

この資料は英語版を翻訳したもので、内容に相違が生じる場合には原文を優先します。こちらの日本語版は参考用としてご利用ください。設計の際には、最新の英語版で内容をご確認ください。

NII51002-8.0.0

### はじめに

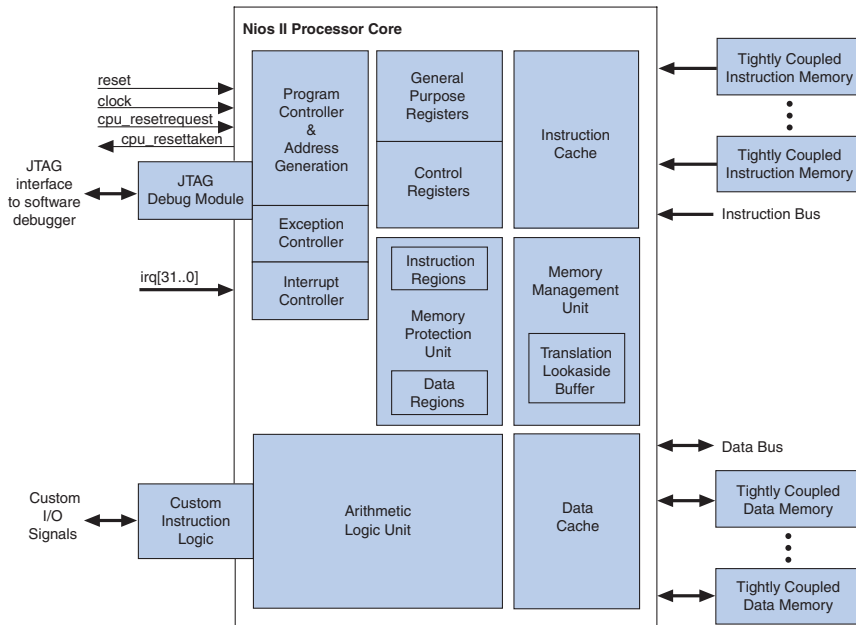
この章では、Nios® II アーキテクチャのすべての機能ユニットおよび Nios II プロセッサ・ハードウェア実装の基礎など、Nios II プロセッサのハードウェア構成について説明します。この章は、以下の項で構成されています。

- 2-3 ページの「プロセッサの実装」
- 2-4 ページの「レジスタ・ファイル」
- 2-4 ページの「演算ロジック・ユニット」
- 2-6 ページの「リセット信号」
- 2-7 ページの「例外および割り込みコントローラ」
- 2-9 ページの「メモリおよび I/O の構成」
- 2-18 ページの「JTAG デバッグ・モジュール」

Nios II アーキテクチャは、命令セット・アーキテクチャ (ISA) を記述します。そのため、ISA は命令を実装する機能ユニットのセットを必要とします。Nios II プロセッサ・コアとは、Nios II 命令セットを実装し、本書で説明する機能ユニットをサポートするハードウェア・デザインです。このプロセッサ・コアには、ペリフェラルや外部との接続ロジックは含まれていません。Nios II アーキテクチャの実装に必要な回路のみ搭載されています。

図 2-1 に、Nios II プロセッサ・コアのブロック図を示します。

図 2-1. Nios II プロセッサ・コアのブロック図



Nios II アーキテクチャは、ユーザーがアクセス可能な以下の機能ユニットを定義しています。

- レジスタ・ファイル
- 演算ロジック・ユニット (ALU)
- カスタム命令ロジックへのインタフェース
- 例外コントローラ
- 割り込みコントローラ
- 命令バス
- データ・バス
- メモリ・マネージメント・ユニット (MMU)
- メモリ・プロテクション・ユニット (MPU)
- 命令キャッシュおよびデータ・キャッシュ・メモリ
- 命令およびデータ用密結合メモリ・インタフェース
- JTAG デバッグ・モジュール

以下の項では、各機能ユニットに関するハードウェア実装の詳細について説明します。

## プロセッサの実装

Nios II アーキテクチャの機能ユニットは、Nios II 命令セットの基礎を形成します。ただし、いずれかのユニットがハードウェアに実装されているという意味ではありません。Nios II アーキテクチャは、特定のハードウェア実装ではなく、命令セットを記述します。機能ユニットは、ハードウェアに実装するか、ソフトウェアでエミュレートするか、あるいは完全に省略することができます。

Nios II 実装とは、特定の Nios II プロセッサ・コアで実現されるデザインの実装セットです。すべての実装が、「[Nios II プロセッサ・リファレンス・ハンドブック](#)」で定義される命令セットをサポートします。各実装は、コア・サイズの小型化や高性能化など、特定の目的を達成します。これにより、Nios II アーキテクチャは、多様なターゲット・アプリケーションの要求に適合します。

実装の変数は、一般に3つのトレードオフ・パターン（機能の増強または削減、機能の追加または除外、機能のハードウェア実装またはソフトウェア・エミュレーション）のいずれかに当てはまります。各トレードオフの例を以下に示します。

- 機能の増強または削減 — 例えば、性能を微調整するために、命令キャッシュ・メモリ容量を増減できます。キャッシュの容量を増やすと大規模なプログラムの実行速度が向上し、キャッシュの容量を減らすとオンチップ・メモリのリソースを節約できます。
- 機能の追加または除外 — 例えば、コストを削減するために、JTAG デバッグ・モジュールを省略するよう選択できます。これによって、オンチップ・ロジックとメモリ・リソースを節約できますが、ソフトウェア・デバッグを使用してアプリケーションをデバッグする機能もなくなります。
- ハードウェア実装またはソフトウェア・エミュレーション — 例えば、複雑な演算を殆ど実行しない制御アプリケーションでは、除算命令をソフトウェアでエミュレートするよう選択できます。除算用のハードウェアをなくすとオンチップ・リソースを節約できますが、除算演算の実行時間が増大します。



Nios II コアがサポートする機能について詳しくは、「[Nios II コアの実装の詳細](#)」の章（Nios II プロセッサ・リファレンス・ハンドブック）を参照してください。ユーザー選択可能な Nios II プロセッサのパラメータについて詳しくは、「[SOPC Builder での Nios II プロセッサの実装](#)」の章（Nios II プロセッサ・リファレンス・ハンドブック）を参照してください。

