで Tcl 文字列を返し、Tcl 変数として保存することも可能です。xget と xset コマンドで使用可能なオプションを、表 13-1 に示します。

```
xset option <value>
xget option
```

表 13-1 : xset および xget コマンドのオプション

オプション名	説明
<b>arch</b>	ターゲット デバイスのアーキテクチャを指定します。
<b>dev</b>	デバイス名を指定します。
<b>enable_par_timing_error</b> [0 1]	1 に設定すると、PAR のタイミング エラーがイネーブルになります。
<b>fpga_imp_mode</b> [0 1]	使用するインプリメンテーション ツールを指定します。 0 : XFLOW 1 : ISE Xplorer
<b>gen_sim_tb</b> [true false]	シミュレーション モデルのテストベンチを生成します。
<b>hdl</b> [vhdl verilog]	使用する HDL 言語を指定します。
<b>hier</b> [top sub]	デザイン階層を指定します。
<b>mix_lang_sim</b> [true false]	使用するシミュレータで VHDL と Verilog の混合フローがサポートされているかどうかを指定します。
<b>package</b>	ターゲット デバイスのパッケージを指定します。
<b>searchpath</b> <dirs>	検索パスを指定します。複数のディレクトリを指定する場合は、セミコロンで区切ります。
<b>speedgrade</b>	ターゲット デバイスのスピード グレードを指定します。
<b>swapps</b>	ソフトウェア アプリケーションのリストを表示します。このオプションは、xset コマンドでは使用できません。

表 13-1 : xset および xget コマンドのオプション (続き)

オプション名	説明
<b>sim_model</b> [structural behavioral timing]	シミュレーション モードを設定します。
<b>simulator</b> [mti ncsim none]	どのシミュレータ用にシミュレーション スクリプトを生成するかを指定します。
<b>sim_x_lib</b> <b>sim_edk_lib</b>	シミュレーション ライブラリのパスを設定します。これらのパスは、 <b>XMP</b> ファイルではなくユーザーの指定したレジストリに保存されます。詳細は、 <a href="#">第 3 章「Simulation Model Generator (Simgen)」</a> を参照してください。
<b>topinst</b> <instname>	プロセッサ デザインがサブモジュールの場合に、そのデザインのインスタンス名を指定します。
<b>ucf_file</b>	インプリメンテーションで使用するユーザー制約ファイル (UCF) ファイルへのパスを指定します。
<b>usercmd1</b>	ユーザー コマンド 1 を設定します。
<b>usercmd2</b>	ユーザー コマンド 2 を設定します。
<b>user_make_file</b> <directory path>	<b>makefile</b> のパスを指定します。 <b>XPS</b> で生成された <b>makefile</b> とは異なるファイルを指定する必要があります。

## フロー コマンドの実行

**run** コマンドを適切なオプションを使用して実行することにより、さまざまなフロー ツールを実行できます。**XPS** はプロジェクトの **makefile** を作成し、適切なターゲットを使用してその **makefile** を実行します。**makefile** は、**run** コマンドを実行するたびに生成されます。**run** コマンドで使用可能なオプションを [表 13-2](#) に示します。

```
run <option>
```

表 13-2 : run コマンドのオプション

オプション名	説明
<b>ace</b>	<b>BIT</b> ファイルをブロック <b>RAM</b> の情報でアップデートした後、 <b>System ACE</b> テクノロジ ファイルを生成します。
<b>assign_default_drivers</b>	<b>MHS</b> ファイルのすべてのペリフェラルにデフォルトのドライバを割り当て、 <b>MSS</b> ファイルを保存します。
<b>bits</b>	インプリメンテーション ツールを実行して、ビットストリームを生成します。
<b>bitsclean</b>	<b>implementation</b> ディレクトリから <b>BIT</b> ファイル、 <b>NCD</b> ファイル、および <b>BMM</b> ファイルを削除します。
<b>bsp</b>	<b>PowerPC®</b> プロセッサシステムの <b>VxWorks</b> ボード サポート パッケージ ( <b>BSP</b> ) を生成します。

表 13-2 : run コマンドのオプション (続き)

オプション名	説明
<b>clean</b>	ツールで生成されたすべてのファイルとディレクトリを削除します。
<b>download</b>	ビットストリームを <b>FPGA</b> にダウンロードします。
<b>hwclean</b>	<b>implementation</b> ディレクトリを削除します。
<b>init_bram</b>	ビットストリームをブロック <b>RAM</b> 初期化情報でアップデートします。
<b>libs</b>	ソフトウェア ライブラリを生成します。
<b>libsclean</b>	ソフトウェア ライブラリを削除します。
<b>netlist</b>	ネットリストを生成します。
<b>netlistclean</b>	<b>NGC</b> または <b>EDN</b> ネットリストを削除します。
<b>program</b>	プログラムをコンパイルし、 <b>ELF (Executable Linked Format)</b> ファイルを生成します。
<b>programclean</b>	<b>ELF</b> ファイルを削除します。
<b>resync</b>	<b>MHS</b> ファイルの変更をメモリに反映させます。
<b>sim</b>	シミュレーション モデルを生成し、シミュレータを実行します。
<b>simmodel</b>	シミュレーション モデルを生成します。シミュレータは起動しません。
<b>swclean</b>	<b>libsclean</b> および <b>programclean</b> を実行します。
<b>simclean</b>	<b>simulation</b> ディレクトリを削除します。

## MHS ファイルの再読み込み

すべての EDK デザイン ファイルは、**MHS** ファイルを参照します。**MHS** ファイルの変更は、ほかのファイルにも影響します。デザインを読み込んだ後、**MHS** ファイルが変更された場合は、次のコマンドを使用します。

```
run resync
```

このコマンドを実行すると、**MHS** ファイル、**MSS** ファイル、**XMP** ファイルが再度読み込まれます。

## ソフトウェア アプリケーションの追加

**xadd\_swapp** コマンドを使用すると、新しいソフトウェア アプリケーション プロジェクトを追加できます。ソフトウェア アプリケーションを追加するには、アプリケーションの名前と、そのアプリケーションを実行するプロセッサ インスタンスを指定する必要があります。デフォルトでは、ソフトウェア アプリケーションの **ELF** ファイルは **<swapp\_name>/bin/<swapp\_name>.elf** です。アプリケーションを作成した後、ディレクトリを変更できます。

```
xadd_swapp <swapp_name> <proc_inst>
```

## ソフトウェア アプリケーションの削除

既存のソフトウェア アプリケーションをプロジェクトから削除するには、`xdel_swapp` コマンドを使用します。この際、削除するソフトウェア アプリケーションの名前を指定する必要があります。

```
xdel_swapp <swapp_name>
```

## ソフトウェア アプリケーションへのプログラム ファイルの追加

既存のソフトウェア アプリケーションにプログラム ファイル (C ソール ファイルまたはヘッダ ファイル) を追加するには、`xadd_swapp_progfile` コマンドを使用します。この際、ファイルを追加するソフトウェア アプリケーションの名前とプログラム ファイルの場所を指定する必要があります。ファイルの拡張子によって、ソース ファイルまたはヘッダ ファイルかが自動的に判断され、追加されます。

```
xadd_swapp_progfile <swapp_name> <filename>
```

## ソフトウェア アプリケーションからのプログラム ファイルの削除

既存のソフトウェア アプリケーションからプログラム ファイル (C ソール ファイルまたはヘッダ ファイル) を削除するには、`xdel_swapp_progfile` コマンドを使用します。この際、ソフトウェア アプリケーションの名前とプログラム ファイルの場所を指定する必要があります。

```
xdel_swapp_progfile <swapp_name> <filename>
```

## ソフトウェア アプリケーションのオプションの設定

ソフトウェア アプリケーションのオプションを設定するには、`xset_swapp_prop_value` コマンドを使用します。`xget_swapp_prop_value` コマンドを使用すると、オプションの現在の値を表示できます。`xget_swapp_prop_value` コマンドでは、オプションの値を Tcl 文字列として返すこともできます。この 2 つのコマンドで使用可能なオプションを、表 13-3 に示します。

```
xset_swapp_prop_value <swapp_name> <option_name> <value>
xget_swapp_prop_value <swapp_name> <option_name>
```

表 13-3 : `xset_swapp_prop_value` コマンドおよび `xget_swapp_prop_value` コマンドのオプション

オプション名	説明
<b>compileroptlevel</b>	コンパイラの最適化レベルを指定します。有効な値は 0 ～ 3 です。
<b>debugsym</b>	デバッグ シンボルを指定します。有効な値は 0 (なし)、1 (-g)、または 2 (-gstabs) です。
<b>executable</b>	実行ファイル (ELF) へのパスを指定します。
<b>sources</b>	ソース ファイルのリストを表示します。ソース ファイルを追加するには、 <code>xadd_swapp_progfile</code> コマンドを使用してください。
<b>globptropt</b> {true false}	グローバル ポインタ最適化を実行するかどうかを指定します。
<b>headers</b>	ヘッダ ファイルのリストを表示します。ヘッダ ファイルを追加するには、 <code>xadd_swapp_progfile</code> コマンドを使用してください。



表 13-3 : xset\_swapp\_prop\_value コマンドおよび xget\_swapp\_prop\_value コマンドのオプション (続き)

オプション名	説明
<b>heapsize</b>	ヒープ サイズを指定します。
<b>init_bram</b>	ブロック RAM の初期化に ELF ファイルを使用するかどうかを指定します。
<b>lflags</b>	リンクするライブラリを指定します (-l)。
<b>linkerscript</b>	リンカ スクリプトを指定します (-Wl, -T -Wl, <linker_script_file>)。
<b>mode</b>	ELF ファイルのコンパイルを XMDStub モード (MicroBlaze™ のみ) または実行モードのどちらで行うかを指定します。
<b>procinst</b>	ソフトウェア アプリケーションに関連付けられているプロセッサ インスタンスを指定します。
<b>progccflags</b>	上記のオプションで指定できないコンパイラ オプションを設定します。
<b>progstart</b>	プログラムの開始アドレスを指定します。
<b>searchlibs</b>	ライブラリの検索パスを指定します (-L)。
<b>searchincl</b>	インクルード検索パスを指定します (-I)。
<b>stacksize</b>	スタック サイズを指定します。

## 特殊なソフトウェア アプリケーションの設定

各プロセッサ インスタンスには、デフォルトでブートループ アプリケーションが生成されます。MicroBlaze インスタンスに対しては、XMDStub も生成されます。

これらの特殊なソフトウェア アプリケーションに対しては、**BRAM** の初期化に使用するかどうかのみを指定できます。この指定には、xset\_swapp\_prop\_value コマンドを使用します。XPS のコマンド ライン モードでは、<procinst>\_bootloop および <procinst>\_xmdstub という名前を使用すると、特殊なソフトウェア アプリケーションであると認識されます。たとえば、プロセッサ インスタンス名が mymblaze である場合は、mymblaze\_bootloop および mymblaze\_xmdstub がソフトウェア アプリケーションとして認識されます。

これらのアプリケーションに init\_bram オプションを設定するには、次の構文を使用します。

```
XPS% xset_swapp_prop_value mymblaze_bootloop init_bram true
XPS% xset_swapp_prop_value mymblaze_xmdstub init_bram false
```

その他のユーザー ソフトウェア アプリケーションにこれらの名前を使用すると、XPS Tcl インターフェイスでは設定を変更できません。XPS のバッチ モードを使用する場合は、ソフトウェア アプリケーションに <procinst>\_bootloop または <procinst>\_xmdstub という名前を付けないでください。この制限は XPS のバッチ モードにのみ適用されるもので、GUI を使用する場合はこの制限はありません。

## 制限

### MSS ファイルの変更

XPS のバッチ モードでは、MSS ファイルに対して限られた編集操作しかできません。MSS ファイルに変更を加える場合は、ファイルをテキスト エディタで編集し、`xload mss` コマンドを使用して編集した MSS ファイルを読み込むようにしてください。この際、プロジェクトを閉じる必要はありません。MSS ファイルを編集して保存し、`xload mss` コマンドを使用して再読み込みすれば、編集した MSS ファイルが使用されます。

### XMP ファイルの変更

XMP ファイルを手動で変更することはお勧めできません。XPS のバッチ モードでは、コマンドを使用してプロジェクト オプションを変更したり、プロセッサへのソース ファイルおよびヘッダ ファイルの追加、コンパイラ オプションの設定を実行できます。それ以外の変更は、XPS の GUI から行う必要があります。

# ザイリンクス Bash シェル

---

この章では、ザイリンクスの Cygwin ベースの Bash シェルについて説明します。この章には、次のセクションが含まれています。

- 概要
- ザイリンクス Bash シェル

## 概要

ザイリンクス エンベデッド開発キット (EDK) には、コンパイラやデバッガなどの GNU ツールと、make ユーティリティが含まれています。NT プラットフォームでは、これらのツールおよびユーティリティに Linux エミュレーション シェルが必要です。そのため、Red Hat Cygwin™ シェルとユーティリティが EDK の一部として提供されています。

## EDK でインストールされる Cygwin 環境

EDK をインストールすると、Cygwin 環境が %XILINX\_EDK%\cygwin にインストールされます。

## 既存の Cygwin 環境を使用するための要件

既存の Cygwin 環境を使用することも可能ですが、次の要件を満たしている必要があります。

- Cygwin のリビジョン レベルが 1.5.17 (2005 年 5 月) 以降である。
- make ユーティリティ (make.exe) が使用可能である。

既存の Cygwin 環境が上記の要件を満たしている場合は、その Cygwin 環境を使用できます。条件を満たしていない場合は、ザイリンクス Bash シェルを使用する必要があります。その場合、既存の Cygwin のステートに基づいて、エラー メッセージまたは警告メッセージが表示されます。

## ザイリンクス Bash シェル

ザイリンクス Bash シェルは、Cygwin に基づく Linux 環境エミュレーション メカニズムであり、Windows プラットフォーム上で Linux のような環境で EDK ツールおよびほかのユーティリティを実行するために使用します。このシェルを起動するには、Windows の [スタート] メニューから、[プログラム] → [Xilinx ISE Design Suite 11] → [EDK] → [アクセサリ] → [Bash シェル] をクリックします。これにより、xbash ユーティリティ (%XILINX\_EDK%\bin\nt\xbash.exe) が起動します。

## xbash の使用

xbash の使用法は、`xbash -help` コマンドを使用すると表示されます。

構文：

```
xbash [-c <COMMAND>] [-override] [-undo]
```

<b>-c &lt;COMMAND&gt;</b>	ザイリンクス Bash シェルで <COMMAND> を実行します。
<b>-override</b>	ローカルにインストールされている Cygwin ではなく、EDK の Cygwin を使用します。
<b>-undo</b>	-override オプションの変更を元に戻します。
<b>-help</b>	ヘルプを表示します。

既存の Cygwin を使用するには、「[既存の Cygwin 環境を使用するための要件](#)」に記載されている要件が満たされている必要があります。要件が満たされていない場合は、新しいバージョンの Cygwin にアップグレードするか、必要なツールをインストールする必要があることを示すメッセージが表示されます。Cygwin のアップグレードが必要な場合は、`-override` および `-undo` オプションを使用して、EDK Cygwin を使用することも可能です。

## -override オプションと -undo オプション

マシンにインストールされている Cygwin が 1.5.17 より以前のバージョンである場合は、`xbash -override` コマンドを使用します。インストールされている Cygwin のバージョンが 1.5.17 である場合は、このオプションは無視されます。

**メモ：**このオプションは 1 回のみ使用し、Windows/DOS コマンド プロンプトから実行する必要があります。`xbash` コマンドまたはザイリンクス Bash シェルを次に実行する際は、このオプションは必要ありません。

**メモ：**このオプションを使用すると、既存の Cygwin 設定が変更されるので、注意が必要です。このオプションでは必要な Cygwin ツールおよび DLL のみがアップグレードされ、すべてのツールがアップグレードされるわけではありません。

この変更を元に戻して Cygwin の元の状態を回復するには、`xbash -undo` を使用します。

## Windows Vista での Cygwin

Windows Vista プラットフォームで EDK を正しく機能させるには、既存の Cygwin がすべて有効であり、正しい権限でインストールされている必要があります。権限に関する次の要件に注意してください。

- 既存のすべての Cygwin で、マシンのすべてのユーザーに対して実行権限が必要です。
- ローカル マシン上の Cygwin が管理者権限を持つユーザーによりインストールされていて、この Cygwin が無効である場合、管理者権限を持つユーザーがこの Cygwin を修正する必要があります。管理者権限以外の権限のユーザーが修正することはできません。

# GNU ユーティリティ

---

この付録では、EDK で使用可能な GNU ユーティリティについて説明します。この付録には、次のセクションが含まれています。

- [MicroBlaze および PowerPC の汎用ユーティリティ](#)
- [MicroBlaze および PowerPC 専用ユーティリティ](#)
- [その他のプログラムおよびファイル](#)

## MicroBlaze および PowerPC の汎用ユーティリティ

### cpp

C および C++ コードのプリプロセッサ。プリプロセッサは GNU コンパイラ (GCC) により自動的に実行され、`file-include`、`define` などの命令をインプリメントします。

### gcov

GCC と共に使用し、ユーザー プログラムのテスト範囲のプロファイル作成および解析を実行します。gprof プロファイル作成プログラムでも使用されます。

## MicroBlaze および PowerPC 専用ユーティリティ

MicroBlaze™ 専用ユーティリティには、次に示すように接頭辞 `mb` が付いています。接頭辞 `powerpc-eabi` が付いているものは PowerPC® プロセッサ専用ユーティリティで、同じ機能を実行します。

