XAPP1273 (v1.0) 2016 年 3 月 14 日

Vivado 高位合成を使用したリード ソロモン消去コーデックのデザイン

著者 : Matt Ruan

概要

このアプリケーション ノートは、ザイリンクス Vivado® 高位合成 (HLS) ツールを使用した消去コーデックのデザインに焦点を当てています。このコーデックは、C プログラミング言語のソース コードを利用し、Kintex® UltraScale™ FPGA 用の高効率で合成可能な Verilog または VHDL コードを生成します。生成行列の定義や消去コード レートなどの消去コード パラメーターを変更する必要がある場合、C ヘッダー ファイルをわずかに修正するだけで済みます。サンプルのリード ソロモン消去コーデックは一般的なアーキテクチャになっており、C ソース コードを変更すれば、別のタイプの消去コーデックが作成できます。

このアプリケーション ノートのリファレンス デザイン ファイルは、ザイリンクスのウェブサイトからダウンロードできます。デザイン ファイルの詳細は、「リファレンス デザイン」を参照してください。

はじめに

現代社会では、日々大量のデータが生成され、消費されています。それらのデータを処理および保管する目的で設営されるデータ センターでは、ストレージ デバイス内のハード ディスク、メモリ、ネットワーク コネクタなどの電子コンポーネントにはエラーの発生が避けられないにもかかわらず、データの消失は絶対に許されません。一般的なデータの保護方法は、同じデータの複数のコピーを別々の場所で保管するというものです ([参照 1]、[参照 2])。そのようにすれば、データの消失はデータのすべてのコピーが失われたり破損したりしない限り起きることはなく、その可能性は冗長コピーの数を増やせばすぐに減らすことができます。消失の可能性を十分低いレベルに維持するために、標準的なデータ センターではデータの少なくとも 3 つの同一コピーを保管します。言い換えれば、冗長データの量は元のデータの 2 倍よりも多くなるということです。これはまったく非効率です。

深刻なノイズやチャネルの悪条件により送信信号が損なわれがちな通信システムでも、エラーからの保護という同様の問題が研究されてきました。システムでは元の信号の複製をいくつも送信するのではなく、レシーバー側で自動的にエラーを修正できる、前方誤り訂正 (FEC) コードが採用されています。このアイデアはデータ センターにも応用されています。リード ソロモン (RS) 消去コードは誤り訂正方式の 1 つであり、追加されるコーディング情報と同数のエラーを完全に回復できます ([参照 1])。詳しく言えば、n 個のワードで成る元データに k 個のワードで成るコーディング情報を追加した場合、合計数 (n+k) 個のワードのうち、k 個までワード エラーを許容できるということです。(k+1) 個のワードにエラーがある場合、エラーは回復できません。誤り訂正コードで対応できるのはこれが最大限であり、コードはかなり慎重に構築する必要があることに注意してください。

たとえば、10 データ ブロックごとに 4 つまでのデータ破損の発生を許容できるようにするには、コーディング データを格納するための 4 つの追加ブロックがあればよく、オーバーヘッドは 40% にまで抑えられます。ただし、RS 消去コーデックは計算量の多いタスクであり、ミドルレンジ サーバーで実行した場合には、プロセッサ実行時間の 30% ～ 40% を占める可能性があります。これはかなりのオーバーヘッドであり、プログラマブル ロジックに基づいて構築された、柔軟性、スケーラビリティ、そしてリソース効率に優れたアクセラレータを必要とします ([参照 3])。

このアプリケーション ノートは、C プログラミング言語で消去コーデックを記述し、その C コードを Vivado HLS ツール (『Vivado Design Suite ユーザー ガイド : 高位合成』(UG902) ([参照 4])) を使用してハードウェア記述言語 (HDL) に合成し、プログラマブル ロジック デバイスにインプリメントする方法を説明します。Vivado HLS は、一定の制約に従って高性能なパイプライン化されたアーキテクチャを生成し、HDL と C コードの動作が同じであることを確認するテストベンチを作成できます。多くの場合、Vivado HLS 合成コードの効率と性能は、経験豊富なロジック エンジニアが設計してコーディングした HDL と同程度となります。クロック レート、ターゲット ロジック デバイス、または生成行列やコード レートなどの消去コーデック パラメーターを変更する必要がある場合には、C ヘッダー ファイルをわずかに修正するだけで済みます。C で設計された消去コーデックは、ソフトウェア エンジニアが新しいアプリケーション向けに簡単にカスタマイズできる、文字どおりの IP になります。

動作理論

このセクションでは、消去チャネルでの前方誤り訂正の基本理論と、消去コーデックの機能の仕方を説明します。

消去チャネルで、送信信号のアルファベットを A と定義します。(ここで、アルファベットとは一連の送信信号を意味します。たとえば、{-1, +1} は BPSK 変調信号のアルファベットです)。受信したどのシンボル S ∈ A でも、受信したシンボルの消去フラグが設定されていなければ、送信シンボルは必ず S になります。設定されていれば、シンボルは等しい確率であらゆるアルファベットの可能性があります。後者の場合には、有用な情報をレシーバーに提供せずにチャネル内でシンボルが消去されたかのようになります。消去フラグは、より複雑な誤り訂正方式を必要とする標準的な通信チャネルでは使用できないことに注意してください。データ センターでは、ストレージ デバイスに組み込まれているデータ整合性チェック回路が消去イベントを示すことができます。

