XAPP467(v1.1) 2003 年 5 月 13 日

Spartan-3 デバイスでのエンベデッド乗算器の使用

概要

Spartan-3 ファミリの専用 18x18 乗算器は、DSP 処理の高速化に役立ちます。この乗算器は、18 ビットまでの符号付き、または符号なしの乗算を行う際に、高速かつ効果的に処理します。基本的な乗算機能に加え、エンベデッド乗算器ブロックはシフタとして、あるいは絶対値や 2 の補数を生成するために使用できます。乗算器は、他の乗算器や CLB ロジックとともにカスケードし、より大規模で複雑な機能を作成できます。

はじめに

Spartan-3 シリーズには、デバイスの演算機能を高める多数の機能があります。前世代デバイスと同様に、キャリーロジックや専用のキャリー配線もあります。また CLB 内の専用 AND ゲートは、アレイ乗算を高速化します。追加された機能で最新かつ最も重要なものは、専用の 2 の補数の 18x18 乗算ブロックです。4 から 最大 104 個ある専用乗算器を使用することで、最小の汎用リソースで高速の演算機能をインプリメントできます。このようなパフォーマンスの向上に加え、専用乗算器は CLB ベースの乗算器と比較して、より少ない電力で動作します。

またエンベデッド乗算器により、符号付き 18x18 ビットの乗算の積を高速かつ効果的な方法で作成できます。乗算器ブロックは、ブロック SelectRAM メモリと配線リソースを共有し、多くのアプリケーションにより大きな効果を与えます。乗算器のカスケードが、ローカル Spartan-3 スライスに追加されたロジック リソースとともにインプリメントされます。

符号付き-符号付き、符号付き-符号なし、符号なし-符号なし乗算、論理シフタ、演算シフタ、バレル シフタ、2 の補数、および絶対値のようなアプリケーションは容易にインプリメントされます。

18x18 ビット乗算器は、CORE Generator を使用して瞬時に作成され、また VHDL あるいは Verilog を使用してインスタンシエート、または推論されます。

2 の補数の符号付き乗算器

データ フロー

各エンベデッド乗算ブロック (MULT18X18 プリミティブ) は、2 つの独立した 18 ビット符号付きまたは 17 ビット符号なしダイナミック データ入力ポートをサポートします。18 ビット符号付または 17 ビット符号なし 出力が積となる一方、2 つの入力は、被乗数、乗数、あるいは係数になります。MULT18X18 プリミティブを 図 1 に示します。

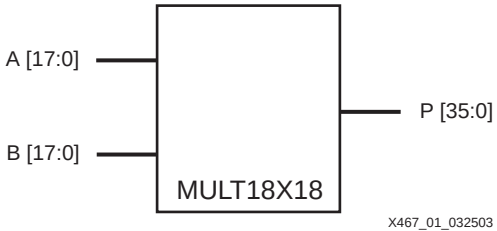


図 1: エンベデッド乗算器

さらにエンベデッド乗算器 4 個、36 ビット加算器 1 個、53 ビット加算器 1 個を使用し、35x35 ビットまでの符号付き乗算器を効果的にカスケードできます。図 6 を参照してください。

バイナリの乗算では、通常の乗算と同様に乗算器の各ビットで乗算した被乗数を使用して部分積を作成し、その部分積を加算して結果を出します。ザイリンクスの乗算器ブロックではブースアルゴリズムを使用しているため、部分積を作成するのにマルチプレクサを使用します。

タイミング仕様

MSB は、より多くの段階の加算が必要なため LSB の結果は MSB より早く出力されます。そのため 36 ビットの乗算出力はそのタイミングがそれぞれ変化します。したがって、設計時には最低必要な出力ビットで使用する必要があります。また、デザインには出来るだけ多くの出力ビットを使用する必要があります。例えば、2 つの符号なしの数値が 2^{35}、またはそれ以上の数値の積を持たない場合、P[35] の出力はかならず 0 になります。どのようなペアの n ビットの符号付き数値についても、$-2^{n-1} \times -2^{n-1}$ を持たない場合には、MSB は常に隣接する下位ビット (P[2n-1] = P[2n-2]) と同じになります。また、いくつかの出力に、より大きい配線遅延が必要な場合、MSB の遅延とのバランスをとるために、その遅延を LSB 出力に加える必要があることに注意してください。

同じ理由で、パイプライン乗算器のデータ入力セットアップ タイムも、LSB より MSB の方がより短くなりますが、タイミング パラメータは、ピンによりセットアップ タイムを区別しません。デザイン上で、さらに安全なゆとりを持たせるためには、MSB での入力をより遅くする必要があります。リセットおよびクロック イネーブル入力のセットアップ タイムは、どのデータ入力よりもはるかに高速であり、それらすべてのホールド タイムは 0 になっています。タイミング パラメータ名 "$t_{MULIDCK}$" (MULtiplier 入力 データから ClocK) は、データおよびコントロール入力の両方に使用されますが、それぞれのタイプにより異なる値になります。

ライブラリ プリミティブ

2 つのライブラリ プリミティブを、このエンベデッド乗算器では使用できます。表 1 に、これらのプリミティブを示します。

表 1: 乗算器プリミティブ

プリミティブ	A 幅	B 幅	P 幅	符号付き/符号なし	出力
MULT18X18	18	18	36	符号付 (2 の補数)	組み合わせ
MULT18X18S	18	18	36	符号付 (2 の補数)	レジスタ付き

レジスタ付きバージョンの乗算器では、クロック入力 C、アクティブ High クロック イネーブル CE、および同期リセット R が加算されます (図 2 を参照)。レジスタは、乗算器自体にインプリメントされ、他のリソースを必要としません。コントロール入力 C、CE、R は、すべて組み込み式のプログラマブル極性を持ちます。データ入力、クロック イネーブルおよびリセットは、すべてクロック エッジの前にセットアップ タイムを満たす必要があり、P 出力のデータはクロック - 出力間の遅延の後に変化します。