## レジスタ・ファイル

Nios II アーキテクチャは、32 個の 32 ビット汎用整数レジスタ、および最大 32 個の 32 ビットのコントロール・レジスタから成るフラットなレジスタ・ファイルをサポートしています。このアーキテクチャでは、スーパーバイザ・モードとユーザー・モードがサポートされており、システム・コードによって不正なアプリケーションからコントロール・レジスタを保護することができます。

Nios II アーキテクチャでは、後で浮動小数点レジスタを追加できます。

## 演算ロジック・ユニット

Nios II ALU は、汎用レジスタに格納されたデータに対して演算を実行します。ALU 演算では、1 つまたは 2 つの入力をレジスタから受け取り、演算結果をレジスタに格納します。ALU は、表 2-1 に示すデータ演算をサポートしています。

カテゴリ	説明
演算	ALU は、符号付きおよび符号なしオペランドに対する加算、減算、乗算、および除算をサポートしています。
関係	ALU は、符号付きおよび符号なしオペランドに対する「等しい」、「等しくない」、「大なりまたは等しい」、「小なり」の関係演算 (==、!=、>=、<) をサポートしています。
論理	ALU は、AND、OR、NOR、および XOR 論理演算をサポートしています。
シフトおよび回転	ALU は、シフト演算と回転演算をサポートしており、命令ごとにデータを 0 ビットから 31 ビット位置だけシフト/回転できます。ALU は、算術右シフトおよび論理右/左シフトをサポートしています。ALU は、左/右回転をサポートしています。

その他の演算を実行する場合は、表 2-1 に示す基本演算の組み合わせをソフトウェアで実行して結果を算出します。

### 未実装命令

一部の Nios II プロセッサ・コアには、完全な Nios II 命令セットをサポートするハードウェアが搭載されていません。このようなコアでは、ハードウェア・サポートのない命令は未実装命令と認識されます。

プロセッサは、未実装命令を発行すると必ず例外を生成し、例外ハンドラがその演算をソフトウェアでエミュレートするルーチンを呼び出します。したがって、プログラマは未実装命令によるプロセッサへの影響を意識する必要はありません。



潜在的な未実装命令のリストは、「プログラミング・モデル」の章 (Nios II プロセッサ・リファレンス・ハンドブック) を参照してください。

## カスタム命令

Nios II アーキテクチャは、ユーザー定義のカスタム命令をサポートしています。Nios II ALU は、カスタム命令ロジックに直接接続されるため、設計者はネイティブ命令と全く同じようにアクセスし、使用できるハードウェア演算にカスタム命令を実装できます。



詳細は、「[Nios II Custom Instruction User Guide](#)」を参照してください。

## 浮動小数点命令

Nios II アーキテクチャは、IEEE Std 754-1985 で規定される単精度浮動小数点命令をサポートします。これらの浮動小数点命令は、カスタム命令として実装されます。表 2-2 に、IEEE 754-1985 への準拠の詳細な説明を記載します。

表 2-2. IEEE 754-1985 浮動小数点に準拠するハードウェア (1 / 2)		
	特長	実装
演算 <sup>(1)</sup>	加算	実装
	減算	実装
	乗算	実装
	除算	オプション
精度	単精度	実装
	倍精度	未実装。倍精度演算はソフトウェアで実装されます。
例外条件	無効演算	結果は非数 (NaN)
	ゼロでの除算	結果は $\pm \infty$
	オーバフロー	結果は $\pm \infty$
	不正確	結果は正常数
	アンダーフロー	結果は $\pm 0$
丸めモード	Round-to-Nearest	実装
	Round-toward-Zero	未実装
	Round-toward- $+\infty$	未実装
	Round-toward- $-\infty$	未実装
NaN	Quiet	実装
	Signaling	未実装
非正規化(デノーマル)数		非正規化オペランドはゼロとして扱われます。浮動小数点カスタム命令は非正規化数を生成しません。

表 2-2. IEEE 754-1985 浮動小数点に準拠するハードウェア (2 / 2)		
特長		実装
ソフトウェア例外		未実装。この表の随所で示すとおり、IEEE 754-1985 例外条件が検出され処理されます。
ステータス・フラグ		未実装。この表の随所で示すとおり、IEEE 754-1985 例外条件が検出され処理されます。

表 2-2 の注：

- (1) Nios II 統合開発環境 (IDE) は、加算、減算、乗算、および除算以外のプリミティブな浮動小数点演算のソフトウェア実装を生成します。これには浮動小数点の変換および比較などの演算が含まれます。これらのプリミティブな演算のソフトウェア実装は IEEE 754-1985 に 100% 準拠しています。



浮動小数点カスタム命令は、どの Nios II プロセッサ・コアにも追加できます。Nios II ソフトウェア開発ツールは、プロセッサ・コアに浮動小数点命令が存在するときに利用できる C コードを認識します。

## リセット信号

Nios II プロセッサ・コアは、2 個のリセット信号をサポートしています。

- `reset` - これはプロセッサ・コアを直ちに強制的にリセットするグローバル・ハードウェア・リセット信号です。
- `cpu_resetrequest` - これは、Nios II システムの他のコンポーネントに影響を与えないでプロセッサをリセットするローカル・リセット信号です。プロセッサはパイプライン内のすべての命令の実行を終了して、リセット状態に入ります。このプロセスには、数クロック・サイクルかかる場合があります。プロセッサ・コアはリセットが完了すると 1 サイクルの間 `cpu_resettaken` 信号をアサートし、`cpu_resetrequest` がアサートされたままの場合はその信号を定期的にアサートします。`cpu_resetrequest` がアサートされている間、プロセッサはリセット状態を維持します。

プロセッサはリセット状態の間、リセット・アドレスから定期的に読み出します。プロセッサは読み出しの結果を廃棄してリセット状態を維持します。

プロセッサは、JTAG デバッグ・モジュールの制御下にあるとき、つまりプロセッサが休止状態のときは、`cpu_resetrequest` に応答しません。JTAG デバッグ・モジュールが制御権を放棄するときに信号がアサートされている場合、プロセッサは各信号ステップ中および実行の再開時に、瞬時的に `cpu_resetrequest` 信号に応答します。

