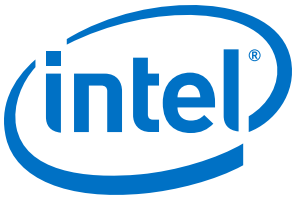


Nios II 浮動小数点ハードウェア 2 コンポーネント･ユーザーガイド

更新情報
フィードバック

ug-1175 | 2016.05.03
最新版をウェブからダウンロード: PDF | HTML

目次

1 Nios II 浮動小数点ハードウェア 2 コンポーネント･ユーザーガイド

1.1 概要

浮動小数点ハードウェア (FPH1)[1]は、単精度の加算、減算、乗算、および除算演算のカスタム命令を実装することで、浮動小数点ソフトウェアのエミュレーションにより大幅な性能の向上を実現します。ソフトウェアは FPH1 機能をエミュレートしてすべての他の浮動小数点演算 (倍精度演算を含む) を実装します。

浮動小数点ハードウェア (FPH2)[2]は、加算、減算、乗算、および除算演算の低サイクルカウントを達成し、追加の浮動小数点演算のカスタム命令を実装することで、さらに高レベルの性能を実現します。

この資料の完全な理解には、以下の資料に精通している必要があります。

- FPH1
- Nios® II Gen2 (or Classic) Processor Reference Handbook
- Nios II Gen2 (or Classic) Software Developer's Handbook
- Nios II Custom Instruction User Guide
- Using Nios II Floating-Point Custom Instructions Tutorial

関連情報

- Nios II Classic Processor Reference Handbook
- Nios II Gen2 Processor Reference Handbook
- Nios II Classic Software Developer's Handbook
- Using Nios II Floating-Point Custom Instructions Tutorial
- GCC Floating-point Custom Instruction Support Overview
- GCC Single-precision Floating-point Custom Instruction Command Line
- Nios II Gen2 Software Developer's Handbook
- Nios II Custom Instruction User Guide

1.1.1 概要

下の図は、ディスプレイ名 **Floating Point Hardware 2** および IP 名 **altera_nios_custom_instr_floating_point_2** の FPH2 コンポーネントの構造を示しています。**Floating Point Hardware 2** は、次のサブコンポーネントから構成されており、単一 Qsys コンポーネントにすべての浮動小数点演算が含まれています。

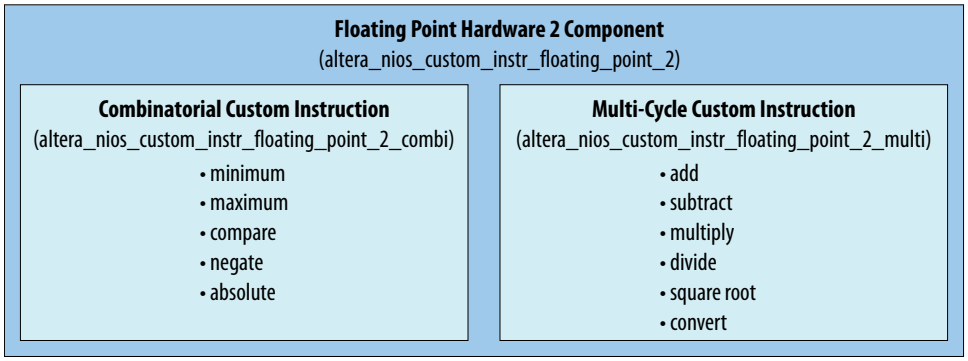
(1) 第一世代

(2) 第二世代

- **altera_nios_custom_instr_floating_point_2_combi**
- **altera_nios_custom_instr_floating_point_2_multi**

図 -1: カスタム命令の実装

この図は各カスタム命令で実装された浮動小数点演算をリストしています。



FPH2 には次の特長があります。

- FPH1 演算 (加算、減算、乗算、除算) のサポート、および平方根、比較、整数変換、最小値、最大値、無効値、絶対値のサポートを追加
- Nios II 汎用レジスターに格納された単精度浮動小数点値
- VHDL 専用
- Qsys のみサポート
- 単精度のみ
- 4 入力 LE および 18 ビット乗算器の FPGA の最適化
- GCC および Nios II SBT (ソフトウェア・ビルド・ツール) ソフトウェアのサポート
- 以下を除いて IEEE 754-2008 に準拠
  - 簡略化された丸め
  - 簡略化された NaN 処理
  - 例外なし
  - ステータスフラグなし
  - 演算のサブセットでサポートされる非正規化数
- FPH1 とのバイナリー互換性
  - FPH1 は Round-To-Nearest の丸めモードを実装し、FPH2 は異なる丸めモードを実装するため、両世代間での結果はわずかに異なる場合があります。

1.1.2 FPH2 のリソース使用率

一般的なシステムに追加した場合、FPH2 は次のリソース率を消費します。

- およそ 2500 の 4 入力 LE
- 9 個の 9 ビット乗算器
- 3 個またはそれ以上の M9K メモリー

1.1.3 浮動小数点ハードウェア 2 のベンチマーク

浮動小数点ハードウェア 2 コンポーネントは次のハードウェア・コンフィグレーションでベンチマークされています。

- Cyclone V 5CGXFC7D6F31C6 デバイス
- Nios II プロセッサー
- オンチップ・メモリー
- パイプライン・ブリッジ
- タイマー
- JTAG UART
- 浮動小数点ハードウェア 2