### mb-addr2line

実行ファイルのデバッグ情報を使用して、プログラム アドレスを対応する行番号とファイル名に変換するプログラム。

### mb-ar

アーカイブからファイルを作成、変更、および抽出するプログラム。アーカイブは、通常ライブラリのオブジェクト ファイルなどの複数のファイルを含むファイルです。

### mb-as

アセンブラ プログラム。

## mb-c++

**mb-gcc** と同じクロス コンパイラで、プログラム言語が **C++** に設定されている場合に実行されます。**mb-g++** と同じです。

## mb-c++filt

アセンブリ リストの **C++** および **Java** 関数名をデマングルするプログラム。

## mb-g++

**mb-gcc** と同じクロス コンパイラで、プログラム言語が **C++** に設定されている場合に実行されます。**mb-c++** と同じです。

## mb-gasp

アセンブラ プログラムのマクロ プリプロセッサ。

## mb-gcc

**C** および **C++** プログラムのクロス コンパイラ。ファイル拡張子から、使用されているプログラム言語を自動的に認識します。

## mb-gdb

プログラムのデバッガ。

## mb-gprof

プログラムの各部分にどれだけの時間がかかるかを解析するプロファイル生成プログラム。ランタイムを最適化するのに有益です。

## mb-ld

リンカ プログラム。ライブラリ ファイルとオブジェクト ファイルを結合し、必要なリロケーションを実行して、実行ファイルを生成します。

## mb-nm

オブジェクト ファイルのシンボルをリストするプログラム。

## mb-objcopy

オブジェクト ファイルの内容をあるフォーマットから別のフォーマットに変換するプログラム。

## mb-objdump

オブジェクト ファイルの情報を表示するプログラム。プログラムのデバッグにおいて有益で、正しいコードおよびデータが正しいメモリ ロケーションにあるかどうかを検証するのに使用されます。

## mb-ranlib

アーカイブ ファイルのインデックスを作成し、アーカイブ ファイルに追加するプログラム。これにより、アーカイブで示されるライブラリへのリンク プロセスを高速化できます。

## mb-readelf

ELF (Executable Linked Format) ファイルの情報を表示するプログラム。

## mb-size

オブジェクト ファイルの各セクションのサイズをリストするプログラム。コードおよびデータのスタティック メモリ要件を判断するのに有益です。

## mb-strings

バイナリ ファイルの内容を判断するのに有益なプログラム。オブジェクト ファイルに含まれる表示可能な文字列をリストします。

## mb-strip

オブジェクト ファイルからシンボルを削除するプログラム。ファイル サイズを削減し、ファイル内のシンボル情報が見られないようにするために使用します。

## その他のプログラムおよびファイル

次の Tcl および Tk シェルは、さまざまなフロント エンド プログラムから起動されます。

- cygitclsh30
- cygitkwish30
- cygtclsh80
- cygwish80
- tix4180





## 割り込み制御

---

この付録では、ザイリンクス エンベデッド ハードウェア システムでの割り込みの設定方法、割り込み中のソフトウェア フローの制御、割り込みを制御するソフトウェア API について説明します。これらの説明は、ハードウェア割り込みとその実用性を理解している場合に有益です。次のセクションが含まれています。

- [関連リソース](#)
- [ハードウェアの設定](#)
- [ソフトウェアの設定と割り込みフロー](#)
- [ソフトウェア API](#)

### 関連リソース

- 『Platform Specification Format Reference Manual』  
[http://japan.xilinx.com/ise/embedded/edk\\_docs.htm](http://japan.xilinx.com/ise/embedded/edk_docs.htm)
- 『PowerPC Processor Reference Guide』  
[http://japan.xilinx.com/ise/embedded/edk\\_docs.htm](http://japan.xilinx.com/ise/embedded/edk_docs.htm)
- 『OS and Libraries Document Collection』  
[http://japan.xilinx.com/ise/embedded/edk\\_docs.htm](http://japan.xilinx.com/ise/embedded/edk_docs.htm)
- アプリケーション ノート XAPP 778 『Using and Creating Interrupt-Based Systems』  
[http://japan.xilinx.com/support/documentation/topicembedprocess\\_processorcore.htm](http://japan.xilinx.com/support/documentation/topicembedprocess_processorcore.htm)
- ザイリンクス デバイス ドライバの資料 (EDK のインストール ディレクトリ内)  
`/doc/japanese/xilinx_drivers.htm`

## ハードウェアの設定

プロセッサで割り込みを受信できるようにするには、まずハードウェアで割り込みを配線する必要があります。

MicroBlaze™ プロセッサには、Interrupt という外部割り込みポートが 1 つあります。PowerPC® 405 プロセッサおよび PowerPC 440 プロセッサには、割り込みを処理するポートが 2 つあります。1 つのポートはクリティカル カテゴリの外部割り込みを生成し、もう 1 つのポートは非クリティカル カテゴリの外部割り込みを生成します。この 2 つのカテゴリの違いは、システムのほかの割り込みおよび例外に対する優先順位です。クリティカル カテゴリが最も優先されます。

PowerPC 405 プロセッサではクリティカル割り込みポートは EICC405CRITINPUTIRQ、非クリティカル割り込みポートは EICC405EXTINPUTIRQ であり、

PowerPC 440 プロセッサではクリティカル割り込みポートは EICC440CRITIRQ、非クリティカル割り込みポートは EICC440EXTIRQ です。

プロセッサに割り込みを配線するには、次の 2 つの方法があります。

- 割り込みペリフェラルからの割り込み信号をプロセッサの割り込みポートに直接接続する。このコンフィギュレーションでは、プロセッサに割り込みことができるのは 1 つのペリフェラルのみです。
- 割り込みペリフェラルからの割り込み信号を割り込みコントローラ コアに接続し、その割り込みコントローラ コアによりプロセッサの割り込みポートに接続した信号に対して割り込みを生成する。このコンフィギュレーションでは、プロセッサに複数のペリフェラルから割り込み信号を送信できます。エンベデッド システムには割り込み機能にアクセスする必要のあるペリフェラルは通常複数あるので、この方法の方がより一般的です。

次の図に、割り込みコンフィギュレーションを図示します。

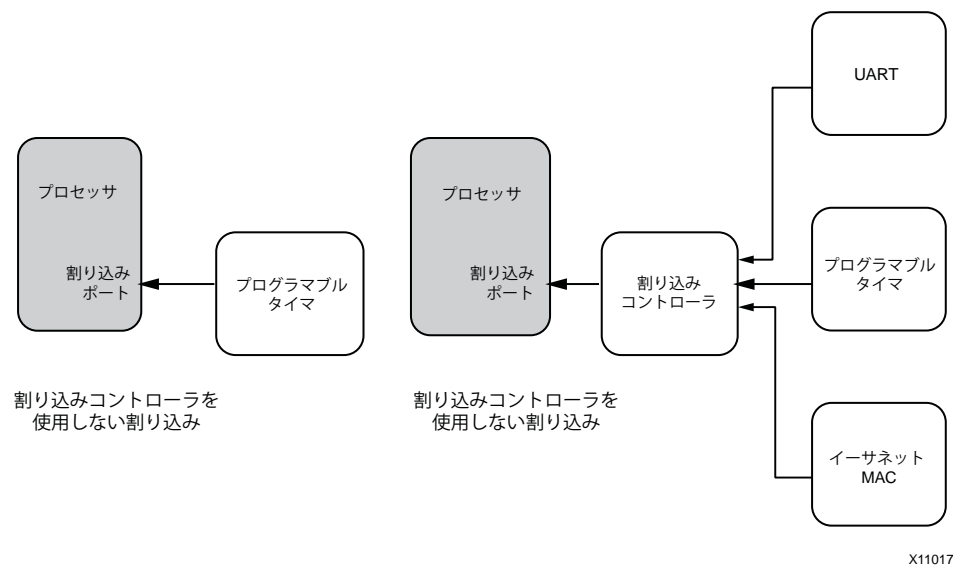


図 B-1 : 割り込みコンフィギュレーション

## ソフトウェアの設定と割り込みフロー

アプリケーションの割り込みハンドラが実行されるまでに、割り込みはソフトウェア プラットフォームの複数のレベルを通過します。ザイリンクス ソフトウェア プラットフォーム (スタンドアロンおよび Xilkernel) では、次の図に示す割り込みフローが使用されます。

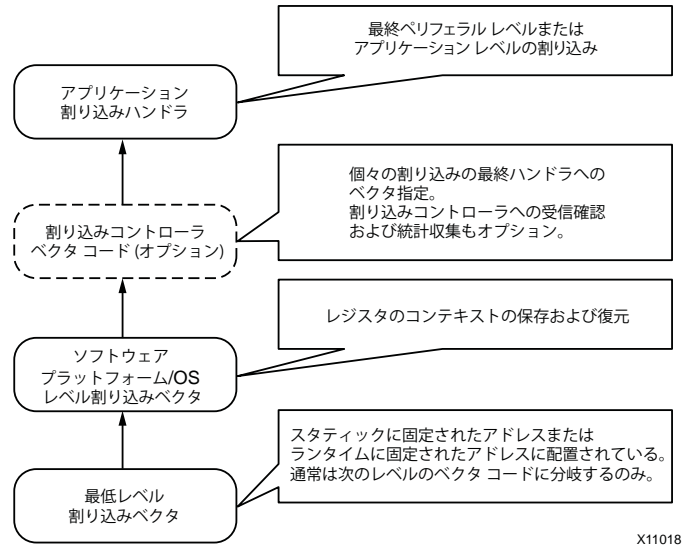


図 B-2: 割り込みフロー

## MicroBlaze システムの割り込みフロー

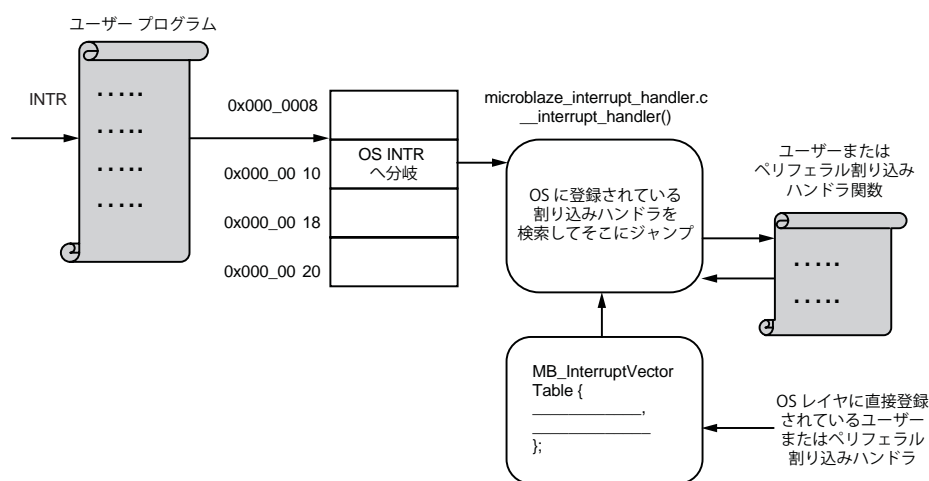
MicroBlaze の割り込みフローは、次のとおりです。

1. マシン ステータス レジスタ (MSR) で適切なビットを設定することにより、MicroBlaze で割り込みをイネーブルにします。
2. 外部割り込み信号が受信されると、プロセッサでそれ以外の割り込みがディスエーブルになり、プロセッサが固定アドレス 0x0000\_0010 にジャンプします。
3. ソフトウェア プラットフォームまたは OS によりこのアドレスにベクタコードが供給されており、このコードにより制御がメイン プラットフォーム割り込みハンドラに移行します。
4. プラットフォーム割り込みハンドラにより、プロセッサ レジスタの内容 (上書きされる可能性がある) が現在のアプリケーション スタックに保存され、制御が次のレベルのハンドラに移行します。次のレベルのハンドラは、システムに割り込みコントローラが存在するかどうかによって異なるので、内部割り込みベクタ テーブルを参照して次のレベルのハンドラの関数アドレスが判断されます。このベクタ テーブルからは、次のレベルのハンドラに渡す必要のあるコールバック値も取得されます。その後、実際の呼び出しが実行されます。
5. 割り込みコントローラを使用するシステムでは、次のレベルのハンドラは割り込みコントローラドライバにより供給されるハンドラです。このハンドラは、割り込みコントローラに対してシステムのアクティブな割り込みすべてに関するクエリを発行します。割り込みコントローラは、内部ベクタ テーブルを参照して各割り込みラインに対してユーザーが登録したハンドラを判断します。ハンドラが登録されていない場合は、デフォルトの何も実行されないハンドラが使用されます。この後、各割り込みのハンドラが割り込みの優先順に実行されます。
6. 割り込みコントローラを使用しないシステムでは、次のレベルのハンドラはアプリケーションで実行される最終割り込みハンドラです。

7. 特定の割り込みの最終割り込みハンドラは、通常割り込みペリフェラルに対してクエリを発行して割り込みの理由を判断し、そのペリフェラルと割り込みの理由に適切な処理を実行します。また、このハンドラは、割り込みペリフェラルに対して割り込みが受信されたことを通知します。この割り込みハンドラの実行が終了すると、ソフトウェア プラットフォーム レベルの割り込みハンドラまで順に戻ります。
8. プラットフォーム レベルの割り込みハンドラによりスタックに保存されているレジスタの内容が復元され、制御が割り込みが発生したプログラム カウンタ (PC) の位置に移行します。この戻り命令により、MicroBlaze プロセッサに対する割り込みがイネーブルになります。この時点で、アプリケーションの通常の実行が再開します。

割り込みハンドラは短時間にし、ほとんどの作業はアプリケーションで処理されるようにすることをお勧めします。このようにするとそのほかの割り込み (現在実行されている割り込みより優先順位が高い場合もあり) が長時間ロックアウトされるのを防ぐことができるので、良いシステム デザインと言えます。

次の図に、割り込みコントローラを使用した場合と使用しない場合の MicroBlaze の割り込みフローを示します。



X11019

図 B-3 : 割り込みコントローラを使用しない場合の MicroBlaze の割り込みフロー

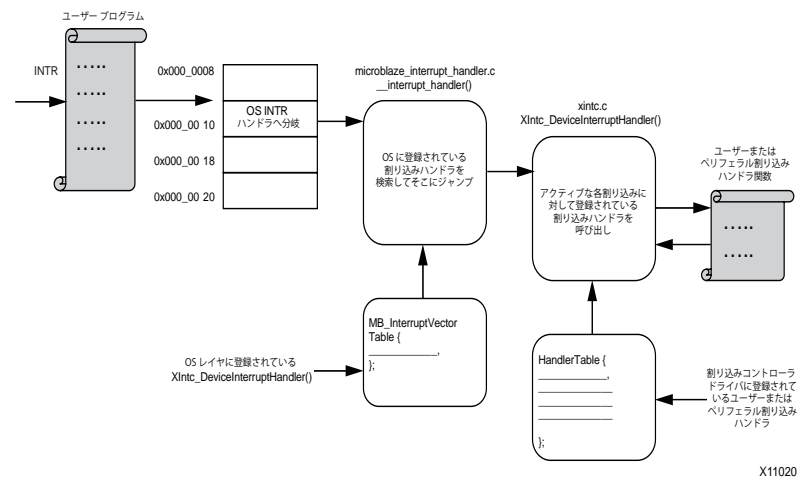


図 B-4 : 割り込みコントローラを使用する場合の MicroBlaze の割り込みフロー

## PowerPC システムの割り込みフロー

PowerPC プロセッサの割り込みフローは、次のとおりです。

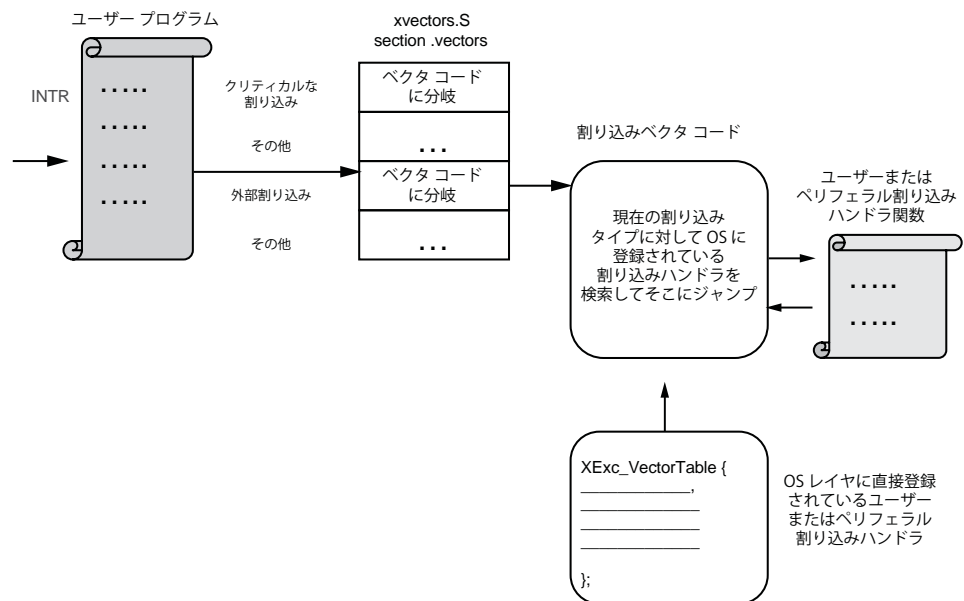
1. マシン ステータス レジスタ (MSR) で適切なビットを設定することにより、PowerPC プロセッサで割り込みをイネーブルにします。クリティカルまたは非クリティカル割り込みのどちら (あるいはその両方) が使用されるかによって、適切なビットを設定します。
2. 外部割り込み信号が受信されると、プロセッサでそれ以外の割り込みがディセーブルになり、プロセッサによりその割り込みタイプのアドレスが算出されてそのアドレスにジャンプします。この算出方法は、PowerPC405 プロセッサと PowerPC440 プロセッサで異なります。
  - ◆ PowerPC 405 プロセッサでは、例外ベクタ接頭辞レジスタ (EVPR) のソフトウェア セット値が参照され、この値 (割り込みタイプにより異なる) に一定のオフセットを加算することによりベクタ コードを配置する最終的な物理アドレスが求められます。
  - ◆ PowerPC 440 プロセッサには、各割り込みタイプに対して個別のオフセット レジスタがあります (IVOR0–IVOR15)。各オフセット レジスタには、割り込みベクタ コードの最終的な物理アドレスを求めるため、割り込みベクタ接頭辞レジスタ (IVPR) に追加される値が含まれています。
3. プロセッサが算出された割り込みベクタ コード アドレスにジャンプします。
4. 各割り込みベクタ位置には、割り込みタイプに適したプラットフォーム割り込みハンドラが含まれています。外部クリティカルおよび非クリティカル割り込みでは、このプラットフォーム割り込みハンドラにより、プロセッサ レジスタの内容 (上書きされる可能性がある) が現在のアプリケーション スタックに保存され、制御が次のレベルのハンドラに移行します。これはシステムに割り込みコントローラが存在するかどうかによって異なるので、内部割り込みベクタテーブルを参照して次のレベルのハンドラの関数アドレスが判断されます。このベクタ テーブルからは、次のレベルのハンドラに渡す必要のあるコールバック値も取得されます。その後、実際の呼び出しが実行されます。

