シングル エラー訂正およびダブル エラー検出

著者 : Simon Tam

XAPP645 (v2.1) 2005 年 7月 20日

概要

このアプリケーション ノートでは、Virtex™-II、Virtex-II Pro、または Virtex-4 デバイスにおけるError Correction Control (ECC) モジュールのインプリメンテーションについて説明します。このデザインでは、すべてのシングル ビット エラー (64 ビット データと 8 パリティ ビットまたは 32 ビット データと 7 パリティ ビットで構成されるコードワード) の検出および訂正を行い、データ内のダブル ビット エラーを検出します。デザインでは、単純でありながら、有効な ECC 処理手法であるハミング コードを利用しています。これによって、非常に高いパフォーマンスと低いリソース使用率を実現します。

はじめに

信頼性およびパフォーマンスの高いアプリケーションの多くに、エラーの検出および訂正機能があります。たとえば、エンタープライズ データ格納システム内には、システムの信頼性を向上させるために、メモリ キャッシュが使用されています。通常、キャッシュは、ホスト インターフェイスとディスク アレイ間にあるコントローラ内に置かれています。たいていの場合、信頼性の高いキャッシュ メモリ デザインには ECC 機能が備わっており、あるエラーによってカスタマ データを損失することを回避しています。ECC は、衛星放送受信機などの通信アプリケーションにとって、重要な機能となってきています。これは、データの再送信よりもパフォーマンスおよびコスト効率が高いエラー訂正方法です。

このアプリケーション ノートに記載されているリファレンス デザイン (**XAPP645.zip**) では、-6 スピード グレード の Virtex-II Pro デバイスを使用し、パイプライン化されていないラインで最高 144MHz、パイプライン化されたラインで最高 313MHz までのデータ読み出し/書き込み速度で、エラーの検出および訂正を行う機能を実現しています。コードワード内の任意の位置で、ダブル ビット エラーが検出され、シングル ビット エラーが訂正されます。また、リファレンス デザインは 72 ビット ダブル データ レート (DDR) DIMM メモリをターゲットとしています。32 ビットのデザインおよびパイプライン化されたデザインも用意されています。また、デザインは容易に変更でき、より狭いデータ幅に対応させることもできます。

ハミング コード

このアプリケーション ノートで説明している ECC 機能は、比較的単純でありながら、非常に有効な ECC コードであるハミング コードを使用して実現されています。ハミング コードは、データを複数のチェック ビット (パリティ) と共に送信し、データ受信の際には、対応するチェック ビットをデコードすることによって、エラーを検出します。

チェック ビットは、元のデータ ワードにある特定のビットを XOR 接続することによって生成される並行のパリティ ビットです。コードワード内にビット エラーがある場合、受信されたデータ ワードのデコード後に、複数のチェック ビットがパリティ エラーを示します。また、これらのチェック ビット エラーの組み合わせによって、エラーのタイプがわかります。さらに、チェック ビットからシングル ビット エラーの位置が特定されます。

ハミング コードワードは、元のデータとチェック ビット (パリティ) が連結したものです。これは、規則的な形式 (d + p,d) で表わされます。ここで、d はデータ幅を示し、p はパリティ幅を示します。パリティのマトリックス [P] は、次のように表すことができます。

$$[P] = [D] \cdot [G]$$

ここで、[D] はデータ マトリックスを示し、[G] は、ジェネレータ マトリックスを示します。この [G] マトリックスは、単位マトリックス [I] および生成マトリックス [C] で構成されます。

$$[G] = [I:C]$$

たとえば、(7,4) ハミング コードは次のようになります。

$$[G] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

シングル ビット エラーの検出に最低限必要なチェック ビット数は、次の等式から求められます。

$$D + P + 1 \leq 2^P$$

このリファレンス デザインでは、(72,64) ハミング コードを使用します。つまり、ハミング コードワード幅が 72 ビットであり、64 データ ビットと 8 チェック ビットで構成されています。64 ビット ワードの場合、シングル ビット エラーの訂正に必要な最低チェック ビット数は 7 です。また、追加分となるチェック ビットによって、ダブル ビット エラーの検出機能を持たせることができます。

