

LogiCORE IP Soft Error Mitigation Controller v4.1

製品ガイド

Vivado Design Suite

PG036 2015 年 9 月 30 日

本資料は表記のバージョンの英語版を翻訳したもので、内容に相違が生じる場合には原文を優先します。資料によっては英語版の更新に対応していないものがあります。日本語版は参考用としてご使用の上、最新情報につきましては、必ず最新英語版をご参照ください。

目次

IP の概要

第 1 章：概要

メモリ タイプ	5
エラー軽減のアプローチ	6
信頼性の推定	6
機能概要	7
アプリケーション	8
サポートされていない機能	8
ライセンスおよび注文情報	9

第 2 章：製品仕様

機能	10
準拠する規格	11
パフォーマンス	11
リソース使用状況	21
ポートの説明	24

第 3 章：コアを使用するデザイン

インターフェイス	29
ビヘイビア	47
システム	58
カスタマイズ	60
データ整合性	65
コンフィギュレーション メモリのマスク	65
クロッキング	66
リセット	66
その他の注意事項	67

第 4 章：デザイン フローの手順

コアのカスタマイズおよび生成	68
コアへの制約	74
シミュレーション	81
合成およびインプリメンテーション	81
統合およびバリデーション	81

第 5 章 : サンプル デザイン

機能	83
ポートの説明	86
デモンストレーション用テストベンチ	90
インプリメンテーション	90
外部メモリ プログラミング ファイル	93

付録 A : 検証、互換性、相互運用性

検証	95
バリデーション	96
適合性検査	96

付録 B : 移行およびアップグレード

Vivado Design Suite への移行	97
Vivado Design Suite でのアップグレード	97

付録 C : デバッグ

ザイリンクス ウェブサイト	98
デバッグ ツール	99
ハードウェア デバッグ	100
インターフェイスのデバッグ	100
クロッキング	101

付録 D : その他のリソースおよび法的通知

ザイリンクス リソース	102
参考資料	102
改訂履歴	103
お読みください : 重要な法的通知	104

はじめに

LogiCORE™ IP Soft Error Mitigation (SEM) Controller は、ザイリンクス FPGA のコンフィギュレーション メモリで発生したソフト エラーを検出および訂正するための事前検証済みソリューションで、自動的に設定されます。ソフト エラーとは、ステート エLEMENT に格納した値が電離放射線によって変化してしまうことをいいます。

SEM Controller はソフト エラーを防ぐのではなく、ソフト エラーのシステム レベルでの影響をより適切に管理するための手段を提供します。これらイベントを適切に管理することで信頼性と可用性が向上し、システム メンテナンスとダウンタイムのコストを削減できます。

機能

- 多くのデバイスにおいて標準的な検出レイテンシは 25ms。
- FPGA 内蔵のエラー検出機能を完全に利用しつつさらに発展させたビルトイン シリコンプリミティブを統合。
- オプションのエラー訂正機能。修復による訂正、拡張修復による訂正、または置換による訂正を選択可能。
 - 修復による訂正は ECC アルゴリズムで実行。
 - 拡張修復による訂正は ECC および CRC アルゴリズムで実行。
 - 置換による訂正はデータの再読み込みによって実行。
- ザイリンクスのエッセンシャル ビット テクノロジーを利用したオプションのエラー分類機能により、ソフト エラーがユーザー デザインの機能に影響したかどうかを判定。
 - 実際のデザインの動作に影響しないエラーについては、動作を中断してリカバリを実行する必要がなくなるため、アップタイムが向上。
 - 実効 FIT (Failures In Time) が改善。
- オプションのエラー挿入機能により、SEM Controller のアプリケーションの評価をサポート。

この LogiCORE IP について	
コアの概要	
サポートされるデバイス ファミリー ⁽¹⁾	Zynq®-7000 All Programmable SoC、7 シリーズ
サポートされるユーザー インターフェイス	RS-232、SPI
リソース	表 2-10 ~ 表 2-14 を参照
コアに含まれるもの	
デザイン ファイル	暗号化済み RTL
サンプル デザイン	VHDL および Verilog
テストベンチ	VHDL および Verilog ⁽²⁾
制約ファイル	XDC
シミュレーション モデル	Verilog/VHDL ビヘイビア モデル またはソース HDL ⁽²⁾
サポートされるソフトウェア ドライバー	N/A
テスト済みデザイン フロー⁽³⁾	
デザイン入力	Vivado® Design Suite
シミュレーション	サポートされるシミュレータについては、『 Vivado Design Suite ユーザー ガイド：リリース ノート ガイド、インストール およびライセンス 』を参照 ⁽²⁾
合成	Vivado 合成
サポート	
ザイリンクス サポート ウェブ ページ で提供	

注記：

1. サポートされているデバイスの一覧は、Vivado IP カタログを参照してください。
2. SEM Controller を含むデザインの機能およびタイミング シミュレーションがサポートされます。ただしシミュレーションでは SEM Controller の動作は観察できません。これにはハードウェアベースの評価が必要です。
3. サポートされているツールのバージョンは、『[Vivado Design Suite ユーザー ガイド：リリース ノート ガイド、インストール およびライセンス](#)』を参照してください。

概要

ほとんどのシリコン デバイスは、電離放射線から望ましくない影響を受ける可能性があります。1 回のイベントで生じる望ましくない影響を、シングル イベント効果 (SEE) と総称します。ほとんどの場合、このイベントによってシリコン デバイスが恒久的な損傷を受けることはありません。このように、デバイスに恒久的な損傷を与えない SEE をソフト エラーと呼びます。ただしソフト エラーによって信頼性が低下する可能性があります。

ザイリンクスのデバイスはソフト エラーの影響を受けにくいように設計されています。ただし、商業性などを考慮した現実的な制約の中でソフト エラーを完全になくすのは不可能であることもザイリンクスは認識しています。このため、ザイリンクスの多くのデバイス ファミリはソフト エラーの検出/訂正機能を内蔵しています。

ソフト エラーは多くのアプリケーションで無視できます。高い信頼性が要求されるアプリケーションも、通常は内蔵のソフト エラー検出/訂正機能で十分に対処できます。特に要求の厳しいアプリケーションでは、SEM Controller を使用することでさらに高い信頼性を確保できます。

メモリ タイプ

ソフト エラーが発生すると、1 ビットまたは複数ビットのメモリ内容が破損します。デバイスのコンフィギュレーション メモリでソフト エラーが発生すると、デザインの動作が影響を受け、デザインのメモリ素子で発生するとデザインのステートが影響を受けます。デバイスには、主に次の 4 種類のメモリがあります。

- **コンフィギュレーション メモリ** : このストレージ素子を使用して、デバイスに読み込まれるデザインの機能を設定します。これには、ファンクションブロックの動作および接続がそれぞれ含まれます。このメモリはデバイス全体に物理的に分散しており、デバイス内で最もビット数の多いメモリです。ただし、デバイスに読み込まれるデザインの実際の動作に影響するエッセンシャルビットは全体のごく一部です。
- **ブロック メモリ** : デザイン ステートの格納に使用する大容量のストレージ素子です。名前が示すように、これらのビットは物理的なブロックにまとめられており、デバイス全体にいくつかのブロックが分散しています。ブロック メモリはデバイス内で 2 番目にビット数の多いメモリです。
- **分散メモリ** : デザイン ステートの格納に使用中容量のストレージ素子です。分散メモリは一部のコンフィギュラブル ロジック ブロック (CLB) に存在し、デバイス全体に分散しています。分散メモリはデバイス内で 3 番目にビット数の多いメモリです。
- **フリップフロップ** : デザイン ステートの格納に使用する小容量のストレージ素子です。このメモリはすべての CLB に存在し、デバイス全体に分散しています。フリップフロップはデバイス内で 4 番目にビット数の多いメモリです。

これ以外のメモリとして、内部デバイス制御レジスタおよびステート エレメントがありますが、これらのビット数はごくわずかです。これらのメモリ領域でソフト エラーが発生すると、局所的またはデバイス全体に障害が起こることがあり、これをシングル イベント ファンクショナル インターラプト (SEFI) と呼びます。これらのメモリはビット数が少ないため、ここでは SEFI イベントの発生確率は無視できるものと見なします。また、ほとんど発生することのないこれらのイベントには SEM Controller は対処しません。

エラー軽減のアプローチ

ブロックメモリ、分散メモリ、フリップフロップに格納したデザインステートに対するソフトエラーは、エラー検出/訂正コードや冗長性などの標準的な手法を用いてデザイン自体で軽減策を実行できます。未使用のデザインステートリソース、すなわちデバイスに物理的に存在していてもデザインが使用していないリソースで発生したソフトエラーは無視されます。特に信頼性が重視されるデザインではリスクエリアを評価し、保証内容に応じてデザインステートに対するエラー軽減手法を採用する必要があります。

コンフィギュレーションメモリに格納されたデザイン機能に対するソフトエラーは、エラー検出および訂正コードを使用して軽減を図ります。

コンフィギュレーションメモリは幅の広いスタティックRAMのようにフレームの配列として構成されます。各フレームは多くのデバイスファミリでECCによって保護されており、フレームの配列全体はすべてのデバイスファミリでCRCによって保護されています。これら2つの手法は補完的な関係にあり、CRCはエラー検出性能が非常に高く、ECCはエラー位置を高精度で特定します。

SEM Controllerはデバイス内蔵ロジックの強力な機能をさらに発展させ、コンフィギュレーションメモリのエラーをエッセンシャルと非エッセンシャルに分類するオプション機能を備えています。この機能は、実際のデザインの動作に影響するエッセンシャルビットがコンフィギュレーションメモリビット全体のごく一部であることを利用しています。

エラー分類を無効にした場合、コンフィギュレーションメモリのすべてのエラーをエッセンシャルと見なす必要があります。エラー分類を有効にするとほとんどのエラーが非エッセンシャルと判定されるため、実害のないエラーに対する警告を防ぐことができ、システムレベルでの動作中断を伴うエラー軽減策の実行頻度が低下します。

また、SEM Controllerはデバイス内蔵の訂正機能を拡張してエラー検出を高速化すると共に、オプションで複数ビットエラーの処理にも対応します。

デバイス内蔵のソフトエラー検出/訂正機能だけでSEUを十分に軽減できる場合、SEM Controllerの機能は不要です。7シリーズFPGAおよびZynq[®]-7000 SoCに内蔵のエラー検出/訂正機能の詳細な使用法は、『7シリーズFPGAコンフィギュレーションユーザーガイド』(UG470) [参照 1] を参照してください。

信頼性の推定

システム信頼性に関する仕様を決定するには、まずシステムデザイン全体のうち特に重要なセクションを特定し、セクションごとに必要な信頼性の値を決定していく必要があります。一般に、信頼性の要件はFIT (Failures In Time) で表します。これは、 10^9 時間 (約 114,155 年) あたりに予想されるデザインの故障回数です。

同じデザインを複数出荷する場合、いずれか1つのデザインがソフトエラーの影響を受ける確率は出荷台数に比例して大きくなります。たとえばあるデザインを製品として1,000個出荷した場合、出荷全数の公称FITは1,000倍になります。出荷全数の公称FITが大きくなると保守/メンテナンスの負担が増大することが考えられます。

この公称FITは、個々のデザインが影響を受ける確率とは異なります。また、ある特定のデザインで2回目のエラーが発生する確率は、出荷全数のFITではなくデザイン個体のFITによって決まります。これは、個々のアプリケーションに適したソフトエラー軽減方針を検討する際に考慮すべき重要な点です。

ソフトエラーに関するFITと製品の推定耐用年数に関連するFITは区別して考える必要があります。後者はシステムの部品の一部交換または物理的な修理が必要になる故障を想定したものです。

ザイリンクスデバイスのFITデータは『デバイス信頼性レポート』(UG116) [参照 2] に記載しています。このデータは、全体的にソフトエラーがほとんど発生しないことを示しています。

ヒント : 故障率は非常に低いため、ソフトエラー軽減策を含むデザインはほとんどありません。



フリップフロップは数が少なく FIT の値も非常に小さいため、デザイン全体の FIT にはほとんど影響しません。とはいえ、フリップフロップに格納されたデザイン ステートを保護することの重要性は変わりません。フリップフロップに格納されたステートがデザインの動作にとってきわめて重要な場合、ソフト エラーを検出および訂正してエラーから回復するためのロジックをアプリケーションに適した形でデザインに実装する必要があります。

分散メモリやブロック メモリを多用するデザインでは、これらリソースがデザイン全体の FIT に大きく影響する可能性があります。前述のとおり、デザインでソフト エラー軽減策をとることで、デザイン全体の FIT への影響を大幅に抑えることができます。たとえばブロック メモリ リソースにはエラー検出/訂正回路が内蔵されており、ブロック メモリのコンフィギュレーションによってはこの軽減機能を利用できます。プログラマブル ロジック リソースを使用したソフト エラー軽減手法は、コンフィギュレーションに依存せずすべてのブロック メモリと分散メモリに使用できます。

コンフィギュレーション メモリはデザイン全体の FIT に大きく影響します。エラー分類機能を使用しない場合、コンフィギュレーション メモリで発生したソフト エラーはすべてエッセンシャルと見なす必要があります。その場合、コンフィギュレーション メモリによる FIT への影響がその他の要因による影響より圧倒的に大きくなります。エラー分類機能を使用するとソフト エラーのほとんどを故障と見なす必要がなくなるため、デザイン全体の FIT への影響を抑えることができます。実害のないソフト エラーは、動作を中断せず訂正できます。

最高レベルの信頼性が要求されるデザインでは、コンフィギュレーション メモリで発生したソフト エラーの分類が必須です。SEM Controller はこの機能を提供します。

機能概要

SEM Controller は、初期化、エラー挿入、エラー検出、エラー訂正、およびエラー分類の主に 5 つの機能を実装しています。初期化とエラー検出を除く機能はすべてオプションで、必要な機能を IP コアの設定および生成プロセスで選択します。

初期化では、FPGA がユーザー モードになった後、SEM Controller が FPGA 内蔵のソフト エラー検出機能を既知のステートに移行させます。この初期化の後、SEM Controller は内蔵のソフト エラー検出機能のステータスを監視します。ECC または CRC エラーを検出すると、SEM Controller は状況を評価してエラーの発生したコンフィギュレーション メモリ位置を特定します。

位置を特定できた場合、SEM Controller はオプションで修復/拡張修復または置換によりソフト エラーを訂正します。修復/拡張修復による訂正はアクティブ パーシャル リコンフィギュレーションを使用し、Read-Modify-Write によってコンフィギュレーション メモリを部分的に訂正します。これらの方法は、アルゴリズムを使用して訂正に必要なエラーを特定します。置換による訂正も同じ目的でアクティブ パーシャル リコンフィギュレーションを使用しますが、この場合は書き込みのみを使用してコンフィギュレーション メモリを元のデータで置き換えます。このデータはインプリメンテーション ツールによって提供され、SEM Controller の外部に格納されます。

SEM Controller には、ルックアップ テーブルを使用してソフト エラーがエッセンシャルかそうでないかを分類するオプション機能があります。エラー分類の実行中、必要に応じて情報がフェッチされます。このデータもインプリメンテーション ツールによって提供され、SEM Controller の外部に格納されます。

SEM Controller がアイドル状態の場合、ユーザーからの入力によってコンフィギュレーション メモリにエラーを挿入することもできます。この機能は、より大規模なシステム デザインに統合した SEM Controller をテストする際に役立ちます。システム検証/バリデーション エンジニアはエラー挿入機能を使用してテスト ケースを構築し、システム全体がソフト エラー イベントに予想どおりの応答を示すかどうかを確認できます。

アプリケーション

SEM Controller は自律動作が可能です。ほとんどのアプリケーションではこのソリューションをアプリケーションレベルの監視機能と組み合わせて使用します。この監視機能は SEM Controller からのイベント報告を監視し、デバイスのリコンフィギュレーションやアプリケーションのリセットなど、何らかの追加措置が必要かどうかを判断します。

システムを設計する際は、各デザインの信頼性要件、および各種情報に基づいて判断を下すシステムレベルの監視機能を慎重に検討してください。

そもそも、エラー軽減ソリューションが必要なのかを検討します。次に、ターゲットデバイス内蔵のソリューションでアプリケーション要件を満たすことができるか、SEM Controller が必要かを判断します。SEM Controller が必要な場合は、どの機能を使用するかを検討します。

アプリケーション要件を考慮して SEM Controller が最善の選択肢であると判断した場合、外部デバイスとの接続に使用するシステムレベル サンプル デザインのコンポーネントを含め、SEM Controller を提供された状態のまま使用することを推奨します。ただし必要であれば、これらのインターフェイスをアプリケーションに合わせて変更することも可能です。



推奨：ザイリンクスは、SEM IP コアをなるべく早い段階で (理想的にはプロジェクトの最初から) 統合することを推奨しています。詳細は、[81 ページの「統合およびバリデーション」](#)を参照してください。

サポートされていない機能

SEM Controller は、ブロックメモリ、分散メモリ、またはフリップフロップで発生したソフトエラーに対しては動作しません。これらメモリリソースで発生したソフトエラーは、冗長性またはエラー検出/訂正コードなどの予防手段を用いてユーザーロジックで軽減策をとる必要があります。

SEM Controller は、ザイリンクス Zynq-7000 デバイスなどに内蔵されるプロセッサシステム内のステートエレメントで発生したソフトエラーに対しては動作しません。これらのメモリリソースで発生したソフトエラーは、プロセッサシステム上で動作するソフトウェアで軽減策をとる必要があります。

サポートされない機能や制限事項に関連して、機能、インプリメンテーション、および使用に関する注意事項があります。詳細は、[第 3 章の「その他の注意事項」](#)を参照してください。

ライセンスおよび注文情報

このザイリンクス LogiCORE IP モジュールは、[ザイリンクス エンドユーザー ライセンス規約](#)のもとザイリンクス Vivado® Design Suite を使用して追加コストなしで提供されています。

この IP およびその他のザイリンクス LogiCORE IP に関する情報は、[ザイリンクス IP コア](#) ページから入手できます。その他のザイリンクス LogiCORE IP モジュールおよびツールの価格や提供状況については、[ザイリンクス販売代理店](#)にお問い合わせください。

ライセンス チェッカー

IP にライセンス キーが必要な場合、そのキーの認証が必要です。Vivado デザイン ツールでは、設計フローにライセンスが必要な IP の使用をゲーティングする、ライセンスチェックポイントが複数あります。ライセンスチェックが正常に終了すると、IP の生成が継続されます。正常に終了しなければ、IP の生成はエラーとなり停止します。ライセンスチェックポイントが適用されるのは、次のツールです。

- Vivado 合成
- Vivado インプリメンテーション
- write_bitstream (Tcl コマンド)



重要 :チェックポイントでは、IP のライセンス レベルは無視されます。有効なライセンスの有無のみを検証します。IP ライセンス レベルは確認しません。

製品仕様

この章では、LogiCORE IP Soft Error Mitigation (SEM) Controller (以下、SEM Controller またはコントローラー) の仕様について説明します。コンフィギュレーション メモリ内のソフト エラーを緩和するこの設定可能なコントローラーには、実際のシステムでの使用例を示したシステム レベルのサンプル デザインも付属します。

機能

SEM Controller には次の機能があります。

- デバイス内蔵のエラー検出機能を利用するシリコン機能を統合。
- ソフト エラーの訂正をサポートしたエラー訂正機能を実装。エラー訂正方法は次のように定義されます。
 - **修復**: ECC アルゴリズムに基づく訂正。この方法は、コンフィギュレーション メモリのフレームで発生した 1 ビットのエラーを訂正します。この方法では、1 ビットの反転イベントがすべて訂正されます。複数ビットの反転イベントの場合も、コンフィギュレーション メモリのインターリーブによってエラーが分散された結果、1 フレームにつき 1 ビットのエラーであれば訂正されます。
 - **拡張修復**: ECC および CRC アルゴリズムに基づく訂正。この方法は、コンフィギュレーション メモリのフレームで発生した 1 ビットのエラーまたは隣接する 2 ビットのエラーを訂正します。この方法では、1 ビットの反転イベントおよび隣接する 2 ビットの反転イベントがすべて訂正されます。複数ビットの反転イベントの場合も、コンフィギュレーション メモリのインターリーブによってエラーが分散された結果、1 フレームにつき 1 ビットまたは隣接する 2 ビットのエラーであれば訂正されます。
 - **置換**: データの再読み込みによる訂正。この方法は、コンフィギュレーション メモリのフレームで発生した任意の数のエラーを訂正します。この方法では、コンフィギュレーション メモリのどのフレームで反転イベントが発生したかさえわかれば、フレーム内の正確なビット位置を特定できなくても訂正が可能です。
- 訂正したエラーが実際のデザインの機能に影響するコンフィギュレーション メモリ位置で発生したかどうかを判定するエラー分類機能を実装。
- コントローラーの検証およびコントローラーのアプリケーション評価に役立つエラー挿入機能をサポート。

サンプル デザインには次のものが含まれます。

- ユーザー設定に基づくコントローラーのインスタンス化。
- コントローラーと外部ストレージを接続するためのインターフェイス。コントローラーの設定でエラー分類または置換によるエラー訂正を有効にした場合、このインターフェイスが必要です。
- コントローラーと外部プロセッサを接続するためのインターフェイス。コントローラーの設定でエラー挿入を有効にした場合、このインターフェイスによって使い勝手が向上します。

準拠する規格

このコアが対象となる、業界標準の適合性検査または認証試験は定義されていません。SEM Controller は、加速粒子の照射を含む大規模なハードウェアバリデーションを受けています。

パフォーマンス

SEM Controller のパフォーマンス メトリクスはシリコン仕様および実測に基づいて求めたものであり、事前の見積もり以外の用途には使用しないでください。実際のパフォーマンスは異なる場合があります。

ソリューションの信頼性

ここからはシステムレベル サンプル デザインを解析し、FPGA にインプリメントしたソリューション自体の FIT の推定値を求めます。この解析手法は、FPGA にインプリメントしたほかの回路の FIT を推定する際にも利用できます。

この解析では、すべての機能を有効にしてすべての信号を I/O ピンに接続した場合を想定しています。Vivado ロジック デバッグ IP は解析には含めていません。このような対話型のデバッグおよび実験機能を含めたままデザインを出荷する可能性は低いからです。このため、この推定値は上限を表しています。

推定データ

ザイリンクス デバイスの FIT データは『デバイス信頼性レポート』(UG116) [参照 2] に記載しています。表 2-1 に、信頼性の推定例で使用するサンプル データを示します。

注記: 表 2-1 のデータは一例です。このデータは説明用のサンプルであり、重要なデザインの判断材料に使用することは避けてください。各デバイスの最新の FIT データは、『デバイス信頼性レポート』(UG116) [参照 2] を参照してください。

表 2-1: デバイスの FIT データ例

メモリ セル タイプ	リアルタイム ソフト エラー率 (FIT/Mb)
コンフィギュレーション メモリ	86
ブロック メモリ	78
分散メモリ (コンフィギュレーション メモリと同じ)	86
フリップフロップ	仕様値なし

表 2-2 に、デバイスの各リソースおよびそのリソースに含まれるコンフィギュレーション メモリ セルの概算値を示します。

表 2-2: デバイス リソースあたりのコンフィギュレーションビット数

デバイス リソース (配線を含む)	コンフィギュレーションビット数 (概算値)
ロジック スライス	1,166
ブロック RAM (36Kb)	9,396
ブロック RAM (18Kb)	4,698
I/O ブロック	2,850

通常、ソフトエラーが発生して実際のデザインに直接影響するのはコンフィギュレーションメモリセル全体の 10% 未満です。したがって、次に示す信頼性推定例では 10% のディレーティング係数を使用しています。

信頼性推定例 (非 SSI デバイス)

中容量の XC7K325T デバイスですべてのオプション機能を有効にした場合、コントローラーとシムは約 250 個のロジックスライス、56 個の I/O ブロック、3 個のブロック RAM (18Kb)、および 9 個のブロック RAM (36Kb) を使用します。コンフィギュレーションビットの FIT は次の式で求めます。

$$\text{Config FIT} = 10\% \times (250 \times 1,166 + 56 \times 2,850 + 3 \times 4,698 + 9 \times 9,396) \times 86 \text{ FIT/Mb}$$

$$\text{Config FIT} = 4.5 \text{ FIT}$$

コントローラーとシムがデータに使用するフリップフロップの数は数百個でビット数が少ないため、FIT の計算には含めません。

コントローラーとシムは 65 個の LUT RAM を使用します。内訳は次のとおりです。

- MON シムがデータバッファとして 31 個の LUT RAM を使用します。ただしバッファは通常空であるため、データの破損はほとんどありません。したがって、これらのメモリビットは無視します。
- コントローラーはデータ格納用に 34 個の LUT RAM を使用します。使用中のメモリ位置でエラーが発生すると、高い確率でコントローラーの動作が停止します。使用するメモリは約 416 ビットです。

$$\text{LUT RAM FIT} = 100\% \times 416 \times 86 \text{ FIT/Mb}$$

$$\text{LUT RAM FIT} = 0.03 \text{ FIT}$$

コントローラーは 3 個のブロック RAM (18Kb) と 9 個のブロック RAM (36Kb) を使用します。内訳は次のとおりです。

- 内部バッファが 1 個のブロック RAM を使用します。データアレイでは、訂正および分類で使用するデータバッファに 9600 ビットが割り当てられます。ここでのソフトエラーが潜在的な問題を引き起こすのは、エラー軽減動作中に発生した場合のみです。ここには恒久的なデータは格納されないため、解析には含めません。これ以外に、定数の格納用として 7480 ビットが割り当てられます。ここでエラーが発生するとコントローラーの動作に支障をきたす可能性が高いため、解析に含める必要があります。残りの 1352 ビットは使用しません。
- コントローラーのファームウェアは 2 個のブロック RAM に格納されます。使用するのは 2048 ワードのうち約 1932 ワードで、そのうち少なくとも 336 ワードはシステム起動時に 1 回だけ実行されるため、解析からは除外します。解析に含めるビット数は 28728 です。
- コントローラー内の拡張修復アルゴリズムは、フレームレベルの CRC を残りのブロック RAM (36Kb) に格納します。これらブロック RAM の内容はブロック RAM 内蔵の ECC によって保護されるため、ブロック RAM ビットの FIT には影響しません。これらのブロック RAM は、先に計算したコンフィギュレーションビットの FIT に影響します。

$$\text{Block RAM FIT} = 100\% \times 36208 \times 78 \text{ FIT/Mb}$$

$$\text{Block RAM FIT} = 2.7 \text{ FIT}$$

したがって、コントローラー全体の FIT は次のとおりです。

$$4.5 \text{ FIT} + 0.03 \text{ FIT} + 2.7 \text{ FIT} \approx 7.2 \text{ FIT}$$

信頼性推定例 (SSI デバイス)

大容量の XC7VX1140T デバイスですべてのオプション機能を有効にした場合、コントローラーとシムは約 1024 個のロジック スライス、80 個の I/O ブロック、13 個のブロック RAM (18Kb)、および 36 個のブロック RAM (36Kb) を使用します。コンフィギュレーションビットの FIT は次の式で求めます。

$$\text{Config FIT} = 10\% \times (1024 \times 1,166 + 80 \times 2,850 + 13 \times 4,698 + 36 \times 9,396) \times 86 \text{ FIT/Mb}$$

$$\text{Config FIT} = 14.9 \text{ FIT}$$

コントローラーとシムがデータに使用するフリップフロップの数は数百個です。これはビット数が少ないため、FIT の計算には含めません。

各コントローラーには 21 個の LUT RAM が含まれます。

- LUT RAM はデータの格納に使用します (スタック、レジスタ、およびスクラッチパッド)。使用中のメモリ位置でエラーが発生すると、高い確率でコントローラーの動作が停止します。使用するメモリは約 416 ビットです。

MON シムには 57 個の LUT RAM が含まれます。

- そのうちの一部の LUT RAM はデータの格納に使用します (スタックおよびレジスタ)。使用中のメモリ位置でエラーが発生すると、高い確率で MON シムの動作が停止します。使用するメモリは約 160 ビットです。
- 一部の LUT RAM はデータバッファとして使用しますが、これらのバッファは通常空であるため、データの破損はほとんどありません。したがって、これらのメモリビットは無視します。

$$\text{LUT RAM FIT} = 100\% \times (4 \times 416 + 1 \times 160) \times 86 \text{ FIT/Mb}$$

$$\text{LUT RAM FIT} = 0.15 \text{ FIT}$$

各コントローラーは 3 個のブロック RAM (18Kb) と 9 個のブロック RAM (36Kb) を使用します。

- 内部バッファが 1 個のブロック RAM を使用します。データアレイでは、訂正および分類で使用するデータバッファに 9,600 ビットが割り当てられます。ここでのソフトエラーが問題を引き起こすのは、エラー軽減動作中に発生した場合のみです。ここには恒久的なデータは格納されないため、エラーを解析には含めません。これ以外に、定数の格納用として 7,480 ビットが割り当てられます。ここでエラーが発生するとコントローラーの動作に支障をきたす可能性が高いため、解析に含める必要があります。残りの 1,352 ビットは使用しません。
- コントローラーのファームウェアは 2 個のブロック RAM に格納されます。使用するのは 2,048 ワードのうち約 1,932 ワードで、そのうち少なくとも 336 ワードはシステム起動時にのみ実行されるため、解析からは除外します。解析に含めるビット数は 28,728 です。
- コントローラー内の拡張修復アルゴリズムは、フレームレベルの CRC を残りのブロック RAM (36Kb) に格納します。これらブロック RAM の内容はブロック RAM 内蔵の ECC によって保護されるため、ブロック RAM ビットの FIT には影響しません。これらのブロック RAM は、先に計算したコンフィギュレーションビットの FIT に影響します。

MON シムには 1 個のブロック RAM (18Kb) が含まれます。

- MON シムのファームウェアが 1 個のブロック RAM を使用します。使用するのは 1,024 ワードのうち約 420 ワードです。解析に含めるビット数は 7,560 です。

$$\text{Block RAM FIT} = 100\% \times (4 \times 36,208 + 1 \times 7,560) \times 78 \text{ FIT/Mb}$$

$$\text{Block RAM FIT} = 11.3 \text{ FIT}$$

したがって、コントローラー全体の FIT は次のとおりです。

$$14.9 \text{ FIT} + 0.15 \text{ FIT} + 11.3 \text{ FIT} \approx 26.4 \text{ FIT}$$

最大周波数

SEM Controller の最大動作周波数は保証されていません。いかなる場合も、使用する FPGA のデータシートでコンフィギュレーション インターフェイスの AC タイミング パラメーター Frbckk として定義されている ICAP (内部コンフィギュレーション アクセス ポート) の F_{Max} を最大動作周波数が上回ることはできません。表 2-3 に、ICAP の F_{Max} の値をまとめます。

表 2-3 : ICAP の最大周波数

デバイス	ICAP F_{Max}
Zynq-7000	100MHz
Zynq-7000A	100MHz
Zynq-7000Q	100MHz
Kintex-7	100MHz
Kintex-7 Low Voltage	70MHz
Kintex-7Q	100MHz
Kintex-7Q Low Voltage	70MHz
Virtex-7 (非 SSI)	100MHz
Virtex-7Q (非 SSI)	100MHz
Virtex-7 (SSI)	70MHz
Artix-7	100MHz
Artix-7 Low Voltage	70MHz
Artix-7Q	100MHz
Artix-7A	100MHz

最大周波数には、その他の制限が適用されることもあります。SEM Controller の最大動作周波数の決定方法の詳細は、第 3 章の「インターフェイス」を参照してください。

ソリューションのレイテンシ

このソリューションのエラー軽減レイテンシは、エラー条件が発生してからエラー軽減プロセスが完了するまでの時間と定義されます。エラー軽減プロセスは、エラー検出、訂正、分類から成ります。