図 1 は、消去コードの構築方法を示しています。

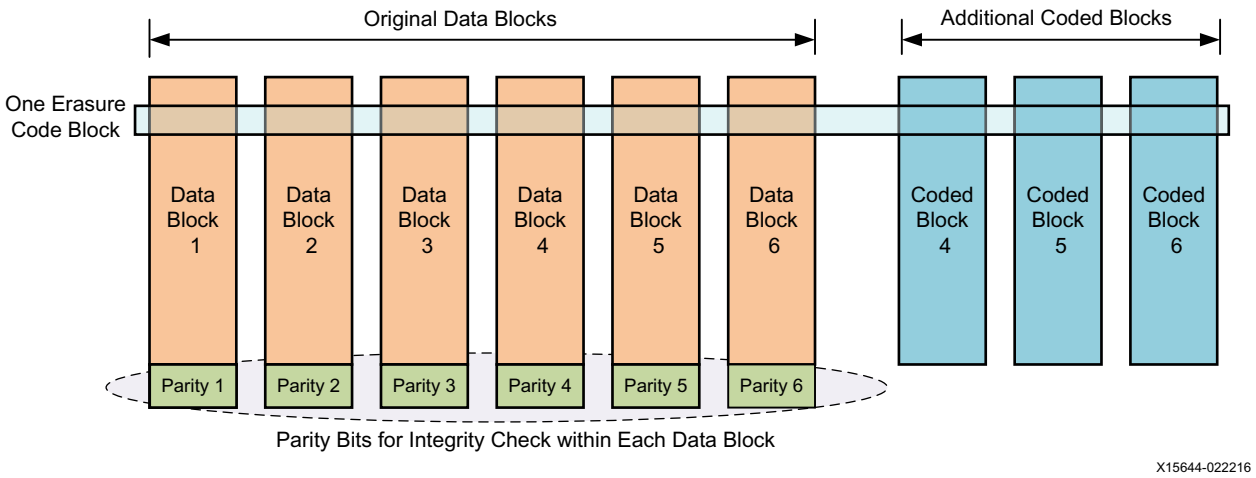


図 1 : 消去コードのブロック図

オレンジ色の囲みは元のデータ ブロックであり、その長さは 1 キロバイトから 128 メガバイトまでさまざまです。緑色の囲みは、整合性テストのために各データ ブロックに追加されたパリティ チェック情報です。パリティ チェック情報のサイズはデータ ブロックのサイズよりもかなり小さく、多くの場合、整合性テストの機能はストレージ デバイスに内蔵されています。図の右側は、消去コーデックで生成されたコード化データ ブロックです。エンコードはすべてのデータ ブロック間で水平に実行されます。図 1 の例では、1 つの消去コード ブロックは、各データ ブロックから 1 つずつの合計で 6 つの情報シンボルと、消去エンコーダーにより計算された 3 つのコード化シンボルで構成されています。

D_i を i 番目のデータ ブロックから取られたシンボル、C_i を i 番目のコード化シンボルとすると、エンコード プロセスは、特定のシンボル ビット幅のガロア体で定義された 1 つの行列ドット乗算で表すことができます ([参照 1])。これは図 2 で示しており、生成行列係数 $\{a_{i,j}\}$ およびドット乗算演算を含むすべてのシンボルは、同じガロア体で定義されます。

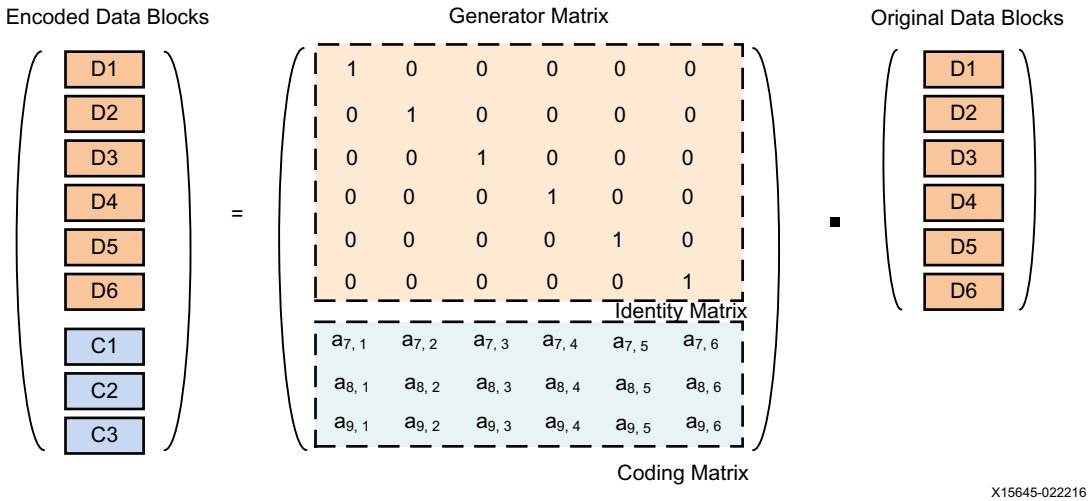
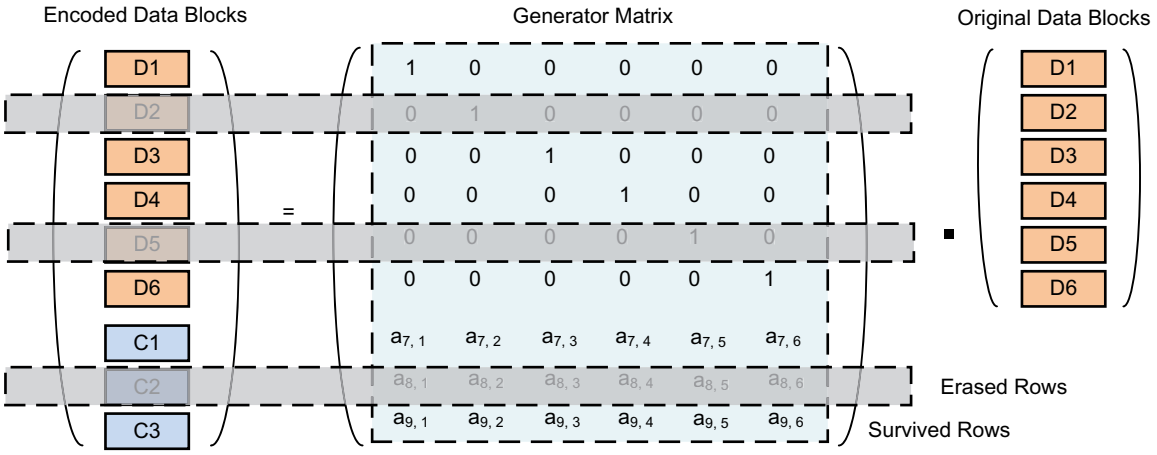
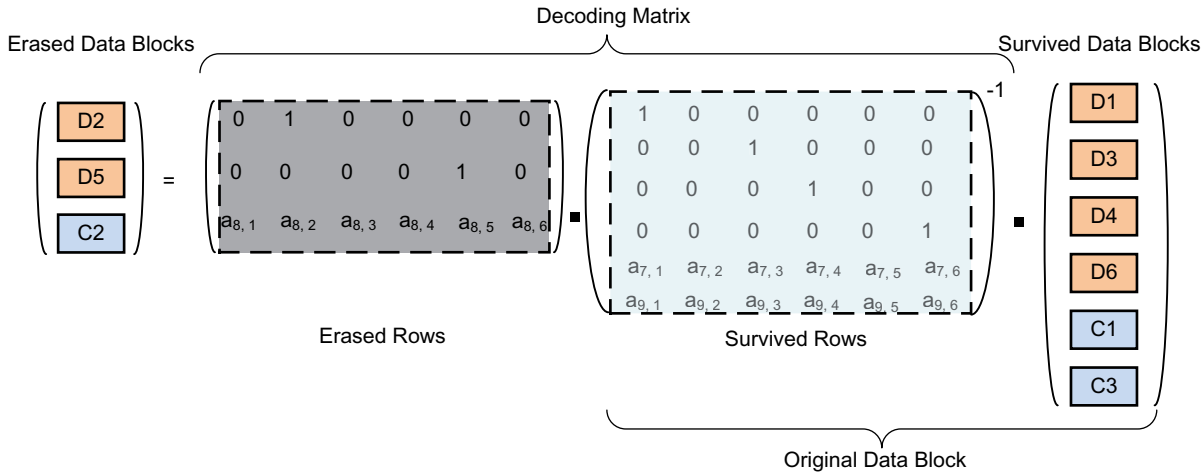