5. 割り込みコントローラを使用するシステムでは、次のレベルのハンドラは割り込みコントローラドライバにより供給されるハンドラです。このハンドラは、割り込みコントローラに対してシステムのアクティブな割り込みすべてに関するクエリを発行します。割り込みコントローラは、内部ベクタ テーブルを参照して各割り込みラインに対してユーザーが登録したハンドラを判断します。  
ハンドラが登録されていない場合は、デフォルトの何も実行されないハンドラが使用されます。この後、各割り込みのハンドラが割り込みの優先順に実行されます。
6. 割り込みコントローラを使用しないシステムでは、次のレベルのハンドラはアプリケーションで実行される最終割り込みハンドラです。
7. 特定の割り込みの最終割り込みハンドラは、通常割り込みペリフェラルに対してクエリを発行して割り込みの理由を判断し、そのペリフェラルと割り込みの理由に適切な処理を実行します。また、このハンドラは、割り込みペリフェラルに対して割り込みが受信されたことを通知します。この割り込みハンドラの実行が終了すると、ソフトウェア プラットフォーム レベルの割り込みハンドラまで順に戻ります。

プラットフォーム レベルの割り込みハンドラによりスタックに保存されているレジスタの内容が復元され、制御が割り込みが発生したプログラム カウンタ (PC) の位置に移行します。この戻り命令により、PowerPC プロセッサに対する割り込みがイネーブルになります。この時点で、アプリケーションの通常の実行が再開します。

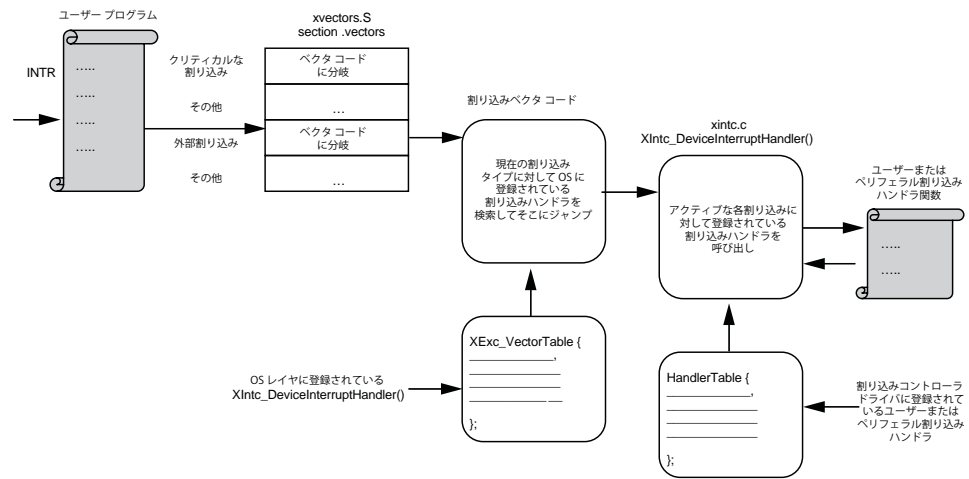
割り込みハンドラは短時間にし、ほとんどの作業はアプリケーションで処理されるようにすることをお勧めします。このようにするとそのほかの割り込み (現在実行されている割り込みより優先順位が高い場合もあり) が長時間ロックアウトされるのを防ぐことができるので、良いシステム デザインと言えます。

次の図に、割り込みコントローラを使用した場合と使用しない場合の PowerPC プロセッサの割り込みフローを示します。



X11021

図 B-5 : 割り込みコントローラを使用しない場合の PowerPC プロセッサの割り込みフロー



X11022

図 B-6: 割り込みコントローラを使用する場合の PowerPC プロセッサの割り込みフロー

## ソフトウェア API

このセクションでは、割り込みを処理および制御に関連するさまざまなソフトウェア API の概要、使用可能なソフトウェア API のリスト、および割り込み制御コードの例を示します。

**メモ:** この章では、API について詳細には説明しません。API の詳細は、割り込みコントローラ デバイスドライバの資料およびスタンドアロンプラットフォームに関する資料を参照してください。

### 割り込みコントローラ ドライバ

ザイリンクスの割り込みコントローラでは、次の機能がサポートされます。

- 特定の割り込みのイネーブルおよびディスエーブル
- 特定の割り込みの確認通知
- 割り込みソースを処理する特定のコールバック関数を添付
- マスタのイネーブルおよびディスエーブル
- 割り込みごとに 1 つのコールバックを送信するか、プロセッサの各割り込みの待機中の割り込みをすべて処理

割り込みコントロール内の割り込みの確認通知をデバイス ハンドラの前に送信するかハンドラが呼び出された後に送信するかを選択可能。割り込み信号入力、エッジまたはレベルで駆動される信号なので、これらの入力のサポートが必要となります。

- エッジで駆動される割り込み信号では、割り込みが実行される前に確認通知を送信し、近接して発生する割り込みが失われないようにする必要があります。
- レベルで駆動される割り込み入力信号では、割り込みが実行された後に確認通知を送信し、割り込みで 1 つの割り込み条件のみが生成されるようにする必要があります。

## API の説明

```
int XIntc_Initialize (XIntc * InstancePtr, ul6 DeviceId)
```

**説明** 特定の割り込みコントローラ インスタンスまたはドライバを初期化します。XIntc 構造のすべてのフィールドおよび内部ベクタ テーブルが初期化されます。すべての割り込みソースはディスエーブルになります。

**パラメータ** *InstancePtr*: XIntc インスタンスへのポインタです。  
*DeviceId*: XIntc インスタンス (xparameters.h から取得) で制御されるデバイス固有の ID です。*DeviceId* が渡されると、呼び出し元またはアプリケーション開発者の選択したとおりに、ジェネリック XIntc インスタンスが特定のデバイスに関連付けられます。

```
int XIntc_Connect (XIntc * InstancePtr, u8 Id,  
XInterruptHandler Handler, void * CallBackRef)
```

**説明** 割り込みソースの *Id* と、その割り込みが発生したときに実行されるハンドラを接続します。*CallBackRef* で供給される引数は、ハンドラが呼び出されたときに引数として使用されます。

**パラメータ** *InstancePtr*: XIntc インスタンスへのポインタです。  
*Id*: 割り込みソースの ID です。可能な値の範囲は 0 ~ (XPAR\_INTC\_MAX\_NUM\_INTR\_INPUTS - 1) で、0 の割り込みが最も優先されます。  
*Handler*: 割り込みのハンドラです。  
*CallBackRef*: コールバック参照です。通常は接続ドライバのインスタンス ポインタです。

**注意** 引数としてハンドラを指定すると、それ以前に接続されているハンドラは無効になります。

```
void XIntc_Disconnect (XIntc* InstancePtr, u8 Id)
```

**説明** XIntc インスタンスの接続を解除します。

**パラメータ** *InstancePtr*: XIntc インスタンスへのポインタです。  
*Id*: 割り込みソースの ID です。可能な値の範囲は 0 ~ (XPAR\_INTC\_MAX\_NUM\_INTR\_INPUTS - 1) で、0 の割り込みが最も優先されます。



**Void XIntc\_Enable** (XIntc \* InstancePtr, u8 Id)

説明 引数 *Id* で指定した割り込みソースをイネーブルにします。指定された *Id* の待機中の割り込み条件は、この関数が呼び出された後に発生します。

パラメータ *InstancePtr*: XIntc インスタンスへのポインタです。  
*Id*: 割り込みソースの ID です。可能な値の範囲は 0 ~ (XPAR\_INTC\_MAX\_NUM\_INTR\_INPUTS - 1) で、0 の割り込みが最も優先されます。

---

**void XIntc\_Disable** (XIntc \* InstancePtr, u8 Id)

説明 割り込みコントローラで引数 *Id* で指定した割り込みが発生しないように、その割り込みソースをディスエーブルにします。割り込みコントローラはその *Id* の割り込み条件を保持しますが、割り込みは発生しません。

パラメータ *InstancePtr*: XIntc インスタンスへのポインタです。  
*Id*: 割り込みソースの ID です。可能な値の範囲は 0 ~ (XPAR\_INTC\_MAX\_NUM\_INTR\_INPUTS - 1) で、0 の割り込みが最も優先されます。

---

**int XIntc\_Start** (XIntc \* InstancePtr, u8 Mode)

説明 割り込みコントローラからプロセッサへの出力をイネーブルにすることにより、割り込みコントローラを開始します。この関数が呼び出されると、割り込みコントローラで割り込みを生成できるようになります。

パラメータ *InstancePtr*: XIntc インスタンスへのポインタです。  
*Mode*: ソフトウェアで割り込みをシミュレーションするか、実際の割り込みを発生させるかを指定します。これらのモードを同時に設定することはできません。割り込みコントローラ ハードウェアは、ソフトウェアで割り込みをシミュレーションするモードが解除されるまで、このモードにリセットされます。このモードを一度解除すると、再び有効にすることはできません。指定可能な値は、次のとおりです。

XIN\_SIMULATION\_MODE: 割り込みのシミュレーションのみをイネーブルにします。

XIN\_REAL\_MODE: ハードウェア割り込みのみをイネーブルにします。

この関数は、XIntc の初期化が完了した後に呼び出す必要があります。

```
void XIntc_Stop (XIntc * InstancePtr)
```

説明                      割り込みコントローラからの出力をディスエーブルにして割り込みコントローラを停止して、割り込みコントローラにより割り込みが発生しないようにします。

パラメータ              *InstancePtr* : XIntc インスタンスへのポインタです。

## MicroBlaze 用のスタンドアロン ソフトウェア プラットフォーム API

次に、スタンドアロン ソフトウェア プラットフォームを使用して MicroBlaze プロセッサの割り込みを処理するために使用する関数を示します。

---

```
void microblaze_register_handler XInterruptHandler Handler,  
void* DataPtr)
```

説明                      システムで割り込みが発生した際に、スタンドアロン ソフトウェア プラットフォームの割り込みハンドラから起動するハンドラを登録します。割り込みコントローラがない場合は、アプリケーションの最終割り込みハンドラを登録します。割り込みコントローラがある場合は、割り込みコントローラドライバのハンドラ `XIntc_DeviceInterruptHandler` を登録し、割り込みコントローラドライバで割り込みを制御できるようにします。

パラメータ              *Handler* : 起動されるハンドラ関数です。

*DataPtr* : 関数が起動されたときに渡されるコールバック値です。

---

```
void microblaze_enable_interrupts(void)
```

説明                      MicroBlaze プロセッサ上の外部割り込みをイネーブルにします。

---

```
void microblaze_disable_interrupts(void)
```

説明                      MicroBlaze プロセッサ上の外部割り込みをディスエーブルにします。

## MicroBlaze の割り込み設定例

次に、割り込みコントローラが存在する場合に **MicroBlaze** で割り込みを初期化するプログラム例を示します。この例は、次のようなプログラムです。

- システムには `xps_timer_0` という名前の `xps_timer` ペリフェラルがあります。
- タイマからの割り込み信号は、`xps_intc_0` という名前の `xps_intc` 割り込みコントローラに接続されています。
- コントローラからの割り込み信号は、**MicroBlaze** の割り込み入力に接続されています。

```
#include "mb_interface.h"
#include "xparameters.h"
#include "xtmrctr.h"
#include "xintc.h"

XIntc sys_intc;
XTmrCtr sys_tmrctr;

void my_timer_handler()
{
    XUint32 ControlStatusReg;

    /*
     * Read the new Control/Status Register content.
     */
    ControlStatusReg = XTimerCtr_mReadReg(sys_tmrctr.BaseAddress,
                                          0,
                                          XTC_TCSR_OFFSET);

    /*
     * Acknowledge the interrupt by clearing the interrupt
     * bit in the timer control status register
     */
    XTmrCtr_mWriteReg(sys_tmrctr.BaseAddress, 0,
                      XTC_TCSR_OFFSET,
                      ControlStatusReg |
                      XTC_CSR_INT_OCCURED_MASK);

    print("Timer interrupt occurred.\r\n");
}

int main()
{
    XStatus Status;

    /*
     * Initialize the interrupt controller driver so that
     * it is ready to use, specify the device ID that is generated in
     * xparameters.h
     */
    Status = XIntc_Initialize(&sys_intc, XPAR_XPS_INTC_0_DEVICE_ID);
    if (Status != XST_SUCCESS)
    {
        return XST_FAILURE;
    }
}
```

```

/*
 * Connect the application handler that will be called when an
interrupt
 * for the timer occurs
 */

Status = XIntc_Connect(&sys_intc,
XPAR_XPS_INTC_0_XPS_TIMER_0_INTERRUPT_INTR,
(XInterruptHandler)my_timer_handler,
(void *)0);
if (Status != XST_SUCCESS)
{
    return XST_FAILURE;
}

/*
 * Start the interrupt controller such that interrupts are enabled
for
 * all devices that cause interrupts.
 */
Status = XIntc_Start(&sys_intc, XIN_REAL_MODE);
if (Status != XST_SUCCESS)
{
    return XST_FAILURE;
}

/*
 * Enable the interrupt for the timer counter
 */
XIntc_Enable(&sys_intc,
XPAR_XPS_INTC_0_XPS_TIMER_0_INTERRUPT_INTR);

/*
 * Initialize the timer counter so that it's ready to use,
 * specify the device ID that is generated in xparameters.h
 */
Status = XTmrCtr_Initialize(&sys_tmrctr,
XPAR_XPS_TIMER_0_DEVICE_ID);
if (Status != XST_SUCCESS)
{
    return XST_FAILURE;
}

/*
 * Enable the interrupt of the timer counter so interrupts will occur
 * and use auto reload mode such that the timer counter will reload
 * itself automatically and continue repeatedly, without this option
 * it would expire once only
 */
XTmrCtr_SetOptions(&sys_tmrctr, 0,
XTC_INT_MODE_OPTION | XTC_AUTO_RELOAD_OPTION);

/*
 * Set a reset value for the timer counter such that it will expire
 * eariler than letting it roll over from 0, the reset value is
loaded
 * into the timer counter when it is started

```

```

    */
    XTmrCtr_SetResetValue(&sys_tmrctr, 0, 0xDEADBEEF);

    /*
    * Start the timer counter such that it's incrementing by default,
    * then wait for it to timeout a number of times
    */
    XTmrCtr_Start(&sys_tmrctr, 0);

    /*
    * Register the intc device driver's handler with the Standalone
    * software platform's interrupt table
    */

    microblaze_register_handler((XInterruptHandler)XIntc_DeviceInterruptHa
    ndler,

                                (void*)XPAR_XPS_INTC_0_DEVICE_ID);

    /*
    * Enable interrupts on MicroBlaze
    */
    microblaze_enable_interrupts();

    /*
    * At this point, the system is ready to respond to interrupts from
the
    * timer
    */
    while(1);
}

```

## PowerPC 405 および 440 プロセッサ用のスタンドアロン ソフトウェア API

次に、スタンドアロン ソフトウェア プラットフォームを使用して PowerPC プロセッサの割り込みを処理するために使用する関数を示します。

---

```
void XExc_Init (void)
```

説明	PowerPC プロセッサ システムの例外 (割り込み) 処理を初期化します。割り込みおよび例外ベクタ テーブルには、すべての例外に対してスタブ ハンドラが設定されています。
----	---

```
void XExc_RegisterHandler(Xuint8 ExceptionId,
    XExceptionHandler Handler, void *DataPtr)
```

説明	システムで例外が発生した際に、スタンドアロン ソフトウェア プラットフォームの例外ハンドラから起動するハンドラを登録します。割り込みはプロセッサで処理可能な例外の 1 つなので、この関数を割り込みハンドラの登録に使用できます。割り込みコントローラがない場合は、アプリケーションの最終割り込みハンドラを登録します。割り込みコントローラがある場合は、割り込みコントローラ ドライバのハンドラ <code>XIntc_DeviceInterruptHandler</code> を登録し、割り込みコントローラ ドライバで割り込みを制御できるようにします。
パラメータ	<p><i>ExceptionId</i>: ハンドラを登録する例外の識別子です。PowerPC プロセッサのクリティカル外部入力割り込みの識別子は <code>XEXC_ID_CRITICAL_INT</code>、非クリティカル外部入力割り込みの識別子は <code>XEXC_ID_NON_CRITICAL_INT</code> です。</p> <p><i>Handler</i>: 起動されるハンドラ関数です。</p> <p><i>DataPtr</i>: 関数が起動されたときに渡されるコールバック値です。</p>

---

```
void XExc_mEnableExceptions(EnableMask)
```

説明	PowerPC プロセッサで指定の例外カテゴリをイネーブルにします。
パラメータ	<p><i>EnableMask</i>: イネーブルにする例外カテゴリです。</p> <p><code>XEXC_CRITICAL</code> は外部クリティカル入力割り込みを指定します。</p> <p><code>XEXC_NON_CRITICAL</code> は外部非クリティカル入力割り込みを指定します。</p>

---

```
void XExc_mDisableExceptions(DisableMask)
```

説明	PowerPC プロセッサで指定の例外 (またはカテゴリ) をディスエーブルにします。
パラメータ	<p><i>DisableMask</i>: ディスエーブルにする例外カテゴリです。</p> <p><code>XEXC_CRITICAL</code> は外部クリティカル入力割り込みを指定します。</p> <p><code>XEXC_NON_CRITICAL</code> は外部非クリティカル入力割り込みを指定します。</p>

## PowerPC プロセッサの割り込み設定例

次に、割り込みコントローラが存在する場合に PowerPC 405 または 440 プロセッサで割り込みを初期化するプログラム例を示します。この例は、次のようなプログラムです。

