XILINX®

XAPP463 (v1.1.2) 2003 年 7 月 23 日

Spartan-3 FPGA でのブロック RAM の使用

概要

Spartan-3™ FPGA には、チップ上で大規模なメモリを必要とするアプリケーションで使用できる効率のよい SelectRAM™ メモリ ブロックが多数搭載されています。さまざまなコンフィギュレーション オプションを使用することで、SelectRAM は、多様なデータ幅とワード数の RAM、ROM、FIFO、大きな LUT、データ幅コンバータ、循環バッファおよびシフト レジスタとして使用できます。 このアプリケーション ノートでは、ブロック SelectRAM の特長と機能について説明し、Xilinx CORE Generator システムを使用して、または VHDL や Verilog インスタンシエーションによって、ブロック RAM のオプションを指定する方法について明らかにします。さらに、ツール、アプリケーション ノートなどを参考に、ブロック RAM を使用できるさまざまなアプリケーションについても説明します。

はじめに

すべての Spartan-3 デバイスには、縦に並んだ複数のブロック RAM メモリあります。表 1 に示すように、ブロック RAM メモリの総数は Spartan-3 デバイスの大きさによって異なります。

表 1: Spartan-3 デバイスのブロック RAM

Spartan-3 デバイス	RAM 列数	列ごとのブロック RAM 数	総ブロック RAM 数	RAM の総ビット	RAM の総 K ビット
XC3S50	1	4	4	73,728	72K
XC3S200	2	6	12	221,184	216K
XC3S400	2	8	16	294,912	288K
XC3S1000	2	12	24	442,368	432K
XC3S1500	2	16	32	589,824	576K
XC3S2000	2	20	40	737,280	720K
XC3S4000	4	24	96	1,769,472	1,728K
XC3S5000	4	26	104	1,916,928	1,872K

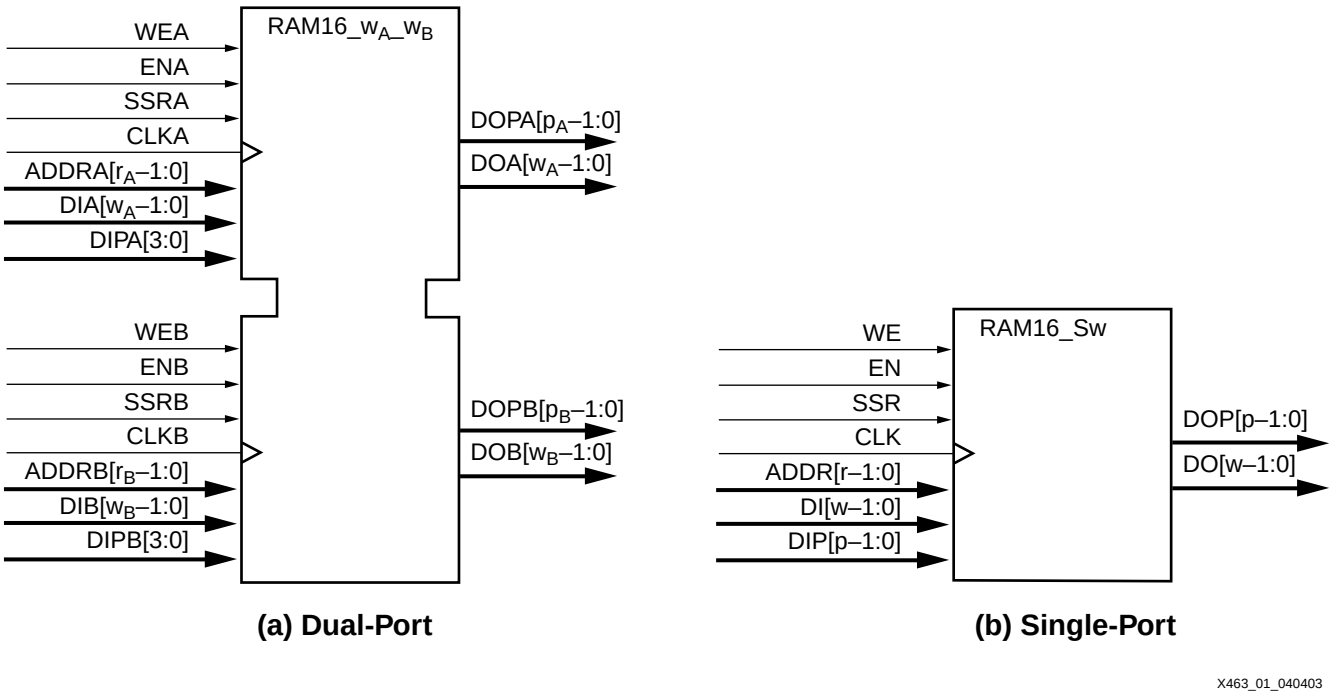
メモ：

1. 1K ビット = 1,024 ビット (メモリ仕様)

各ブロック RAM は、18,432 ビットの高速スタティック RAM で構成され、メモリ コンフィギュレーションによって、16K ビットはデータ領域に割り当てられます。また、残りの 2K ビットはパリティ ビットあるいは追加のデータ ビットとして使用できます。ブロック RAM メモリには、完全に独立した 2 つのアクセス ポート (ポート A およびポート B) があり、その構造は対称で、2 つのポートは交換可能です。また、両ポートでデータの書き込み/読み出しを行うことができます。 各メモリ ポートは、それぞれのクロック、クロック イネーブル、およびライト イネーブルに同期します。読み出しも同期で実行されるため、クロック エッジとクロック イネーブルが必要です。

ブロック RAM は物理的にデュアル ポート メモリですが、図 1 に示すように、1 つのアプリケーションではシングル ポート メモリとしても使用できます。さらに、各ブロック メモリはいくつかのアスペクト比にコンフィギュレーションできます。表 2 に SelectRAM の特長を示します。

また、複数のブロック RAM をカスケード接続することで、ビット数およびワード数の多いメモリを作成し、特別な配線リソースを使用してタイミング遅延を最小限にすることができます。



メモ：

1. w_A および w_B は、ポート A、ポート B それぞれでの総データ パス幅 (パリティ ビットを加えたデータ ビット)を示す整数
2. p_A および p_B は、パリティ ビットとして機能するデータ パス幅を示す整数
3. r_A および r_B は、ポート A とポート B それぞれのアドレス バス幅を示す整数
4. 両ポートにある制御信号 CLK、WE、EN および SSR は、極性を反転できます。

図 1: デュアル ポート (a) およびシングル ポート (b) の SelectRAM 18K ブロック

表 2: SelectRAM 18K ブロック メモリ特性およびアプリケーション

パリティを含む RAM の総ビット数	18,432 (16K データ + 2K パリティ)
メモリ構造	16Kx1 8Kx2 4Kx4 2Kx8 (パリティなし) 2Kx9 (x8 + パリティ) 1Kx16 (パリティなし) 1Kx18 (x16 + 2 パリティ) 512x32 (パリティなし) 512x36 (x32 + 4 パリティ) 256x72 (シングル - ポートのみ)
パリティ	データ幅が、1 バイトより大きい場合のみオプションとして使用でき、データ ビットとしても使用可能
パフォーマンス	200 MHz (概算)

表 2: SelectRAM 18K ブロック メモリ特性およびアプリケーション *(Continued)*

タイミング インターフェイス	単純な同期インターフェイス。書き込みにはセットアップ時間、読み出しには clock-to-output 遅延があり、レジスタからの読み出し/書き込みと同様。
シングル ポート	可能
True デュアル ポート	可能
ROM、初期値を持つ RAM	可能
複数のデータ ポート幅設定	可能
電源投入時の状態	ユーザー定義データ (デフォルトではゼロ)
可能なアプリケーション	ローカル データ ストレージ、FIFO、エラスティック保存、レジスタ ファイル、バッファ、循環バッファ、シフト レジスタ、遅延ライン、データの波形保存と生成、ダイレクト デジタル合成、CAM、連想メモリ、ファンクション テーブル、ファンクション ジェネレータ、多入力ロジック ファンクション、コード コンバータ、エンコーダ、デコーダ、カウンタ、ステート マシン、マイクロシーケンサ、エンベデッド プロセッサ用プログラム ストレージ

ザイリンクス CORE Generator システムは、Spartan-3 用のブロック RAM を含む多様なモジュールをサポートします。以下は、その例です。

- エンベデッド デュアルまたはシングル ポート RAM モジュール
- ROM モジュール
- 同期および非同期 FIFO モジュール
- CAM (連想メモリ) モジュール

さらに、ザイリンクス デザイン ライブラリにある適切な RAMB16 モジュールを使用すると、合成ベースのどのデザインでもブロック RAM を使用できます。

このアプリケーション ノートでは、Spartan-3 ブロック RAM で使用できる信号と属性についてだけでなく、ブロック RAM アプリケーションについても説明します。

ブロック RAM 配置

前述したように、ブロック RAM は縦に並べられ、図 2 に示す XC3S200 には、デバイスの左エッジにある CLB 2 列の隣にブロック RAM が1 列あります。XC3S50 より大きな Spartan-3 デバイスには、ダイの左右、各エッジにある I/O から CLB 2 列分離れた位置にブロック RAM が 1 列ずつあります。各エッジにあるブロック RAM に加えて、XC3S4000 および XC3S5000 では、エッジ間を均等に分割するように 2 列のブロック RAM があり、合計 4 列になります。表 1 にデバイスごとの列数およびブロック RAM 数を示します。このように、ブロック RAM 列が各エッジにあるために、バッファリングまたはデバイスにつながるバスの再同期に非常に有効です。

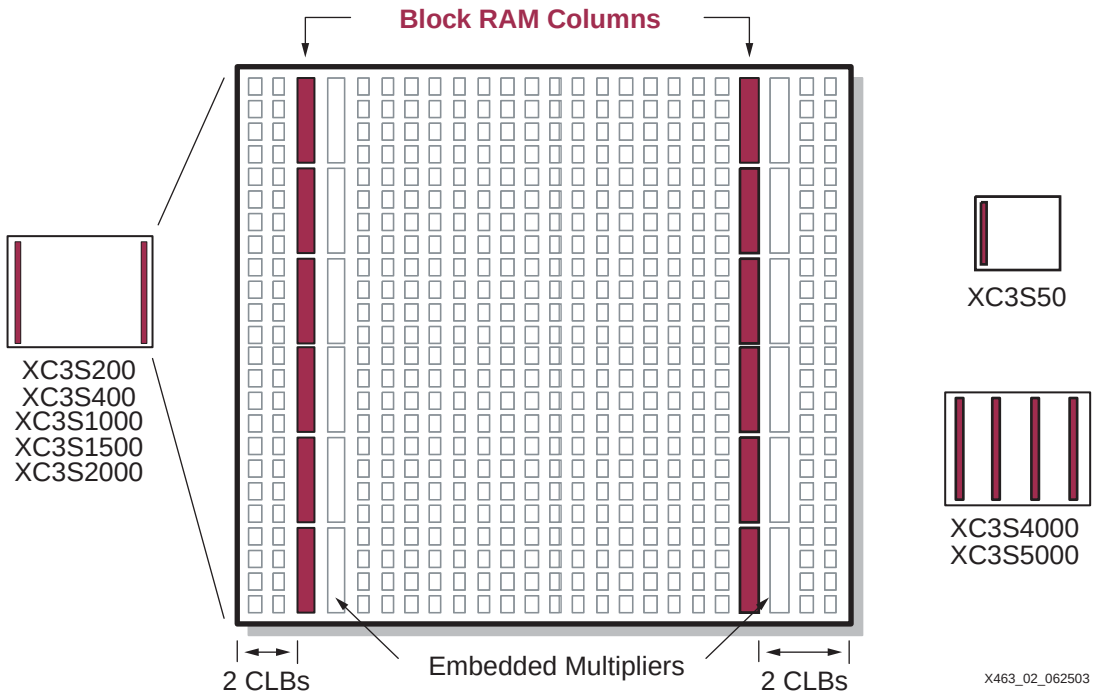


図 2: XC3S200 のブロック RAM 配置

各ブロック RAM に隣接しているのは、18x18 エンベデッド乗算器です。ブロック RAM とエンベデッド乗算器を隣り合わせに配置することによって、デジタル信号処理機能を向上させることができます。

ブロック RAM と周囲のブロックを相互接続させることによって、アドレスおよびデータへの信号分配を効果的に行うことができ、さらに、複数のブロック RAM をカスケード接続すると、ワード数、ビット数の大きなメモリを作成できます。

データ フロー

Spartan-3 ブロック RAM は完全なデュアル ポート メモリであり、図 3 に示すデータ フローのすべてを同時にサポートします。 両ポートは、同じメモリ ビットにアクセスしますが、ポートのデータ幅によってアドレス設定仕様は異なる場合があります。

1. ポート A は、1 セットのアドレス ラインを使用し、同時に読み出し / 書き込みを行う独立したシングル ポート RAM として機能します。
2. ポート B は、1 セットのアドレス ラインを使用し、同時に読み出し / 書き込みを行う独立したシングル ポート RAM として機能します。
3. ポート A は書き込みポート、ポート B は読み出しポートであり、個別のアドレスを持ちます。ポート A とポート B のデータ幅は異なることがあります。
4. ポート B は書き込みポート、ポート A は読み出しポートであり、個別のアドレスを持ちます。ポート B とポート A のデータ幅は異なることがあります。

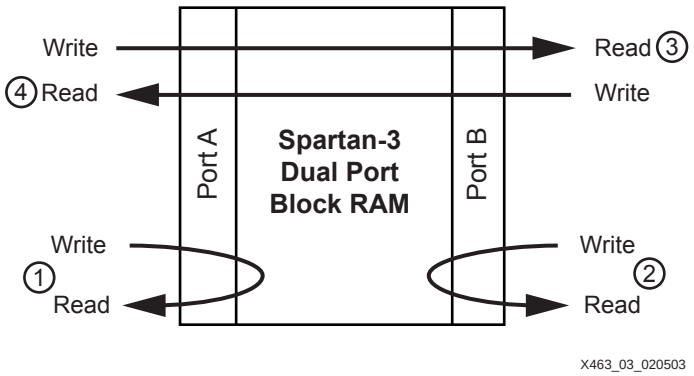


図 3: シングルおよびデュアル ポート ブロック RAM でのデータ転送

信号

ブロック RAM プリミティブに接続される信号は、次に示すように 4 つに分類されます。表 3 には、ブロック RAM インターフェイス信号、シングル ポートとデュアル ポート メモリに使用する信号名、および信号方向を示します。

1. データ入力および出力
2. パリティ入力および出力 (データ ポート幅が 1 バイト以上の場合に使用可能)
3. 特定のメモリ位置を選択するためのアドレス入力
4. 読み出し、書き込み、またはセット / リセット処理に機能するさまざまな制御信号

表 3: ブロック RAM インターフェイス信号

信号	シングルポート	デュアル ポート		方向
		ポート A	ポート B	
データ入力バス	DI	DIA	DIB	入力
パリティ データ入力バス (データ幅が 1 バイト以上の場合)	DIP	DIPA	DIPB	入力
データ出力バス	DO	DOA	DOB	出力
パリティ データ出力バス (データ幅が 1 バイト以上の場合のみ)	DOP	DOPA	DOPB	出力
アドレス バス	ADDR	ADDRA	ADDRB	入力
ライト イネーブル	WE	WEA	WEB	入力
クロック イネーブル	EN	ENA	ENB	入力
同期セット/リセット	SSR	SSRA	SSRB	入力
クロック	CLK	CLKA	CLKB	入力

データ入力および出力

図 4 に示す例では、データ ポート幅全体にデータ バスとパリティ バスの両方を含みます。たとえば、512x36 の場合、36 ビット データ ポート幅には MSB として 4 パリティ ビットが含まれ、それに LSB として 32 データ ビットが続きます。

データおよびパリティ出入力信号は、常にバスです。つまり、1 ビット幅のコンフィギュレーションで、データ入力信号は DI[0] となり、データ出力信号は DO[0] となります。