図 2 : 消去コードのエンコード

図 3 (a) に示すとおり、一部のデータ ブロックが失われたかまたは破損した場合、残存データ ブロックと生成行列に関する既知の情報を使用してデータを回復できます。計算手順は図 3 (b) に示しています。ここで、残存行 (水色) の逆数で残存データ ブロックのドット乗算を行うと、元のデータ ブロックが得られ、次に消去された行 (グレー) でドット乗算を行うと、消去されたデータ ブロックが回復されます。



(a) Erased data blocks and rows



(b) Recovery of the erased data blocks

X15670-022216

図 3 : 消去コードのデコード

図 3 (b) に示すとおり、消去コードのデコードでは、残存行で構成される行列の逆数の存在が暗黙に想定されています。この想定は、一部の特殊構築された生成行列に当てはまります。その 1 つは、このアプリケーション ノートでサンプルとして使用しているリード ソロモン コード ([参照 1]) です。ただし、ここで説明されている設計手法は、その他の多様な消去コードにも適用されます。

図 3 は、エンコード プロセスを、元のデータ ブロックがすべて残存してコード化ブロックが消去される特殊ケースのデコードと見なすことができるというもう 1 つの事実を示しています。したがって、必要とされるのは、エンコードにも対応できる消去デコーダーを設計することのみです。

RS 消去コーデックのアーキテクチャ

データ センターでは、重要度が異なるデータ ブロックに合わせて、多数のエラー保護レベルが必要です。これには、多数のコード レートをサポートするためのコーデックが必要です。短いデータ ブロックでオンザフライのエンコードおよびデコード方式を使用できるように、レイテンシを最小化する必要もあります。

このアプリケーション ノートでは、このようなマルチレートの消去コーデックを、通常 300Mhz 以上のクロック周波数で動作するザイリンクス Kintex UltraScale FPGA で設計する例を示します。このデザインは、非常に小さいデコード レイテンシのストリーミング入力インターフェイスを特徴とする一方で、表 1 にまとめられているとおり、1/3 から 2/3 までを範囲とする 4 つのコード レートをサポートしています。デザインにわずかな変更を加えれば、さらに幅広いコード レートをサポートできます。

注記 : この例は Kintex FPGA 向けですが、方式や設計は基本的にすべてのデバイスに適合します。

表 1 : 低レイテンシ マルチレート消去コーデックのコード レート

データ ブロックの数	コード ブロックの数	オーバーヘッドの比率
6	4	67%
8	4	50%
10	4	40%
12	4	33%

図 4 は、消去コーデックの理想的なインターフェイス タイミング図を示しています。水色、黄色、および緑色は、消去デコード パラメーター (残存パターンと消去コード タイプ) が異なる 3 つのデータ セットを表しています。パラメーターの変更にギャップ挿入は必要ないため、ほかのシステム コンポーネントとの相互接続を簡単にするために、ストリーミング インターフェイスへの入力は簡略化されています。

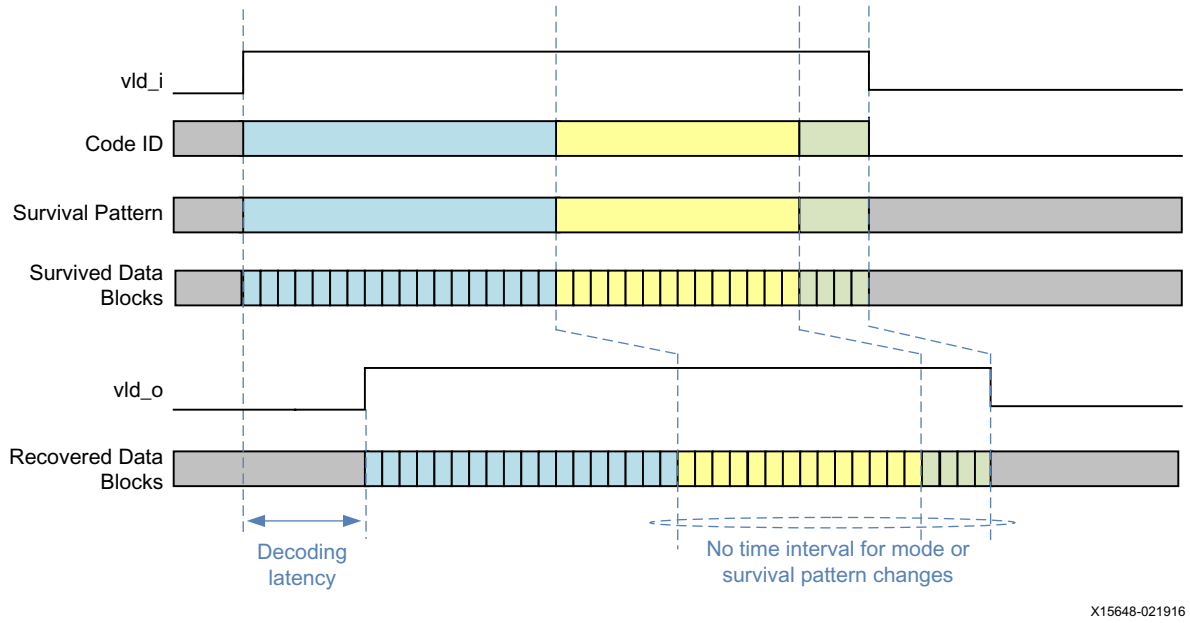


図 4 : 消去コーデックのインターフェイス タイミング図

単純なストリーミング インターフェイスを実現するために、完全にパイプライン化されたアーキテクチャを選択します。行列反転の完了には何百ものクロック サイクルが必要となるため、大きな課題になります。これに対処するための 1 つの解決策は、サポート対象のすべてのコードおよび考えられるすべての残存パターンのすべてのデコード行列を事前に計算し、プログラマブル ロジック デバイスの内部メモリに保存しておくことです。この場合、消去コーデックのアーキテクチャ全体は図 5 のようになります。