推定データ

このソリューションの動作の基本は、FPGA コンフィギュレーション メモリのフレーム処理にあります。1 ビットエラーは常に 1 つのフレームで発生します。N ビット エラーの発生方法はいくつかあり、すべてのビット エラーが 1 つのフレームで発生することもあれば、N 個のフレームでそれぞれ 1 ビット エラーが発生することもあります。複数のフレームでエラーが発生している場合、検出、訂正、分類のシーケンスをそのフレームの数だけ繰り返します。

このソリューションは任意の数のエラーを適切に軽減します。任意の数のエラーの場合、エラー軽減レイテンシの推定は複雑です。このセクションでは、一般的な例として 1 フレームのみのエラーを取り上げますが、この例を通じて理解したコントローラーの動作はほかのケースにも応用がききます。

スタートアップレイテンシ

スタートアップレイテンシとは、FPGA のコンフィギュレーションが完了してから SEM Controller の初期化が完了するまで (すなわち SEM Controller が監視ステートに移移するまで) の遅延をいいます。このレイテンシは FPGA のサイズ (フレーム数) およびソリューションのクロック周波数によって変わります。また、選択した訂正モードによっても変わります。

スタートアップレイテンシは一度だけ加算されます。これはエラー軽減プロセスの一部ではありません。表 2-4 に示すように、スタートアップレイテンシはブート時間と初期化時間で構成されます。ブート時間はどの訂正モードでも同じですが、初期化時間は選択した訂正モードによって異なります。

表 2-4 : 最大スタートアップレイテンシ (ICAP F_{Max} の場合)

デバイス	ブート時間 (ICAP Fmax)	修復/置換の初期化時間 (ICAP Fmax)	拡張修復の初期化時間 (ICAP Fmax)
XC7A15T	110ms	13.8ms	1.7s
XC7A35T	110ms	13.8ms	1.7s
XC7A50T	110ms	13.8ms	1.7s
XC7A75T	110ms	24.1ms	2.9s
XC7A100T	110ms	24.1ms	2.9s
XC7A200T	110ms	55.4ms	6.6s
XC7K70T	110ms	17.8ms	2.1s
XC7K160T	110ms	38.8ms	4.6s
XC7K325T	110ms	71.0ms	9.3s
XC7K355T	110ms	79.9ms	11.9s
XC7K410T	110ms	91.5ms	15.4s
XC7K420T	110ms	106.6ms	21.1s
XC7K480T	110ms	106.6ms	21.1s
XC7VX330T	110ms	77.3ms	11.1s
XC7VX415T	110ms	98.1ms	17.7s
XC7VX485T	110ms	115.7ms	24.5s
XC7VX550T	110ms	163.5ms	48.7s
XC7VH580T (SSI)	110ms	74.3ms	10.2s
XC7V585T	110ms	124.4ms	28.3s
XC7VX690T	110ms	163.5ms	48.7s
XC7VH870T (SSI)	110ms	74.3ms	10.2s
XC7VX980T	110ms	213.3ms	82.6s
XC7VX1140T (SSI)	110ms	74.3ms	10.2s
XC7V2000T (SSI)	110ms	95.4ms	16.7s
XC7Z010	110ms	12.2ms	1.5s
XC7Z015	110ms	21.7ms	2.6s
XC7Z020	110ms	24.2ms	2.9s
XC7Z030	110ms	34.1ms	4.1s
XC7Z035	110ms	82.0ms	11.8s

表 2-4: 最大スタートアップレイテンシ (ICAP F_{Max} の場合) (続き)

デバイス	ブート時間 (ICAP Fmax)	修復/置換の初期化時間 (ICAP Fmax)	拡張修復の初期化時間 (ICAP Fmax)
XC7Z045	110ms	82.0ms	11.8s
XC7Z100	110ms	103.4ms	19.6s

ブート時間と初期化時間を合計したものがスタートアップレイテンシです。初期化時間は選択した訂正モードにより異なるため、上の表の該当する列の値を選択する必要があります。実際の動作周波数でのスタートアップレイテンシは、表 2-4 のデータと式 2-1 から推定できます。

$$StartUpLatency_{ACTUAL} = StartUpLatency_{ICAP_F_{Max}} \cdot \left[\frac{ICAP_F_{Max}}{Frequency_{ACTUAL}} \right] \quad \text{式 2-1}$$

エラー検出レイテンシ

エラー軽減レイテンシ全体に占める割合が最も大きいのが、エラー検出レイテンシです。このレイテンシは FPGA のサイズ (フレーム数) およびソリューションのクロック周波数によって変わります。また、エラーのタイプ、およびシリコン リードバック プロセスの位置を基準としたエラーの相対位置によっても変わります。表 2-5 に、デバイス全体のスキャン時間を示します。

 表 2-5: 最大デバイス スキャン時間 (ICAP F_{Max} の場合)

デバイス	スキャン時間 (ICAP F _{Max})
XC7A15T	4.6ms
XC7A35T	4.6ms
XC7A50T	4.6ms
XC7A75T	8.0ms
XC7A100T	8.0ms
XC7A200T	18.3ms
XC7K70T	5.9ms
XC7K160T	12.9ms
XC7K325T	23.5ms
XC7K355T	26.5ms
XC7K410T	30.3ms
XC7K420T	35.3ms
XC7K480T	35.3ms
XC7VX330T	25.6ms
XC7VX415T	32.5ms
XC7VX485T	38.3ms
XC7VX550T	54.1ms
XC7VH580T (SSI)	24.6ms
XC7V585T	41.2ms
XC7VX690T	54.1ms
XC7VH870T (SSI)	24.6ms
XC7VX980T	70.7ms

表 2-5 : 最大デバイス スキャン時間 (ICAP F_{Max} の場合) (続き)

デバイス	スキャン時間 (ICAP F_{Max})
XC7VX1140T (SSI)	24.6ms
XC7V2000T (SSI)	31.6ms
XC7Z010	4.0ms
XC7Z015	7.2ms
XC7Z020	8.0ms
XC7Z030	11.3ms
XC7Z035	27.2ms
XC7Z045	27.2ms
XC7Z100	34.3ms

実際の動作周波数でのターゲット デバイスのスキャン時間は、表 2-5 のデータと式 2-2 から推定できます。

$$ScanTime_{ACTUAL} = ScanTime_{ICAP_F_{Max}} \cdot \left[\frac{ICAP_F_{Max}}{Frequency_{ACTUAL}} \right] \quad \text{式 2-2}$$

エラー検出レイテンシには次の制限があります。

- 絶対最小エラー検出レイテンシは実質 0
- ECC による検出の場合、平均エラー検出レイテンシは $0.5 \times ScanTime_{ACTUAL}$
- ECC による検出の場合、最大エラー検出レイテンシは $ScanTime_{ACTUAL}$
- CRC のみによる検出の場合、絶対最大エラー検出レイテンシは $2.0 \times ScanTime_{ACTUAL}$

フレーム ベースの ECC は、1 フレーム内の 1、2、3 ビットおよびすべての奇数ビット エラーを常に検出します。その他のタイプのエラーも、通常はフレーム ベースの ECC で検出できます。ECC では検出できず、CRC でのみ検出可能なエラーが発生することはほとんどありません。

エラー訂正レイテンシ

エラーを検出すると、このソリューションは訂正を試みます。エラーが訂正可能かどうかは、選択した訂正モードとエラータイプによって決まります。表 2-6 と表 2-7 に、コンフィギュレーション フレームが反転した場合のエラー訂正レイテンシを示します。これらの値は、モニター インターフェイスでスロットリングが発生していないことを前提としています。

表 2-6 : 非 SSI : 最大エラー訂正レイテンシ (100MHz、モニター インターフェイスでのスロットリングなし)

訂正モード	フレーム内のエラー数 (訂正可能/不能)	エラー訂正ステート (ICAP F_{Max})
修復	1 ビット (訂正可能)	610 μ s
	2 ビット (訂正不能)	25 μ s
拡張修復	1 ビット (訂正可能)	610 μ s
	2 ビット (訂正可能)	18790 μ s
	2 ビット (訂正不能)	9110 μ s
	BFR ⁽¹⁾ のみ (訂正不能)	10 μ s
置換	任意ビット (訂正可能)	830 μ s
すべて	CRC のみ (訂正不能)	10 μ s

注記 :

- BFR は、ブロック RAM に格納した拡張リペア チェックサムで発生した複数ビットのアップセットによるエラー条件です。

表 2-7 : SSI : 最大エラー訂正レイテンシ (70MHz、モニター インターフェイスでのスロットリングなし)

訂正モード	フレーム内のエラー数 (訂正可能/不能)	エラー訂正ステート (ICAP_F _{Max})
修復	1 ビット (訂正可能)	915µs
	2 ビット (訂正不能)	70µs
拡張修復	1 ビット (訂正可能)	910µs
	2 ビット (訂正可能)	26900µs
	2 ビット (訂正不能)	13010µs
	BFR ⁽¹⁾ のみ (訂正不能)	10µs
置換	任意ビット (訂正可能)	1220µs
すべて	CRC のみ (訂正不能)	10µs

注記 :

1. BFR は、ブロック RAM に格納した拡張リペア チェックサムで発生した複数ビットのアップセットによるエラー条件です。

実際の動作周波数でのエラー訂正レイテンシは、表 2-6 のデータと式 2-3 から推定できます。

$$CorrectionLatency_{ACTUAL} = CorrectionLatency_{ICAP_F_{Max}} \cdot \left[\frac{ICAP_F_{Max}}{Frequency_{ACTUAL}} \right] \quad \text{式 2-3}$$

エラー分類レイテンシ

エラーの訂正を試みた後、このソリューションはエラーを分類します。分類結果は、訂正モード、エラー タイプ、エラー位置、および選択した分類モードによって異なります。表 2-8 と表 2-9 に、コンフィギュレーションフレームが反転した場合のエラー分類レイテンシを示します。これらの値は、モニター インターフェイスでスロットリングが発生していないことを前提としています。

表 2-8 : 非 SSI : 最大エラー分類レイテンシ (100MHz、モニター インターフェイスでのスロットリングなし)

訂正モード	フレーム内のエラー数 (訂正可能/不能)	分類モード	エラー分類ステート (ICAP_F _{Max})
すべて	訂正可能	有効	750µs
すべて	訂正不能	無効	10µs
すべて	訂正不能	すべて	10µs

表 2-9 : SSI : 最大エラー分類レイテンシ (70MHz、モニター インターフェイスでのスロットリングなし)

訂正モード	フレーム内のエラー数 (訂正可能/不能)	分類モード	エラー分類ステート (ICAP_F _{Max})
すべて	訂正可能	有効	1090µs
すべて	訂正不能	無効	10µs
すべて	訂正不能	すべて	10µs

実際の動作周波数でのエラー分類レイテンシは、表 2-8 のデータと式 2-4 から推定できます。

$$ClassificationLatency_{ACTUAL} = ClassificationLatency_{ICAP_F_{Max}} \cdot \left[\frac{ICAP_F_{Max}}{Frequency_{ACTUAL}} \right] \quad \text{式 2-4}$$

その他の要因によるレイテンシ

モニター インターフェイスでのスロットリングは全体的なエラー軽減レイテンシを増大させるため、極力避けてください。

- エラーの訂正を試みた後、コントローラーはモニター インターフェイスから検出/訂正レポートを出力してから、エラー訂正ステートを終了します(このとき、訂正可能ステータスフラグが更新される)。このレポート生成中に MON シムの送信 FIFO がフルになると、コントローラーは MON シムの送信 FIFO にレポート全体の書き込みが完了するまでこのステートを終了できません。この場合、エラー訂正レイテンシが増大します。
- エラーの分類を試みた後、コントローラーはモニター インターフェイスから分類レポートを出力してから、エラー分類ステートを終了します(このとき、エッセンシャル ステータス フラグが更新される)。このレポート生成中に MON シムの送信 FIFO がフルになると、コントローラーは MON シムの送信 FIFO にレポート全体の書き込みが完了するまでこのステートを終了できません。この場合、エラー分類レイテンシが増大します。

このようなボトルネックを完全になくすには、MON シムを削除してモニター インターフェイスの使用をやめるか、またはバッファフル信号を決して送信しないペリフェラルと一緒にモニター インターフェイスを使用するかのどちらかの方法をとります。モニター インターフェイスを使用しない場合も、ステータス インターフェイスで動作を監視できます。

ボトルネックの発生が問題になるペリフェラルの場合、回避策をとることができます。予想される最大バースト長のステータス メッセージを格納できるように送信 FIFO のサイズを調整すると、エラー軽減中に送信 FIFO がフルになることを回避できます。

送信 FIFO がフルになった場合、全体的なエラー軽減レイテンシのおおよその増加分は、式 2-5 で推定できます。

$$\text{AdditionalLatency} = \frac{\text{MessageLength} - \text{BufferDepth}}{\text{TransmissionRate}} \quad \text{式 2-5}$$

式 2-5 で、「MessageLength-BufferDepth」はメッセージ バイト数、「Transmission Rate」は単位時間あたりのバイト数です。

レイテンシの推定例

最初の例は、XC7K325T デバイス (66MHz クロック) にソリューションをインプリメントした場合の、1 ビット エラーの軽減レイテンシを計算で推定します。ソリューションの設定では修復による訂正を選択し、エラー分類は無効にしています。ここでは、モニター インターフェイスのスロットリングは発生しないものと仮定します。

$$\text{DetectionLatency} = 0.5 \cdot \text{ScanTime}_{\text{ACTUAL}} = 0.5 \cdot 23.5\text{ms} \cdot \left[\frac{100\text{MHz}}{66\text{MHz}} \right] = 17.803\text{ms} \quad \text{式 2-6}$$

$$\text{CorrectionLatency} = 610\mu\text{s} \cdot \left[\frac{100\text{MHz}}{66\text{MHz}} \right] = 0.924\text{ms} \quad \text{式 2-7}$$

$$\text{ClassificationLatency} = 10\mu\text{s} \cdot \left[\frac{100\text{MHz}}{66\text{MHz}} \right] = 0.015\text{ms} \quad \text{式 2-8}$$

$$\text{MitigationLatency} = 17.803\text{ms} + 0.924\text{ms} + 0.015\text{ms} = 18.742\text{ms} \quad \text{式 2-9}$$

次の例は、XC7K325T デバイス (66MHz クロック) にソリューションをインプリメントした場合の、2 ビット エラーの軽減レイテンシを計算で推定します。ソリューションの設定では置換による訂正を選択し、エラー分類は有効にしています。ここでも、モニター インターフェイスのスロットリングは発生しないものと仮定します。

$$DetectionLatency = 0.5 \cdot ScanTime_{ACTUAL} = 0.5 \cdot 23.5ms \cdot \left[\frac{100MHz}{66MHz} \right] = 17.803ms \quad \text{式 2-10}$$

$$CorrectionLatency = 830\mu s \cdot \left[\frac{100MHz}{66MHz} \right] = 1.258ms \quad \text{式 2-11}$$

$$ClassificationLatency = 750\mu s \cdot \left[\frac{100MHz}{66MHz} \right] = 1.136ms \quad \text{式 2-12}$$

$$MitigationLatency = 17.803ms + 1.258ms + 1.136ms = 20.197ms \quad \text{式 2-13}$$

最後に、モニター インターフェイスでスロットリングが発生した場合に追加されるレイテンシの推定例を示します。1 番目と 2 番目の例で、メッセージ長を約 80 バイト、MON シムのバッファ深さを 32 バイトと仮定します。また、MON シムのビット レートを 9600 ボーから 460800 ボーへ変更しています。標準 8-N-1 プロトコルを使用しており、1 バイトのペイロードをシリアルリンクで送信するのに 10 ビット時が必要です。

$$AdditionalLatency = \frac{80bytes - 32bytes}{\left[\frac{460800bittimes}{s} \cdot \frac{byte}{10bittimes} \cdot \frac{s}{1000ms} \right]} = 1.042ms \quad \text{式 2-14}$$

この結果が示すように、特にデータ送信がシリアライズされデータ レートが低い場合、モニター インターフェイスのスロットリングによるレイテンシの増加分は非常に大きくなります。

スループット

SEM Controller のスループット メトリクスは仕様で定義されていません。

消費電力

SEM Controller の電力メトリクスは仕様で定義されていません。

リソース使用状況

SEM Controller のリソース使用量のメトリクスは合成後レポートから求めたものであり、事前の見積もり以外の用途には使用しないでください。実際のリソース使用量は異なることがあります。

表 2-10 : Zynq-7000 デバイスのリソース使用量⁽¹⁾⁽²⁾

デバイス	IP コアのコンフィギュレーション	LUT	FF	I/O	ブロック RAM
Zynq-7000 全デバイス	完全ソリューション (オプション機能すべて無効)	509	361	11	3 RAMB18
Zynq-7000 XC7Z010	完全ソリューション (オプション機能すべて有効)	863	681	56	3 RAMB18、2 RAMB36
Zynq-7000 XC7Z015	完全ソリューション (オプション機能すべて有効)	834	682	58	3 RAMB18、3 RAMB36
Zynq-7000 XC7Z020	完全ソリューション (オプション機能すべて有効)	838	682	56	3 RAMB18、3 RAMB36
Zynq-7000 XC7Z030	完全ソリューション (オプション機能すべて有効)	887	683	56	3 RAMB18、5 RAMB36
Zynq-7000 XC7Z035	完全ソリューション (オプション機能すべて有効)	994	696	56	3 RAMB18、10 RAMB36
Zynq-7000 XC7Z045	完全ソリューション (オプション機能すべて有効)	994	696	56	3 RAMB18、10 RAMB36
Zynq-7000 XC7Z100	完全ソリューション (オプション機能すべて有効)	1065	696	56	3 RAMB18、13 RAMB36

注記 :

1. 完全ソリューションとは、SEM Controller とサンプル デザインを組み合わせたものをいいます。これらは併用を想定しています。
2. エラー挿入インターフェイスを I/O に接続します。ロジック デバッグ IP (VIO) を使用すると、LUT/FF 数は増加しますが I/O 数は減少します。

表 2-11 : Kintex-7 デバイスのリソース使用量⁽¹⁾⁽²⁾

デバイス	IP コアのコンフィギュレーション	LUT	FF	I/O	ブロック RAM
Kintex-7 全デバイス	完全ソリューション (オプション機能すべて無効)	511	361	11	3 RAMB18
Kintex-7 XC7K70T	完全ソリューション (オプション機能すべて有効)	849	682	56	3 RAMB18、3 RAMB36
Kintex-7 XC7K160T	完全ソリューション (オプション機能すべて有効)	887	683	56	3 RAMB18、5 RAMB36
Kintex-7 XC7K325T	完全ソリューション (オプション機能すべて有効)	991	686	56	3 RAMB18、9 RAMB36
Kintex-7 XC7K355T	完全ソリューション (オプション機能すべて有効)	994	696	56	3 RAMB18、10 RAMB36
Kintex-7 XC7K410T	完全ソリューション (オプション機能すべて有効)	1028	696	56	3 RAMB18、11 RAMB36
Kintex-7 XC7K420T	完全ソリューション (オプション機能すべて有効)	1065	696	56	3 RAMB18、13 RAMB36

表 2-11 : Kintex-7 デバイスのリソース使用量⁽¹⁾⁽²⁾ (続き)

デバイス	IP コアのコンフィギュレーション	LUT	FF	I/O	ブロック RAM
Kintex-7 XC7K480T	完全ソリューション (オプション機能すべて有効)	1065	696	56	3 RAMB18、13 RAMB36
Kintex-7 Low Voltage、 Kintex-7Q、 Kintex-7Q Low Voltage 全デバイス	Kintex-7 と同じ				

注記 :

1. 完全ソリューションとは、SEM Controller とサンプル デザインを組み合わせたものをいいます。これらは併用を想定しています。
2. エラー挿入インターフェイスを I/O に接続します。ロジック デバッグ IP (VIO) を使用すると、LUT/FF 数は増加しますが I/O 数は減少します。

表 2-12 : Virtex-7 デバイスのリソース使用量 (非 SSI)⁽¹⁾⁽²⁾

デバイス	IP コアのコンフィギュレーション	LUT	FF	I/O	ブロック RAM
Virtex-7 全デバイス	完全ソリューション (オプション機能すべて無効)	510	361	11	3 RAMB18
Virtex-7 XC7VX330T	完全ソリューション (オプション機能すべて有効)	996	696	56	3 RAMB18、10 RAMB36
Virtex-7 XC7VX415T	完全ソリューション (オプション機能すべて有効)	1065	696	56	3 RAMB18、12 RAMB36
Virtex-7 XC7VX485T	完全ソリューション (オプション機能すべて有効)	1101	697	56	3 RAMB18、14 RAMB36
Virtex-7 XC7VX550T	完全ソリューション (オプション機能すべて有効)	1218	691	56	3 RAMB18、20 RAMB36
Virtex-7 XC7V585T	完全ソリューション (オプション機能すべて有効)	1073	698	56	3 RAMB18、15 RAMB36
Virtex-7 XC7VX690T	完全ソリューション (オプション機能すべて有効)	1218	691	56	3 RAMB18、20 RAMB36
Virtex-7 XC7VX980T	完全ソリューション (オプション機能すべて有効)	1366	692	56	3 RAMB18、26 RAMB36
Virtex-7Q 全デバイス	Virtex-7 と同じ				

注記 :

1. 完全ソリューションとは、SEM Controller とサンプル デザインを組み合わせたものをいいます。これらは併用を想定しています。
2. エラー挿入インターフェイスを I/O に接続します。ロジック デバッグ IP (VIO) を使用すると、LUT/FF 数は増加しますが I/O 数は減少します。

表 2-13 : Virtex-7 デバイスのリソース使用量 (SSI)⁽¹⁾⁽²⁾

デバイス	IP コアのコンフィギュレーション	LUT	FF	I/O	ブロック RAM
Virtex-7 XC7VH580T	完全ソリューション (オプション機能すべて無効)	1394	980	19	7 RAMB18
Virtex-7 XC7VH580T	完全ソリューション (オプション機能すべて有効)	2410	1692	64	7 RAMB18、18 RAMB36

表 2-13 : Virtex-7 デバイスのリソース使用量 (SSI)⁽¹⁾⁽²⁾ (続き)

デバイス	IP コアのコンフィギュレーション	LUT	FF	I/O	ブロック RAM
Virtex-7 XC7VH870T	完全ソリューション (オプション機能すべて無効)	1853	1289	27	10 RAMB18
Virtex-7 XC7VH870T	完全ソリューション (オプション機能すべて有効)	3245	2230	72	10 RAMB18、27 RAMB36
Virtex-7 XC7VX1140T	完全ソリューション (オプション機能すべて無効)	2245	1598	35	13 RAMB18
Virtex-7 XC7VX1140T	完全ソリューション (オプション機能すべて有効)	4032	2768	80	13 RAMB18、36 RAMB36
Virtex-7 XC7V2000T	完全ソリューション (オプション機能すべて無効)	2256	1598	35	13 RAMB18
Virtex-7 XC7V2000T	完全ソリューション (オプション機能すべて有効)	4342	2808	80	13 RAMB18、48 RAMB36
Virtex-7Q 全デバイス	Virtex-7 と同じ				

注記 :

1. 完全ソリューションとは、SEM Controller とサンプル デザインを組み合わせたものをいいます。これらは併用を想定しています。
2. エラー挿入インターフェイスを I/O に接続します。ロジック デバッグ IP (VIO) を使用すると、LUT/FF 数は増加しますが I/O 数は減少します。

表 2-14 : Artix-7 デバイスのリソース使用量⁽¹⁾⁽²⁾

デバイス	IP コアのコンフィギュレーション	LUT	FF	I/O	ブロック RAM
Artix-7 全デバイス	完全ソリューション (オプション機能すべて無効)	512	361	11	3 RAMB18
Artix-7 XC7A15T	完全ソリューション (オプション機能すべて有効)	800	681	56	3 RAMB18、2 RAMB36
Artix-7 XC7A35T	完全ソリューション (オプション機能すべて有効)	802	681	56	3 RAMB18、2 RAMB36
Artix-7 XC7A50T	完全ソリューション (オプション機能すべて有効)	800	681	56	3 RAMB18、2 RAMB36
Artix-7 XC7A75T	完全ソリューション (オプション機能すべて有効)	837	682	56	3 RAMB18、3 RAMB36
Artix-7 XC7A100T	完全ソリューション (オプション機能すべて有効)	837	682	56	3 RAMB18、3 RAMB36
Artix-7 XC7A200T	完全ソリューション (オプション機能すべて有効)	919	681	56	3 RAMB18、7 RAMB36
Artix-7 Low Voltage、 Artix-7A、 Artix-7Q 全デバイス	Artix-7 と同じ				

注記 :

1. 完全ソリューションとは、SEM Controller とサンプル デザインを組み合わせたものをいいます。これらは併用を想定しています。
2. エラー挿入インターフェイスを I/O に接続します。ロジック デバッグ IP (VIO) を使用すると、LUT/FF 数は増加しますが I/O 数は減少します。

ポートの説明

SEM Controller は、ソフト エラー軽減ソリューションの中心的な役割を果たします。図 2-1 に SEM Controller のポートを示します。ポートは 6 つのグループにまとめられています。グレーで示したポート グループは、一部のコンフィギュレーションにのみ存在します。

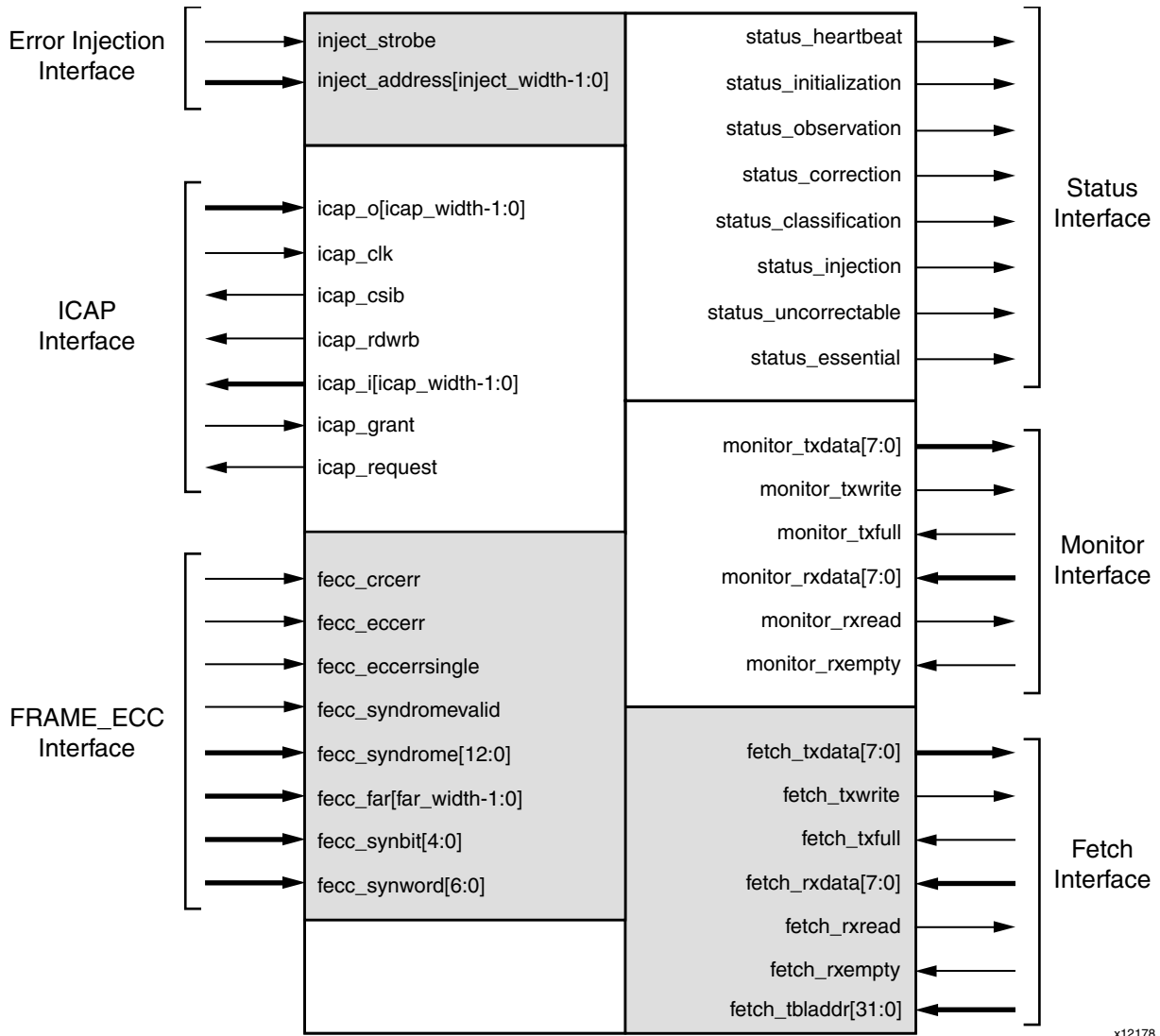


図 2-1 : SEM Controller のポート



ヒント : システム レベル サンプル デザインは SEM Controller と各種シムをカプセル化しており、これらのシムが SEM Controller とほかのデバイスを接続するインターフェイスとしての役割を果たします (図 5-1 参照)。このサンプル デザインはリファレンス デザインではありませんが、ソフト エラー軽減ソリューション全体を構成する必須要素です。図 2-1 に示した SEM Controller のポートは、サンプル デザインに含まれるシムに接続します。付属のサンプル デザインの詳細は、第 5 章「サンプル デザイン」を参照してください。このソリューションのシステム レベルのポートについては、第 5 章の「ポートの説明」を参照してください。

SEM Controller にはリセット入出力はありません。グローバル GSR 信号のディアサートから生成した内部同期リセットによって自動的に自己初期化します。

SEM Controller は、icap_clk を唯一のクロックとして使用する完全同期デザインです。すべてのステート エレメントはこのクロックの立ち上がりエッジに同期します。このため、インターフェイスもすべてこのクロックの立ち上がりエッジに同期します。

ICAP インターフェイス

ICAP インターフェイスは SEM Controller と ICAP プリミティブをポイント ツー ポイントで接続します。ICAP プリミティブにより、FPGA コンフィギュレーション システム内のレジスタへの読み出しおよび書き込みアクセスが可能になります。ICAP プリミティブおよびこのインターフェイスの信号の動作『7 シリーズ FPGA コンフィギュレーション ユーザー ガイド』(UG470) [参照 1] を参照してください。

表 2-15 : ICAP インターフェイスの信号

名前	センス	方向	説明
icap_o[icap_width-1:0]	HIGH	入力	ICAP の O 出力を受信します。変数 icap_width は 32 です。
icap_csib	LOW	出力	ICAP の CSIB 入力を駆動します。
icap_rdwrb	LOW	出力	ICAP の RDWRB 入力を駆動します。
icap_i[icap_width-1:0]	HIGH	出力	ICAP の I 入力を駆動します。変数 icap_width は 32 です。
icap_clk	エッジ	入力	デザインのクロックを受信します。このクロックを ICAP の CLK 入力にも印加する必要があります。このクロック周波数は、ターゲット デバイスのデータシートに記載された ICAP 入力クロックの要件を満たしている必要があります。
icap_request	HIGH	出力	将来のために予約。このポートはオープンのままとします。
icap_grant	HIGH	入力	このポートは VCC に接続します。ユーザーからの ICAP 初期化許可信号を受信します。icap_grant を使用してコントローラーが初期化ステートに移行するのを遅らせることができます。