データ入力バス — DI[#:0] (DIA[#:0]、DIB[#:0])

データ入力バスは、RAM に書き込まれるデータ ソースです。

DI 入力バスにあるデータは、クロック イネーブル EN および ライト イネーブル WE が High のとき、クロック入力が Low から High への立ち上がりエッジでアドレス入力バス ADDR によって指定された RAM の位置に書き込まれます。

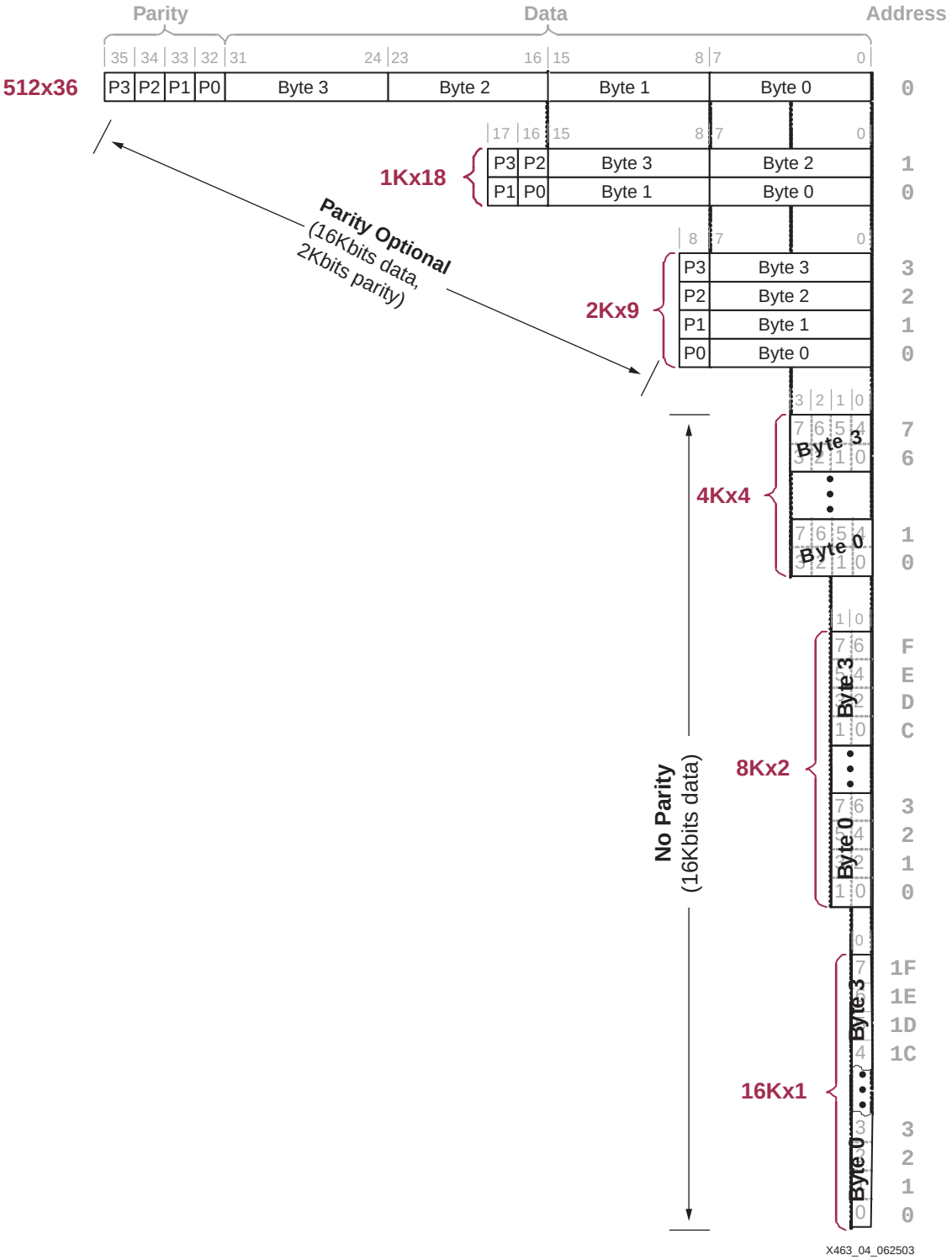


図 4: Data Organization and Mapping Between Modes

データ出力バス — DO[#:0] (DOA[#:0]、DOB[#:0])

読み出し処理では、アクティブなクロック エッジでアドレス バス ADDR が指定したメモリ セルの内容が、データ出力バス DO に送信されます。同時に実行されている書き込みでは、データ出力のラッチは WRITE_MODE 属性によって制御されています (同時書き込み中の読み出し — WRITE_MODE、ページ 14 を参照してください)。

パリティ入力および出力

パリティはデータ パスが 1 バイト以上の場合のみ使用できます。

パリティ ビットは、そこに適切なデータがあることを示す役割を果たしますが、パリティ入力および出力が特定の機能を持つわけではありませんので、追加のデータ ビットとしても使用できます。たとえば、コードまたはデータ、正負、新旧などの情報をデータにタグ付けするようなデータ ワードに関する追加情報のために、このパリティ ビットを使用することが可能です。

ブロック RAM にはパリティを生成またはチェックするための特別な回路はありません。アプリケーションでこのような機能を使用する場合には、CLB ロジック リソースを使用し、作成してください。

データ入力パリティ バス — DIP[#:0] (DIPA[#:0]、DIPB[#:0])

DIP 入力バスにあるデータは、クロック イネーブル EN およびライト イネーブル WE 入力が High のとき、クロック入力が Low から High への立ち上がりエッジでアドレス入力バス ADDR によって指定された RAM の位置に書き込まれます。

データ出力パリティ バス — DOP[#:0] (DOPA[#:0]、DOPB[#:0])

読み出し処理では、アクティブなクロック エッジでアドレス バス ADDR が指定したメモリ セルの内容が、データ出力バス DOP に送信されます。同時に実行されている書き込みでは、データ出力のラッチは WRITE_MODE 属性によって制御されています (同時書き込み中の読み出し — WRITE_MODE、ページ 14 を参照してください)。

アドレス入力

デュアル ポート RAM の 2 つのポートは、独立して同一の 18K ビット メモリ セルにアクセスできます。

アドレス バス — ADDR[#:0] (ADDRA[#:0]、ADDRB[#:0])

アドレス バスは、読み出しまたは書き込みを行うメモリ セルを選択します。表 5 に示すように、アドレス バス入力によって、必要なアドレス バスの幅は異なります。

制御入力

クロック — CLK (CLKA、CLKB)

各ポートは、それぞれのクロック ピンに完全に同期します。すべてのポートの入力ピンにはセットアップ タイムがあり、CLK ピンを基準とします。また、データ バスの clock-to-out タイムも、そのポートの CLK ピンを基準とします。クロックの極性は変更でき、デフォルトでは、立ち上がりエッジです。

このデフォルト設定では、クロック (CLK) 入力の Low から High で読み出し、書き込み、およびリセットが制御されます。

イネーブル — EN (ENA、ENB)

イネーブル入力 EN は、読み出し、書き込み、セット/リセットを制御します。EN が Low のときは、データは書き込まれず、出力 DO と DOP は前の状態の値を維持します。EN の極性は設定でき、デフォルトではアクティブ High です。

EN がアサートされ、アクティブな同期セット/リセット入力やライト イネーブル入力がない場合、ブロック RAM は、クロックの立ち上がりエッジでアドレス バス ADDR が指定した位置にあるデータをメモリから読み出します。

ライト イネーブル — WE (WEA、WEB)

ライト イネーブル入力 WE は RAM へのデータの書き込みを制御します。クロックの立ち上がりエッジで EN および WE がアサートされると、アドレス バスで指定されたメモリ セルにデータおよびパリティ入力の値が書き込まれます。

出力ラッチにデータが送信されるかどうかは、WRITE_MODE 属性によって異なります。

WE の極性は設定可能で、デフォルトではアクティブ High です。

同期セット/リセット — SSR (SSRA、SSRB)

同期セット/リセット入力 SSR によって、データ出力ラッチは SRVAL 属性が指定した値になります。SSR とイネーブル信号 EN が High の状態になると、SRVAL 属性によって DO と DOP のデータ出力ラッチは、0 または 1 に同期設定されます。

この同期セット/リセットのために RAM メモリ セルが変化することはなく、もう一方のポートでの書き込みにも影響を与えません。

SSR の極性は設定可能でデフォルトではアクティブ High です。

グローバル セット/リセット — GSR

グローバル セット/リセット信号 GSR は、デバイスのコンフィギュレーション直後に自動的にアサートされます。STARTUP プリミティブをインスタンシエートし、GSR をアサートすると Spartan-3 の初期状態に常に戻すことができます。GSR 信号を使用すると、出力ラッチは INIT の値に初期化されますが、内部メモリの内容には影響を与えません。

GSR はグローバル信号であり、自動的にデバイス全体に接続されるため、ブロック RAM プリミティブの GSR 入力ピンはありません。

制御ピンの反転

各ポートの 4 つの制御ピン - CLK、EN、WE および SSR - は、それぞれ個別に反転できます。制御信号はアクティブ High とアクティブ Low のどちらにも設定でき、クロックの立ち上がり/立ち下がりエッジでアクティブにできます。反転するためのロジック リソースの追加は必要ありません。

未使用の入力

未使用のデータまたはアドレス入力は、ロジック 1 にする必要があります。未使用の入力を High に接続すると、Low に接続する場合よりも、使用するロジックと配線リソースが少なくなります。

属性

表 4 に示すように、ブロック RAM には VHDL または Verilog で使用できる多くの属性があります。ただし、CORE Generator システムで使用する値は、次に示すものとわずかに異なります。

表 4: ブロック RAM 属性および VHDL/Verilog 属性名

機能	VHDL または Verilog 属性	デフォルト値
ポート数	適切な RAMB16 プリミティブをインスタンシエートすることで定義	なし
メモリ構造	適切な RAMB16 プリミティブをインスタンシエートすることで定義	なし
コンフィギュレーション中にデータ メモリの内容を初期化	INIT_xx	0 に初期化
コンフィギュレーション中にパリティ メモリの内容を初期化	INITP_xx	0 に初期化
データ出力ラッチの初期化	INIT (シングル ポート) INIT_A、INIT_B (デュアル ポート)	0 に初期化

表 4: ブロック RAM 属性および VHDL/Verilog 属性名 *(Continued)*

機能	VHDL または Verilog 属性	デフォルト値
データ出力ラッチ同期セット/リセット	SRVAL (シングル ポート) SRVAL_A、SRVAL_B (デュアル ポート)	0 にリセット
書き込み中のデータ出力ラッチ	WRITE_MODE	WRITE_FIRST
ブロック RAM 配置	LOC	なし

ポート数

ブロック RAM は物理的にデュアル ポート メモリですが、シングル ポートまたはデュアル ポート メモリとして使用できます。ポート数の指定方法は、デザイン入力に使用するツールによって異なります。

CORE Generator システム

図 5 に示すように、CORE Generator を使用し、さまざまなタイプのメモリ ブロック用のモジュールを生成できます。シングルまたはデュアル ポート ブロック メモリを選択するか、ハイレベルの機能を使用し、FIFO、CAM (連想メモリ) などを生成します。

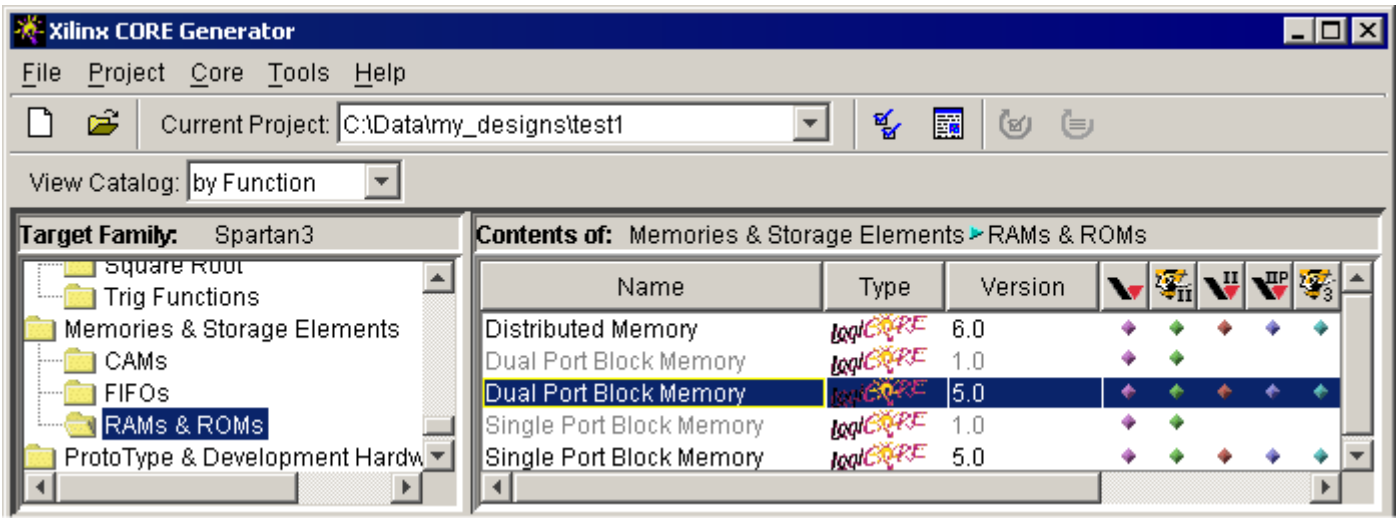


図 5: CORE Generator システムでのブロック RAM 機能の選択

VHDL または Verilog のインスタンシエーション

ザイリンクスのデザイン ライブラリには、図 1 に示されているものと類似したシングルおよびデュアル ポート メモリ プリミティブがあります。メモリ構造、アスペクト比だけでなく、シングル ポートまたはデュアル ポート メモリかを決定し、適切なプリミティブを選択します。シングル ポートおよびデュアル ポートのブロック RAM プリミティブについては、表 5 と表 6 を参照してください。

メモリ構成/アスペクト比

ブロック RAM のメモリ構成やアスペクト比は、表 5 に示すように設定できます。データ パス幅が 1 バイト以上の場合、ブロック RAM には、パリティをサポートする追加ビットが各バイトごとにあります。したがって、1Kx18 のメモリでは、16 ビット (2 バイト)のデータと各バイトごとに 1 ビットのパリティ ビットがあるため、ビット幅が 18 となっています。また、ポートからアクセスできるメモリの物理的な大きさは、メモリ構造によって異なります。1 バイト以上のメモリでは、18K ビットがアクセス可能であり、幅がそれ以下の場合は、パリティ ビットがないため 16 K ビットのみにアクセスできます。基本的に、18k ビットのブロック RAM では、16K ビットがデータに割り当てられ、2K ビットがパリティ

に割り当てられます。各メモリ構成での、データ マッピングについての詳細は、図 4 を参照してください。

表 5: ブロック RAM データ構成/アスペクト比

構成	ワード数	データ幅	パリティ	DI/DO	DIP/DOP	ADDR	シングル ポート プリミティブ	RAM の総 K ビット数
512x36	512	32	4	(31:0)	(3:0)	(8:0)	**RAMB16_S36**	18K
1Kx18	1024	16	2	(15:0)	(1:0)	(9:0)	**RAMB16_S18**	18K
2Kx9	2048	8	1	(7:0)	(0:0)	(10:0)	**RAMB16_S9**	18K
4Kx4	4096	4	-	(3:0)	-	(11:0)	**RAMB16_S4**	16K
8Kx2	8192	2	-	(1:0)	-	(12:0)	**RAMB16_S2**	16K
16Kx1	16384	1	-	(0:0)	-	(13:0)	**RAMB16_S1**	16K

CORE Generator システム — メモリ サイズ

CORE Generator を使用することによって、さまざまなアスペクト比で使用できるメモリを作成することができます。実際のブロック RAM プリミティブとは異なり、CORE Generator システムはデータ ビットとパリティ ビットを区別せずにすべてのビットをデータ ビットと判断します。そして、デュアル ポート メモリの場合、各ポートにぞれぞれの構成とアスペクト比を設定できます。