- システムには `xps_timer_0` という名前の `xps_timer` ペリフェラルがあります。
- タイマからの割り込み信号は、`xps_intc_0` という名前の `xps_intc` 割り込みコントローラに接続されています。
- コントローラからの割り込み信号は、PowerPC プロセッサの外部非クリティカル割り込み入力に接続されています。

```
#include "xexception_l.h"
#include "xparameters.h"
#include "xtmrctr.h"
#include "xintc.h"

XIntc sys_intc;
XTmrCtr sys_tmrctr;

void my_timer_handler()
{
    XUint32 ControlStatusReg;

    /*
     * Read the new Control/Status Register content.
     */
    ControlStatusReg = XTimerCtr_mReadReg(sys_tmrctr.BaseAddress,
                                           0,
                                           XTC_TCSR_OFFSET);

    /*
     * Acknowledge the interrupt by clearing the interrupt
     * bit in the timer control status register
     */
    XTmrCtr_mWriteReg(sys_tmrctr.BaseAddress, 0,
                      XTC_TCSR_OFFSET,
                      ControlStatusReg |
                      XTC_CSR_INT_OCCURED_MASK);

    print("Timer interrupt occurred.\r\n");
}

int main()
{
    XStatus Status;

    /*
     * Initialize the interrupt controller driver so that
     * it is ready to use, specify the device ID that is generated in
     * xparameters.h
     */
    Status = XIntc_Initialize(&sys_intc, XPAR_XPS_INTC_0_DEVICE_ID);
    if (Status != XST_SUCCESS)
    {
        return XST_FAILURE;
    }
}
```

```

/*
 * Connect the application handler that will be called when an
interrupt
 * for the timer occurs
 */

Status = XIntc_Connect(&sys_intc,
XPAR_XPS_INTC_0_XPS_TIMER_0_INTERRUPT_INTR,
(XInterruptHandler)my_timer_handler,
(void *)0);
if (Status != XST_SUCCESS)
{
    return XST_FAILURE;
}

/*
 * Start the interrupt controller such that interrupts are enabled
for
 * all devices that cause interrupts.
 */
Status = XIntc_Start(&sys_intc, XIN_REAL_MODE);
if (Status != XST_SUCCESS)
{
    return XST_FAILURE;
}

/*
 * Enable the interrupt for the timer counter
 */
XIntc_Enable(&sys_intc,
XPAR_XPS_INTC_0_XPS_TIMER_0_INTERRUPT_INTR);

/*
 * Initialize the timer counter so that it's ready to use,
 * specify the device ID that is generated in xparameters.h
 */
Status = XTmrCtr_Initialize(&sys_tmrctr,
XPAR_XPS_TIMER_0_DEVICE_ID);
if (Status != XST_SUCCESS)
{
    return XST_FAILURE;
}

/*
 * Enable the interrupt of the timer counter so interrupts will occur
 * and use auto reload mode such that the timer counter will reload
 * itself automatically and continue repeatedly, without this option
 * it would expire once only
 */
XTmrCtr_SetOptions(&sys_tmrctr, 0,
XTC_INT_MODE_OPTION | XTC_AUTO_RELOAD_OPTION);

/*

```



```
    * Set a reset value for the timer counter such that it will expire
    * earlier than letting it roll over from 0;
    * the reset value is loaded
    * into the timer counter when it is started
    */
    XTmrCtr_SetResetValue(&sys_tmrctr, 0, 0xDEADBEEF);

    /*
    * Start the timer counter such that it's incrementing by default,
    * then wait for it to timeout a number of times
    */
    XTmrCtr_Start(&sys_tmrctr, 0);

    /*
    * Initialize the Standalone software platform exception system.
    */
    XExc_Init();

    /*
    * Register the interrupt controller handler with the Standalone
    * software platform's exception table.
    */
    XExc_RegisterHandler(XEXC_ID_NON_CRITICAL_INT,
                        (XExceptionHandler)XIntc_DeviceInterruptHandler,
                        (void*)XPAR_XPS_INTC_0_DEVICE_ID);

    /*
    * Enable non-critical exceptions on PowerPC.
    */
    XExc_mEnableExceptions(XEXC_NON_CRITICAL);

    /*
    * At this point, the system is ready to respond to interrupts
    * from the timer
    */
    while(1);
}
}
```



# EDK Tcl インターフェイス

---

この付録では、EDK ツールで使用可能なツール コマンド 言語 (Tcl) アプリケーション プログラム インターフェイス (API) について、および EDK ツールから Tcl API を使用して情報にアクセスする方法を説明します。

この付録は、次のセクションから構成されています。

- はじめに
- その他のリソース
- ハンドル
- データ構造の作成
- Tcl コマンドの使用法
- EDK ハードウェア Tcl コマンド
- ハードウェア API を使用する Tcl の例
- ソフトウェア Tcl コマンド
- ハードウェア プラットフォーム生成の Tcl フロー
- 統合ハードウェア データ構造の追加のキーワード
- ソフトウェア プラットフォーム生成の Tcl フロー

## はじめに

EDK ツールを実行すると、デザインのランタイム データ構造が毎回構築されます。このデータ構造には、MHS (Microprocessor Hardware Specification) や MSS (Microprocessor Software Specification) などのユーザー デザイン ファイル、MPD (Microprocessor Peripheral Definition)、MDD (Microprocessor Driver Definition)、MLD (Microprocessor Library Definition) などのライブラリ データ ファイルに関する情報が含まれます。データ構造にアクセスするには、Tcl API を使用します。IP、ドライバ、ライブラリ、OS ライタは、デザイン要件に基づき、データ構造情報にアクセスしてツールの処理手順を追加します。EDK ツールでも、Tcl を使用してさまざまな DRC (デザイン ルール チェック) を実行し、デザインのデータ構造を限られた範囲内でアップデートします。

## その他のリソース

- 『Platform Specification Format Reference Manual』  
[http://japan.xilinx.com/ise/embedded/edk\\_docs.htm](http://japan.xilinx.com/ise/embedded/edk_docs.htm)

## ハンドル

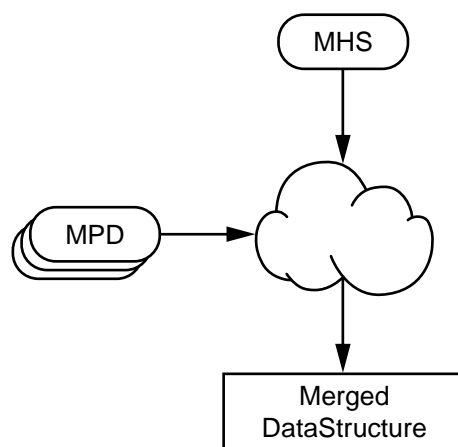
ツールは、API 関数を使用してデータ構造にアクセスします。各 API 関数には、システム情報の形の引数が必要です。これらの引数は、「ハンドル」と呼ばれます。

たとえば、MHS ファイルで定義される IP や、MSS ファイルで定義されるドライバは、ハンドラとなり得ます。ハンドラには、アクセスするデータのタイプに応じてさまざまなタイプがあります。データ タイプには、インスタンス名、ドライバ名、ハードウェア パラメータ、ハードウェア ポートなどがあります。特定のハンドルから、そのハンドルに関する情報を取得したり、関連するほかのハンドルを取得したりできます。

## データ構造の作成

EDK ツールでは、2 つの基本的なランタイム情報にアクセスできます。

- オリジナル デザインとライブラリ データファイルのデータ構造
- オリジナル データ構造では、さまざまなデータ ファイルに存在する情報のみにアクセス可能です。MHS、MSS、MPD、MDD、MLD などのファイルへのハンドルを取得できます。これらのハンドルを使用すると、対応するファイルの内容を参照できます。
- 統合データ構造
- EDK ツールを実行すると、デザイン ファイル (MHS または MSS) の情報がライブラリ ファイル (MPD、MDD、MLD) からの対応する情報と統合され、ハードウェア統合データ構造 (ハードウェア統合オブジェクト) およびソフトウェア統合データ構造 (ソフトウェア統合オブジェクト) の統合データ構造が作成されます。統合データ構造の作成プロセスで、接続やアドレスマップなどのさまざまなデザイン特性も解析され、その情報も統合データ構造に保存されます。統合データ構造では、この解析情報に簡単にアクセスできます。たとえば、MHS ファイル内の IP のインスタンスは対応する MPD と統合されるので、統合インスタンスを使用すると、1 つのハンドルで完全な情報を取得でき、IP インスタンスのハンドルと MPD のハンドルに別々にアクセスする必要はありません。



X10582

図 C-1 : 統合ハードウェア データ構造の作成

## Tcl コマンドの使用法

### 一般的な規則

Tcl API には、返すデータのタイプによって 2 種類あります。Tcl API は、次のものを返します。

- オブジェクトのハンドルまたはハンドルのリスト
- 値または値のリスト

すべての Tcl API に共通の規則があります。

- 別のオブジェクトへの要求されたハンドルが見つからなかった場合、NULL ハンドルが返されます。
- 値が空であるか特定できない場合、空の文字列が返されます。

### Tcl コマンド使用前の確認事項

XPS をコマンド ラインモード (xps -nw) で使用する場合は、xload コマンドを使用してプロジェクトを読み込み、内部ツール データベース (ランタイム データ構造) を初期化する必要があります。

```
xload <filetype> <filename>.{MHS/MSS/XMP}
```

xload コマンドの詳細は、[第 13 章「コマンド ライン モード」](#) を参照してください。

MHS ハンドルまたは統合 MHS ハンドルにアクセスするには、プロジェクトを読み込んだ後に次のいずれかのコマンドを使用します。

```
XPS% set original_mhs_handle [xget_handle mhs]
```

または

```
XPS% set merged_mhs_handle [xget_handle merged_mhs]
```

次のセクションで、EDK ハードウェア Tcl コマンドについて詳細に説明します。

## EDK ハードウェア Tcl コマンド

### 概要

このセクションでは、EDK のハードウェア データ構造で使用可能な Tcl API を示します。これらのコマンドの説明で使用する用語を次に定義します。

#### オリジナル MHS ハンドル (original\_mhs\_handle)

MHS 情報のみを含むハンドルで、MPD 情報は含まれません。MHS で IP パラメータを指定していない場合、そのパラメータは含まれません。

#### 統合 MHS ハンドル (merged\_mhs\_handle)

MHS と MPD 両方の情報を含むハンドルです。ハードウェア データ構造/統合オブジェクトは、MHS と MPD 情報が統合される際に作成されます。

**メモ :** Tcl プロシージャは、Platgen、Libgen、Simgen などのバッチ ツールからも呼び出されます。バッチ ツールで提供されるハンドルでは、常に統合 MHS ハンドルが参照されます。バッチ ツールからオリジナル MHS ハンドルにアクセスすることはできません。オリジナル MHS ハンドルは、MHS デザイン ファイルをアップデートするために API を使用してデザインを変更する場合にのみ必要です。

#### オリジナル IP インスタンス ハンドル (original\_IP\_handle)

MHS ファイルに示される情報のみを含むオリジナル MHS ハンドルから取得した IP インスタンスへのハンドルです。

#### 統合 IP インスタンス ハンドル (merged\_IP\_handle)

統合 MHS ハンドルから取得した IP ハンドルです。統合 IP インスタンス ハンドルには、MHS と MPD 両方の情報が含まれます。

**メモ :** Platgen などのバッチ ツールは、統合 IP インスタンス ハンドルにのみアクセスし、オリジナル IP インスタンス ハンドルにはアクセスしません。そのため、パラメータ ハンドルやポート ハンドルなどのプロパティ ハンドルも、オリジナル ハンドルではなく統合ハンドルです。

## ハードウェア読み出しアクセス API

次の表に、定義済みのハードウェア読み出しアクセス API の一覧を示します。この表で API のリンクをクリックすると、その説明にジャンプします。

### ハードウェア読み出しアクセス API の一覧

表 C-1 : ハードウェア読み出しアクセス API

[xget\\_hw\\_busif\\_value <handle> <busif\\_name>](#)  
[xget\\_hw\\_bus\\_slave\\_addrpairs <merged\\_bus\\_handle>](#)  
[xget\\_hw\\_busif\\_handle <handle> <busif\\_name>](#)  
[xget\\_hw\\_connected\\_busifs\\_handle <merged\\_mhs\\_handle> <businst\\_name> <busif\\_type>](#)  
[xget\\_hw\\_connected\\_ports\\_handle <merged\\_mhs\\_handle> <connector\\_name> <port\\_type>](#)  
[xget\\_hw\\_ioif\\_handle <handle> <ioif\\_name>](#)  
[xget\\_hw\\_ioif\\_value <handle> <ioif\\_name>](#)  
[xget\\_hw\\_ipinst\\_handle <mhs\\_handle> <ipinst\\_name>](#)  
[xget\\_hw\\_mpd\\_handle <ipinst\\_handle>](#)  
[xget\\_hw\\_name <handle>](#)  
[xget\\_hw\\_option\\_handle <handle> <option\\_name>](#)  
[xget\\_hw\\_option\\_value <handle> <option\\_name>](#)  
[xget\\_hw\\_parameter\\_handle <handle> <parameter\\_name>](#)  
[xget\\_hw\\_parameter\\_value <handle> <parameter\\_name>](#)  
[xget\\_hw\\_pcore\\_dir\\_from\\_mpd <mpd\\_handle>](#)  
[xget\\_hw\\_pcore\\_dir <ipinst\\_handle>](#)  
[xget\\_hw\\_port\\_connectors\\_list <ipinst\\_handle> <portName>](#)  
[xget\\_hw\\_parent\\_handle <handle>](#)  
[xget\\_hw\\_port\\_connectors\\_list <ipinst\\_handle> <portName>](#)  
[xget\\_hw\\_port\\_handle <handle> <port\\_name>](#)  
[xget\\_hw\\_port\\_value <handle> <port\\_name>](#)  
[xget\\_hw\\_proj\\_setting <prop\\_name>](#)  
[xget\\_hw\\_proc\\_slave\\_periphs <merged\\_proc\\_handle>](#)  
[xget\\_hw\\_subproperty\\_handle <property\\_handle> <subprop\\_name>](#)  
[xget\\_hw\\_subproperty\\_value <property\\_handle> <subprop\\_name>](#)  
[xget\\_hw\\_value <handle>](#)

### ハードウェア読み出しアクセス API の説明

---

**xget\_hw\_busif\_handle** <handle> <busif\_name>

**説明**                      ハンドルに関連するバス インターフェイスへのハンドルを返します。

**引数**                      <handle> : MPD、オリジナル IP インスタンス、または統合 IP インスタンスへのハンドルです。

<busif\_name> : ハンドルを取得するバス インターフェイスの名前です。アスタリスク (\*) を指定すると、バス インターフェイス ハンドルのリストが返されます。個々のバス インターフェイス ハンドルにアクセスするには、Tcl のリストに対して繰り返し処理を実行します。

**xget\_hw\_busif\_value** <handle> <busif\_name>

- 説明** 指定したバス インターフェイスの値を返します。値は通常、指定したバス インターフェイスに接続されているバスのインスタンス名です。透過バス インターフェイスの場合は、値はバス インスタンス名ではなくコネクタです。
- 引数** <handle> : MPD、オリジナル IP インスタンス、または統合 IP インスタンスへのハンドルです。
- <busif\_name> : 値を取得するバス インターフェイスの名前です。

**xget\_hw\_bus\_slave\_addrpairs** <merged\_bus\_handle>

- 説明** 指定したバス ハンドルに関連するスレーブ アドレスのリストを返します。戻り値は、次のような整数値のリストです。
- 最初の値は、接続されたペリフェラルのベース アドレスです。
  - 2 番目の値は、そのハイ アドレスです。
  - その後の値は、その他のペリフェラルのベース アドレスとハイ アドレスがペアでリストされます。
- 引数** <merged\_bus\_handle> : バス インスタンスを示す統合 IP インスタンスへのハンドルです。

**xget\_hw\_connected\_busifs\_handle** <merged\_mhs\_handle>  
<businst\_name> <busif\_type>

- 説明** 指定したバスに接続されているバス インターフェイスへのハンドルのリストを返します。
- 引数** <merged\_mhs\_handle> : 統合 MHS へのハンドルです。
- <businst\_name> : 接続されたバス インスタンスの名前です。
- <busif\_type> : MASTER、SLAVE、TARGET、INITIATOR、または ALL のいずれかです。



```
xget_hw_connected_ports_handle <merged_mhs_handle>  
    <connector_name> <port_type>
```

**説明** 指定したコネクタに関連しているポートへのハンドルのリストを返します。  
有効なハンドル タイプは統合 MHS です。

**引数** <merged\_mhs\_handle> : 統合 MHS へのハンドルです。  
<connector\_name> : コネクタの名前です。  
<port\_type> : source、sink、または all のいずれかです。  
この API は、<port\_type> によって、次のようなポートのハンドルのリストを返します。

- source : 指定の信号を駆動するポートのリスト
- sink : 指定の信号で駆動されるポートのリスト
- all : 指定の信号に接続されるすべてのポートのリスト

---

```
xget_hw_ioif_handle <handle> <ioif_name>
```

**説明** ハンドルに関連する I/O インターフェイスへのハンドルを返します。

**引数** <handle> : MPD または統合 IP インスタンスへのハンドルです。  
**メモ** : オリジナル IP インスタンスへのハンドルを指定した場合、NULL が返されます。  
<ioif\_name> : ハンドルを取得する I/O インターフェイスの名前です。アスタリスク (\*) を指定すると、I/O インターフェイス ハンドルのリストが返されます。個々の I/O インターフェイス ハンドルにアクセスするには、Tcl のリストに対して繰り返し処理を実行します。

---

```
xget_hw_ioif_value <handle> <ioif_name>
```

**説明** 指定した I/O インターフェイスの値を返します。値は MPD ファイルで指定されており、MHS で上書きすることはできません。

**引数** <handle> : MPD または統合 IP インスタンスへのハンドルです。  
<ioif\_name> : 値を取得する I/O インターフェイスの名前です。

---

```
xget_hw_ipinst_handle <mhs_handle> <ipinst_name>
```

**説明** 指定した IP インスタンスのハンドルを返します。

**引数** <mhs\_handle> : オリジナル MHS または統合 MHS へのハンドルです。

<ipinst\_name> : ハンドルを取得する IP インスタンスの名前です。アスタリスク (\*) を指定すると、IP インスタンス ハンドルのリストが返されます。個々の IP インスタンス ハンドルにアクセスするには、Tcl のリストに対して繰り返し処理を実行します。

---

```
xget_hw_mpd_handle <ipinst_handle>
```

**説明** 指定した IP インスタンスに関連する MPD オブジェクトへのハンドルを返します。

**引数** <ipinst\_handle> : 統合 IP インスタンスへのハンドルです。

---

```
xget_hw_name <handle>
```

**説明** 指定したハンドルの名前を返します。

**引数** <handle> : ハンドルのタイプです。

<handle> が IP インスタンスの場合、返される名前はその IP インスタンスの名前です。たとえば、ハンドルが MHS ファイル内の mymb という MicroBlaze インスタンスを参照する場合、API は mymb を返します。パラメータ ハンドルからパラメータの名前を取得する場合にも、このコマンドを使用します。