エラーを検出するため、コードワード ベクタにジェネレータ マトリックスの転置したものを掛け合わせることによって、シンドローム ベクタとして知られる 8 ビットのベクタ [S] が求められます。

$$[S] = [D,P] \cdot [G']$$

シンドローム ベクタのすべての要素がゼロの場合、エラーはありません。ゼロ以外の場合は、ビット エラー タイプおよびシングル ビット エラーの位置が示されます。そして、このベクタを使用して、元の入力データが修正されます。

ハミング コードを具体的に理解するために、次の表を参照してください。図 1 に示すように、各データ ビットおよびチェック ビットの位置は、シンドローム表にマップされています。表内のセルの各位置は行列で示されます。たとえば、データ ビット 60 は 100 列、1000 行にあり、1000100 位置にあると言うことができます。各位置にあるビットのパリティ ビット (偶数または奇数) を算出することによって、7 チェック ビットが導かれます。チェック ビットの等式は、⊕ と示される XOR 演算子によって構成されています。次に、チェック ビット 1 (CB1) のロジック的な等式例を示します。

CB1 = D0 ⊕ D1 ⊕ D3 ⊕ D4 ⊕ D6 ⊕ D8 ⊕ D10 ⊕ D11 ⊕ D13 ⊕ D15 ⊕ D17 ⊕ D19 ⊕ D21 ⊕ D23 ⊕ D25 ⊕ D26 ⊕ D28 ⊕ D30 ⊕ D32 ⊕ D34 ⊕ D36 ⊕ D38 ⊕ D40 ⊕ D42 ⊕ D44 ⊕ D46 ⊕ D48 ⊕ D50 ⊕ D52 ⊕ D54 ⊕ D56 ⊕ D57 ⊕ D59 ⊕ D61 ⊕ D63

基本的に、表におけるセル位置の最下位ビットが 1 であるすべてのデータ ビットが CB1 を生成するために選択されます(図 1 を参照)。また、CB2 の生成には、最後から 2 番目のビットが選択され、以降も同様です。

111	110	101	100	011	010	001	000	
D63	D62	D61	D60	D59	D58	D57	*CB7*	1000
D56	D55	D54	D53	D52	D51	D50	D49	0111
D48	D47	D46	D45	D44	D43	D42	D41	0110
D40	D39	D38	D37	D36	D35	D34	D33	0101
D32	D31	D30	D29	D28	D27	D26	*CB6*	0100
D25	D24	D23	D22	D21	D20	D19	D18	0011
D17	D16	D15	D14	D13	D12	D11	*CB5*	0010
D10	D9	D8	D7	D6	D5	D4	*CB4*	0001
D3	D2	D1	*CB3*	D0	*CB2*	*CB1*	No Error	0000

x645_01_022103

図 1： シンドローム表

図 2 に示すように、ビット エラーがない場合、チェック ビットはデータの算出したチェック ビットと一致します。その結果、すべてのシンドローム ビットはゼロになり、No Error 位置を示します。

111	110	101	100	011	010	001	**000**	
D63	D62	D61	D60	D59	D58	D57	*CB7*	1000
D56	D55	D54	D53	D52	D51	D50	D49	0111
D48	D47	D46	D45	D44	D43	D42	D41	0110
D40	D39	D38	D37	D36	D35	D34	D33	0101
D32	D31	D30	D29	D28	D27	D26	*CB6*	0100
D25	D24	D23	D22	D21	D20	D19	D18	0011
D17	D16	D15	D14	D13	D12	D11	*CB5*	0010
D10	D9	D8	D7	D6	D5	D4	*CB4*	0001
D3	D2	D1	*CB3*	D0	*CB2*	*CB1*	No Error	**0000**

x645_01_022003

図 2： ビット エラーなしの検出

図 3 に示すように、シングル ビット エラーが発生した場合は、いくつかのシンドローム ビットが奇数のパリティを持ち (結果はロジック 1 になる)、そこで行列の位置が決定されます。たとえば、D28 がエラーの場合、CB1、CB2、および CB6 のパリティがエラーとなります。その結果、シンドローム表では D28 がエラー ビットと認識されます。