FRAME_ECC インターフェイス

FRAME_ECC インターフェイスは、SEM Controller と FRAME_ECC プリミティブをポイント ツー ポイントで接続します。FRAME_ECC プリミティブは出力専用のプリミティブで、このプリミティブを通じて FPGA コンフィギュレーション システムのソフト エラー検出機能を監視できます。FRAME_ECC プリミティブおよびこのインターフェイスの信号の動作については、AR# [54350](#) を参照してください。

表 2-16 : FRAME_ECC インターフェイスの信号

名前	センス	方向	説明
fecc_crcerr	HIGH	入力	FRAME_ECC の CRCERROR 出力を受信します。
fecc_eccerr	HIGH	入力	FRAME_ECC の ECCERROR 出力を受信します。
fecc_eccerrsingle	HIGH	入力	FRAME_ECC の ECCERRORSINGLE 出力を受信します。
fecc_syndromevalid	HIGH	入力	FRAME_ECC の SYNDROMEVALID 出力を受信します。
fecc_syndrome[12:0]	HIGH	入力	FRAME_ECC の SYNDROME 出力を受信します。
fecc_far[25:0]	HIGH	入力	FRAME_ECC の FAR 出力を受信します。
fecc_synbit[4:0]	HIGH	入力	FRAME_ECC の SYNBIT 出力を受信します。
fecc_synword[6:0]	HIGH	入力	FRAME_ECC の SYNWORD 出力を受信します。

ステータス インターフェイス

ステータス インターフェイスは、現在のコントローラーの動作を抽象度の高い形式にデコードした各種信号を出力します。

表 2-17: ステータス インターフェイスの信号

名前	センス	方向	説明
status_heartbeat	HIGH	出力	この信号は、status_observation がアサートされている間アクティブです。この信号は、150 クロック サイクルごとに少なくとも 1 回、1 サイクルの High パルスを出力します。この信号を使用して外部ウォッチドッグ タイマーを実装すると、ソフト エラーによるコントローラーまたはクロック分配の動作停止を検出できます。status_observation がディアサートされている場合、このハートビート信号の動作は未定義です。
status_initialization	HIGH	出力	この信号は、デザインが動作を開始した後に 1 回だけ発生するコントローラー初期化の間アクティブです。
status_observation	HIGH	出力	この信号は、コントローラーがエラー検出信号を監視している間アクティブです。エラー検出後、コントローラーがハードウェアに情報を問い合わせている間もこの信号はアクティブのままです。
status_correction	HIGH	出力	この信号は、コントローラーがエラーを訂正している間アクティブです。訂正を無効にした場合も、コントローラーが訂正ステートを遷移中はアクティブになります。
status_classification	HIGH	出力	この信号は、コントローラーがエラーを分類している間アクティブです。分類を無効にした場合も、コントローラーが分類ステートを遷移中はアクティブになります。
status_injection	HIGH	出力	この信号は、コントローラーがエラーを挿入している間アクティブです。エラー挿入が完了し、コントローラーが次のエラーを挿入できる状態または監視ステートに復帰可能な状態になると、この信号は非アクティブに戻ります。
status_essential	HIGH	出力	この信号は、エラー分類ステータス信号です。分類ステート終了前に、コントローラーはエラーがエッセンシャルビットで発生したかどうかをこの信号で示します。その後、コントローラーは分類ステートを終了します。
status_uncorrectable	HIGH	出力	この信号は、エラー訂正ステータス信号です。訂正ステート終了前に、コントローラーはエラーが訂正可能かどうかをこの信号で示します。その後、コントローラーは訂正ステートを終了します。

status_heartbeat 出力は、コントローラーがアクティブであることを示します。コントローラーはソフト エラーを軽減するためのものですが、それ自体もソフト エラーの影響を受ける可能性があります。たとえば、コントローラーのクロックがソフト エラーによって停止する可能性もあります。このような場合、ユーザーは status_heartbeat 信号の停止を検出して対策をとることができます。



ヒント: コントローラー自体がソフト エラーの影響を受けた場合の実行可能な対策の詳細は、[第 3 章の「システム」](#)を参照してください。

status_initialization、status_observation、status_correction、status_classification、status_injection 出力はコントローラーの現在の状態を示します。status_uncorrectable および status_essential 出力は、検出したエラーの性質を示します。

上記の 5 つのコントローラー ステート出力から、さらに 2 つのコントローラー ステートをデコードできます。5 つの信号すべてが **Low** の場合、コントローラーはアイドル ステート (いつでも動作を再開可能な非アクティブ状態) です。5 つの信号すべてが **High** の場合、コントローラーは重大エラー ステートで動作を停止しています。

エラー挿入インターフェイス

エラー挿入インターフェイスには、コンフィギュレーション メモリにビット エラーを挿入するようコントローラーにコマンドを送信するための入力があります。

表 2-18 : エラー挿入インターフェイスの信号

名前	センス	方向	説明
inject_strobe	HIGH	入力	エラー挿入要求を示すエラー挿入制御信号です。inject_strobe 信号を icap_clk に同期して 1 サイクルだけ High にパルスすると同時に、inject_address 入力に有効なアドレスを入力します。この信号は、コントローラーがアイドル ステート以外の場合使用できません。
inject_address[39:0]	HIGH	入力	エラー挿入のパラメーターを指定するエラー挿入アドレスバスです。inject_strobe がアクティブとしてサンプルされると同時にこのバスの値が取り込まれます。

ユーザーは inject_address にエラー挿入のアドレスとコマンドを入力し、inject_strobe をアサートしてエラー挿入要求を示します。

この要求にตอบสนองして、コントローラーは 1 ビット エラーを挿入します。コントローラーは、status_injection をアサートしてエラー挿入コマンドを受信したことを通知します。エラー挿入コマンドの実行が完了すると、コントローラーは status_injection をデアサートします。複数ビットに影響するエラーを挿入する場合は、上記のエラー挿入を複数回実行します。

エラー挿入コマンドの詳細は、第 3 章の「エラー挿入インターフェイス」を参照してください。

モニター インターフェイス

モニター インターフェイスは、ユーザーとコントローラーが双方向に情報をやりとりする手段として使用します。

コントローラーがモニター インターフェイスから読み出すコマンド、およびモニター インターフェイスに書き込むステータス情報はいずれも ASCII 文字列です。モニター インターフェイスは、ステータス インターフェイスおよびエラー挿入インターフェイスのステータスおよびコマンド機能を含みます。モニター インターフェイスは、プロセスベースシステムでの使用を想定しています。

表 2-19 : モニター インターフェイスの信号

名前	センス	方向	説明
monitor_txdata[7:0]	HIGH	出力	コントローラーからのパラレル送信データ。
monitor_txwrite	HIGH	出力	パラレル送信データが有効であることを示す書き込みストロープ。
monitor_txfull	HIGH	入力	この信号で、シム (ペリフェラル) からコントローラーに対する送信チャンネルのフロー制御を実装します。
monitor_rxdata[7:0]	HIGH	入力	シム (ペリフェラル) からのパラレル受信データ
monitor_rxread	HIGH	出力	パラレル受信データの受信完了を通知する読み出しストロープ。
monitor_rxempty	HIGH	入力	この信号で、シム (ペリフェラル) からコントローラーに対する受信チャンネルのフロー制御を実装します。

モニター インターフェイスは、システム レベルのサンプル デザインでは MON シムに接続します。MON シムは、コマンドおよびステータスの交換用に RS-232 プロトコル互換の全二重シリアル ポートを実装しています。MON シムの機能の詳細は、第 3 章の「モニター インターフェイス」を参照してください。

フェッチ インターフェイス

フェッチ インターフェイスは、コントローラーが外部ソースからデータを要求する手段として使用します。

エラー訂正およびエラー分類を実行中、コントローラーは 1 フレーム分のコンフィギュレーション データまたはエッセンシャル ビット データをフェッチする必要があります。コントローラーは、必要なデータを記述したコマンドをバイナリ形式でフェッチ インターフェイスに書き込むよう設計されています。外部ソースはこの情報を使用してデータをフェッチし、フェッチ インターフェイスに返す必要があります。

表 2-20: フェッチ インターフェイスの信号

名前	センス	方向	説明
fetch_txdata[7:0]	HIGH	出力	コントローラーからのパラレル送信データ。
fetch_txwrite	HIGH	出力	パラレル送信データが有効であることを示す書き込みストロープ。
fetch_txfull	HIGH	入力	この信号で、シム (ペリフェラル) からコントローラーに対する送信チャンネルのフロー制御を実装します。
fetch_rxdata[7:0]	HIGH	入力	シム (ペリフェラル) からのパラレル受信データ
fetch_rxread	HIGH	出力	パラレル受信データの受信完了を通知する読み出しストロープ。
fetch_rxempty	HIGH	入力	この信号で、シム (ペリフェラル) からコントローラーに対する受信チャンネルのフロー制御を実装します。
fetch_tbladdr[31:0]	HIGH	入力	この信号で、外部ソースにあるコントローラー データ テーブルの開始アドレスを指定します。

コアを使用するデザイン

この章では、ボトムアップアプローチを使用して3つの異なるレベルでコアをアプリケーションに統合する方法について説明します。この章には次のセクションがあります。

- 「**インターフェイス**」では、ソリューションへの接続方法について説明します。
- 「**ビヘイビア**」では、インターフェイスを介してソリューションと通信する方法について説明します。
- 「**システム**」では、ソリューションを大規模なシステムへ統合する方法について説明します。

インターフェイス

システムレベル サンプル デザインには、生成時に選択したオプションに応じて4～6個の外部インターフェイスがあります。このセクションでは、各インターフェイスとそれぞれの接続方法について説明します。

クロック インターフェイス

ここでは、入力クロックに関する推奨事項について説明します。これらの推奨事項は、FPGA データシートに記載された FPGA コンフィギュレーション システムに印加されるクロック信号の要件に基づいて決定しています。

- デューティ サイクル: 45% 以上、55% 以下

入力クロックの周波数が高いほど、ソリューションのエラー軽減レイテンシは小さくなります。したがって、周波数はなるべく高い方が理想です。入力クロックの最大周波数を決定する際には、いくつかの重要な要因を考慮する必要があります。

- FPGA コンフィギュレーション システムの最大クロック周波数を超えないこと。具体的な数値は、ターゲット デバイスのデータシートを参照してください。
- スタティック タイミング解析ツールで報告された最大クロック周波数を超えないこと。通常、この条件が制限となることはありません。

システムレベル サンプル デザインを使用する場合、完全同期設計手法に基づき、クロック周波数を選択する際には外部インターフェイスのタイミングとの関係も考慮する必要があります。

- EXT シムとメモリ インターフェイス:
 - SPI バスの最大クロック周波数を決定するには、SPI バス タイミング バジレットの評価が必要です。解析例は [33 ページの「外部インターフェイス」](#) を参照してください。
 - SPI バス クロックは入力クロックの 1/2 です。したがって、入力クロックは SPI バスの最大クロック周波数の 2 倍以下とする必要があります。

- MON シムとシリアル インターフェイス：
 - 入力クロックとシリアル インターフェイスのボーレートは、16 の整数倍の関係です。標準ボーレートが必要な場合、非常に高いボーレートまたは非常に低い入力クロック周波数では解空間が限られることがあります。
 - 解析例は [32 ページ](#) の「[モニター インターフェイス](#)」を参照してください。

これらの要因を考慮して、すべての条件を満たす入力クロック周波数を選択します。

ステータス インターフェイス

ステータス インターフェイスからは、ロジック信号に基づくイベント レポートが直接出力されます。ステータス インターフェイスは多くの用途に利用できますが、その使用は必須ではありません。このインターフェイスで報告される情報は、次の 3 つの種類に分類されます。

- ステート：コントローラーの現在の動作を示します。
- フラグ：検出したエラーのタイプを特定します。
- ハートビート：スキャンが動作中であることを示します。

システム レベル サンプル デザインをスタックド シリコン インターコネクト (SSI) デバイスにインプリメントした場合、各 SLR (Ruper Logic Region) に 1 つずつコントローラー インスタンスがあるため、ステータス インターフェイスも SLR ごとに個別に存在します。ほとんどの場合、ステータス インターフェイスの信号は必要なものだけを FPGA の I/O ピンに接続します。システム レベル サンプル デザインでは、すべての信号を I/O ピンに接続しています。

ステータス信号は外部でインジケータに接続して表示することも、別のデバイスに接続して観測することもできます。別のデバイスに接続してイベント報告を正しく取り込むには、ステータス インターフェイスのスイッチ特性を考慮する必要があります。

特に SSI デバイスへのインプリメンテーションでは信号の数が多くなるため、ステータス インターフェイスは複雑になりがちです。ただしステータス インターフェイスでしか取得できないのはハートビート イベントのみです。それ以外の情報はモニター インターフェイスでも得られます。

ステータス インターフェイスの信号は、システム レベル サンプル デザインに供給されるクロックを使用してコントローラー内部の順次ロジック処理で生成されます。このため、パルス幅は常にクロック サイクルの整数倍です。

[図 3-1](#) に、ステート信号 `status_initialization`、`status_observation`、`status_correction`、`status_classification`、および `status_injection` の全体的なスイッチ特性を示します。この図で、`status_[state]` 信号は 5 つのステート信号をグループとして表したもので、これによってコントローラーのステートが示されます。

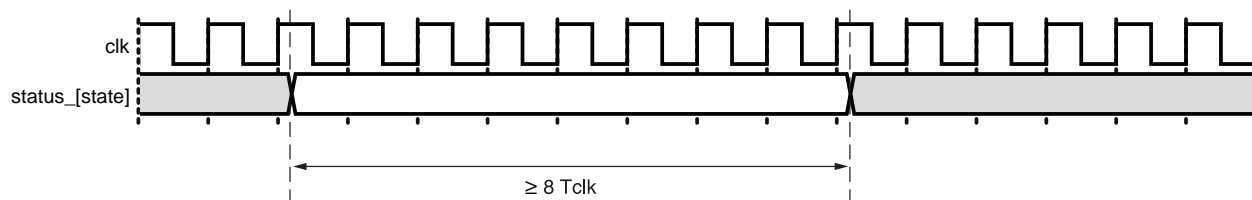


図 3-1: ステータス インターフェイスのステート信号のスイッチ特性

フラグ信号 `status_uncorrectable` および `status_essential` はステート終了時に更新されます。図 3-2 と図 3-3 に、ステート終了時点に基づいたスイッチ特性を示します。これらの図は、コントローラーがステートを終了する時点に基づいてフラグが有効な期間を示しており、この期間内にフラグを更新できます。この波形には特定のフラグ値は示していません。

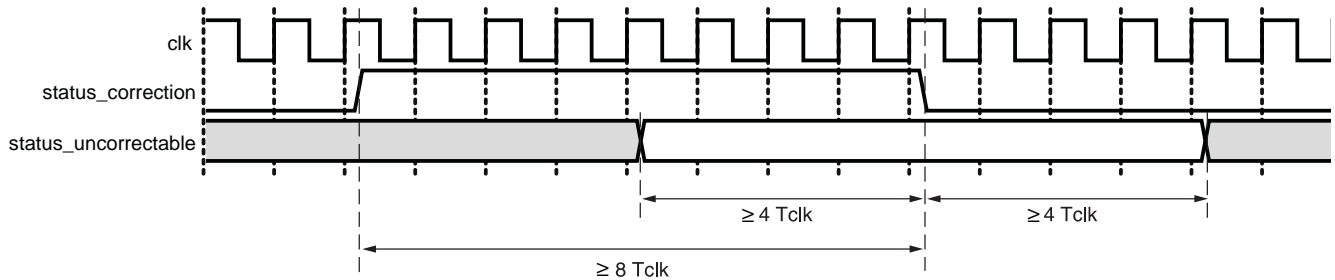


図 3-2 : ステータス インターフェイスの uncorrectable フラグのスイッチ特性

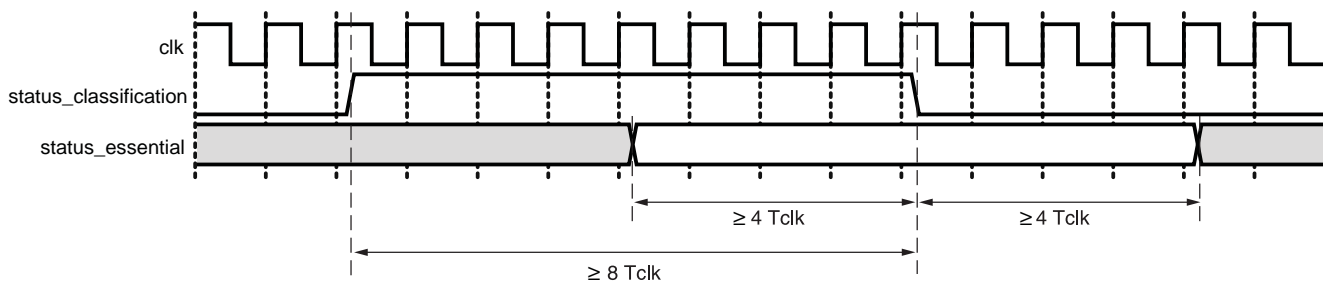


図 3-3 : ステータス インターフェイスの essential フラグのスイッチ特性

ハートビート信号 `status_heartbeat` のスイッチ特性を図 3-4 に示します。この信号はリードバック プロセスから直接出力され、監視ステートの間アクティブです。監視ステートに遷移し、リードバック プロセスがエラーをスキャンしていると、このハートビート信号がアクティブになります。監視ステート中に検出した最初のハートビートパルスを使用して、ハートビートの停止を監視する回路を起動する必要があります。監視ステート以外では、ハートビート信号の動作は定義されていません。

監視ステートに遷移した後の最初のハートビート信号は、リードバック スキャンを 3 回実行するまでの期間に発生するはずですが、リードバック スキャン時間は、使用するデバイスおよびクロック周波数により異なります。各デバイスのリードバック スキャン時間は、表 2-5 を参照してください。

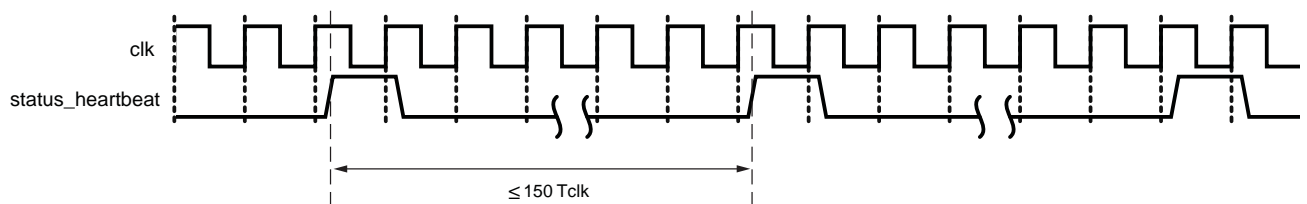


図 3-4 : ステータス インターフェイスのハートビートのスイッチ特性

このパルス幅は非常に小さいため、エンベデッド プロセッサの GPIO を利用してソフトウェア ポーリングでステータス インターフェイス信号をサンプリングする方法は正しく動作しないことが考えられます。この方法ではなく、カウンタ/タイマー入力、エッジセンス割り込み入力、またはイベント キャプチャ機能を備えた入力などを使用してください。

モニター インターフェイス

モニター インターフェイスは2つの信号で構成され、これらを使用してコマンドおよびステータス情報を交換するためのRS-232プロトコル互換の全二重シリアルポートを実装します。次の設定を使用します。

- ボー：9600
- 設定：8-N-1
- フロー制御：なし
- 端末設定：VT100
 - TX 改行：CR (改行コードとしてCR [0x0D] を端末から送信)
 - RX 改行：CR+LF (改行コードとしてCR [0x0D] を端末が受信し、CR+LF [0x0D, 0x0A] に展開)
 - ローカルエコー：なし

モニター インターフェイスに接続する外部デバイスは、この設定をサポートしている必要があります。図 3-5 に、送信および受信のスイッチ特性を示します。



図 3-5：モニター インターフェイスのスイッチ特性

送信および受信タイミングは、システム レベル サンプル デザイン 内部でカウンターを使用して生成される 16x ビットレート イネーブル信号から求めます。このカウンターの動作は 0 からカウントを開始し、ターミナル カウント (カウンターを同期的にリセットするために検出および使用する条件) までカウントします。ターミナル カウント出力は、送信および受信プロセスにもタイム ベースとして供給されます。

9600 ボーは標準ビット レートであり、さまざまな入力クロック周波数から生成できるため、互換性には優れています。このため、システム レベル サンプル デザイン でもこのボー レートを使用しています。

しかし実際には、ビット レートが低いためデータ レートとレイテンシの両面で通信パフォーマンスが低いという問題があります。これはコントローラーでスロットリングが発生する原因となります。このため、より高いビット レートに変更することを強く推奨します。115200、230400、460800、921600 ボーなどの標準ビット レートを含め、さまざまな設定が可能です。

システム レベル サンプル デザイン の MON シム モジュールでは、パラメーター V_ENABLETIME で通信ビット レートを設定します。V_ENABLETIME の値は次の式で求めます。

$$V_ENABLETIME = \text{round to integer} \left[\frac{\text{input clock frequency}}{16 \times \text{nominal bitrate}} \right] - 1 \quad \text{式 3-1}$$

上の式で V_ENABLETIME を求めると、最大で ±0.5 の丸め誤差が発生します。この誤差により、公称ビット レートとは若干異なるビット レートが生成されます。RS-232 デバイス間での許容誤差が 2% とすると、デバイスあたりのビット レートの許容誤差は ±1% までとなります。

例：入力クロックが 66MHz、目的のビット レートが 115200 ボーの場合

$$V_ENABLETIME = \text{round to integer} \left[\frac{66000000}{16 \times 115200} \right] - 1 = 35 \quad \text{式 3-2}$$

実際に得られるビット レートは約 114583 ボーで、公称ビット レートの 115200 ボーに対する誤差は -0.54% です。これは、誤差が ±1% 以内であるため許容範囲内です。

公称ビット レートに対する誤差が ±1% を超える場合、誤差が小さくなるようにビット レートと入力クロック周波数の組み合わせを変えてください。それ以外のスイッチ特性は定義されていません。

電氣的に、モニター インターフェイスが使用する I/O ピンは LVCMOS 信号で、ほかのデバイスとの接続に適しています。特別な I/O モードは必要ありません。RS-232 との完全な電氣的互換性が必要な場合は、外部レベル変換器を使用する必要があります。



ヒント : モニター インターフェイスに接続する MON シムのカスタマイズの詳細は、「[MON シムのカスタマイズ](#)」を参照してください。

外部インターフェイス

外部インターフェイスは 4 つの信号で構成され、これらを使用して SPI バス プロトコル互換の全二重シリアルポートを実装します。このインターフェイスは、次に示すコントローラー オプションの少なくとも 1 つを有効にした場合のみ存在します。

- 置換によるエラー訂正
- エラー分類

これら機能を実装するには、外部ストレージが必要です。システム レベル サンプル デザインでは、SPI バス マスターに機能が固定された EXT シムを使用して外部 SPI フラッシュ デバイスからデータをフェッチします。表 3-1 に、サポートされる各 FPGA で必要な SPI フラッシュの容量を示します。

表 3-1 : 外部ストレージの要件

デバイス	エラー分類のみ	置換によるエラー訂正のみ	エラー分類と置換によるエラー訂正
XC7A15T	16Mb	16Mb	32Mb
XC7A35T	16Mb	16Mb	32Mb
XC7A50T	16Mb	16Mb	32Mb
XC7A75T	32Mb	32Mb	64Mb
XC7A100T	32Mb	32Mb	64Mb
XC7A200T	64Mb	64Mb	128Mb
XC7K70T	32Mb	32Mb	64Mb
XC7K160T	64Mb	64Mb	128Mb
XC7K325T	128Mb	128Mb	256Mb
XC7K355T	128Mb	128Mb	256Mb
XC7K410T	128Mb	128Mb	256Mb
XC7K420T	128Mb	128Mb	256Mb
XC7K480T	128Mb	128Mb	256Mb
XC7VX330T	128Mb	128Mb	256Mb
XC7VX415T	128Mb	128Mb	256Mb
XC7VX485T	128Mb	128Mb	256Mb
XC7VX550T	256Mb	256Mb	512Mb
XC7VH580T (SSI)	256Mb	256Mb	512Mb
XC7V585T	128Mb	128Mb	256Mb
XC7VX690T	256Mb	256Mb	512Mb
XC7VH870T (SSI)	256Mb	256Mb	512Mb
XC7VX980T	256Mb	256Mb	512Mb
XC7VX1140T (SSI)	512Mb	512Mb	1024Mb
XC7V2000T (SSI)	512Mb	512Mb	1024Mb

表 3-1：外部ストレージの要件 (続き)

デバイス	エラー分類のみ	置換によるエラー訂正のみ	エラー分類と置換によるエラー訂正
XC7Z010	16Mb	16Mb	32Mb
XC7Z015	32Mb	32Mb	64Mb
XC7Z020	32Mb	32Mb	64Mb
XC7Z030	64Mb	64Mb	128Mb
XC7Z035	128Mb	128Mb	256Mb
XC7Z045	128Mb	128Mb	256Mb
XC7Z100	128Mb	128Mb	256Mb

EXT シムは高速読み出しコマンド (0x0b) を使用します。また、何種類かある SPI フラッシュ ファミリの 1 つをサポートするように設定できます。デフォルトでサポートされるファミリは、表 3-1 に示した外部ストレージの要件によって異なります。システム レベル サンプル デザインの EXT シム モジュールには、SPI フラッシュ デバイスに送信するコマンド シーケンスを制御するためのパラメーターが 3 つあります。

- **B_ISSUE_WREN**
 - デバイスの動作を変更するコマンドの前に書き込みイネーブル コマンド (0x06) を発行する必要があるかどうかを示します。
 - 通常は「0」に設定し、N25Q デバイスの場合のみ「1」に設定します。
- **B_ISSUE_WVCR**
 - 高速読み出しダミー サイクル数を明示的に 8 サイクルに設定するために揮発性コンフィギュレーション レジスタへの書き込みコマンド (0x81) を発行する必要があるかどうかを示します。EXT シム内部のステートマシンはバイト指向のため、高速読み出しダミー サイクル数を 8 に設定する必要があります。揮発性コンフィギュレーション レジスタのデータは上書きされます (0x8b)。
 - 通常は「0」に設定し、N25Q デバイスの場合のみ「1」に設定します。
- **B_ISSUE_EN4B**
 - 4 バイト アドレス指定モードに明示的に移行するために 4 バイト アドレス指定イネーブル コマンド (0xb7) を発行する必要があるかどうかを示します。
 - 128Mb を超えるデバイスでは「1」に設定します。

ストレージ要件が 128Mb 以下の場合、EXT シムはデフォルトで M25P デバイスをサポートします (B_ISSUE_WREN = 0、B_ISSUE_WVCR = 0、B_ISSUE_EN4B = 0)。これらのデバイスは 4 バイト アドレス指定モードには対応しません。

ストレージ要件が 128Mb を超える場合、EXT シムはデフォルトでより大容量の M25Q デバイスをサポートします (B_ISSUE_WREN = 1、B_ISSUE_WVCR = 1、B_ISSUE_EN4B = 1)。これらのデバイスは 4 バイト アドレス指定モードに対応します。

これ以外に、ストレージ要件が 128Mb 以下の場合には小容量の N25Q デバイス (B_ISSUE_WREN = 1、B_ISSUE_WVCR = 1、B_ISSUE_EN4B = 0)、ストレージ要件が 128Mb を超える場合は大容量の MX25 デバイス (B_ISSUE_WREN = 0、B_ISSUE_WVCR = 0、B_ISSUE_EN4B = 1) もサポートされます。

図 3-6 に、FPGA と SPI フラッシュ デバイスの接続を示します。「LT」はレベル変換器を表しています。一般的な SPI フラッシュ デバイスは 3.3V I/O を使用しますが、FPGA または I/O バンク電圧によってはこの電圧を利用できないことがあるため、レベル変換器が必要です。

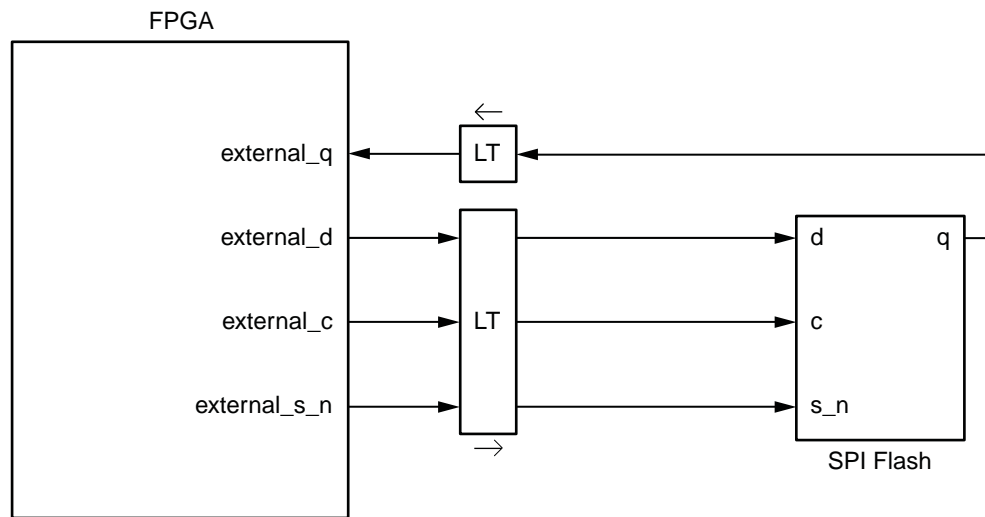


図 3-6：SPI フラッシュ デバイスの接続 (レベル変換器を含む)

SPI バスの性能を最大化するには、伝搬遅延の小さいレベル変換器を使用する必要があります。SPI バスの性能がシステム レベル サンプル デザイン 全体の最大動作周波数に影響することがあります。

この後のセクションでは、SPI バス タイミング バジレットの解析方法について説明します。これは非常に重要な解析であり、SPI バスでのデータ転送の信頼性を確保するには必ず実行する必要があります。この例に示すタイミング バジレットが必ず満たされるよう、インプリメンテーションごとに十分な評価を実行してください。

SPI バス クロックの波形およびタイミング バジレット

SPI フラッシュ デバイスには、入力クロックのスイッチ特性に関する要件があります。ここでは、システム レベル サンプル デザインによって生成される SPI フラッシュ デバイス用のクロック信号について解析します。この解析を実行するには、ボード レベルのシグナル インテグリティ シミュレーション機能が必要です。

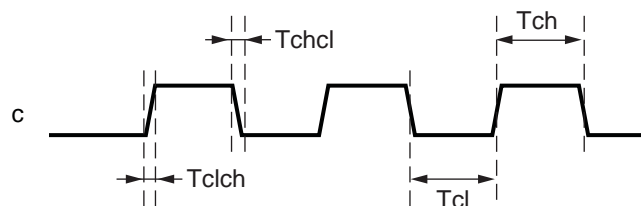


図 3-7：SPI フラッシュ デバイスの入力クロック要件

図 3-7 に示したように、SPI フラッシュ デバイスの入力クロックの要件として次のパラメーターが定義されています。

- T_{clch} = SPI バス クロックの最大立ち上がり時間
- T_{chl} = SPI バス クロックの最大立ち下がり時間
- T_{cl} = SPI バス クロックの最小 Low 時間
- T_{ch} = SPI バス クロックの最小 High 時間

SPI バスの物理的な構成、FPGA の I/O 特性、およびレベル変換器を使用する場合はその I/O 特性により、FPGA から送信される SPI バス クロック信号の立ち上がり時間 (T_{rise}) と立ち下がり時間 (T_{fall}) は、SPI フラッシュ デバイスに到達した時点で最大となります。 T_{rise} と T_{fall} がそれぞれ T_{clch} と T_{chcl} の要件を満たしていることを検証する必要があります。 T_{clch} と T_{chcl} の要件が満たされていない場合、次のような対策をとることができます。