## 例外および 割り込み コントローラ

### 例外コントローラ

Nios II アーキテクチャでは、ベクタ化されていないシンプルな例外コントローラにより、すべての例外タイプを処理できます。ハードウェア割り込みを含め、どの例外が発生してもプロセッサは1つの例外アドレスに実行を移します。このアドレスの例外ハンドラは、例外の原因を特定して、適切な例外ルーチンをディスパッチします。

例外アドレスは、システム生成時に **SOPC Builder** で指定されます。

すべての例外は正確です。正確ということは、プロセッサがフォールトを起こした命令より前に実行した命令を完了しており、フォールトを起こした命令の次にある命令の実行を開始していないことを意味します。正確な例外により、例外ハンドラが例外をクリアすると、プロセッサはプログラムの実行を再開できます。

### 統合割り込みコントローラ

Nios II アーキテクチャは、32 の外部ハードウェア割り込みをサポートしています。プロセッサ・コアは、`irq0` から `irq31` までの32レベルの割り込み要求 (IQR) 入力を備えており、各割り込みソースに対して固有の入力を提供します。IQR の優先順位は、ソフトウェアで決定されます。アーキテクチャは、ネスト型割り込みをサポートしています。

このソフトウェアでは、各 IRQ 入力用の割り込みイネーブル・ビットを備えた `ienable` コントロール・レジスタを介して、割り込みソースを個別にイネーブルおよびディセーブルできます。また、`status` コントロール・レジスタの `PIE` ビットを介して、割り込みをグローバルにイネーブルおよびディセーブルできます。ハードウェア割り込みは、以下の3つの条件がすべて満たされた場合にのみ生成されます。

- `status` レジスタの `PIE` ビットが1
- 割り込み要求入力 `irq<n>` がアサートされている
- `ienable` レジスタの対応するビット `n` が1

### 割り込みベクタ・カスタム命令

Nios II プロセッサ・コアは、割り込みベクタのディスパッチを高速化する割り込みベクタ・カスタム命令を提供します。このカスタム命令を含めることで、プログラムの割り込みレイテンシを短縮します。

割り込みベクタ・カスタム命令は、各割り込みの1つの入力が Nios II プロセッサに接続されたプライオリティ・エンコーダに基づいています。割り込みベクタ・カスタム命令のコストは、Nios II プロセッサに接続さ

れた割り込み数により異なります。最悪ケースは32の割り込みを持つシステムです。このケースでは、割り込みベクタ・カスタム命令は約 50 のロジック・エレメント (LE) を消費します。

多数の割り込みが接続されている場合、割り込みベクタ・カスタム命令をシステムに追加すると、 $f_{MAX}$  が低下することがあります。



Nios II プロセッサへの割り込みベクタ・カスタム命令の追加のガイドについては、「SOPC Builder での Nios II プロセッサのインスタンス化」の章 (Nios II プロセッサ・リファレンス・ハンドブック) を参照してください。

表 2-3 に、割り込みベクタ・カスタム命令の実装の詳細を示します。

表 2-3. 割り込みベクタ・カスタム命令

### ALT\_CI\_EXCEPTION\_VECTOR\_N

動作: `if (ipending == 0) | (estatus.PIE == 0)`  
`then rC ← 負の値`  
`else rC ← 8 × ipending レジスタ (ct14) の最下位 1 ビットのビット番号`

アセンブラの構文: `custom ALT_CI_EXCEPTION_VECTOR_N, rC, r0, r0`

例: `custom ALT_CI_EXCEPTION_VECTOR_N, et, r0, r0`  
`blt et, r0, not_irq`

説明: 割り込みベクタ・カスタム命令は、割り込みベクタ・ディスパッチを高速化します。このカスタム命令は最優先の割り込みを識別し、ベクタ・テーブル・オフセットを生成して、このオフセットを rC に格納します。ハードウェア割り込みが存在しない場合 (すなわち、例外はトラップなどのソフトウェア条件によって発生します)、この命令は負のオフセットを生成します。

使用方法: 割り込みベクタ・カスタム命令は、例外ハンドラによって排他的に使用されます。

命令タイプ: R

命令フィールド: C = オペランド rC のレジスタ・インデックス  
 N = ALT\_CI\_EXCEPTION\_VECTOR\_N の値

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				0				C				0	0	1	N							0x32									



命令リファレンス・フォーマットの説明については、「Nios II プロセッサ・リファレンス・ハンドブック」の「命令セット・リファレンス」の章を参照してください。

## メモリおよび I/O の構成

この項では、Nios II のメモリおよび I/O 構成のハードウェア実装の詳細を説明します。ここでは、すべての Nios II プロセッサ・システムに該当する一般的な概念、およびシステムごとに異なる機能について解説します。

Nios II のメモリおよび I/O 構成の柔軟性は、Nios II プロセッサ・システムと従来型のマイクロコントローラの違いを最も顕著に示すものです。Nios II プロセッサ・システムはコンフィギュレーション可能なため、メモリとペリフェラルはシステムごとに異なります。その結果、メモリと I/O 構成もシステムごとに異なります。

Nios II コアは、以下の 1 つまたは複数を使用してメモリおよび I/O アクセスを提供します。

- 命令マスタ・ポート — システム・インタコネクタ・ファブリックを介してインストラクション・メモリに接続される Avalon-MM マスタ・ポート
- 命令キャッシュ—Nios II コアに内蔵されている高速キャッシュ・メモリ
- データ・マスタ・ポート — システム・インタコネクタ・ファブリックを介してデータ・メモリとペリフェラルに接続される Avalon-MM マスタ・ポート
- データ・キャッシュ—Nios II コアに内蔵されている高速キャッシュ・メモリ
- 密結合インストラクションまたはデータ・メモリ・ポート —Nios II コアの外部にある高速オンチップ・メモリへのインタフェース