111	110	101	100	**011**	010	001	000	
D63	D62	D61	D60	D59	D58	D57	*CB7*	1000
D56	D55	D54	D53	D52	D51	D50	D49	0111
D48	D47	D46	D45	D44	D43	D42	D41	0110
D40	D39	D38	D37	D36	D35	D34	D33	0101
D32	D31	D30	D29	**D28**	~~D27~~	~~D26~~	~~*CB6*~~	**0100**
D25	D24	D23	D22	D21	D20	D19	D18	0011
D17	D16	D15	D14	D13	D12	D11	*CB5*	0010
D10	D9	D8	D7	D6	D5	D4	*CB4*	0001
D3	D2	D1	*CB3*	D0	*CB2*	*CB1*	No Error	0000

x645_02_022003

図 3 : シングル ビット エラーの検出

図 4 に示すように、ダブル ビット エラーが発生した場合は、エラー ビット位置が指定されないか、誤った位置が指定されます。たとえば、ダブル ビット エラーが D28 および D22 で発生すると、シンドローム位置は列 111 および行 0111 となります。しかし、すべてのデータ ビットを変換させるチェック ビット (CB8) を追加することによって、ダブル エラーを検出できます。CB8 が 0 に戻り、CB1 から CB7 が 0 以外の値の場合、ダブル ビット エラーが発生していることがわかります。

111	110	101	100	011	010	001	000	
D63	D62	D61	D60	D59	D58	D57	*CB7*	1000
~~D56~~	~~D55~~	~~D54~~	~~D53~~	~~D52~~	~~D51~~	~~D50~~	~~D49~~	**0111**
D48	D47	D46	D45	D44	D43	D42	D41	0110
D40	D39	D38	D37	D36	D35	D34	D33	0101
D32	D31	D30	D29	**D28**	D27	D26	*CB6*	0100
D25	D24	D23	**D22**	D21	D20	D19	D18	0011
D17	D16	D15	D14	D13	D12	D11	*CB5*	0010
D10	D9	D8	D7	D6	D5	D4	*CB4*	0001
D3	D2	D1	*CB3*	D0	*CB2*	*CB1*	No Error	0000

x645_03_022003

図 4 : ダブル ビット エラーの検出

デザイン概要

図 5 に、ECC 機能のある DDR メモリ コントローラを使用する場合のブロック図を示します。この例で示す DDR DIMM は、Micron MT18VDDT6472G、ECC 構成モジュールです。また、リファレンス デザインには、パリティ エンコーダおよびパリティ デコーダユニットがあります。エンコーダはジェネレータ マトリックスの機能を果たし、デコーダはエラー検出および訂正の役割を果たします。さらに、診断機能がサポートされています。次に、これらの機能について説明します。

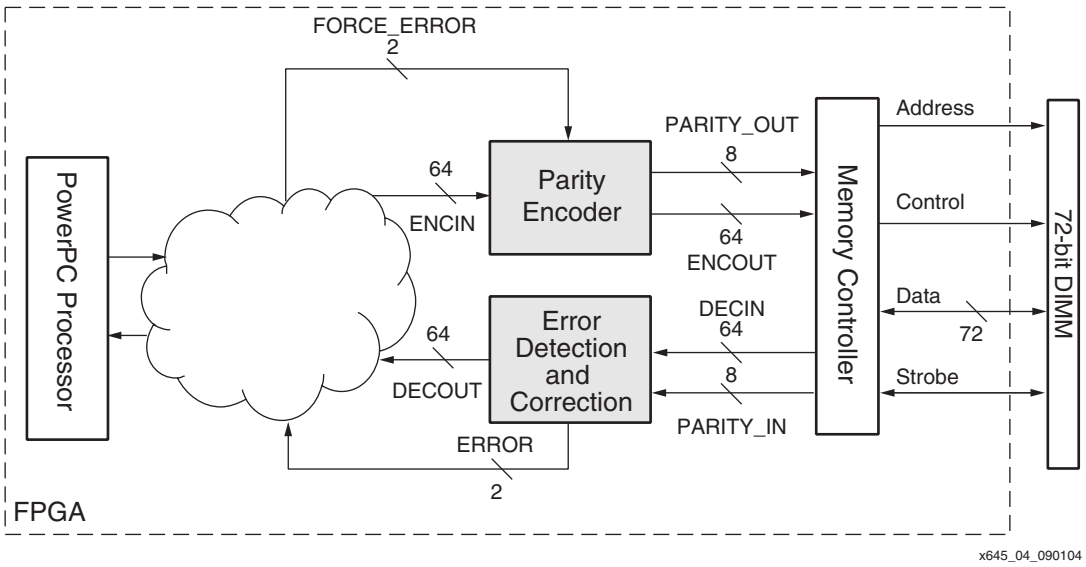

図 5： メモリ システムにおける ECC

パリティ エンコーダ

エンコーダは、ルックアップ テーブル (LUT) にインプリメントされた XOR およびビット エラー ジェネレータで構成されます。オプションとして、これをパイプライン化することよって、パフォーマンスを向上させることができます。図 6 に、パリティ エンコーダのブロック図を示します。