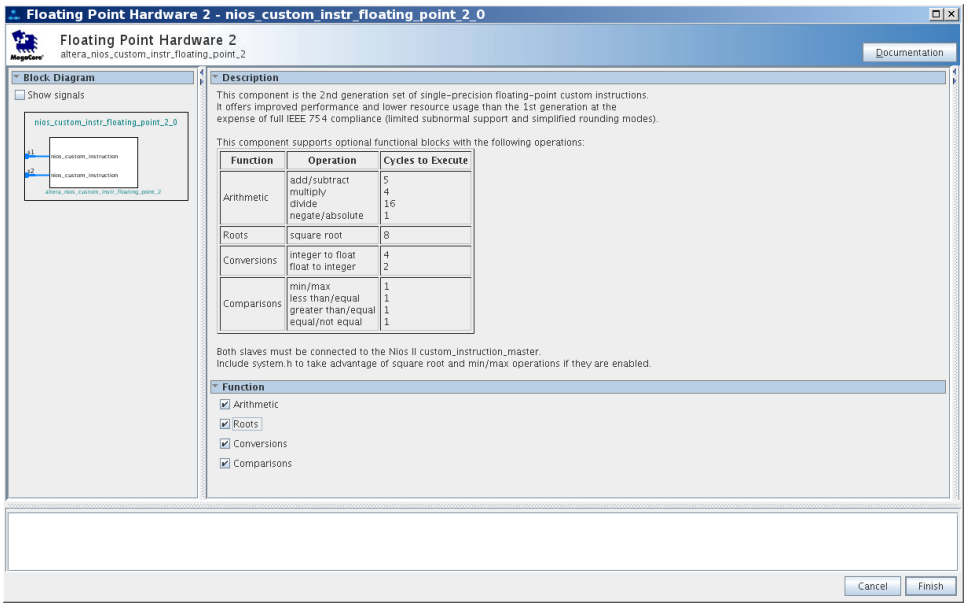
ハードウェア・デザインは F_{MAX}=125 MHz を達成します。

1.2 Qsys での FPH2 の使用

FPH2 コンポーネントをシステム内でインスタンス化するには、Qsys で、Component ライブラリーの Project エリアの **Floating Point Hardware 2** コンポーネントを探します。FPH2 コンポーネントは、Component ライブラリー内の「Embedded Processors」グループの下にあります。

Floating Point Hardware 2 コンポーネント・エディターは、下の図で示すように、浮動小数点のカスタム命令のいくつかのグループのいずれかを選択して有効にすることができます。デフォルトではすべての命令が有効です。

図 -2: Floating Point Hardware 2 コンポーネント・エディター



ほとんどの場合、有効になっているすべての浮動小数点のカスタム命令は放置してもかまいません。しかしながら、特定のコンフィグレーションでの MAX 10 デバイスファミリーでは、**Roots** グループを無効にしなければなりません。

MAX 10 デバイスは次のコンフィグレーションで FPH2 の平方根命令をサポートすることができません。

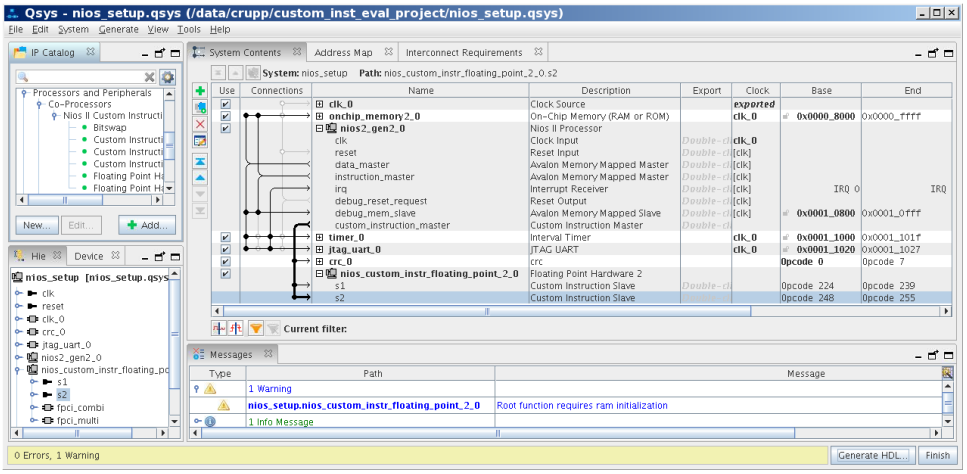
- Dual configuration モード
- Compressed configuration モード
- 無効である外部 RAM 初期化

平方根命令はルックアップ・テーブルを使用するため、MAX 10 がこれらのコンフィグレーションでサポートできない初期化を必要とします。これらのコンフィグレーションのうちの 1 つを対象とする MAX 10 デバイスがある場合は、**Roots** オプションをオフにしてください。

浮動小数点の命令グループの 1 つを無効にすると、必要に応じて、ソフトウェアはそのグループ (この場合は平方根) に関数を実装しなければなりません。BSP ジェネレーターはこのサポートを自動的に作成します。詳しくは「Nios II SBT」を参照してください。

下の図は Nios II が FPH2 に接続された Qsys を示しています。FPH2 はスレーブが 2 つ (s1 と s2) あります。1 つのスレーブは組み合わせカスタム命令で、もう 1 つはマルチサイクル・カスタム命令です。両スレーブを Nios II custom_instruction_master に接続するには、接続パッチパネルのドットをクリックします。接続状態の外観図を下に示します。

図 -3: Qsys での浮動小数点ハードウェア 2 コンポーネント



上の図の例は MAX 10 デバイスを対象としています。平方根関数における RAM の初期化に問題がある可能性があることを示す警告メッセージに注意してください。

FPH2 を Nios II に接続した後、通常どおりに Qsys でシステムを生成します。次に、Quartus® Prime ソフトウェアを使用して生成した RTL をコンパイルするか、もしくは、Modelsim™のような RTL シミュレーターを使用して、シミュレーションを実行します。

注意: Nios II ソフトウェア・ビルド・ツール (SBT) を使用してソフトウェア・プロジェクトを作成する場合、BSP ジェネレーターは浮動小数点ハードウェア用のカスタム Newlib ライブラリーを作成します。浮動小数点ハードウェア・コンフィグレーションを変更する場合は、Newlib が正しくビルドされるように BSP を再生成して再ビルドする必要があります。詳しくは、「Nios II SBT」を参照してください。

関連情報

- 20 ページの Nios II SBT

- Quartus Prime Standard Edition Handbook Volume 1: Verification
- 19 ページの Newlib ライブラリー

1.3 浮動小数点のバックグラウンド

関連情報

- GCC Floating-point Custom Instruction Support Overview
- GCC Single-precision Floating-point Custom Instruction Command Line