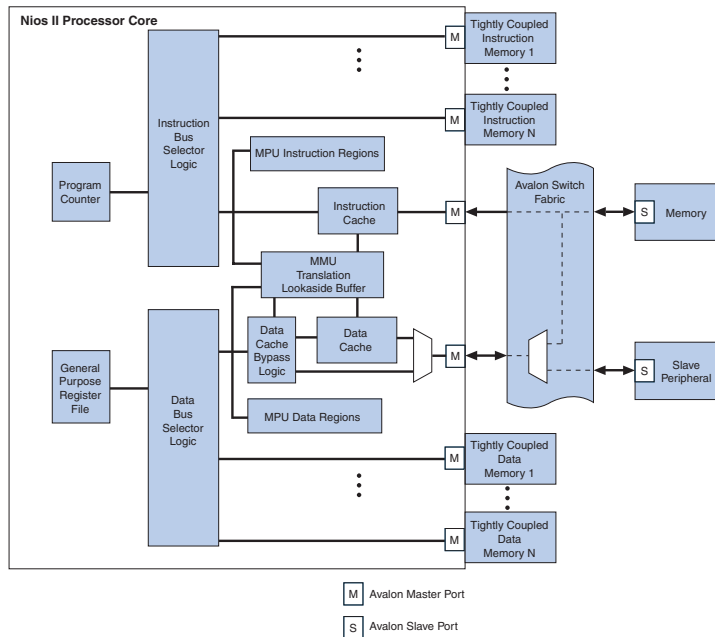
Nios II アーキテクチャでは、プログラマからはハードウェアの詳細が認識されないため、プログラマはハードウェア実装を意識することなく、Nios II アプリケーションを開発できます。



プログラミングへの影響について詳しくは、「Nios II プロセッサ・リファレンス・ハンドブック」の「[プログラミング・モデル](#)」の章を参照してください。

図 2-2 に、Nios II プロセッサ・コアのメモリおよび I/O 構成の図を示します。

図 2-2. Nios II のメモリおよび I/O 構成



## 命令バスおよびデータ・バス

Nios II アーキテクチャは、独立した命令バスとデータ・バスをサポートしているため、Harvard アーキテクチャに分類されます。命令バスとデータ・バスはともに、Avalon-MM インタフェース仕様に準拠する Avalon-MM マスタ・ポートとして実装されます。データ・マスタ・ポートは、メモリ・コンポーネントとペリフェラル・コンポーネントの両方に接続されますが、命令マスタ・ポートはメモリ・コンポーネントにのみ接続されます。



Avalon-MM インタフェースの詳細は、「[Avalon Memory Mapped Interface Specification](#)」を参照してください。

## メモリ・アクセスおよびペリフェラル・アクセス

Nios II アーキテクチャでは、メモリ・マップト I/O アクセスが利用できます。データ・メモリとペリフェラルはともに、データ・マスタ・ポートのアドレス空間にマップされます。Nios II アーキテクチャは、リトル・エンディアンです。ワードおよびハーフワードは、上位アドレスを上位バイトとしてメモリに格納されます。

Nios II アーキテクチャは、メモリとペリフェラルの使用に関して何も規定していないため、メモリとペリフェラルの数量、タイプ、および接続はシステムに依存します。通常、Nios II プロセッサ・システムは、高速オンチップ・メモリと低速オフチップ・メモリを組み合わせで搭載しています。ペリフェラルは一般にオンチップにも搭載されていますが、オフチップ・ペリフェラルへのインタフェースも存在します。

## 命令マスタ・ポート

Nios II 命令バスは、32 ビット Avalon-MM マスタ・ポートとして実装されます。命令マスタ・ポートは、1つの機能、つまりプロセッサが実行するフェッチ命令を実行します。命令マスタ・ポートは、書き込み操作は実行しません。

命令マスタ・ポートはパイプライン化された Avalon-MM マスタ・ポートです。パイプライン化された Avalon-MM 転送のサポートにより、パイプライン・レイテンシを伴う同期メモリの影響が最小になり、システムの全体的な  $f_{MAX}$  が上昇します。命令マスタ・ポートは、前の要求からデータが返される前に連続リード要求を発行することができます。Nios II プロセッサは、シーケンシャル命令をプリフェッチして分岐予測を実行し、命令パイプラインを可能な限りアクティブに保持します。

命令マスタ・ポートは、常に 32 ビットのデータを取得します。命令マスタ・ポートは、システム・インタコネクト・ファブリックに含まれるダイナミック・バス・サイジング・ロジックに依存します。ダイナミック・バス・サイジングによって、ターゲット・メモリの幅に関係なく、すべての命令フェッチがフル命令ワードを返します。結果的に、プログラムは Nios II プロセッサ・システムにおけるメモリ幅を意識する必要はありません。

Nios II アーキテクチャは、低速メモリにアクセスするときの平均命令フェッチ性能を改善するために、オンチップ・キャッシュ・メモリをサポートしています。詳細は、[2-13 ページの「キャッシュ・メモリ」](#)を参照してください。Nios II アーキテクチャは、オンチップ・メモリへの確実な低レイテンシ・アクセスを提供する密結合メモリをサポートします。詳細は、[2-15 ページの「密結合メモリ」](#)を参照してください。

## データ・マスタ・ポート

Nios II データ・バスは、32 ビット Avalon-MM マスタ・ポートとして実装されます。データ・マスタ・ポートは以下の 2 つの機能を実行します。

- プロセッサがロード命令を実行するときに、メモリまたはペリフェラルからデータを読み込む
- プロセッサがストア命令を実行するときに、メモリまたはペリフェラルからデータを書き込む

マスタ・ポート上のバイト・イネーブル信号は、ストア操作中に書き込む 4 つのバイト・レーンのいずれかを指定します。Nios II コアが 4 バイトを超えるデータ・キャッシュ・ライン・サイズでコンフィギュレーションされているとき、データ・マスタ・ポートはパイプライン化された Avalon-MM 転送をサポートします。データ・キャッシュ・ライン・サイズが 4 バイトしかないとき、データ・マスタ・ポートはどのメモリ・パイプライン・レイテンシも待ち状態として認識します。データ・マスタ・ポートがゼロ・ウェイト・ステート・メモリに接続されているときは、ロードおよびストア操作は 1 クロック・サイクルで完了できます。