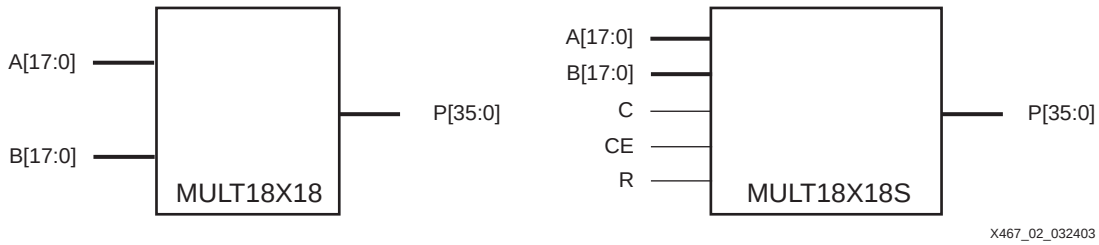


図 2: 組み合わせ、またはレジスタ付きの乗算器のプリミティブ

FPGA Editor のようなザイリンクス インプリメンテーション ツールで使用されるピン名は、ライブラリ プリミティブで使用されるピン名と同様です。

VHDL インスタンシエーション テンプレート

```
-- Component Declaration for MULT18X18 should be placed
-- after architecture statement but before begin keyword
component MULT18X18
  port ( P : out STD_LOGIC_VECTOR (35 downto 0);
         A : in STD_LOGIC_VECTOR (17 downto 0);
         B : in STD_LOGIC_VECTOR (17 downto 0));
end component;
-- Component Attribute specification for MULT18X18
-- should be placed after architecture declaration but
-- before the begin keyword
-- Attributes should be placed here
-- Component Instantiation for MULT18X18 should be placed
-- in architecture after the begin keyword
MULT18X18_INSTANCE_NAME : MULT18X18
  port map (P => user_P,
            A => user_A,
            B => user_B);
```

Verilog インスタンシエーション テンプレート

```
MULT18X18 MULT18X18_instance_name (.P (user_P),
                                   .A (user_A),
                                   .B (user_B));
```

MULT_STYLE 制約

MULT_STYLE 制約は、MULT18X18 プリミティブのインプリメンテーションを制御します。Project Navigator (図 3 を参照) では、ザイリンクス合成ツール (XST) が、インプリメンテーションの最適なタイプを選択するようにデフォルト設定されています。エンベデッド乗算器を使用するためには、MULT_STYLE = Block を設定、もしくは Project Navigator の［Multiplier Style］のプロパティで［Block］を選択します。MULT_STYLE 制約は、XST コマンド ラインで適用させるか、または MULT18X18 プリミティブに加えることでグローバルに設定できます。 MULT18X18S では、MULT_STYLE 制約を出力バスではなくコンポーネントに適用させます。詳細については、『制約ガイド』を参照してください。

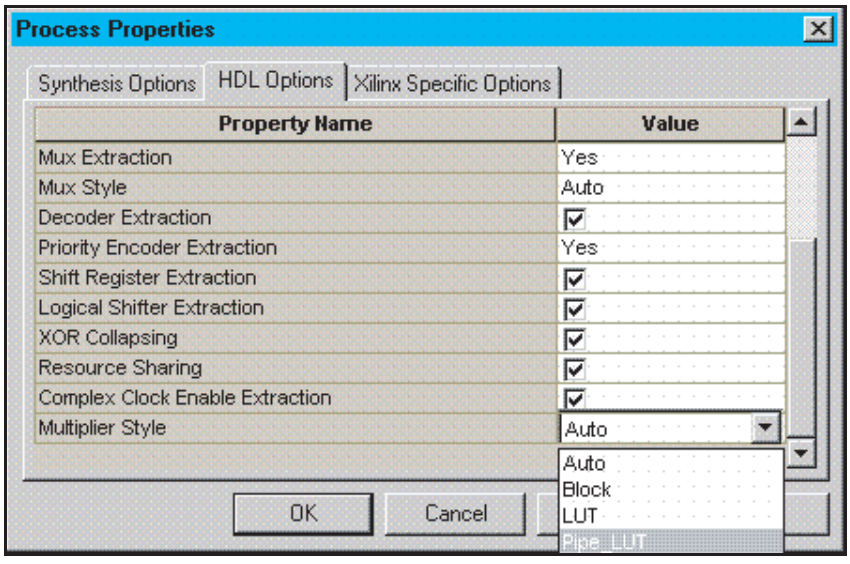
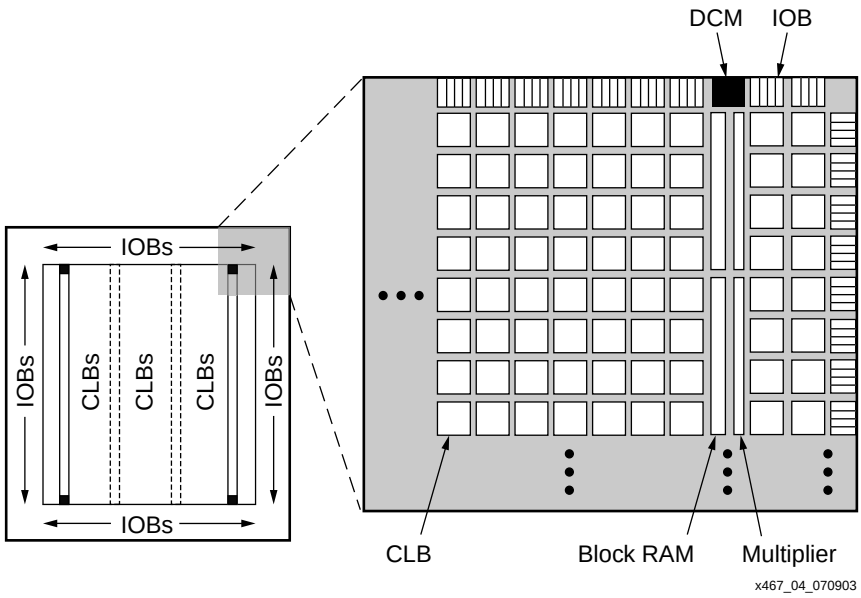


図 3: Project Navigator プロセス プロパティでの Multiplier Style の設定

Spartan-3 アーキテクチャの乗算器

乗算器はブロック RAM に隣接しているため、入力や結果をブロック メモリに簡単に保存できます (図 4 を参照)。各デバイスには、2 列あるいは 4 列の乗算器があります。2 列の場合には、列とエッジの間に 2 行の CLB があるため、乗算器は CLB または IOB のロジックで簡単に駆動できます。一方の乗算器ブロック上には 4 個の CLB（16 個のスライスまたは 32 個の LUT）があり、32 入力および出力信号は、隣接した乗算器ブロックにすぐに関連づけられます。高速な結果を得る配置として考えられる 1 つの方法は、片側に A[15:0] 、もう一方に B[15:0] を配置し、両側に P[31:0] 出力を分散して配置できます。フルサイズの 18x18 乗算器には、追加の入力および出力を隣接した CLB の列につなげます。パフォーマンスを最高な状態にするために、隣接した CLB のレジスタで入力をパイプライン化します。