1.3.1 IEEE 754 形式

下の図は IEEE 754 の 32 ビット単精度値のフィールドを示しています。表はフィールドの概要です。通常の単精度浮動小数点数は、値 $(-1)^{S} * 1.FRAC * 2^{EXP-127}$ です。

図 -4: 単精度形式

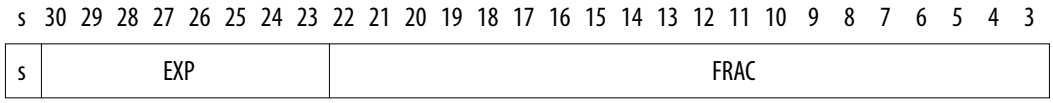


表 1. 単精度フィールドの概要

ニーモニック	名前	説明
FRAC	仮数部	仮数の小数部分（二進数の右側）を指定する。整数値（二進数の左側）は常に値 1 とみなされるため省略される。この省略された値は隠れビットと呼ばれる。仮数範囲は ≥1.0 ~ <2.0 である。
EXP	バイアス指数部	値 127 でバイアスされた指数を含む。バイアスされた指数値 0x0 はゼロ値および非正規化数値に予約される。バイアスされた指数値 0xff は無限大および NaN に予約される。バイアスされた指数の範囲は正規化数で 1 ~ 0xfe（バイアスを差し引くと -126 ~ 127）である。
S	符号	符号を指定する。1 = 負、0 = 正。正規化数値、ゼロ、無限大、非正規化数値はすべて符号化される。NaN には符号がないため、S フィールドは無視される。

IEEE 754 標準は次の特別な値を提供しています。

- ゼロ (EXP=0, FRAC=0)
- 非正規化数 (EXP=0, FRAC≠0)
- 無限大 (EXP=255, FRAC=0)
- NaN (EXP=255, FRAC≠0)

注意: ゼロ、非正規化数、および無限大は S フィールドで指定されたように符号化されます。NaN には符号がないため、S フィールドは無視されます。

1.3.1.1 最終桁単位 (Unit in the Last Place : ULP)

最終桁単位 (ULP) は値 2^{-23} を表し、およそ 1.192093e-07 です。ULP は、指数範囲に上界がないと仮定して、広がっている最も近い浮動小数点数 a と b (a ≤ x ≤ b, a ≠ b) の間の距離です。IEEE の Round-to-Nearest モードは、2 分の 1 ULP の最大エラーの結果を生成します。その他の IEEE の丸めモード (Round-to-Zero、Round-to-Positive-Infinity、および Round-to-Negative-Infinity) は 1 ULP の最大エラーの結果を生成します。

1.3.1.2 値のエンコーディング

次の表は、単精度の値が 0x0000_0000 から 0xffff_ffff までの 32 ビットの範囲でどのようにエンコードされるかを示しています。単精度浮動小数点数には次の特長があります。

- 精度 (ρ) = 24 ビット (FRAC の 23 ビット + 1 隠れビット)
- 基数 (β) = 2
- e_{min} = -126
- e_{max} = 127

FRAC の 最上位ビット (MSB) は、Signaling NaN (sNaN) では 0、Quiet NaN (qNaN) では 1 です。

表 2. 値のエンコーディング

16 進数の値	名前	S	EXP	FRAC	値 (十進数)
0x0000_0000	+0	0	0x00	0x00_0000	0.0
0x0000_0001	最小 pos 非正規化数	0	0x00	0x00_0001	1.40129846e–45 ($\beta^{emin-\rho+1} = 2^{-126-24+1} = 2^{-149}$)
0x007f_ffff	最大 pos 非正規化数	0	0x00	0x7f_ffff	1.1754942e-38
0x0080_0000	最小 pos 正規化数	0	0x01	0x00_0000	1.17549435e–38 ($\beta^{emin} = 2^{-126}$)
0x3f80_0000	1	0	0x7f	0x00_0000	1.0 ($1.0x2^{0}$)
0x4000_0000	2	0	0x80	0x00_0000	2.0 ($1.0x2^{1}$)
0x7f7f_ffff	最大 pos 正規化数	0	0xfe	0x7f_ffff	3.40282347e+38 ($(\beta - \beta^{1-\rho}) * 2^{emax} = (2 - 2^{1-24}) * 2^{127} = (2 - 2^{23}) * 2^{127}$
0x7f80_0000	+∞	0	0xff	0x00_0000	
0x7f80_0001	最小 sNaN (pos 符号)	0	0xff	0x00_0001	
0x7fdf_ffff	最大 sNaN (pos 符号)	0	0xff	0x3f_ffff	
0x7fe0_0000	最小 qNaN (pos 符号)	0	0xff	0x40_0000	
0x7fff_ffff	最大 qNaN (pos 符号)	0	0xff	0x7f_ffff	
0x8000_0000	-0	1	0x00	0x00_0000	-0.0
0x8000_0001	最大 neg 非正規化数	1	0x00	0x00_0001	-1.40129846e–45
0x807f_ffff	最小 neg 非正規化数	1	0x00	0x7f_ffff	-1.1754942e-38
0x8080_0000	最大 neg 正規化数	1	0x01	0x00_0000	-1.17549435e–38
0xff7f_ffff	最小 neg 正規化数	1	0xfe	0x7f_ffff	-3.40282347e+38
0xff80_0000	-∞	1	0xff	0x00_0000	
0xff80_0001	最大 sNaN (neg 符号)	1	0xff	0x00_0001	

continued...