---

```
xget_hw_option_handle <handle> <option_name>
```

**説明** オプションに関連するハンドルを返します。

**引数** <handle> : 関連するオプションです。

<option\_name> : ハンドルを取得するオプションの名前です。

アスタリスク (\*) を指定すると、オプション ハンドルのリストが返されます。個々のオプション ハンドルにアクセスするには、Tcl のリストに対して繰り返し処理を実行します。

**xget\_hw\_option\_value** <handle> <option\_name>

**説明**                    オプションの値を返します。値は MPD ファイルで指定されており、MHS で上書きすることはできません。

**引数**                    <handle> : MPD または統合 IP インスタンスへのハンドルです。  
                         <option\_name> : ハンドルを取得するオプションの名前です。

---

**xget\_hw\_parameter\_handle** <handle> <parameter\_name>

**説明**                    関連するパラメータへのハンドルを返します。

**引数**                    <handle> : MPD、オリジナル IP インスタンス、または統合 IP インスタンスへのハンドルです。  
  
                         <parameter\_name> : ハンドルを取得する関連するパラメータの名前です。アスタリスク (\*) を指定すると、パラメータハンドルのリストが返されます。個々のパラメータハンドルにアクセスするには、Tcl のリストに対して繰り返し処理を実行します。

---

**xget\_hw\_parameter\_value** <handle> <parameter\_name>

**説明**                    指定したパラメータの値を返します。

**引数**                    <handle> : MPD、オリジナル IP インスタンス、または統合 IP インスタンスへのハンドルです。  
  
                         <parameter\_name> : 値を取得する関連するパラメータの名前です。

---

**xget\_hw\_parent\_handle** <handle>

**説明** 指定したハンドルの親へのハンドルを返します。親ハンドルのタイプは、指定したハンドルのタイプによって決まります。指定したハンドルが統合ハンドルの場合は、この API で取得される親も統合ハンドルです。

**引数** <handle> は、次のいずれかです。

- **PARAMETER** : 親は MPD、IP インスタンス、または統合 IP インスタンスオブジェクトです。
- **PORT** : 親は MPD、IP インスタンス、統合 IP インスタンス、または MHS オブジェクトです。
- **BUS\_INTERFACE** : 親は MPD、IP インスタンス、または統合 IP インスタンスオブジェクトです。
- **IO\_INTERFACE** : 親は MPD または統合 IP インスタンスオブジェクトです。
- **OPTION** : 親は MPD または統合 IP インスタンスオブジェクトです。
- **IPINST** : 親は MHS または統合 MHS オブジェクトです。
- **MHS または MPD** : 親は NULL ハンドルです。

---

**xget\_hw\_pcore\_dir\_from\_mpd** <mpd\_handle>

**説明** MPD の pcore ディレクトリ パスを返します。

**引数** <mpd\_handle> : MPD へのハンドルです。

---

**xget\_hw\_pcore\_dir** <ipinst\_handle>

**説明** 指定の IP インスタンスの pcore ディレクトリ パスを返します。

**引数** <ipinst\_handle> : IP インスタンスへのハンドルです。

---

**xget\_hw\_port\_connectors\_list** <ipinst\_handle> <portName>

**説明** ポートの値 (コネクタ) が & で区切られたリストの場合、そのリストを分割し、文字列 (コネクタ名) のリストを返します。

**引数** <ipinst\_handle> : IP インスタンス (統合またはオリジナル) へのハンドルです。

<option\_name> : コネクタを取得するポートの名前です。

**xget\_hw\_port\_handle** <handle> <port\_name>

- 説明**                    ハンドルに関連するポートへのハンドルを返します。ハンドルのタイプが **MHS** である場合、返されるハンドルは指定した名前のグローバル ポートを示します。
- 引数**                    <handle> : MPD、オリジナル IP インスタンス、統合 IP インスタンス、オリジナル MHS、または統合 MHS へのハンドルです。
- <port\_name> : ハンドルを取得するポートの名前です。
- アスタリスク (\*) を指定すると、ポート ハンドルのリストが返されます。個々のポート ハンドルにアクセスするには、Tcl のリストに対して繰り返し処理を実行します。
- ハンドルのタイプが **MHS** (オリジナルまたは統合) である場合、返されるハンドルは指定した名前のグローバル ポートを示します。

---

**xget\_hw\_port\_value** <handle> <port\_name>

- 説明**                    指定したポートの値を返します。ポートの値は、ポートに接続されている信号の名前です。
- 引数**                    <handle> : MPD、オリジナル IP インスタンス、統合 IP インスタンス、オリジナル MHS、または統合 MHS へのハンドルです。
- <port\_name> : 値を取得するポートの名前です。

---

**xget\_hw\_proj\_setting** <prop\_name>

- 説明**                    <prop\_name> で指定したプロパティの値を返します。
- 引数**                    <prop\_name> : 値を取得するプロパティの名前です。fpga\_family、fpga\_subfamily、fpga\_partname、fpga\_device、fpga\_package、fpga\_speedgrade のいずれかです。

---

**xget\_hw\_proc\_slave\_periphs** <merged\_proc\_handle>

- 説明**                    指定したプロセッサでアドレス指定可能なスレーブへのハンドルのリストを返します。
- 引数**                    <merged\_proc\_handle> : プロセッサ インスタンスを示す統合 IP インスタンスへのハンドルです。返されるリストには、プロセッサに直接接続されておらず、バス間ブリッジ (opb2plb\_bridge など) を介してアクセス可能なスレーブが含まれます。
- 入力ハンドルは、プロセッサ インスタンスへの IP インスタンス ハンドルである必要があります、オリジナル MHS からではなく 統合 MHS からのみ取得可能です。

**xget\_hw\_subproperty\_handle** <property\_handle> <subprop\_name>

**説明** <property\_handle> で指定したプロパティ ハンドルに関連するサブプロパティへのハンドルを返します。

**引数** <property\_handle> : PARAMETER、PORT、BUS\_INTERFACE、IO\_INTERFACE、または OPTION のいずれかへのハンドルです。  
 <subprop\_name> : ハンドルを取得するサブプロパティの名前です。サブプロパティのリストは、『Platform Specification Format Reference Manual』の「Microprocessor Peripheral Definition (MPD)」および [270 ページの「統合ハードウェア データ構造の追加のキーワード」](#) を参照してください。

**xget\_hw\_subproperty\_value** <property\_handle> <subprop\_name>

**説明** 指定したサブプロパティの値を返します。

**引数** <property\_handle> : PARAMETER、PORT、BUS\_INTERFACE、IO\_INTERFACE、または OPTION のいずれかです。  
 <subprop\_name> : 値を取得するサブプロパティの名前です。サブプロパティのリストは、『Platform Specification Format Reference Manual』の「Microprocessor Peripheral Definition (MPD)」および [270 ページの「統合ハードウェア データ構造の追加のキーワード」](#) を参照してください。

**xget\_hw\_value** <handle>

**説明** 指定したハンドルに関連する値を取得します。

**引数** <handle> : ハンドルのタイプです。  
 <handle> のタイプが IP インスタンスの場合、返される値は IP モジュール名です。たとえば、ハンドルが MHS ファイルの MicroBlaze™ インスタンスを参照する場合、IP 名である microblaze が返されます。パラメータ ハンドルからパラメータの値を取得する場合にも、このコマンドを使用します。

## ハードウェア API を使用する Tcl の例

次に、ハードウェア API Tcl コマンドを使用する Tcl の例を示します。

### 例 1

この例では、特定の IPTYPE の IP のリストを取得する方法を示します。EDK レポジトリに含まれる各 IP には、MPD ファイルの IPTYPE オプションで指定する IP タイプが対応しています。merged\_mhs\_instance には、MHS ファイルと MPD ファイルの両方からの情報が含まれます。特定の IPTYPE の IP のリストを取得するには、次の手順に従います。

1. merged\_mhs\_instance を使用してすべての IP のリストを取得します。
2. このリストに対して繰り返し処理を実行して各 IP の OPTION IPTYPE の値を取得し、指定の IP タイプと比較します。

次のコードは、特定の IP の IPTYPE を取得する方法を示します。

```
## Procedure to get a list of IPs of a particular IPTYPE
proc xget_ipinst_handle_list_for_iptype {merged_mhs_handle iptype}
{
  ##Get a list of all IPs
  set ipinst_list [xget_hw_ipinst_handle $merged_mhs_handle "*"]
  set ret_list ""
  foreach ipinst $ipinst_list {
    ## Get the value of the IPTYPE Option.
    set curiptype [xget_hw_option_value $ipinst "IPTYPE"]
    ##if curiptype matches the given iptype, then add it to
    ## the list that this proc returns.
    if {[string compare -nocase $curiptype $iptype] == 0}{
      lappend ret_list $ipinst
    }
  }
  return $ret_list
}
```

### 例 2

この例では、デザインのメモリ コントローラであるコアのリストを取得する方法を示します。メモリ コントローラ コアには、アドレス パラメータに ADDR\_TYPE = MEMORY というタグが付いています。

```
## Procedure to get a list of memory controllers in a design.
proc xget_hw_memory_controller_handles { merged_mhs } {
  set ret_list ""

  # Gets all MhsInsts in the system
  set mhsinsts [xget_hw_ipinst_handle $merged_mhs "*"]

  # Loop through each MhsInst and determine if it has
  # "ADDR_TYPE = MEMORY" in the parameters.

  foreach mhsinst $mhsinsts {

    # Gets all parameters of the IP
    set params [xget_hw_parameter_handle $mhsinst "*"]

    # Loop through each param and find tag "ADDR_TYPE = MEMORY"
```

```

foreach param $params {
    if {$param == 0} {
        continue
    } elseif {$param == ""} {
        continue
    }
    set addrTypeValue [xget_hw_subproperty_value $param"ADDR_TYPE"]

    # Found tag! Add MhsInst to list and break to go to next MhsInst
    if {[string compare -nocase $addrTypeValue "MEMORY"] == 0} {
        lappend ret_list $mhsinst
        break
    }
}

return $ret_list
}

```

## アドバンス書き込みアクセス API

アドバンス書き込みアクセス API は、メモリの MHS オブジェクトを変更します。これらのコマンドは、オリジナル MHS ハンドルおよび MHS ハンドルから取得されるハンドルを処理します。書き込みアクセス API はプロジェクトの作成でのみ使用可能で、Platgen フローではディスエーブルになっています。

### ハードウェア アドバンス書き込みアクセス ハードウェア API の一覧

次の表に、ハードウェア アドバンス書き込みアクセス API の一覧を示します。この表で API のリンクをクリックすると、その説明にジャンプします。

表 C-2 : ハードウェア アドバンス書き込みアクセス API

#### 追加コマンド

```

xadd_hw_hdl_srcfile <ipinst_handle> <fileuse> <filename> <hdl_lang>
xadd_hw_ipinst_busif <ipinst_handle> <busif_name> <busif_value>
xadd_hw_ipinst_port <ipinst_handle> <port_name> <connector_name>
xadd_hw_ipinst <mhs_handle> <inst_name> <ip_name> <hw_ver>
xadd_hw_ipinst_parameter <ipinst_handle> <param_name> <param_value>
xadd_hw_subproperty <prop_handle> <subprop_name> <subprop_value>
xadd_hw_toplevel_port <mhs_handle> <port_name> <connector_name> <direction>

```

#### 削除コマンド

```

xdel_hw_ipinst <mhs_handle> <inst_name>
xdel_hw_ipinst_busif <ipinst_handle> <busif_name>
xdel_hw_ipinst_port <ipinst_handle> <port_name>
xdel_hw_ipinst_parameter <ipinst_handle> <param_name>
xdel_hw_subproperty <prop_handle> <subprop_name>
xdel_hw_toplevel_port <mhs_handle> <port_name>

```

#### 変更コマンド

```

xset_hw_parameter_value <busif_handle> <busif_value>
xset_hw_port_value <port_handle> <port_value>
xset_hw_busif_value <busif_handle> <busif_value>

```



## ハードウェア アドバンス書き込みアクセス API の説明

### 追加コマンド

---

```
xadd_hw_hdl_srcfile <ipinst_handle> <fileuse> <filename>
<hdlldlang>
```

**説明** PAO に HDL ファイルを追加します。この API は、Platgen や Simgen などのパッチ ツールでのみ使用し、デザイン入力方法として xps パッチで使用しないでください。

VHDL ファイルを追加する場合、インスタンス特定のカスタマイズであることが想定され、<instname>\_<wrapper>\_<hwver> という論理ライブラリに追加されます。

VHDL ファイルは、<projdir>/hdl/elaborate/<instname>\_<wrapper>\_<hwver> ディレクトリに生成する必要があります。

Verilog ではライブラリは使用されませんが、ファイルを特定のディレクトリ構造で特定の場所に生成する必要があります。

**引数** <ipinst\_handle> : IP インスタンスのハンドルです。  
 <fileuse> : lib、synlib、simlib のいずれかです。  
 <filename> : 指定のファイル名です。  
 <hdlldlang> : vhd1 または verilog です。

**例**

```
xadd_hw_hdl_srcfile $ipinst_handle "lib"
"xps_central_dma.vhd" "vhd1"
```

---

```
xadd_hw_ipinst_busif <ipinst_handle> <busif_name> <busif_value>
```

**説明** <busif\_name> および <busif\_value> で指定したバス インターフェイスを作成し、<ipinst\_handle> で指定した IP インスタンスに追加します。処理が正しく実行された場合は新しく作成されたバス インターフェイスへのハンドルが返され、正しく実行されなかった場合は NULL が返されます。

**引数** <ipinst\_handle> : バス インターフェイスを追加する IP インスタンスへのハンドルです。  
 <busif\_name> : バス インターフェイスの名前です。  
 <busif\_value> : バス インターフェイスの値です。

**例** MicroBlaze からの ILMB バス インターフェイスを ilmb\_0 バスに接続  

```
xadd_hw_ipinst_busif $mb_handle "ILMB" "ilmb_0"
```

```
xadd_hw_ipinst <mhs_handle> <inst_name> <ip_name> <hw_ver>
```

**説明**                   新しい MHS インスタンスを <mhs\_handle> で指定した MHS に追加します。処理が正しく実行された場合は新しく作成されたインスタンスへのハンドルが返され、正しく実行されなかった場合は NULL が返されます。

**引数**                   <mhs\_handle>: MHS インスタンスを追加する MHS へのハンドルです。  
                           <inst\_name>: 追加する IP インスタンスの名前です。  
                           <ip\_name>: 追加する IP の名前です。  
                           <hw\_ver>: 追加する IP のバージョンです。

**例**                    mblaze というインスタンス名の Microblaze v7.00.a IP を MHS に追加

```
                          xadd_hw_ipinst $mhs_handle "mblaze" "microblaze"
                          "7.00.a"
```

```
xadd_hw_ipinst_port <ipinst_handle> <port_name> <connector_name>
```

**説明**                   <port\_name> および <connector\_name> で指定したポートを作成し、<ipinst\_handle> で指定した IP インスタンスに追加します。処理が正しく実行された場合は新しく作成されたポートへのハンドルが返され、正しく実行されなかった場合は NULL が返されます。

**引数**                   <inst\_handle>: ポートを追加する IP インスタンスへのハンドルです。  
                           <port\_name>: ポートの名前です。  
                           <connector\_name>: コネクタの名前です。

**例**                    MicroBlaze インスタンスにクロック ポートを追加し、sys\_clk\_s 信号に接続

```
                          xadd_hw_ipinst_port $mb_handle "Clk" "sys_clk_s"
```

```
xadd_hw_ipinst_parameter <ipinst_handle> <param_name>
<param_value>
```

**説明**                   <param\_name> および <param\_value> で指定したパラメータを作成し、<ipinst\_handle> で指定した IP インスタンスに追加します。処理が正しく実行された場合は新しく作成されたパラメータへのハンドルが返され、正しく実行されなかった場合は NULL が返されます。

**引数**                   <inst\_handle>: パラメータを追加する IP インスタンスへのハンドルです。  
                           <param\_name>: パラメータの名前です。  
                           <param\_value>: パラメータの値です。

**例**                    MicroBlaze インスタンスに C\_DEBUG\_ENABLED パラメータを追加し、値を 1 に設定

```
                          xadd_hw_ipinst_parameter $mb_handle "C_DEBUG_ENABLED"
                          "1"
```

**xadd\_hw\_subproperty** <prop\_handle> <subprop\_name> <subprop\_value>

**説明** プロパティ (パラメータ、ポート、またはバス インターフェイス) にサブプロパティを追加します。

**引数** <prop\_handle> : パラメータ、ポート、またはバス インターフェイスへのハンドルです。

<subprop\_name> : サブプロパティの名前です。

<subprop\_value> : サブプロパティの値です。サブプロパティのリストは、『Platform Specification Format Reference Manual』の「Microprocessor Peripheral Definition (MPD)」および [270 ページの「統合ハードウェア データ構造の追加のキーワード」](#) を参照してください。

**例** ポートに DIR を追加

```
xadd_hw_subproperty $port_handle "DIR" "I"
```

---

**xadd\_hw\_toplevel\_port** <mhs\_handle> <port\_name> <connector\_name>  
<direction>

**説明** 新しい最上位ポートを <mhs\_handle> で指定した MHS に追加します。処理が正しく実行された場合は新しく作成されたポートへのハンドルが返され、正しく実行されなかった場合は NULL が返されます。

**引数** <mhs\_handle> : 最上位ポートを追加する MHS へのハンドルです。

<port\_name> : 追加するポートの名前です。

<connector\_name> : コネクタの名前です。

<direction> : ポートの方向 (I、O、または IO) です。

**例** コネクタ dcm\_clk\_s を持つ sys\_clk\_pin という名前の最上位ポートを追加

```
xadd_hw_toplevel_port $mhs_handle "sys_clk_pin"  
"dcm_clk_s" "I"
```