メモ：

1. XC3S4000 および XC3S5000 デバイスの 2 つの追加ブロック RAM/乗算器の列は、点線で表示しています。XC3S50 デバイスには、その左端に沿って 1 列のブロック RAM/乗算器があります。

図 4: Spartan-3 アーキテクチャでの乗算器の配置

Spartan-3 の乗算器の 18 ビット幅は通常あまり使用されませんが、パリティビット付きブロック RAM の 18 ビット幅と一致します。通常よく使用される 8 ビットあるいは 16 ビット乗算器は、乗算器ブロックの一部を使用して作成でき、また 32 ビット乗算器は、乗算器ブロックをカスケードすることで作成できます。ザイリンクスのアーキテクチャでは、標準で使用されることの多いビット幅以外でもインプリメントでき、アプリケーションの要求に的確に合わせることができます。未使用の乗算器の入力は、0 に設定された未使用の LUT に自動的に接続されることで 0 になります。

表 2: Spartan-3 デバイスごとの乗算器の数

デバイス	乗算器の行	乗算器
XC3S50	1	4
XC3S200	2	12
XC3S400	2	16
XC3S1000	2	24
XC3S1500	2	32
XC3S2000	2	40
XC3S4000	4	96
XC3S5000	4	104

機能拡張された乗算器

18 ビットを超える入力値での乗算は、乗算プロセスをより小さいサブプロセスに分割することが可能です。どちらの入力のバイナリ表現も正しい重み付けとMSB の符号を考慮すれば、どのポイントでも分岐させることができます。入力を MSB から 18 ビットで分岐させることで、18 ビットの符号付き乗算器を最大限使用できます。

例えば、図 5 は、22x16 の乗算器のインプリメント例を示します。22 ビットの値は、18 ビット符号付きの値とLSB からの 4 ビット符号なしの値に分割されます。2 つの部分積加算器が形成されます。1 つは、16 ビットの符号付き値を 4 ビットの符号なしセクションで乗算した結果の 20 ビット符号付きの積です。もう 1 つは、16 ビットの符号付き値を 18 ビットの符号付きセクションで乗算した結果の 34 ビット符号付きの積です。また加算のプロセスによって重み付けが復元され（1 つ目の乗算結果の下位ビットは加算器をバイパスすることに注意）、38 ビットの結果が形成されます。最初の乗算器は符号付きであるため、20 ビットの値は加算の前に符号を拡張する必要があります。加算器自体は、34 ビットであるため 17 スライスで実現できます。

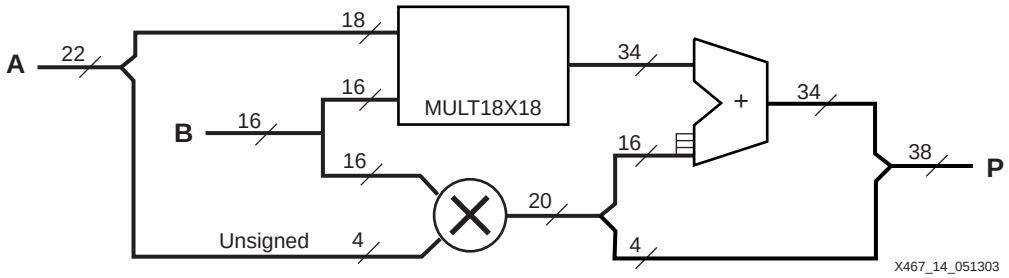


図 5: 22x16 乗算器のインプリメンテーション

インプリメンテーションは、どの程度のパフォーマンスが必要か、どのくらいのリソースが使用できるかによって変更できます。2 つめの乗算器が小さい場合には、MULT18X18 リソースまたは CLB を使用してインプリメントできます。エンベデッド乗算器に組み込まれた機能でパイプライン化することにで、パフォーマンスを向上させることができます。両方の入力が 18 ビットを超える場合、4 つの部分積が形成されますが、LSB からの完全な符号なしの結果は、単に MSB の 36 ビット符号付き加算器にまとめられ、他の 2 つの結果に追加されます。