16 進数の値	名前	S	EXP	FRAC	値 (十進数)
0xffdf_ffff	最小 sNaN (neg 符号)	1	0xff	0x3f_ffff	
0xffe0_0000	最大 qNaN (neg 符号)	1	0xff	0x40_0000	
0xffff_ffff	最小 qNaN (neg 符号)	1	0xff	0x7f_ffff	

1.3.2 丸め手法

浮動小数点演算の正しい結果が浮動小数点の値として正確に表現できない場合、丸め処理が行われます。

IEEE 754-2008 標準はデフォルトの丸めモードを「Round-to-Nearest RoundTiesToEven」として定義しています。これは IEEE 754-1985 標準では「Round-to-Nearest-Even」と呼ばれています。加えて、両標準は「Round-to-Zero」、「Round-to-Negative-Infinity」、および「Round-to-Positive-Infinity」の丸めモードも定義しています。IEEE 754-2008 標準では、「Round-to-Nearest RoundTiesAway」という新たな丸めモードのオプションが導入されました。

FPH2 演算は、Nearest Rounding (RoundTiesAway)、Truncation Rounding、または Faithful Rounding をサポートしています。丸めの種類は演算関数および表 4-2 で指定されています。ソフトウェア・エミュレーション・ライブラリー (FPH 演算が未提供の場合に使用される) および FPH1 は Round-to-Nearest RoundTiesToEven を実装するため、FPH2 とその他のソリューション間の結果が異なる場合があります。

1.3.2.1 Nearest Rounding

Nearest Rounding は IEEE 754-2008 標準の「Round-to-Nearest RoundTiesAway」の丸めモードに対応しています。Nearest Rounding は最も近い単精度数の方向へ丸めます。結果が 2 つの単精度数の値の中間の場合、丸め処理はより上限の数 (正の結果ならより大きい方の値、負の結果ならより小さい方の値) を採用します。

Nearest Rounding は 2 分の 1 ULP の最大誤差があります。結果がランダムに分布している場合、Nearest Rounding はより下限数よりもより上限数を採用するため、誤差は均等には分布されません。

1.3.2.2 Truncation Rounding

Truncation Rounding は IEEE 754-2008 標準の「Round-To-Zero」の丸めモードに対応しています。Truncation Rounding はより下限に最も近い単精度数の結果になります。

Truncation Rounding は 1 ULP の最大誤差があります。誤差は均等には分布されません。

1.3.2.3 Faithful Rounding

Faithful Rounding は、より上限かより下限に最も近い単精度数の結果になります。したがって、Faithful Rounding は 2 つの正の値のうちの 1 つになります。2 つの値の間の選択は定義されていません。

Faithful Rounding は 1 ULP の最大誤差があります。誤差は均等には分布されません。

注意: Faithful Rounding モードは IEEE 754 で定義されていません。

1.3.2.4 丸めの例

下の表は、1 桁での整数または小数のように、2 桁の精度で丸められた正規化された十進数値とみなす十進数でサポートされる丸めの例を示しています。

表 3. 十進数丸めの例

丸められていない値	Nearest Rounding	Truncation Rounding	Faithful Rounding
3.34	3.3	3.3	3.3 または 3.4
6.45	6.5	6.4	6.4 または 6.5
2.00	2.0	2.0	2.0 または 2.1
8.99	9.0	8.9	8.9 または 9.0
-1.24	-1.2	-1.2	-1.2 または -1.3
-3.78	-3.8	-3.7	-3.7 または -3.8

1.3.3 特殊なケース

次の表は、いくつかの IEEE 754 標準の特殊ケースの結果を示しています。*x* は正規化数値を表しています。FPH2 はこれらのすべてのケースに準拠しています。

結果は正しく符号化されているとみなされ、重要ではない場合は符号は省略されます。符号が関係する場合、(+∞) のように値を括弧で囲んで符号を表現します。表にある値 *x* は NaN 以外の値を表します。

比較は等式ではゼロの符号を無視します。例えば、(-0) == (+0) と (+0) ≤ (-0) です。> や < のような等式を含まない比較は、-0 を +0 未満であるとみなしません。一方または両方の入力が NaN の場合、比較は false を返します。また、片方の入力が NaN で他方が NaN 以外の場合、min と max 演算は NaN 以外の入力を返します。いずれかまたはすべての入力が NaN である場合、浮動小数点の結果を生成するその他の演算は NaN を返します。

表 4. 特殊なケース

演算	特殊なケース			
fdivs	0/0=NaN	∞/∞=NaN	0/∞=0, ∞/0=∞	NaN/*x*=NaN, *x*/NaN=NaN, NaN/NaN=NaN
fsubs	(+∞)-(+∞)=NaN	(-∞)-(-∞)=NaN	(-0)-(-0)=+0	NaN-*x*=NaN, *x*-NaN=NaN, NaN-NaN=NaN
fadds	(+∞)+(-∞)=NaN	(-∞)+(+∞)=NaN	(+0)+(-0)=+0, (-0)+(+0)=+0	NaN+*x*=NaN, *x* +NaN=NaN, NaN +NaN=NaN
fmuls	0*∞=NaN	∞*0=NaN		NaN*x=NaN, *x*NaN=NaN, NaN*NaN=NaN
fsqrts	sqrt(-0) =-0	sqrt(*x*) =NaN, *x*<-0		sqrt(NAN) =NaN
fixsi & round	int(>2^{31}-1)= 0x7fffffff, int(+∞)=0x7fffffff	int(>2^{31}-1)= 0x7fffffff, int(+∞)=0x7fffffff		int(NaN)=undefined
fmins	min((+0),(-0))=(-0)	min((-0),(+0))=(-0)		min(NaN,n)=*x*, min(*x*,NaN)=*x*, min(NaN,NaN)=NaN, min(+∞,*x*)=*x*, min(-∞,*x*)=-∞

continued...

演算	特殊なケース			
fmaxs	max((+0),(-0))=(+0)	max((-0),(+0))=(+0)		max(NaN,*x*)=*x*, max(*x*,NaN)=*x*, max(NaN,NaN)=NaN, max(+∞,*x*)=+∞, max(-∞,*x*)=*x*
fcmplts (<)	(+∞)<(+∞)=0	(-∞)<(-∞)=0	(-0)<(+0)=0, (+0)<(-0)=0	NaN<*x*=0, *x*<NaN=0, NaN<NaN=0
fcmples (≤)	(+∞)≤(+∞)=1	(-∞)≤(-∞)=1	(+0)≤(-0)=1, (-0)≤(+0)=1	NaN≤*x*=0, *x*≤NaN=0, NaN≤NaN=0
fcmpgts (>)	(+∞)>(+∞)=0	(-∞)>(-∞)=0	(-0)>(+0)=0, (+0)>(-0)=0	NaN>*x*=0, *x*>NaN=0, NaN>NaN=0
fcmpges (≥)	(+∞)≥(+∞)=1	(-∞)≥(-∞)=1	(-0)≥(+0)=1, (+0)≥(-0)=1	NaN≥*x*=0, *x*≥NaN=0, NaN≥NaN=0
fcmpeqs (=)	(+∞)=(+∞)=1	(-∞)=(-∞)=1	(-0)=(+0)=1	(NaN==*x*)=0, (*x*==NaN)=0, (NaN==NaN)=0
fcmpnes (≠)	(+∞)≠ (+∞)=0	(-∞)≠ (-∞)=0	(-0)≠(+0)=0	NaN≠*x*=0, *x*≠NaN=0, NaN≠NaN=0