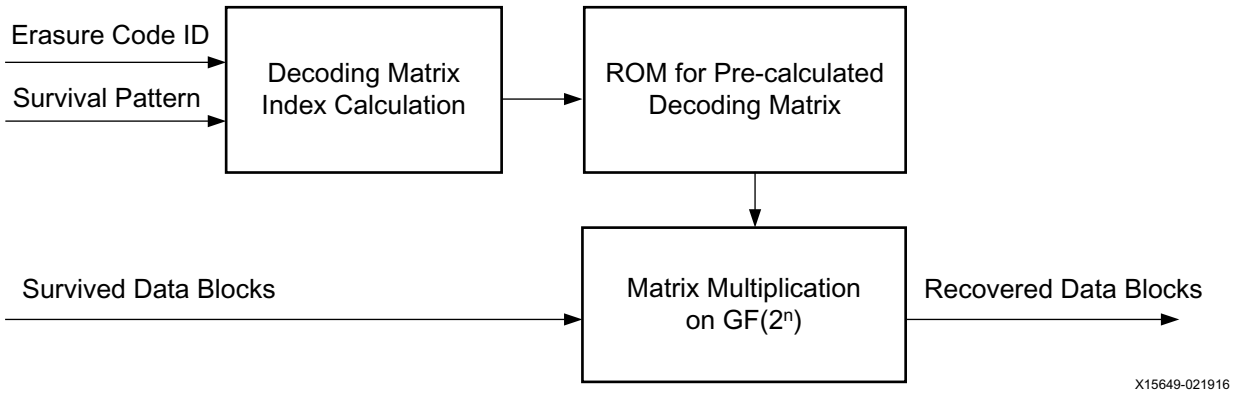


図 5 : 消去コーデックの高位アーキテクチャ

デコード行列 ROM のサイズは、プログラマブル デバイスに適したものにする必要があります。元のデータ ブロック m 個とコード化ブロック k 個で構成される特定の消去コードに対する、考えられる残存パターンの数は (m+k)! / m! / k! で求めることができます。ここで n! は、n 未満のすべての正の整数の積を表します。4 つすべての消去コード タイプの残存パターンの総数は、式 1 に示すように算出できます。

$$10! / 4! / 6! + 12! / 4! / 8! + 14! / 4! / 10! + 16! / 4! / 12! = 3,526 \quad \text{式 1}$$

各デコード行列は、GF(2^8) では最大で 12 × 4 = 48 の要素を持ち、ROM のサイズは 3,526 × 48 × 8 = 1,353,984 ビットとなります。これは 20nm UltraScale FPGA には非常に適しています。

高位アーキテクチャに基づいて、アルゴリズムは C プログラミング言語で記述され、Vivado HLS ツールは各種のハードウェア インプリメンテーション オプションを調べ、デザインに最も適したものを選択します。

インプリメンテーションの詳細

消去コーデックのアーキテクチャ (図 5 参照) は、大きく分けて次の 2 つの計算ステップで構成され、C プログラミング言語で実装されます。1 つは特定の残存パターンとコード ID のデコード行列インデックスの計算であり、もう 1 つはデコード行列と残存データ ブロックの乗算です。

デコード行列インデックスの計算

残存パターンは、残存データ ブロックを示す 1 と消去されるデータ ブロックを示す 0 を用いたビットの文字列で表すことができます。図 3 に示す例では、残存パターンは 2 進数では `101101`、16 進数では `0x2D` と表記できます。表記を簡単にするために、残存パターンは次のように XOR 演算によって消去パターンに変換できます。

```
// translate the survival pattern into erasure pattern
unsigned short errpat = survival_pattern^((1<<RSCODE_LEN)-1);
```

ここで、この場合では (k+m) と等しい `RSCODE_LEN` は、パリティを含むデータ ブロックの総数です。消去パターンとデコード行列との間には 1 対 1 のマッピング関係がありますが、(k+m) が大きい場合、ルックアップ テーブルは大きくなります。たとえば、m = 12 かつ k = 4 であるとすると、(12+4)! / 12! / 4! = 1,820 のデコード行列があることになり、ルックアップ テーブルのサイズは $2^{(12+4)} \times \text{ceil}(\log_2 1820) = 720,896$ ビットになります。これは 40 個の BRAM18K を占めるビット数です。特定の消去パターンからデコード行列インデックスを計算するには、さらに効率的な方式が必要です。

式 2 に示すように、整数を 2 進数形式で書きます。

$$\bar{z} = \sum_{i=0}^{L-1} z_i \bullet 2^i \qquad \text{式 2}$$

ここで、$z_i \in \{0,1\}$ および L はビットの数です。

$\Omega(\bar{y}_{L,p}) = \{\bar{z} | \bar{y}_{L,p} < \bar{z} < 2^L$ および $\sum_{i=0}^{L-1} z_i = p\}$ の整数セットを定義します。

ここで $\bar{y}_{L,p}$ は、p 個の消去されたデータ ブロックの消去パターンを表す、L ビット長の整数です。Ω の構文により、セット内のすべての L ビットの整数は $\bar{y}_{L,p}$ よりも必ず大きくなり、厳密に p 個のビットが 1 になります。どの消去パターン $\bar{y}_{L,p}$ でも、$\Phi(\bar{y}_{L,p})$ は $\Omega(\bar{y}_{L,p})$ 内の要素数を示します。$\Phi(\bar{y}_{L,p})$ には次の反復関係が見られます。

$$\Phi(\bar{y}_{L,p}) = \begin{cases} 0 & \text{if } L \le 1 \text{ or } p = 0, \\ \Phi(\bar{y}_{L-1,p}) + f(L,p) & \text{if } y_{L-1} = 0, \\ \Phi(\bar{y}_{L-1,p-1}) & \text{otherwise.} \end{cases}$$

where

$$f(L,p) = \begin{cases} 0 & \text{if } p = 0, \\ 1 & \text{if } p = 1, \\ \dfrac{(L-1)!}{(p-1)!\,(L-p)!} & \text{otherwise.} \end{cases}$$