CORE Generator では図 6 に示すように、望ましいメモリ サイズを指定します。

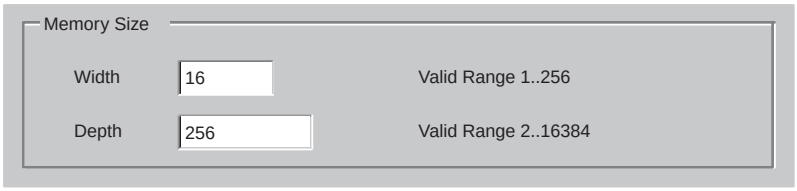


図 6: CORE Generator システムでのメモリ幅とワード数の指定

VHDL または Verilog のインスタンシエーション

適切な SelectRAM コンポーネントを指定またはインスタンシエートし、アスペクト比と定義します。表 5 にシングル ポート RAM 用の SelectRAM コンポーネントを示します。 シングル ポート RAM の場合、適切なコンポーネント名は RAMB16_S***n*** となります。ここで、***n*** はデータビットとパリティ ビットを含むデータ幅です。 たとえば、1Kx18 シングル ポート RAM が使用するコンポーネントは RAMB16_S18 です。ここでは、16 データ ビットと 2 パリティ ビットあるため、***n***=18 となります。

デュアル ポート メモリでは、2 つのメモリ ポートのアスペクト比が異なる場合があるため、シングル ポートの場合よりわずかに複雑になります。デュアル ポート RAM の適切なコンポーネント名は、RAMB16_S***m***_S***m*** となります。ここで ***m*** および ***n*** は、ポート A、B それぞれのデータ パス幅を表します。たとえば、表 6 に示されているサフィックスを使用すると、2Kx9 のポート A と 1Kx18 のポート B を持つ適当なデュアル ポート RAM コンポーネントは RAMB16_S9_S18 になります。この例では、***m***=9、***n***=18 となります。

表 6: RAMB16 デュアル ポート RAM コンポーネント サフィックス

		ポート A					
		16Kx1	8Kx2	4Kx4	2Kx9	1Kx18	512x36
ポート B	16Kx1	**_S1_S1**					
	8Kx2	**_S1_S2**	**_S2_S2**				
	4Kx4	**_S1_S4**	**_S2_S4**	**_S4_S4**			
	2Kx9	**_S1_S9**	**_S2_S9**	**_S4_S9**	**_S9_S9**		
	1Kx18	**_S1_S18**	**_S2_S18**	**_S4_S18**	**_S9_S18**	**_S18_S18**	
	512x36	**_S1_S36**	**_S2_S36**	**_S4_S36**	**_S9_S36**	**_S18_S36**	**_S36_S36**

ポート間のアドレスおよびデータ マッピング

デュアル ポート モードでは、2 つのポートで同時に同じメモリ セルにアクセスすることが可能ですが、メモリ構成やアスペクト比は必ずしも同じではありません。図 4 にアスペクト比の異なるデータ セットを示します。

データ幅が、1 バイト以上の場合、過剰なビットをメモリのパリティ ビットとして使用できます。パリティ ビットは、特定のバイトに関連して使用され、データ ポートの MSB にあります。たとえば、x36 データ ワード (32 データと 4 パリティ) が 2 つの x18 ハーフワード (16 データと 2 パリティ) でアドレス指定されていると、各データ バイトに関連したパリティ ビットは、ブロック RAM 内で適切なパリティ ビットにマッピングされます。x36 データ ワードを 4 つの x9 ワードとしてマップした場合も同様ですが、x4、x2 または x1 として使用するとパリティ ビットは使用できません。

2 つのポートが異なるメモリ構造の場合に、開始アドレスおよび終了アドレスを求める式は、次のようになり、ポート Y のアドレスおよびポート幅とポート X のポート幅から、それらのアドレスがわかります。

$$START\_ADDRESS_X = INTEGER\left(\frac{ADDRESS_Y \bullet WIDTH_Y}{WIDTH_X}\right)$$

$$END\_ADDRESS_X = INTEGER\left(\frac{((ADDRESS_Y + 1) \bullet WIDTH_Y) - 1}{WIDTH_X}\right)$$

ただし、デュアル ポートで一方のポートにのみパリティ ビットが含まれている場合には、この式は当てはまりません。データ ビットのみを含む値を使用して計算してください。8 ビット以下のポート幅では、パリティ ビットは使用できません。

メモリ内容の初期化

デフォルトでは、ブロック RAM メモリは、デバイスのコンフィギュレーション シーケンス中にすべてゼロに初期化されますが、メモリ内容もユーザー定義のデータに初期化することが可能です。さらに、RAM の内容は、コンフィギュレーション中に誤って書き込まれることがないように保護されています。

CORE Generator システム — Init ファイルの取り込み

CORE Generator ブロック RAM ファンクション用に RAM の初期内容を指定するには、coefficients (.coe) ファイルを作成します。この coefficients (.coe) ファイルの簡単な例を図 7 に示します。データを

初期化するには、まず 2、10、16 のような基数を指定し、0 の位置にあるデータから順次、RAM の内容を指定します。

```
memory_initialization_radix=16;
memory_initialization_vector= 80, 0F, 00, 0B, 00, 0C, ÅE 81;
```

図 7: 簡単な COE ファイル例

COE ファイルを含むには、CORE Generator で適切な画面を開き、図 8 に示すように、Load Init File にチェックを入れます。次に Load File をクリックし、COE ファイルを選択してください。

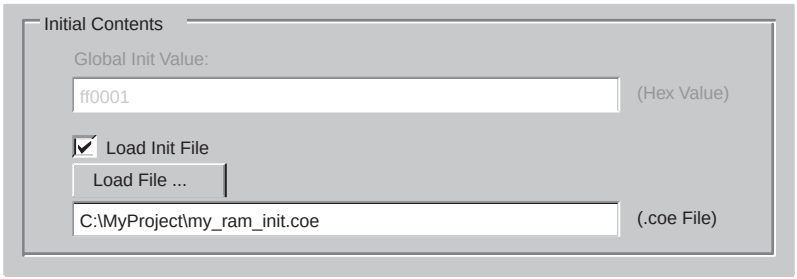


図 8: CORE Generator 使用した RAM の内容の初期化

VHDL または Verilog のインスタンシエーション — INIT_xx、INITP_xx

VHDL および Verilog インスタンシエーションには、2 タイプの属性を使用します。**INIT_xx** 属性はデータ メモリ位置の初期内容を指定します。**INITP_xx** 属性はパリティ メモリ位置の初期内容を指定します。

INIT_xx 属性はデータ メモリの初期内容を指定し、**INIT_00** から **INIT_3F** の 64 個の初期化属性があります。**INIT_xx** 属性は 64 桁 (256 ビット) のビット ベクトルです。一部のメモリ内容を初期化することも可能で、初期値を指定しない部分は、自動的にゼロになります。

次に各 **INIT_xx** 属性のビット位置を求める式を示します。

yy が 16 進数 (xx) を 10 進数に変換した値の場合、**INIT_xx** は、次のようなメモリ セルに対応します。

- 開始位置 : [(yy + 1) * 256] -1
- 終了位置 : (yy) * 256

たとえば、INIT_1F 属性 は次のように変換されます。

- yy = 16 進数 (0x1F) を 10 進数に変換した値 = 31
- 開始位置 : [(31+1) * 256] -1 = 8191
- 終了位置 : 31 * 256 = 7936

表 7: ブロック RAM の VHDL/Verilog RAM インスタンシエーション属性

属性	開始	終了
INIT_00	255	0
INIT_01	511	256
INIT_02	767	512
...	...	...
INIT_3F	16383	16128

INITP_xx 属性は、DIP/DOP バスに接続しているようなパリティ ビットに対応したメモリ セルの内容を初期化します。デフォルトでは、これらのメモリ セルもすべてゼロに初期化されます。

INITP_00 から **INITP_07** の 8 つの初期化属性は、パリティ ビットのメモリ内容を表し、各 **INITP_xx** は、**INIT_xx** 属性と同様に機能する 16 進数の 64 桁 (256 ビット) ベクタ ビットです。特定の **INITP_xx** 属性で初期化されるビット位置も、同じ式で計算できます。

データ出力ラッチの初期化

コンフィギュレーション後、またはグローバル セット/リセット信号 GSR がアサートされた後に、ブロック RAM 出力ラッチをユーザー指定の値に初期化できます。各ポートに異なる初期値を設定することも可能ですが、値を設定しない場合はゼロになります。

CORE Generatorシステム — グローバル Init 値

図 9 は CORE Generator でのデータ出力ラッチの初期値指定画面です。16 進数で指定する値は、データ幅と同じビット数でなければならず、デュアル ポート メモリでは、各ポートで別々の初期値を設定できます。

図 9: ブロック RAM データ出力ラッチの初期値指定

VHDL または Verilog インスタンシエーション — INIT (INIT_A および INIT_B)

VHDL または Verilog では、INIT 属性 (デュアル ポート メモリでは INIT_A と INIT_B) で、コンフィギュレーション後の出力ラッチの値を指定します。INIT (または INIT_A と INIT_B) 属性は、データの初期値と、場合によってはパリティ ビットを指定します。図 4 にパリティ ビットを含む各メモリ構成のビット フォーマットを示します。ここで、パリティ ビットはデータ ビットの前、MSB にあります。たとえば、2Kx9 メモリの初期値は 1 パリティ ビットとそれに続く 8 データ ビットの 9 ビット幅であり、これらの属性は 16 進数のビット ベクタ、デフォルト値はゼロになります。

データ出力ラッチ同期セット/リセット

同期セット/リセット入力 SSR がアサートされると、セット/リセット属性によって、データ出力ラッチが設定され、デュアル ポート メモリでは、各ポートでそれぞれの初期値を設定できます。

設定しない場合は、有効な同期セット/リセット中に、出力ラッチがゼロにリセットされます。

CORE Generatorシステム — Init 値 SINIT)

図 10 に CORE Generator での、データ出力ラッチ同期セット/リセット値の指定方法を示します。SINIT pin のチェック ボックスをオンにし、16 進数で、同期セット/リセット値を設定します。この値は、データ幅と同じビット数でなければならず、デュアル ポート メモリでは、各ポートで別々の初期値を設定できます。

図 10: データ出力ラッチ セット/リセット値の設定

VHDL または Verilog インスタンシエーション — SRVAL (SRVAL_A および SRVAL_B)

VHDL または Verilog では、SRVAL 属性 (デュアル ポート メモリでは SRVAL_A と SRVAL_B) で、コンフィギュレーション後の出力ラッチの値を指定します。SRVAL (または SRVAL_A と SRVAL_B) 属性は、データの初期値と、場合によってはパリティ ビットを指定します。図 4 にパリティ ビットを含む各メモリ構成のビット フォーマットを示します。ここで、パリティ ビットはデータ ビットの前、MSB にあります。 これらの属性は、16 進数のビット ベクタで、デフォルト値は 0 です。

同時書き込み中の読み出し — WRITE_MODE

ブロック RAM の各ポートでの書き込みには 3 つのモードがあり、各クロック エッジでのデータ処理能力と使用率を最大にします。この 3 つのモードによって、有効な書き込みクロック エッジの後の出力ラッチのデータは異なります。デフォルトの WRITE_FIRST を使用すると、Virtex™/E および Spartan-IIE FPGA アーキテクチャと互換性を持たせることができ、これは Virtex-II/Pro デバイスでもデフォルトの設定となっています。クロック サイクルごとのブロック RAM の効率を上げ、デザインで最大のバンド幅を使用できる点から、READ_FIRST モードが最も有用です。この READ_FIRST モードでは、タイミングの問題なしに、1 つのクロック エッジで同一アドレスの読み出しと書き込みを行えます。

表 8 に WRITE_MODE 設定と同一ポートの出力ラッチのデータについて、および同時に同じアドレスにアクセスした場合の、もう一方のポートでの出力ラッチのデータについて簡潔に示します。

表 8: 書き込み WRITE_MODE と出力ラッチのデータ

書き込みモード	同一ポートへの影響	もう一方のポートへの影響 (デュアル ポートで同一アドレスにアクセスする場合のみ)
WRITE_FIRST 書き込み後に読み出し (デフォルト)	DI、DIP の入力データを RAM の指定された位置にを書き込むのと同時に、DO、DOP に出力	DO、DOP 出力データは無効
READ_FIRST 書き込み前に読み出し (推奨)	RAM の指定された位置にあるデータを DO、DOP に出力 DI、DIP 入力のデータの書き込み	RAM の指定された位置にあるデータを DO、DOP に出力
NO_CHANGE 書き込み中の読み出しなし	DO、DOP 出力のデータは変化せず DI、DIP 入力のデータの書き込み	DO、DOP 出力データは無効

これらの書き込みモードは、コンフィギュレーションで設定し、属性を使用することで各ポートに個別に設定できます。デフォルトでは WRITE_FIRST に設定されています。

WRITE_FIRST または トランスペアレント モード (デフォルト)

以前のデザインと互換性を持たせることができるため WRITE_FIRST モードがデフォルト設定となっていますが、新規のデザインには READ_FIRST モードを設定することをお奨めします。

WRITE_FIRST モードでは、入力データを RAM に書き込むのと同時にデータ出力ラッチに送信するため、図 11 に示すようにトランスペアレント書き込みになります。このモードに設定すると、Virtex/E お

よび Spartan-II/E FPGA の 4K ビット ブロック RAM と互換性があり、さらに Virtex-II/Pro ブロック RAM ではデフォルト設定となっています。

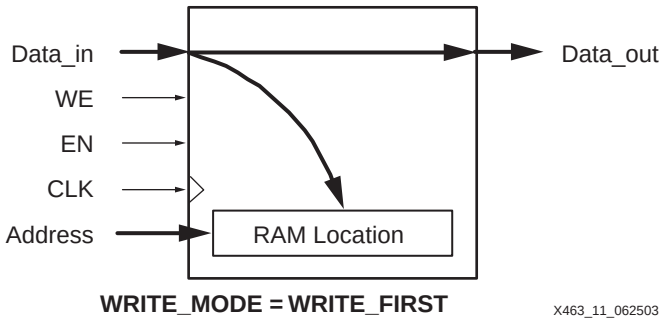


図 11: WRITE_FIRST モード書き込みのデータ フロー

図 12 に書き込まれた有効なデータが、同時にデータ出力に送信される書き込み処理を示します。

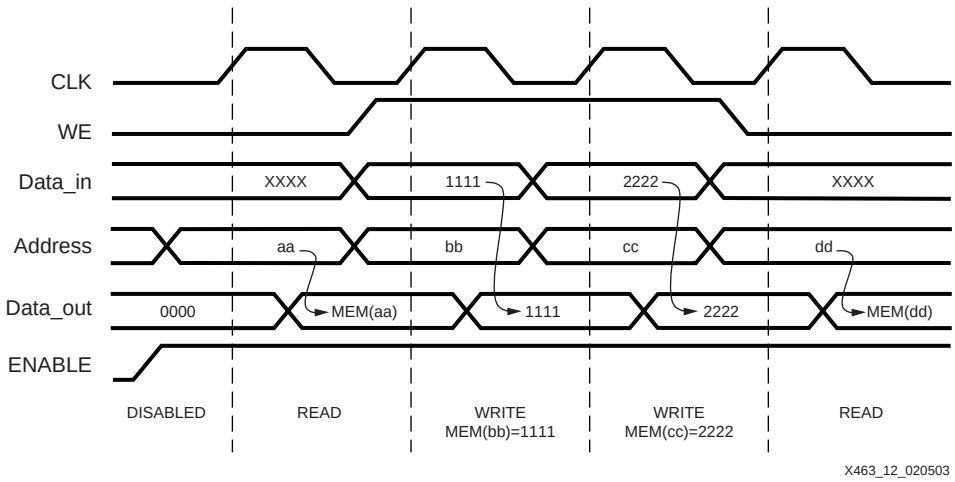


図 12: WRITE_FIRST モード波形

READ_FIRST または 書き込み前に読み出しモード

図 13 に示すように、READ_FIRST モードでは、書き込み前に読み出し処理を実行します。この場合、書き込みアドレスに保存されているデータを出力ラッチに送信すると同時に、入力データをメモリに保存します。ここでは、以前の RAM データが出力され、新しいデータが指定されたアドレスに保存されます。書き込みモードとしては、この READ_FIRST を推奨しています。