## 削除コマンド

---

```
xdel_hw_ipinst <mhs_handle> <inst_name>
```

**説明** 指定した名前の IP インスタンスを削除します。

**引数** <mhs\_handle> : オリジナル MHS へのハンドルです。  
<inst\_name> : 削除するインスタンスの名前です。

**例** mymb という名前のインスタンスを削除  
xdel\_hw\_ipinst \$mhs\_handle "mymb"

---

```
xdel_hw_ipinst_busif <ipinst_handle> <busif_name>
```

**説明** IP インスタンス ハンドル上の指定のバス インターフェイスを削除します。

**引数** <ipinst\_handle> : IP インスタンスのハンドルです。  
<busif\_name> : 削除するバス インターフェイスの名前です。

**例** MicroBlaze インスタンスから ILMB バス インターフェイスを削除  
xdel\_hw\_ipinst\_busif \$mb\_handle "ILMB"

---

```
xdel_hw_ipinst_port <ipinst_handle> <port_name>
```

**説明** IP インスタンス ハンドル上の指定のポートを削除します。

**引数** <ipinst\_handle> : IP インスタンスのハンドルです。  
<port\_name> : 削除するポートの名前です。

**例** MicroBlaze インスタンス上の Clk ポートを削除  
xdel\_hw\_ipinst\_port \$mb\_handle "Clk"

---

```
xdel_hw_ipinst_parameter <ipinst_handle> <param_name>
```

**説明** IP インスタンス ハンドル上の指定のパラメータを削除します。

**引数** <ipinst\_handle> : IP インスタンスへのハンドルです。  
<param\_name> : 削除するパラメータの名前です。

**例** MicroBlaze インスタンスから C\_DEBUG\_ENABLED パラメータを削除  
xdel\_hw\_ipinst\_parameter \$mb\_handle "C\_DEBUG\_ENABLED"

**xdel\_hw\_subproperty** *<prop\_handle>* *<subprop\_name>*

説明 プロパティ ハンドルから指定のサブプロパティを削除します。

引数 *<prop\_handle>*: パラメータ、ポート、またはバス インターフェイスへのハンドルです。

*<subprop\_name>*: サブプロパティの名前です。

例 ポートから SIGIS サブプロパティを削除

```
xdel_hw_subproperty $port_handle "SIGIS"
```

---

**xdel\_hw\_toplevel\_port** *<mhs\_handle>* *<port\_name>*

説明 指定した名前の最上位ポートを削除します。

引数 *<mhs\_handle>*: オリジナル MHS へのハンドルです。

*<port\_name>*: 削除するポートの名前です。

例 sys\_clk\_pin という名前の最上位ポートを削除

```
xdel_hw_toplevel_port $mhs_handle "sys_clk_pin"
```

#### 変更コマンド

**xset\_hw\_parameter\_value** *<busif\_handle>* *<busif\_value>*

説明 パラメータの値を指定値に設定します。

引数 *<port\_handle>*: 値を設定するポートのハンドルです。

*<port\_value>*: 設定する値です。

例 パラメータの値を 2 に設定

```
xset_hw_parameter_value $param_handle 2
```

---

**xset\_hw\_port\_value** *<port\_handle>* *<port\_value>*

説明 ポートの値を指定値に設定します。

引数 *<port\_handle>*: 値を設定するポートのハンドルです。

*<port\_value>*: 設定する値です。

例 ポートの値を my\_connection に設定

```
xset_hw_port_value $port_handle "my_connection"
```

```
xset_hw_busif_value <busif_handle> <busif_value>
```

説明 バス インターフェイスの値を指定値に設定します。

引数 <busif\_handle>: 値を設定するバス インターフェイスのハンドルです。  
<busif\_value>: 設定する値です。

例 バス インターフェイスの値を my\_bus に設定  

```
xset_hw_busif_value $busif_handle "my_bus"
```

## ソフトウェア Tcl コマンド

このセクションでは、EDK ソフトウェア Tcl API で使用される用語を説明し、使用可能な Tcl ソフトウェア API を示します。

### ソフトウェア API で使用される用語

次の表に、ソフトウェア Tcl API で使用される用語を説明します。

表 C-3 : ソフトウェア API の用語

オリジナル MSS	MSS 情報のみを含むハンドルで、MDD または MLD 情報は含まれません。MSS でドライバまたはライブラリ パラメータを指定していない場合、そのパラメータは含まれません。
統合 MSS	MSS と MDD または MLD の情報を含むハンドルです。このデータ構造オブジェクトは、MDD または MLD 情報を MSS 情報と統合することにより形成されます。
オリジナル プロセッサ インスタンス	オリジナル MSS から取得されるプロセッサ ハンドルで、MSS に示される情報のみが含まれます。
統合プロセッサ	統合 MSS から取得されるプロセッサ ハンドルで、プロセッサからアクセス可能な統合ドライバおよび統合ライブラリのリスト、プロセッサに割り当てられた統合 OS インスタンスなど、MDD 情報およびその他の接続情報が含まれます。このハンドルは、Libgen を実行した後に使用可能になります。
オリジナル ドライバ インスタンス ハンドル	オリジナル MSS から取得されるドライバ ハンドルで、MSS に示される情報のみが含まれます。
統合ドライバ	使用するペリフェラルのリストと、すべての統合パラメータ値を含むドライバです。統合プロセッサ オブジェクトで示される接続情報が含まれます。
オリジナル OS インスタンス ハンドル	オリジナル MSS から取得される OS ハンドルで、MSS に示される情報のみが含まれます。
統合 OS ハンドル	統合 MSS から取得される OS ハンドルで、MLD に示される情報も含まれます。

表 C-3 : ソフトウェア API の用語 (続き)

オリジナル ライブラリ インスタンス	オリジナル MSS から取得されるライブラリ ハンドルで、MSS に示される情報のみが含まれます。
統合ライブラリ	統合 MSS から取得されるライブラリ ハンドルで、MLD に示される情報も含まれます。

## ソフトウェア読み出しアクセス API

このセクションでは、ソフトウェア読み出しアクセス API をリストしています。次の表は API の一覧で、リンクをクリックするとその説明にジャンプします。この一覧の後に、各 API の説明を示します。

### ソフトウェア読み出しアクセス API の一覧

表 C-4 : ソフトウェア読み出しアクセス API

```

xget_sw_array_handle <handle> <array_name>
xget_libgen_proc_handle
xget_sw_array_element_handle <handle> <element_name>
xget_sw_driver_handle <mss_handle> <driver_name>
xget_sw_driver_handle_for_ipinst <merged_processor_handle> <ipinst_name>
xget_sw_function_handle <handle> <function_name>
xget_sw_ipinst_handle <handle> <ipinst_name>
xget_sw_ipinst_handle_from_processor <ipinst_name> <merged_processor_handle>
xget_sw_iplist_for_driver <merged_driver_handle>
xget_sw_interface_handle <handle> <interface_name>
xget_sw_library_handle <mss_handle> <library_name>
xget_sw_mdd_handle <handle>
xget_sw_mld_handle <handle>
xget_sw_name <handle>
xget_sw_parameter_handle <handle> <parameter_name>
xget_sw_parameter_value <handle> <parameter_name>
xget_sw_os_handle <mss_handle> <os_name>
xget_sw_option_handle <handle> <option_name>
xget_sw_option_value <handle> <option_name>
xget_sw_parent_handle <handle>
xget_sw_processor_handle <mss_handle> <processor_name>
xget_sw_property_handle <handle> <property_name>
xget_sw_subproperty_handle <property_handle> <subprop_name>
xget_sw_property_value <handle> <property_name>
xget_sw_subproperty_value <property_handle> <subprop_name>

```

## ソフトウェア読み出しアクセス API の説明

---

### **xget\_libgen\_proc\_handle**

説明	Libgen が現在実行している統合プロセッサへのハンドルを返します。この API は、Libgen を実行中にのみ使用可能です。
引数	なし
例	ドライバ Tcl ファイルで、Libgen が実行されている統合プロセッサ インスタンスを取得 <pre>set proc_handle [xget_libgen_proc_handle]</pre>

---

### **xget\_sw\_array\_handle** <handle> <array\_name>

説明	ハンドルに関連するアレイへのハンドルを返します。
引数	<handle>: ハンドルのタイプです。有効なハンドル タイプは、MDD、MLD、MSS、統合 MSS、オリジナルドライバ インスタンス、統合ドライバ、オリジナルプロセッサ インスタンス、統合プロセッサ、オリジナル OS インスタンス、統合 OS、オリジナルライブラリ インスタンス、または統合ライブラリです。 <array_name>: アレイの名前です。アスタリスク (*) を指定すると、アレイハンドルのリストが返されます。個々のアレイハンドルにアクセスするには、Tcl のリストに対して繰り返し処理を実行します。
例	MSS ハンドルに関連するアレイ ハンドルのリストを取得 <pre>set array_handle [xget_sw_array_handle \$mss_handle *]</pre>

---

### **xget\_sw\_array\_element\_handle** <handle> <element\_name>

説明	ハンドルに関連するアレイ エlement へのハンドルを返します。
引数	<handle>: ハンドルのタイプです。有効なハンドル タイプは、アレイまたはアレイ インスタンスです。 <element_name>: アレイ エlement です。アスタリスク (*) を指定すると、Element ハンドルのリストが返されます。個々のElement ハンドルにアクセスするには、Tcl のリストに対して繰り返し処理を実行します。
例	<pre>set elem_handle [xget_sw_array_element_handle \$array_handle "myelement"]</pre>

---



---

```
xget_sw_driver_handle <mss_handle> <driver_name>
```

**説明** <mss\_handle> で指定した **MSS** ハンドルに関連するドライバ (<driver\_name>) へのハンドルを返します。

**引数** <driver\_name>: ドライバの名前です。  
<mss\_handle>: MSS ファイルへのハンドルです。

**例**

```
set drv_handle [xget_sw_driver_handle $mss_handle  
"<driver_name>"]
```

---

```
xget_sw_driver_handle_for_ipinst <merged_processor_handle>  
<ipinst_name>
```

**説明** <ipinst\_name> で指定した **IP** インスタンスに割り当てられた統合ドライバオブジェクトへのハンドルを返します。統合ドライバオブジェクトは、統合ドライバを使用するペリフェラルのリストとパラメータ値を含むドライバで、統合プロセッサオブジェクトで示される接続情報が含まれます。

**引数** <merged\_processor\_handle>: **Libgen** を実行中にのみ使用可能な統合プロセッサオブジェクトです。xget\_libgen\_proc\_handle API を使用すると取得できます。  
<ipinst\_name>: 統合ドライバ情報を取得する **IP** インスタンスです。

**例** 割り込みコントローラに接続されている **IP** ソースのドライバを取得

```
set sw_proc_handle [xget_libgen_proc_handle]  
set ip_driver [xget_sw_driver_handle_for_ipinst  
$sw_proc_handle $ip_name]
```

**メモ:** この例は、intc ドライバ Tcl ファイルからのものです。

---

```
xget_sw_function_handle <handle> <function_name>
```

**説明** <function\_name> で指定したハンドルに関連する関数へのハンドルを返します。

**引数** <handle>: インターフェイスハンドルです。  
<function\_name>: 関数の名前です。アスタリスク (\*) を指定すると、関数ハンドルのリストが返されます。個々の関数ハンドルにアクセスするには、Tcl のリストに対して繰り返し処理を実行します。

**例**

```
set func_handle [xget_sw_function_handle $swif_handle  
"<function_name>"]
```

**xget\_sw\_ipinst\_handle** <handle> <ipinst\_name>

説明 <ipinst\_name> で指定した IP インスタンスへのハンドルを返します。

引数 <handle> : 統合プロセッサ インスタンスです。

<ipinst\_name> : IP インスタンスの名前です。

例 set ipinst [xget\_sw\_ipinst\_handle \$mpi\_handle "<ipname>"]

**xget\_sw\_iplist\_for\_driver** <merged\_driver\_handle>

説明 <merged\_driver\_handle> に関連するドライバに割り当てられたペリフェラルへのハンドルのリストを返します。

引数 <merged\_driver\_handle> : Libgen の実行中のみ使用可能なドライバハンドルです。xget\_sw\_driver\_handle\_for\_ipinst API を使用すると取得できます。

例 uart\_driver\_handle を使用するドライバ uartlite を使用するすべてのペリフェラルのリストを取得

```
set periphs [xget_sw_iplist_for_driver
$uart_driver_handle]
```

**xget\_sw\_ipinst\_handle\_from\_processor** <ipinst\_name>  
<merged\_processor\_handle>

説明 統合プロセッサ ハンドルに関連する IP インスタンスへのハンドルを返します。

引数 <ipinst\_name> : 統合プロセッサ ハンドルに関連する IP インスタンスです。

<merged\_processor\_handle> : 統合プロセッサの名前です。

xget\_libgen\_proc\_handle API を使用すると取得できます。

例 my\_plb\_ethernet という名前のインスタンスへのハンドルを取得

```
set sw_proc_handle [xget_libgen_proc_handle]
set inst_handle [xget_sw_ipinst_handle_from_processor
$sw_proc_handle "my_plb_ethernet"]
```

**xget\_sw\_interface\_handle** <handle> <interface\_name>

説明	<interface_name> で指定したハンドルに関連するインターフェイスへのハンドルを返します。
引数	<p>&lt;handle&gt;: インターフェイス ハンドルです。有効なハンドル タイプは、MDD、MLD、オリジナルドライバ インスタンス、統合ドライバ、オリジナルプロセッサ インスタンス、統合プロセッサ、オリジナル OS インスタンス、統合 OS、オリジナル ライブラリ インスタンス、または統合ライブラリです。</p> <p>&lt;interface_name&gt;: インターフェイスです。アスタリスク(*)を指定すると、インターフェイス ハンドルのリストが返されます。個々のインターフェイス ハンドルにアクセスするには、Tcl のリストに対して繰り返し処理を実行します。</p>
例	<pre>set swif_handle [xget_sw_interface_handle \$mld_handle "&lt;interface_name&gt;"]</pre>

**xget\_sw\_library\_handle** <mss\_handle> <library\_name>

説明	<mss_handle> で指定した MSS ハンドルに関連するライブラリ (<library_name>) へのハンドルを返します。
引数	<p>&lt;library_name&gt;: ライブラリの名前です。</p> <p>&lt;mss_handle&gt;: MSS ファイルへのハンドルです。</p>
例	<pre>set lib_handle [xget_sw_library_handle \$mss_handle "&lt;library_name&gt;"]</pre>

**xget\_sw\_mdd\_handle** <handle>

説明	指定したドライバまたはプロセッサ インスタンスに関連する MDD オブジェクトへのハンドルを返します。
引数	<handle>: ハンドルのタイプです。有効なタイプは、オリジナルドライバ インスタンス、オリジナルプロセッサ インスタンス、統合ドライバ、または統合プロセッサです。
例	<pre>set mdd_handle [xget_sw_mdd_handle \$drv_handle]</pre>

**xget\_sw\_mld\_handle** <handle>

説明	指定した OS またはライブラリ インスタンスに関連する MLD オブジェクトへのハンドルを返します。
引数	<handle>: ハンドルのタイプです。有効なタイプは、オリジナル OS インスタンス、オリジナルライブラリ インスタンス、統合 OS、または統合ライブラリです。
例	<pre>set mld_handle [xget_sw_mld_handle \$os_handle]</pre>

---

**xget\_sw\_name** <handle>

**説明** 指定したハンドルの名前を返します。たとえば、MSS ファイル内の standalone という OS インスタンスの場合、standalone が返されます。パラメータ ハンドルからパラメータの名前を取得する場合にも、このコマンドを使用します。

**引数** <handle> : ハンドルのタイプです。

**例** OS インスタンスとその名前を取得  

```
set os_name [xget_sw_name $os_handle]
```

---

**xget\_sw\_parameter\_handle** <handle> <parameter\_name>

**説明** ハンドルに関連するパラメータへのハンドルを返します。

**引数** <handle> : ハンドルのタイプです。有効なハンドル タイプは、MDD、MLD、MSS、統合 MSS、オリジナルドライバ インスタンス、統合ドライバ、オリジナルプロセッサ インスタンス、統合プロセッサ、オリジナル OS インスタンス、統合 OS、オリジナルライブラリ インスタンス、または統合ライブラリです。

**メモ** : ハンドルのタイプによって、オリジナルまたは統合パラメータ ハンドルが返されます。

<parameter\_name> : パラメータです。アスタリスク (\*) を指定すると、パラメータ ハンドルのリストが返されます。個々のパラメータ ハンドルにアクセスするには、Tcl のリストに対して繰り返し処理を実行します。

**例** OS インスタンスの MSS ファイルに含まれる stdin というパラメータのハンドルを、os\_handle から取得  

```
set stdin_handle [xget_sw_parameter_handle $os_handle]
```

---

**xget\_sw\_parameter\_value** <handle> <parameter\_name>

**説明** 指定したパラメータの値を返します。

**引数** <handle> : ハンドルのタイプです。

<parameter\_name> : 指定のパラメータです。

**例** OS インスタンスの MSS ファイルで uart0 と指定されている stdin というパラメータの値を取得 (戻り値はUART0)  

```
set stdin_value [xget_sw_parameter_value $os_handle]
```

**xget\_sw\_option\_handle** <handle> <option\_name>

- 説明**                    ハンドルに関連するオプションへのハンドルを返します。
- 引数**                    <handle>: ハンドルのタイプです。有効なハンドル タイプは、MDD、MLD、オリジナルドライバ インスタンス、統合ドライバ、オリジナル プロセッサ インスタンス、統合プロセッサ、オリジナル OS インスタンス、統合 OS、オリジナル ライブラリ インスタンス、または統合ライブラリです。
- <option\_name>: オプションの名前です。アスタリスク (\*) を指定すると、オプション ハンドルのリストが返されます。個々のオプション ハンドルにアクセスするには、Tcl のリストに対して繰り返し処理を実行します。
- 例**                    OS インスタンスの MSS ファイルで standalone\_drc と指定されている DRC というオプションのハンドルを、os\_handle から取得
- ```
set drc_handle [xget_sw_option_handle $os_handle]
```

---

**xget\_sw\_option\_value** <handle> <option\_name>

- 説明**                    <handle> に関連した指定の <option\_name> の値を返します。
- 引数**                    <handle>: ハンドルのタイプです。
- <option\_name>: オプションの名前です。
- 例**                    OS インスタンスの MLD ファイルで standalone\_drc と指定されている drc というオプションの値を、os\_handle から取得
- ```
set drc_value [xget_sw_option_value $os_handle]
```