- システム レベル サンプル デザインの SPI バス クロック出力に対する I/O スルー レートを変更する。
- システム レベル サンプル デザインの SPI バス クロック出力に対する I/O 駆動能力を変更する。
- I/O 特性の適したレベル変換器に交換する。

一般に、 T_{clch} と T_{chcl} の要件は容易に満たすことができます。これらの要件が存在するのは、システム レベル サンプル デザインで使用しているポイント ツー ポイント方式ではなく、多くの負荷を接続したバスで立ち上がり時間と立ち下がり時間が極端に長くなるのを防ぐためです。

システム レベル サンプル デザインによって生成される SPI バス クロックは、入力クロックを 2 分周したものです。したがって、SPI バス クロックの High および Low 時間は名目上は T_{clk} と同じです。ただし実際の T_{rise} および T_{fall} を考慮して、次の要件も満たす必要があります。

- $T_{clk} \geq T_{rise} + T_{ch}$
- $T_{clk} \geq T_{fall} + T_{cl}$

例：

- $T_{clch} = 33\text{ns}$ (SPI フラッシュのデータシートより)
- $T_{chcl} = 33\text{ns}$ (SPI フラッシュのデータシートより)
- $T_{cl} = 9\text{ns}$ (SPI フラッシュのデータシートより)
- $T_{ch} = 9\text{ns}$ (SPI フラッシュのデータシートより)
- $T_{rise} = 2\text{ns}$ (PCB シミュレーションより)
- $T_{fall} = 2\text{ns}$ (PCB シミュレーションより)

これらのデータより、次の解析を実行します。

1. チェック： $T_{clch} \geq T_{rise}$ であるか。すなわち $33\text{ns} \geq 2\text{ns}$ であるか。Yes
2. チェック： $T_{chcl} \geq T_{fall}$ であるか。すなわち $33\text{ns} \geq 2\text{ns}$ であるか。Yes
3. 計算： $T_{clk} \geq T_{rise} + T_{ch}$ より、 $T_{clk} \geq 2\text{ns} + 9\text{ns}$ 、すなわち $T_{clk} \geq 11\text{ns}$
4. 計算： $T_{clk} \geq T_{fall} + T_{cl}$ より、 $T_{clk} \geq 2\text{ns} + 9\text{ns}$ 、すなわち $T_{clk} \geq 11\text{ns}$

立ち上がり時間の要件は満たされています。 T_{clk} に対するこれらの要件により、SPI バス クロックの波形およびタイミング バジレットではシステム レベル サンプル デザインの入力クロック サイクル時間が 11ns 以上に制限されます。

SPI バス送信の波形およびタイミング バジレット

SPI フラッシュ デバイスには、入力クロックを基準とした入力データのスイッチ特性に関する要件があります。ここでは、システム レベル サンプル デザインから受信したデータを SPI フラッシュ デバイスでキャプチャする場合について解析します。

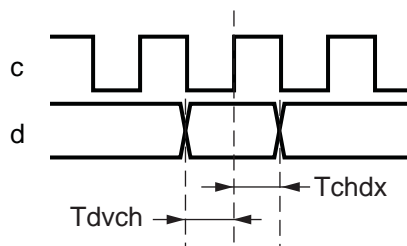


図 3-8：SPI フラッシュ デバイスの入力データ キャプチャ要件

図 3-8 に示したように、SPI フラッシュ デバイスがデータを正しく取り込むための要件として次のパラメーターが定義されています。

- T_{dvch} = クロックを基準とした SPI フラッシュの最小データ セットアップ時間
- T_{chdx} = クロックを基準とした SPI フラッシュの最小データ ホールド時間

この解析は、最小伝搬遅延を 0 と仮定しています。また、次に示すスキューが無視できるものと仮定しています。

- FPGA 出力フリップフロップへの入力クロック分配のスキュー。
- FPGA 出力フリップフロップから FPGA ピンへの出力信号パスのスキュー。
- PCB レベル変換器のチャンネル遅延のスキュー。この条件を満たすには、クロックとデータパスのレベル変換器遅延が一致していなければなりません。
- PCB トレース セグメント遅延のスキュー。この条件を満たすには、クロックとデータパスのトレース遅延が一致していなければなりません。
- デューティ サイクルの歪み。

EXT シムと PCB のインプリメンテーション パラメーターとして次のパラメーターが定義されています。

- T_{clk} = 入力クロック サイクル時間 (icap_clk)
- T_{qfpga} = icap_clk を基準とした FPGA 出力遅延
- T_{w1} = FPGA からレベル変換器までの PCB トレース遅延
- T_{w2} = レベル変換器から SPI フラッシュまでの PCB トレース遅延
- T_{dly} = レベル変換器のチャンネル遅延

図 3-9 に、EXT シム インプリメンテーションによって生成されるメモリ システムの信号を示します。

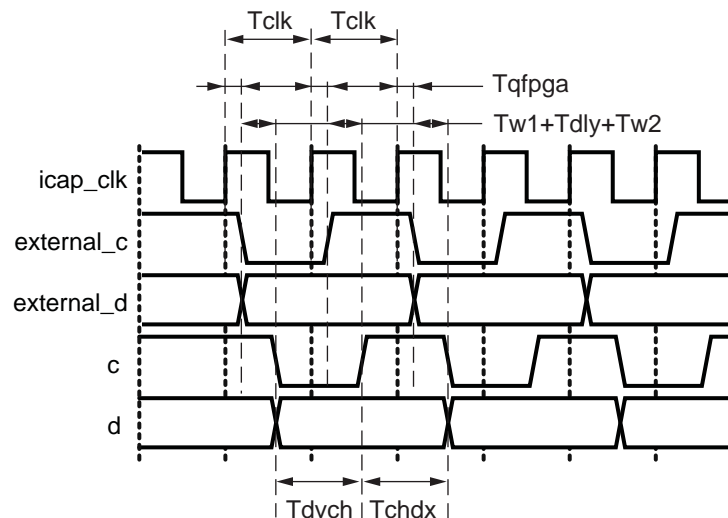


図 3-9: 入力データ キャプチャのタイミング

前述の仮定により、クロックとデータパスの遅延はどちらも同じで、PVT (プロセス、電圧、温度) ばらつきの影響も同じです。次の関係が成り立ちます。

- $T_{clk} \geq T_{dvch}$
- $T_{clk} \geq T_{chdx}$

例：

- $T_{dvch} = 2\text{ns}$ (SPI フラッシュのデータシートより)
 - $T_{chdx} = 5\text{ns}$ (SPI フラッシュのデータシートより)
1. 計算： $T_{clk} \geq T_{dvch}$ 、すなわち $T_{clk} \geq 2\text{ns}$
 2. 計算： $T_{clk} \geq T_{chdx}$ 、すなわち $T_{clk} \geq 5\text{ns}$

T_{clk} に対するこれらの要件により、SPI バス送信の波形およびタイミング バジレットではシステム レベル サンプル デザインの入力クロック サイクル時間が 5ns 以上に制限されます。

SPI バス受信の波形およびタイミング バジレット

SPI フラッシュ デバイスは、入力クロックを基準とした出力データの出力スイッチ特性が決まっています。ここでは、SPI フラッシュ デバイスから受信したデータをシステム レベル サンプル デザインでキャプチャする場合について解析します。

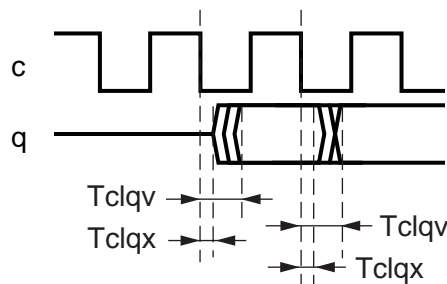


図 3-10：SPI フラッシュ デバイスの出力データ スイッチ特性

図 3-10 に示したように、SPI フラッシュ デバイスの出力スイッチ動作として次のパラメーターが定義されています。

- T_{clkq} = クロックを基準とした SPI フラッシュの最大出力 Valid 時間
- T_{clkx} = クロックを基準とした SPI フラッシュの最小出力ホールド時間

この解析は、最小伝搬遅延を 0 と仮定しています。また、次に示すスキューが無視できるものと仮定しています。

- FPGA 出力および入力フリップフロップへの入力クロック分配のスキュー。
- PCB レベル変換器のチャネル遅延のスキュー。この条件を満たすには、クロックとデータパスのレベル変換器の遅延が一致していなければなりません。
- デューティ サイクルの歪み。

EXT シムと PCB のインプリメンテーション パラメーターとして次のパラメーターが定義されています。

- T_{clk} = 入力クロック サイクル時間 (icap_clk)
- T_{qfpga} = icap_clk を基準とした FPGA 出力遅延
- T_{sfpga} = icap_clk を基準とした FPGA 入力セットアップ時間
- T_{hfpga} = icap_clk を基準とした FPGA 入力ホールド時間
- T_{w1} = FPGA からレベル変換器までの PCB トレース遅延
- T_{w2} = レベル変換器から SPI フラッシュまでの PCB トレース遅延
- T_{w3} = SPI フラッシュからレベル変換器までの PCB トレース遅延
- T_{w4} = レベル変換器から FPGA までの PCB トレース遅延
- T_{dly} = レベル変換器のチャネル遅延

このタイミングパスはEXTシムに対しては2サイクルパスですが、SPIフラッシュデバイスに対しては1サイクルパスです。タイミング解析では、SPIフラッシュデバイスのClock-to-Outを組み合わせ遅延としてモデル化します。FPGA側でのセットアップとホールドの両方の要件を考慮する必要があります。

図3-11と図3-12に、EXTシムインプリメンテーションによって生成されるメモリシステムの信号を示します。

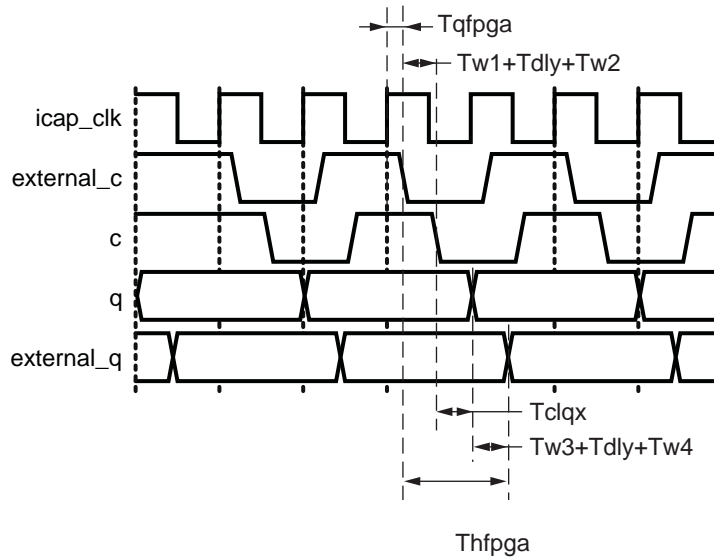


図3-11：出力データキャプチャタイミング(ホールド解析)

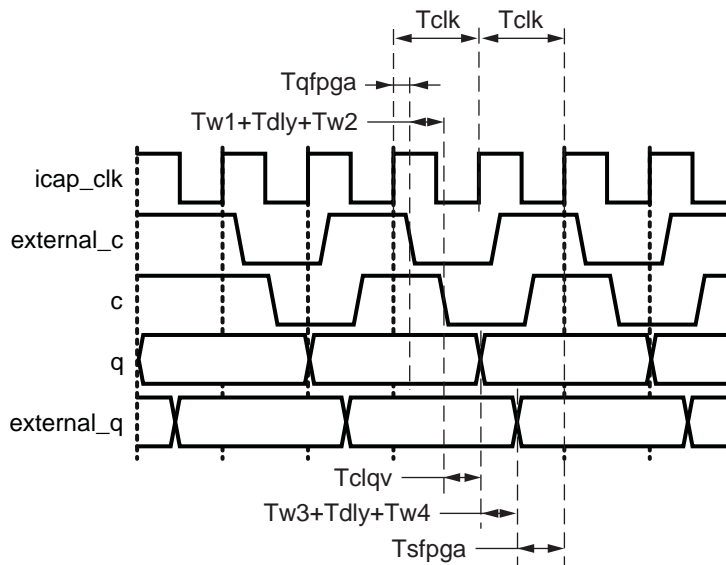


図3-12：出力データキャプチャタイミング(セットアップ解析)

ホールドパス解析は合否判定です。ホールドパス解析は最小値の遅延を使用して計算し、次の関係が成り立つかを確認する必要があります。

$$T_{hfpga} \leq T_{qfpga,min} + T_{w1} + T_{dly} + T_{w2} + T_{clkx} + T_{w3} + T_{dly} + T_{w4}$$

遅延 T_{w1} 、 T_{w2} 、 T_{w3} 、 T_{w4} 、および T_{dly} に安全な最小値として0を代入すると、次式が得られます。

$$T_{hfpga} \leq T_{qfpga,min} + T_{clkx}$$

セットアップパス解析は最大値の遅延を使用して計算する必要があります。

$$T_{clk} \geq 0.5 \times (T_{qfpga,max} + T_{w1} + T_{dly} + T_{w2} + T_{clqv} + T_{w3} + T_{dly} + T_{w4} + T_{sfpga})$$

例 : Vivado Design Suite、Kintex-7 FPGA

- $T_{clqv} = 8\text{ns}$ (SPI フラッシュのデータシートより)
- $T_{clqx} = 0\text{ns}$ (SPI フラッシュのデータシートより)
- $T_{dly} = 3\text{ns}$ (レベル変換器のデータシートより)
- $T_{w1} = 1\text{ns}$ (ボード シミュレーションより)
- $T_{w2} = 1\text{ns}$ (ボード シミュレーションより)
- $T_{w3} = 1\text{ns}$ (ボード シミュレーションより)
- $T_{w4} = 1\text{ns}$ (ボード シミュレーションより)

FPGA タイミング パラメーターは、アプリケーションで使用するターゲット FPGA にインプリメントしたシステム レベル サンプル デザインでのタイミング レポートから取得する必要があります。必要なレポートを生成するには、「report_timing_summary」に「min_max」オプションを付けて実行します。

次に示す例は、Kintex-7 デバイスにインプリメントしたシステム レベル サンプル デザインで生成したタイミング レポートの抜粋です。これは、タイミング レポートから必要な情報を得るための例として示しています。レポート内に必要な情報が見つからない場合は、レポートに含めるパスの最大数を増やしてください。

タイミング レポートの「Destination: external_c」と表示された場所で、「Max at Slow Process Corner」にあるフリップフロップからパッドまでのパス解析を参照し、 T_{qfpga} の値を確認します。

- $T_{qfpga} = \text{I/O データパス遅延 (external_c)}$
- $T_{qfpga} = 3.211\text{ns}$ 、最大

```
Slack (MET) :                20.670ns
  Source:                    example_ext/example_ext_byte/ext_c_ofd/C
                             (rising edge-triggered cell FDRE clocked by clk {rise@0.000ns
fall@7.576ns period=15.151ns})
  Destination:               external_c
                             (output port clocked by clk {rise@0.000ns fall@7.576ns
period=15.151ns})
  Path Group:                 clk
  Path Type:                  Max at Slow Process Corner
  Requirement:                15.151ns
  Data Path Delay:            3.211ns (logic 3.211ns (100.000%) route 0.000ns (0.000%))
  Logic Levels:               1 (OBUF=1)
  Output Delay:               -15.151ns
  Clock Path Skew:           -6.385ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):  0.000ns
    Source Clock Delay (SCD):        6.385ns
    Clock Pessimism Removal (CPR):   0.000ns
  Clock Uncertainty:          0.035ns ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
    Total System Jitter (TSJ):       0.071ns
    Total Input Jitter (TIJ):        0.000ns
    Discrete Jitter (DJ):            0.000ns
    Phase Error (PE):                0.000ns
```

Location	Delay type	Incr(ns)	Path(ns)	Netlist Resource(s)
R24	(clock clk rise edge)	0.000	0.000	r
R24	net (fo=0)	0.000	0.000	r clk
R24	IBUF (Prop_ibuf_I_O)	1.176	1.176	r example_ibuf/I
R24	net (fo=1, routed)	3.130	4.305	r example_ibuf/O
BUFGCTRL_X0Y0				r example_bufg/I
BUFGCTRL_X0Y0	BUFG (Prop_bufg_I_O)	0.093	4.398	r example_bufg/O
example_ext_byte/icap_clk	net (fo=477, routed)	1.987	6.385	r example_ext/
OLOGIC_X0Y37				r example_ext/
example_ext_byte/ext_c_ofd/C				
OLOGIC_X0Y37	FDRE (Prop_fdre_C_Q)	0.366	6.751	r example_ext/
example_ext_byte/ext_c_ofd/Q	net (fo=1, routed)	0.000	6.751	n_96_example_ext
AB20				r external_c_OBUF_inst/I
AB20	OBUF (Prop_obuf_I_O)	2.845	9.596	r external_c_OBUF_inst/O
AB20	net (fo=0)	0.000	9.596	external_c
AB20				r external_c
	(clock clk rise edge)	15.151	15.151	r
	clock pessimism	0.000	15.151	
	clock uncertainty	-0.035	15.116	
	output delay	15.151	30.267	
	required time		30.267	
	arrival time		-9.596	
	slack		20.670	

タイミングレポートの「Destination: external_c」と表示された場所で、「Min at Fast Process Corner」にあるフリップフロップからパッドまでのパス解析を参照し、 T_{qfpga} の値を確認します。

- T_{qfpga} = I/O データパス遅延 (external_c)
- T_{qfpga} = 1.379ns、最小

```
Slack (MET) :          4.460ns
Source:          example_ext/example_ext_byte/ext_c_ofd/C
                 (rising edge-triggered cell FDRE clocked by clk {rise@0.000ns
fall@7.576ns period=15.151ns})
Destination:    external_c
                 (output port clocked by clk {rise@0.000ns fall@7.576ns
period=15.151ns})
Path Group:     clk
Path Type:      Min at Fast Process Corner
Requirement:    0.000ns
Data Path Delay: 1.379ns (logic 1.379ns (100.000%) route 0.000ns (0.000%))
Logic Levels:   1 (OBUF=1)
Output Delay:   0.000ns
Clock Path Skew: -3.117ns (DCD - SCD - CPR)
Destination Clock Delay (DCD): 0.000ns
```

```

Source Clock Delay      (SCD):    3.117ns
Clock Pessimism Removal (CPR):    -0.000ns
Clock Uncertainty:     0.035ns  ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
Total System Jitter    (TSJ):    0.071ns
Total Input Jitter     (TIJ):    0.000ns
Discrete Jitter        (DJ):     0.000ns
Phase Error            (PE):     0.000ns
    
```

Location	Delay type	Incr(ns)	Path(ns)	Netlist Resource(s)

	(clock clk rise edge)	0.000	0.000	r
R24		0.000	0.000	r clk
	net (fo=0)	0.000	0.000	clk
R24				r example_ibuf/I
R24	IBUF (Prop_ibuf_I_O)	0.616	0.616	r example_ibuf/O
	net (fo=1, routed)	1.675	2.291	clk_ibufg
BUFGCTRL_X0Y0				r example_bufg/I
BUFGCTRL_X0Y0	BUFG (Prop_bufg_I_O)	0.026	2.317	r example_bufg/O
	net (fo=477, routed)	0.800	3.117	example_ext/
example_ext_byte/icap_clk				
OLOGIC_X0Y37				r example_ext/
example_ext_byte/ext_c_ofd/C				

OLOGIC_X0Y37	FDRE (Prop_fdre_C_Q)	0.192	3.309	r example_ext/
example_ext_byte/ext_c_ofd/Q				
	net (fo=1, routed)	0.000	3.309	n_96_example_ext
AB20				r external_c_OBUF_inst/I
AB20	OBUF (Prop_obuf_I_O)	1.187	4.495	r external_c_OBUF_inst/O
	net (fo=0)	0.000	4.495	external_c
AB20				r external_c

	(clock clk rise edge)	0.000	0.000	r
	clock pessimism	0.000	0.000	
	clock uncertainty	0.035	0.035	
	output delay	-0.000	0.035	

	required time		-0.035	
	arrival time		4.495	

	slack		4.460	

タイミングレポートの「Source: external_q」と表示された場所で、「Max at Slow Process Corner」にあるパッドからリップフロップまでのパス解析を参照し、 T_{sfpga} の値を確認します。

- T_{sfpga} = I/O データパス遅延 (external_q)
- T_{sfpga} = 7.813ns、最大

```

Slack (MET) :          28.041ns
Source:          external_q
                 (input port clocked by clk {rise@0.000ns fall@7.576ns
period=15.151ns})
Destination:     example_ext/example_ext_byte/ext_q_ifd/D
                 (rising edge-triggered cell FDRE clocked by clk {rise@0.000ns
fall@7.576ns period=15.151ns})
Path Group:      clk
    
```

```

Path Type:                Max at Slow Process Corner
Requirement:              15.151ns
Data Path Delay:          7.813ns (logic 7.813ns (100.000%) route 0.000ns (0.000%))
Logic Levels:             2 (IBUF=1 ZHOLD_DELAY=1)
Input Delay:              -15.151ns
Clock Path Skew:          5.588ns (DCD - SCD + CPR)
  Destination Clock Delay (DCD):    5.588ns
  Source Clock Delay (SCD):          0.000ns
  Clock Pessimism Removal (CPR):     0.000ns
Clock Uncertainty:        0.035ns ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
  Total System Jitter (TSJ):         0.071ns
  Total Input Jitter (TIJ):          0.000ns
  Discrete Jitter (DJ):              0.000ns
  Phase Error (PE):                  0.000ns

```

Location	Delay type	Incr(ns)	Path(ns)	Netlist Resource(s)

	(clock clk rise edge)	0.000	0.000	r
	input delay	-15.151	-15.151	
AD21		0.000	-15.151	r external_q
	net (fo=0)	0.000	-15.151	external_q
AD21				r external_q_IBUF_inst/I
AD21	IBUF (Prop_ibuf_I_O)	1.161	-13.990	r external_q_IBUF_inst/O
	net (fo=1, routed)	0.000	-13.990	example_ext/
example_ext_byte/external_q_IBUF				
	ILOGIC_X0Y30			r example_ext/
example_ext_byte/ext_q_ifd_OPT_INSERTED/DLYIN				
	ILOGIC_X0Y30 ZHOLD_DELAY (Prop_zhold_delay_DLYIN_DLYIFF)	6.797	-7.193	r example_ext/
example_ext_byte/ext_q_ifd_OPT_INSERTED/DLYIFF				
	net (fo=1, routed)	0.000	-7.193	example_ext/
example_ext_byte/OPT_ZHD_N_ext_q_ifd				
	ILOGIC_X0Y30			r example_ext/
example_ext_byte/ext_q_ifd/D				
	ILOGIC_X0Y30 FDRE (Setup_fdre_C_D)	-0.145	-7.338	example_ext/
example_ext_byte/ext_q_ifd				

	(clock clk rise edge)	15.151	15.151	r
R24		0.000	15.151	r clk
	net (fo=0)	0.000	15.151	clk
R24				r example_ibuf/I
R24	IBUF (Prop_ibuf_I_O)	1.113	16.264	r example_ibuf/O
	net (fo=1, routed)	2.604	18.868	clk_ibufg
BUFGCTRL_X0Y0				r example_bufg/I
BUFGCTRL_X0Y0	BUFG (Prop_bufg_I_O)	0.083	18.951	r example_bufg/O
	net (fo=477, routed)	1.788	20.739	example_ext/
example_ext_byte/icap_clk				
	ILOGIC_X0Y30			r example_ext/
example_ext_byte/ext_q_ifd/C				
	clock pessimism	0.000	20.739	
	clock uncertainty	-0.035	20.703	

	required time		20.703	
	arrival time		7.338	

	slack		28.041	

タイミングレポートの「Source: external_q」と表示された場所で、「Min at Fast Process Corner」にあるパッドからフリップフロップまでのパス解析を参照し、 T_{hfga} の値を確認します。

- T_{hfga} = I/O データパス遅延 (external_q)
- T_{hfga} = -3.386ns、最小

```
Slack (MET) :                29.797ns
  Source:                    external_q
                          (input port clocked by clk {rise@0.000ns fall@7.576ns
period=15.151ns})
  Destination:              example_ext/example_ext_byte/ext_q_ifd/D
                          (rising edge-triggered cell FDRE clocked by clk {rise@0.000ns
fall@7.576ns period=15.151ns})
  Path Group:                clk
  Path Type:                 Min at Fast Process Corner
  Requirement:              0.000ns
  Data Path Delay:          3.386ns (logic 3.386ns (100.000%) route 0.000ns (0.000%))
  Logic Levels:             2 (IBUF=1 ZHOLD_DELAY=1)
  Input Delay:              30.302ns
  Clock Path Skew:          3.856ns (DCD - SCD - CPR)
    Destination Clock Delay (DCD):  3.856ns
    Source Clock Delay (SCD):        0.000ns
    Clock Pessimism Removal (CPR):  -0.000ns
  Clock Uncertainty:        0.035ns ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
    Total System Jitter (TSJ):       0.071ns
    Total Input Jitter (TIJ):        0.000ns
    Discrete Jitter (DJ):            0.000ns
    Phase Error (PE):                0.000ns
```

Location	Delay type	Incr(ns)	Path(ns)	Netlist Resource(s)
	(clock clk rise edge)	0.000	0.000	r
	input delay	30.302	30.302	
AD21		0.000	30.302	r external_q
	net (fo=0)	0.000	30.302	external_q
AD21				r external_q_IBUF_inst/I
AD21	IBUF (Prop_ibuf_I_O)	0.601	30.903	r external_q_IBUF_inst/O
	net (fo=1, routed)	0.000	30.903	example_ext/
example_ext_byte/external_q_IBUF				r example_ext/
ILOGIC_X0Y30				
example_ext_byte/ext_q_ifd_OPT_INSERTED/DLYIN				
ILOGIC_X0Y30	ZHOLD_DELAY (Prop_zhold_delay_DLYIN_DLYIFF)	2.939	33.842	r example_ext/
example_ext_byte/ext_q_ifd_OPT_INSERTED/DLYIFF				
	net (fo=1, routed)	0.000	33.842	example_ext/
example_ext_byte/OPT_ZHD_N_ext_q_ifd				
ILOGIC_X0Y30				r example_ext/
example_ext_byte/ext_q_ifd/D				
ILOGIC_X0Y30	FDRE (Hold_fdre_C_D)	-0.154	33.688	example_ext/
example_ext_byte/ext_q_ifd				

	(clock clk rise edge)	0.000	0.000	r	
R24		0.000	0.000	r	clk
	net (fo=0)	0.000	0.000		clk
R24				r	example_ibuf/I
R24	IBUF (Prop_ibuf_I_O)	0.782	0.782	r	example_ibuf/O
	net (fo=1, routed)	1.984	2.765		clk_ibufg
BUFGCTRL_X0Y0				r	example_bufg/I
BUFGCTRL_X0Y0	BUFG (Prop_bufg_I_O)	0.030	2.795	r	example_bufg/O
	net (fo=477, routed)	1.061	3.856		example_ext/
example_ext_byte/icap_clk					
ILOGIC_X0Y30				r	example_ext/
example_ext_byte/ext_q_ifd/C					
	clock pessimism	0.000	3.856		
	clock uncertainty	0.035	3.892		

	required time				-3.892
	arrival time				33.688

	slack				29.797

チェック :

- $T_{hfpga} \leq T_{qfpga,min} + T_{clq}$ であるか。
- すなわち $-3.386ns \leq 1.379ns + 0ns$ であるか。
- すなわち $-3.386ns \leq 1.379ns$ であるか。 Yes

計算 :

$$T_{clk} \geq 0.5 \times (T_{qfpga,max} + T_{w1} + T_{dly} + T_{w2} + T_{clqv} + T_{w3} + T_{dly} + T_{w4} + T_{sfpga})$$

すなわち

$$T_{clk} \geq 0.5 \times (3.211ns + 1ns + 3ns + 1ns + 8ns + 1ns + 3ns + 1ns + 7.813ns)$$

すなわち

$$T_{clk} \geq 14.512ns$$

ホールド要件は満たされています。また、Tclk に対する要件により、SPI バス受信の波形およびタイミング バジレットではシステム レベル サンプル デザインの入力クロック サイクル時間が 14.512ns 以上に制限されます。

SPI バス タイミング バジレットのまとめ

EXT シムと外部メモリ システムが存在する場合、インプリメンテーションの堅牢性を確保するには SPI バス タイミング バジレットを解析する必要があります。この解析結果により、外部メモリ システムが正しく機能することが確認され、システム レベル サンプル デザインの入力クロックの最大周波数に対する制約がわかります。

例のまとめ

Kintex-7 にインプリメントした SEM IP コアの Vivado Design Suite タイミング レポートで報告されたサンプルデータにより、メモリ インターフェイスが正しく機能することが確認されます。メモリ インターフェイスは入力クロック周波数が 68.908MHz 以下でないと動作しないため、Tclk に対する最も厳しい要件は $T_{clk} \geq 14.512ns$ です。ICAP (内部コンフィギュレーション アクセス ポート) の最大クロック周波数やシステム レベル サンプル デザインの最大クロック周波数など、ほかの入力クロック周波数の制限も考慮する必要があります。

エラー挿入インターフェイス

エラー挿入インターフェイスは入力バスと入力ストロブで構成され、シンプルなパラレル入力ポートを実装しています。このインターフェイスは、エラー挿入を有効にして I/O ピンを選択した場合のみ存在します。エラー挿入インターフェイスからはロジック信号に基づくエラーを直接挿入できます。このインターフェイスの使用は完全に任意です。このインターフェイスは次に示す 5 種類のコマンドを受け付けます。

- ・ アイドル ステートに移行するためのコマンド
- ・ 監視ステートに移行するためのコマンド
- ・ ソフトウェア リセット (再起動と初期化) を実行するためのコマンド
- ・ 物理フレーム アドレスを指定してエラーを挿入するためのコマンド
- ・ リニア フレーム アドレスを指定してエラーを挿入するためのコマンド



ヒント : エラー挿入インターフェイス コマンドの詳細は、「[エラー挿入インターフェイスのコマンド](#)」を参照してください。

システム レベル サンプル デザインを SSI デバイスにインプリメントした場合、各 SLR に 1 つずつコントローラー インスタンスがあり、すべてのコントローラー インスタンスがエラー挿入インターフェイスからの信号を受信します。多くの場合、エラー挿入インターフェイスからの信号は FPGA の I/O ピンに接続できます。

システム レベル サンプル デザインのコンフィギュレーションによっては、すべての信号が I/O ピンに接続されます。I/O ピンでなく Vivado Design Suite のデバッグ機能インターフェイスを選択してコンフィギュレーションすると、エラー挿入信号を FPGA 内部で生成して駆動できます。

エラー挿入信号を外部で別のデバイスに接続して制御することもできます。入力されたコマンドを正しく取り込むために、別のデバイスに接続する際はエラー挿入インターフェイスのタイミング要件を考慮する必要があります。

エラー挿入インターフェイスの信号は、入力レジスタを有効にするストロブを使用してコントローラー内部の順次ロジック処理により受信します。

マスクしたフレーム、あるいはデバイスまたは SEU でカバーされないアドレス空間にあるフレームにエラーを挿入した場合、エラー挿入コマンドは無視され、監視ステートでエラーは検出されません。