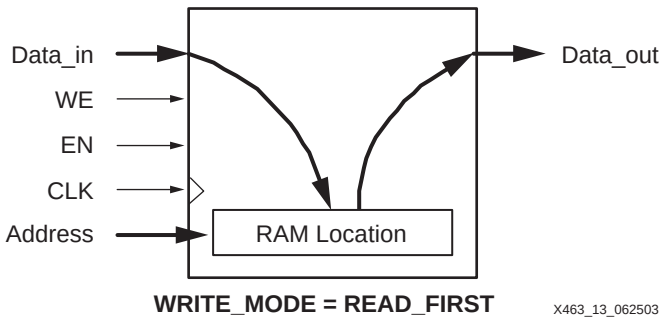


図 13: READ_FIRST モード書き込みのデータ フロー

図 14 は、同時に書き込みが行われているにもかかわらず、それ以前の RAM データが読み出されていることを示します。

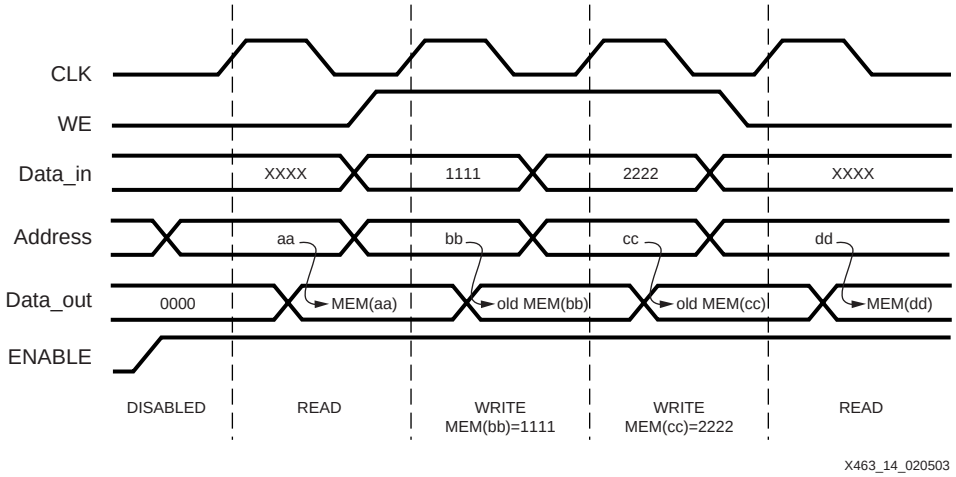


図 14: READ_FIRST モード波形

このモードは、特に循環バッファおよび大規模でブロック RAM ベースのシフト レジスタを作成する場合に役立ち、DSP アプリケーションで FIR フィルタ タップを保存する際にも有用です。古いデータは RAM からコピー され、新規データが RAM に書き込まれます。

NO_CHANGE モード

図 15 に示すように、NO_CHANGE モードでは、同時に実行されている書き込み中に出力ラッチは変化しません。これは、1 クロック サイクルで読み込み/書き出しのどちらかを実行する簡単な同期メモリに類似しています。

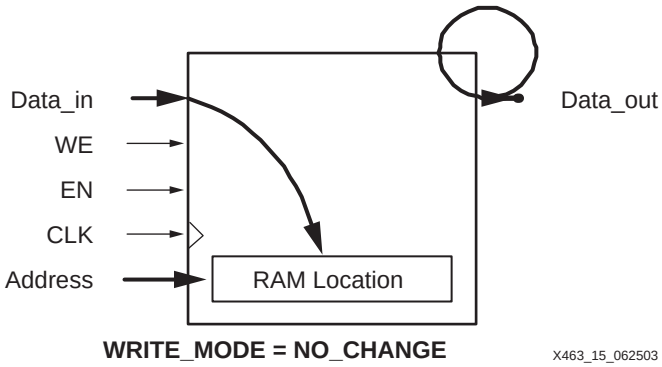


図 15: NO_CHANGE モード書き込みのデータ フロー

NO_CHANGE モードは、ブロック RAM に波形、ファンクション テーブル、係数などが含まれるアプリケーションで使用すると有効です。また、出力に影響を与えずにメモリを更新できます。

データ出力は、同じポートでの書き込みの影響を受けず、最後に読み出されたデータが維持されていることを 図 16 に示します。

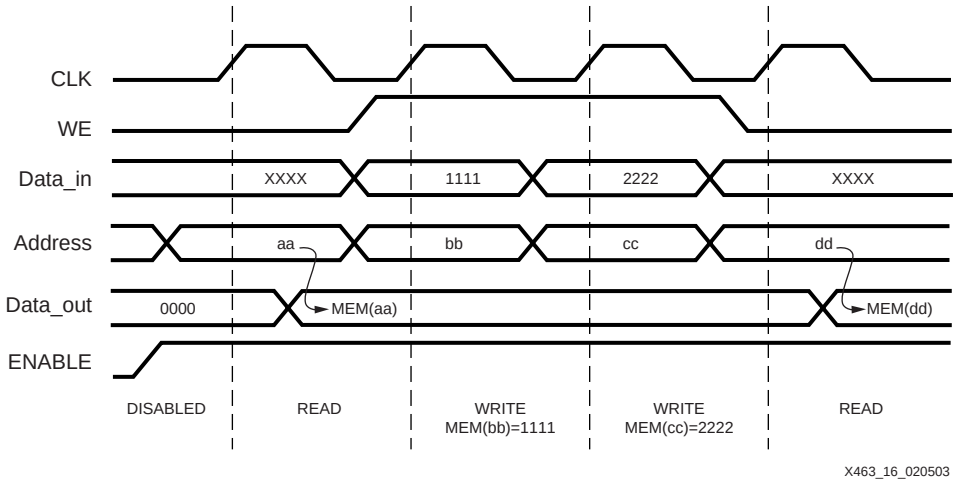


図 16: NO_CHANGE モード波形

CORE Generator システム — 書き込みモード

CORE Generator システムでは、図 17 のような書き込みモード設定画面を表示し、Read After Write (WRITE_FIRST)、Read Before Write (READ_FIRST)、または No Read On Write (NO_CHANGE) から選択します。

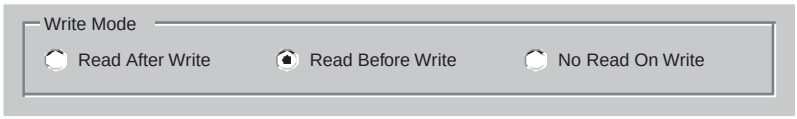


図 17: CORE Generator での書き込みモード選択

VHDL または Verilog のインスタンシエーション — WRITE_MODE

ブロック RAM をインスタンシエーションするときに WRITE_MODE 属性を使用し、書き込みモードを指定します。WRITE_FIRST、READ_FIRST および NO_CHANGE のような使用可能な値については、付録の例に示します。

配置制約 (LOC)

通常は、ザイリンクス ISE ソフトウェアを使用してブロック RAM を配置するのが最もよい方法ですが、LOC プロパティを適用し、Spartan-3 デバイス上での配置を制約することもできます。ブロック RAM の配置ロケーションの表記方法はデバイスごとに異なり、CLB 配置の命名方法とも異なるため、ほかのアレイでも容易に LOC プロパティを使用できます。

LOC プロパティは、次の形式で使用します。

```
LOC = RAMB16_X#Y#
```

図 18 に示すように、**RAMB16_X0Y0** は、デバイスの左下にあるブロック RAM の位置です。右上にあるブロック RAM の位置は、表 1 に示されているブロック RAM の列数 **n** およびその行数 **m** によって決定されます。

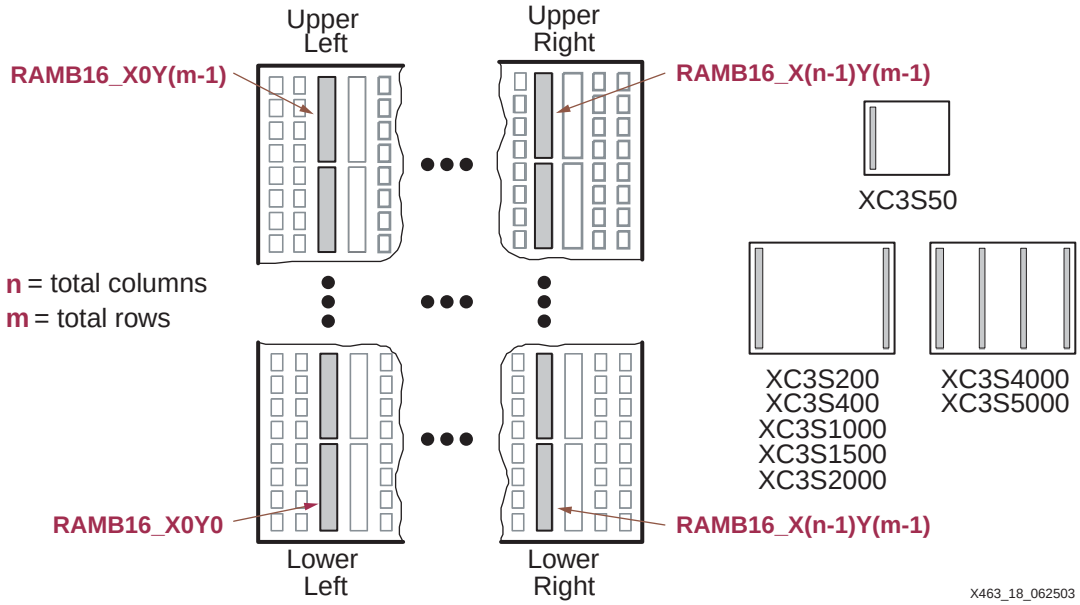


図 18: ブロック RAM での LOC 座標

CORE Generator で、ロケーションを制約する属性を直接、指定することはできませんが、VHDL または Verilog インスタンシエーションでは追加可能です。

ブロック RAM

表 9 にブロック RAM のビヘイビアを示します。ここでは、すべての制御信号はデフォルトのアクティブ High で動作するという前提ですが、制御信号は、必要に応じて反転できます。次に、シングル メモリ ポートについて説明します。デュアル ポート モードでは、両ポートがそれぞれに独立したシングル ポート メモリとして機能します。

ブロック RAM に対するすべての読み出し/書き込みは同期で、すべての入力にクロックへのセットアップ タイムがあり、すべての出力には clock-to-output タイムがあります。

表 9: ブロック RAM の機能

入力信号								出力信号		RAM 内容	
GSR	EN	SSR	WE	CLK	ADDR	DIP	DI	DOP	DO	パリティ	Data
コンフィギュレーション直後											
コンフィギュレーション中の送信								X	X	INITP_xx[2]	INIT_xx[2]
コンフィギュレーション直後のグローバル セット/リセット											
1	X	X	X	X	X	X	X	INIT[3]	INIT	No Chg	No Chg
RAM ディスエーブル											
0	0	X	X	X	X	X	X	No Chg	No Chg	No Chg	No Chg
同期セット/リセット											
0	1	1	0	↑	X	X	X	SRVAL[4]	SRVAL	No Chg	No Chg
RAM 書き込み中の同期セット/リセット											
0	1	1	1	↑	addr	pdata	Data	SRVAL	SRVAL	RAM(addr) ←pdata	RAM(addr) ← data

表 9: ブロック RAM の機能 *(Continued)*

入力信号								出力信号		RAM 内容	
GSR	EN	SSR	WE	CLK	ADDR	DIP	DI	DOP	DO	パリティ	Data
RAM の読み出しのみ (書き込みなし)											
0	1	0	0	↑	addr	X	X	RAM(pdata)	RAM(data)	No Chg	No Chg
同時に行われる RAM の読み出し/書き込み											
0	1	0	1	↑	addr	pdata	Data	WRITE_MODE = WRITE_FIRST5 (デフォルト)			
								pdata	data	RAM(addr) ←pdata	RAM(addr) ← data
								WRITE_MODE = READ_FIRST6 (推奨)			
								RAM(data)	RAM(data)	RAM(addr) ←pdata	RAM(addr) ←pdata
								WRITE_MODE = NO_CHANGE7			
								No Chg	No Chg	RAM(addr) ←pdata	RAM(addr) ←pdata

メモ：

1. No Chg = 変化なし、addr = RAM へのアドレス、data = RAM データ、pdata = RAM パリティ データ
2. メモリ内容の初期化、ページ 11 を参照
3. データ出力ラッチの初期化、ページ 13 を参照
4. データ出力ラッチ同期セット / リセット、ページ 13 を参照
5. WRITE_FIRST または トランスペアレント モード (デフォルト)、ページ 14 を参照
6. READ_FIRST または 書き込み前に読み出しモード、ページ 15 を参照
7. NO_CHANGE モード、ページ 16 を参照

コンフィギュレーション中の RAM 内容の初期化

Spartan-3 コンフィギュレーション実行中に RAM の内容を初期化することができ、指定のない場合、RAM セルはゼロになります。さらに、RAM の内容は、コンフィギュレーション中に誤って書き込まれることがないように保護されています。

グローバル セット/リセットを使用した出力ラッチの初期化

コンフィギュレーション終了後に、Spartan-3 デバイスはスタートアップを開始し、グローバル セット/リセット信号、GSR をアサートして、フリップフロップおよびレジスタの状態を初期化します。ブロック RAM 出力ラッチの値、INIT は非同期で初期化されます。また、GSR 信号が RAM の内容に影響を与えたり、再初期化を行うことはありません。

イネーブル入力

たとえば、EN が Low のようにブロック RAM がディスエーブルである場合、ブロック RAM は現在の値を維持します。イネーブル入力を使用して処理を実行するためには、High にする必要があります。

同期セット/リセットによるデータ出力ラッチの初期化

ブロック RAM がイネーブル (EN が High) で、同期セット/リセット信号が High にアサートされている場合、データ出力ラッチは、次の立ち上がりクロック エッジで初期化されます。SRVAL 属性を使用して、データ出力ラッチの同期セット/リセットを指定しますが、これはグローバル セット/リセット信号、GSR を使用する場合とは異なります。GSR 信号はデバイス全体を制御しますが、同期セット/リセット入力は、特定の RAM ブロックのみに影響を与えます。

同時書き込みおよび同期セット/リセット

同期セット/リセットの実行中に同時書き込みを実行すると、DI および DIP 入力が ADDR 入力によって指定された RAM のアドレスに保存されますが、上記したように、データ出力ラッチは SRVAL 属性で指定された値に初期化されます。

信号がアサートされている場合、クロック エッジごとに読み出しを実行

読み出しは同期で実行されるため、クロック エッジとクロック イネーブルが必要です。データ出力は、書き込みおよび読み出しを同時に実行するかによって異なります。

読み出しと同時に書き込みが実行されている場合、読み出しポートのアドレスが取り込まれ、RAM へのアクセス時間後にそのアドレスに保存されているデータが、出力ラッチに送信されます。

しかし、データの出力と同時に書き込みサイクルが実行されているため、次に示す 3 つの書き込みモードによって、その出力は変化します。

書き込み中の読み出しはデータ出力ラッチに影響

書き込みと同時に読み出しを行う場合、WRITE_MODE 属性を使用して、出力ラッチのデータを決定します（同時書き込み中の読み出し — WRITE_MODE、ページ 14 を参照）。 デフォルトの WRITE_MODE は WRITE_FIRST であり、書き込むのと同時に、入力データで出力ラッチとアドレス指定された RAM のデータを更新します。WRITE_MODE が READ_FIRST の場合、アドレスに保存されているデータを出力ラッチに送信し、DI および DIP の新しい入力データを RAM に保存します。NO_CHANGE モードでは、出力ラッチは書き込みの影響は受けず、データはそのまま維持されます。