$\Phi(\bar{y}_{m+k,k})$ を、消去パターン $\bar{y}_{m+k,k}$ のデコード行列インデックスとして定義します。上記の反復関係は、次のような C コードで記述されたビット単位の走査アルゴリズムになります。

```
// pad 1's to the msb of the error pattern
unsigned char  padlen  = (3^(codeidx&3))<<1;
unsigned short padmask = ((1<<padlen)-1)<<(RSCODE_LEN-padlen);
unsigned short errpat  = erasure_pattern | padmask;

// Initialize the number of remaining ones = (p+1)
char p = NUM_EQUATION-1 + padlen;

// Scan the bits of errpat from MSB to LSB
for(k=0; k<RSCODE_LEN; k++){

   // number of remaining bits = (L+1)
   unsigned char L = (k^0xf)&0xf;

   // extract the msb of the remaining bits
   unsigned char msb = (errpat>>(RSCODE_LEN-1))&1;

   //if this bit is 0, then count the number of integers larger than it
   //if this bit is 1, decrease p by 1
   if (msb) p--;
   else if(p>=0) decmat_idx+=F_tbl[((p&3)<<4) | (L&0xf)];

   // dump the msb and move to the next bit
   errpat=(errpat<<1);
}
```

ここで f(L, p) 関数は、ルックアップ テーブル `F_tbl` により実現します。複数の消去コード タイプをサポートするために、最初にベース アドレスが `decmat_idx` に割り当てられ、ベース アドレスに対するオフセットとして $\Phi(y_{m+k,k})$ が計算されます。

図 6 は、デコード行列インデックス計算の一例です。コード ID は 0x1 で、残存パターン内に 8 + 4 = 12 の有効ビットを持つ RS(8, 4) コードであることを示しています。消去パターンはビット 12 から 15 に 1 がパディングされています。デコード行列インデックス `decmat_idx` は、RS(8, 4) コードの ROM ベース アドレス、256 で初期化されます。`errpat` の 5 つの最上位ビットはすべて 1 であるため、`decmat_idx` の値はビット 10 まで同じです。ビット 10 では f(L,p) = f(10,3) = 45 によるインクリメントが行われます。次に続く 3 つのビットであるビット 9 から 7 まではすべて 0 であるため、ビット 6 で次の 1 に達するまで、`decmat_idx` の値は 386 にインクリメントされます。この反復プロセスは、ビット文字列の末尾まで続行されます。

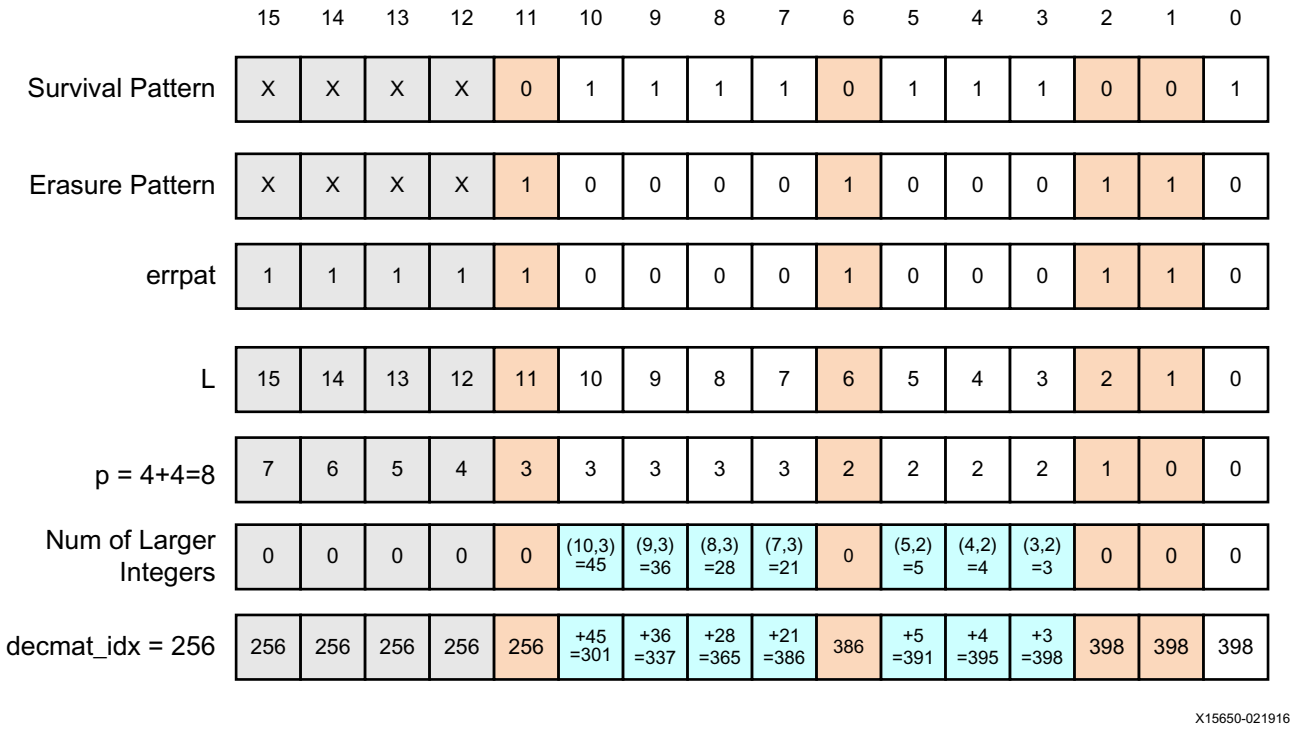


図 6 : デコード行列インデックス計算の一例 (codeid = 1、survival_pattern = 0x7B9)

GF(2^n) での行列乗算

GF(2^n) では、すべての整数が項 x^i の係数を示す i 番目のビットで多項式を表します。x はガロア体生成多項式のルートです。次に、データ ブロック d の 1 つのシンボルとデコード行列 r の 1 つの要素を、次の多項式形式に書き込むことができます。

$$\overline{d} = \sum_{i=0}^{n-1} d_i x^i, \qquad d_i \in \{0,1\}$$

$$\overline{r} = \sum_{j=0}^{n-1} r_j x^j, \qquad r_j \in \{0,1\}$$

Therefore,

$$\overline{r} \cdot \overline{d} = \overline{r} \cdot \sum_{i=0}^{n-1} d_i x^i = \sum_{i=0}^{n-1} d_i \cdot (x^i \cdot \overline{r}) = \sum_{i=0}^{n-1} d_i \cdot R_i$$

where

$$R_i \triangleq x^i \cdot \overline{r} = x \cdot x^{i-1} \cdot \overline{r} = x \cdot R_{i-1}.$$

with $R_0 = \overline{r}$.