1.4 機能の概要

FPH2 は組み合わせカスタム命令 1 つとマルチサイクル・カスタム命令 1 つで実装されています。組み合せカスタム命令は、比較、最小、最大、負、および絶対の演算を実装しています。マルチサイクル・カスタム命令は、加算、減算、乗算、除算、平方根、および変換の演算を実装しています。

注意: すべての演算を必要とします。コンフィグレーション可能なオプションはありません。

1.4.1 IEEE 754 準拠

浮動小数点ハードウェア 2 の演算は、以下を除いて IEEE 754-2008 標準に準拠しています。

- トラップ方式なし / 例外方式なし
- ステータスフラグなし
- 二進数と十進数の間の剰余演算および変換演算はサポートされません。これらはソフトウェア・エミュレーション・ライブラリーで提供されます。
- Round-to-Nearest-Even モードはサポートされません。Nearest Rounding、Truncation Rounding、または Faithful Rounding はオペレーターに起因して使用されます。
- 非正規化数は、加算、減算、乗算、除算、および平方根の演算ではサポートされません。非正規化数の入力は、符号付きゼロとして扱われ、非正規化数の出力は生成されません (代わりに結果は符号付きゼロになります)。この非正規化数の値の処理は flush-to-zero と呼ばれます。(3)

(3) 非正規化数は比較、最小、最大、浮動 - 整数、否定および絶対の演算でサポートされるため、これらの演算は IEEE 754-2008 に準拠しています。

- 非正規化数は integer2float 変換演算で生成できません。この動作は IEEE 754 に準拠しています。
- Signaling と Quiet NaN の間は入力オペランドとして区別しません。NaN となるすべての結果は、Signaling または Quiet NaN のいずれかになる場合があります。
- 1 つ以上の NaN 入力オペランドを有する NaN の結果は、入力 NaN 値のいずれも返されません。つまり、NaN の結果は入力する NaN とは異なる NaN になります。

1.4.2 例外処理

FPH2 コンポーネントは例外をサポートしていません。代わりに、特別な結果を作成します。次の表は、IEEE 754 の例外をトリガーし得る演算のために作成された FPH2 の結果を示しています。

表 5. IEEE 754 の例外のケース

IEEE 754 の例外	FPH2 の結果
無効	NaN
0 除算	符号付き整数
オーバーフロー	符号付き整数
アンダーフロー	符号付き 0
不正確	正規化数

1.4.3 演算

下の表は FPH2 演算の概要の詳細です。「a」と「b」は単精度浮動小数点値とみなされます。各列についての情報をリストしています。

- **演算** [4]—浮動小数点演算の名前を提供します。名前は、GCC サポートがない「丸め」を除き、対応する GCC 浮動小数点ハードウェアのコマンドラインのオプションの名前に一致します。
- **N**—演算で 8 ビット固定カスタム命令 N 値を提供します。FPH2 コンポーネントは上位 32 の Nios II カスタム命令 N 値 (224 ~ 255) を占める固定 N 値を使用します。また、FPH1 も固定 N 値 (252 ~ 255) を使用します。FPH2 は互換性の維持のために同じ演算をこれらの N 値に割り当てます。
- **サイクル** [5]—命令を実行するのに必要なサイクル数を指定します。組み合わせカスタム命令は 1 サイクルかかります。マルチサイクル・カスタム命令は常に少なくとも 2 サイクルが必要です。Nios II はカスタム命令の結果をレジスターし、ソースオペランド・バイパス・マルチプレクサーの g 配線遅延でもう一つのサイクルも可能にするため、N サイクルカスタム命令はカスタム命令の内部に N-2 レジスターステージを有します。Nios II/f プロセッサーは、命令が 2 サイクル内で結果を使用する場合にマルチサイクル・カスタム命令に従って命令をストールするため、**サイクル**列は余分なサイクル (最大 2) を含みません。マルチサイクル命令は遅延の結果命令であるため、これらの余分なサイクルが必要です。
- **結果**—演算で実行される計算を記述します。

---

[4] 詳しくは、「-mcustom-<operation>」を参照してください。

[5] 詳しくは、Nios II プロセッサー・リファレンス・ハンドブックの 1 つを参照してください。

- **非正規化数**—演算で非正規化数の入力および出力を扱う方法を記述します。
- **丸め** [6]—FPH2 コンポーネントが結果を丸める方法を記述します。可能な選択肢は、Nearest、Truncation、Faithful、および None です。
- **GCC インターフェイス**—GCC がカスタム命令の動作を推測する C コードを示します。

表 6. FPH2 演算の概要

演算	N	サイクル	結果	非正規化数	丸め	GCC インターフェイス
fdivs	255	16	a/b	flush-to-0	Nearest	a/b
fsubs	254	5	a-b	flush-to-0	Faithful	a-b
fadds	253	5	a+b	flush-to-0	Faithful	a+b
fmuls	252	4	a*b	flush-to-0	Faithful	a*b
fsqrts	251	8	sqrt(a)	flush-to-0	Faithful	sqrtf()
floatis	250	4	int_to_float(a)	適用なし	適用なし	キャスティング
fixsi	249	2	float_to_int(a)	flush-to-0	Truncation	キャスティング
round	248	2	float_to_int(a)	flush-to-0	Nearest	lroundf()[7]
reserved	234 ~ 247	未定義	未定義			
fmins	233	1	(a<b) ? a : b	サポート可	None	fminf()[7]
fmaxs	232	1	(a<b) ? b : a	サポート可	None	fmaxf()[7]
fcmplts	231	1	(a<b) ? 1 : 0	サポート可	None	a<b
fcmples	230	1	(a≤b) ? 1 : 0	サポート可	None	a<=b
fcmpgts	229	1	(a>b) ? 1 : 0	サポート可	None	a>b
fcmpges	228	1	(a≥b) ? 1 : 0	サポート可	None	a>=b
fcmpeqs	227	1	(a=b) ? 1 : 0	サポート可	None	a==b
fcmpnes	226	1	(a≠b) ? 1 : 0	サポート可	None	a!=b
fnegs	225	1	-a	サポート可	None	-a
fabss	224	1	|a|	サポート可	None	fabsf()