一般的な特性

- 書き込み 1 回につき、クロック エッジが 1 つ必要です。
- 読み出し 1 回につき、クロック エッジが 1 つ必要です。
- すべての入力はポート クロックに同期して取り込まれ、setup-to-clock タイミング仕様があります。
- すべての出力は、WE ピンの状態によって、読み出しまたは書き込み中に読み出す 3 つのモードのうちのいずれかになります。また、ポート クロックに関連した出力は、clock-to-out タイミング後に有効です。
- ブロック RAM セルは、同期 RAM メモリであり、アドレスから出力間に組み合わせパスはありません。
- 各ポートは完全に独立し、それぞれのクロック、制御、アドレス、読み出し/書き込み、初期化機能、およびデータ幅を持ちます。
- 出力ポートには、タイミングを自己調整する回路のあるラッチがあり、読み出しにグリッチは発生しません。また、別の読み出し/書き込みを実行するまでは、出力ポートの値は一定です。

ザイリンクス FPGA ファミリでの機能

Spartan-3 FPGA のブロック RAM の機能は、ザイリンクス Virtex-II/Pro FPGA ファミリのブロック RAM と類似しているため、Virtex-II および Virtex-II Pro ブロック RAM をサポートするツールは、同様に Spartan-3 もサポートします。

デュアル ポート RAM の競合とその解消

デュアル ポートのブロック RAM は、2 つのポートで、同時に同じメモリ セルにアクセスできますが、次のような状況では、競合が発生する場合があります。

1. 2 つのポートへのクロック入力が非同期で、セットアップ タイムに違反があると、競合が発生します。
2. 書き込みサイクル中に、2 つのメモリ ポートから RAM の同じ位置に異なるデータを書き込んだ場合、競合が発生します。

3. 1 つのポートの WRITE_MODE を NO_CHANGE または WRITE_FIRST に設定し、書き込みを実行すると、もう一方のポートでの出力データが無効になります。

構成や幅の異なるポート A およびポート B で競合が発生した場合、重複するビットのみが無効になります。

タイミング違反競合

Spartan-3 データシートに指定されている clock-to-clock セットアップ ウィンドウ内では、一方のポートで、あるメモリ セルに書き込みを行うときに、もう一方のポートで、書き込みまたは読み出しのアドレスを指定してはいけません。図 19 に、両ポートが非同期クロックで動作する例を示します。

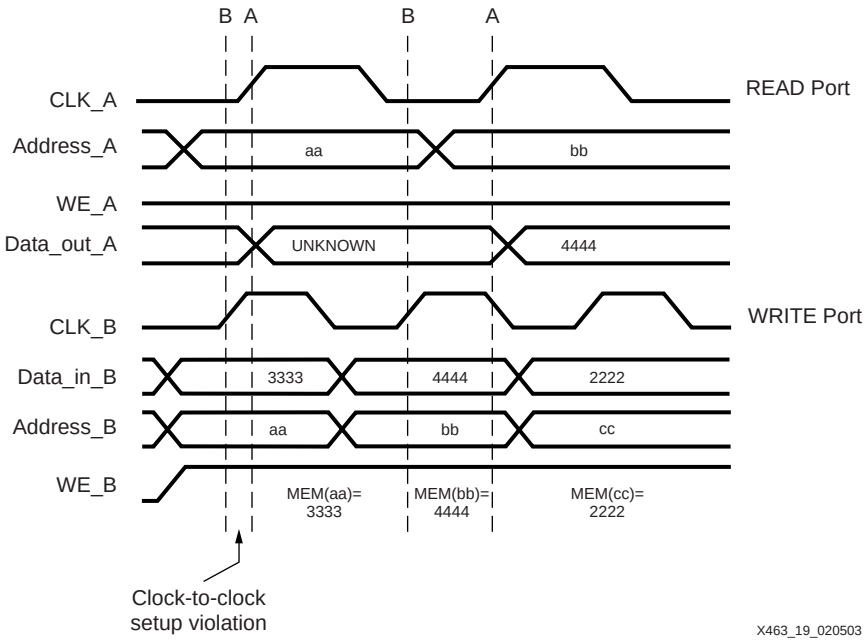


図 19: Clock-to-Clock タイミング競合

CLK_B のクロック エッジからの時間が短いため、CLK_A の最初の立ち上がりエッジはセットアップ パラメータに違反します。Data_in_B、Address_B、および WE_B は、CLK_B の立ち上がりエッジの前に、セットアップ タイムを満たし、ポート B での書き込みは有効になりますが、ポート A での読み出しは無効です。これは、Address_B に書き込まれたデータによって異なる上、書き込みクロック CLK_B から読み出しクロック CLK_A までの時間が短すぎるためです。

CLK_B の 2 つ目の立ち上がりエッジでは、ポート B への別の書き込みが有効で、メモリ アドレス (bb) に 4444 があります。CLK_A は、まだ立ち上がりエッジになっていないので、Data_out_A ポートのデータは無効ですが、CLK_A の 2 番目の立ち上がりエッジでは、(bb) にある新しいデータ 4444 が読み出されます。この際、CLK_B と CLK_A 間に十分なセットアップ タイムがあるため、実行された読み出しは有効です。

両ポートから同時に異なるデータを書き込む場合の競合

表 10 に示しているように、両ポートで同時に同じメモリ セルに異なるデータを書き込む場合、そのセルに書き込まれたデータは無効になります。

表 10: 同一アドレスへ書き込む場合の RAM 競合

入力信号								RAM 内容	
ポート A				ポート B					
WEA	CLKB	DIPA	DIA	WEB	CLKA	DIPB	DIB	パリティ	Data
1	↑	DIPA	DIA	1	↑	DIPB	DIB	?	?

メモ：

1. ADDRA=ADDRB、ENA=1、ENB=1、DIPA ... DIPB、DIA ... DIB、?=不明または無効なデータ

出力ラッチでの書き込み競合

一方のポートが書き込み、もう一方ポートが読み出しを実行すると競合が発生する場合があります。この場合、書き込みは常に成功しますが、書き込みポートの出力ラッチに送信されるデータは、WRITE_MODE 属性により変化します。WRITE_MODE を NO_CHANGE または WRITE_FIRST に設定している場合、出力ラッチのデータは無効になります。表 11 を参照してください。

また、書き込みが READ_FIRST モードの場合は、読み出しポートに影響を与えません。

表 11: WRITE_MODE による出力ラッチでの競合

入力信号								出力信号			
ポート A				ポート B				ポート A		ポート B	
WEA	CLKB	DIPA	DIA	WEB	CLKA	DIPB	DIB	DOPA	DOA	DOPB	DOB
WRITE_MODE_A=NO_CHANGE											
1	↑	DIPA	DIA	0	↑	DIPB	DIB	No Chg	No Chg	?	?
WRITE_MODE_B=NO_CHANGE											
0	↑	DIPA	DIA	1	↑	DIPB	DIB	?	?	No Chg	No Chg
WRITE_MODE_A=WRITE_FIRST											
1	↑	DIPA	DIA	0	↑	DIPB	DIB	DIPA	DOA	?	?
WRITE_MODE_B=WRITE_FIRST											
0	↑	DIPA	DIA	1	↑	DIPB	DIB	?	?	DIPB	DIB
WRITE_MODE_A=WRITE_FIRST、WRITE_MODE_B=WRITE_FIRST											
0	↑	DIPA	DIA	1	↑	DIPB	DIB	?	?	?	?

メモ：

1. ADDRA=ADDRB、ENA=1、ENB=1、?=不明または無効なデータ

競合の解消

両ポートが同じアドレスにアクセスした際に、その調整を行うための専用のモニタはありません。2 つのクロックは、アプリケーションで適切に設定されなければなりませんが、同時に同一のアドレスに書き込みを行うことで、物理的な破損はありません。

ブロック RAM デザイン入力

ザイリンクス CORE Generator システムおよび適切なライブラリ プリミティブを使用した VHDL や Verilog インスタンシエーションを含め、Spartan-3 ブロック RAM の設計には、さまざまなツールを使用できます。

ザイリンクス CORE Generator システム

図 5 に示すように、ザイリンクス CORE Generator システムは、シングルおよびデュアル ポート ブロック メモリ モジュールの両方を作成できるツールです。どちらのモジュールを作成した場合でも、選択した制御信号によって RAM、ROM、書き込みのみのファンクションをサポートし、そのアーキテクチャで作成可能なすべてのメモリをサポートします。

これらモジュールは、ほとんどの CORE Generator モジュールと同様にパラメータを指定することができます。モジュールを作成する際は、コンポーネント名を指定し、制御信号の有無を選択した後、制御信号の極性を決定します。また、デュアル ポート ブロック メモリを作成する際に、メモリ構成やポート A のアスペクト比を決定すると、ポート B に対して有効なオプションのみが表示されます。

オプションとしてメモリの初期値を指定できますが、特に指定のない場合はゼロになります。メモリ初期化ファイルで、その初期値を指定しますが、このファイルでは、各メモリ アドレスごとに 1 行のバイナリ データを記述します。デフォルトのファイルが CORE Generator システムによって生成される代わりに、COE ファイル (.coe) を作成する必要があります。この COE ファイルでは、初期値の基数 (2、10、16) だけでなく、CORE Generator 用のその他の制御パラメータをすべて定義します。

CORE Generator からの出力は、選択したオプションおよび必要なデバイス リソースについてのレポートを含みます。また、ワード数の非常に多いメモリを作成すると、外部にマルチプレクサが必要な場合があり、これらのリソースは必要なロジック スライスとしてレポートされます。さらに、まだ 100% は使用されていないブロック RAM にある使用可能なビット数もわかります。CORE Generator を使用して、シミュレーション用に VHDL または Verilog ビヘイビア モデルを作成できます。

- CORE Generator: シングル ポート ブロック メモリ モジュール (RAM または ROM)
- CORE Generator: デュアル ポート ブロック メモリ モジュール (RAM または ROM)

VHDL および Verilog のインスタンシエーション

VHDL および Verilog で記述された合成を基本とするデザインは、ブロック RAM を推論または直接インスタンシエートできますが、これは使用したロジック合成ツールによって異なります。

ブロック RAM の推論

ザイリンクス合成ツール (XST) または Synplicity Synplify のような VHDL および Verilog ロジック合成ツールは、記述されたハードウェアに基づいてブロック RAM を推論します。ザイリンク ISE Project Navigator には、ブロック RAM 推論用のテンプレートが含まれます。Project Navigator でこのテンプレートを使用するには、まずメニューで Edit から Language Templates を選択します。次に VHDL または Verilog を選択し、その下に表示される Synthesis Templates から RAM を展開表示し、適切なブロック RAM テンプレートを選択します。

デザインに推論したブロック RAM がある場合でも、直接インスタンシエートすることも可能です。

インスタンシエーション テンプレート

デザインのスピードを向上するために、VHDL および Verilog で記述されたデザインで使用できる、さまざまなインスタンシエーション テンプレートがあります。ザイリンクス ISE Project Navigator のメニューで Edit から Language Templates を選択します。次に VHDL または Verilog を選択し、その下に表示される Component Instantiation を展開表示し Block RAM を選択します。

付録に VHDL および Verilog でブロック RAM を推論するためのコード例を示します。

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する必要があります。アプリケーションで使用される信号名は、アーキテクチャ セクションのポートマップにある必要があります

SelectRAM_Ax テンプレート (x = 1、2、4、9、18、36) はシングル ポート モジュールであり、対応する RAMB16_Sx モジュールをインスタンシエートします。

SelectRAM_Ax_By テンプレート (x = 1、2、4、9、18、36 および y = 1、2、4、9、18、36) はデュアル ポート モジュールであり、対応する RAMB16_Sx_Sy モジュールをインスタンシエートします。

VHDL および Verilog コードの初期化

ブロック RAM メモリは、合成およびシミュレーション用に VHDL または Verilog コードで初期化できます。合成を行う場合、ブロック RAM のインスタンシエーションに属性を指定すると、その属性は、ザイリンクス Alliance シリーズ ツールでコンパイルされる EDIF 出力ファイルにコピーされます。VHDL コード シミュレーションは、**`generic`** パラメータを使用して属性を渡します。Verilog コード シミュレーションは、**`defparam`** パラメータを使用して属性を渡します。

これらの例は、付録の VHDL および Verilog コードに示されています。

ブロック RAM アプリケーション

一般的に、ブロック RAM はローカルで保存アプリケーションとして使用されますが、次のセクションでは、あまり知られてはいませんが、使用方法によっては非常に有効となるブロック RAM の追加機能について紹介します。

大型 RAM 構造の作成

ブロック SelectRAM の列は特別な方法で配線されるため、最小の配線遅延でカスケード接続することが可能です。したがって、少ない遅延で、ビット数およびワード数の多い RAM を作成できます。

Read-Only Memory (ROM) としてのブロック RAM

ライト イネーブル入力を Low にすると、オプションとして、ブロック RAM をレジスタ付きブロック ROM として使用できます。同期し、クロック入力を必要とする ROM の出力は、ブロック RAM の読み出しとほぼ同様に動作します。ROM の内容はデザインの初期値によって指定され、デザインのコンパイル後に Data2BRAM ユーティリティを使用して更新できます。

FIFO

エラスティックにデータを保存するものとして知られている First-In First-Out (FIFO) は、任意にデータを保存する以外の目的で使用するブロック RAM の最も一般的なアプリケーションです。FIFO は、2 つの異なるクロック ドメイン間のデータ、または、同じクロックで動作していますが、レートの異なるデータを再同期するのが一般的です。ザイリンクス CORE Generator システムでは、パラメータ指定可能な 2 つの FIFO モジュールを作成できます。1 つは読み出し/書き込みクロックが互いに同期している同期 FIFO であり、もう 1 つは読み出し/書き込みクロックが異なる非同期 FIFO です。

アプリケーション ノート XAPP261 では、FIFO の読み出し/書き込みポートへの異なるデータ幅の設定について説明しています。

アプリケーション ノート XAPP291 では、連続したデータ ストリームでのデータ スロットリングに有効な自動アドレッシング FIFO について説明しています。

- CORE Generator: 同期 FIFO モジュール
- CORE Generator: 非同期 FIFO モジュール
- XAPP258: ブロック *RAM* を使用した *FIFO* (リファレンス デザインを含む)
- XAPP261: ブロック *RAM* を使用したデータ幅変換 *FIFO* (リファレンス デザインを含む)
- XAPP291: 自動アドレッシング *FIFO*

エンベデッド プロセッサ向けストレージ機能

ブロック RAM は、エンベデッド プロセッサで次に示すような機能を果たし、アプリケーションをより効果的にすることができます。

- プロセッサ レジスタ セット用のレジスタ ファイル ストレージ (分散 RAM の方が適しているプロセッサもあり)
- スタック ベースのアーキテクチャおよびスタック コール用のスタックまたは LIFO
- 高速ローカル コード ストレージ (内部ブロック RAM に高速にアクセスすることによって、エンベデッド プロセッサの処理能力は向上。ただし、チップ上で使用できるブロック RAM 数には制

限あり)

- 外部プロセッサまたは DSP デバイスと共有して使用される大型のデュアル ポート メールボックス メモリ
- アプリケーションのデバッグを容易にする一時的なトレース バッファ (循環バッファ、シフト レジスタ、および遅延ライン を参照)

ビットストリームの直接修正によるブロック RAM/ROM の内容更新

一般的なデータ フローでは、デザインの段階でブロック RAM/ROM の内容が決定し、デバイスのビットストリームにコンパイルされ、そのビットストリームが Spartan-3 FPGA にダウンロードおよびコンフィギュレーションされます。

しかし、アプリケーションによっては、ビットストリームが作成される段階では実際のメモリ内容が不明な場合や後に変更される場合があります。Spartan-3 に組み込まれたプロセッサが、プログラム コードを格納するためにブロック RAM を使用する場合が、その一例です。ザイリンクスの Data2BRAM を使用すると、既存の FPGA ビットストリームを新しいブロック RAM/ROM 内容で更新でき、コードの変更のために FPGA デザインすべてを再コンパイルする必要がなくなります。