Nios II アーキテクチャは、低速メモリにアクセスするときの平均データ転送性能を改善するために、オンチップ・キャッシュ・メモリをサポートしています。詳細は、“[キャッシュ・メモリ](#)” を参照してください。Nios II アーキテクチャは、オンチップ・メモリへの確実な低レイテンシ・アクセスを提供する密結合メモリをサポートします。詳細は、[2-15 ページの「密結合メモリ」](#)を参照してください。

## 命令およびデータ用の共有メモリ

通常、命令マスタ・ポートとデータ・マスタ・ポートは、命令とデータの両方を格納する 1 つのメモリを共有します。プロセッサ・コアは独立した命令バスとデータ・バスを搭載していますが、Nios II プロセッサ・システム全体としては、外部に対して 1 つの共有命令 / データ・バスを提供することができます。Nios II プロセッサ・システムの外形は、システム内のメモリとペリフェラル、および Avalon スイッチ・ファブリックの構造によって異なります。

データ・マスタ・ポートおよび命令マスタ・ポートによって、一方のポートが他方のポートを枯渇させるグリッドロック状態が発生することはありません。最高の性能を実現するには、命令マスタ・ポートとデータ・マスタ・ポートが共有するメモリ上で、データ・マスタ・ポートに高いアビリティ・プライオリティを割り当てる必要があります。

## キャッシュ・メモリ

Nios II アーキテクチャは、命令マスタ・ポート（命令キャッシュ）とデータ・マスタ・ポート（データ・キャッシュ）の両方でキャッシュ・メモリをサポートしています。キャッシュ・メモリは、Nios II プロセッサ・コアの主要部分として、オンチップに搭載されています。キャッシュ・メモリを利用すると、プログラムやデータ格納用の SDRAM などの低速オフチップ・メモリを使用する Nios II プロセッサ・システムの平均メモリ・アクセス時間を改善できます。

命令キャッシュとデータ・キャッシュは、実行時には常にイネーブルされますが、ペリフェラル・アクセス時にキャッシュされたデータが返されないように、ソフトウェアによってデータ・キャッシュをバイパスする手段が用意されています。キャッシュ管理とキャッシュ・コヒーレンスは、ソフトウェアで処理されます。Nios II 命令セットには、キャッシュ管理用の命令があります。

### コンフィギュレーション可能なキャッシュ・メモリ・オプション

キャッシュ・メモリはオプションです。高いメモリ性能（および、それに伴うキャッシュ・メモリ）の必要性は、アプリケーションによって決まります。多くのアプリケーションでは、可能な限り小さなプロセッサ・コアを必要とし、性能を犠牲にしてサイズを優先させる場合があります。

Nios II プロセッサ・コアには、キャッシュ・メモリの一方または両方を搭載するか、あるいはどちらも搭載しないことが可能です。さらに、データ・キャッシュと命令キャッシュの両方またはいずれか一方を備えたコアの場合、キャッシュ・メモリのサイズはユーザーが設定 / 構成可能です。キャッシュ・メモリを搭載してもプログラムの機能には影響はありませんが、プロセッサが命令をフェッチする速度とデータを読み出し / 書き込みする速度は影響を受けます。

### キャッシュ・メモリの効果的な使用法

性能を改善するためのキャッシュ・メモリの効果は、以下の前提に基づきます。

- 通常のメモリがオフチップで配置され、オンチップ・メモリと比較してアクセス時間が長い
- 性能が重視される最大命令ループが命令キャッシュよりも小さい
- 性能が重視されるデータの最大ブロックがデータ・キャッシュよりも小さい

対象となるアプリケーションにおける効果は設計者が判断できますが、最適なキャッシュ構成はアプリケーションによって異なります。例えば、Nios II プロセッサ・システムに、高速オンチップ・メモリしかない（つまり、低速オフチップ・メモリにはアクセスしない）場合、命令キャッシュまたはデータ・キャッシュによる性能向上は期待できません。別の例では、プログラムの重要なループが 2 K バイトでも、命令キャッシュのサイズが 1 K バイトであれば、命令キャッシュによって実行速度が向上することはありません。実際には、性能は低下すると考えられます。

性能上の理由から、特定のデータまたはコード・セクションがキャッシュ・メモリに存在することをアプリケーションが要求する場合、密結合メモリ機能によってより適切なソリューションが提供されることがあります。詳細は、[2-15 ページ](#)の「[密結合メモリ](#)」を参照してください。

### キャッシュ・バイパス方法

Nios II アーキテクチャはデータ・キャッシュのバイパスに、以下の方法を提供します。

- I/O ロードおよびストア命令
- ビット 31 キャッシュ・バイパス

#### I/O ロードおよびストア命令手法

ldio や stio などのロードおよびストア I/O 命令は、データ・キャッシュをバイパスして、Avalon-MM データを指定アドレスに強制的に転送します。

#### ビット 31 キャッシュ・バイパス手法

データ・マスタ・ポートのビット 31 キャッシュ・バイパス手法では、プロセッサがキャッシュとの間でデータ転送を行うかキャッシュをバイパスするかを示すタグとしてアドレスのビット 31 を使用します。これは特定のアドレスをキャッシュし、その他をバイパスする必要があるソフトウェアにとって便利です。ソフトウェアは、アドレス指定されたデータをキャッシュするかどうかに関する追加情報を指定しなくても、ファンクション間でアドレスをパラメータとして渡すことができます。



コアが実装するキャッシュ・バイパス手法を決定するには、「[Nios II コアの実装の詳細](#)」の章（Nios II プロセッサ・リファレンス・ハンドブック）を参照してください。

## 密結合メモリ

密結合メモリは、性能重視のアプリケーションに保証された低レイテンシ・メモリ・アクセスを提供します。キャッシュ・メモリと比較して、密結合メモリには以下の利点があります。

- キャッシュ・メモリに類似した性能
- ソフトウェアは、性能重視のコードまたはデータが密結合メモリに存在することを保証できます。
- ローディング、無効化、またはメモリのフラッシュなどのリアルタイムでのキャッシュによるオーバーヘッドはありません。