関連情報

- 9 ページの 丸め手法
- 15 ページの -mcustom-<operation>
- Nios II Classic Processor Reference Handbook
- Nios II Gen2 Processor Reference Handbook
- 19 ページの round()、fmins()、および fmaxs() での C マクロ
- GCC Command Line Options

---

[6] 詳しくは、「丸め手法」を参照してください。丸め「None」は結果を丸める必要性がないことを意味します。

[7] Nios II GCC はこれらの Newlib 浮動小数点関数への呼び出しを同等のカスタム命令で確実に置き換えることはできません。これらの関数の使用については、「round()、fmins()、および fmaxs() での C マクロ」を参照してください。

- Newlib Documentation page
- GCC Floating-point Custom Instruction Support Overview
- GCC Single-precision Floating-point Custom Instruction Command Line
- Nios II Custom Instruction User Guide

1.5 ソフトウェアの問題

1.5.1 Nios II GCC

Nios II エンベデッド・デザインスイートは Nios II GCC を含んでいます。FPH2 コンポーネントは Nios II EDS のバージョン 15.1 以降 (GCC v4.7.3 以降) でサポートされています。

関連情報

- GCC Command Line Options
- GCC Floating-point Custom Instruction Support Overview
- GCC Single-precision Floating-point Custom Instruction Command Line

1.5.1.1 インターフェイス

GCC コンパイラーはほとんどの FPH2 演算を C ソースコードから推測します。「演算」にある表はすべての演算をリストしており、FPH2 がどのように推測するかを示しています。

注意: GCC は Newlib 数学関数を推測しません。これらの関数は、GCC の __builtin_custom_ * 機能を使用して同等のカスタム命令に置き換えられます。

`system.h` ヘッダーファイルは、対応しているカスタム命令に代わって使用するために必要な Newlib 数学関数を再定義する C `#define` マクロ宣言を提供します。

関連情報

- 19 ページの round()、fmins()、および fmaxs() での C マクロ
- 12 ページの 演算
- Newlib Documentation page

1.5.1.2 変換

FPH2 コンポーネントは符号付き整数型 (C `short`、`int` と `long` 型) と 32 ビット単精度浮動小数点型 (C `float` 型) の間の変換での関数を提供します。Nios II GCC コンパイラーは、コンパイルされたコードが C キャストなどこれらの型の間でデータを変換する際に、これらのハードウェア関数を推測します。

FPH2 コンポーネントは符号なし整数型と浮動小数点の間の変換での関数を提供していません。符号なし整数型と浮動小数点型の間で変換を行う場合、コンパイラーはソフトウェア・エミュレーションを実装します。したがって、符号なし整数型との変換は符号付き整数との変換よりもはるかに遅くなります。

浮動小数点数を符号なし整数に直接変換する際に得られる正の値の余分な範囲が必要ない場合は、目的の符号なし整数型にキャストするように C コードを変更すると、FPH2 を使用でき、ソフトウェア・エミュレーションの使用を回避できます。例えば、

```
float f;
unsigned int s = (unsigned int)f; // Software emulation
```

に代わって、次を使用します。

```
float f;
unsigned int s = (unsigned int)(int)f; // FPH2
```

FPH2 は単精度浮動小数点値を符号付き整数値に変換するための 2 つの演算を提供します。

- `fixsi`
- `round`

`fixsi` 演算は浮動小数点から符号付き整数に変換する際に切り捨てを実行します。例えば、`fixsi` 演算は 4.8 から 4 および -1.5 から -1 へと変換します。GCC は C 標準に準拠し、ソースコードがキャストを使用するたびに、または C が自動的に浮動小数点を符号付き整数に変換するたびに、`fixsi` 演算を呼び出します。

`round` 演算は浮動小数点から符号付き整数に変換する際に Nearest Rounding (tie-rounds-away) を実行します。例えば、`round` は 4.8 から 5、および -1.5 から -2 へと変換します。ソフトウェアはカスタム命令を直接呼び出すか、`lroundf()`関数を置き換える `system.h` に提供されている `#define` を使用することで、`round` 演算を呼び出すことができます。

関連情報

Newlib Documentation page

1.5.1.3 Nios II 浮動小数点演算

GCC の Nios II ポートでのみ提供される GCC オプションを説明します。

1.5.1.3.1 -mcustom-<operation>

`-mcustom-<operation>`コマンドライン・オプションは指定された演算のエミュレーションに代わり、カスタム命令を呼び出すように GCC に指示します。`-mcustom-<operation>`の構文は次のとおりです。

```
-mcustom-<operation>=N
```

N カスタム命令値は符号なしの十進数です。演算とそれらの N 値の完全なリストについては「演算」にある表を参照してください。

デフォルトでは、コンパイラーはソフトウェアですべての浮動小数点演算を実装します。また、`-mno-custom-<operation>`コマンドライン・オプションを使用して個別の命令に対してソフトウェア・エミュレーションを指定することもできます。

注意: コマンドラインは複数の`-mcustom-`スイッチを指定することができます。競合が発生した場合、コマンドラインの最後のスイッチが有効になります。

GCC が推測できる FPH2 提供のすべての演算の使用を指示するには、次のコマンドライン・オプションを GCC に渡す必要があります。詳しくは、「インターフェイス」を参照してください。