チェック ビットは 64 ビットのデータと共にメモリに書き込まれます。メモリの読み出しでは、データとチェック ビットが同時に読み出されます。そして、FPGA とメモリ間の読み出しまたは書き込み中に発生したエラーが検出されます。

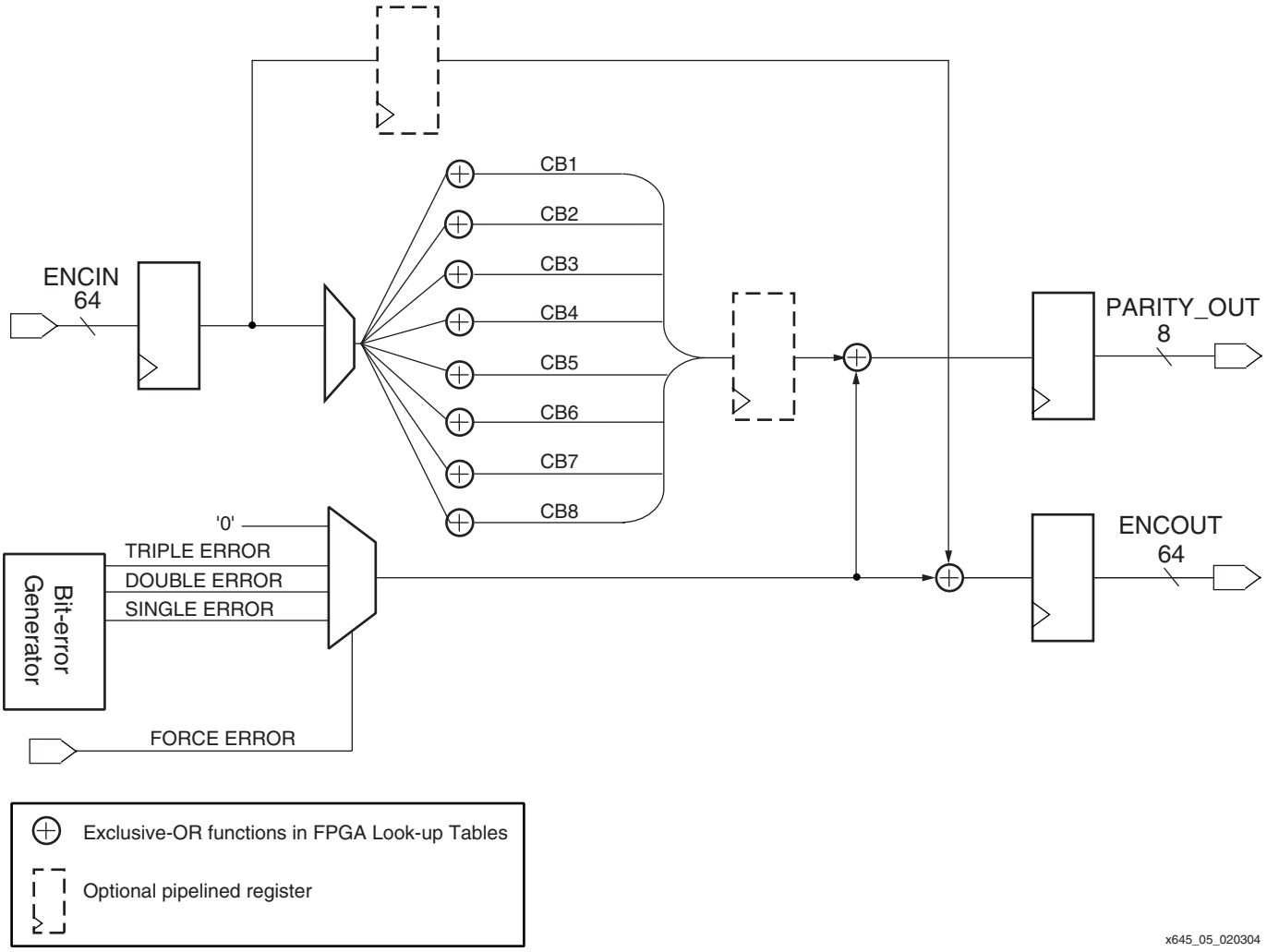

図 6： パリティ エンコーダのブロック図

パリティ ビットは未修正のハミング コードに基づいて生成されます。表 1 に、(72,64) コードワードの生成に関連するビットを示し、表 2 に、(39,32) コードワードの生成に関連するビットを示します。