GF(2^n) での x による乗算では、整数の単純な左シフトの後に生成多項式による簡約演算が続きます。したがって、R_i は反復的に計算でき、$d_i \in \{0,1\}$ での乗算は 0 のマルチプレクサーで計算できます。

具体的には、GF(2^n) での行列乗算は、次の C コードで記述できます。

```
// initialize the decoding matrix polynomial
for(k=0;k<NUM_ELEMENT;k++) r[k] = DECMAT_ROM[decmat_idx][k];

// reset all the code bits
for(k=0; k<NUM_EQUATION; k++) c[k] = 0;

// loop for all the bits of the input data
for(i=0; i<GF_ORDER; i++)
   // loop for all survived data blocks
   for(j=0; j<NUM_TAPS; j++)
      // loop for all coding equations
      for(k=0; k<NUM_EQUATION; k++){
         unsigned char idx = k*NUM_TAPS+j;
         unsigned char  tmp = r[idx];
         // update c
         c[k] = c[k] ^  ( ((d[j]>>i)&1)? tmp : 0 );
         // update r
         r[idx] = ((tmp>>7)&1) ? ((tmp<<1)^gf_poly) : (tmp<<1);
```

指示子

C コードを HDL に変換する際、Vivado HLS はパラメーターを記述するためにいくつかの福次情報を必要とします。この情報には、ループの完了に使用できるクロック サイクル数や、モジュールが古い入力をすべて処理する前に新しい入力を受け入れることができるかどうかなどがあります。デザインの重要部分であるこれらの指示子は、理想的な動作のために C コードを HDL に合成する方法を指定します。既存のデザインを新しいアプリケーションに移植する場合、C コードには変更を加えず、指示子にわずかな変更を加えるだけで済むことがあります。

消去コーデックの場合、理想的な動作を実現するために次の指示子が使用されます。指示子は最小限にとどめられるので、後の段階で、Vivado 合成ツールはデザイン全体を把握して、より柔軟に合成オプションを選択できます。たとえば、デコード行列 ROM は、ブロック RAM または分散 RAM のいずれかにインプリメントできます。この決定をより適正に行うために、Vivado 合成ツールを使用します。このツールでは、ほかのシステム モジュールとの統合後にリソース使用率のバランスを取ることができます。

```
# The function takes pipelined architecture and accepts new inputs every clock cycle
set_directive_pipeline -II 1  rs_erasure

# the arrays c and d should be partitioned completely for the access
# to all the elements of the array in one clock cycle
set_directive_array_partition -type complete rs_erasure c
set_directive_array_partition -type complete rs_erasure d

# The interface definition
set_directive_interface -mode ap_none  rs_erasure d
set_directive_interface -mode ap_vld   rs_erasure codeidx
```

合成の結果

ザイリンクス Vivado HLS はすべてのデザイン ファイルを分析し、合成指示子の形式で指定された目標のクロック周波数とデータ スループットを満たす適切なハードウェア アーキテクチャを自動的に選択します。C コンパイルの完了後には、合成された HDL に関する基本情報を見直し、デザイン目標に照らして確認できます。図 7 は、Vivado HLS で生成された消去コーデック合成レポートを示しています。

Synthesis Report for 'rs_erasure'

General Information

Date: Thu Nov 05 13:54:28 2015
Version: 2015.3 (Build 1368829 on Mon Sep 28 20:31:51 PM 2015)
Project: ProjRSErasure
Solution: SolutionX
Product family: kintexu
Target device: xcku040-ffva1156-2-e

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	3.30	2.96	0.30

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
27	27	1	1	function

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	15418
FIFO	-	-	-	-
Instance	-	-	0	16
Memory	96	-	144	9
Multiplexer	-	-	-	10
Register	-	-	3948	114
Total	96	0	4092	15567
Available	1200	1920	484800	242400
Utilization (%)	8	0	~0	6

X15671-021916

図 7 : C 合成レポート

レポートによると、HLS で生成された RTL は 1/2.96ns = 338Mhz で動作すると推定されており、これは 300Mhz という要件を満たしています。C 合成段階でのクロック周波数推定は、最終結果と ±15% の誤差が生じる可能性があります。これは配置配線が完了するまでロジック配線レイテンシが確定しないためです。さらに合成レポートは、デザインのデコード レイテンシが 27 クロック サイクル (300Mhz クロックでは約 90ns) であることを示しています。パイプライン処理の入力間隔は 1 です。これはコーデックの最大スループットが、入力の場合は 12 データ ブロック × データ ブロックあたり 8 ビット × 300MHz = 28.8Gb/s、出力の場合は 4 データ ブロック × データ ブロックあたり 8 ビット × 300MHz = 9.6Gb/s であることを意味します。必要な場合、複数の消去コーデックを並列に機能するようにインスタンシエートして、より高いデータ スループットを実現できます。

検証結果

Vivado HLS デザイン フローでは、機能検証は 2 つのステップで構成されています。

最初のステップは、ゴールデン テスト ベクターに照らして C コードを検証する、C 機能検証です。C ライブラリはリッチ ファイル I/O 機能を提供しているため、格納済みの入力および出力テスト ベクターに基づいて C テスト ベンチをコーディングすることは簡単です。消去コーデックのリファレンス デザインは、このアプローチに従います。ゴールデン テスト ベクターには、4 つすべての消去コード レートに対応する実行数 1000 の消去デコードが含まれており、各コード レートに対してランダムに選択された 25 の残存パターン、および各残存パターン用の 10 セットのデータ ブロックが設定されています。

C の動作が検証され、C の関数が HDL に合成されると、Vivado HLS は C のテスト コードに従って HDL テスト ベンチを自動的に生成できます。このステップは、C/RTL 協調シミュレーションと呼ばれ、HDL の動作と C の機能を確実に一致させます。図 8 に示すとおり、Vivado HLS は C/RTL 協調シミュレーション用に多数のシミュレータと HDL コード形式をサポートしています。