Nios II SBT のユーザーの場合、これらのコマンドライン引数は生成された makefile により GCC の呼び出しに自動的に追加されます。詳しくは「Nios II SBT」を参照してください。

```
-mcustom-fabss=224
-mcustom-fnegs=225
-mcustom-fcmpnes=226
-mcustom-fcmpeqs=227
-mcustom-fcmpges=228
-mcustom-fcmpgts=229
-mcustom-fcmples=230
-mcustom-fcmplts=231
-mcustom-fmins=232
-mcustom-fmaxs=233
-mcustom-round=248
-mcustom-fixsi=249
-mcustom-floatis=250
-mcustom-fmuls=252
-mcustom-fadds=253
-mcustom-fsubs=254
-mcustom-fdivs=255
```

注意: 丸め演算のコマンドライン・オプションはありません。

関連情報

- 14 ページの インターフェイス
- 12 ページの 演算
- 20 ページの Nios II SBT

1.5.1.3.2 pragma

GCC はソースコード・ファイルにある pragma をサポートし、`-mcustom` コマンドライン・オプションをオーバーライドします。pragma はソースファイル全体に影響を与えます。

次の pragma は、指定した浮動小数点演数を実装するためにカスタム命令 N (N は 0 ~ 255 の十進整数) を呼び出すように GCC に指示します。

```
#pragma GCC target(“custom-<operation>=N”)
```

次の pragma は、指定した浮動小数点演数を実装するためにカスタム命令の代わりにソフトウェア・エミュレーションを使用するように GCC に指示します。

```
#pragma GCC targer(“no-custom-<operation>”)
```

注意: 丸め演算の pragma のサポートはありません。

1.5.1.3.3 -mcustom-fpu-cfg

GCC リンカーのコマンドラインで`-mcustom-fpu-cfg` オプションを指定すると、浮動小数点サポートを有するプリコンパイルされた Newlib ライブラリーが選択されます。プリコンパイルされたライブラリーは FPH1 でサポートされている演算 (加算、減算、乗算、除算) のみを使用します。

注意: アルテラは、FPH2 での`-mcustom-fpu-cfg` オプションの使用を推奨していません。

関連情報

- Newlib Documentation page
- 19 ページの Newlib ライブラリー

- "Nios II Options" in GCC Command Options (gcc.gnu.org)

1.5.1.4 ジェネリック浮動小数点のオプション

GCC はオプションを提供しており、Nios II GCC による提供のみではありません。ただし、これらのオプションには Nios II 固有の動作がある場合があります。

1.5.1.4.1 -fno-math-errno

GCC の資料より：

```
“Do not set ERRNO after calling math functions that are executed with a single
instruction, e.g., sqrt. A program that relies on IEEE exceptions for math error
handling may want to use this flag for speed while maintaining IEEE arithmetic
compatibility.”
```

GCC コマンドラインで`-fno-math-errno` を指定すると、コンパイラーは `sqrtf()`への呼び出しを `fsqrts` カスタム命令に直接マップします。それ以外の場合、デフォルトの GCC は `fsqrts` カスタム命令の後に NaN の結果を確認するためにいくつかの命令を追加し、負の数の平方根を取得するように表示します。`fsqrts` が NaN を返す場合、コードは Newlib `sqrtf()`関数を呼び出して C `errno` 変数を設定します。

一般的に、オーバーヘッドは好ましくありません。アルテラは `-fno-math-errno` を有効にし `sqrtf()`の呼び出しのオーバーヘッドを排除することを推奨します。

Nios II SBT を使用する場合は、生成した makefile はデフォルトで`-fno-math-errno` を設定します。この動作は CPPFLAGS make 変数で `-fmath-errno` を設定することでオーバーライドできます。

`-ffinite-math-only` オプションも平方根の NaN の結果を確認するオーバーヘッドを排除します。しかしながら、このオプションは他に影響を与えます。このオプションについて詳しくは「-ffinite-math-only」を参照してください。

関連情報

- 20 ページの Nios II SBT
- 18 ページの -ffinite-math-only
- Newlib Documentation page

1.5.1.4.2 -fsingle-precision-constant

GCC の資料より：

```
“Treat floating-point constants as single-precision constants instead of
implicitly converting them to double-precision constants.”
```

FPH2 では、Nios II SBT はデフォルトにより makefile GCC コマンドラインから`-fsingle-precision-constant` を省略します。この動作は FPH1 の SBT サポートとは対照的で、`-mcustom-fpu-cfg` でこのオプションを設定します。SBT は倍精度コードで問題が発生するため、FPH2 において `-fsingle-precision-constant` を使用しません。

コードで問題が発生しないことが確かな場合は、-fsingle-precision-constant を有効にできます。一般的には、これらのアプローチは局在化し、コンパイラーのオプションに依存しないため、float 型に浮動小数点の定数をキャストするか、「f」の接尾辞 (例 :3.14f) を使用する方が好ましいです,

関連情報

20 ページの Nios II SBT

1.5.1.4.3 -funsafe-math-optimizations

GCC の資料より：

```
“Allow optimizations for floating-point arithmetic that (a) assume that arguments
and results are valid and (b) may violate IEEE or ANSI standards. When used at
link-time, it may include libraries or startup files that change the default FPU
control word or other similar optimizations.”
```

FPH2 は超越関数 (sin()、cos()、tan()、atan()、exp()、および log()) を実装しないため、-funsafe-math-optimizations オプションは必要ありません。

このオプションは浮動小数点ハードウェアが超越関数を実装している場合に必要となります。GCC は、問題の可能性があるハードウェア・アクセラレーターをアプリケーション・コードが誤って使用しないようにするために、このオプションが必要です。

1.5.1.4.4 -ffinite-math-only

GCC の資料より：

```
“Allow optimizations for floating-point arithmetic that assume that arguments and
results are not NaNs or +-Infs.”
```

プログラマーは、どのようにコードに影響するかを調べるためにこのオプションの試用をお勧めします。

-ffinite-math-only オプションは、–fno-math-errno のような sqrtf()の呼び出しで作成された GCC オーバーヘッドも排除します 。