表 1 : 64 ビット ハミング コード

関連するデータ ビット	生成されたチェック ビット							
	CB1	CB2	CB3	CB4	CB5	CB6	CB7	CB8
0	√	√						√
1	√		√					√
2		√	√					√
3	√	√	√					√
4	√			√				√
5		√		√				√
6	√	√		√				√
7			√	√				√
8	√		√	√				√
9		√	√	√				√
10	√	√	√	√				√
11	√				√			√
12		√			√			√
13	√	√			√			√
14			√		√			√
15	√		√		√			√
16		√	√		√			√
17	√	√	√		√			√
18				√	√			√
19	√			√	√			√
20		√		√	√			√
21	√	√		√	√			√
22			√	√	√			√
23	√		√	√	√			√
24		√	√	√	√			√
25	√	√	√	√	√			√
26	√					√		√
27		√				√		√
28	√	√				√		√
29			√			√		√
30	√		√			√		√
31		√	√			√		√
32	√	√	√			√		√
33				√		√		√
34	√			√		√		√

表 1 : 64 ビット ハミング コード (続き)

関連するデータ ビット	生成されたチェック ビット							
	CB1	CB2	CB3	CB4	CB5	CB6	CB7	CB8
35		√		√		√		√
36	√	√		√		√		√
37			√	√		√		√
38	√		√	√		√		√
39		√	√	√		√		√
40	√	√	√	√		√		√
41					√	√		√
42	√				√	√		√
43		√			√	√		√
44	√	√			√	√		√
45			√		√	√		√
46	√		√		√	√		√
47		√	√		√	√		√
48	√	√	√		√	√		√
49				√	√	√		√
50	√			√	√	√		√
51		√		√	√	√		√
52	√	√		√	√	√		√
53			√	√	√	√		√
54	√		√	√	√	√		√
55		√	√	√	√	√		√
56	√	√	√	√	√	√		√
57	√						√	√
58		√					√	√
59	√	√					√	√
60			√				√	√
61	√		√				√	√
62		√	√				√	√
63	√	√	√				√	√

表 2 : 32 ビット ハミング コード

関連するデータ ビット	生成されたチェック ビット						
	CB0	CB1	CB2	CB3	CB4	CB5	CB6
0	√	√					√
1	√		√				√
2		√	√				√
3	√	√	√				√
4	√			√			√
5		√		√			√
6	√	√		√			√
7			√	√			√
8	√		√	√			√
9		√	√	√			√
10	√	√	√	√			√
11	√				√		√
12		√			√		√
13	√	√			√		√
14			√		√		√
15	√		√		√		√
16		√	√		√		√
17	√	√	√		√		√
18				√	√		√
19	√			√	√		√
20		√		√	√		√
21	√	√		√	√		√
22			√	√	√		√
23	√		√	√	√		√
24		√	√	√	√		√
25	√	√	√	√	√		√
26	√					√	√
27		√				√	√
28	√	√				√	√
29			√			√	√
30	√		√			√	√
31		√	√			√	√