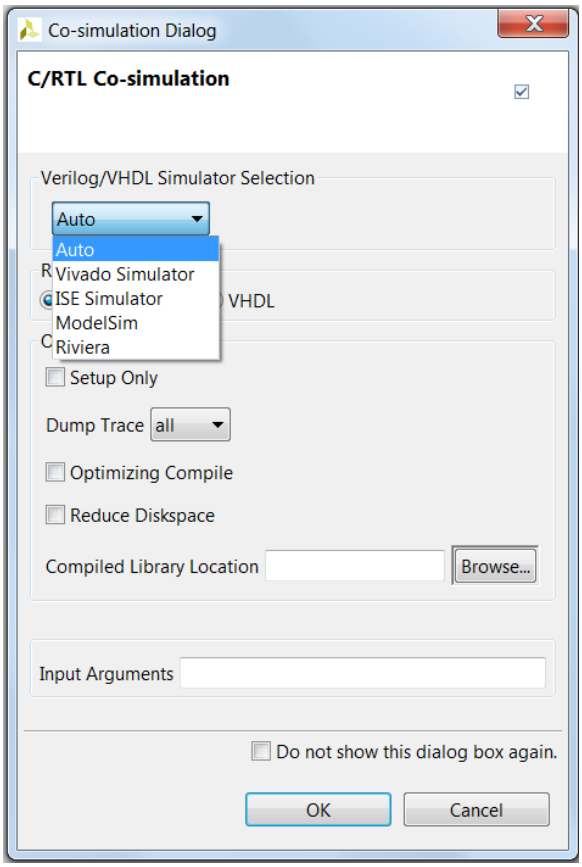


図 8 : C および HDL 協調シミュレーション ダイアログ ウィンドウ

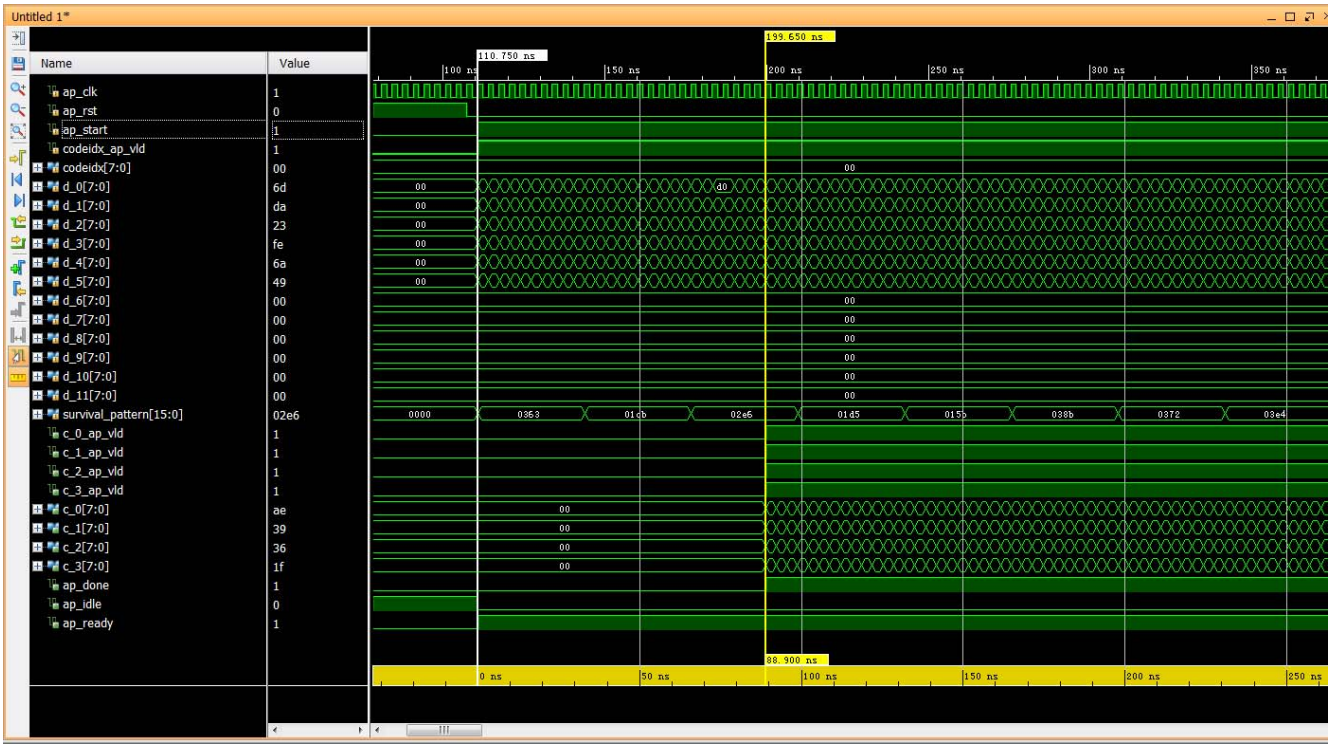
HDL デザインの出力は、機能が正しいことを確認するために、ゴールデン テスト ベクターの出力と比較されます。シミュレーションの最後に、Vivado HLS は次のようなチェック後の結果を出力します。

…..

```
****** xsim v2015.3 (64-bit)
  **** SW Build 1368829 on Mon Sep 28 20:06:43 MDT 2015
  **** IP Build 1367837 on Mon Sep 28 08:56:14 MDT 2015
    ** Copyright 1986-2015 Xilinx, Inc. All Rights Reserved.

source xsim.dir/rs_erasure/xsim_script.tcl
# xsim {rs_erasure} -maxdeltaid 10000 -autoloadwcfg -tclbatch {rs_erasure.tcl}
Vivado Simulator 2015.3
Time resolution is 1 ps
source rs_erasure.tcl
## add_wave -r /
WARNING: Simulation object /apatb_rs_erasure_top/next_trigger_ready_cnt was not
traceable in the design for the following reason:
Vivado Simulator does not yet support tracing of Verilog named events.
## save_wave_config rs_erasure.wcfg
## run all
$finish called at time : 3512850 ps : File
"C:/MattRuan/appnote/ProjRSErasure/SolutionX/sim/verilog/rs_erasure.autotb.v" Line
1276
## quit
INFO: [Common 17-206] Exiting xsim at Thu Nov 05 13:58:56 2015...
[0]   0 out of 250 test vectors failed.
[1]   0 out of 250 test vectors failed.
[2]   0 out of 250 test vectors failed.
[3]   0 out of 250 test vectors failed.
Total 1000 Test Vectors, Err Count = 0.
Test Passed!
@I [SIM-1000] *** C/RTL co-simulation finished: PASS ***
```

波形での手動デバッグのために、信号トレースをダンプするオプションもあります。図 9 は、多様なコード レートと残存パターンのテスト ケース用に Vivado Simulator 2015.3 が生成したシミュレーション波形です。図 9 に示す波形から、マルチモード消去コーデックのレイテンシが 88.9ns/3.3ns = 27 クロック サイクルであり、合成レポートの値と一致することが確認できます。