図 6 は、35x35 ビットの符号付き乗算器を 4 つのエンベデッド乗算器と 2 つの加算器を使用してインプリメントされた回路図を示しています。

使用する加算器は、53 ビット幅です (下位 17 ビットは、一方の入力を必ず 0 になります)。

34x34 ビット符号なしサブモジュールでは、それぞれのオペランドの最上位ビットを Low に固定するだけで同様の手順で作成できます。

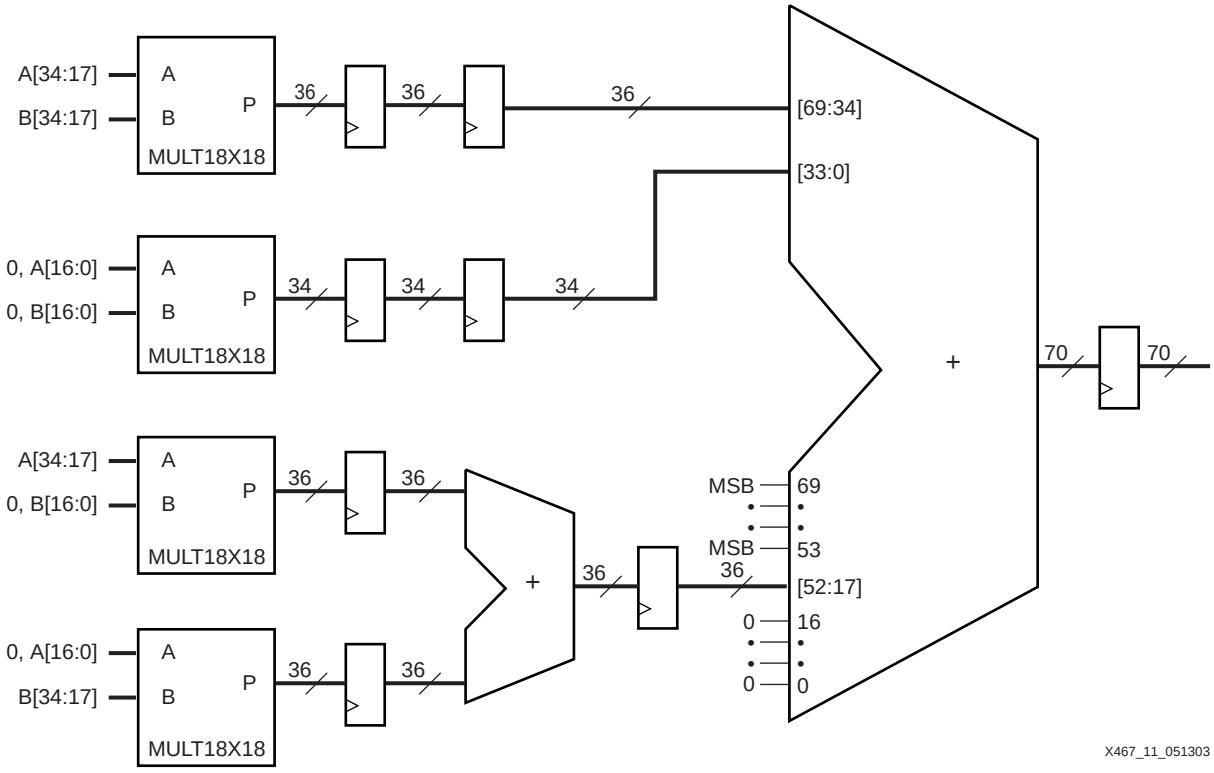


図 6: 35x35 符号付き乗算器

シングル プリミティブの中の 2 の乗算器

エンベデッド乗算器は、より小さな値を 2 個同時に乗算するために使用できます。その結果が出力で互いに重ならないように、LSB 側と MSB 側に 1 つずつ値を置くことで 2 つの独立した結果を得ることができます。MSB へ n 位置の値の 1 つをシフトすることは、2^n で乗算することと同様です。MSB にシフトした値が X の場合、新しい値は $X * 2^n$ になります。LSB の値が Y の場合、完了した乗算器の入力は $X * 2^n + Y$ になります。

簡素化した例として、同じ MULT18X18 プリミティブに 2 平方の値がインプリメントされているとみなします。次の論理式は、乗算の形式を示しています。

プリミティブごとの 2 の乗算器

$$(X * 2^n + Y)(X * 2^n + Y) = (X^2 * 2^{2n}) + (XY * 2^{n+1}) + (Y^2)$$

X または Y が 0 の場合、論理式は次のようになります。

$X^2 * 2^{2n}$ {Y=0} (X^2 on the output MSBs)

Y^2 {X=0} (Y^2 on the output LSBs)

0 {X=0, Y=0}

X と Y の値が 0 以外の場合、MSB および LSB 上の結果と中間の条項 $(XY * 2^{n+1})$ が重複するのを避けるよう注意が必要です。X および Y が 0 ではなく、次の不等式が成り立つ場合には、2 個の乗算器で、1 個の MULT18X18 のプリミティブを共用できます。

プリミティブごとの 2 の乗算器の不等式の条件

$$(X^2 * 2^{2n})_{min} > (XY * 2^{n+1})_{max}, (XY * 2^{n+1})_{min} > (Y^2)_{max}$$

表 3 は、上記の条件下での X と Y の値を示します。

表 3: 許容サイズの MULT18X18 ごとの 2 の乗算器

X * X		Y * Y	
符号付きサイズ	符号なしサイズ	符号付きサイズ	符号なしサイズ
7 X 7	6 X 6	-	4 X 4
6 X 6	5 X 5	-	5 X 5
5 X 5	4 X 4	3 X 3	6 X 6
4 X 4	3 X 3	3 X 3	7 X 7
3 X 3	2 X 2	4 X 4	8 X 8

図 7 は、6 ビットの符号付き値と 5 ビットの符号なしの値の算出についての MULT18X18 の連結を示します。

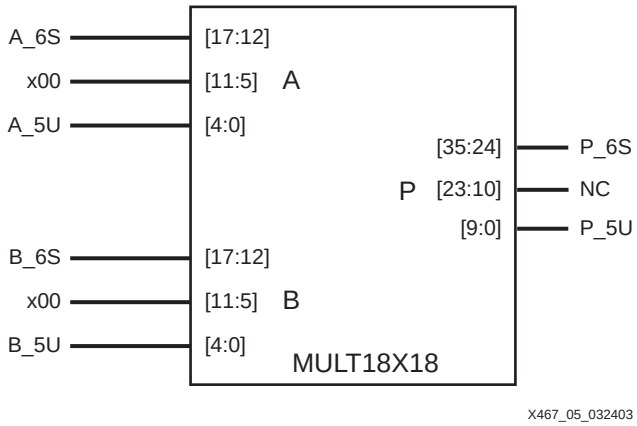


図 7: 1 プリミティブ内の 2 つの乗算器

デザイン入力

Spartan-3 の乗算器をデザインの中で使用するのには、多数のオプションがあります。先に記述したライブラリ プリミティブ MULT18X18 と MULT18X18S は、回路図または HDL コード内でインスタンシエートできます。Xilinx XST、Synplicity の Synplify、また Mentor の LeonardoSpectrum などの合成ツールは、乗算器ブロックを乗算演算子から推論できます。同期乗算器の演算がクロックで制御されている場合には、MULT18X18S を推測します。

LeonardoSpectrum では何段かのレジスタをロジックに挿入することでパイプライン化された乗算器を作成しますが、この場合はエンベデッド乗算器の代用として CLB リソースを使用します。パイプライン化された乗算器の機能を有効にするには、RTL ソース コードの中である特定の標記が必要になります。詳細については、『Synthesis and Simulation Design Guide』を参照してください。

次の VHDL の例では、XST または Synplify を使用して MULT18X18S を推測します。

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity mult18x18s is
  port ( a : in std_logic_vector(7 downto 0);
         b : in std_logic_vector(7 downto 0);
         CLK: in std_logic;
         prod : out std_logic_vector(15 downto 0));
end mult18x18s;
architecture arch_mult18x18s of
```

```
  mult18x18s is
begin
process(clk) is begin
  if clk'event and clk = '1' then
    prod <= a*b;
  end if;
end process;
end arch_mult18x18s;
```

LeonardoSpectrum 用にコード記述された同期乗算器の VHDL 例

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity mult18x18s is
  port(  clk: in std_logic;
         a: in std_logic_vector(7 downto 0);
         b: in std_logic_vector(7 downto 0);
         prod: out std_logic_vector(15 downto 0));
end mult18x18s;
architecture arch_mult18x18s of
  mult18x18s is
signal reg_prod : std_logic_vector(15 downto 0);
begin
process(clk)
begin
  if(rising_edge(clk))then
    reg_prod <= a * b;
    prod <= reg_prod;
  end if;
end process;
end arch_mult18x18s;
```