パリティ デコーダ

図 7 に示すように、デコーダ回路は 3 つのブロックで構成されます。

- シンドローム生成
- シンドローム LUT およびマスク生成
- データ訂正

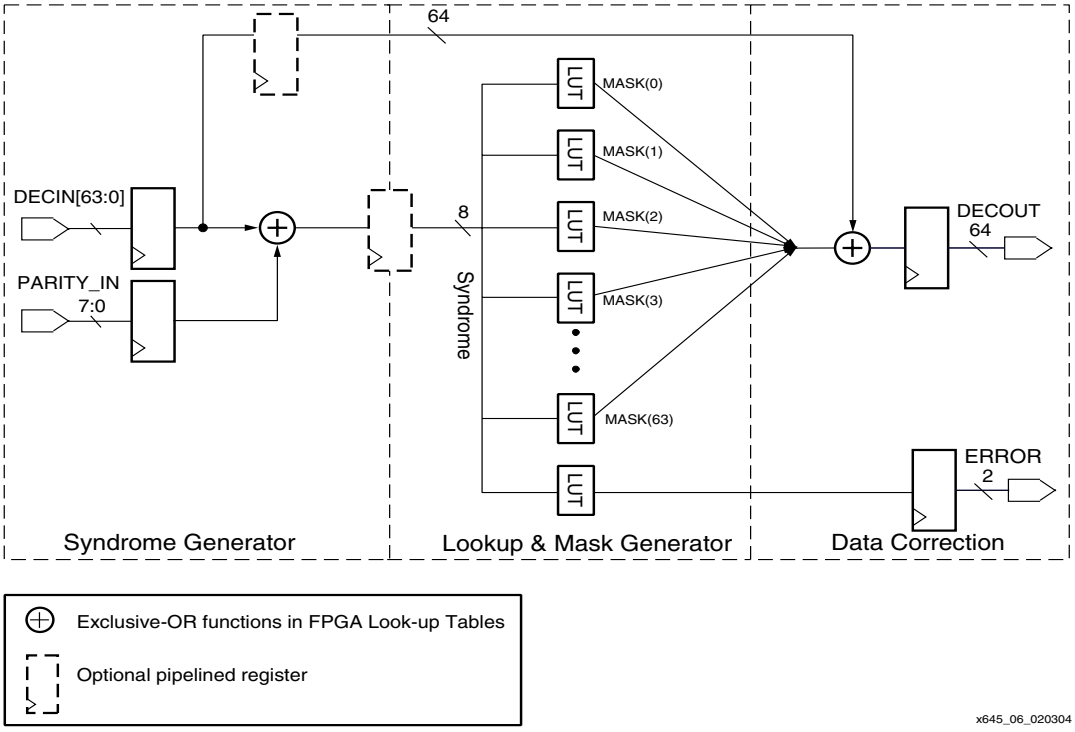

図 7： ECC 機能ブロック図

シンドローム生成

64 ビットの入力データは、8 ビットのパリティと共に XOR を介し、8 ビットのシンドローム (S1 から S8) を生成します。これは、チェック ビットの生成と非常に類似しています。次の例を参照してください。

S1 = DECIN0 ⊕ DECIN1 ⊕ DECIN3 ⊕ DECIN4 ⊕ DECIN6 ⊕ DECIN8 ⊕ DECIN10 ⊕ DECIN11 ⊕ DECIN13 ⊕ DECIN15 ⊕ DECIN17 ⊕ DECIN19 ⊕ DECIN21 ⊕ DECIN23 ⊕ DECIN25 ⊕ DECIN26 ⊕ DECIN28 ⊕ DECIN30 ⊕ DECIN32 ⊕ DECIN34 ⊕ DECIN36 ⊕ DECIN38 ⊕ DECIN40 ⊕ DECIN42 ⊕ DECIN44 ⊕ DECIN46 ⊕ DECIN48 ⊕ DECIN50 ⊕ DECIN52 ⊕ DECIN54 ⊕ DECIN56 ⊕ DECIN57 ⊕ DECIN59 ⊕ DECIN61 ⊕ DECIN63 ⊕ PARITY_IN(1)

次に、シンドロームを使用して、エラー タイプおよびその位置の検出を行います。オプションとして、パイプライン化することによって、パフォーマンスを向上させることができます。

シンドローム LUT およびマスク生成

シングル ビット エラーを訂正するために、64 ビットの訂正マスクが作成されます。このマスクの各ビットは、前段階のシンドロームの結果に基づいて作成されます。エラーが検出されない場合、マスクのすべてのビットは 0 になります。シングル ビット エラーが検出されると、対応するマスクが、エラー ビット以外のビットをマスク アウトします。次に、マスクと元のデータが XOR に入力されます。その結果、エラー ビットが正しいステートに反転 (または訂正) されます。ダブル ビット エラーが検出された場合は、すべてのマスクビットが 0 になります。そして、同一クロック サイクルで、エラー タイプおよびそれに対応する訂正マスクが生成されます。

データ訂正

データ訂正の段階では、マスクと元の入力データが XOR され、データ訂正が必要な場合、エラー ビットが訂正されます。シングル ビット エラーまたはダブル ビット エラーが発生していない場合、すべてのマスク ビットは 0 です。その結果、入力データは、元のデータを維持したまま ECC 回路を通過します。

エラー診断

リファレンス デザインは、エラー タイプを示すだけでなく、診断モードもサポートしています。出力コードワードには、シングル、ダブル、およびトリプル ビット エラーが発生する可能性があります。