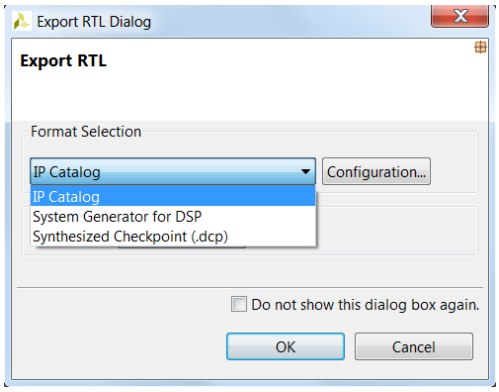


X15673-021916

図 9 : C および HDL 協調シミュレーションの波形

インプリメンテーションの結果

ザイリンクス Vivado HLS は、C 関数の HDL コードを生成するだけでなく、HDL を IP にパッケージする多数のオプションを提供するため、System Generator、エンベデッド開発キット (EDK)、IP インテグレーターなどの Vivado Design Suite を使用して、より大規模なデザインに容易に統合できます。例示目的で、サンプルのリファレンス デザイン用に IP カタログが選択されています (図 10 参照)。



X15674-021916

図 10 : HDL エクスポート ダイアログ ウィンドウ

Vivado HLS ツールは、Vivado プロジェクトを自動的に作成してすべての HDL コードを合成し、インプリメンテーションのパフォーマンスを検証します。次のような消去コーデックの最終インプリメンテーション レポートによって、デザイン目標が達成されていることを確認します。

```
Implementation tool: Xilinx Vivado v.2015.3
Device target:       xcku040-ffva1156-2-e
Report date:         Thu Nov 05 14:30:47 +0800 2015
#=== Resource usage ===
CLB:           4411
LUT:          26010
FF:            5264
DSP :             0
BRAM:             0
SRL:            110
#=== Final timing ===
CP required:    3.300
CP achieved:    3.143
Timing met
```

このレポートは、合成ツールが、デコード行列 ROM のインプリメンテーションに、ブロック RAM ではなく分散 RAM を選択したことを示しています。これは、RS(12, 4) のみがフル ストレージ容量を使用し、その他のコードではデコード行列 ROM に多くのゼロがパディングされているためです。合成ツールは ROM のスパース性を検出し、タイミング パフォーマンスを向上させるために ROM を分散 RAM にインプリメントすることを決定します。

リファレンス デザイン

このアプリケーション ノートのリファレンス デザイン ファイルは、ザイリンクスのウェブサイトからダウンロードできます。

デザイン ファイルの階層

最上位フォルダーの下のディレクトリ構造は次のとおりです。

```
\src
 | This folder contains C design files and header files.
 |
\tb
 | This folder contains a C design file that serves as the test bench.
 |
\tv
 | This folder contains the input and output golden test vectors for
 |   verification purposes.
 |
```

インストールと操作の手順

1. ザイリンクス Vivado ツールのバージョン 2015.3 以降をインストールします。
2. デザイン ファイルをクリーン ディレクトリに解凍します。
3. Vivado HLS コマンド ライン ウィンドウで、次を十国します。
   a. cd を実行して、デザインのルート ディレクトリに移動します。
   b. 「vivado_hls run.tcl」と入力します。
   c. 合成されたデザインが想定を満たしていることを確認します。

表 2 に、リファレンス デザインの詳細を示します。

表 2 : リファレンス デザインの詳細

パラメーター	説明
一般	
開発者	Matt Ruan
ターゲット デバイス	Kintex® UltraScale™ FPGA (KU040、KU060) Virtex®-7 FPGA (7V690) **注記** : リファレンス デザインは Virtex-7 および Kintex UltraScale FPGA でテストされていますが、Zynq UltraScale+ MPSoC、さらには Artix®-7 や Spartan®-6 FPGA といったほかのデバイスでも動作するはずです。Spartan-6 および Artix-7 デバイスで達成可能なスループットは、Kintex および Virtex UltraScale デバイスで達成可能なスループットよりも低いと予想されます。
ソース コードの提供	あり
ソース コードの形式	C、テスト ベクター、および合成スクリプト
既存のザイリンクス アプリケーション ノート/リファレンス デザイン、またはサードパーティからデザインへのコード/IP の使用	なし
シミュレーション	
論理シミュレーションの実施	あり
タイミング シミュレーションの実施	なし
論理シミュレーションおよびタイミング シミュレーションでのテストベンチの利用	あり
テストベンチの形式	C
使用したシミュレータ/バージョン	Vivado Simulator 2015.3
SPICE/IBIS シミュレーションの実施	なし
インプリメンテーション	
使用した合成ツール/バージョン	Vivado 高位合成 2015.3
使用したインプリメンテーション ツール/バージョン	Vivado Design Suite 2015.3
スタティック タイミング解析の実施	あり
ハードウェア検証	
ハードウェア検証の実施	なし
検証に使用したハードウェア プラットフォーム	N/A

まとめ

このアプリケーション ノートは、C コードを入力として利用し、FPGA 上で合成可能な HDL コードを生成する Vivado HLS ツール チェーンを使用して、低レイテンシのマルチレート消去コーデックを作成する方法を示しています。C ソース コードは維持が簡単であり、多様な FPGA デバイス、消去コード レート、およびシステム クロック周波数に合わせて拡張できます。デコード行列 ROM を変更すれば、パイプライン化された Gf(2n)^n) 汎用行列乗算アーキテクチャからほかのタイプの消去コーデックの作成も容易です。

参考資料

注記：日本語版のバージョンは、英語版より古い場合があります。

1. Burkhard, W. A. and Menon, J. (June 1993) "Disk array storage system reliability."Proceedings of the 23rd Annual International Symposium on Fault-Tolerant Computing, pp. 432–441
2. Dimakis, A.G.; Prabhakaran, V.; Ramchandran, K. (June 2006) "Decentralized erasure codes for distributed networked storage."IEEE Transactions on Information Theory, pp. 2809–2816, Volume 52, Issue 6
3. Sobe, Peter (March 2009) "Coding for Reliable Data Storage on Different Hardware Platforms."Architecture of Computing Systems - ARCS 2009: 22nd International Conference Proceedings, pp. 1-6
4. 『Vivado Design Suite ユーザー ガイド : 高位合成』(UG902 : 英語版、日本語版)

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	内容
2016 年 3 月 14 日	1.0	初版

法的通知

この資料に関するフィードバックおよびリンクなどの問題につきましては、jpn_trans_feedback@xilinx.com まで、または各ページの右下にある [フィードバック送信] ボタンをクリックすると表示されるフォームからお知らせください。いただきましたご意見を参考に早急に対応させていただきます。なお、このメール アドレスへのお問い合わせは受け付けておりません。あらかじめご了承ください。