物理的には、密結合メモリ・ポートは Nios II プロセッサ・コア上の独立したマスタ・ポートで、命令またはデータ・マスタ・ポートに類似しています。Nios II コアは、ゼロ、1 つ、または複数の密結合メモリを持つことができます。Nios II アーキテクチャは、命令アクセスとデータ・アクセスの両方に対して密結合メモリをサポートします。各密結合メモリ・ポートは、保証された固定低レイテンシにより 1 つのメモリにのみ直接接続されます。メモリは Nios II コアの外部にあり、通常はチップ上に存在します。

## 密結合メモリへのアクセス

システム・インタコネクト・ファブリックを介して接続された他のメモリ・デバイスと同様に、密結合メモリは通常のアドレス空間を占有します。密結合メモリ（存在する場合）のアドレス範囲は、システム生成時に決定されます。

ソフトウェアは通常のロードおよびストア命令を使用して、密結合メモリにアクセスします。ソフトウェアの観点からは、密結合メモリへのアクセスと他のメモリへのアクセスの違いはありません。

## 密結合メモリの効果的な使用

システムは密結合メモリを使用して、コードまたはデータの特定セクションへのアクセスに対して最大性能を達成できます。例えば、割り込みを多用するアプリケーションは、例外ハンドラ・コードを密結合メモリに配置して、割り込みレイテンシを最小化することができます。同様に、計算量の多いデジタル信号処理（DSP）アプリケーションは、データ・バッファを密結合メモリ内に配置して、可能な限り高速なデータ・アクセスを達成できます。

アプリケーションのメモリ要件がチップ内に完全に収まるほど小さい場合は、密結合メモリをコードおよびデータ専用で使用できます。より大規模なアプリケーションでは、密結合メモリ内に何を配置するかを選択して、コストと性能のトレードオフを最大化する必要があります。

## アドレス・マップ

Nios II プロセッサ・システムにおけるメモリおよびペリフェラルのためのアドレス・マップは、デザインごとに異なります。システム生成時にアドレス・マップを指定します。

プロセッサの一部で、特に説明すべきアドレスは次の3つです。

- リセット・アドレス
- 例外アドレス
- ブレーク・ハンドラ・アドレス

プログラムはマクロとドライバを使用して、メモリおよびペリフェラルにアクセスします。したがって、アプリケーション開発者がアドレス・マップの柔軟性を考慮する必要はありません。

## メモリ・マネージメント・ユニット

オプションの Nios II MMU は次の機能を備えています。

- 仮想-物理アドレス・マッピング
- メモリ保護

Nios II MMU は以下の2つの機能を備えています。

- 32 ビットの仮想および物理アドレス。4 G バイトの仮想アドレス空間を 4 G バイトの物理メモリにマッピングします。
- 4 K バイトのページおよびフレーム・サイズ
- 直接アクセスに使用可能なわずか 512 M バイトの物理アドレス空間
- ハードウェア変換索引バッファ (TLB) によるアドレス変換の高速化
  - 命令アクセスおよびデータ・アクセス用の独立した TLB
  - リード、ライト、および実行の許可をページ単位で制御
  - デフォルトのキャッシュ動作をページ単位で制御
  - ソフトウェア・ページ・テーブル用の  $n$  方向セット連想キャッシュとして動作する TLB
  - システム生成時に構成可能な TLB サイズおよび連想性
- システム・ソフトウェアで決定されるページ・テーブル (または同等なデータ構造) のフォーマット
- システム・ソフトウェアで決定される TLB エントリの置換方式
- システム・ソフトウェアで決定される TLB エントリの書き込みポリシー
- すべての例外は正確



MMU 実装について詳しくは、「Nios II プロセッサ・リファレンス・ハンドブック」の「[プログラミング・モデル](#)」の章を参照してください。

Nios II ハードウェア・システムで Nios II プロセッサをインスタンス化するときには、オプションで MMU を含めることができます。MMU は存在するときは常にイネーブルされ、データ・キャッシュおよび命令キャッシュは VIPT (Virtually-Indexed, Physically-Tagged) 方式のキャッシュです。いくつかのパラメータを使用でき、システム・ニーズに合わせて MMU を最適化することができます。

ユーザー選択可能な Nios II MMU のパラメータについて詳しくは、**SOPC Builder** での **Nios II プロセッサの実装** の章 (Nios II プロセッサ・リファレンス・ハンドブック) を参照してください。



Nios II MMU はオプションで、Nios II MPU と相互に排他的です。Nios II システムは、MMU または MPU のいずれかを含むことができますが、同一デザインで MMU と MPU の両方を含むことはできません。

## メモリ保護ユニット

オプションの Nios II MPU は、次の機能を備えています。

### ■ メモリ保護

Nios II MPU は次の機能を備えています。

- 最大 32 の命令領域と 32 のデータ領域
- 可変命令およびデータ領域サイズ
- サイズまたは上位アドレス制限により定義される領域メモリ量
- データ領域に対するリードおよびライト・アクセス許可
- 命令領域に対するアクセス許可の実行
- オーバラップ領域
- すべての例外は正確



MPU 実装について詳しくは、「Nios II プロセッサ・リファレンス・ハンドブック」の「**プログラミング・モデル**」の章を参照してください。

Nios II ハードウェア・システムで、Nios II プロセッサをインスタンス化するときには、オプションで MPU を含めることができます。MPU は存在するときには常にイネーブルされます。いくつかのパラメータを使用でき、システム・ニーズに合わせて MPU を最適化することができます。



ユーザー選択可能な Nios II MPU のパラメータについて詳しくは、**SOPC Builder** での **Nios II プロセッサの実装** の章 (Nios II プロセッサ・リファレンス・ハンドブック) を参照してください。



Nios II MPU はオプションで、Nios II MMU と相互排他的です。Nios II システムは、MPU または MMU のいずれかを含むことができますが、同一デザインで MPU と MMU の両方を含むことはできません。

## JTAG デバッグ・ モジュール

Nios II アーキテクチャでは、オン・チップ・エミュレーション機能を可能にする JTAG デバッグ・モジュールがサポートされているため、プロセッサをホスト PC からリモートで制御できます。PC ベースのソフトウェア・デバッグ・ツールは、JTAG デバッグ・モジュールと通信し、以下のような機能を実現します。