図 20 に示すように、Data2BRAM への入力は、エンベデッド プロセッサ コンパイラ/リンカからの出力、現在の FPGA ビットストリーム、およびシステム アドレス スペースと各ブロック RAM で使用されるアドレッシング間のマッピング情報と各ブロック RAM の物理的な位置などの RAM に関する新しいデータを含みます。

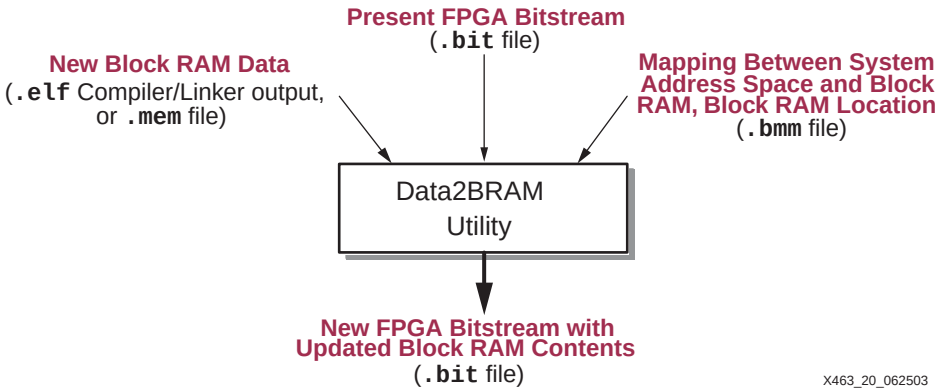


図 20: ビットストリームで Data2BRAM ユーティリティを使用したブロック RAM の更新

1 ブロック RAM を 2 つの独立したシングル ポート RAM として使用

アプリケーションによっては、デバイス上にある RAM ブロックよりさらに多くのシングル ポート RAM が必要な場合がありますが、シングル ブロック RAM を 2 つの完全に独立したシングル ポート メモリとして使用することで、デバイス上の RAM ブロック数を効果的に 2 倍にできます。ただし、各 RAM ブロックのサイズは、元のブロックの半分になり、最大で 9K ビットです。

図 21 に 1 ブロック RAM から 2 つの独立したシングル ポート RAM を作成する方法を示します。1 つのポートのアドレス MSB を High に、もう一方のポートは Low に接続し、ポート間で RAM を均等に分割します。

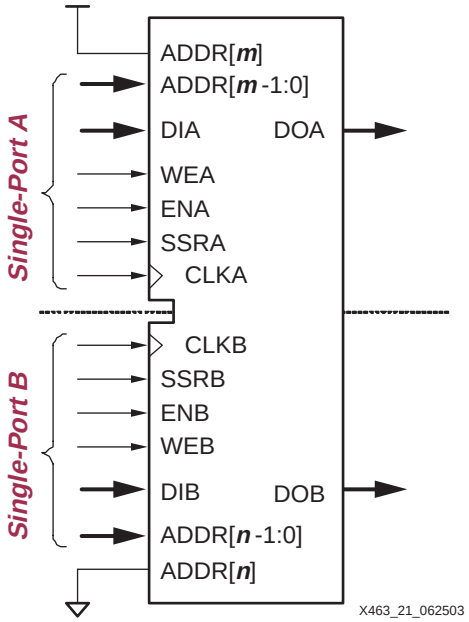


図 21: 1 ブロック RAM から 2 つの独立したシングル ポート RAM を作成

両ポートには、それぞれのメモリ構成、データ出入力、クロック入力、および制御信号があり、完全に独立しています。たとえば、ポート A が 256x36 で、ポート B が 2Kx4 のメモリを作成できます。

図 21 では使用可能なメモリを 2 つのポートで均等に分割しています。上のアドレス ラインにロジックを追加すると、異なる比率でメモリを分割できます。

1 ブロック RAM を使用して 256x72 シングル ポート RAM を作成

図 22 にブロック RAM を使用した、256×72 シングル ポート RAM の作成方法を示します。前述した例と同様に、メモリ アレイは二分割されます。それぞれが 36 ビットであり、効果的に 72 ビット幅のメモリを作成します。

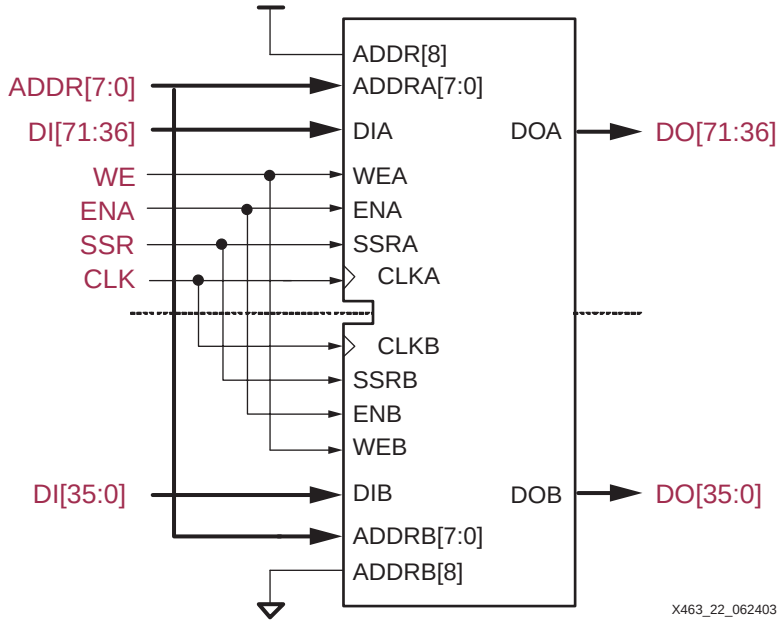


図 22: 1 ブロック RAM を使用して 256x72 シングル ポート RAM を作成

アドレス ラインの MSB である ADDR[8] は、1つのポートで High に接続され、もう一方では Low に接続されます。また、両ポートは、アドレス入力、制御入力、およびクロック入力を共有します。

循環バッファ、シフト レジスタ、および遅延ライン

循環バッファは、Finite Impulse Response (FIR) フィルタ、マルチ チャネル フィルタ、さらに相関関数や相互相関関数のような多様なデジタル信号処理アプリケーションに使用されます。また、データを遅らせ、ほかのデータと再同期するために循環バッファを使用することも可能です。

図 23 は循環バッファの概図です。データがバッファに書き込まれた ***n*** クロック サイクル後に、そのデータが送信され、同じ位置に新しいデータが書き込まれます。

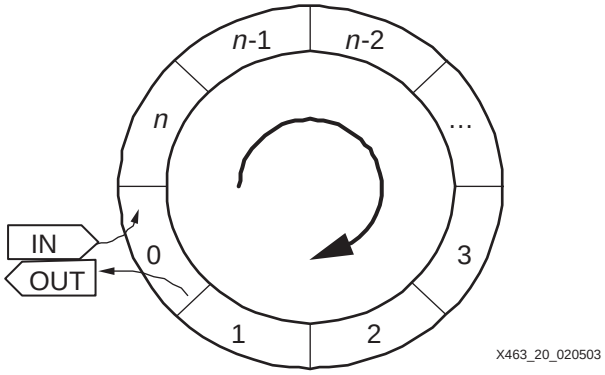


図 23: 循環バッファ

図 24 に、ブロック RAM を使用して循環バッファを作成する場合のハードウェア インプリメンテーションを示します。モジュロ ***n*** 計算器によって、シングル ポート ブロック RAM にアドレスが入力され、単純なデータ遅延ラインを使用して、ブロック RAM はクロック サイクルごとに新しいデータを書き込みます。

また、循環バッファは、クロック エッジごとに遅延データを読み出します。ブロック RAM の書き込みモードを READ_FIRST に設定すると、同じクロック入力およびクロック エッジで入力データが書き込まれ、出力データが送信されるため、デザインが簡潔化し、全体のパフォーマンスも向上します。実際の書き込み/読み出しについては図 17 に示します。

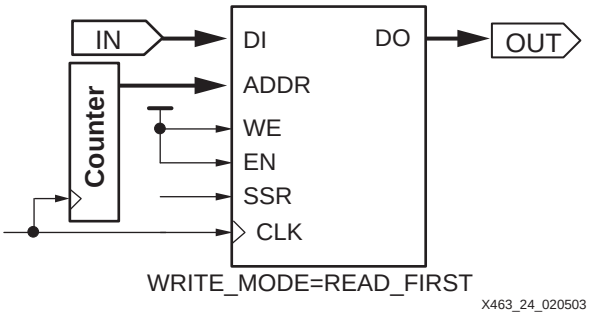


図 24: ブロック RAM およびカウンタを使用した循環バッファのインプリメンテーション

図 24 では IN および OUT データ ポート幅は一致していますが、必ずしも同一である必要はなく、デュアル ポート モードでは各ポート幅が異なる場合があります。図 25 には、1 バイトのデータがブロック RAM に入り、32 ビット ワードのデータとして出力される例を示しています。なお、最大 2,048 クロック サイクルまでデータを遅らせて送信できます。

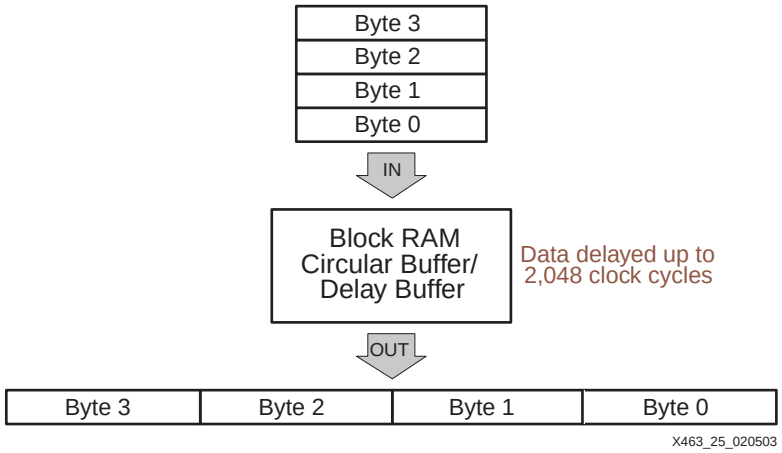


図 25: シングル ブロック RAM で使用する循環バッファとポート幅変換器

シングル ブロック RAM はデュアル ポート メモリとしても使用できます。1 バイトの入力データが、2k ×9 にコンフィギュレーションされたポート B に入り、ポート A には 32 ビットの出力データが送信されるため、結果として、ポート A は 512x36 メモリにコンフィギュレーションされます。

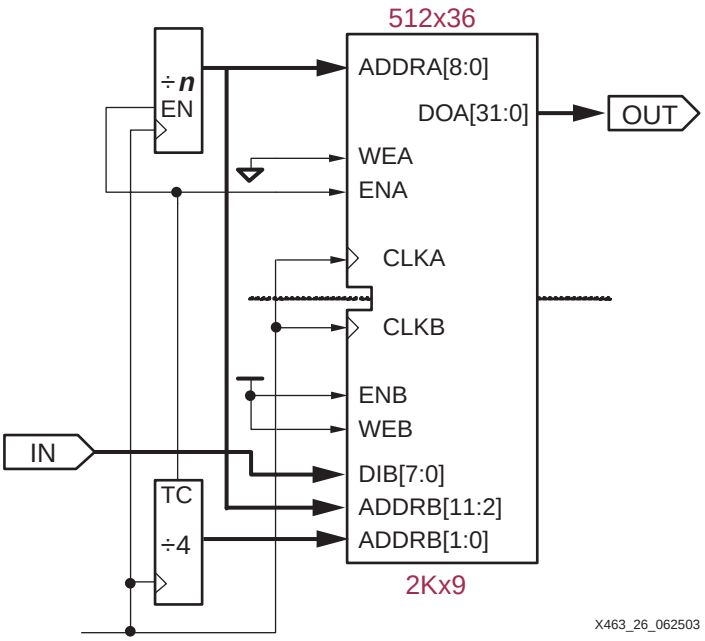


図 26: 1 バイト幅のデータが 4n クロック サイクル後に 32 ビット データに変換

両ポートに入力されるアドレスを操作することによって、4***n*** バイト クロック遅れて出力できます。32 ビットの出力ワードごとに 4 つの 1 バイト入力が必要であり、4 分割カウンタは、2 つの下位アドレス ビット ADDRB[1:0] を入力します。4 バイトが保存された後、下にあるカウンタからのターミナル カウント (TC) がポート A と ***n*** 分割カウンタをイネーブルにします。イネーブル信号によって、ポート B では 32 ビットの出力データがラッチに送信され、上部のカウンタはインクリメントされます。このように、4 分割カウンタと ***n*** 分割カウンタを組み合わせると、効果的に 4***n*** 分割カウンタとして使用できます。***n*** 分割カウンタからの出力は、ポート B のアドレス ビットの MSB である ADDRB[11:2] およびポート A のアドレス全体のビットである ADDRA[9:0] になります。

高速で複雑なステート マシンおよびマイクロシーケンサ

ブロック RAM の初期値は任意に指定できるため、このメモリを使用し、ステート マシーンとして機能するデュアル ポートのレジスタ付き ROM を作成できます。たとえば、図 27 に示すように、1 つのブロック RAMを使用し、128 ステートおよび 8 ブランチで、38 ステート出力の有限ステートマシンを作成できます。

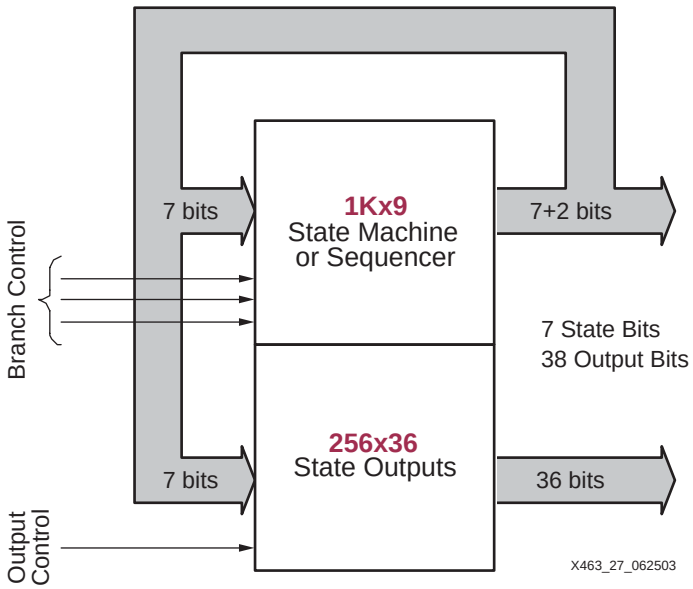


図 27: シングル ブロック RAM で作成できる 128 ステートおよび 38 出力の有限ステート マシン

図 21 と同様に、デュアル ポート ブロック RAM メモリは、1 つのポートのアドレス ビットの MSB を High に接続し、もう一方を Low に接続することで、サイズが半分の、完全に独立した 2 つのシングル ポート メモリとして使用できます。ポート A は、2Kx9 にコンフィギュレーションされ、1K x 9 シングル ポート ROM として使用できます。ここで、7 出力がアドレスとして再入力し、128 のステートを持ちますが、1Kx9 ROM にある 10 アドレス ラインのうち、7 つが現在のステート入力となり、残りの 3 アドレス入力は 8 つのブランチを決定します。128 ステートは、これらの 3 アドレス入力に制御され、一定条件の下で 8 つの新しいステートのうち、どのステートにも進むことができます。

ポート B は、512 x 36 にコンフィギュレーションされ、256 x 36 シングル ポート ROM として使用できます。ポート A から現在のステート アドレスが 7 ビットで入力され、36 ビットが出力されます。出力は各ステートによって任意に定義され、異なりますが、ブロック ROM は本来同期メモリであるため、256x36 ROM からの 36 出力は 1 クロック サイクル遅れています。また、8 アドレス入力になると、36 出力も変化します。1Kx9 ブロックから 2 ステート ビットを追加できますが、これらビットは 1 クロック サイクル分の遅延はありません。