ERROR ポートが 00 の場合、シングル ビット エラー、ダブル ビット エラー、またはそれ以上のビット エラーはいずれも検出されていません。つまり、検索したデータには、パリティ エラーがないことになります。ERROR ポートが 01 の場合、72 ビット コードワード内でシングル ビット エラーが発生したことを示します。そのエラーは修正され、データにエラーはなくなります。また、ERROR ポートが 10 の場合、データ ワード内でダブル ビット エラーが発生しています。このエラーに対し、エラー修正は実行できません。最後に、ERROR ポートが 11 の場合、コードワード内で、検知機能では検知できないエラーが発生していると考えられ、エラー修正は実行できません。これは、無効なエラー タイプです。

エンコーダの出力で、コードワードに意図的にビット エラーを挿入し、システムのテストを行うことができます。Force_error を使用して、いくつかのエラー モード タイプを設定できます。

Force_error = 00

通常の動作モードです。エンコーダの出力でビット エラーは挿入されていません。

Force_error = 01

シングル ビット エラー モードです。コードワードでは、クロックの立ち上がりエッジごとに、1 ビットが反転します (0 が 1 に、または 1 が 0 になります)。シングル ビット エラーは、シーケンスに従い、コードワードのビット 0 からビット 72 に移動します。このシーケンスは、エラー モードがアクティブである間継続します。

Force_error = 10

ダブル ビット エラー モードです。コードワードでは、クロックの立ち上がりエッジごとに、2 つの連続したビットが反転します (0 が 1 に、または 1 が 0 になります) 。ダブル ビット エラーは、シーケンスに従い、コードワードのビット (0,1) からビット (71, 72) に移動します。このシーケンスは、エラー モードがアクティブである間継続します。

Force_error = 11

トリプル ビット エラー モードです。コードワードでは、クロックの立ち上がりエッジごとに、3 ビットが反転します (0 が 1 に、または 1 が 0 になります)。トリプル ビット エラーは、シーケンスに従い、コードワードのビット (0,1, 2) からビット (70, 71, 72) に連続して移動します。シーケンスは、エラー モードがアクティブである間連続します。

使用率およびパフォーマンス

リファレンス デザインでは、最小限のリソースで、高いパフォーマンスを実現しています。表 3 に、パフォーマンスおよび使用率の概要を示します。このデザインは、Xilinx Synthesis Tool (XST) を使用して合成されています。パフォーマンスの概要は、ISE 7.1i SP2 スピード ファイル バージョン 1.53 に基づいており、64 ビット バージョンの ECC リファレンス デザインにのみ反映されます。また、全体的なパフォーマンスは、デザインによって異なる場合があります。

表 3 : パフォーマンスおよび使用率の概要

デバイス	使用率(1)	パフォーマンス	
		パイプラインなし	1 段階のパイプライン
XC2VP4 -6	16%	144MHz	313MHz
XC2VP7 -6	10%	136MHz	298MHz
XC2VP20 -6 または XC2VPX20 -6	5%	132MHz	232MHz
XC2VP50 -6	2%	127MHz	176MHz
XC4VLX15 -11	6%	178MHz	253MHz
XC4VFX20 -11	5%	182MHz	252MHz
XC4VSX35 -11	3%	169MHz	245MHz
XC4VFX60 -11	2%	155MHz	219MHz

メモ :

1.474 スライスの Virtex-II Pro デバイスでの使用率、または 416 スライスの Virtex-4 デバイスでの使用率です。

レイテンシ

必要条件ではありませんが、モジュールの I/O にはレジスタが使用されています。エンコーダでは、入力データが ENCIN に現れてから、デコードされたデータが ENCOUT で使用可能になるまでのレイテンシは 2 クロック サイクル (パイプラインなし)、または 3 クロック サイクル (パイプランあり) です。デコーダでは、入力データが DECIN に現れてから、処理されたデータが DECOUT で使用可能になるまでのレイテンシは 2 クロック サイクル (パイプラインなし)、または 3 クロック サイクル (パイプランあり) です。ステータス信号 ERROR は DECOUT に同期しています。図 8 に、タイミング レイテンシの図を示します。