- メモリへのプログラムのダウンロード
- 実行の開始および停止
- ブレークポイントおよびウォッチポイントの設定
- レジスタおよびメモリの解析
- リアルタイム実行トレース・データの収集



Nios II MMU は、JTAG デバッグ・モジュール・トレースをサポートしていません。

デバッグ・モジュールは、アルテラの FPGA の JTAG 回路に接続されます。これにより、外部デバッグ・プローブから FPGA 上の標準 JTAG インタフェースを介してプロセッサにアクセスできます。プロセッサ側では、デバッグ・モジュールがプロセッサ・コア内部の信号に接続されます。デバッグ・モジュールには、プロセッサに対してマスク不能な制御を行い、テスト中のアプリケーションにリンクされたソフトウェア・スタブは不要です。スーパーバイザ・モードでプロセッサがアクセス可能なすべてのシステム・リソースをデバッグ・モジュールで利用できます。トレース・データ収集の場合、デバッグ・モジュールはオンチップまたはデバッグ・プローブ内のメモリにトレース・データを格納します。

デバッグ・モジュールは、ハードウェア・ブレーク信号をアサートするか、実行するプログラム・メモリ内にブレーク命令を書き込むことによって、プロセッサの制御を取得します。どちらの場合も、プロセッサはブレーク・アドレスに配置されるルーチンに制御を移します。ブレーク・アドレスは、システム生成時に指定されます。

Nios II プロセッサなどのソフトコア・プロセッサは、従来型の固定プロセッサにはない独自のデバッグ機能を提供します。Nios II プロセッサのソフトコア特性により、フル機能を備えたデバッグ・コアを使用して開発中のシステムをデバッグし、後でデバッグ機能を削除してロジック・リソースを節約することができます。製品のリリース・バージョンでは、JTAG デバッグ・モジュール機能は縮小するか、完全に取り除くことができます。

以下の項では、Nios II JTAG デバッグ・モジュール・ハードウェアの機能について説明します。すべてのハードウェア機能の使用は、Nios II IDE のように、ターゲット・プロセッサへの接続を管理し、デバッグ・プロセスを制御するホスト・ソフトウェアに依存します。

## JTAG ターゲット接続

JTAG ターゲット接続とは、アルテラ FPGA 上の標準 JTAG ピンを介して CPU に接続することを意味します。この機能により、プロセッサの実行開始および停止、レジスタおよびメモリの確認 / 編集が可能になります。JTAG ターゲット接続は、Nios II IDE フラッシュ・プログラマを利用するための最低条件でもあります。

## ダウンロードおよび実行ソフトウェア

ソフトウェアのダウンロードとは、JTAG 接続を介して、実行可能コードとデータをプロセッサのメモリにダウンロードすることです。ソフトウェアをメモリにダウンロードすると、JTAG デバッグ・モジュールはデバッグ・モードを終了して、実行可能コードの先頭に実行を移すことができます。

## ソフトウェア・ブレークポイント

ソフトウェア・ブレークポイントでは、RAM に存在する命令に対してブレークポイントを設定できます。ソフトウェア・ブレークポイント・メカニズムは、RAM に格納されている実行可能コードにブレーク命令を書き込みます。プロセッサがブレーク命令を実行すると、制御は JTAG デバッグ・モジュールに移されます。

## ハードウェア・ブレークポイント

ハードウェア・ブレークポイントでは、フラッシュ・メモリなどの不揮発性メモリに存在する命令に対してブレークポイントを設定できます。ハードウェア・ブレークポイント・メカニズムは、プロセッサの現在の命令アドレスを継続的にモニタします。命令アドレスがハードウェア・ブレークポイント・アドレスと一致すると、JTAG デバッグ・モジュールがプロセッサを制御します。

ハードウェア・ブレークポイントは、JTAG デバッグ・モジュールのハードウェア・トリガ機能を使用して実現されます。

## ハードウェア・トリガ

ハードウェア・トリガは、リアルタイム・プログラム実行中の命令バスまたはデータ・バスの状態に基づいて、デバッグ動作をアクティブにします。トリガは単にプロセッサの実行を停止するだけではありません。例えば、リアルタイム・プロセッサ実行中のトレース・データ収集をイネーブルするのにも使用できます。

表 2-4 に、トリガが発生するすべての条件を示します。ハードウェア・トリガ条件は、命令バスまたはデータ・バスのいずれかに基づきます。同一バス上のトリガ条件は論理積をとることができるため、例えば JTAG デバッグ・モジュールをライト・サイクルでのみ特定のアドレスにトリガさせることができます。

表 2-4. トリガ条件		
条件	バス (1)	説明
特定のアドレス	D、I	バスが特定のアドレスにアクセスしたときにトリガ。
特定のデータ値	D	バス上に特定のデータ値が現れたときにトリガ。
リード・サイクル	D	リード・バス・サイクルでのトリガ。
ライト・サイクル	D	ライト・バス・サイクルでのトリガ。
アーム条件	D、I	アーム付きトリガ・イベント後にのみトリガ。2-20 ページの「アーム付きトリガ」を参照してください。
範囲	D	アドレス値、データ値、またはその両方の範囲でのトリガ。2-21 ページの「値の範囲でのトリガ」を参照してください。
注： (1) “I” は命令バス、“D” はデータ・バスを示します。		

プロセッサの実行中にトリガ条件が満たされると、JTAG デバッグ・モジュールは、実行の停止やトレース・キャプチャの開始などの動作をトリガします。表 2-5 に Nios II JTAG デバッグ・モジュールでサポートされるトリガ動作を示します。

### アーム付きトリガ

JTAG デバッグ・モジュールには、アーム付きトリガと呼ばれる 2 つのレベルの機能があります。アーム付きトリガにより、JTAG デバッグ・モジュールは、イベント A の発生時にのみイベント B でトリガできます。この例では、イベント A によって、イベント B のトリガをイネーブルするトリガ動作が発生します。

## 値の範囲でのトリガ

JTAG デバッグ・モジュールは、データ・バス上のデータまたはアドレス値の範囲でトリガすることができます。このメカニズムは、2つのハードウェア・トリガを併用し、指定範囲内において、ある値の範囲でアクティブになるトリガ情報を作成します。