同様の基本的なアーキテクチャを使用すると、4 ブランチの 256 ステートの有限ステート マシン、または 16 ブランチの 64 ステートのステート マシンを作成でき、ブランチ制御信号がさらに必要な場合は、マルチプレクサを使用できます。このデザインの利点としては、低コスト (シングル ブロック RAM) であること、高性能であること (125+MHz)、配置配線の問題がないこと、および任意に設計できることが考えられます。

RAM を使用した高速カウンタ

カウンタは、現在のステートによって次のステートが変化する単純なステート マシンの一例です。たとえば、バイナリ カウンタは現在のステートをインクリメントし、次のステートを生成します。図 28 にクロック イネーブルと同期リセットがシングル ブロック RAM にインプリメントされている 20 ビット バイナリ カウンタを示します。

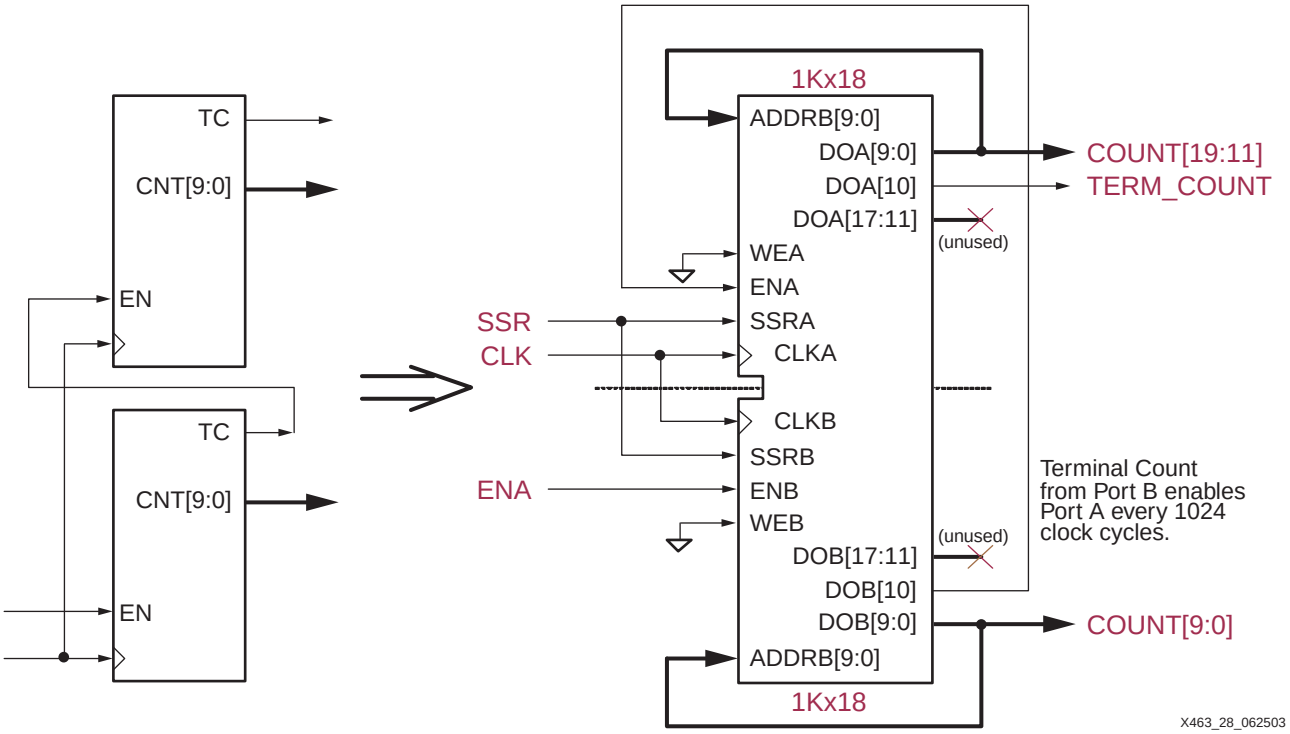


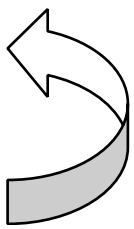
図 28: シングル ブロック RAM を使用し、2 つの 10 ビット カウンタから 1 つの 20 ビット バイナリ カウンタを作成

20 ビットのバイナリ カウンタは、2 つの同一 10 ビット バイナリ カウンタから作成できます。これらのカウンタは、1024 クロック サイクルごとに下位の 10 ビット カウンタが上位の 10 ビット カウンタをイネーブルにするものです。この例では、ポート B は下位の 10 ビット カウンタを作成する 1Kx18 ROM (WEB は Low) で、現在のステートに対応した 10 LSB データ出力は、そのまま 10 ビットのアドレス入力 ADDRB[9:0] になります。ROM では、アドレス ピンに送信された現在のステートを使用して、次のステートが固定されます。11 番目のデータである D[10] が、カウンタからのターミナル カウント出力になり、この例では、データ ビット DOB[17:11] は使用されません。

表 12 にバイナリ カウンタのステート ロジックを示します。カウンタは、0 または INIT や SRVAL 属性で指定した値から開始し、ターミナル カウントがアクティブで、コントローラが 0 に戻る 0x3FF (1023 10進) までカウントします。

表 12: バイナリ アップ カウンタの次のステート ロジック

現在のステート	ステート出力	次のステート
	TC	COUNT
ADDR[9:0] (Hex)	D[10]	D[9:0] (Hex)
0	0	1
1	0	2
2	0	3
...	...	...
3FFF	1	0

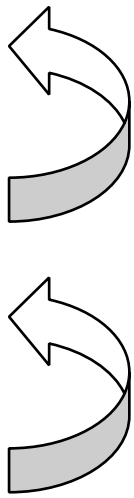


ポート A は、ポート B からのターミナル カウンタ出力によってイネーブルになる点とその 10 ビット カウンタのインクリメント率がポート B の 1/1024 番目 である点を除いて、ポート B と同様です。

この例に簡単な修正を加えることで、20 ビット アップカウンタから 18 ビット アップ/ダウン カウンタを作成できます。アドレス信号の MSB を方向制御に使用すると、表 13 に示すように、同様のカウンタ アーキテクチャで、インクリメントまたはデクリメントできます。この場合、カウンタは、アップ/ダウン コントロール信号が Low でインクリメントし、High でデクリメントします。ROM メモリは、次のステート ロジックのインクリメントとデクリメントの間で分割されます。

表 13: バイナリ アップ/ダウンカウンタのステート ロジック

アップ/ダウン制御	現在のステート	ステート出力	次のステート
		TC	COUNT
ADDR[9]	ADDR[8:0] (Hex)	D[10]	D[9:0] (Hex)
0 (Up)	0	0	1
	1	0	2
	2	0	3
	...	...	...
	1FFF	1	0
1 (Down)	1FFF	0	1FFE
	1FFE	0	1FFD
	1FFD	0	1FFC
	...	...	...
	0	1	1FFF



次に例を示すようなカウンタとしても使用できます。

- ポート A および B でインプリメントされたカウンタのモジュロを組み合わせたバイナリ アップおよびアップ/ダウン カウンタ
- 高速グレイ コード カウンタを含む、その他のインクリメントとデクリメント パターンを持ったカウンタ
- 512x36 ブロック ROM と 1 つの CLB にコンフィギュレーションされた 6 桁の BCD カウンタ

4 ポート メモリ

各ブロック RAM は物理的にデュアル ポート メモリですが、ブロック RAM のアクセス スピードは高速であるため、メモリへの出入力信号を時分割処理し、複数ポートのメモリを作成できます。ブロック RAM にいくつかのロジックを追加すると 4 ポートまでサポートすることができますが、各ポートは、さらにアクセス レイテンシを持つことになります。詳細およびリファレンス デザインについては、次のアプリケーション ノートを参照してください。

- XAPP228: *Virtex* デバイスでの *4* ポート メモリ (リファレンス デザインを含む)

CAM (連想メモリ)

連想メモリとしても知られる Content-Addressable Memory (CAM) は、多様なネットワークおよびデータ プロセッシング アプリケーションで使用されます。ほとんどの場合、メモリ アプリケーションの内容はアドレスによって参照されますが、CAM アプリケーションでは、メモリ内容が入力となり、その出力によってメモリ内にデータがあるかどうかがわかり、ある場合にはロケーションも出力されます。

ここで CAM の動作についてより理解するために、本のインデックスを思い浮かべてみてください。あるアイテムを検索する場合に、まず、これがインデックスに含まれているかを確認します。ある場合には、そこに記載されているページ番号などから内容を参照します。

- CORE Generator: Content-Addressable Memory モジュール
- XAPP260:『ブロック RAM を使用した高性能読み出し/書き込み CAM』
- XAPP201:『CAM デザインの概要』このアプリケーション ノートは Virtex/E および Spartan-II/E アーキテクチャ用に書かれていますが、Spartan-3 でも有用な手法が記載されています。

ブロック RAM を使用したロジック ファンクション

すべての Spartan-3 ロジック セルには、ルック アップ テーブル (LUT) と呼ばれる 4 入力 RAM/ROM があり、この 4 入力のロジック ファンクションとして機能する LUT が、Spartan-3 アーキテクチャの基本となります。

また、RAM は入力数の多い LUTとして別の使用方法があります。メモリ構成の 1 つとして、RAM を ROM として使用し、14 入力 1 出力のブロック RAM も作成できます。このように、ブロック RAM は、入力信号の複雑さや反転機能に関係なく、14 入力までの任意のロジック ファンクションをインプリメントできますが、その際に、いくつかの制限があります。

- ロジック内では、ラッチを作成するような非同期フィードバック パスを使用できません。
- 出力は、クロック入力と必ず同期である必要があります。ブロック RAM は、非同期読み出し出力をサポートしません。

これらの条件を満たしたロジック ファンクションでは、シングル ブロック RAM で次のような機能をインプリメントできます。

- 最大 14 入力までのブール ロジック関数
- 11 入力を共有して使用する 9 つのブール ロジック関数
- その他の組み合わせも可能ですが、入力数、共有入力数、またはロジック ファンクションの複雑さなどに制限がある場合があります。

CLB ロジックの柔軟性とスピードを考慮すると、ブロック RAM は、複数入力を AND ゲートで接続するアドレス デコーダのような単純なロジック ファンクションには、必ずしも効果的ではありませんが、主要なデコーダ、相関器のような複雑なロジック ファンクションにとっては、高速で非常に有効です。

ファジー マッチ回路例

図 29 に、正確に一致した回路とほぼ一致した回路例を示しています。各入力ビットは必要な MATCH パターンに一致し、ロジック的に必要ではないビットはマスク オフするため、特定のビットは常に一致していることになります。このように一致したビット数はカウントおよび起動しきい値と比較され、この数がしきい値より大きい場合、入力データはほぼ必要なパターンに一致し、MATCH 出力は High になります。

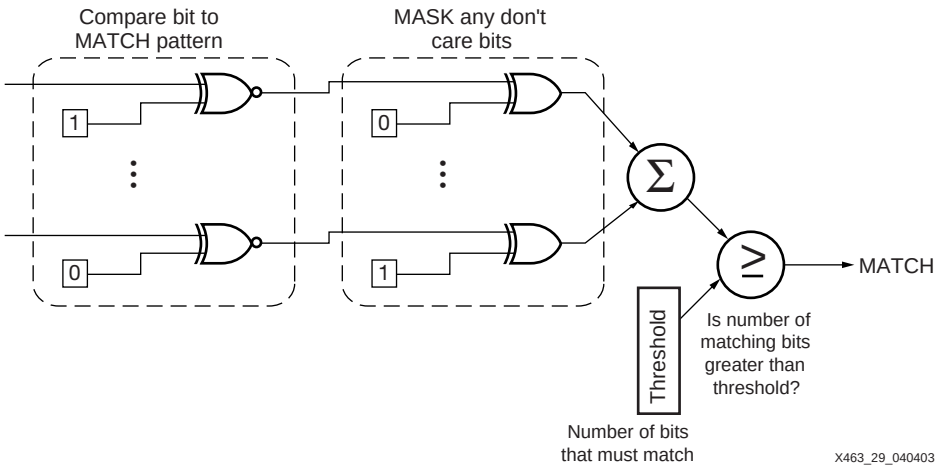


図 29: シングル ブロック RAM での 14 入力 ファジー マッチング回路

新規の一致パターンや異なるロジック ファンクションが必要な場合は、もう一方のメモリ ポートを使用して取り込むことができます。

CLB ロジックで使用されるため、このファンクションには多くのロジック セルと複数レイヤが必要となりますが、MATCH、MASK、およびしきい値はあらかじめわかっているため、ファンクションを事前に計算し、ブロック RAM に置くことが可能です。たとえば、各入力がアドレス 0 から開始し、メモリ全体にインクリメントされる場合、この出力はあらかじめ計算できます。14 入力のファジーマッチング回路には、シングル ブロック RAM が必要であり、1 クロック サイクルで実行されます。

MAP -bp オプションを使用したブロック RAM へのロジック マッピング

ザイリンクス ISE ソフトウェアでは、ロジック ファンクションはブロック RAM に自動的にマップされないため、マッピング オプションを使用します。

このブロック RAM のマッピングに有効なオプションが、MAP -bp オプションです。オプションが有効な場合、ザイリンクス ISE のロジック マッピング ツールは、LUT およびそれに接続されたフリップフロップを 1 出力のシングル ポート ブロック RAM に配置します。最終的なフリップフロップ出力は、ブロック RAM が同期であるため、レジスタ付きの出力になります。また、マッピング ソフトウェアは、フリップフロップと LUT ロジックが駆動するものをパックします。LUT ロジックなしで、レジスタをブロック RAM にパックすることはできず、逆に LUT ロジックがある場合は、必ずパックされます。

ブロック RAM 出力に変換されるレジスタ出力を指定するには、レジスタ出力に接続されたネット名のリストを含むファイルを作成します。 環境変数 XIL_MAP_BRAM_FILE をファイル名に設定することで、マッピング ソフトウェアは作成したファイルを使用し、オプションが指定されると、この環境変数を検索し MAP が実行されます。ファイルにある出力ネットのみがブロック RAM 出力に変換されます。

- PC の場合:

  ```
  set XIL_MAP_BRAM_FILE=file_name
  ```

- UNIX の場合:

  ```
  setenv XIL_MAP_BRAM_FILE file_name
  ```

ブロック RAM を使用した波形ストレージ、ファンクション テーブル、Direct Digital Synthesis (DDS)

サインやコサインという三角関数のような関数を含む波形ストレージもブロック RAM のすぐれたアプリケーションの 1 つです。 サインおよびコサインは、DDS のようなアプリケーションの出力波形を作成する際の基本波形となります。ザイリンクス CORE Generator システムには、次の 2 つのパラメータ指定可能なモジュールがあります。

- CORE Generator: Sine/Cosine Look-Up Table モジュール
- CORE Generator: Direct Digital Synthesizer (DDS) モジュール

波形ストレージは、さまざまな信号コンパンダ (コンプレッサ/エキスパンダ) および正規化回路内で、必要なバンド幅で重要な信号をブーストするために使用されるアプリケーションとしても利用可能です。例として、直線データ、u-Law でエンコードされたデータおよび A-Law でエンコードされデータ間の変換器があり、一般にテレコミュニケーションで使用されます。

デュアル ポートのブロック RAM によって、波形ストレージが容易になるだけでなく、完全な新規データ、または修正や正規化されたデータで波形を更新することができます。図 30 の例では、ポート A に現在アクティブな波形があり、ポート B で新しい波形を取り込みます。