Vivado Design Suite のデバッグ機能の使用

同期入力ポートで受信したステータス信号は仮想 LED で示されます。inject_address 出力値はデータ入力用に HEX として示します。正しく動作させるには、inject_strobe 制御出力を同期 1 サイクルパルス出力として示す必要があります。VIO コアの使用方法の詳細は、『Vivado Design Suite ユーザー ガイド : プログラムおよびデバッグ』(UG908) [参照 4] を参照してください。

ビヘイビアー

システム レベル サンプル デザインは、動作中にコントローラー デザインに基づいていくつかの高次機能ビヘイビアーを示します。このセクションでは、予想されるビヘイビアーおよびシステム レベル サンプル デザインとの通信方法について説明します。システム レベル サンプル デザインを SSI にインプリメントした場合の複数コントローラー インスタンスのビヘイビアーは、非 SSI にインプリメントした場合の単一コントローラーのビヘイビアーと同じです。

コントローラーの動作

初期化

コントローラーは FPGA のグローバル セット/リセット信号によって非アクティブな状態に置かれます。コンフィギュレーションが完了すると、FPGA コンフィギュレーション システムがグローバル セット/リセット信号をディassertし、コントローラーがブートします。ブート プロセスの間、コントローラーはステータス インターフェイスの5つのステート ビットをすべてディassertしたまま維持します。

コントローラーはブート中に `icap_grant` 入力をポーリングし、初期化ステートに移行して ICAP の使用を開始できるかどうかを確認します。ほとんどのデバイスへのインプリメンテーションでは、`icap_grant` を High に接続します。ただし Zynq-7000 デバイスへのインプリメンテーションでは、`icap_grant` 信号に特別な処理が必要です。

Zynq-7000 のプロセッシング システム (PS) のブート中、PS は PCAP (プロセッサ コンフィギュレーション アクセスポート) を通じてデバイス内のコンフィギュレーション ロジックにアクセスできます。このパスを利用して、PS ブートローダーはビットストリームを Zynq-7000 のプログラマブル ロジック (PL) にダウンロードします。PS ブートローダーが完了すると、PS による PL のパーシャル リコンフィギュレーションをサポートするために PS と PCAP がコンフィギュレーション ロジックを常時制御します。

ただし、PS と PCAP がコンフィギュレーション ロジックを制御している間、PL と ICAP はコンフィギュレーション ロジックにロックしません。コントローラーが正しく機能するには、ICAP 経由でコンフィギュレーション ロジックにアクセスする必要があります。そのためには、PS デバイス コンフィギュレーション制御レジスタ (DEVCFG CTRL、アドレス 0xF8007000) の PCAP_PR (ビット 27) をクリアします。PCAP の詳細は、『Zynq-7000 All Programmable SoC テクニカル リファレンス マニュアル』(UG585) [参照 3] を参照してください。図 3-13 に、PCAP、ICAP、PCAP_PR の相互接続を示します。

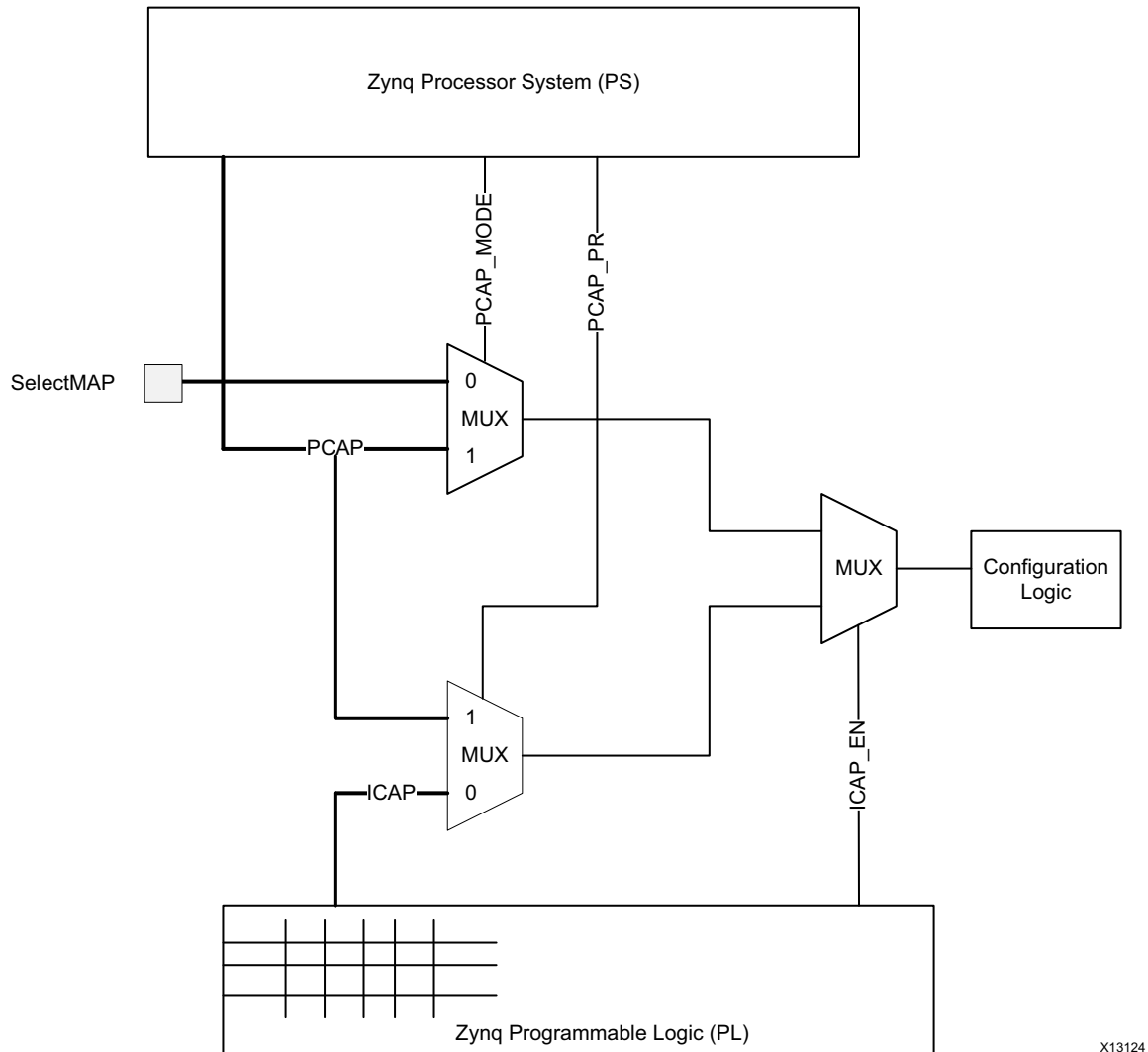


図 3-13 : Zynq-7000 のコンフィギュレーション ロジックへのアクセス

ICAP がロックしていないことをコントローラーが簡単に知る方法はありません。初期化ステートの間、コントローラーはコンフィギュレーション ロジックの **IDCODE** レジスタを読み出すために **ICAP** をポーリングします。これは、期待されるベンダー ID 値が検出されるまで続けられます。ただし、コントローラーが **ICAP** へのアクセスを試みている間に **PS** が **PCAP_PR** をクリアした場合、コンフィギュレーション ロジックは不正な形式の **ICAP** トランザクションを受信することがあります。この場合、コンフィギュレーション ロジックの動作は予測不能となります。

このような可能性を取り除くには、**PS** が **GPIO** 経由でコントローラーの **icap_grant** 入力を駆動し、**PCAP_PR** がクリアされるまでコントローラーが初期化ステートに移行しないようにする必要があります。使用する **GPIO** は **PS** からの **EMIO** の場合と **PL** の **GPIO** の場合がありますが、どちらの場合も **PL** コンフィギュレーションが完了してすぐに **icap_grant** がディアサートされるのを検出できるように初期化しておく必要があります。

PS 上で動作するソフトウェアは、必要な **PCAP** アクティビティをすべて完了すると **PCAP_PR** をクリアした後、コントローラーの **icap_grant** 入りに接続された **GPIO** をセットします。これによって、コントローラーは初期化ステートに進むことができます。**icap_grant** 入りに印加される信号は、**icap_clk** 信号に正しく同期している必要があります。この動作をソフトウェアでインプリメントする方法は、ここでは取り上げません。ベアメタルおよび Linux 環境におけるソフトウェア開発の詳細は、『Zynq-7000 All Programmable SoC ソフトウェア開発者向けガイド』(UG821) および『OS およびライブラリ資料コレクション』(UG643) を参照してください。

初期化ステートの間、`status_initialization` は **TRUE** です。初期化ステートでは、モニター インターフェイスから出力されるステータス レポートなど直接観測可能なイベント以外に、内部のハウスキーパー処理もいくつか実行されます。具体的には、次の処理が実行されます。

- 1 回目のリードバック サイクルでフレーム レベルの **ECC** チェックサムを計算
- 2 回目のリードバック サイクルでデバイス レベルの **CRC** チェックサムを計算
- 追加のリードバック サイクルで追加のチェックサムを計算
- 追加のリードバック サイクルでフレーム レベルの **CRC** チェックサムを計算し、ブロック **RAM** に格納 (拡張修復による訂正を使用する場合のみ)

初期化が完了すると、コントローラーは監視ステートへ遷移します。

監視

コントローラーはほとんどすべての時間を監視ステートで費やします。監視ステートの間、`status_observation` は **TRUE** で、コントローラーは **FPGA** コンフィギュレーション システムを監視してエラー条件を検出します。エラーが存在していない場合、エラー挿入インターフェイスまたはモニター インターフェイスからコマンドを受信すると、コントローラーはそのコマンドを処理します。監視ステートでサポートされるコマンドは `enter idle` と `status report` の 2 つのみです。コントローラーはそれ以外のコマンドをすべて無視します。

`enter idle` コマンドはコントローラーをアイドル状態にしてほかのコマンドを実行できるようにするためのもので、エラー挿入インターフェイスまたはモニター インターフェイスから入力できます。このコマンドを実行するとコントローラーはアイドル ステートへ遷移します。

`status report` は診断情報を取得するためのコマンドで、コントローラーに対する **ping** のような働きをします。このコマンドはモニター インターフェイスからのみ入力できます。

エラーが検出された場合、コントローラーは訂正を試みるために必要な情報をハードウェアから読み出します。情報の収集が完了すると、コントローラーは訂正ステートへ遷移します。

訂正

訂正ステートでは、コントローラーはエラーの訂正を試みます。訂正機能を有効にしていない場合も、コントローラーは必ず訂正ステートを通過します。訂正ステートの間、`status_correction` は **TRUE** です。

エラーが **CRC** のみのエラーの場合、コントローラーは `status_uncorrectable` をセットしてモニター インターフェイスからレポートを出力した後、分類ステートに遷移します。エラーが **CRC** のみのエラーでない場合、コントローラーの動作はエラー訂正方法の設定によって異なります。

置換による訂正を選択した場合、コントローラーは置換データの要求をフェッチ インターフェイスに出力します。システム レベル サンプル デザインでは、この要求を **EXT** シムが外部メモリの読み出しに変換します。外部からのデータは、**EXT** シムを経由してコントローラーへ返されます。次に、コントローラーはアクティブ パーシャル リコンフィギュレーションを実行してフレームを正しい内容で書き換えます。コントローラーは `status_uncorrectable` をクリアしてモニター インターフェイスからレポートを出力した後、分類ステートに遷移します。

修復による訂正または拡張修復による訂正を選択した場合は、アルゴリズムを用いてエラーの訂正を試みます。エラーが訂正可能な場合、コントローラーはアクティブ パーシャル リコンフィギュレーションを実行してフレームを訂正後の内容で書き換え、`status_uncorrectable` をクリアします。エラーが訂正不能な場合、コントローラーは `status_uncorrectable` をセットします。どちらの場合も、コントローラーはレポートを出力した後、分類ステートに遷移します。

分類

分類ステートでは、コントローラーはエラー进行分类します。分類機能を有効にしていない場合も、コントローラーは必ず分類ステートを通過します。分類ステートの間、`status_classification` は TRUE です。

訂正ステート中に訂正不能と報告されたエラーはすべてエッセンシャルとして報告されます。エラーを訂正できないのは、エラーの場所を特定できないのが唯一の理由です。また、訂正不能なエラーの場合はエッセンシャルかどうかをルックアップテーブルで判定できません。この場合、コントローラーは `status_essential` をセットし、レポートを出力した後、アイドルステートに遷移します。訂正不能エラーが1つ検出されると、コントローラーはその時点でエラーの検出を中止します。この場合、FPGA のリコンフィギュレーションが必要です。

訂正ステートで訂正可能と報告されたエラーの取り扱いは、コントローラーのオプション設定により異なります。エラー分類が無効な場合、訂正可能エラーはすべて無条件でエッセンシャルとして報告されます。エラー分類が有効な場合、コントローラーは分類データの要求をフェッチ インターフェイスに出力します。システム レベル サンプル デザインでは、この要求を EXT シムが外部メモリの読み出しに変換します。外部からのデータは、EXT シムを経由してコントローラーへ返されます。このデータに基づき、コントローラーはエラーがエッセンシャルかどうかを判定します。判定が完了するとコントローラーはレポートを出力し、`status_essential` を適切に変更した後、監視ステートに遷移してエラーの監視を続けます。

アイドル

アイドルステートでは、コントローラーは FPGA コンフィギュレーション システムのエラー条件を監視しませんが、それ以外は監視ステートと同じです。アイドルステートでは、ステータス インターフェイスの5つのステートビットがすべてディASSERTされます。エラー挿入インターフェイスまたはモニター インターフェイスからコマンドを受信すると、コントローラーはそのコマンドを処理します。エラー挿入およびソフトウェア リセット コマンドはアイドルステートでのみサポートされます。

`enter observation` コマンドはコントローラーを監視ステートに戻してエラー検出を再開するためのもので、エラー挿入インターフェイスまたはモニター インターフェイスから入力できます。

`status report` は診断情報を取得するためのコマンドで、コントローラーに対する「ping」のような働きをします。このコマンドはモニター インターフェイスからのみ入力できます。

`error injection` コマンドは、エラー挿入インターフェイスまたはモニター インターフェイスから何度でも入力できます。これらのコマンドは、コントローラーに対してエラー挿入を実行するよう命令します。アイドルステートの主な存在理由は、エラー検出に対する応答動作を停止して複数ビットエラーを挿入できるようにすることにあります。

`software reset` コマンドもエラー挿入インターフェイスまたはモニター インターフェイスから入力できます。このコマンドは、コントローラーに対してソフトウェア リセットを実行するよう命令します。

エラー挿入

エラー挿入ステートでは、コントローラーはエラーを挿入します。アイドルステートで有効なエラー挿入コマンドを受信すると、コントローラーは必ず挿入ステートを通過します。エラー挿入を無効にしている場合もモニター インターフェイスは存在するため、モニター インターフェイスからエラー挿入コマンドが入力されると挿入ステートを通過します。挿入ステートの間、`status_injection` は TRUE です。

エラー挿入プロセスは、エラー挿入コマンドで指定したコンフィギュレーション メモリ アドレスの1ビットをシンプルな Read-Modify-Write によって反転させます。

コントローラーは、エラー挿入ステートの次は必ずアイドルステートに遷移します。エラー挿入コマンドを繰り返し実行すると、1回のコマンド実行のたびにエラー挿入ステートへ遷移して複数ビット エラーを挿入できます。エラーの挿入が完了したら、コントローラーをアイドルステートから監視ステートに戻す必要があります。

エラー挿入

1ビット エラーを挿入するフレームのアドレス指定方法には2種類があります。1つはリニア フレーム アドレス指定で、もう1つは物理フレーム アドレス指定です。リニア フレーム アドレス指定はシンプルですが、このアドレスはフレームのタイプおよび物理位置に関する情報は提供しません。図 3-16 に、リニア フレーム アドレスの場合のエラー挿入コマンドのフォーマットを示します。境界線はニブル境界を表しており、コマンドの各ビット フィールドはグレーの濃淡で表しています。

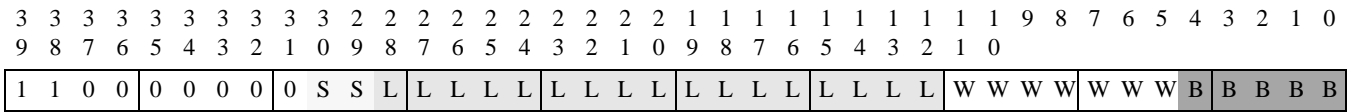


図 3-16: エラー挿入コマンド (リニア フレーム アドレス)

説明:

- SS = SSI の場合はハードウェア SLR 番号 (2 ビット)。非 SSI の場合は 00
- LLLLLLLLLLLLLLLLLL = リニア フレーム アドレス (17 ビット) [0..MF]
- WWWWWWW = ワード アドレス (7 ビット) [0..100]
- BBBBB = ビット アドレス (5 ビット) [0..31]

図 3-17 に、物理フレーム アドレスを使用したエラー挿入コマンドのフォーマットを示します。

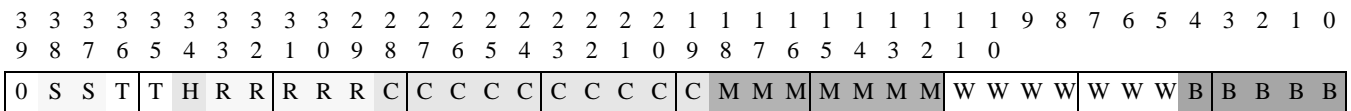


図 3-17: エラー挿入コマンド (物理フレーム アドレス)

説明:

- SS = SSI の場合はハードウェア SLR 番号 (2 ビット)。非 SSI の場合は 00
- TT = ブロック タイプ (2 ビット)
- H = ハーフ アドレス (1 ビット)
- RRRRR = 行アドレス (5 ビット)
- CCCCCCCCCC = 列アドレス (10 ビット)
- MMMMMMM = マイナー アドレス (7 ビット)
- WWWWWWW = ワード アドレス (7 ビット) [0..100]
- BBBBB = ビット アドレス (5 ビット) [0..31]

このコマンドの実行が完了すると、ステータス インターフェイスの status_injection 出力がディASSERTされ、コントローラーがエラー挿入ステートを終了したことが示されます。このコマンドの実行が完了すると、モニター インターフェイスでは、コマンドの実行が完了するとコントローラーがエラー挿入ステートを終了したことを示すコマンド プロンプトが返されます。

ソフトウェア リセット

コントローラーがアイドル ステートの場合、ソフトウェア リセット コマンドを使用してソフトウェア リセットを実行すると、初期化ステートを経由して監視ステートに移行させることができます。図 3-18 に、このコマンドのフォーマットを示します。「X」はドントケアを表します。

3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0										
1	0	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

図 3-18: ソフトウェア リセット コマンド

このコマンドの実行が完了すると、ステータス インターフェイスのステート出力が変化し、コントローラーが監視ステートに移行したことが示されます。またモニター インターフェイスでは、コマンドの実行が完了すると初期化レポートが出力された後、監視ステートに移行したことを示すコマンド プロンプトが返されます。

モニター インターフェイスのコマンド

ここでは、モニター インターフェイスからユーザーがコントローラーに送信できるコマンドについて説明します。これらのコマンドの一部は、エラー挿入インターフェイスで利用できるコマンドと同じです。

特定ステートへの移行

特定ステートへの移行コマンドを使用すると、コントローラーのステートを監視ステートとアイドル ステートの間で切り替えることができます。I コマンドはコントローラーをアイドル ステートに移行させます。O コマンドはコントローラーを監視ステートに移行させます。

ステータス レポート

S コマンドはコントローラーに対してステータス レポートの出力を要求します。ステータス レポートのフォーマットは次の「モニター インターフェイスのメッセージ」で説明します。このコマンドは、コントローラーがアイドル ステートまたは監視ステートの場合のみ実行できます。

エラー挿入

N コマンドはエラー挿入を実行します。このコマンドは、コントローラーがアイドル ステートの場合のみ実行できます。このコマンドのフォーマットは次のとおりです。

N {10-digit hex value}

このコマンドは、エラー挿入インターフェイスのエラー挿入コマンドと同じ働きをします。このコマンドの 10 桁の 16 進数値は、エラー挿入インターフェイスのエラー挿入コマンドで指定する 40 桁の 2 進数値と同じものを指定します。

ソフトウェア リセット

R コマンドはソフトウェア リセットを実行します。このコマンドは、コントローラーがアイドル ステートの場合のみ実行できます。このコマンドのフォーマットは次のとおりです。

R {2-digit hex value}

このコマンドは、エラー挿入インターフェイスのソフトウェア リセット コマンドと同じ働きをします。このコマンドで指定する 2 桁の 16 進数はドントケアです。

モニター インターフェイスのメッセージ

ここでは、コントローラーがモニター インターフェイスから出力するメッセージについて説明します。これらのメッセージの一部は、ステータス インターフェイスから出力されるレポートと同じものです。

初期化レポート

コントローラーが初期化シーケンスを実行中、初期化レポートが生成されます。このレポートには診断情報が含まれます。このレポートは、コントローラーの初回起動時およびソフトウェア リセット時に生成されます。

```

X7_SEM_V4_1      Name and Version
SC 01            State Change, Initialization
FS {2-digit hex value} Core Configuration Information
ICAP OK         Status: ICAP Available
RDBK OK        Status: Readback Active
INIT OK        Status: Completed Setup
SC 02          State Change, Observation
  
```

コマンド プロンプト

コントローラーから出力されるコマンド プロンプトは、コントローラーのステートに応じて2種類あります。コントローラーが監視ステート (初期化完了後のデフォルト ステート) の場合は、コマンド プロンプト `O>` が出力されます。コントローラーがアイドルステートの場合は、コマンド プロンプト `I>` が出力されます。

ステート変化レポート

コントローラー ステートが変化すると、必ずステート変化レポートが出力されます。このレポートは1行で、フォーマットは次のとおりです。

```
SC {2-digit hex value}
```

2桁の16進数値でステータス インターフェイスの出力を表現します。

表 3-2: ステート変化レポートのデコード

レポートの文字列	ステート名
SC 00	アイドル
SC 01	初期化
SC 02	監視
SC 04	訂正
SC 08	分類
SC 10	エラー挿入
SC 1F	重大エラー

重大エラー ステートへの移行はいつでも起こる可能性があります。また、ステート変更レポートが明示的に出力されない場合もあります。重大エラー ステートに移行すると、コントローラーは次の重大エラー メッセージを出力します。

```
HLT
```

フラグ変化レポート

コントローラーは、フラグを変更すると必ずフラグ変化レポートを出力します。このレポートは1行で、フォーマットは次のとおりです。

```
FC {2-digit hex value}
```

2桁の16進数値でステータス インターフェイスの出力を表現します。

表 3-3: フラグ変化レポートのデコード

レポートの文字列	変更後のフラグ
FC 00	訂正可能、非エッセンシャル
FC 20	訂正不能、非エッセンシャル
FC 40	訂正可能、エッセンシャル
FC 60	訂正不能、エッセンシャル

ステータス レポート

ステータスレポートからは、コントローラーのステートに関するより詳細な情報が得られます。このレポートは、コントローラーが監視ステートまたはアイドル ステートのときにs コマンドを入力すると、複数行のレポートとして出力されます。非 SSI デバイスのステータスレポートのフォーマットは次のとおりです。

```
MF {8-digit hex value} Maximum Frame (linear count)
SN {2-digit hex value} Hardware SLR Number
SC {2-digit hex value} Current State
FC {2-digit hex value} Current Flags
FS {2-digit hex value} Feature Set
```

SSI デバイスのステータスレポートも基本的なフォーマットは非 SSI デバイスと同じですが、SLR ごとにサブレポートが出力されます。サブレポートはハードウェア SRL 番号の順にソートされます。たとえば4つのSLRで構成されるデバイスの場合、ステータスレポートのフォーマットは次のとおりです。

```
MF {8-digit hex value} Maximum Frame (linear count)
SN {2-digit hex value} Hardware SLR Number
SC {2-digit hex value} Current State for this Hardware SLR
FC {2-digit hex value} Current Flags for this Hardware SLR
FS {2-digit hex value} Feature Set
MF {8-digit hex value} Maximum Frame (linear count)
SN {2-digit hex value} Hardware SLR Number
SC {2-digit hex value} Current State for this Hardware SLR
FC {2-digit hex value} Current Flags for this Hardware SLR
FS {2-digit hex value} Feature Set
MF {8-digit hex value} Maximum Frame (linear count)
SN {2-digit hex value} Hardware SLR Number
SC {2-digit hex value} Current State for this Hardware SLR
FC {2-digit hex value} Current Flags for this Hardware SLR
FS {2-digit hex value} Feature Set
MF {8-digit hex value} Maximum Frame (linear count)
SN {2-digit hex value} Hardware SLR Number
SC {2-digit hex value} Current State for this Hardware SLR
FC {2-digit hex value} Current Flags for this Hardware SLR
FS {2-digit hex value} Feature Set
```

エラー検出レポート

エラー条件を検出すると、コントローラーはなるべく短時間でエラーを訂正しようと試みます。したがって、エラーが訂正可能であるという前提に立ち、最初に訂正を実行してからレポート情報が生成されます。次のようなレポートが出力されます。

診断 : CRC エラーのみ (エラー ビットの位置または数を特定できない)。

```
SC 04
CRC
```

診断 : 拡張修復チェックサム バッファオーバー訂正不能エラー。

```
SC 04
BFR
```

診断 : 1 ビット ECC エラー、SYNDROME は有効。コントローラーは物理フレーム アドレス、リニア フレーム アドレス、フレーム内のワード位置、ワード内のビット位置を報告します。

```
SC 04
SED OK
PA {8-digit hex value}
LA {8-digit hex value}
WD {2-digit hex value} BT {2-digit hex value}
```

診断 : 1 ビット ECC エラー、SYNDROME は無効、予測しない値域外のワード アドレス。コントローラーは物理フレーム アドレスとリニア フレーム アドレスを報告します。この診断コードが出力された場合、FPGA のリコンフィギュレーションが必要です。

```
SC 04
SED NG
PA {8-digit hex value}
LA {8-digit hex value}
```

診断 : 2 ビット ECC エラー。コントローラーは物理フレーム アドレスとリニア フレーム アドレスを報告します。

```
SC 04
DED
PA {8-digit hex value}
LA {8-digit hex value}
```

エラー訂正レポート

エラー訂正プロセスは、コントローラーの設定、検出されたエラーの種類、およびエラーが訂正可能かどうかにより異なります。

訂正不能エラー、または訂正を無効にした場合のレポートのフォーマットは次のとおりです。

```
COR
END
```

この後に、次の行が続きます。

```
FC 20          Bit 5, uncorrectable set (stale essential flag)
```

すなわち

```
FC 60          Bit 5, uncorrectable set (stale essential flag)
```

訂正可能エラーの場合のレポートのフォーマットは次のとおりです。

```
COR
{correction list}
END
```

この後に、次の行が続きます。

```
FC 00          Bit 5, uncorrectable cleared (stale essential flag)
```

すなわち

```
FC 40          Bit 5, uncorrectable cleared (stale essential flag)
```

{correction list} には、1 件の訂正されたエラーにつきフレーム内のワード位置とワード内のビット位置が 1 行で表示されます。このリストは数千行を超えることもあります。この記法は、エラー検出レポートの場合と同じです。リストの各行は次のフォーマットで表示されます。

```
WD {2-digit hex value} BT {2-digit hex value}
```

エラー分類レポート

エラー分類プロセスでは、フレーム内で発生したエラーの中にエッセンシャルなものがあるかどうかをルックアップテーブルで判定します。1 つまたは複数のエラーがエッセンシャルと判定された場合、イベント全体がエッセンシャルと見なされます。

エラー分類が有効な場合、訂正可能な非エッセンシャル イベントのレポートのフォーマットは次のとおりです。

```
SC 08
CLA
END
FC 00          Bit 6, essential is cleared
```

エラー分類が有効な場合、訂正可能なエッセンシャル イベントのレポートのフォーマットは次のとおりです。

```
SC 08
CLA
{classification list}
END
FC 40          Bit 6, essential is set
```

{classification list} には、1 件のエッセンシャルビットにつきフレーム内のワード位置とワード内のビット位置が 1 行で表示されます。このリストは数千行を超えることもあります。この記法は、エラー検出レポートの場合と同じです。リストの各行は次のフォーマットで表示されます。

```
WD {2-digit hex value} BT {2-digit hex value}
```

エラー分類を無効にした場合は、詳細な分類リストは出力されません。この場合、コントローラーはエラーがエッセンシャルでないと判定する根拠を持たないため、すべてのエラーをエッセンシャルと見なす必要があります。訂正可能イベントの場合のレポートのフォーマットは次のとおりです。

```
SC 08
FC 40          Bit 6, essential is set
```

エラーが訂正不能な場合、コントローラーはエラーがエッセンシャルでないと判定する根拠を持たないため、すべての訂正不能エラーをエッセンシャルと見なす必要があります。訂正不能イベントの場合のレポートのフォーマットは次のとおりです。

```
SC 08
FC 60          Bit 6, essential is set
```

システム

このソフト エラー軽減ソリューションは自律動作が可能ですが、多くのアプリケーションではシステム レベルの監視機能と組み合わせて使用します。システム レベルの監視機能を実装すべきかどうか、そしてこの監視機能にどれだけの役割を持たせるかは、個々のシステムにより異なります。

システム レベルの監視機能を用いてソフト エラー軽減ソリューションを監視する方法には、次のものがあります。



ヒント：これらはいずれも必須ではありません。

- ソフト エラー軽減ソリューションを監視し、ソフト エラー イベントへの応答としてシステム レベルでの追加措置が必要かどうかを判定する。これには、ソフト エラー イベントのログを記録するといった一般的な対処から、エラーのタイプと分類結果に応じたアプリケーション固有の非常に複雑な応答（ロジックのリセット、デバイスのリコンフィギュレーションや再起動など）までさまざまなものがあります。

ソフト エラー軽減ソリューションのイベント レポートを監視するには、ステータス インターフェイスの `status_correction` および `status_uncorrectable` 信号、ステータス インターフェイスの `status_classification` および `status_essential` 信号、またはモニター インターフェイスの `monitor_tx` 信号によるエラー検出、訂正、分類レポートを使用します。

- ソフト エラー軽減ソリューションを監視し、ソリューション自体に問題がないかを確認する。[第2章の「ソリューションの信頼性」](#)で数値を示して説明したとおり、ごくわずかながらソフト エラー軽減ソリューション自体が障害を起こす可能性があります。統計的に、この障害はコントローラーのすべての状態で発生する可能性があります。

- ブートおよび初期化**：ソフト エラー軽減ソリューションを監視し、ソリューションがブートおよび初期化の後、監視ステートに移行するかを確認します。ザイリンクスは、ソフト エラー軽減ソリューションが監視ステートに移行する前にブートおよび初期化を実行する時間を[表 2-4](#) および[式 2-1](#) で定義しています。ただしこれは `icap_grant` 信号がアサートされており、ソフト エラー軽減ソリューションが ICAP プリミティブ経由で FPGA コンフィギュレーション ロジックを利用でき、モニター インターフェイスでスロットリングが発生していない場合の値です。

ソフト エラー軽減ソリューションが初期化に失敗または監視ステートへの移行に失敗する主な理由は、ソフト エラー イベントではなくデザイン エラーです。これには、`icap_grant` 信号の制御の誤りや ICAP 共有の実装の誤りがあります。また、ソフト エラー軽減ソリューションが ICAP プリミティブ経由で FPGA コンフィギュレーション ロジックを利用できないことが原因となることもあります。この問題が発生する理由としては、ソフト エラー軽減ソリューションと互換性のないビットストリーム オプションを使用した、JTAG 経由で FPGA に発行された FPGA コンフィギュレーション命令をシステム レベルの JTAG コントローラーが正しく完了またはクリアできなかった、などが考えられます。