Synplify と XST 用にコード記述された同期乗算器の VHDL 例

```
module mult18x18s(a,b,clk,prod);
  input [7:0] a;
  input [7:0] b;
  input clk;
  output [15:0] prod;
  reg [15:0] prod;
  always @(posedge clk) prod <= a*b;
endmodule
```

LeonardoSpectrum 用にコード記述された同期乗算器の Verilog 例

```
module mult18x18s (a,b,clk,prod);
  input [7:0] a;
  input [7:0] b;
  input clk;
  output [15:0] prod;
  reg [15:0] reg_prod, prod;
  always @(posedge clk) begin
  reg_prod <= a*b;
  prod <= reg_prod;
endmodule
```

CORE Generator システムを使用する場合

エンベデッド Spartan-3 の 18x18 ビット 2 の補数の乗算器を使用した乗算器は、CORE Generator v6.0 の Multiplier Module を使用して容易に生成できます。このコアは、CORE Generator システムの 5.1i 以降のバージョンで使用できます。Multiplier Generator には、次のような機能があります。

- エンベデッド乗算器ブロックを使用して、並列乗算器を生成します。
  - 並列乗算器には他のリソースが使用でき、シーケンシャル / シリアル シーケンシャル、固定 / 再読込み可能な係数固定の乗算器を生成できます。
- 符号付き 2 の補数/符号なしモードをサポートします。
- 1 ～ 64 ビット幅までの入力をサポートします。
- 1 ～ 129 ビット幅までの出力をサポートします。
- 組み合わせ、あるいは完全にパイプライン化されたインプリメンテーションを生成します。
- オプションで、クロック イネーブル付きレジスタ出力、および同期/非同期クリアを使用できます。
- オプションでいくつかのハンドシェイク信号を使用できます。

図 8 は、Core Multiplier Generator のロジック シンボルを示します。RFD (Ready For Data) 出力は、High になり、乗算器がデータを受け取れる状態であることを示します。ND (新しいデータ) 入力は、乗算器の入力で新しいデータが有効であることを表示するためアサートされます。RDY (Ready) 信号は、その出力が現在の積であることを示します。LOADB と SWAPB は、係数固定の乗算器で使用されます。

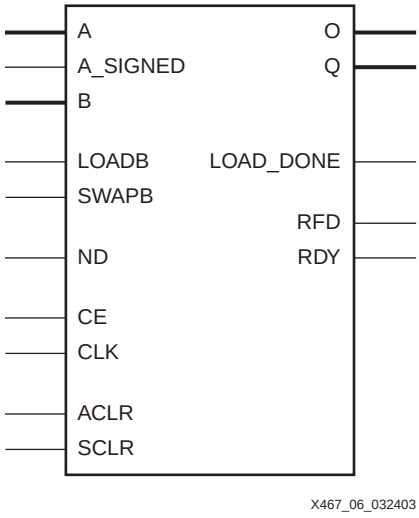


図 8: Core Multiplier Generator のシンボル

CORE Generator システムでは、デフォルトのパラレル乗算器タイプとして、エンベデッド乗算器を使用します。乗算器の生成オプションでは、代わりにこの機能を LUT を使用してインプリメントすることを選択できます。

図 9 は、Multiplier Generator のタイミング図を示します。

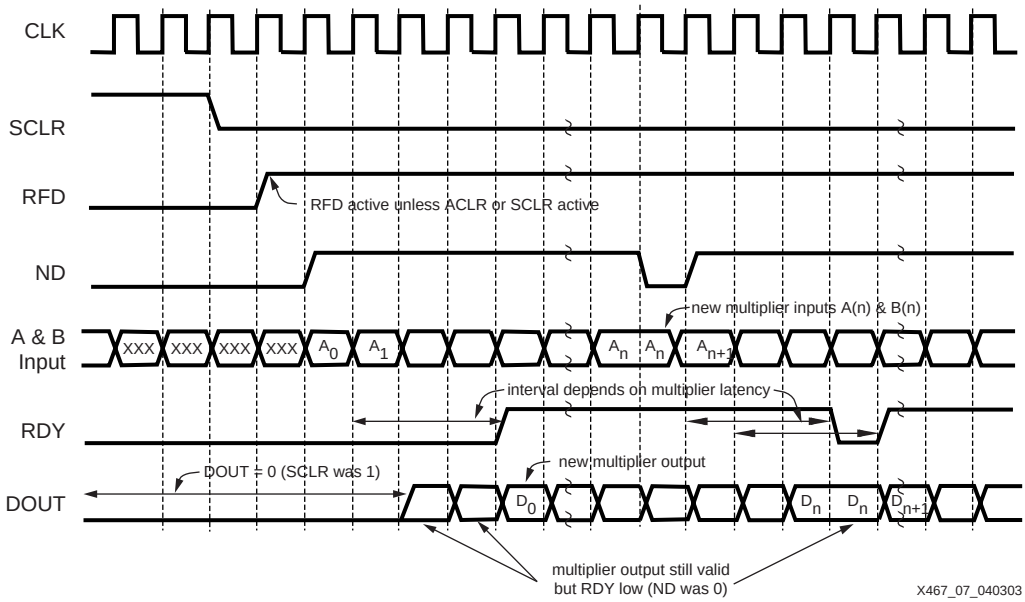


図 9: Multiplier Generator タイミング図

System Generator

Multiplier Generator は、MULT ブロックが使用された場合 DSP の System Generator で使用されます。System Generator では、上位で抽象的なデザイン設計できますが、シリコンの主な機能に関連付けることでパフォーマンスの高い FPGA インプリメンテーションも可能です。System Generator では、MATLAB® M コードを合成可能な HDL コードにコンパイルするためのブロックも提供されます。System Generator は、GUI でパラレルが選択されているときと専用の乗算器の使用にチェックがされている場合には、エンベデッド乗算器を使用します。

MAC Cores

CORE Generator システムと System Generator は、乗算器を building block として使用した複雑な機能をインプリメントできます。Multiply Accumulator (MAC) コアは、32 ビットまでの入力とオプションでユーザー指定のパプライン化をサポートします。エンベデッドまたは LUT ベースのインンプリメンテーションというオプションは、エンベデッド乗算器または CLB リソースを使用して作成するかを選択できます。MAC インプリメンテーションは、指定の乗算器以外の CLB リソースはほとんど使用しないため、最も低集積度で低コストなデザインの実現も可能になります。