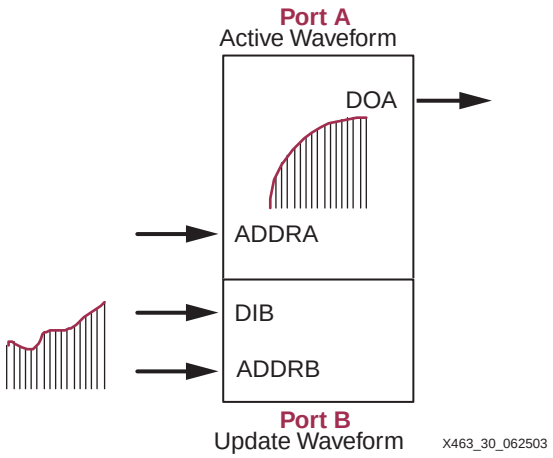


図 30: デュアル ポート ブロック RAM を使用した波形ストレージおよび更新

エンジニアリングの世界で、答えを導き出すよりも探す方が容易な場合があるように、デジタル デザインでも同様のことが言えます。ブロック RAM は、出力 y が 入力 x または y=f(x) の関数である場合、あらかじめ計算された関数テーブルの保存に役立ちます。

たとえば、次の多項式をインプリメントする CLB ロジックを作成する代わりに、この関数を計算し、ブロック RAM に保存できます。

$$Y = Ax^3 - Bx^2 + Cx + D$$

A、B、C、および D の値は、すべて定数であり、出力 y は、入力 x によってのみ変化します。したがって、出力値は各入力値 x ごとにあらかじめ計算でき、メモリに保存されます。ただし、入力 x の範囲または出力 y の大きさによっては、1 つのロジック ブロックでは十分でない場合があります。512x36 ブロック RAM を使用し、入力が 0 から 511 の範囲で、上の等式をインプリメントする場合、x の値は指数乗で y に影響を与えるため、その範囲に限界があります。この例で、x が最大値をとると、y は少なくとも 28 出力ビットが必要になります。

同様に、シングル ブロック RAM では、次のようなファンクションを実現できます。

- log(x) や平方根(x) のような関数の組み合わせを含む、1 入力の複雑な関数。2 つの値の乗算器も可能ですが、一般にブロック RAM 入力数による制限があり、単純な乗算関数には、Spartan-3 の 18x18 エンベデッド乗算器がよりすぐれたソリューションです。
- 1Kx18 ブロック RAM を使用した 11 ビット バイナリから 4 桁 BCD への 2 つの独立した変換器。各コンバータの LSB は、ROM をバイパスし、その結果が偶数か奇数かといったオリジナルの値と同一かを示します。
- 2Kx9 ブロック RAM を使用した 2 つの独立した 3 桁 BCD から 10 ビット バイナリへのコンバータ。LSB は変換器をバイパスします。
- 一方のポートをサインに、もう一方をコサインに使用し、90 度シフトしたアドレス、18 ビットの振幅、10 ビットの角度をもつサイン-コサイン LUT。
- 1Kx18 ブロック RAM での 10 ビット バイナリ から 3 桁、7 セグメント LED 出力への 2 つの独立した変換器。ただし、ゼロはブランクで表示されます。入力が、1023 までに限定されているため、LED の表示は 0 から 3FF までとなり、MSB のロジックには、4 入力 (セグメント a=d=g、セグメント f はつねに High) のみが必要となります。

関連文献および情報

- ìUsing Leftover Multipliers and Block RAM in Your Designî Peter Alfke, Xilinx, Inc.
  http://www.xilinx.co.jp/support/techxclusives/leftover-techX11.htm
- ìThe Myriad Uses of Block RAMî Jan Gray, Gray Research, LLC.
  http://www.fpgacpu.org/usenet/bb.html
- ザイリンクス ISE 5.2i **ライブラリ ガイド**
  - *Adobe Acrobat [PDF]*
    http://toolbox.xilinx.com/docsan/xilinx5/pdf/docs/lib/lib.pdf、1593・ 640 ページ

    ISE 5.2i をデフォルトのディレクトリにインストールしている場合、このライブラリ ガイドは、次のパスまたは、Project Navigator で**ヘルプ**から**オンライン マニュアル**を選択すると参照できます。Acrobat でドキュメントが表示されたら、左側にある目次で Libraries Guide をクリックしてください。
    `C:\Xilinx\doc\usenglish\docs\lib\lib.pdf`
  - `RAMB16_Sn` *プリミティブ [HTML]*
    http://toolbox.xilinx.com/docsan/xilinx5/data/docs/lib/lib0371_355.html
  - `RAMB16_Sm_Sn` *プリミティブ [HTML]*
    http://toolbox.xilinx.com/docsan/xilinx5/data/docs/lib/lib0372_356.html

おわりに

Spartan-3 FPGA には、豊富、高速、かつ柔軟なブロック RAM があり、これは、オンチップのストレージとして、スクラッチパッド メモリ、FIFO、バッファ、LUT などに非常に有益です。ブロック RAM 独自の機能を使用すると、シフト レジスタ、遅延ライン、カウンタ、および他入力で複雑なロジック ファンクションをインプリメントできます。

ブロック RAM は、CORE Generator を含むザイリンクス ISE 開発ソフトウェアの広域スペクトルを使用するアプリケーションでサポートされ、合成の際に、VHDL または Verilog で推論や直接インスタンシエートすることができます。

付録 A: VHDL インスタンシエーション例

次は、Synopsys FPGA Express システムの場合の、VHDL インスタンシエーション例です。この例にある XC3S_RAMB_1_PORT **モジュールは、SelectRAM_A36.vhd** VHDL テンプレートを使用します。また、このテンプレートを含むその他のテンプレートは、次のウェブ サイトからダウンロード可能です。ここで示している VHDL コードは、断片的であり、完全なものではありません。

- ftp://ftp.xilinx.com/pub/applications/xapp/xapp463_vhdl.zip

```
-- Module: XC3S_RAMB_1_PORT
-- Description: 18Kb Block SelectRAM example
-- Single Port 512 x 36 bits
-- Use template “SelectRAM_A36.vhd"
--
-- Device: Spartan-3 Family
----------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
--
-- Syntax for Synopsys FPGA Express
-- pragma translate_off
library UNISIM;
use UNISIM.VCOMPONENTS.ALL;
-- pragma translate_on
--
entity XC3S_RAMB_1_PORT is
port (
    DATA_IN     : in std_logic_vector (35 downto 0);
    ADDRESS     : in std_logic_vector (8 downto 0);
    ENABLE      : in std_logic;
    WRITE_EN    : in std_logic;
    SET_RESET   : in std_logic;
```

```
    CLK         : in std_logic;
    DATA_OUT    : out std_logic_vector (35 downto 0)
);
end XC3S_RAMB_1_PORT;
--
architecture XC3S_RAMB_1_PORT_arch of XC3S_RAMB_1_PORT is
--
-- Components Declarations:
--
component BUFG
port (
    I : in std_logic;
    O : out std_logic
);
end component;
--
-- Syntax for Synopsys FPGA Express
component RAMB16_S36
-- pragma translate_off
generic (
-- "Read during Write" attribute for functional simulation
WRITE_MODE : string := "READ_FIRST" ; -- WRITE_FIRST(default)/ READ_FIRST/
NO_CHANGE
-- Output value after configuration
INIT : bit_vector(35 downto 0) := X"000000000";
-- Output value if SSR active
SRVAL : bit_vector(35 downto 0) := X"012345678";
-- Initialize parity memory content
INITP_00 : bit_vector(255 downto 0) :=
X"0000000000000000000000000000000000000000000000000FEDCBA9876543210";
INITP_01 : bit_vector(255 downto 0) :=
X"0000000000000000000000000000000000000000000000000000000000000000";
... (snip)
INITP_07 : bit_vector(255 downto 0) :=
X"0000000000000000000000000000000000000000000000000000000000000000";
-- Initialize data memory content
INIT_00 : bit_vector(255 downto 0) :=
X"0000000000000000000000000000000000000000000000000FEDCBA9876543210";
INIT_01 : bit_vector(255 downto 0) :=
X"0000000000000000000000000000000000000000000000000000000000000000";
... (snip)
INIT_3F : bit_vector(255 downto 0) :=
X"0000000000000000000000000000000000000000000000000000000000000000"
);
-- pragma translate_on
port (
    DI    : in std_logic_vector (31 downto 0);
    DIP   : in std_logic_vector (3 downto 0);
    ADDR  : in std_logic_vector (8 downto 0);
    EN    : in STD_LOGIC;
    WE    : in STD_LOGIC;
    SSR   : in STD_LOGIC;
    CLK   : in STD_LOGIC;
    DO    : out std_logic_vector (31 downto 0);
    DOP   : out std_logic_vector (3 downto 0)
);
end component;
--
-- Attribute Declarations:
attribute WRITE_MODE : string;
attribute INIT: string;
attribute SRVAL: string;
-- Parity memory initialization attributes
attribute INITP_00: string;
attribute INITP_01: string;
... (snip)
attribute INITP_07: string;
-- Data memory initialization attributes
```

```
attribute INIT_00: string;
attribute INIT_01: string;
... (snip)
attribute INIT_3F: string;
--
-- Attribute "Read during Write mode" = WRITE_FIRST(default)/ READ_FIRST/
NO_CHANGE
attribute WRITE_MODE of U_RAMB16_S36: label is "READ_FIRST";
attribute INIT of U_RAMB16_S36: label is "000000000";
attribute SRVAL of U_RAMB16_S36: label is "012345678";
--
-- RAMB16 memory initialization for Alliance
-- Default value is "0" / Partial initialization strings are padded
-- with zeros to the left
attribute INITP_00 of U_RAMB16_S36: label is
"0000000000000000000000000000000000000000000000000FEDCBA9876543210";
attribute INITP_01 of U_RAMB16_S36: label is
"0000000000000000000000000000000000000000000000000000000000000000";
... (snip)
attribute INITP_07 of U_RAMB16_S36: label is
"0000000000000000000000000000000000000000000000000000000000000000";
--
attribute INIT_00 of U_RAMB16_S36: label is
"0000000000000000000000000000000000000000000000000FEDCBA9876543210";
attribute INIT_01 of U_RAMB16_S36: label is
"0000000000000000000000000000000000000000000000000000000000000000";
... (snip)
attribute INIT_3F of U_RAMB16_S36: label is
"0000000000000000000000000000000000000000000000000000000000000000";
--
-- Signal Declarations:
--
-- signal VCC : std_logic;
-- signal GND : std_logic;
signal CLK_BUFG: std_logic;
signal INV_SET_RESET : std_logic;
--
begin
-- VCC <= '1';
-- GND <= '0';
--
-- Instantiate the clock Buffer
U_BUFG: BUFG
port map (
    I => CLK,
    O => CLK_BUFG
);
--
-- Use of the free inverter on SSR pin
INV_SET_RESET <= NOT SET_RESET;
-- Block SelectRAM Instantiation
U_RAMB16_S36: RAMB16_S36
port map (
    DI   =>  DATA_IN (31 downto 0),   -- insert 32 bits data-in bus (<31 downto 0>)
    DIP  =>  DATA_IN (35 downto 32),  -- insert 4 bits parity data-in bus (or <35
                                      --    downto 32>)
    ADDR =>  ADDRESS (8 downto 0),    -- insert 9 bits address bus
    EN   =>  ENABLE,                  -- insert enable signal
    WE   =>  WRITE_EN,                -- insert write enable signal
    SSR  =>  INV_SET_RESET,           -- insert set/reset signal
    CLK  =>  CLK_BUFG,                -- insert clock signal
    DO   =>  DATA_OUT (31 downto 0),  -- insert 32 bits data-out bus (<31 downto 0>)
    DOP  =>  DATA_OUT (35 downto 32)  -- insert 4 bits parity data-out bus (or <35
                                      --    downto 32>)
);
--
end XC3S_RAMB_1_PORT_arch;
```

付録 B: Verilog インスタンシエーション例

次は、Synopsys FPGA Express システムの場合の、Verilog インスタンシエーション例です。この例にある XC3S_RAMB_1_PORT モジュールは、**SelectRAM_A36.v** Verilog テンプレートを使用します。また、このテンプレートを含むその他のテンプレートは、次のウェブ サイトからダウンロード可能です。ここで示している Verilog コードは、断片的であり、完全なものではありません。

- ftp://ftp.xilinx.com/pub/applications/xapp/xapp463_verilog.zip

```
// Module: XC3S_RAMB_1_PORT
// Description: 18Kb Block SelectRAM-II example
// Single Port 512 x 36 bits
// Use template "SelectRAM_A36.v"
//
// Device: Spartan-3 Family
//-----------------------------------------------------------------------
module XC3S_RAMB_1_PORT (CLK, SET_RESET, ENABLE, WRITE_EN, ADDRESS, DATA_IN,
DATA_OUT);
    input CLK, SET_RESET, ENABLE, WRITE_EN;
    input [35:0] DATA_IN;
    input [8:0] ADDRESS;
    output [35:0] DATA_OUT;
    wire CLK_BUFG, INV_SET_RESET;
//Use of the free inverter on SSR pin
assign INV_SET_RESET = ~SET_RESET;
// initialize block ram for simulation
// synopsys translate_off
defparam
//Read during Writeî attribute for functional simulation
U_RAMB16_S36.WRITE_MODE = READ_FIRSTî, //WRITE_FIRST(default)/ READ_FIRST/ NO_CHANGE
//Output value after configuration
U_RAMB16_S36.INIT = 36'h000000000,
//Output value if SSR active
U_RAMB16_S36.SRVAL = 36'h012345678,
//Initialize parity memory content
U_RAMB16_S36.INITP_00 =
256'h0123456789ABCDEF0000000000000000000000000000000000000000000000000,
U_RAMB16_S36.INITP_01 =
256'h0000000000000000000000000000000000000000000000000000000000000000,
... (snip)
U_RAMB16_S36.INITP_07 =
256'h0000000000000000000000000000000000000000000000000000000000000000,
//Initialize data memory content
U_RAMB16_S36.INIT_00 =
256'h0123456789ABCDEF0000000000000000000000000000000000000000000000000,
U_RAMB16_S36.INIT_01 =
256'h0000000000000000000000000000000000000000000000000000000000000000,
... (snip)
U_RAMB16_S36.INIT_3F =
256'h0000000000000000000000000000000000000000000000000000000000000000;
// synopsys translate_on
//Instantiate the clock Buffer
BUFG U_BUFG ( .I(CLK), .O(CLK_BUFG));
//Block SelectRAM Instantiation
RAMB16_S36 U_RAMB16_S36 (
    .DI(DATA_IN[31:0]),
    .DIP(DATA_IN-PARITY[35:32]),
    .ADDR(ADDRESS),
    .EN(ENABLE),
    .WE(WRITE_EN),
    .SSR(INV_SET_RESET),
    .CLK(CLK_BUFG),
    .DO(DATA_OUT[31:0]),
    .DOP(DATA_OUT-PARITY[35:32]));
// synthesis attribute declarations
/* synopsys attribute
WRITE_MODE "READ_FIRST"
INIT "000000000"
```

```
SRVAL "012345678"
INITP_00
"0123456789ABCDEF0000000000000000000000000000000000000000000000000000"
INITP_01
"0000000000000000000000000000000000000000000000000000000000000000000"
... (snip)
INITP_07
"0000000000000000000000000000000000000000000000000000000000000000000"
INIT_00
"0123456789ABCDEF0000000000000000000000000000000000000000000000000000"
INIT_01
"0000000000000000000000000000000000000000000000000000000000000000000"
... (snip)
INIT_3F
"0000000000000000000000000000000000000000000000000000000000000000000"
*/
endmodule
```

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
04/05/03	1.0	初版リリース
05/12/03	1.1	XC3S50 デバイスに関するブロック RAM 情報の更新 表 1、図 2、図 18、図 26 および はじめに とブロック RAM 配置 に関連した文章の更新 表 2 にシングル ポートモードを追加 関連文献および情報、付録 A: VHDL インスタンシエーション例、および付録 B: Verilog インスタンシエーション例 のハイパーテキスト リンクの更新 図 22 の修正。DIB および ADDRB の変更
06/25/03	1.1.1	マイナー編集改正
07/23/03	1.1.2	図 5 の更新