ソリューションが初期化を完了して監視ステートに移行したことを確認するには、システム レベルの監視機能でステータス インターフェイスの `status_initialization` および `status_observation` 信号がアサートされるのを監視するか、モニター インターフェイスの `monitor_tx` 信号から予想される初期化レポートが出力されるのを監視します。

- 監視ステート**：コントローラーはほとんどすべての時間をこのステートで費やします。このステートのコントローラーを監視する方法は少なくとも3つあり、コントローラーの状態に関して得られる情報がそれぞれ少しずつ異なります。
 - コントローラーのハートビート信号 `status_heartbeat`：この信号は、ソフト エラー軽減ソリューションから直接出力されます。この信号は、リードバック プロセスが動作中であることを示すパルスです（仕様の詳細は「ステータス インターフェイス」を参照）。監視ステート中にこのパルスが仕様から逸脱した場合、システム レベルの監視機能はリードバック プロセスに障害が発生したものと判定できます。これは訂正不能なエッセンシャル エラーです。SSI へのインプリメンテーションでは SLR ごとに `status_heartbeat` 出力があるため、すべての SLR からのハートビート信号を監視する必要があります。

`status_heartbeat` はほかのコントローラー ステートでは未定義のため、監視ステートでのみ監視するようにしてください。

- CRC エラー ステータス信号 INIT_B: この信号は、リードバック プロセスから直接出力されます。リードバック プロセスで CRC エラーが検出されると、INIT_B がアサートされます。監視ステート中に INIT_B がアサートされてから 1 秒以内に訂正ステートに移行しない場合、コントローラーに障害が発生しています。ステートの移行は、ステータス インターフェイスの status_correction 信号またはモニター インターフェイスのステート変化レポートで確認できます。これは訂正不能なエッセンシャル エラーです。SSI デバイスへのインプリメンテーションでは SLR ごとに内部 CRC エラー ステータス信号があるため、これらの信号をワイヤード OR によって 1 つの INIT_B デバイスピンに接続します。ただしステータス インターフェイスは SLR ごとに status_correction 信号があります。

INIT_B はほかのコントローラー ステートでは未定義のため、監視ステートでのみ監視するようにしてください。

- コントローラーのステータス コマンドおよびレポート: モニター インターフェイスの monitor_rx および monitor_tx 信号を使用してシステム レベルの監視機能から周期的にステータス コマンドを送信し、予想されるステータス レポートが出力されるかを確認します。コントローラーのステートが変化していない場合、1 秒以内に予想されるステータス レポートが出力されなければ、システム レベルの監視機能はコントローラーに障害が発生したものと判定できます。これは訂正不能なエッセンシャル エラーです。

この方法を使用する場合、「コントローラー応答なし」条件を検出する時間を許容範囲内としつつ、ステータス コマンドの送信間隔をなるべく長くする必要があります。

ステータス コマンドとレポートをコントローラーで処理する方法は、レイテンシを増大させる要因となることがあります。たとえばステータス コマンドを 60 秒ごとに送信するのは、一般的な動作のレイテンシをそれほど増やすことなく、ほとんど発生しない「コントローラー応答なし」条件を防ぐことができるため、妥当なトレードオフといえます。これに対し、ステータス コマンドを 1 秒ごとに送信するのは賢明な選択とはいえません。この場合、ステータス レポートによって MON シムの送信バッファが空になる時間がほとんどなく、モニター インターフェイスでスロットリングが発生してエラー検出、訂正、および分類のレイテンシが増大します。

コントローラーのステータス コマンドおよびレポートを使用する方法は、監視およびアイドル ステートでしか利用できない点に注意が必要です。ほかのステート中にステータス コマンドを送信した場合、MON シムの受信バッファがオーバーフローしなければコマンドはバッファ内に残り、監視またはアイドル ステートに移行した時点で処理されます。

- 。 **訂正および分類ステート**: モニター インターフェイスでスロットリングが発生していなければ、ソフト エラー軽減ソリューションは訂正および分類ステートに表 2-6/式 2-3 および表 2-8/式 2-4 に示した時間内に遷移します。ソフト エラーが発生する確率は非常に低いため、コントローラーがこれらのステートで費やす時間は非常に短く、通常はすぐに監視ステート（またはごくまれにアイドル ステート）に戻ります。ステータス インターフェイスの status_correction および status_classification 信号、またはモニター インターフェイスのステート変化レポートを監視して、コントローラーが訂正または分類ステートにとどまる時間が 1 秒を超えたことが確認されると、システム レベルの監視機能はコントローラーに障害が発生したものと判定できます。これは訂正不能なエッセンシャル エラーです。

これとは別に、ソフト エラー軽減ソリューションが同じアドレスを繰り返して訂正していないかをシステム レベルの監視機能で監視することもできます。このような症状が起こることはほとんどありませんが、その原因はコントローラーのソフト エラーやデバイス自体のハード エラーなどいくつか考えられます。

- 。 **アイドルおよびエラー挿入ステート**: コントローラーがアイドル ステートに遷移するのは、訂正不能エラーが発生した場合かアイドル ステートへの移行コマンドが実行された場合のみです。訂正不能エラーによってアイドル ステートに遷移した場合は、イベント レポートの監視に関するセクションを参照してください。アイドル ステートへの移行コマンドは、主にエラー挿入や ICAP 共有などほかのコマンドを発行するために使用します。「監視ステート」の「コントローラーのステータス コマンドおよびレポート」で説明した方法をアイドル ステートで実装するのは、アプリケーション レベルでほかのプロセスによって発行されたコマンドと競合する可能性があるため推奨しません。発行済みのコマンドが完了して 1 秒以内に応答が生成されることをアプリケーション レベル プロセスで確認する方法を推奨します。1 秒以内に応答が生成されない場合は訂正不能なエッセンシャル エラーが発生しており、アプリケーションからシステムに報告する必要があります。

- 。 **重大エラー ステート**：コントローラーは、内部ステートに不整合性が検出された場合のみこのステートに移行します。このステートに移行すると、ステータス インターフェイスの5つのステート信号がすべてアサートされ、モニター インターフェイスからは **HLT** メッセージが出力されます。

SSI デバイスへのインプリメンテーションでは複数のコントローラー インスタンスが存在しており、1つまたは複数のコントローラー インスタンスが動作を停止するとソリューション全体が重大エラー ステートと見なされます。これは訂正不能なエッセンシャル エラーです。

カスタマイズ

システム レベル サンプル デザインは **SEM Controller** と各種シムをカプセル化しており、これらのシムが **SEM Controller** とほかのデバイスを接続するインターフェイスとしての役割を果たします。シムには、**I/O** ピン、**Vivado Design Suite** デバッグ機能インターフェイス、**I/O** インターフェイス、メモリ コントローラー、アプリケーション固有のシステム管理インターフェイスなどがあります。

付属するシステム レベル サンプル デザインはリファレンス デザインではありませんが、ソリューション全体を構成する必須要素であり、ザイリンクスによって完全に検証されています。システム レベル サンプル デザインは設計者が自由に変更できますが、そのまま使用することを推奨します。ただし、変更が必要な場合はこのセクションの情報を参考にして正しくカスタマイズしてください。

このセクションでは、システム レベル サンプル デザインに含まれるアプリケーションのカスタマイズ方法は説明しません。このアプリケーションはデモ用で、その機能はソフト エラー軽減とは無関係です。このアプリケーションは削除できますが、カスタマイズは想定していません。

HID シムのカスタマイズ

HID シムはコントローラーとインターフェイス デバイスを接続するブリッジの役割を果たします。このシムが提供するインターフェイスを使用して、コントローラーとの間でコマンドとステータス情報を交換します。**HID** シムは、コントローラーの設定によっては存在しないこともあります。**HID** シムが存在する場合、ステータス インターフェイスとエラー挿入インターフェイスに **Vivado** ロジック解析機能からアクセスできます。**HID** シムが存在しない場合、ステータス インターフェイスとエラー挿入インターフェイスには **I/O** ピンからしかアクセスできません。

ステータス インターフェイスとエラー挿入インターフェイスは別の方法で接続することもできます。これらのインターフェイスは **HID** シムまたは **I/O** ピンとの接続を簡単に解除してレジスタ ファイルや有限ステート マシンなどほかのロジックに接続できます。詳細は [30 ページの「ステータス インターフェイス」](#) および [46 ページの「エラー挿入インターフェイス」](#) を参照してください。

MON シムのカスタマイズ

MON シムはコントローラーと標準 **RS-232** ポートを接続するブリッジの役割を果たします。このシムが提供するインターフェイスを使用して、コントローラーとの間でコマンドとステータス情報を交換します。このインターフェイスは、プロセッサと接続するように設計されています。

ビット レートを引き上げる

RS-232 インターフェイスを使用する場合は、なるべく高いビット レートに変更することを強く推奨します。こうすると、コントローラーがステータス レポートを送信する際にスロットリングが発生しにくくなります。詳細は、[32 ページの「モニター インターフェイス」](#) を参照してください。

バッファの段数を増やす

コントローラーがステータスレポートを送信する際のスロットリングを防ぐ最も簡単な方法は、MON シムのビットレートを引き上げることです。それ以外に、バッファ段数を増やす方法もあります。MON シムには送信バッファと受信バッファの2つの FIFO があります。

これらバッファの段数は非対称でもかまいません。受信バッファの段数を増やしてもほとんどメリットはありませんが、送信バッファの深さを大きくするとコントローラーがステータスレポートを送信する際にスロットリングが発生しにくくなります。

MON シムの FIFO を置き換える場合は、ザイリンクス LogiCORE IP FIFO Generator を使用できます。FIFO は共通クロック (すなわち単一クロックに完全同期) に設定し、FWFT (First Word Fall Through) を有効にする必要があります。データ幅は 8 とし、深さは必要なだけ大きく設定します。



ヒント : FIFO を置き換えると、empty フラグは data present フラグの論理値を反転したものになる点に注意してください。

別の機能への置き換え (非 SSI デバイス)

RS-232 以外のインターフェイスが必要な場合、MON シムを別の機能に置き換えることができます。たとえば MON シムをカスタム プロセッサ インターフェイスや、プロセス間通信をサポートする別の方式と置き換えることができます。MON シムを別の機能に置き換える場合、コントローラーのモニター インターフェイスの動作を十分に理解しておく必要があります。

これら信号の概要については、32 ページの「モニター インターフェイス」を参照してください。このインターフェイスでは ASCII データが交換されます。ステータスおよびコマンドのフォーマットの概要は、53 ページの「モニター インターフェイスのコマンド」および 54 ページの「モニター インターフェイスのメッセージ」を参照してください。

図 3-19 に、モニター インターフェイスの送信プロトコルを示します。1 バイトのデータを送信する場合、コントローラーはまず monitor_txfull 信号をサンプリングして送信バッファに 1 バイトのデータを格納できる空きがあるかを調べます。monitor_txfull が Low の場合、コントローラーは monitor_txdata[7:0] にデータを印加し、monitor_txwrite を 1 クロック サイクルだけアサートして 1 バイトのデータを送信します。

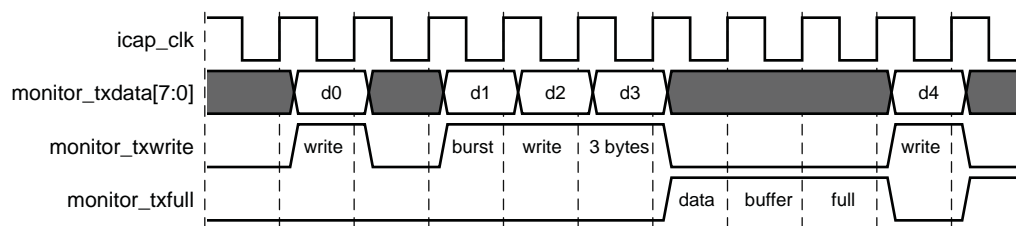


図 3-19 : モニター インターフェイスの送信プロトコル

コントローラーは、monitor_txdata[7:0] にデータシーケンスを印加し、monitor_txwrite を複数サイクル続けてアサートすることでバースト書き込みを実行できます。ただしコントローラーは monitor_txfull を監視することにより、送信バッファ オーバーランを防いでいます。

データ受信側のペリフェラルは、バッファの空きを正しく追跡して monitor_txfull で報告する必要があります。ペリフェラルは、monitor_txwrite = High をサンプリングした次のサイクルで、データの書き込みによってバッファが完全にフルである場合は monitor_txfull をアサートする必要があります。

また、ペリフェラルはコントローラーからの monitor_txwrite への応答としてのみ monitor_txfull をアサートする必要があります。ペリフェラル内部のイベントに基づいて monitor_txfull をアサートすることはできません。これは、コントローラーが書き込みを実行する何サイクルか前に monitor_txfull をサンプリングすることがあるためです。

ペリフェラルが `monitor_txfull` をアサートしている間、コントローラーは送信待ちのデータがある場合でも動作を停止します。これにより、コントローラーのエラー軽減パフォーマンスが低下します。たとえばエラー訂正が発生してコントローラーがエラーを正しく訂正(または処理)した場合、コントローラーはステータスレポートを送信します。ペリフェラルがこのメッセージを完全に受信するまでコントローラーは動作を停止し、監視状態に戻ることができません。したがって、カスタムペリフェラルを設計する際はバッファ深さとデータ消費レートのバランスを考慮する必要があります。

図 3-20 に、モニター インターフェースの受信プロトコルを示します。1 バイトのデータを受信する場合、コントローラーはまず `monitor_rxempty` 信号をサンプリングして受信バッファに 1 バイトのデータが格納されているかを調べます。`monitor_rxempty` が Low の場合、コントローラーは `monitor_rxdata[7:0]` から 1 バイトのデータを受信し、`monitor_rxread` を 1 クロック サイクルだけアサートして受信完了を通知します。

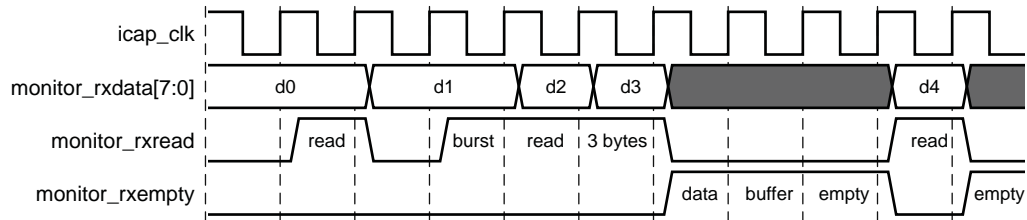


図 3-20: モニター インターフェースの受信プロトコル

コントローラーは、`monitor_rxdata[7:0]` からデータシーケンスを取得し、`monitor_rxread` を複数サイクル続けてアサートすることでバースト読み出しを実行できます。ただしコントローラーは `monitor_rxempty` を監視することにより、受信バッファアンダーランを防いでいます。

データ送信側のペリフェラルは、バッファのステータスを正しく追跡して `monitor_rxempty` で報告する必要があります。ペリフェラルは、`monitor_rxread = High` をサンプリングした次のサイクルで、データの読み出しによってバッファが完全に空である場合は `monitor_rxempty` をアサートする必要があります。

また、ペリフェラルはコントローラーからの `monitor_rxread` への応答としてのみ `monitor_rxempty` をアサートする必要があります。ペリフェラル内部のイベントに基づいて `monitor_rxempty` をアサートすることはできません。これは、コントローラーが読み出しを実行する何サイクルか前に `monitor_rxempty` をサンプリングすることがあるためです。

初期化状態中、コントローラーは受信バッファが空になるまでデータを読み出して破棄して受信バッファをページします。コントローラーには送信バッファの状態を監視する手段がないため、送信バッファの準備がただちに完了したものと仮定し、送信バッファが空になるまで待ちません。

別の機能への置き換え (SSI デバイス)

SSI デバイスへのインプリメンテーション用のシステム レベル サンプル デザインに含まれる MON シムには、複数のコントローラー インスタンスに対応したアービトレーションやレポート照合などの機能が追加されています。

SSI デバイスの場合、MON シムを別の機能で置き換えないことを強く推奨します。

削除

MON シムを削除することは可能ですが、そのようなカスタマイズは避けることを強く推奨します。MON シムを削除する場合は、MON シムが使用する 2 つの I/O ピンをテスト ポイントでプローブできるように確保することを推奨します。アプリケーションによっては MON シムが不要なこともあります。MON シムにはデバッグに役立つ重要な機能があり、この情報がザイリンクスのテクニカル サポートで必要となることがあります。

MON シムを削除する場合は、RS-232 の信号をデザイン最上位の I/O ピンに接続するための信号およびポートをすべて接続解除して削除してください。次に、コントローラーのモニター インターフェイスを次のように接続します。

- コントローラーのモニター インターフェイスのすべての出力をオープンまたは未接続のままにしておく。
- すべての monitor_txfull 入力を論理 0 に接続する。
- すべての monitor_rxempty 入力を論理 1 に接続する。
- すべての monitor_rxdata[7:0] 入力を論理 0 に接続する。

MON シムを削除するとソリューションのサイズが小さくなり、ステータス レポート送信時のスロットリングも発生しません。

EXT シムのカスタマイズ

EXT シムはコントローラーと標準 SPI バスを接続するブリッジの役割を果たします。このシムが提供するインターフェイスを使用してコントローラーは外部からデータをフェッチします。このシムは標準 SPI フラッシュに接続するためのもので、コントローラーの設定によっては存在しないこともあります。

コマンド シーケンスの変更

SPI バス インターフェイスが必要な場合、SPI フラッシュ デバイスに送信するコマンド シーケンスに変更が必要となることがあります。EXT シムがサポートする SPI フラッシュ ファミリーは、いくつかの種類から選択できます。詳細は、「外部インターフェイス」を参照してください。

別の機能への置き換え (非 SSI デバイス)

SPI バス以外のインターフェイスが必要な場合、EXT シムを別の機能に置き換えることができます。たとえば EXT シムをパラレル フラッシュ メモリ コントローラーや、プロセス間通信をサポートする別の方式と置き換えることができます。EXT シムをほかの機能に置き換える場合、コントローラーのフェッチ インターフェイスの動作を理解しておくことが重要です。

これら信号の概要は、第 2 章の「フェッチ インターフェイス」を参照してください。フェッチ インターフェイスのバイト転送プロトコルは、モニター インターフェイスと同じです。

図 3-21 に、フェッチ インターフェイスの送信プロトコルを示します。1 バイトのデータを送信する場合、コントローラーはまず fetch_txfull 信号をサンプリングして送信バッファーに 1 バイトのデータを格納できる空きがあるかを調べます。fetch_txfull が Low の場合、コントローラーは fetch_txdata[7:0] にデータを印加し、fetch_txwrite を 1 クロック サイクルだけアサートして 1 バイトのデータを送信します。

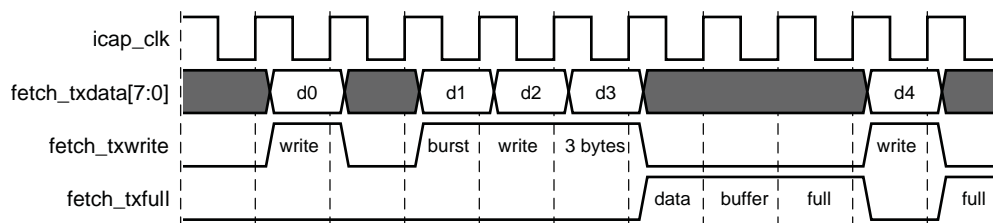


図 3-21: フェッチ インターフェイスの送信プロトコル

コントローラーは、fetch_txdata[7:0] にデータ シーケンスを印加し、fetch_txwrite を複数サイクル続けてアサートすることでバースト書き込みを実行できます。ただしコントローラーは fetch_txfull を監視することにより、送信バッファー オーバーランを防いでいます。

データ受信側のペリフェラルは、バッファーの空きを正しく追跡して fetch_txfull で報告する必要があります。ペリフェラルは、fetch_txwrite = High をサンプリングした次のサイクルで、データの書き込みによってバッファーが完全にフルである場合は fetch_txfull をアサートする必要があります。

また、ペリフェラルはコントローラーからの `fetch_txwrite` への応答としてのみ `fetch_txfull` をアサートする必要があります。ペリフェラル内部のイベントに基づいて `fetch_txfull` をアサートすることはできません。これは、コントローラーが書き込みを実行する何サイクルか前に `fetch_txfull` をサンプリングすることがあるためです。

ペリフェラルが `fetch_txfull` をアサートしている間、コントローラーは送信待ちのデータがある場合でも動作を停止します。これにより、コントローラーのエラー軽減パフォーマンスが低下します。

図 3-22 に、フェッチ インターフェイスの受信プロトコルを示します。1 バイトのデータを受信する場合、コントローラーはまず `fetch_rxempty` 信号をサンプリングして受信バッファに 1 バイトのデータが格納されているかを調べます。`fetch_rxempty` が Low の場合、コントローラーは `fetch_rxdata[7:0]` から 1 バイトのデータを受信し、`fetch_rxread` を 1 クロック サイクルだけアサートして受信完了を通知します。

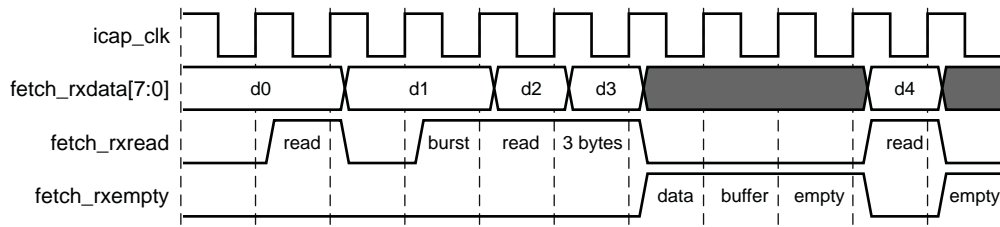


図 3-22：フェッチ インターフェイスの受信プロトコル

コントローラーは、`fetch_rxdata[7:0]` からデータ シーケンスを取得し、`fetch_rxread` を複数サイクル続けてアサートすることでバースト読み出しを実行できます。ただしコントローラーは `fetch_rxempty` を監視することにより、受信バッファ アンダーランを防いでいます。

データ送信側のペリフェラルは、バッファのステータスを正しく追跡して `fetch_rxempty` で報告する必要があります。ペリフェラルは、`fetch_rxread = High` をサンプリングした次のサイクルで、データの読み出しによってバッファが完全に空である場合は `fetch_rxempty` をアサートする必要があります。

また、ペリフェラルはコントローラーからの `fetch_rxread` への応答としてのみ `fetch_rxempty` をアサートする必要があります。ペリフェラル内部のイベントに基づいて `fetch_rxempty` をアサートすることはできません。これは、コントローラーが読み出しを実行する何サイクルか前に `fetch_rxempty` をサンプリングすることがあるためです。

初期化ステート中、コントローラーは受信バッファが空になるまでデータを読み出して破棄して受信バッファをページします。その後、コントローラーはさらに 2 回の読み出しを実行します。空のバッファからの読み出しは、リセット信号としてデコードされます。コントローラーには送信バッファの状態を監視する手段がないため、リセット信号によって送信バッファが初期化されるものと仮定し、送信バッファが空になるまで待ちません。

データは 2 進数で交換され、コマンド/応答ペアを表します。コントローラーが 6 バイトのコマンド シーケンスを送信するとデータ フェッチが開始します。このコマンド シーケンスは次のとおりです。

- バイト 1: ADD[31:24]
- バイト 2: ADD[23:16]
- バイト 3: ADD[15:8]
- バイト 4: ADD[7:0]
- バイト 5: LEN[15:8]
- バイト 6: LEN[7:0]

これに回答して、ペリフェラルは ADD[31:0] で指定されたアドレスを先頭に LEN[15:0] で指定されたバイト数のデータをフェッチして、このデータをコントローラーに返す必要があります。コントローラーがコマンドを発行した後、ペリフェラルは要求された正確なバイト数を返してコマンドを完了する必要があります。アドレス空間におけるデータ構成の詳細は、第 5 章の「外部メモリ プログラミング ファイル」を参照してください。

別の機能への置き換え (SSI デバイス)

SSI デバイスへのインプリメンテーション用のシステム レベル サンプル デザインに含まれる EXT シムは、複数のコントローラー インスタンスに対応したアービトレーションなどの機能を追加しており、複雑です。SSI デバイスの場合、EXT シムを別の機能で置き換えないことを強く推奨します。

データ整合性

置換によるエラー訂正およびエラー分類のオプション機能を使用する場合、コントローラーから外部データへのアクセスが必要です。このデータは、`write_bitstream` によって FPGA のプログラミング ファイルと同時に作成されます。これらのファイルは相互に関連しています。

FPGA デザインを変更して新しいプログラミング ファイルを作成する際は、コントローラーが使用する外部データ ファイルも更新する必要があります。ハードウェア デザインを新しいプログラミング ファイルで更新する際に、外部に格納するデータも更新する必要があります。



重要 : データの整合性が失われた場合、コントローラーの動作は予測不能となります。プログラミング ファイルと外部データ ファイルが常に同期するような更新方法を採用することを推奨します。

コンフィギュレーション メモリのマスク

デザインによっては、動作中にコンフィギュレーション メモリの一部のビット値が変化することがあります。よくあるのは、分散 RAM やシフト レジスタなどの LUTRAM 機能を実装するようにロジック スライス リソースを設定した場合です。また、ダイナミック リコンフィギュレーション ポートを持つその他のタイプのリソースがデザインの動作中に更新される場合にもこのような状況が発生します。

エラーの誤検出を防ぐには、これらリソースに関連するメモリ ビットはマスクしておき、CRC および ECC 計算から除外しておく必要があります。ザイリンクスの FPGA デバイスには、このようなエラー誤検出を防ぐためにコンフィギュレーション メモリのマスク機能が実装されています。マスクを有効にするかどうかは、グローバル制御信号の `GLUTMASK` で選択します。コントローラーは常にマスクを有効にします。

7 シリーズ FPGA と Zynq-7000 SoC は、リソース レベルで細粒度のマスクを実装しています。つまり、個々のリソースをダイナミック動作に設定すると、そのリソースのコンフィギュレーション メモリ ビットがマスクされます。無関係なメモリ ビットには影響せず、必要なメモリ ビットのみがマスクされます。マスクしたビットは、コントローラーによる監視を受けません。

マスクしたリソースに関するコンフィギュレーション メモリ ビットを読み出すと、定数値 (論理 1 または論理 0) が返されます。これにより、エラーの誤検出を防ぎます。マスクされたリソースに関するコンフィギュレーション メモリ ビットへの書き込みは破棄されます。これにより、ダイナミック ステート エレメントの内容を古いデータで上書きするのを防ぎます。ただしその副作用として、マスクしたリソースにエラーを挿入してもエラーが検出されません。

LUTRAM 機能など多くの場合には、ユーザー デザインでこれらのビットに対するデータ保護を実装してソフト エラーを軽減できます。または、ユーザー デザインを変更してコンフィギュレーション メモリのマスクを必要とするような機能を使用しないようにする方法もあります。

クロッキング

マスター クロックはコントローラーにとって非常に重要であるため、なるべく信頼性の高いクロック ソースから供給する必要があります。最大限の信頼性を確保するには、クロック ソースとコントローラーをできるだけ直接接続します。つまり、目的の周波数の外部オシレーターを使用し、グローバル クロック バッファに直接関係するピンに直接接続します。

このクロック パスにその他のロジックやインターコネクトを含めると、クロックとコントローラーの接続を制御するために使用するコンフィギュレーション メモリの数が増大します。この結果、コントローラーの推定 FIT に悪影響が及びます。この影響はそれほど大きくはありませんが、設計負担がそれほど増えないのであれば、信頼性向上のために実行しておくことを推奨します。

このクロック パスにクロック管理ブロックやロジック ベースのクロック分周器などの余分なロジックが存在する場合、過渡中を含め FPGA コンフィギュレーション システムの最大クロック周波数とシステム レベル サンプル デザインおよびコントローラーの最大クロック周波数の要件に違反しないようにデザインで保証する必要があります。たとえば DLL や PLL のクロック出力は、これらの機能がロックするまでの間、仕様外の値となることがあります。この問題に対処するには、イネーブル信号を備えたグローバル クロック バッファを使用し、ロックが完了した後でグローバル クロック バッファを有効にするという方法があります。

システム レベル サンプル デザイン、コントローラー、およびコンフィギュレーション システムはすべてスタティックです。つまり、FPGA コンフィギュレーション システムの仕様上の最大許容周波数、またはシステム レベル サンプル デザインおよびコントローラーの最大クロック周波数のどちらか低い方まで任意のクロック周波数を使用できます。ただしクロック レートは高い方がエラー軽減を高速に実行できるという利点があります。

リセット

デバイスのコンフィギュレーション全体をリセットすることはできないため、コントローラーにはリセットがありません。このコントローラーは、デバイスのコンフィギュレーション完了から電源停止またはリコンフィギュレーションまでの間、デバイスのコンフィギュレーションを監視します。コントローラーは、オリジナルのコンフィギュレーション ステートを監視および維持することを役割とし、エラーを含む可能性が高い暫定ステートからは再起動しません。

その他の注意事項

SEM コアには、次に示す設計上の制限事項があります。

- 置換によるエラー訂正には、EasyPath™ デバイスは使用できません。
- SEM Controller は、ソフト エラー軽減のための FPGA 内蔵シリコン機能を初期化および管理します。SEM Controller をデザインに含める場合、FPGA 内蔵の検出機能を有効にするようなデザイン制約またはオプションは使用しないでください。検出に必要な機能は、SEM Controller によって自動的に有効にされます。たとえば POST_CRC、POST_CONFIG_CRC、またはこれらに関連する制約を設定しないでください。同様に、GLUTMASK を無効にするようなオプションも使用しないでください。SEM コアによるエラーの誤検出を防ぐには、デフォルト値の YES としておく必要があります。
- ソフトウェアで計算した ECC および CRC 値はサポートされません。
- コントローラーをインスタンス化したデザインのシミュレーションはサポートされます。ただしシミュレーションではコントローラーの動作は観察できません。コントローラーを含むデザインのシミュレーションはコンパイルできますが、コントローラーは初期化ステートから遷移しません。コントローラーの動作はハードウェアで評価する必要があります。
- ビットストリーム セキュリティの手段として、コントローラーは、ビットストリーム暗号化をサポートしていません。ただしビットストリーム認証はサポートしていません。
- コントローラーは SelectMAP の persist をサポートしていません。
- 置換による訂正のためにコンフィギュレーション データのストレージが必要な場合、このデータをフェッチ インターフェイスから (通常は EXT シム経由で) コントローラーに供給する必要があります。これによりコントローラーと FPGA コンフィギュレーション方法が分離され、ユーザーがコンフィギュレーション方法、コンフィギュレーション データ ストレージ、およびソフト エラー軽減ソリューションのデータ ストレージを柔軟に選択できるようになります。
- EXT シムのインプリメンテーションがサポートする SPI フラッシュ読み出しコマンドは、1 つの SPI フラッシュ デバイスに対する SPI モード 0 (CPOL = 0、CPHA = 0) の高速読み出しのみです。
- FPGA と標準 SPI フラッシュ デバイスの I/O 電圧に互換性がない場合には、SPI メモリ システムのデザインにレベル変換器が必要です。
- ICAP アービトレーションおよび ICAP スイッチオーバーはサポートされません。ICAP インスタンスは、プライマリ/最上位の物理位置に 1 つだけサポートされます。
- CAPTURE プリミティブおよび関連する機能を含め、デザイン キャプチャの使用はサポートされません。
- 7 シリーズ FPGA および Zynq-7000 SoC にインプリメントした場合、コントローラーはタイプ 0、タイプ 2、タイプ 3 のコンフィギュレーション フレームで発生したソフト エラーに対して動作します。コンフィギュレーション フレームのタイプの詳細は、『7 シリーズ FPGA コンフィギュレーション ユーザー ガイド』(UG470) [参照 1] を参照してください。