---

**xget\_sw\_os\_handle** <mss\_handle> <os\_name>

- 説明**                    <mss\_handle> で指定した MSS ハンドルに関連する OS (<os\_name>) へのハンドルを返します。
- 引数**                    <os\_name>: OS の名前です。
- <mss\_handle>: MSS ファイルへのハンドルです。
- 例**                    set os\_handle [xget\_sw\_os\_handle \$mss\_handle "<os\_name>"]

**xget\_sw\_parent\_handle** <handle>

説明	指定したハンドルの親へのハンドルを返します。
引数	<p>&lt;handle&gt;: ハンドルのタイプです。親ハンドルのタイプは、指定したハンドルのタイプによって異なります。指定したハンドルが統合ハンドルの場合は、この API で取得される親も統合ハンドルです。指定したハンドル タイプによる親ハンドルのタイプは、次のとおりです。</p> <ul style="list-style-type: none"> <li>• <b>PARAMETER</b>: 親は MDD、MLD、プロセッサ インスタンス、ドライバ インスタンス、OS インスタンス、ライブラリ インスタンス、統合プロセッサ インスタンス、統合ドライバ インスタンス、統合 OS インスタンス、または統合ライブラリ インスタンス オブジェクトです。</li> <li>• <b>ARRAY</b>: 親は MDD、MLD、ドライバ インスタンス、プロセッサ インスタンス、OS インスタンス、ライブラリ インスタンス、統合インスタンス (プロセッサ インスタンス、OS インスタンス、ライブラリ インスタンス、ドライバ インスタンス) のいずれか、または <b>MSS</b> オブジェクトです。</li> <li>• <b>ELEMENT</b>: 親はアレイ オブジェクトです。</li> <li>• <b>INTERFACE</b>: 親は MDD、MLD、ドライバ インスタンス、プロセッサ インスタンス、OS インスタンス、ライブラリ インスタンス、または統合インスタンス (プロセッサ インスタンス、OS インスタンス、ライブラリ インスタンス、ドライバ インスタンス) のいずれかです。</li> <li>• <b>FUNCTION</b>: 親はインターフェイス オブジェクトです。</li> <li>• <b>OPTION</b>: 親は MDD、MLD、ドライバ インスタンス、プロセッサ インスタンス、OS インスタンス、ライブラリ インスタンス、または統合インスタンス (プロセッサ インスタンス、OS インスタンス、ライブラリ インスタンス、ドライバ インスタンス) のいずれかです。</li> <li>• <b>DRVINST</b>: 親は MSS または統合 MSS オブジェクトです。</li> <li>• <b>PROCINST</b>: 親は MSS または統合 MSS オブジェクトです。</li> <li>• <b>OSINST</b>: 親は MSS または統合 MSS オブジェクトです。</li> <li>• <b>LIBINST</b>: 親は MSS または統合 MSS オブジェクトです。</li> <li>• <b>MSS、MDD、または MLD</b>: 親は NULL ハンドルです。</li> </ul>
例	<p>パラメータの親を取得</p> <pre>set parent_handle [xget_sw_parent_handle \$param_handle]</pre>

**xget\_sw\_processor\_handle** <mss\_handle> <processor\_name>

説明	<mss_handle> で指定した <b>MSS</b> ハンドルに関連するプロセッサ (<processor_name>) へのハンドルを返します。
引数	<p>&lt;processor_name&gt;: &lt;mss_handle&gt; に関連するプロセッサの名前です。</p> <p>&lt;mss_handle&gt;: <b>MSS</b> ファイルへのハンドルです。</p>
例	<pre>set proc_handle [xget_sw_processor_handle \$mss_handle "&lt;processor_name&gt;"]</pre>

**xget\_sw\_property\_handle** <handle> <property\_name>

**説明**                    ハンドルに関連する <property\_name> で指定したプロパティへのハンドルを返します。有効なハンドル タイプは、インターフェイス、アレイ、および関数です。

**引数**                    <handle> : ハンドルのタイプです。有効なタイプは、インターフェイス、アレイ、および関数です。

                         <property\_name> : プロパティの名前です。アスタリスク (\*) を指定すると、プロパティ ハンドルのリストが返されます。個々のプロパティ ハンドルにアクセスするには、Tcl のリストに対して繰り返し処理を実行します。

**例**                      set prop\_handle [xget\_sw\_property\_handle \$swif\_handle "HEADER"]

---

**xget\_sw\_property\_value** <handle> <property\_name>

**説明**                    指定したプロパティの値を返します。

**引数**                    <handle> : ハンドルのタイプです。

                         <property\_name> : プロパティの名前です。

**例**                      set prop\_val [xget\_sw\_property\_value \$swif\_handle "HEADER"]

---

**xget\_sw\_subproperty\_handle** <property\_handle> <subprop\_name>

**説明**                    <property\_handle> で指定したプロパティ ハンドルに関連するサブプロパティへのハンドルを返します。

**引数**                    <property\_handle> : プロパティの名前です。有効なオプションは、PARAMETER、ARRAY、ELEMENT、FUNCTION、PROPERTY、INTERFACE、または OPTION です。

                         <subprop\_name> : サブプロパティの名前です。

**例**                      set subprop\_handle [xget\_sw\_subproperty\_handle \$prop\_handle "<subprop\_name>"]

---

**xget\_sw\_subproperty\_value** <property\_handle> <subprop\_name>

**説明**                    指定したサブプロパティの値を返します。

**引数**                    <property\_handle> : プロパティの名前です。

                         <subprop\_name> : サブプロパティの名前です。

**例**                      set subprop\_value [xget\_sw\_subproperty\_handle \$prop\_handle "<subprop\_name>"]

**xget\_sw\_value** <handle>

説明	指定したハンドルに関連する値を取得します。ハンドルのタイプが <b>PARAMETER</b> の場合、値はパラメータの値です。
引数	<handle>: ハンドルのタイプです。
例	OS インスタンスの <b>MSS</b> ファイルで <b>UART0</b> と指定されている <b>stdin</b> というパラメータの値を取得 (戻り値は <b>uart0</b> ) <pre>set stdin_value [xget_sw_value \$stdin_param_handle]</pre>

## ハードウェア プラットフォーム生成の Tcl フロー

### 入力ファイル

Platgen、Simgen、Libgen など、ハードウェア プラットフォームを作成するツールは、**MHS** デザイン ファイルと **IP** データ ファイル (**MPD**) を使用します。内部的には、これらのツールはこれらのファイルに基づいてシステム ビューを作成します。デザインに含まれる各 **IP** には、**MPD** ファイルが関連付けられています。オプションで、**Tcl** ファイルを関連付けることもできます。**Tcl** ファイルは、**DRC** プロシージャやパラメータの計算を自動化するプロシージャを含めたり、またはその他のタスクを実行したりできます。ハードウェア プラットフォームの生成中に使用される **Tcl** ファイルは、コアのディレクトリに **MPD** ファイルと共に配置されています。ザイリンクスが提供するコアの **Tcl** ファイルは、<EDK インストール ディレクトリ>/hw/XilinxProcessorIPLib/pcores/<corename>/data/ ディレクトリにあります。

### ハードウェア プラットフォーム生成中に呼び出される Tcl プロシージャ

Platgen (および Libgen、Simgen、Bitinit などの EDK バッチ ツール) は、各 **IP** に関連する定義済みの **Tcl** プロシージャを実行して **DRC** を実行し、**IP** 上の一部のパラメータの値を算出します。**IP** の **Tcl** ファイルについては、『Platform Format Specification Reference Manual』を参照してください。235 ページの「その他のリソース」に、このマニュアルへのリンクがあります。

このセクションでは、**Tcl** プロシージャについて説明し、ユーザー **IP** に対して呼び出す方法を説明します。**Tcl** プロシージャは、次のように分類されます。

- **Tcl** プロシージャで実行されるアクション
  - ◆ **DRC**

システムに対して **DRC** を実行しますが、システムのステートは変更しません。これらのプロシージャで生成されるリターン コードは、**Platgen** によりキャプチャされます。**DRC** プロシージャでエラー ステータスが返された場合、**Platgen** でエラーが認識され、適切な時点で実行を停止します。
  - ◆ **UPDATE**

システムが正しいステートであると想定し、**Tcl API** を使用してデザイン データ構造を参照して特定のパラメータの値を算出します。ツールは、これらのプロシージャで返された文字列を使用して、デザインを **Tcl** で算出された値でアップデートします。
- **Tcl** プロシージャが呼び出されるハードウェア プラットフォーム作成の段階
  - ◆ **IPLEVEL**

ツール処理の初期段階で呼び出されます。これらのプロシージャでは、デザイン解析は実行されておらず、システム レベルの情報はないと想定されます。



#### ◆ SYSLEVEL

ツールでデザインのシステム レベルの解析が実行され、一部のパラメータがアップデートされた後に呼び出されます。これらのパラメータのリストは、第 5 章「Platform Specification Utility (PsfUtil)」の「予約済みパラメータ」を参照してください。IP の Tcl プロシージャでアップデートされるパラメータもあります。これらのパラメータは IP Tcl でのみ処理されるので、MPD の資料にはリストされていません。

各 Tcl プロシージャは、引数を 1 つ使用します。引数は、データ構造内の特定のタイプのハンドルです。ハンドル タイプは、Tcl プロシージャが関連付けられているオブジェクトのタイプによって異なります。パラメータに関連付けられている Tcl プロシージャには、そのパラメータへのハンドルが引数として指定されます。

IP に関連付けられている Tcl プロシージャには、デザインで使用する IP の特定のインスタンスへのハンドルが引数として指定されます。次に、IP インスタンスに対して呼び出すことができる Tcl プロシージャを示します。

**メモ :** Tcl プロシージャ名を指定する MPD タグ名は、前述した Tcl プロシージャのカテゴリを示します。

次の各タグは、MPD ファイルの名前と値のペアで、値はそのタグに関連付けられている Tcl プロシージャを指定します。IP の Tcl ファイルにその Tcl プロシージャが存在することを確認してください。

- ツール特定の Tcl 呼び出し
  - ◆ Platgen または Simgen に特定の呼び出しを指定できます。

#### MPD 内の Tcl プロシージャの実行順

MPD で指定されている Tcl プロシージャは、ハードウェア プラットフォームの生成中に次の順序で実行されます。

1. IPLEVEL\_UPDATE\_VALUE\_PROC (パラメータに対して)
2. IPLEVEL\_DRC\_PROC (パラメータに対して)
3. IPLEVEL\_DRC\_PROC (IP に対して、オプションで指定)
4. SYSLEVEL\_UPDATE\_VALUE\_PROC (パラメータに対して)
5. SYSLEVEL\_UPDATE\_PROC (IP に対して、オプションで指定)
6. SYSLEVEL\_DRC\_PROC (パラメータ、ポートに対して)
7. SYSLEVEL\_DRC\_PROC (IP に対して、オプションで指定)
8. FORMAT\_PROC (パラメータに対して)
9. ヘルパ コア Tcl プロシージャ

### システム レベル解析前のパラメータ用の UPDATE プロシージャ

パラメータ サブプロパティ IPLEVEL\_UPDATE\_VALUE\_PROC を使用すると、同じ IP のほかのパラメータに基づいてパラメータ値を算出する Tcl プロシージャを指定できます。入力ハンドルは、その IP の特定のインスタンスのパラメータ オブジェクトに関連します。

```
## MPD snippet
PARAMETER C_PARAM1 = 4, ...,
PARAMETER C_PARAM2 = 0, ..., IPLEVEL_UPDATE_VALUE_PROC = update_param2

## Tcl computes value based on other parameters on the IP
## Argument param_handle points to C_PARAM2 because the Tcl is
## associated with C_PARAM2
proc update_param2 {param_handle} {
    set retval 0;
    set mhsinst [xget_hw_parent_handle $param_handle]
    set paramlval [xget_hw_param_value $mhsinst "C_PARAM1"]
    if {$paramlval >= 4} {
        set retval 1;
    }
    return $retval
}
```

### システム レベル解析前のパラメータ用の DRC プロシージャ

パラメータ サブプロパティ IPLEVEL\_DRC\_PROC を使用すると、そのパラメータ専用の DRC を実行する Tcl プロシージャを指定できます。これらの DRC は、その IP のほかのパラメータ値とは独立している必要があります。

たとえば次の DRC は、パラメータに有効な値が指定されていることを確認するために使用します。入力ハンドルは、その IP の特定のインスタンスに対するパラメータ オブジェクトへのハンドルです。

```
## MPD snippet
PARAMETER C_PARAM1 = 0, ..., IPLEVEL_DRC_PROC = drc_param1

## Tcl snippet
## Argument param_handle points to C_PARAM1 since the Tcl is
## associated with C_PARAM1
proc drc_param1 {param_handle} {
    set paramlval [xget_hw_value $param_handle]
    if {$paramlval >= 5} {
        error "C_PARAM1 value should be less 5"
        return 1;
    } else {
        return 0;
    }
}
```

## システム レベル解析前の IP 用の DRC プロシージャ

OPTION IPLEVEL\_DRC\_PROC を使用すると、この DRC を実行する Tcl プロシージャを指定できます。このプロシージャは、IPLEVEL で DRC を実行するために使用します (2 つのパラメータ値が一致しているかなど)。ここで実行される DRC は、システム (MHS) での IP の使用とは独立しており、パラメータ、バス インターフェイス、ポート設定のみを使用する必要があります。入力ハンドルは、IP のインスタンスへのハンドルです。

```
## MPD Snippet
OPTION IPLEVEL_DRC_PROC = iplevel_drc
BUS_INTERFACE BUS = SPLB, BUS_STD = PLB, BUS_TYPE = SLAVE
PORT MYPORT = "", DIR = I

## Tcl snippet
proc iplevel_drc {ipinst_handle} {
    set splb_handle [xget_hw_busif_handle $ipinst_handle "SPLB"]
    set splb_conn [xget_hw_value $splb_handle]
    set myport_handle [xget_hw_port_handle "MYPORT"]
    set myport_conn [xget_hw_value $myport_handle]
    if {$splb_conn == "" || $myport_conn == ""} {
        error "Either busif SPLB or port MYPORT must be connected in the design"
    }
    return 1;
}
else {
    return 0;
}
}
```

## システム レベル解析後のパラメータ用の UPDATE プロシージャ

パラメータ サブプロパティ SYSLEVEL\_UPDATE\_VALUE\_PROC を使用すると、同じ IP のほかのパラメータに基づいてパラメータ値を算出する Tcl プロシージャを指定できます。入力ハンドルは、その IP の特定のインスタンスに対するパラメータ オブジェクトへのハンドルです。このプロシージャが呼び出されたときには、Platgen により算出されたシステム レベルのパラメータ (バス上の C\_NUM\_MASTERS など) は既に正しい値でアップデートされています。

```
## MPD snippet
PARAMETER C_PARAM1 = 5, ..., SYSLEVEL_UPDATE_VALUE_PROC =
sysupdate_param1

## Tcl snippet
proc sysupdate_param1 {param_handle} {
    set retval [somehow_compute_param1]
    return $retval;
}
```

### システム レベル解析後の IP インスタンス用の UPDATE プロシージャ

OPTION SYSLEVEL\_UPDATE\_PROC を使用すると、特定の IP に関連するアクションを実行できます。このプロシージャは、特定のパラメータではなく IP に関連付けられているので、特定のパラメータの値をアップデートするためには使用できません。

たとえば、このプロシージャを使用して、特定のディレクトリにある IP に関連付けられているファイルをコピーできます。入力ハンドルは、IP のインスタンスへのハンドルです。

```
## MPD Snippet
OPTION SYSLEVEL_UPDATE_PROC = syslevel_update_proc
## Tcl snippet
Proc myip_syslevel_update_proc {ipinst_handle} {
    ## do something
    return 0;
}
```

### システム レベル解析後のパラメータ用の DRC プロシージャ

タグ SYSLEVEL\_DRC\_PROC を使用すると、IP がシステムでどのように使用されるかに基づいて、完全な IP に DRC を実行する Tcl プロシージャを指定できます。入力ハンドルは、その IP の特定のインスタンスに対するパラメータ オブジェクトへのハンドルです。

```
PARAMETER C_MYPARAM = 5, ..., SYSLEVEL_DRC_PROC = sysdrc_myparam
```

### システム レベル解析後の IP 用の DRC プロシージャ

OPTION SYSLEVEL\_DRC\_PROC を使用すると、Platgen でシステム レベル情報がアップデートされた後に DRC を実行する Tcl プロシージャを指定できます。入力ハンドルは、IP のインスタンスへのハンドルです。たとえば、IP がインスタンシエートされている場合、このプロシージャでこの IP のインスタンスの数が制限内か、常に別の IP と共に使用されているか、別の IP とは共に使用されないかが確認されます。

```
## MPD Snippet
OPTION SYSLEVEL_DRC_PROC = syslevel_drc
BUS_INTERFACE BUS = SPLB, BUS_STD = PLB, BUS_TYPE = SLAVE
PORT MYPORT = "", DIR = 0
## Tcl snippet
proc syslevel_drc {ipinst_handle} {
    set myport_conn [xget_hw_port_value $ipinst_handle "MYPORT"]
    set mhs_handle [xget_hw_parent_handle $ipinst_handle]
    set sink_ports [xget_hw_connected_ports_handle $mhs_handle
$myport_conn "SINK"]
    if {[llength $sink_ports] > 5} {
        error "MYPORT should not drive more than 5 signals"
        return 1;
    }
    else {
        return 0;
    }
}
```

## Platgen 特定の呼び出し

共通の Tcl プロシージャがすべて呼び出された後、`OPTION PLATGEN_SYSLEVEL_UPDATE_PROC` が呼び出されます。特定のアクションが **Platgen** の実行でのみ行われるようにし、ほかのツールの実行では行われないようにする場合、このプロシージャを使用できます。

```
## MPD Snippet
OPTION PLATGEN_SYSLEVEL_UPDATE_PROC = platgen_syslevel_update
```

## Simgen 特定の呼び出し

共通の Tcl プロシージャがすべて呼び出された後、`OPTION SIMGEN_SYSLEVEL_UPDATE_PROC` が呼び出されます。特定のアクションが **Simgen** の実行でのみ行われるようにし、ほかのツールの実行では行われないようにする場合、このプロシージャを使用できます。

```
## MPD Snippet
OPTION SIMGEN_SYSLEVEL_UPDATE_PROC = simgen_syslevel_update
```

## FORMAT\_PROC

`FORMAT_PROC` キーワードは、パラメータの値をフォーマットする特殊なフォーマット プロシージャを提供できるようにする Tcl エントリ ポイントを定義します。