関連情報

17 ページの -fno-math-errno

1.5.1.4.5 -fno-trapping-math

GCC の資料より：

```
“Compile code assuming that floating-point operations cannot generate user-
visible traps. These traps include division by zero, overflow, underflow, inexact
result and invalid operation. This option implies -fno-signaling-nans. Setting
this option may allow faster code if one relies on “non-stop” IEEE arithmetic,
for example.”
```

プログラマーは、どのようにコードに影響するかを調べるためにこのオプションの試用をお勧めします。

1.5.1.4.6 -frounding-math

GCC の資料より：

```
“Disable transformations and optimizations that assume default floating point
rounding behavior. This is round-to-zero for all floating point to integer
conversions, and round-to-nearest for all other arithmetic truncations. This
option should be specified for programs that change the FP rounding mode
dynamically, or that may be executed with a non-default rounding mode. This
option disables constant folding of floating point expressions at compile-time
(which may be affected by rounding mode) and arithmetic transformations that are
unsafe in the presence of sign-dependent rounding modes.”
```

プログラマーは、どのようにコードに影響するかを調べるためにこのオプションの試用をお勧めします。

1.5.2 Newlib ライブラリー

Nios II SBT はプリコンパイルおよびソースバージョンの Newlib ライブラリー (C および数学) を含んでいます。しかしながら、プリコンパイルされた Newlib ライブラリーは FPH2 では推奨しません。

ハードウェア・コンフィグレーションと一致するように選択した個別の–mcustom-<operation>オプションを使用して、ソースコードから Newlib をコンパイルする必要があります。これにより、Newlib は GCC により推測できるすべての FPH2 演算の利点を組み込むことができます。Nios II ソフトウェア・ビルド・ツールを使用する場合、BSP ジェネレーターがこれを処理します。

`math.h` で定義された Newlib `isgreater()`、`isgreaterequal()`、`isless()`、`islessequal()`、および `islessgreater` マクロは通常の比較演算子 (< や >= など) を使用するため、これらのマクロは FPH2 比較演算子を自動的に使用します。

Newlib `fmaxf()`および `fminf()`関数は、これらの引数の最大値または最小値を返します。NaN 引数はデータが欠落しているとみなされます。1 つの引数が NaN でその他の数字である場合、関数は数字の値を返します。FPH2 fmaxs/fmins 演算はこの動作と一致します。

注意: 浮動小数点のハードウェア・コンフィグレーションを変更する場合は、Newlib が正しくビルドされるように BSP を再生成して再ビルドする必要があります。詳しくは、「Nios II SBT」を参照してください。

関連情報

- 20 ページの Nios II SBT
- 15 ページの -mcustom-<operation>
- 16 ページの pragma
- Newlib Documentation page
- GCC Floating-point Custom Instruction Support Overview
  GCC Floating-point Custom Instruction Support Overview についての詳細。

1.5.3 round()、fmins()、および fmaxs() での C マクロ

Nios II GCC は次の Newlib 浮動小数点関数への呼び出しを同等のカスタム命令に置き換えることができませんが、–mcustom-<operation>コマンドライン・オプションと pragma サポートがあります。

- `round()`
- `fmins()`
- `fmaxs()`

代わりに、これらのカスタム命令は GCC の __builtin_custom_* 機能を使用して直接呼び出される場合があります。system.h は直接カスタム命令を呼び出すために必要な#define マクロを提供します。Nios II ソフトウェア・ビルド・ツールはこのヘッダーファイルを C ソースファイルに自動的に含んでいます。ビルドイン関数について詳しくは、*Nios II Custom Instruction User Guide* を参照してください。

関連情報

- GCC Command Line Options
- Newlib Documentation page
- Nios II Custom Instruction User Guide

1.5.4 Nios II SBT

ソフトウェア・ビルド・ツール (SBT) は、Nios II 上で動作するエンベデッド・ソフトウェア用のアルテラの HAL ベースのボード・サポート・パッケージ (BSP) とアプリケーションおよびライブラリー makefile の作成に使用されるツールです。これらのツールは、コマンドラインと Eclipse GUI ベース形式で提供されます。

SBT についてより詳しい情報は、Nios II ソフトウェア開発のハンドブックのうちの 1 つを参照してください。

FPH2 コンポーネントで Nios II 用の BSP を生成して、これらのツールをその Nios II に接続すると、コンポーネントの sw.tcl ファイルは、BSP および存在する FPH2 を認識する BSP を使用するアプリケーションまたはライブラリーを引き起こします。特に、sw.tcl は次の機能を実行します。

- Qsys で作成したシステムを調べ、浮動小数点ハードウェアの正しい GCC フラグを決定する。
- -mcustom-<operation>オプションを GCC に渡す makefile ルールを作成するため、指定した浮動小数点演算を実装するためにソフトウェア・エミュレーション・コードの代わりに使用可能な FPH2 演算を使用することが把握できる。
- NaN 結果を検出するオーバーヘッドを排除し、sqrtf()呼び出し用の errno 変数を設定するために、-fno-math-errno オプションを GCC に渡す makefile ルールを作成する。
- GCC がカスタム命令で確実に置き換えられない Newlib 数学ライブラリー・ルーチンの #define マクロ宣言を system.h に追加する。詳しくは、「float()、fmins()、および fmaxs() での C マクロ」を参照すること。
- Newlib の正しいバージョンを生成するために makefile ルールを作成する。ハードウェア・システムで決定された GCC フラグを使用する。

注意: 浮動小数点ハードウェア構成を変更する場合は、Newlib が正しくビルドされるように BSP を再生成して再ビルドする必要があります。

関連情報

- 19 ページの round()、fmins()、および fmaxs() での C マクロ
- 12 ページの 演算
- Nios II Classic Software Developer's Handbook
- GCC Floating-point Custom Instruction Support Overview
  GCC Floating-point Custom Instruction Support Overview についての詳細。
- Nios II Gen2 Software Developer's Handbook

1.6 Nios II 浮動小数点ハードウェア 2 コンポーネント・ユーザーガイドの改訂履歴

日付	バージョン	変更内容
2016 年 5 月	2016.05.03	Enhanced Qsys Component Editor およびソフトウェア・ビルド・ツールは、浮動小数点関数の選択的な実装が可能。
2015 年 5 月	2015.05.22	初版