デザイン フローの手順

この章では、コアのカスタマイズと生成、制約、およびシミュレーション/合成/インプリメンテーションの手順について説明します。一般的な Vivado® デザイン フローおよび Vivado IP インテグレーターの詳細は、次の Vivado Design Suite ユーザー ガイドを参照してください。

- 『Vivado Design Suite ユーザー ガイド : IP インテグレーターを使用した IP サブシステムの設計』(UG994) [参照 5]
- 『Vivado Design Suite ユーザー ガイド : IP を使用した設計』(UG896) [参照 6]
- 『Vivado Design Suite ユーザー ガイド : 入門』(UG910) [参照 7]
- 『Vivado Design Suite ユーザー ガイド : ロジック シミュレーション』(UG900) [参照 8]

コアのカスタマイズおよび生成

ここでは、ザイリンクス ツールを使用し、Vivado Design Suite でコアをカスタマイズおよび生成する方法について説明します。コアをカスタマイズして生成するには、まず Vivado IP カタログで [FPGA Features and Design] → [Soft Error Mitigation] を展開し、その下にある [Soft Error Mitigation] をクリックして選択します。[Project Manager] ウィンドウの [Details] エリアにこのソリューションに関する重要な情報が表示されます。この内容を確認してから次の手順に進みます。

IP カタログでこの IP コアをダブルクリックすると、[Customize IP] ダイアログ ボックスが開きます (図 4-1)。

注記 : この章の図には Vivado 統合設計環境 (IDE) のスクリーンショットが使用されていますが、現在のバージョンとはレイアウトが異なる場合があります。

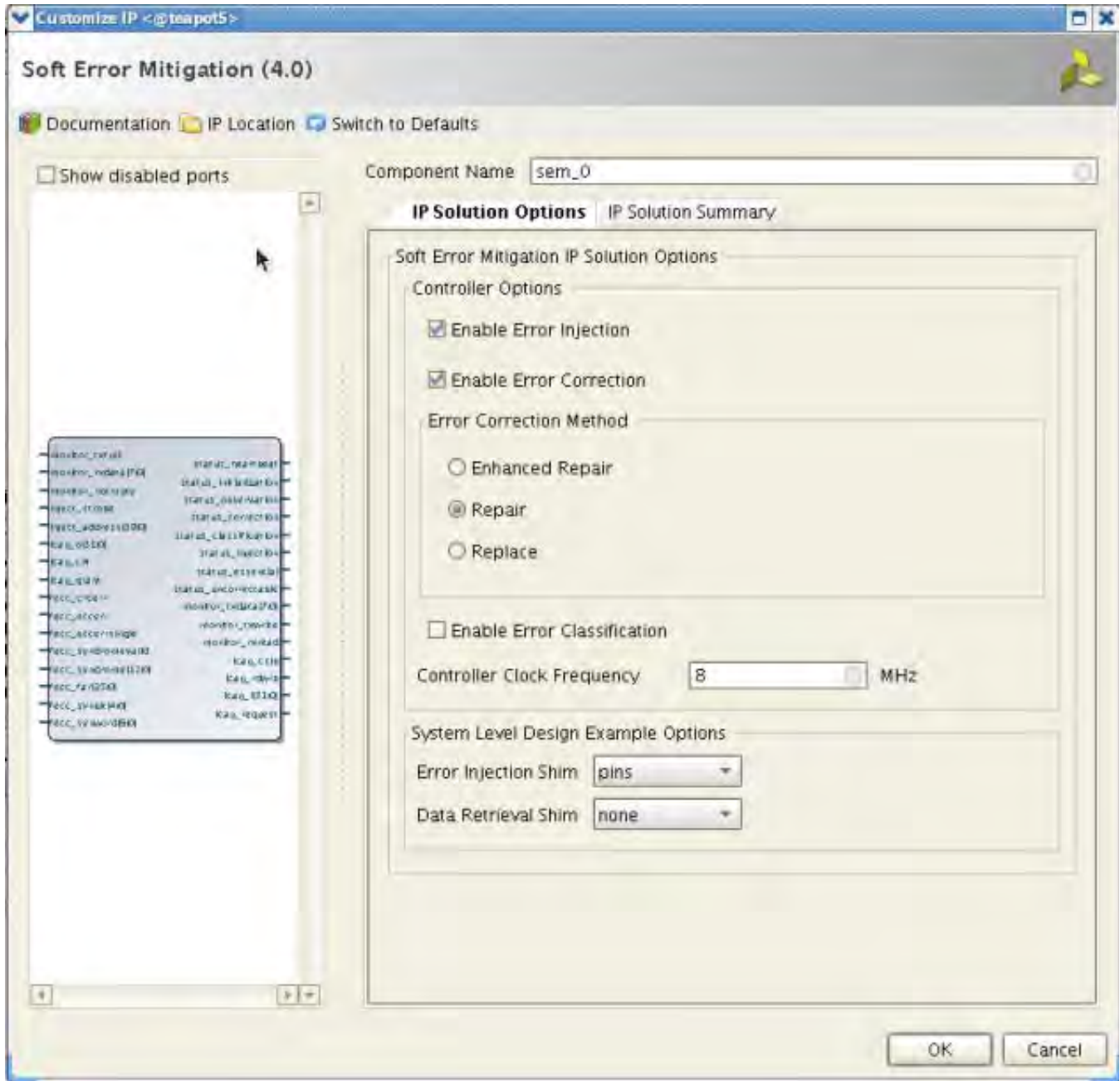


図 4-1 : [Customize IP] ダイアログ ボックスの [IP Solution Options] タブ

最終的に SEM Controller ソリューションが、組み込まれるプロジェクト全体の要件を満たすように、ここで各オプションの設定を確認し、必要に応じて変更します。ここからは、各オプションの詳細について説明します。

[Component Name] およびシンボル

[Component Name] では、生成されるコンポーネントの名前を設定します。この例では、「sem_0」を使用しています。

このダイアログ ボックスの左側にはコンポーネント シンボルが表示されます。ここには、現在のオプション設定でコンポーネントに存在するポートが視覚的に表示されます。オプション設定を変更すると、この図は自動的に更新されます。

[Controller Options] : [Enable Error Injection]

エラー挿入機能の有効/無効をこのチェック ボックスで設定します。エラー挿入はデザインを検証するための機能で、ソフト エラー イベントを模したエラーをユーザーがコンフィギュレーション メモリに発生させることができます。この機能は、統合またはシステム レベル テストの際にコントローラーがシステム レベルの監視ロジックと正しく接続されているか、そしてソフト エラー イベントが発生した場合にシステムが正しく応答するかを検証する際に使用します。

エラー挿入を有効にした場合、エラー挿入インターフェイスが生成され、コンポーネント シンボルに表示されます。コントローラーはユーザーのコマンドにตอบสนองしてエラー挿入を実行します。これらのコマンドは、エラー挿入インターフェイスまたはモニター インターフェイスから入力できます。

エラー挿入を無効にした場合、エラー挿入インターフェイスは削除され、コンポーネント シンボルからも削除されます。コントローラーはエラー挿入を実行しません。



ヒント : エラー挿入インターフェイスを削除してもモニター インターフェイスは存在します。エラー挿入コマンドをモニター インターフェイスに入力しても、コントローラーはコマンドを解析するだけで無視します。

[Controller Options] : [Enable Error Correction]

エラー訂正機能の有効/無効をこのチェック ボックスで設定します。この設定にかかわらず、コントローラーはエラー 検出回路を監視し、エラー条件を報告します。

エラー訂正を有効にした場合、コントローラーは検出したエラーの訂正を試みます。エラーの訂正方式は選択できます。ほとんどのエラーは訂正可能で、訂正に成功するとコントローラーは訂正可能なエラーが発生して訂正されたことを報告します。訂正不能なエラーの場合、コントローラーは訂正不能なエラーが発生したことを報告します。

エラー訂正を無効にした場合、コントローラーはエラーの訂正を試みません。エラー イベントを 1 回検出すると、コントローラーはそのエラー条件を報告した後でスキャンを停止します。また、エラー訂正を無効にした場合はエラー 分類機能も無効になります。

[Controller Options] : [Error Correction Method]

エラー訂正を有効にした場合、ここでエラー訂正方式を選択します。拡張修復による訂正 ([Enhanced Repair])、修復による訂正 ([Repair])、置換による訂正 ([Replace]) を選択できます。コントローラーは、ここで選択した方式でエラーを訂正します。各訂正方式の詳細は次のとおりです。

修復による訂正

- 1 ビットのエラーを修復によって訂正します。ECC シンドロームを使用してフレーム内の正確なエラー位置を特定します。エラーを含むフレームを読み出し、該当するビットを反転してフレームを書き戻します。このエラーは訂正可能エラーとして報告されます。

拡張修復による訂正

- 1 ビットおよび隣接する 2 ビットのエラーを修復によって訂正します。1 ビット エラーの場合の動作は修復による訂正と同じです。2 ビット エラーの場合は、隣接する 2 ビットのエラーを訂正可能な拡張 CRC ベースのアルゴリズムを使用します。エラーを含むフレームを読み出し、該当するビットを反転してフレームを書き戻します。このエラーは訂正可能エラーとして報告されます。

置換による訂正

- 1 ビットおよび複数ビットのエラーを置換によって訂正します。エラーを含むフレームを読み出します。コントローラーは、フェッチ インターフェイスに対してフレーム全体の置換データを要求します。置換データを取得すると、コントローラーは影響を受けたフレームと置換フレームを比較し、エラー位置を特定します。次に、置換データを書き戻します。このエラーは訂正可能エラーとして報告されます。

これら3つの訂正方式の違いは、修復または拡張修復による訂正では訂正できない数のビット エラーが発生した場合を考えると明らかです (実際にそのようなエラーが発生することはほとんどない)。このような場合、修復または拡張修復では正しく訂正できません。しかし置換によって訂正される場合、影響をうけたビット数にかかわらずこのエラーは訂正可能です。

基本的に、これらの訂正方式にはエラー訂正能力とデータ ストレージ要件に伴うコストのトレードオフがあります。修復による訂正と比較した場合、各方式には次の特徴があります。

- 拡張修復による訂正 : フレーム レベル CRC を格納するためのブロック RAM 容量が必要になりますが、エラー訂正能力は向上します。
- 置換による訂正 : 外部 SPI フラッシュにゴールデン フレーム置換データを格納しておくことで、最も高いエラー訂正能力が得られます。

置換によるエラー訂正には、EasyPath™ デバイスは使用できません。

[Controller Options] : [Enable Error Classification]

エラー分類機能の有効/無効をこのチェック ボックスで設定します。エラー訂正を無効にした場合、エラー分類は自動的に無効になります。

エラー分類機能は、検出および訂正されたソフト エラーがユーザー デザインの機能に影響したかどうかをザイリンクスのエッセンシャルビット テクノロジーを利用して判定します。エッセンシャルビットとは、デザインの回路に係るビットをいいます。エッセンシャルビットが変化すると、デザインの回路が変化します。しかし必ずしもデザインの機能に影響するとは限りません。

どのビットがエッセンシャルかが不明な場合、システムは検出したソフト エラーのすべてがデザインの正確さに影響したと見なす必要があります。多くの場合、システム レベルでエラー軽減を実行すると、FPGA コンフィギュレーションが修復されてデザインのリセットまたは再起動が完了するまでサービスが中断または低下します。

しかし Vivado Bitstream Generator のレポートでデザインの動作に影響するエッセンシャルビットがコンフィギュレーション メモリ全体の 20% であると報告された場合、システムレベルのエラー軽減が必要なのは 10 回のソフト エラーのうち平均 2 回しかありません。エラー分類機能は、ソフト エラー イベントがコンフィギュレーション メモリのエッセンシャルビット位置で発生したかどうかをルックアップ テーブルを利用して判定します。この機能を使用するとデザインの実効 FIT が改善します。ただしエラー分類を有効にすると、ルックアップ テーブルを格納するための外部ストレージが必要になります。エラー分類を有効にすると、コントローラーが外部データを取得するためのインターフェイスとしてフェッチ インターフェイスが生成され、コンポーネント シンボルにも表示されます。

エラー分類を有効にした場合、検出したエラーの訂正が完了すると、コントローラーはルックアップ テーブルを参照してエラー位置を特定します。このテーブルの情報に基づき、コントローラーはエラーがエッセンシャルか非エッセンシャルかを報告します。検出したエラーを訂正できないのは、エラー位置を特定できないためです。したがって、訂正不能なエラーはエッセンシャルかどうかをルックアップ テーブルで判定できないため、安全のためにすべてのエラーをエッセンシャルとして報告します。

同様に、エラー分類を無効にした場合もコントローラーはエラーがエッセンシャルかどうかを判定できないため、すべてのエラーを無条件にエッセンシャルとして報告します。



ヒント : エラー分類は必ずしもコントローラーで実行する必要はありません。コントローラーによるエラー分類を無効にし、インプリメンテーション ツールによって提供されるエッセンシャルビット データおよびコントローラーのモニター インターフェイスから出力されるエラー レポート メッセージを使用して、システム内の別のブロックでエラー分類を実行させることもできます。

[Controller Options] : [Controller Clock Frequency]

コントローラーのクロック周波数を設定します。コントローラーのクロック周波数を引き上げると、エラー軽減時間が短縮されます。したがって、なるべく高い周波数を設定するのが理想です。目的の周波数がターゲット デバイスの能力を超える場合、警告が表示されます。

エラー分類または置換によるエラー訂正のために外部データをフェッチするデータ取得インターフェイスが必要なデザインでは、さらに別の考慮事項が存在します。サンプル デザインはコントローラーに同期した外部メモリ インターフェイスをインプリメントします。このため、外部メモリのサイクル時間もコントローラーのクロック周波数によって決まります。外部メモリの最小サイクル時間がコントローラーの最大クロック周波数を制限する要因になるため、外部メモリ システムを解析してこの最小サイクル時間を決定する必要があります。

この解析の実行方法は、第 3 章の「インターフェイス」で説明しています。ただしこの解析には、インプリメンテーション結果から得たタイミング データが必要です。したがって、ザイリンクスでは次の手順を推奨しています。

1. 目的の周波数設定でソリューションを生成する。
2. インプリメンテーション結果から必要なタイミング データを抽出する。
3. タイミング バジレット解析を実行して最大周波数を決定する。
4. 計算で求めた最大動作周波数と同じかそれ以下の周波数でソリューションを再生成する。

[System Level Design Example Options] : [Error Injection Shim]

カスタマイズでエラー挿入のオプションを有効にした場合、サンプル デザインのシムにはエラー挿入インターフェイスの外部制御に関するオプションが 2 つあります。

- 物理ピンで直接制御する
- Vivado Design Suite のデバッグ機能を使用して JTAG 経由で間接的に制御する

Vivado ラボ ツールを使用してエラー挿入インターフェイスを制御するように設定した場合、SEM サンプル デザインの一部として VIO LogiCORE IP が生成されます。



[System Level Design Example Options] : [Data Retrieval Shim]

エラー分類または置換によるエラー訂正のオプションを有効にした場合、外部データをフェッチするためのデータ取得インターフェイスが必要で、そのためにはコントローラーのフェッチ インターフェイスを外部ストレージ デバイスに接続するブリッジが必要です。このサンプル デザインには、外部 SPI フラッシュ デバイスに接続するシムが唯一のオプションとして用意されています。データ取得インターフェイスが不要な場合、シムは生成されません。

[IP Solution Summary]

次に、[IP Solution Summary] タブをクリックします (図 4-2)。

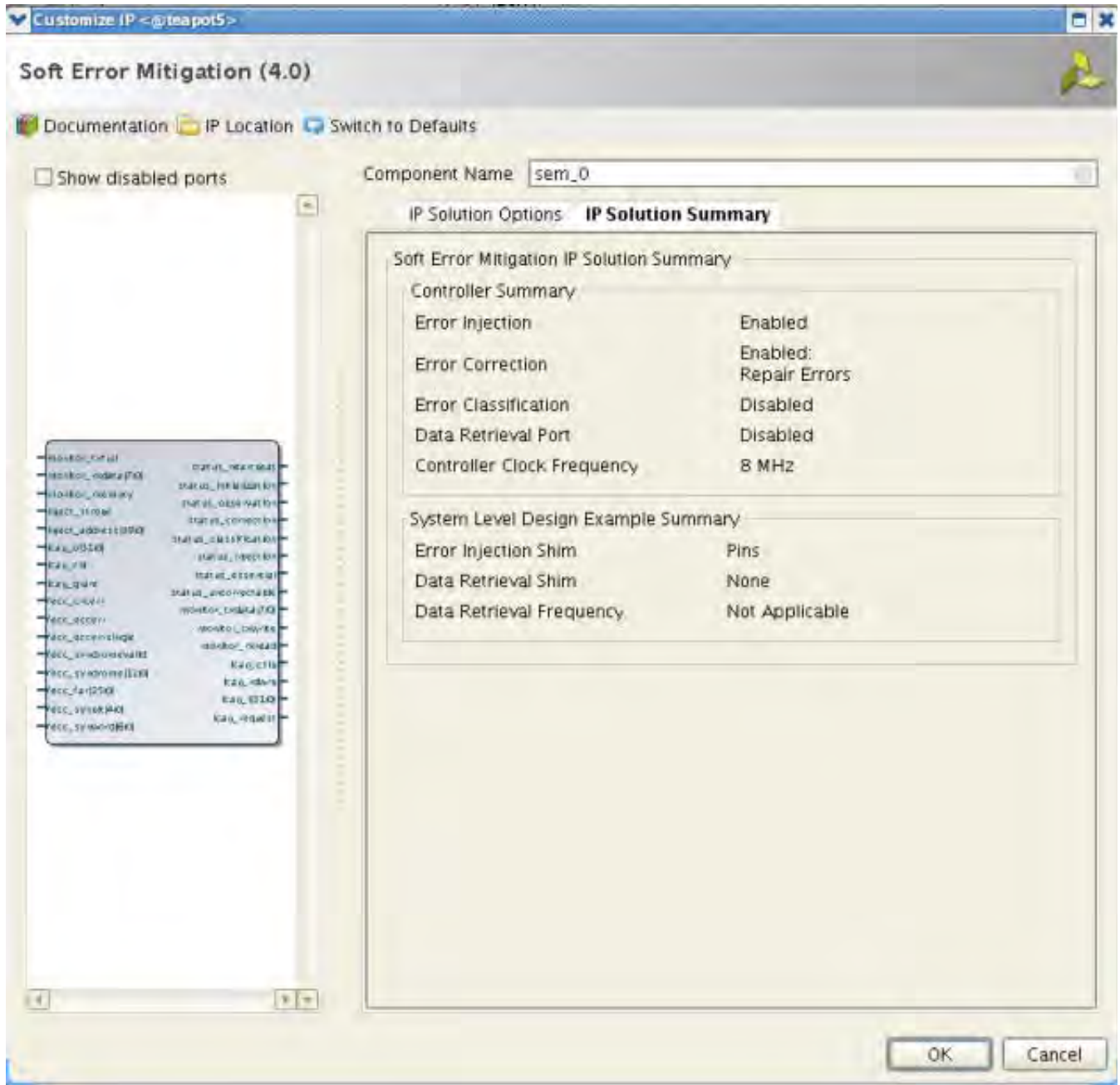


図 4-2 : [Customize IP] ダイアログ ボックスの [IP Solution Summary] タブ

各オプションの設定が正しいことを確認します。設定が正しくない場合は、[IP Solution Options] タブに戻って設定を変更します。すべてのオプション設定が正しいことを確認したら、[OK] をクリックして IP のカスタマイズを完了します。

ユーザー パラメーター

表 4-1 に、Vivado IDE のフィールドとユーザー パラメーターの対応関係を示します。ユーザー パラメーターは Tcl コンソールで表示できます。

表 4-1: Vivado IDE のパラメーターとユーザー パラメーターの対応

Vivado IDE のパラメーター	ユーザー パラメーター	デフォルト値
clock_freq	c_clock_per	8MHz
enable_injection	c_feature_set	TRUE
injection_shim	c_feature_set、 c_hardware_cfg	pins
enable_correction	c_feature_set	TRUE
correction_method	c_feature_set	repair
enable_classification	c_feature_set	FALSE
retrieval_shim	N/A	none

出力の生成

SEM IP ソリューションのファイルは、いくつかのファイル グループとして生成されます。生成されたファイル グループは、[Sources] ウィンドウの [IP Sources] ビューでプロジェクトの IP ごとに一覧で表示されます。SEM コアには次のファイル グループがあります。

- **Examples** : IP のサンプル デザイン プロジェクトを開いてインプリメントするのに必要なすべてのソースが含まれます (サンプル デザインの HDL ファイルと XDC ファイル)。
- **Synthesis** : コアが必要とするすべての合成ソースが含まれます。SEM コアの場合、暗号化されたソースと暗号化されていないソースの両方が存在します。暗号化されていないソースのみ表示できます。
- **Instantiation Template** : サンプルのインスタンス化テンプレートです。

コアへの制約

このセクションでは、このコアに適用される制約について説明します。

必須の制約

SEM Controller とシステム レベル サンプル デザインがパフォーマンス要件を満たした機能結果を得るには、物理的なインプリメンテーション制約を指定する必要があります。これらの制約は、システム レベル サンプル デザイン生成時に XDC ファイルとして作成されます。この XDC ファイル (<component name>_sem_example.xdc) は、[Sources] ウィンドウの [IP Sources] ビューで Examples ファイルグループにあります。

一貫したインプリメンテーション結果を得るには、ソリューションに付属の XDC ファイルを使用する必要があります。XDC または特定の制約の定義および使用法については、[Vivado Design Suite の資料ページ](#)にある『制約ガイド』を参照してください。

このソリューションをより大規模なプロジェクトに統合する場合、またはシステム レベル サンプル デザインに変更を加えた場合、制約の変更が必要となることがあります。制約の変更は、各制約を十分に理解したうえで行ってください。また、提供されている制約を規定から逸脱した形で使用したデザインに対してはサポートを提供していません。

XDC (ザイリンクス デザイン制約) ファイルの内容

XDC ファイルはソリューションを生成するたびに作成されます。XDC ファイルの全体的な構造と制約シーケンスは毎回同じですが、その内容は生成時に設定したオプションにより異なります。ここからは、Kintex™-7 デバイスへのインプリメンテーションを例に、XDC ファイルの構造と制約シーケンスについて説明します。

コントローラーの制約

コントローラーは、単体で考えた場合、生成時のオプション設定にかかわらず完全同期デザインです。基本的に、必要な制約はマスター クロック入力に対するクロック周期制約のみです。ジェネリック XDC では、この制約はシステム レベル サンプル デザイン クロック入力に適用され、コントローラーに伝搬します。この制約については、「[サンプル デザインの制約](#)」で説明します。

コントローラーと FPGA コンフィギュレーション システム プリミティブの間の信号パスは同期パスと見なす必要があります。デフォルトでは、ICAP (内部コンフィギュレーション アクセス ポート) プリミティブとコントローラーの間のパスはマスター クロックに対するクロック周期制約の一部として解析されます。これは、ICAP クロック ピンを同じマスター クロック信号に接続する必要があるためです。

しかし FRAME_ECC プリミティブにはクロック ピンがないため状況異なります。FRAME_ECC プリミティブと ICAP プリミティブを組み合わせて使用する場合、FRAME_ECC プリミティブのピンはマスター クロック信号に同期することが知られています。したがって、クロック周期制約から求めた値による制約を次のように追加します。

```
set_max_delay 12.151 -from [get_pins example_cfg/example_frame_ecc/*] -quiet -datapath_only
set_max_delay 30.302 -from [get_pins example_cfg/example_frame_ecc/*] -to [all_outputs]
-quiet -datapath_only
```

サンプル デザインの制約

サンプル デザインの制約は制約のタイプごとではなくインターフェイスごとにまとめられています。最初のグループは、デザイン全体に対するマスター クロック入力への制約です。ここでは IOSTANDARD 制約と PERIOD 制約を適用します。PERIOD 制約の値は、生成時に設定したオプションにより異なります。

```
create_clock -name clk -period 15.151 [get_ports clk]
set_property IOSTANDARD LVCMOS25 [get_ports clk]
```

2 番目のグループはステータス インターフェイスへの制約で、IOSTANDARD 制約と出力タイミング制約を適用します。出力タイミング制約は PERIOD 制約の 2 倍に設定します。

```
set_property DRIVE 8 [get_ports status_initialization]
set_property SLEW FAST [get_ports status_initialization]
set_property IOSTANDARD LVCMOS25 [get_ports status_initialization]

set_property DRIVE 8 [get_ports status_observation]
set_property SLEW FAST [get_ports status_observation]
set_property IOSTANDARD LVCMOS25 [get_ports status_observation]

set_property DRIVE 8 [get_ports status_correction]
set_property SLEW FAST [get_ports status_correction]
set_property IOSTANDARD LVCMOS25 [get_ports status_correction]

set_property DRIVE 8 [get_ports status_classification]
set_property SLEW FAST [get_ports status_classification]
set_property IOSTANDARD LVCMOS25 [get_ports status_classification]

set_property DRIVE 8 [get_ports status_injection]
set_property SLEW FAST [get_ports status_injection]
set_property IOSTANDARD LVCMOS25 [get_ports status_injection]
```

```

set_property DRIVE 8 [get_ports status_uncorrectable]
set_property SLEW FAST [get_ports status_uncorrectable]
set_property IOSTANDARD LVCMOS25 [get_ports status_uncorrectable]

set_property DRIVE 8 [get_ports status_essential]
set_property SLEW FAST [get_ports status_essential]
set_property IOSTANDARD LVCMOS25 [get_ports status_essential]

set_property DRIVE 8 [get_ports status_heartbeat]
set_property SLEW FAST [get_ports status_heartbeat]
set_property IOSTANDARD LVCMOS25 [get_ports status_heartbeat]

set_output_delay -clock clk -15.151 [get_ports [list status_observation
status_correction status_classification status_injection status_uncorrectable
status_essential status_heartbeat status_initialization]] -max
set_output_delay -clock clk 0 [get_ports [list status_observation status_correction
status_classification status_injection status_uncorrectable status_essential
status_heartbeat status_initialization]] -min

```

3 番目のグループは MON シムへの制約で、IOSTANDARD 制約と I/O タイミング制約を適用します。I/O タイミング制約は PERIOD 制約の 2 倍に設定します。

```

set_property DRIVE 8 [get_ports monitor_tx]
set_property SLEW FAST [get_ports monitor_tx]
set_property IOSTANDARD LVCMOS25 [get_ports monitor_tx]
set_property IOSTANDARD LVCMOS25 [get_ports monitor_rx]

set_input_delay -clock clk -max -15.151 [get_ports monitor_rx]
set_input_delay -clock clk -min 30.302 [get_ports monitor_rx]
set_output_delay -clock clk -15.151 [get_ports monitor_tx] -max
set_output_delay -clock clk 0 [get_ports monitor_tx] -min

```

次のグループは EXT シムへの制約で、生成時のオプション設定により EXT シムが生成される場合のみ存在します。ここでは IOSTANDARD 制約と I/O タイミング制約を適用します。SPI バス タイミング バジレットの解析では実際のタイミングを使用する必要があるため、I/O タイミングは非常に重要です。ただし FPGA インプリメンテーションの I/O タイミングに対するハード要件はありません。これらの要件は選択したデバイスおよびスピード グレードにより異なります。

このため、I/O タイミング制約は暫定的に PERIOD 制約の 2 倍に設定します。IOB フリップフロップを使用する制約を追加すると、制約の値が示すよりも I/O タイミングが大幅に改善します。SPI バス タイミング バジレットの解析では、制約の値ではなくタイミング レポートから取得した実際のタイミングを使用します。

```

set_property IOB TRUE [get_cells example_ext/example_ext_byte/ext_c_ofd]
set_property IOB TRUE [get_cells example_ext/example_ext_byte/ext_d_ofd]
set_property IOB TRUE [get_cells example_ext/example_ext_byte/ext_q_ifd]
set_property IOB TRUE [get_cells example_ext/example_ext_s_ofd]

set_property DRIVE 8 [get_ports external_c]
set_property SLEW FAST [get_ports external_c]
set_property IOSTANDARD LVCMOS25 [get_ports external_c]

set_property DRIVE 8 [get_ports external_d]
set_property SLEW FAST [get_ports external_d]
set_property IOSTANDARD LVCMOS25 [get_ports external_d]

set_property DRIVE 8 [get_ports external_s_n]
set_property SLEW FAST [get_ports external_s_n]
set_property IOSTANDARD LVCMOS25 [get_ports external_s_n]

set_property IOSTANDARD LVCMOS25 [get_ports external_q]

```

```
set_input_delay -clock clk -max -15.151 [get_ports external_q]
set_input_delay -clock clk -min 30.302 [get_ports external_q]
set_output_delay -clock clk -15.151 [get_ports [list external_d external_s_n
external_c]] -max
set_output_delay -clock clk 0 [get_ports [list external_d external_s_n external_c]]
-min
```

次のグループは HID シムへの制約で、HID シムが I/O ピンの場合のみ存在します。ここでは IOSTANDARD 制約と入力タイミング制約を適用します。入力タイミング制約は PERIOD 制約の 2 倍に設定します。

```
set_property IOSTANDARD LVCMOS25 [get_ports inject_strobe]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[0]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[1]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[2]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[3]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[4]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[5]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[6]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[7]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[8]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[9]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[10]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[11]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[12]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[13]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[14]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[15]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[16]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[17]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[18]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[19]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[20]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[21]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[22]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[23]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[24]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[25]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[26]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[27]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[28]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[29]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[30]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[31]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[32]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[33]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[34]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[35]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[36]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[37]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[38]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[39]}]

set_input_delay -clock clk -max -14.285 [get_ports [list {inject_address[0]}
{inject_address[1]} {inject_address[2]} {inject_address[3]} {inject_address[4]}
{inject_address[5]} {inject_address[6]} {inject_address[7]} {inject_address[8]}
{inject_address[9]} {inject_address[10]} {inject_address[11]} {inject_address[12]}
{inject_address[13]} {inject_address[14]} {inject_address[15]} {inject_address[16]}
{inject_address[17]} {inject_address[18]} {inject_address[19]} {inject_address[20]}]
```

```

{inject_address[21]} {inject_address[22]} {inject_address[23]} {inject_address[24]}
{inject_address[25]} {inject_address[26]} {inject_address[27]} {inject_address[28]}
{inject_address[29]} {inject_address[30]} {inject_address[31]} {inject_address[32]}
{inject_address[33]} {inject_address[34]} {inject_address[35]} {inject_address[36]}
{inject_address[37]} {inject_address[38]} {inject_address[39]} inject_strobe]]
set_input_delay -clock clk -min 28.57 [get_ports [list {inject_address[0]}
{inject_address[1]} {inject_address[2]} {inject_address[3]} {inject_address[4]}
{inject_address[5]} {inject_address[6]} {inject_address[7]} {inject_address[8]}
{inject_address[9]} {inject_address[10]} {inject_address[11]} {inject_address[12]}
{inject_address[13]} {inject_address[14]} {inject_address[15]} {inject_address[16]}
{inject_address[17]} {inject_address[18]} {inject_address[19]} {inject_address[20]}
{inject_address[21]} {inject_address[22]} {inject_address[23]} {inject_address[24]}
{inject_address[25]} {inject_address[26]} {inject_address[27]} {inject_address[28]}
{inject_address[29]} {inject_address[30]} {inject_address[31]} {inject_address[32]}
{inject_address[33]} {inject_address[34]} {inject_address[35]} {inject_address[36]}
{inject_address[37]} {inject_address[38]} {inject_address[39]} inject_strobe]]