EDK ツールでは、**Verilog** および **VHDL** の 2 つの HDL 出力ファイルを生成できます。各フォーマットの構文では、処理に適した記述にするためパラメータ値を正規化する必要があります。たとえば、**Verilog** では大文字と小文字が区別され、文字列処理関数はありません。IP を開発する際に、この Tcl エントリ ポイントを使用して、HDL の要件に適合するよう文字列値をフォーマットするプロシージャを指定できます。詳細および例は、『**Platform Specification Format Reference Manual**』を参照してください。[235 ページの「その他のリソース」](#)に、このマニュアルへのリンクがあります。

## ヘルパ コア Tcl プロシージャ

ここに示すすべての Tcl プロシージャは、最上位コアで指定する必要があります。最上位コアでヘルパまたはライブラリ コアを使用している場合は、`SYSLEVEL_GENERIC_PROC` または `SYSLEVEL_ARCHSUPPORT_PROC` のいずれかのプロシージャを使用すると、それらのヘルパ コアに特定の Tcl プロシージャを実行できます。ヘルパ コアに特定の Tcl プロシージャは、`data` ディレクトリで指定し、ほかの `PSF` ファイルと同じ命名規則に従う必要があります。たとえば、`proc_common_v1_00_a` コアの Tcl ファイルは、対応する命名規則に従って、`proc_common_v2_1_0.tcl` のような名前を付ける必要があります。

- `SYSLEVEL_GENERIC_PROC` プロシージャ：すべてのメッセージを表示するジェネリック プロシージャです。
- `SYSLEVEL_ARCHSUPPORT_PROC` プロシージャ：廃止予定のヘルパ コアを通知するために使用します。

たとえば、`proc_common_v1_00_a` コアが廃止予定である場合、次のように `proc_common_v1_00_a` コアの `proc_common_v2_1_0.tcl` ファイルに含まれるヘルパコアの Tcl ファイルにこのプロシージャを含めることにより、このコアが廃止予定でない最上位コアで使用されるたびにツールでメッセージが表示されるようにすることができます。

```
proc syslevel_archsupport_proc { mhsinst } {
    print_deprecated_helper_core_message $mhsinst proc_common_v1_00_a
}
```

PRINT\_DEPRECATED\_HELPER\_CORE\_MESSAGE プロシージャは、廃止予定のコアに対して標準メッセージが表示されるようにするために使用します。最上位コアのハンドルと廃止予定のヘルパコアの名前を引数として指定します。

## 統合ハードウェア データ構造の追加のキーワード

統合ハードウェア データ構造のパラメータ、ポート、およびバス インターフェイスに対して、オプションで作成されているキーワード (サブプロパティ) があります。これらのキーワードはツール内部で使用され、また DRC 用の Tcl でも使用できます。これらのキーワードについて次に説明します。

- **MHS\_VALUE** : 統合オブジェクトが作成される際、MHS と MPD 両方からの情報が統合されます。デフォルト値は MPD で設定されていますが、MHS で設定した値が優先されます。条件によって、自動算出された値が MHS の値より優先される場合もあります。MHS で指定された値は、MHS\_VALUE サブプロパティに保存されます。
- **MPD\_VALUE** : 統合オブジェクトが作成される際、MHS と MPD 両方からの情報が統合されます。デフォルト値は MPD で設定されていますが、MHS で設定した値が優先されます。条件によって、自動算出された値が MHS の値より優先される場合もあります。MPD で指定された値は、MPD\_VALUE サブプロパティに保存されます。
- **CLK\_FREQ\_HZ** : 統合ハードウェア データ構造の各クロックの周波数は、そのポートの CLK\_FREQ\_HZ というサブプロパティに保存されます。これは内部サブプロパティであり、周波数値の単位は Hz です。
- **RESOLVED\_ISVALID** : MPD でパラメータ、ポート、またはバス インターフェイスに ISVALID というサブプロパティが定義されている場合、ツールにより論理式が真 (1) か偽 (0) かが評価され、この値がそのプロパティの RESOLVED\_ISVALID という内部サブプロパティに保存されます。
- **RESOLVED\_BUS** : IP のポートまたはパラメータにコロンで区切られたバスのリスト (BUS タグで指定) があり、MPD ファイルで関連付けることができる場合、その IP の接続が解析されて IP が接続されているバスが検出され、そのバス インターフェイスの名前が RESOLVED\_BUS タグに保存されます。

## ソフトウェア プラットフォーム生成の Tcl フロー

ドライバおよびライブラリは、データ定義ファイル (MDD または MLD) および対応するデータ生成ファイル (Tcl) を使用してコンフィギュレーションされます。Tcl ファイルでは、プロシージャが定義されています。これらの各プロシージャは、ソフトウェアおよびハードウェア両方のアクセスコマンドを使用します。Tcl プロシージャは、Libgen の自動ソフトウェア生成の一部として実行されます。次のセクションでは、ドライバまたはライブラリに対する Libgen と Tcl プロシージャの関係について説明します。Tcl プロシージャは、ハンドルを使用してシステム データ構造にアクセスします。詳細は、[236 ページの「ハンドル」](#)を参照してください。

### 入力ファイル

Libgen は、入力ファイル (MSS または MHS) と、IP、ドライバ、OS、プロセッサ、ライブラリのデータ ファイル (MPD、MDD、MLD、または Tcl) を使用し、これらのファイルに基づいてシステム ビューを作成します。MSS ファイルで定義された各ドライバ、OS、プロセッサ、ライブラリには、MDD または MLD ファイルと Tcl ファイルが関連付けられています。Tcl ファイルには、MSS ファイルの入力に基づいてドライバおよびライブラリの正しいコンフィギュレーションを生成するプロシージャが含まれています。ソフトウェア プラットフォームの生成中に使用される Tcl ファイルは、ドライバのディレクトリに MDD ファイルと共に配置されています。ザイリンクスが提供するコアのファイルは、<EDK インストール ディレクトリ>/sw/XilinxProcessorIPLib/drivers/<driver\_name>/data/ ディレクトリにあります。

### Libgen からの Tcl プロシージャ呼び出し

Libgen を実行すると、MSS ファイルに含まれる各ドライバ、OS、プロセッサ、およびライブラリに対して次の Tcl プロシージャがリストされている順に呼び出されます。

- **DRC** : DRC プロシージャの名前は、MDD または MLD ファイルの **OPTION** で指定されます。これが、ドライバ、OS、プロセッサ、ライブラリに対して Libgen で呼び出されるプロシージャです。たとえばドライバに対しては、MDD と Tcl に DRC プロシージャを定義する次の構文があります。

```
MDD/MLDOPTION DRC = mydrc
Tclprocedure mydrc {driver_handle} {
    ...
}
```

- **generate** : Tcl プロシージャ generate の実行中、関連のドライバ、OS、プロセッサ、ライブラリ ファイルがコピーされ対応する DRC プロシージャが実行された後、Libgen により MSS ファイルに含まれるすべてのドライバ、OS、プロセッサ、ライブラリが要求されます。各ドライバ、OS、プロセッサ、ライブラリの Tcl ファイルでこのプロシージャが定義されます。このプロシージャは、Libgen から対応するドライバ、OS、プロセッサ、またはライブラリ ハンドルを使用して呼び出されます。たとえばドライバの Tcl ファイルには、generate プロシージャを定義する次の構文があります。

```
procedure generate {driver_handle} {
    ...
}
```

- **post\_generate** : Tcl プロシージャ **post\_generate** の実行中、Tcl プロシージャ **generate** が呼び出された後、**Libgen** により **MSS** ファイルに含まれるすべてのドライバ、**OS**、プロセッサ、ライブラリが要求されます。各ドライバ、**OS**、プロセッサ、ライブラリの **Tcl** ファイルでこのプロシージャが定義されます。このプロシージャは、**Libgen** から対応するドライバ、**OS**、プロセッサ、またはライブラリ ハンドルを使用して呼び出されます。たとえばドライバの **Tcl** ファイルには、**post\_generate** プロシージャを定義する次の構文があります。

```
procedure post_generate {driver_handle} {
    ...
}
```

- **execs\_generate** : Tcl プロシージャ **post\_generate** が呼び出された後、**Libgen** により **MSS** ファイルに含まれるすべてのドライバ、**OS**、プロセッサ、ライブラリに対して呼び出される **Tcl** プロシージャです。各ドライバ、**OS**、プロセッサ、ライブラリの **Tcl** ファイルでこのプロシージャが定義されます。このプロシージャは、**Libgen** から対応するドライバ、プロセッサ、またはライブラリ ハンドルを使用して呼び出されます。たとえばドライバの **Tcl** ファイルには、**execs\_generate** プロシージャを定義する次の構文があります。

```
procedure execs_generate {driver_handle} {
    ...
}
```

ドライバ、**OS**、またはライブラリ ライタは、**Tcl** プロシージャ (**drc**、**generate**、**post\_generate**、または **execs\_generate**) に含まれる読み出し専用のソフトウェア アクセス コマンドおよびハードウェア アクセス コマンドを使用して、システム データ構造にアクセスできます。



## 用語集

---

### B

#### BBD ファイル

ブラック ボックス定義ファイル。ペリフェラルで使用するネットリストファイルがリストされます。

#### BFL

Bus Functional Language (バス ファンクション言語) の略。

#### BFM

Bus Functional Model (バス ファンクション モデル) の略。

#### BIT ファイル

ビットストリーム ファイル。

#### BitInit

Bitstream Initializer の略。FPGA 上のプロセッサの命令メモリを初期化し、ブロック RAM の命令メモリに格納するツールです。

#### BMM ファイル

ブロック RAM のマップ ファイル。各ブロック RAM がどのように連続した論理データ空間を構成するかが記述されています。Data2MEM は、このファイルを使用してデータを最適な初期化形式に変換します。BMM ファイルはテキスト ファイルなので、直接編集可能です。

#### BSB

Base System Builder の略。EDK デザインを作成するためのウィザードです。BSB ウィザードで使用するファイル タイプも BSB です。

#### BSP

[「スタンドアロン ライブラリ」](#) を参照してください。

## C

## CFI

Common Flash Interface (共通フラッシュ インターフェイス) の略。

## D

## DCM

Digital Clock Manager (デジタル クロック マネージャ) の略。

## DCR

デバイス コントロール レジスタ。

## DLMB

データ側のローカル メモリ バス。「[LMB](#)」も参照してください。

## DMA

Direct Memory Access の略。

## DOPB

データ側のオンチップ ペリフェラル バス。「[OPB](#)」も参照してください。

## DRC

デザイン ルール チェック。

## DSPLB

データ側のプロセッサ ローカル バス。「[ISPLB](#)」も参照してください。

## E

## EDIF ファイル

Electronic Data Interchange Format の略。業界標準のネットリスト ファイル形式です。

## EDK

エンベデッド開発キット。

## ELF ファイル

Executable Linked Format の略。

## EMC

Enclosure Management Controller の略。

## EST

エンベデッド システム ツール。

## F

### FATfs (XilFATfs)

LibXil FATFile システム。XilFATfs ファイル システムのアクセス ライブラリを使用すると、ザイリンクス System ACE のコンパクト フラッシュまたは IBM のマイクロドライブ デバイスに保存されたファイルに対して読み出し/書き込みを実行できます。

### Flat View

[IP Catalog] および [System Assembly View] の [Name] 列に表示される項目が展開可能なリストに区分されず、すべて直接表示されます。

### FPGA

フィールド プログラマブル ゲート アレイ。

### FSL

MicroBlaze の高速シンプレックス リンク。単一方向のポイント トゥ ポイント (PPP) データ ストリーミング インターフェイスで、ハードウェア間でのデータ送信を高速にします。MicroBlaze プロセッサには、FSL インターフェイスが直接接続されています。

## G

### GDB

GNU デバッガ。

### GPIO

汎用入出力。オンチップ ペリフェラル バスに接続される 32 ビットのペリフェラルです。

## H

### HDL

ハードウェア記述言語。

## Hierarchical View

[IP Catalog] および [System Assembly View] の両方でのデフォルト表示で、IP がインスタンスごとにグループ化されています。IP インスタンスは、分類条件 (上からプロセッサ、バス、バスブリッジ、ペリフェラル、汎用 IP) に基づいてリストされています。同じ種類の IP インスタンスは、アルファベット順に表示されます。IP 別に表示すると、IP インスタンスに関連するデータすべてを簡単に確認できます。この表示は、ハードウェアプラットフォームに IP インスタンスを追加する場合に特に便利です。

### IBA

Integrated Bus Analyzer の略。

### IDE

Integrated Design Environment の略。

### ILA

Integrated Logic Analyzer の略。

### ILMB

命令側のローカル メモリ バス。「LMB」も参照してください。

### IOPB

命令側のオンチップ ペリフェラル バス。「OPB」も参照してください。

### IPIC

IP インターコネクト。

### IPIF

IP インターフェイス。

### ISA

命令セット アーキテクチャ。プロセッサの命令セット、レジスタ、割り込み、例外、アドレスなどをプログラマに対してどう表示するかが記述されています。

### ISC

割り込みソース コントローラ。

### ISE ファイル

ザイリンクス ISE® Project Navigator プロジェクト ファイル。

## ISOCM

命令側のオンチップ メモリ。

## ISPLB

命令側のペリフェラル論理バス。「[DSPLB](#)」も参照してください。

## ISS

命令セット シミュレータ。

## J

## JTAG

Joint Test Action Group の略。

## L

## Libgen

XPS (Xilinx Platform Studio™) に含まれる Library Generator の略。

## LMB

ローカル メモリ バス。主にオンチップ ブロック RAM にアクセスするのに使用するレイテンシの低い同期バスです。MicroBlaze プロセッサには、ILMB バスと DLMB バスが含まれます。

## M

## MDD ファイル

Microprocessor Driver Definition の略。

## MDM

Microprocessor Debug Module の略。

## MFS ファイル

LibXil Memory File System の略。ファイル ハンドルの形式で、ユーザーがプログラム メモリを管理できます。

## MHS ファイル

Microprocessor Hardware Specification の略。バス、ペリフェラル、プロセッサ、接続、アドレス空間などのエンベデッド プロセッサ システムのコンフィギュレーションを定義するファイルです。

## MLD ファイル

Microprocessor Library Definition の略。

## MOST®

Media Oriented Systems Transport の略。オートモーティブ ネットワーク デバイスの開発規格です。

## MPD ファイル

Microprocessor Peripheral Definition の略。ペリフェラルで使用可能なポートおよびハードウェア パラメータがすべて含まれます。

## MSS ファイル

Microprocessor Software Specification の略。

## MVS ファイル

Microprocessor Verification Specification の略。

# N

## NCF ファイル

ネットリスト制約ファイル。

## NGC ファイル

論理デザイン データと制約の情報を含むネットリスト ファイル。EDIF および NCF ファイルからの情報がこのファイルに含まれています。

## NGD ファイル

Native Generic Database の略。デザイン全体を記述したネットリスト ファイル。

## NGO ファイル

ザイリンクス特定のフォーマットのバイナリ ファイルで、オリジナルのコンポーネントと階層によるデザインの論理記述が含まれます。

## NPI

Native Port Interface の略。ネイティブ ポート インターフェイス。

## NPL ファイル

ザイリンクス ISE Project Navigator の以前のプロジェクト ファイル。

## O

### OCM

オンチップ メモリ。

### OPB

オンチップ ペリフェラル バス。

## P

### PACE

Pinout and Area Constraints Editor の略。

### PAO ファイル

Peripheral Analyze Order ファイル。合成およびシミュレーションに必要な HDL ファイルの解析順を定義します。

### PBD ファイル

Processor Block Diagram ファイル。

### Platgen

XPS に含まれる Hardware Platform Generator の略。

### PLB

プロセッサ ローカル バス。

### PROM

プログラマブル ROM。

### PSF

Platform Specification Format の略。EDK ツールを駆動するデータ ファイルの形式です。

## S

### SDF ファイル

Standard Data Format ファイル。プログラム間のデータを転送するために固定長フィールドを使用するデータ形式です。

### SDK

ソフトウェア開発キット。

## SDMA

Soft Direct Memory Access の略。

## Simgen

XPS に含まれる Simulation Generator の略。

## SPI

Serial Peripheral Interface (シリアル ペリフェラル インターフェイス) の略。

## SVF ファイル

Serial Vector Format ファイル。

## U

## UART

Universal Asynchronous Receiver-Transmitter の略。

## UCF

ユーザー制約ファイル。

## V

## VHDL

VHSIC ハードウェア記述言語。

## X

## XBD ファイル

ザイリンクス ボード定義ファイル。

## XCL

Xilinx CacheLink の略。MicroBlaze プロセッサで使用可能な高パフォーマンスの外部メモリ キャッシュ インターフェイスです。

## Xilkernel

EDK に含まれるザイリンクス エンベデッド カーネル。ザイリンクス エンベデッド ソフトウェア プラットフォーム用のコンフィギュレーション可能なモジュール型の小型 RTOS です。

## XMD

Xilinx Microprocessor Debugger の略。



## XMP ファイル

Xilinx Microprocessor Project の略。EDK デザインの最上位のプロジェクトファイルです。

## XPS

Xilinx Platform Studio の略。エンベデッド デザインを開発する GUI 環境です。

## XST

Xilinx Synthesis Technology の略。

## Z

## ZBT

Zero Bus Turnaround™ の略。

## さ

## スタンドアロン ライブラリ

プロセッサに特有の関数にアクセスするソフトウェア モジュールのセット。

## ソフトウェア プラットフォーム

アプリケーションを構築するソフトウェア ドライバおよび OS をまとめたもの。ハードウェア プラットフォームの流動性およびザイリンクスおよびサードパーティ パートナーのサポートにより、各ハードウェア プラットフォームに対して複数のソフトウェア プラットフォームを作成できます。

## は

## ハードウェア プラットフォーム

ザイリンクスの FPGA 技術を使用すると、プロセッサ サブシステムのハードウェア ロジックをカスタマイズできます。標準的なマイクロプロセッサチップまたはコントローラ チップでは、このようなカスタマイズは不可能です。ハードウェア プラットフォームは、ザイリンクスの技術を使用して作成するエンベデッド プロセッサ サブシステムです。

## ブロック RAM

デバイスに組み込まれた LUT ベースのランダム アクセス メモリ (RAM) のブロック。分散 RAM とは区別されます。