動作	説明
ブレーク	実行を停止し、制御を JTAG デバッグ・モジュールに移します。
外部トリガ	トリガ信号出力をアサートします。このトリガ出力を使用して、例えば外部ロジック・アナライザをトリガすることができます。
トレース・オン	トレース収集をオンにします。
トレース・オフ	トレース収集をオフにします。
トレース・サンプル (1)	バスの 1 つのサンプルをトレース・バッファに格納します。
アーム	アーム付きトリガをイネーブルします。

注：  
 (1) データ・バス上の状態によってのみ、この動作がトリガされます。

## トレース・キャプチャ

トレース・キャプチャとは、プロセッサがリアルタイムでコードを実行中に、プロセッサの 1 命令ごとの実行を記録することです。JTAG デバッグ・モジュールには、以下のトレース機能があります。

- 実行とレースをキャプチャする (命令バス・サイクル)。
- データ・トレースをキャプチャする (データ・バス・サイクル)。
- 各データ・バス・サイクルでアドレス、データ、またはその両方をキャプチャする。
- トリガに基づいて、リアルタイムでのトレースのキャプチャを開始および停止する。
- ホスト制御のもとで手動でトレースを開始および停止する。
- トレース・バッファがいっぱいになったときに、オプションでトレースのキャプチャを停止して、プロセッサを実行させたままにする。
- JTAG デバッグ・モジュールのオンチップ・メモリ・バッファにトレース・データを格納する。(このメモリは、JTAG 接続によってのみアクセス可能。)
- トレース・データをオフチップ・デバッグ・プローブの大規模バッファに格納する。

一部のトレース機能には、サードパーティのデバッグ・プロバイダが提供する追加ライセンスまたはデバッグ・ツールが必要です。例えば、オンチップ・トレース・バッファは、Nios II プロセッサの標準機能ですが、オフチップ・トレース・バッファを使用するには、First Silicon Solution (FS2) または Lauterbach が提供する別のデバッグ・ソフトウェアとハードウェアが必要です。



詳細は、[www.fs2.com](http://www.fs2.com) および [www.lauterbach.com](http://www.lauterbach.com) を参照してください。

## 実行とデータ・トレース

JTAG デバッグ・モジュールは、命令バスのトレース（実行トレース）、データ・バスのトレース（データ・トレース）、あるいは同時に両方のトレースをサポートします。実行トレースでは、実行された命令のアドレスのみが記録されるため、ユーザーはメモリ内のどの場所（つまり、どの関数）でコードが実行されたかを解析できます。データ・トレースでは、データ・バス上の各ロードおよびストア操作に関連するデータが記録されます。

JTAG デバッグ・モジュールは、リアルタイムでデータ・バス・トレースをフィルタして、以下のキャプチャを実行できます。

- ロード・アドレスのみ
- ストア・アドレスのみ
- ロード・アドレスとストア・アドレスの両方
- ロード・データのみ
- ロード・アドレスとロード・データ
- ストア・アドレスとストア・データ
- ロードとストア両方のアドレスとデータ
- トリガ・イベント時のデータ・バスのシングル・サンプル

## フレームのトレース

フレームとは、トレース・データの収集用に割り当てられたメモリ単位です。ただし、フレームはトレースの深さを示す絶対的な尺度ではありません。

リアルタイムで動作中のプロセッサを追跡するために、実行トレースは、分岐、呼び出し、トラップ、割り込みなど、選択したアドレスのみを保存するように最適化されています。これらのアドレスから、ホスト側のデバッグ・ソフトウェアは、厳密な命令ごとの実行トレースを後で再構築することができます。さらに、実行トレース・データは、1つのフレームが複数の命令を表すような圧縮形式で保存されます。これらの最適化によって、実行中におけるトレース収集の実際の開始点と停止点は、ユーザーが指定した開始点および停止点とわずかに異なる場合があります。

データ・トレースでは、要求されたロードおよびストアの 100% がリアルタイムでトレース・バッファに格納されます。トレース・バッファへの格納時には、データ・トレース・フレームの優先順位は実行とレース・フレームより低くなります。したがって、データ・フレームは常に古いものから順に格納されますが、実行とレースとデータ・トレースが互いに正確に同期しているかどうかは保証されません。

## 参考資料

この章では以下のドキュメントを参照しています。

- 「Nios II コア実装の詳細」の章 (Nios II プロセッサ・リファレンス・ハンドブック)
- 「Instantiating the Nios II Processor in SOPC Builder」の章 (Nios II プロセッサ・リファレンス・ハンドブック)
- 「Nios II Custom Instruction User Guide」
- 「Instruction Set Reference」の章 (Nios II プロセッサ・リファレンス・ハンドブック)
- 「プログラミング・モデル」の章 (Nios II プロセッサ・リファレンス・ハンドブック)
- 「Avalon Memory Mapped Interface Specification」

## 改訂履歴

表 2-6 に、本資料の改訂履歴を示します。

表 2-6. 改訂履歴		
日付および ドキュメント・ バージョン	変更内容	概要
2008年5月 v8.0.0	MMU および MPU の項を追加。	MMU および MPU の項を追加。
2007年10月 v7.2.0	変更なし	
2007年5月 v7.1.0	<ul style="list-style-type: none"> <li>● 「はじめに」の項に目次を追加。</li> <li>● 「参考資料」の項を追加。</li> </ul>	
2007年3月 v7.0.0	前バージョンからの内容の変更はありません。	
2006年11月 v6.1.0	割り込みベクタ・カスタム命令の説明	割り込みベクタ・カスタム命令を追加
2006年5月 v6.0.0	<ul style="list-style-type: none"> <li>● オプションの <code>cpu_resetrequest</code> および <code>cpu_resettaken</code> の説明を追加</li> <li>● 単精度浮動小数点の項を追加。</li> </ul>	
2005年10月 v5.1.0	前バージョンからの内容の変更はありません。	
2005年5月 v5.0.0	密結合メモリを追加。	
2004年12月 v1.2	新しいコントロール・レジスタ <code>ct15</code> を追加	
2004年9月 v1.1	Nios II 1.01 のリリースにより更新。	
2004年5月 v1.0	初版	