MAC および MAC ベースの FIR フィルタでは、システム クロック パフォーマンスに合わせて自動的にパイプライン化する機能があります。パイプラインの段数は、スピード/エリアの最適なトレードオフを考慮し、デザインに応じて自動的に追加されます。

乗算器サブモジュール

このセクションでは、Spartan-3 で使用できるサブモジュール例について説明します。表 4 は、乗算器と 1 個のレジスタのない MULT18X18 プリミティブを使用し、2 の補数のリターン機能について表記しています。

表 4: 1 つの MULT18X18 をサブモジュールするエンベデッド乗算器

サブモジュール	A 幅	B 幅	P 幅	符号付き/符号なし
MULT17X17_U	17	17	34	符号なし
MULT8X8_S	8	8	16	符号付
MULT8X8_U	8	8	16	符号なし
MULT4X4_S	4	4	8	符号付
MULT4X4_U	4	4	8	符号なし
TWOS_CMP18	18	-	18	-
TWOS_CMP9	9	-	9	-
MAGNTD_18	18	-	17	-

図 10 および 図 11 は、4x4 ビットの符号付き乗算器と 4x4 ビットの符号なし乗算器のインプリメンテーションをそれぞれ示します。

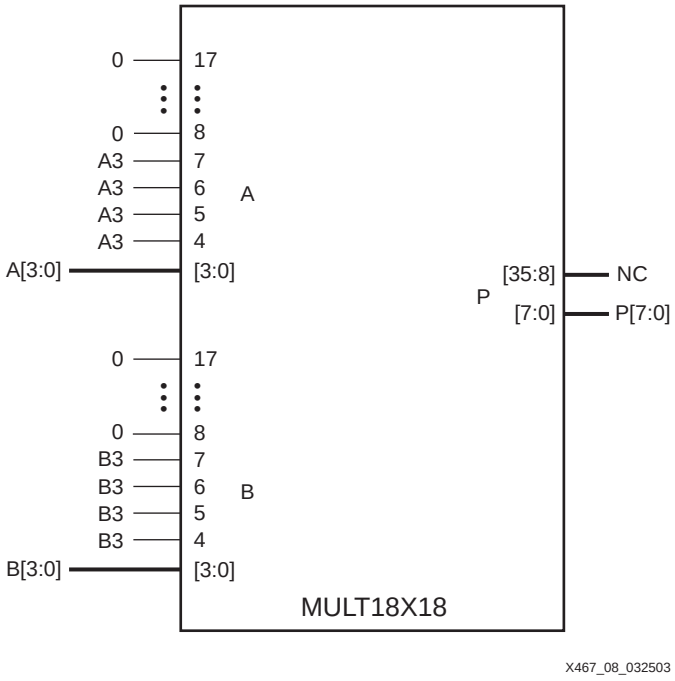


図 10: MULT4X4_S サブモジュール

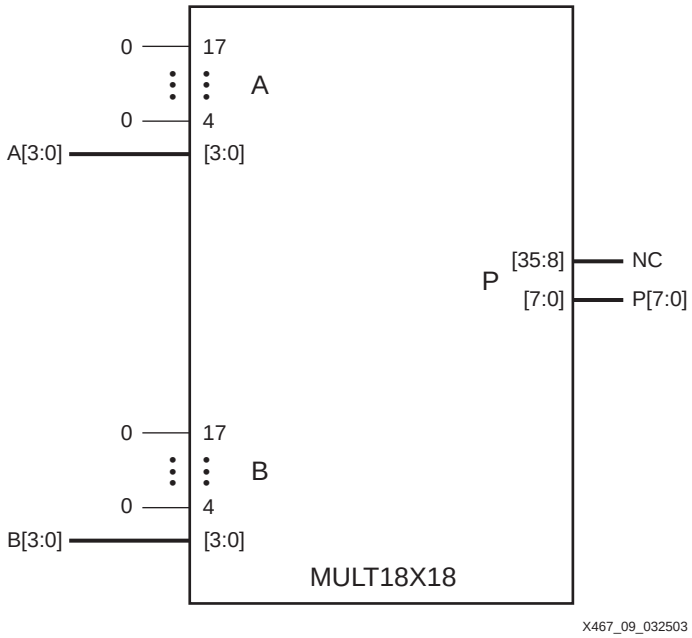


図 11: MULT4X4_U サブモジュール

サブモジュール MAGNTD_18 は、2 の補数の絶対値を返します。負の数を入力した時は正の数で出力され、正の数を入力した場合にはそのままの値を出力します。サブモジュール TWOS_CMP18 と TWOS_CMP9 により、2 の補数のリターン機能を実現します。追加のスライス ロジックは、これらのサブモジュールとともに使用し、符号付き絶対値を 2 の補数あるいはその逆に効果的に変換することができます。

図 12 は、サブモジュール TWOS_CMP9 を作成する MULT18X18 への連結を示します。

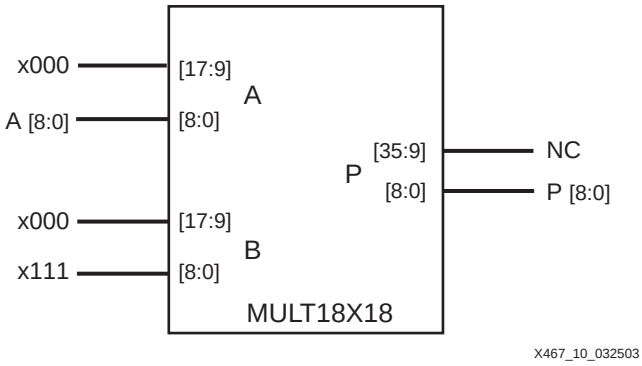


図 12: TWOS_CMP9 サブモジュール

VHDL および Verilog のインスタンシエーション

VHDL と Verilog のインスタンシエーションのテンプレートは、プリミティブとサブモジュールの例として使用できます (VHDL および Verilog のテンプレート, 13ページを参照)。

VHDL には、各テンプレートにコンポーネント宣言とアーキテクチャ セクションが含まれます。VHDL デザイン ファイルにテンプレートの各パートを挿入します。またアーキテクチャ セクションのポートマップは、デザインの信号名にします。

ポート信号

Data In − A

データ入力 A には 18 ビットまでの新しいデータを提供し、乗算のオペランドの 1 つとして使用されます。

Data In − B

データ入力 B には 18 ビットまでの新しいデータを提供し、乗算のオペランドの 1 つとして使用されます。

Data Out − P

データ出力バスの P は、オペランド A および B 用の 36 ビットまでの 2 の補数の乗算のデータ値を示します。

配置制約

エンベデッド乗算器インスタンスの MULT18X18 に LOC プロパティを適用し、配置制約をかけることができます。MULT18X18 の配置ロケーションを CLB 配置の名称とは異なる表記規則にすることにより、LOC プロパティを容易にアレイからアレイに移動できるようにします。