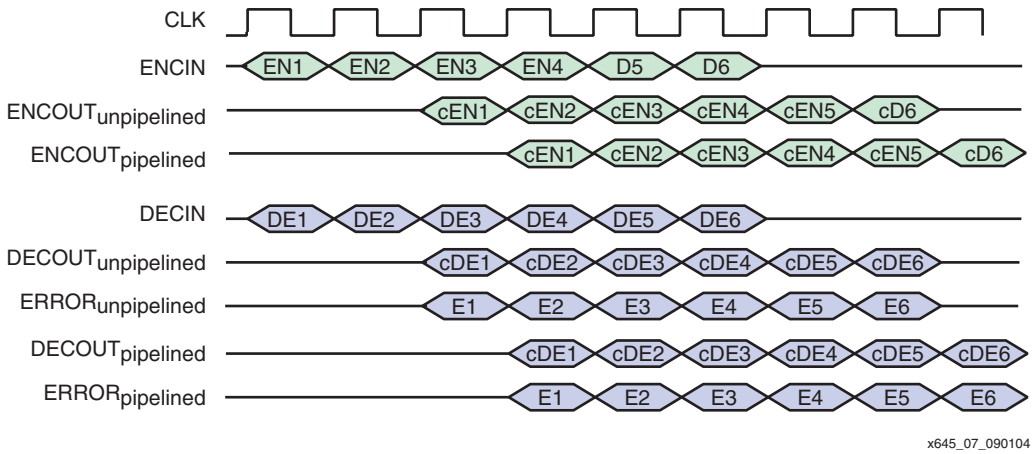

図 8 : タイミング図

レイテンシについて

ENx = エンコード前の書き込みデータ

ENx = エンコーダから出力された書き込みデータであり、チェック ビットは書き込みのために使用可能

DEx = ECC ユニットに入力前の読み出しデータ

cDE = ECC ユニットから出力された訂正済み読み出しデータ

Ex = ECC ユニットから生成されたエラー ステータス

ピンについて

表 4 に、立ち上がりエッジで使用できる ECC モジュールのユーザー インターフェイス ピンの一覧を示します。

表 4 : ECC モジュールのピン

ピン名	入力/出力	幅 (64 ビット	幅 (32 ビット)	説明
CLK	入力			クロック入力
RESET	入力			アクティブ Low リセット
ENCIN	入力	63:0	31:0	エンコーダへの元の入力データ
ENCOUT	出力	63:0	31:0	エンコーダを通りラッチされた元のデータ
PARITY_OUT	出力	7:0	6:0	データ (ENCIN) に基づいてエンコーダから、同一クロック エッジで生成されたパリティ ビット
DECIN	入力	63:0	31:0	デコーダへの入力データ
DECOUT	出力	63:0	31:0	DECIN から修正されたデータ
PARITY_IN	入力	7:0	6:0	同一立ち上がりエッジでラッチされた入力データ (DECIN) に付随するパリティ ビット
FORCE_ERROR	入力	1:0	1:0	テストの際にエンコードされたデータ ワードにビット エラーを導入 00 - 通常の動作 01 - シングル ビット エラーを挿入 10 - ダブル ビットエラーを挿入 11 - ダリプル ビット エラーを挿入
ERROR	出力	1:0	1:0	エラー ステータス 00 - エラーなし 01 - シングル ビット エラーの検出および訂正 10 - ダブル ビット エラーの検出。訂正なし 11 - 無効なビット エラーの検出

リファレンス デザイン ファイル

VHDL および Verilog のリファレンス デザイン ファイルは、次のザイリンクス ウェブサイトから入手できます。

http://www.xilinx.co.jp/bvdocs/appnotes/xapp645.zip

おわりに

このアプリケーション ノートでは、Virtex-II、Virtex-II Pro、および Virtex-4 デバイスを使用した場合のエンコーディングおよびハミング コード検出の簡潔な方法を示しました。

改訂履歴

次の表に、このアプリケーション ノートの改訂履歴を示します。

日付	バージョン	改訂内容
2003/03/03	1.0	初版リリース
2003/09/17	1.1	32 ビット データの場合のエラー検出および訂正 (Error Detection and Correction 、EDC) 機能についての説明を更新。 スピード ファイル バージョン 1.81 を反映させたパフォーマンスを記載。
2004/02/03	1.2	パイプラインを使用したアプリケーションの説明を追加。
2004/09/01	2.0	Virtex-4 FPGA の記載を追加。
2005/07/20	2.1	パフォーマンスおよび使用率の概要の表を更新 (表 3)