LOC プロパティでは、次のフォームが使用されます。

LOC = MULT18X18_X#Y#

例) MULT18X18_X0Y0 は、デバイスの左下に配置された MULT18X18 です。

VHDL および Verilog のテンプレート

VHDL および Verilog テンプレートは、プリミティブとサブモジュールに使用できます。

次は、プリミティブのテンプレートです。

- SIGNED_MULT_18X18 (primitive: MULT18X18)

次に示すのは、サブモジュールのテンプレートです。

- UNSIGNED_MULT_17X17 (submodule: MULT17X17_U)
- SIGNED_MULT_8X8 (submodule: MULT8X8_S)
- UNSIGNED_MULT_8X8 (submodule: MULT8X8_U)
- SIGNED_MULT_4X4 (submodule: MULT4X4_S)
- UNSIGNED_MULT_4X4 (submodule: MULT4X4_U)
- TWOS_COMPLEMENTER_18BIT (submodule: TWOS_CMP18)
- TWOS_COMPLEMENTER_9BIT (submodule: TWOS_CMP9)
- MAGNITUDE_18BIT (submodule: MAGNTD_18)

対応するサブモジュールは、デザインに合成する必要があります。

SIGNED_MULT_18X18 モジュールのテンプレートは、VHDL と Verilog コードでその例が示されます。

VHDL のテンプレート

```
-- Module: SIGNED_MULT_18X18
-- Description: VHDL instantiation template
-- 18-bit X 18-bit embedded signed multiplier (asynchronous)
--
-- Device: Spartan-3 Family
----------------------------------------------------------------------
-- Components Declarations:
component MULT18X18
  port(
       A :  in std_logic_vector (17 downto 0);
       B :  in std_logic_vector (17 downto 0);
       P : out std_logic_vector (35 downto 0)
  );
end component;
--
-- Architecture  Section:
--
U_MULT18X18 : MULT18X18
  port map (
    A => , -- insert input signal #1
    B => , -- insert input signal #2
    P =>   -- insert output signal
  );
```

Verilog のテンプレート

```
// Module: SIGNED_MULT_18X18
// Description: Verilog instantiation template
// 18-bit X 18-bit embedded signed multiplier (asynchronous)
//
// Device: Spartan-3 Family
//----------------------------------------------------------------------
// Instantiation Section
//
MULT18X18 U_MULT18X18
  (
    .A () , // insert input signal #1
    .B () , // insert input signal #2
    .P ()   // insert output signal
  );
```

乗算器の可変アプリケーション

2^n のバイナリ乗算とは、n の数値分をシフトさせることであるため、乗算器は、シフタあるいは他の汎用のリソースとして使用できます。つまり、アプリケーションが使用可能な乗算器を多数必要としない場合に、上記のようにみなすことができます。

シフタ

乗算器は、シフタとして使用できます。一方のオペランドが 2 の累乗 (2^n) の場合、もう一方のオペランドは n 数値分シフトされ、出力に配線されます。シフトを制御するために符号付きビット (MSB) を使用できないため、2 の補数付き18x18 ビットの乗算器は 0 から 16 のビット位置でシフトできます。

36 出力のラインに、シフトされたデータラインより下位のものについては自動的に 0 が入力され、シフトされたデータよりも上位のものについては、MSB 入力の状態で 0 または 1 が入力されます。これは、2 の補数で乗算するのと同様の結果です。

ユーザーは、MSB 入力を Low にすることにより 17 入力ビットのロジック シフトの実行、あるいは MSB の符号を効果的に拡張して 18 ビットの 2 の補数の値の演算シフトを実行できます。

従来の CLB ベースのシフタは、各 n 入力では、n 個のマルチプレクサのアレイを使用しているため、多数の配線リソースを必要とします。18 ビットを超える乗算器ベースのシフタや任意の長さのバレル シフタは、出力の外部に OR ゲートを必要としますが、CLB リソースは少なくてすみます。

絶対値の戻り

乗算により絶対値を生成するには、正の数の場合 1 で乗算し (MSB は 0)、負の場合に -1 で乗算します (MSB は 1)。2 の補数の表記では、1 を表現する場合、LSB 以外はすべて 0 に、-1 の場合には LSB も含めてすべてのビットが 1 になります。したがって、絶対値を生成するには、LSB に 1 をかけるか入力値の値の MSB を他のすべてのビットにかけることで実現できます。図 13 は、絶対値の Generator を示します。

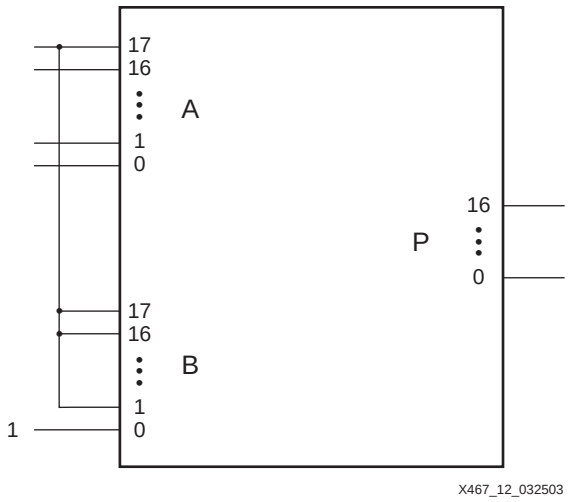


図 13: 絶対値の戻り

2 の補数の戻り値

2 の補数の生成には、通常 1 ビットごとに 1 個の LUT を使用します。また、多ビットになる場合にはキャリーロジックを使用します。したがって LUT を多く使用する場合には、乗算器を使用して入力に対し 2 の補数を戻すことができます。入力された数をすべての同じビット幅の数でそれぞれ乗算すると、出力ビットのビット幅よりも大きい 2 の補数の数が生成されます。関係のない高位ビットは無視されます。図 14 は、2 の補数値の生成器を示しています。

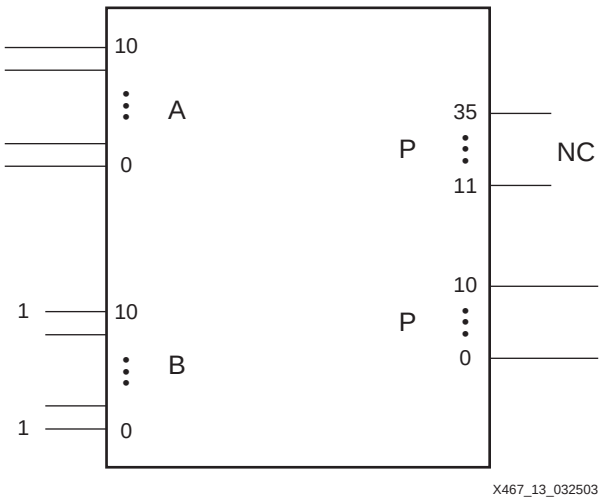


図 14: 2 の補数の戻り値

複素数を含む乗算

複雑な乗算は、-1 の平方根と同じユニット i の付いた実数と虚数コンポーネントを含む複雑な数の乗算です。複雑な乗算は、次の 3 つの実数の乗算のみを使用して実行できます : ac、bd、および (a + b)(c + d)。(a + ib)(c + id) の実数部は、ac - bd になり、虚部は (a + b)(c + d) - ac - bd になります。Spartan-3 アーキテクチャには、多数の乗算器があり、複雑な乗算であっても簡単にします。

行列演算で見る時分割処理について

CG やビデオ分野のパイプライン化された多数の機能は、matrix 演算で示されています。A 3x3 matrix 乗算には、3x3 matrix の結果を出すのに、27 の乗算と 18 の加算が必要になります。定数での 3 x 3 matrix の乗算については、カラー変換で表示できます。この乗算では、9 個の乗算と 6 個の加算によって 3 つの結果が生成されます。

Spartan-3 デバイスの高速な機能により、ユーザーが乗算器を使い回しできるようになります。9 個の乗算器の代わりに、デザインはシステムのクロック レートの 9 倍の結果である 9 セットの入力を供給し、乗算器の数を 1 個に減らします。加算器ロジックは CLB リソースでインプリメントされ、加算器の出力は、3 番目のクロックごとに 3 つの結果が出力レジスタに保存されます。詳細については XAPP284 を参照してください。

浮動小数点の乗算

小数点の乗算に使用される数と符号付きビットに指数の浮動小数点が加えられます。 32 ビット 浮動小数点の乗算器は、専用乗算器ブロック 4 個と CLB リソースを使用してインプリメントできます。これらの乗算器は Xilinx AllianceCORE™ パートナーから入手できます。

関連文献および情報

- Spartan-3 ファミリ データシート
  アーキテクチャの詳細とタイミング パラメータ
  DS099-1, ***Spartan-3 1.2V FPGA Family:*** Introduction and Ordering Information (Module 1)
  DS099-2, ***Spartan-3 1.2V FPGA Family:*** Functional Description (Module 2)
  DS099-3, ***Spartan-3 1.2V FPGA Family:*** DC and Switching Characteristics (Module 3)
  DS099-4, ***Spartan-3 1.2V FPGA Family:*** Pinout Tables (Module 4)
- DSP セントラル (http://www.xilinx.co.jp/xil_prodcat_landingpage.jsp?title=Xilinx+DSP)
  DSP ソリューションで最大の効果を上げるための情報が参照できます。
- IP センタ (http://www.xilinx.co.jp/ipcenter)
  ザイリンクスおよびアライアンス パートナーのコア ソリューションについて参照できます。
- ソフトウェア マニュアル (http://support.xilinx.co.jp/support/sw_manuals/xilinx5/download/)
  ライブラリ ガイド MULT18X18/S の詳細、合成とシミュレーション デザイン ガイドのインスタンシエーション例について参照いただけます。
- XAPP284 Matrix Math, Graphics, and Video
  9 倍のクロックレートで駆動する乗算器を使用して、1 クロック周期で、3x3 マトリックス乗算の 9 個の結果を出力します。
- XAPP636 Optimal Pipelining of the I/O Ports of Virtex-II Multipliers
  パイプライン化された入力と出力により発生する専用乗算器の高速、最適なインプリメンテーション、および効果的な配置配線の制約について説明します。
- TechXclusives (http://www.xilinx.co.jp/support/techxclusives/techX-home.htm)
- 『Using Leftover Multipliers and Block RAM』（Peter Alfke 著）また 『Expanding Virtex-II Multipliers』（Ken Chapman 著）を参照してください。

おわりに

FPGA には、ロジックを特定のアプリケーションにカスタマイズできるということで、汎用の DSP チップに比べ大きな利点があります。いくつかの機能は、それまでよりも大幅に少ないリソースで 100 倍以上速いスピードで駆動させることができます。活用できる重要な機能として、エンベデッド乗算器ブロックがあります。乗算ロジックの自動最適化を活用すれば、必要となる正確な結果を得ることができます。CORE Generator システムでより単純な乗算器を生成でき、またこれらの乗算器を組み合わせることで MAC のような、より複雑な機能にすることができます。

付録 A: 2 の補数の乗算

2 の補数の表示により、符号付き整数でバイナリ演算操作、正確な 2 の補数結果の生成が可能になります。2 の補数の正の値は、単純なバイナリで表されます。2 の補数の負の値は、同じ絶対値 0 を持つ正の数が加えられた際にはバイナリ数値で表されます。2 の補数を整数で計算するには、すべての 0 の値を 1 に、1 の値を 0 に反転させ (1 の補数ともいう) 、最後に 1 を足します。1 番左にある MSB ビットは整数を示しており、符号付きビットとも呼ばれます。符号付きビットが 0 の場合には、数は正になります。符号付きビットが 1 の場合には、数は負になります。符号付き整数をより大きな幅に拡張するには、数の左側で MSB を重複させます。

2 の補数の乗算は、バイナリ乗算と同じ AND ゲートの真と同様のルールに従って実行されます。

0 x 0 = 0

0 x 1 = 0

1 x 0 = 0

1 x 1 = 1、キャリーまたはボロー ビットはなし

例)

1111 1100 = -4

0000 0100 = +4

1111 0000 = -16

改訂履歴

次の表に、この文書の改訂履歴を示します。

日付	バージョン	改訂内容
04/06/03	1.0	初版リリース
05/13/03	1.1	Spartan-3 アーキテクチャの乗算器 セクションで、デバイス XC3A50 の乗算器の情報を更新 新しいセクション 機能拡張された乗算器を追加 関連文献および情報セクションに、TechXclusive についての記述の追加 その他、微調整