```

次のグループは、システム レベル サンプル デザインの一部を選択したデバイスの特定領域に配置する **pblock** を実装します。**pblock** に含まれるインスタンスは、生成時に設定したオプションにより異なります。範囲の値は、使用するデバイスにより異なります。

pblock により、ソフト エラー軽減ロジックはデバイスの ICAP サイトに物理的に隣接した領域にパックして配置されます。特に重要な点として、これによりタイミング結果の再現性が維持されます。また、**pblock** はよりコンパクトなパックが可能ためリソース使用量が抑えられます。リソース使用量のサマリも生成されるため、システム レベル サンプル デザインの FIT を推定するのに役立ちます。

```

create_pblock SEM_CONTROLLER
resize_pblock -pblock SEM_CONTROLLER -add RAMB18_X2Y50:RAMB18_X4Y59
resize_pblock -pblock SEM_CONTROLLER -add RAMB36_X2Y25:RAMB36_X4Y29
resize_pblock -pblock SEM_CONTROLLER -add SLICE_X36Y125:SLICE_X47Y149
add_cells_to_pblock -pblock SEM_CONTROLLER -cells [get_cells example_controller/*]
add_cells_to_pblock -pblock SEM_CONTROLLER -cells [get_cells example_mon/*]
add_cells_to_pblock -pblock SEM_CONTROLLER -cells [get_cells example_ext/*]

set_property LOC FRAME_ECC_X0Y0 [get_cells example_cfg/example_frame_ecc]
set_property LOC ICAP_X0Y1 [get_cells example_cfg/example_icap]

```

次のグループは、I/O ピン ロケーションをシステム レベル サンプル デザインの最上位ポートに割り当てるためのテンプレートです。これらの割り当てはボードごとに異なるため、自動的には生成できません。これらの制約を適用するには、コメント記号を削除してターゲット ボードに応じた適切な I/O ピン ロケーションを指定してください。

```

## set_property LOC <pin loc> [get_ports clk]

## set_property LOC <pin loc> [get_ports status_initialization]
## set_property LOC <pin loc> [get_ports status_observation]
## set_property LOC <pin loc> [get_ports status_correction]
## set_property LOC <pin loc> [get_ports status_classification]
## set_property LOC <pin loc> [get_ports status_injection]
## set_property LOC <pin loc> [get_ports status_uncorrectable]
## set_property LOC <pin loc> [get_ports status_essential]
## set_property LOC <pin loc> [get_ports status_heartbeat]

## set_property LOC <pin loc> [get_ports monitor_tx]
## set_property LOC <pin loc> [get_ports monitor_rx]

## set_property LOC <pin loc> [get_ports external_c]
## set_property LOC <pin loc> [get_ports external_d]
## set_property LOC <pin loc> [get_ports external_q]
## set_property LOC <pin loc> [get_ports external_s_n]

```

```

## set_property LOC <pin loc> [get_ports inject_strobe]
## set_property LOC <pin loc> [get_ports {inject_address[0]}]
## set_property LOC <pin loc> [get_ports {inject_address[1]}]
## set_property LOC <pin loc> [get_ports {inject_address[2]}]
## set_property LOC <pin loc> [get_ports {inject_address[3]}]
## set_property LOC <pin loc> [get_ports {inject_address[4]}]
## set_property LOC <pin loc> [get_ports {inject_address[5]}]
## set_property LOC <pin loc> [get_ports {inject_address[6]}]
## set_property LOC <pin loc> [get_ports {inject_address[7]}]
## set_property LOC <pin loc> [get_ports {inject_address[8]}]
## set_property LOC <pin loc> [get_ports {inject_address[9]}]
## set_property LOC <pin loc> [get_ports {inject_address[10]}]
## set_property LOC <pin loc> [get_ports {inject_address[11]}]
## set_property LOC <pin loc> [get_ports {inject_address[12]}]
## set_property LOC <pin loc> [get_ports {inject_address[13]}]
## set_property LOC <pin loc> [get_ports {inject_address[14]}]
## set_property LOC <pin loc> [get_ports {inject_address[15]}]
## set_property LOC <pin loc> [get_ports {inject_address[16]}]
## set_property LOC <pin loc> [get_ports {inject_address[17]}]
## set_property LOC <pin loc> [get_ports {inject_address[18]}]
## set_property LOC <pin loc> [get_ports {inject_address[19]}]
## set_property LOC <pin loc> [get_ports {inject_address[20]}]
## set_property LOC <pin loc> [get_ports {inject_address[21]}]
## set_property LOC <pin loc> [get_ports {inject_address[22]}]
## set_property LOC <pin loc> [get_ports {inject_address[23]}]
## set_property LOC <pin loc> [get_ports {inject_address[24]}]
## set_property LOC <pin loc> [get_ports {inject_address[25]}]
## set_property LOC <pin loc> [get_ports {inject_address[26]}]
## set_property LOC <pin loc> [get_ports {inject_address[27]}]
## set_property LOC <pin loc> [get_ports {inject_address[28]}]
## set_property LOC <pin loc> [get_ports {inject_address[29]}]
## set_property LOC <pin loc> [get_ports {inject_address[30]}]
## set_property LOC <pin loc> [get_ports {inject_address[31]}]
## set_property LOC <pin loc> [get_ports {inject_address[32]}]
## set_property LOC <pin loc> [get_ports {inject_address[33]}]
## set_property LOC <pin loc> [get_ports {inject_address[34]}]
## set_property LOC <pin loc> [get_ports {inject_address[35]}]
## set_property LOC <pin loc> [get_ports {inject_address[36]}]
## set_property LOC <pin loc> [get_ports {inject_address[37]}]
## set_property LOC <pin loc> [get_ports {inject_address[38]}]
## set_property LOC <pin loc> [get_ports {inject_address[39]}]
    
```

SSI デバイスの制約

システムレベル サンプル デザインでは、デバイスによって 2～4 個のコントローラー インスタンスが生成されます。これらのコントローラー インスタンスには、0～3 の ID 番号を含む名前が付けられます。コントローラー インスタンスの番号は、ハードウェア SLR (Super Logic Region) 番号と同じです。

各コントローラーは、各 SLR の関連するコンフィギュレーション ロジック プリミティブの近くに集中して配置されるようにエリア制約が適用されます。コンフィギュレーション ロジック プリミティブにも配置制約があります。これは非 SSI デバイスの制約とほぼ同じもので、これらをすべての SLR に繰り返し適用します。

HID シム、MON シム、EXT シムなどの共有ブロックも、マスター SLR に配置されるようにエリア制約が適用されず。マスター SLR はデバイスの中央に位置し、HID シムが使用する BSCAN プリミティブもマスター SLR にあります。

Vivado でのエッセンシャルビット生成の有効化

エラー分類または置換によるエラー訂正を有効にした場合、関連ファイルを生成するようにビットストリームプロパティを設定してからビットストリームを生成する必要があります。このプロパティは、XDC ファイルまたは Vivado の Tcl コンソールで設定します。

```
set_property bitstream.seu.essentialbits yes [current_design]
```

デバイス、パッケージ、スピード グレード

デバイス、パッケージ、スピード グレードに関する特別な制約はありません。

クロック周波数

クロック周波数に関する特別な制約はありません。

クロック管理

クロック管理に関する特別な制約はありません。

クロック配置

クロック配置に関する特別な制約はありません。

I/O ピン

I/O ピンに関する制約は、このセクションで説明したとおりです。

I/O 規格

I/O 規格に関する特別な制約はありません。

I/O バンク

I/O バンクに関する特別な制約はありません。

I/O 配置

I/O 配置に関する特別な制約はありません。

シミュレーション

コントローラーをインスタンス化してデザインしたシミュレーションはサポートされます。つまり、コントローラーを大規模なプロジェクトに統合した場合、コントローラーに関係しない機能のシミュレーションは通常どおり実行できます。ただしシミュレーションではコントローラーの動作は観察できません。コントローラーを含むデザインのシミュレーションはコンパイルできますが、コントローラーは初期化ステートから遷移しません。

コントローラーの動作はハードウェアで評価する必要があります。

Vivado シミュレーション コンポーネントについて、またサポートされているサードパーティ ツールについては、『Vivado Design Suite ユーザー ガイド：ロジック シミュレーション』(UG900) [参照 8] を参照してください。



重要：7 シリーズまたは Zynq-7000 デバイスをターゲットにしたコアでは、UNIFAST ライブラリはサポートされません。ザイリンクスの IP は UNISIM ライブラリでのみテストと認定が行われています。

合成およびインプリメンテーション

SEM コアは付属のサンプル デザインと一緒に合成およびインプリメントしてください。詳細は、第5章の「インプリメンテーション」を参照してください。

合成およびインプリメンテーションの詳細は、『Vivado Design Suite ユーザー ガイド：IP を使用した設計』(UG896) [参照 6] を参照してください。

統合およびバリデーション

複雑なシステムを開発する場合のベスト プラクティスとして、主要なファンクションブロックとインターフェイスをなるべく早い段階で統合し、継続的にバリデーションを実行することが重要です。開発サイクルの終盤になってデザインを大幅に変更すること（統合の遅れ）や、システムパフォーマンスの計測を先延ばしすること（バリデーションの遅れ）は、リスクを増大させます。

統合とバリデーションの主要なスケジュールはプロジェクトのプランニング段階で決定しておくこと、そして開発サイクル全体を通じてシステムの機能とパフォーマンスを計画的にチェックしてこれらのスケジュールをサポートすることが重要です。できるだけ早期に統合を開始し、可能な限り多くの機能を段階的に追加していくと、代表的なワークロードでのシステム評価の時間を最大限に確保できます。この手法は、デザイン再利用と IP ベース設計を促進する一般的なボトムアップ設計アプローチを補完するものとして推奨されます。⁽¹⁾

SEM IP コアの統合

SEM IP コアはプログラマブル ロジックのフットプリントをわずかしか占有しませんが、プログラマブル ロジックのコンフィギュレーション メモリ システムを活性化します。コンフィギュレーション メモリ システムの動作は通常の SRAM とほとんど同じですが、プログラマブル ロジック アレイ全体に物理的に分散している点が異なります。デバイス内のほかのデジタル スイッチング アクティビティ同様、コンフィギュレーション メモリ システムのアクティビティも電力ノイズを発生させます。

1. 『Comprehensive Full-Chip Methodology to Verify EM and Dynamic Voltage Drop on High Performance FPGA Designs in the 20nm Technology』(DesignCon 2014 でのプレゼンテーション) [参照 9]

SEM IP コアがサポートするデバイス ファミリでは、インプリメンテーション要件と設計ガイダンスを守っていれば、コンフィギュレーション メモリ システムから発生する電力ノイズはわずかです。



推奨 : SEM IP コアをシステムに統合する際は、最新バージョンのコアを使用するとともに、[SEM IP ソリューションに関するマスターアンサー](#)を参照して SEM IP コアに関するデザインアドバイザリや既知の問題など最新の情報を入手することを強く推奨します。

SEM IP コアはなるべく早い段階で (理想的にはプロジェクトの最初から) 統合してください。SEM IP コアは自動的に初期化できるため、最初の統合はコアをインスタンス化してクロックを接続し、ほかの入力ポートとは未接続にするだけで完了します。この早期インフラストラクチャの一部として SEM IP コアのプロビジョニング制御を実装し、システムが SEM IP コア クロックおよび SEM IP コアの `icap_grant` のイネーブル/ディスエーブルを行えるようにすることを推奨します。SEM IP コアのシステムプロビジョニングにより導入の柔軟性が向上すると同時に、SEM IP コアの統合のデバッグも容易になります。

その後、システムと SEM IP コアの間でコマンド/ステータス情報を交換するためのインターフェイスを定義および実装して統合を拡張できます。このインターフェイスには、SEM IP コアのモニター インターフェイスを利用した ASCII 通信を推奨します。UART ヘルパー ブロックはシリアル接続の場合のみ使用し、通信 FIFO を用いたパラレル接続の場合には使用しません。

SEM IP コアによって報告されたイベントのログをシステムで記録して解析するだけならステータス情報を交換するだけで十分ですが、エラー挿入をサポートするにはコマンドを交換する必要があります。放射線試験施設での加速粒子テスト以外では、SEM IP コアの統合および SEM IP コアのイベント レポートに対するシステム応答を完全にテストできる現実的な方法はエラー挿入しかありません。エラー挿入は、継続的なバリデーションにおいても一部の用途で役立ちます。

SEM IP コアのバリデーション

SEM IP コアを統合したシステムでは、SEM IP コアの初期化ステート中のコンフィギュレーション メモリ システムの動作は読み出しと書き込みが混在しています。その後の監視ステートでは、SEU を検出するためにコンフィギュレーション メモリ を常時スキャンするため、読み出し動作が 100% となります。地上環境におけるザイリンクスのコンフィギュレーション メモリの反転率は非常に低く (たとえばニューヨーク市の海拔 0m 地点では平均反転間隔は数十年)、SEM IP コアが監視ステートから訂正ステート (書き込み動作が発生するステート) に遷移することはほとんどありません。

SEM IP コアを統合したシステムの継続的なバリデーションでは、SEM IP コアの代表的なワークロードを使用して実環境のシステムに即したバリデーション結果が得られるようにする必要があります。システム システム プロビジョニングによって SEM IP コアが初期化を完了して監視ステートに移行する場合、デフォルトのワークロードではコンフィギュレーション メモリ をスキャンしてもソフト エラーはありません。これは作業が簡単であるだけでなく、バリデーションに適した代表的なワークロードともいえます。

必要であれば、SEM IP コアのエラー挿入機能を使用して散発的にエラー検出および訂正をワークロードに組み入れることもできます。



重要 : 頻繁にエラーを挿入して大量のエラー検出および訂正イベントを発生させるような「ストレステスト」は推奨しません。このようなスティミュラスは現実には即しておらず、システムの動作信頼性とは無関係なバリデーション結果となるためです。

システムバリデーションは研究開発段階だけでなく量産テストでも継続的に実施してください。SEM IP コアの場合、量産テストでのシステムプロビジョニングは SEM IP コアを有効にするだけでよいいため、作業負担の増加はほとんどまたはまったくありません。

サンプル デザイン

この章では、SEM Controller のシステム レベル サンプル デザインの概要と、そのインターフェイスについて説明します。システム レベル サンプル デザインはコントローラーと各種シムをカプセル化しており、これらのシムがコントローラーとほかのデバイスを接続するインターフェイスとしての役割を果たします。シムには、I/O ピン、VIO LogiCORE IP、I/O インターフェイス、メモリ コントローラー、アプリケーション固有のシステム管理インターフェイスなどがあります。

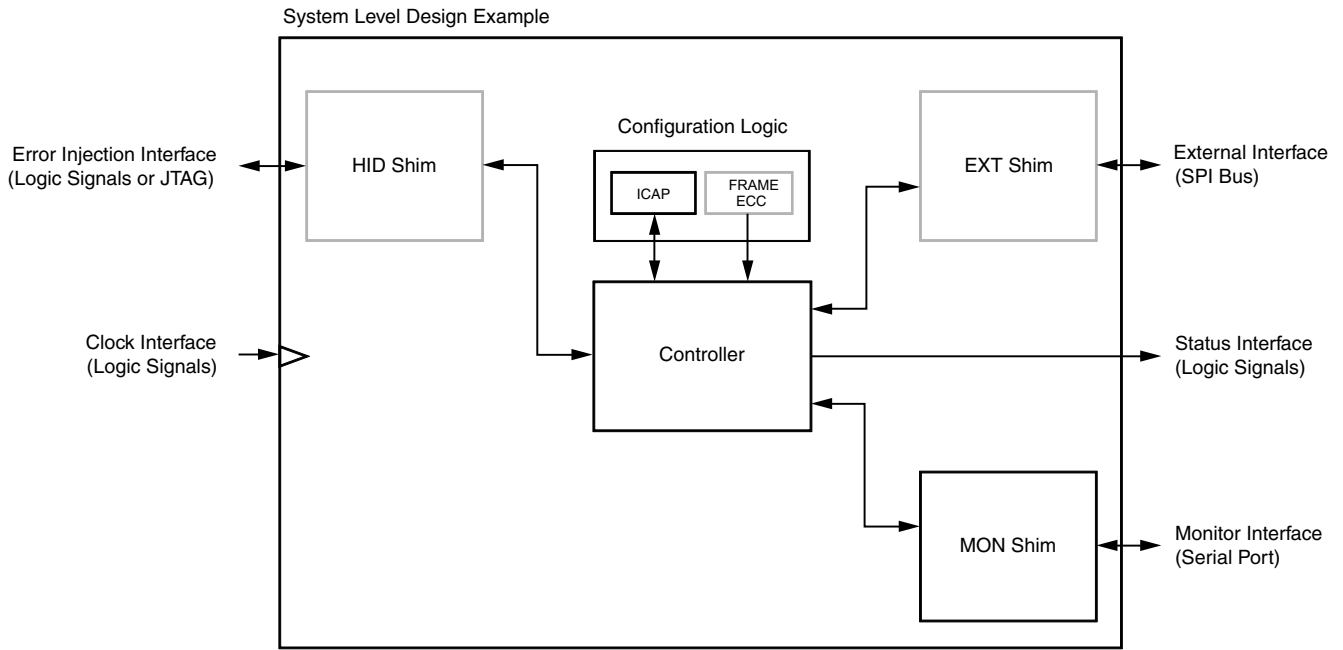
システム レベル サンプル デザインはコントローラーと一緒に検証されています。付属するシステム レベル サンプル デザインはリファレンス デザインではありませんが、ソリューション全体を構成する必須要素です。システム レベル サンプル デザインはユーザーが自由に変更できますが、そのまま使用することを推奨します。

機能

システム レベル サンプル デザインはコントローラー自体のインスタンスシート手段として利用できる以外に、主に次の 4 つの機能があります。

- **FPGA** コンフィギュレーション システムのプリミティブおよびこれらとコントローラーとの接続。
- **MON** シム : コントローラーと標準 **RS-232** ポートを接続するブリッジの役割を果たします。このシムが提供するインターフェイスを使用して、コントローラーとの間でコマンドとステータス情報を交換します。このインターフェイスは、プロセッサと接続するように設計されています。
- **EXT** シム : コントローラーと標準 **SPI** バスを接続するブリッジの役割を果たします。このシムが提供するインターフェイスを使用してコントローラーは外部からデータをフェッチします。このシムは標準 **SPI** フラッシュに接続するためのもので、コントローラーの設定によっては存在しないこともあります。
- **HID** シム : コントローラーとインターフェイス デバイスを接続するブリッジの役割を果たします。このシムが提供するインターフェイスを使用して、コントローラーとの間でコマンドとステータス情報を交換します。このシムは、コントローラーの設定によっては存在しないこともあります。

図 5-1 に、非 SSI (スタックド シリコン インターコネクト) デバイスのシステム レベルのサンプル デザインのブロック図を示します。グレーで示したブロックは、一部のコンフィギュレーションにのみ存在します。



X12177

図 5-1: サンプル デザインのブロック図 (非 SSI デバイスの場合)

図 5-2 に、SSI デバイスのシステム レベル サンプル デザインのブロック図を示します。グレーで示したブロックは、一部のコンフィギュレーションにのみ存在します。

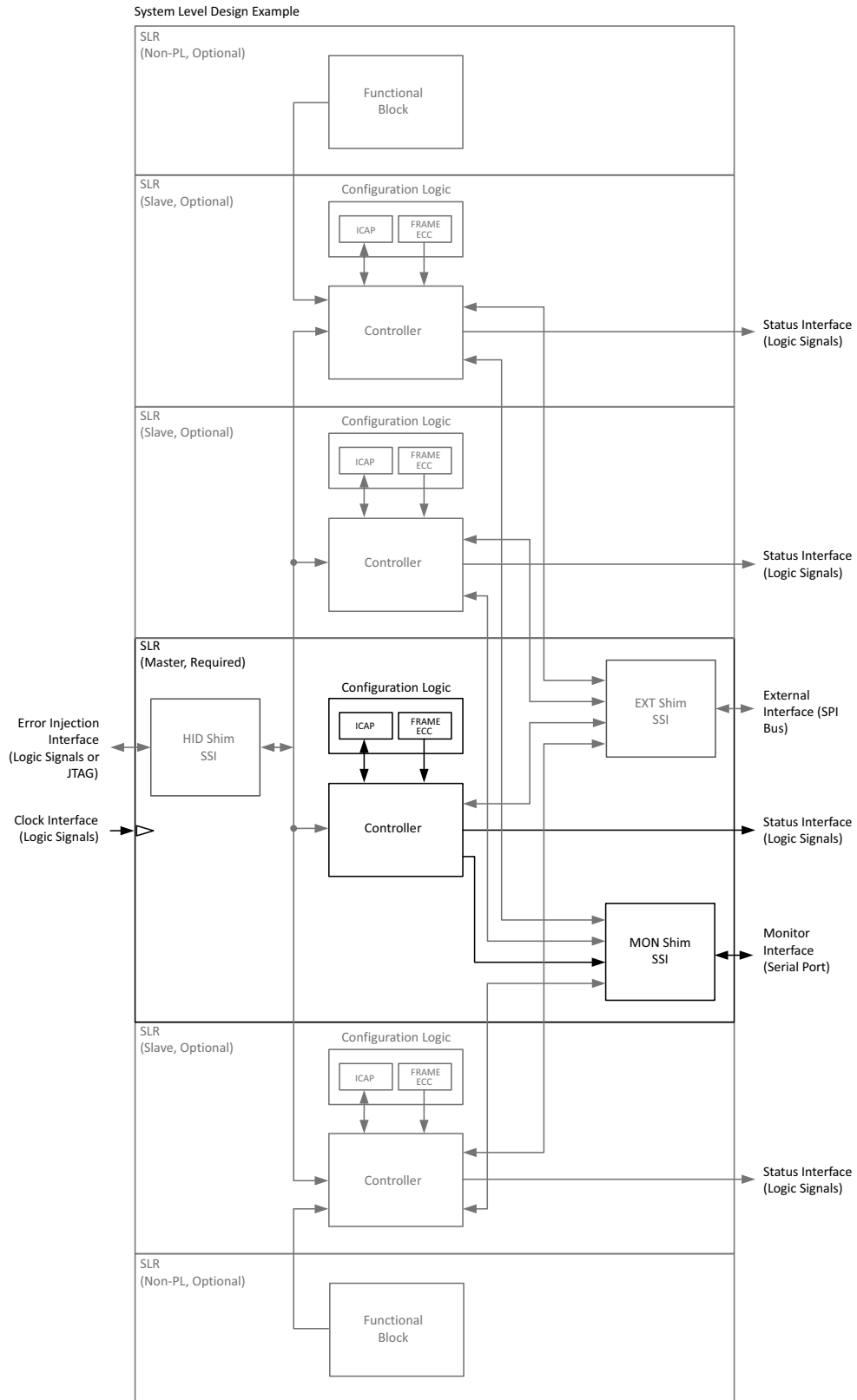
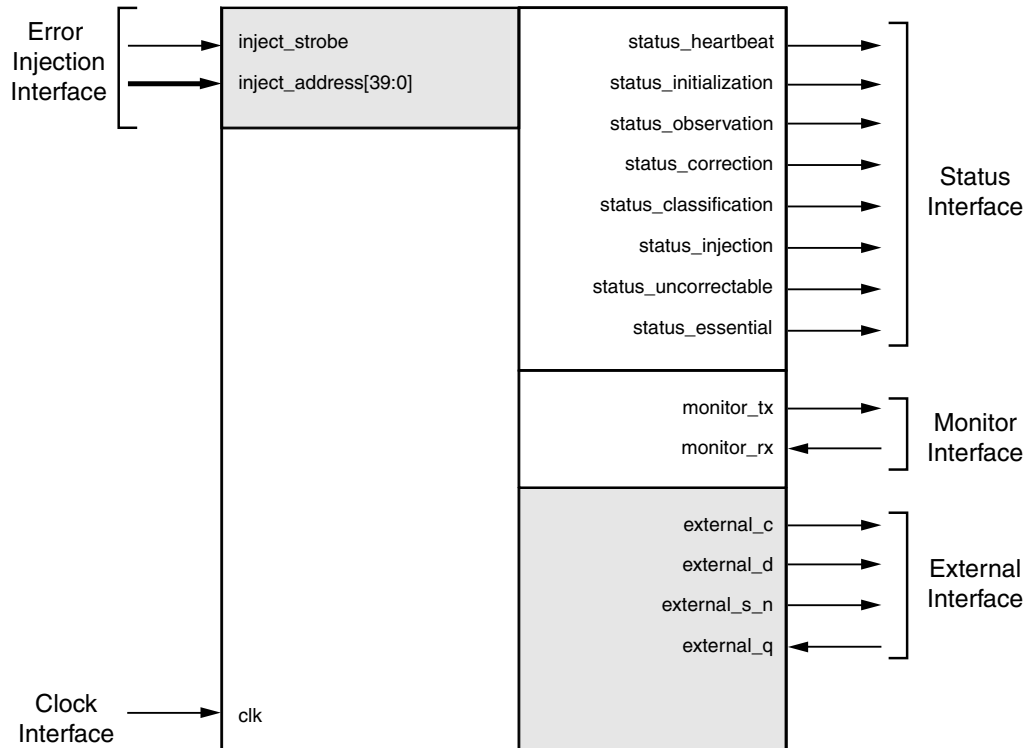


図 5-2 : サンプル デザインのブロック図 (SSI デバイスの場合)

システム レベル サンプル デザインは、システム レベルのインターフェイスに柔軟性をもたせるために提供しています。このため、コントローラーとは異なりシステム レベル サンプル デザインは RTL ソースとして提供されています。

ポートの説明

図 5-3 に、非 SSI デバイスの場合のサンプル デザインのポートを示します。ポートは 5 つのグループにまとめられています。グレーで示したグループは、一部のコンフィギュレーションにのみ存在します。



X12176

図 5-3 : サンプル デザインのポート (非 SSI デバイスの場合)

SSI デバイスでは、各 SLR (Super Logic Region) に番号が付けられます。これには、ハードウェア SLR 番号とソフトウェア SLR 番号の 2 種類があります。

ハードウェア SLR 番号は、デバイス内で SLR がコンフィギュレーションされる順番を表します。マスター SLR は常に存在し、ハードウェア SLR 番号は 0 です。それ以外のスレーブ SLR のハードウェア SLR 番号は、基本的にマスター SLR からの距離に近いものから順に番号が割り当てられます。

SLR に配置されたコントローラー インスタンスは、実行時にその SLR の ICAP (内部コンフィギュレーション アクセス ポート) 経由で IDCODE レジスタを読み出してハードウェア SLR 番号を確認します。SSI デバイスにインプリメントされたコントローラーとコマンドおよびステータス情報を交換する際は、必ずハードウェア SLR 番号を使用します。

ソフトウェア SLR 番号は、デバイスの最下位から最上位に向かって SLR の物理的な順番を表します。マスター SLR は常に存在しますが、そのソフトウェア SLR 番号はデバイスにより異なります。ソフトウェア SLR 番号はザイリンクス開発ツールのデバイス ビューに表示されます。

表 5-1 に、ハードウェア SLR 番号とソフトウェア SLR 番号の対応関係を示します。

表 5-1 : デバイスの SLR 番号

デバイス	ソフトウェア SLR 番号	ハードウェア SLR 番号	SLR タイプ
XC7VH580T	2	X	GTZ
	1	1	スレーブ
	0	0	マスター
XC7VH870T	4	X	GTZ
	3	2	スレーブ
	2	0	マスター
	1	1	スレーブ
	0	X	GTZ
XC7VX1140T	3	3	スレーブ
	2	2	スレーブ
	1	0	マスター
	0	1	スレーブ
XC7V2000T	3	3	スレーブ
	2	2	スレーブ
	1	0	マスター
	0	1	スレーブ



ヒント : SLR 番号の詳細および変換方法を知らなくてもコントローラーを SSI デバイスにインプリメントするのに支障はありません。しかし特定の SLR にエラーを挿入したい場合はこの情報が役立ちます。

図 5-4 に、SSI デバイスの場合のサンプル デザインのポートを示します。ポートは 6 つのグループにまとめられています。グレーで示したグループは、一部のコンフィギュレーションにのみ存在します。SSI デバイスでは、ステータス インターフェイスのポートはバスになり、そのバス幅は SSI デバイスの SLR の数で決まります。

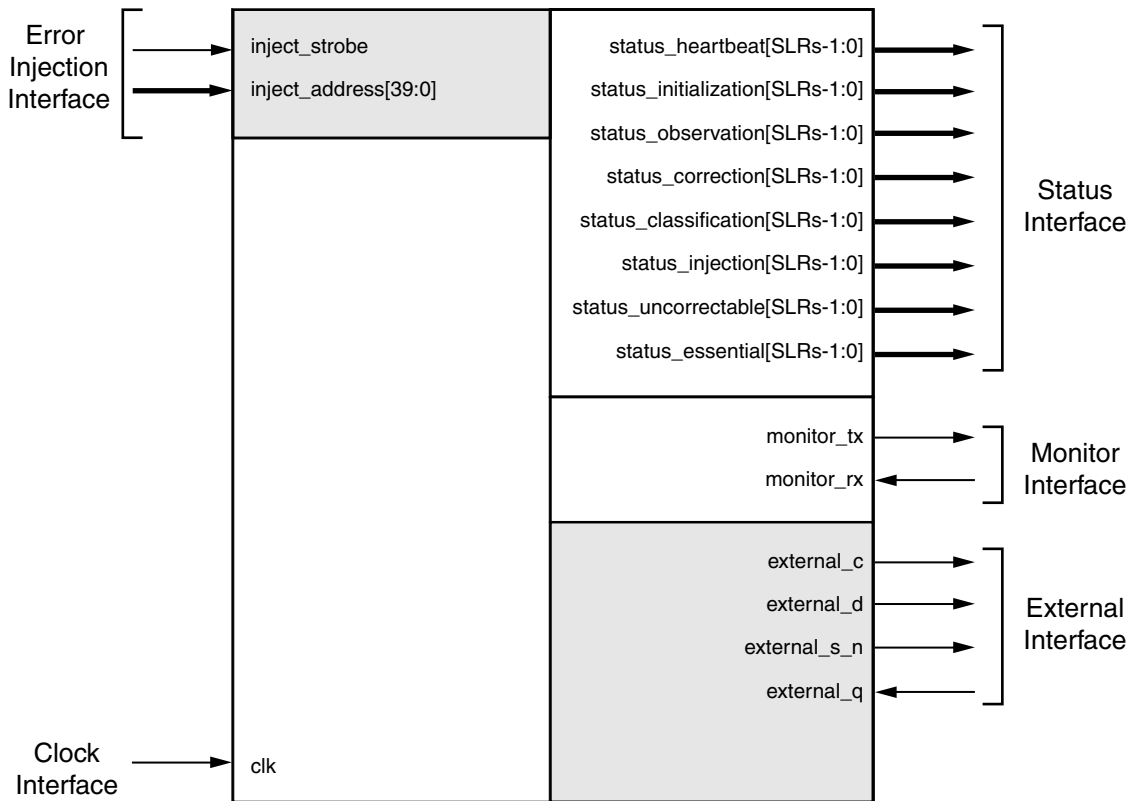


図 5-4 : サンプル デザインのポート (SSI デバイスの場合)

システム レベル サンプル デザインにはリセット入出力はありません。コントローラーが自動的に自己初期化を実行します。次に、コントローラーは必要に応じてシムを初期化します。

システム レベル サンプル デザインは、clk を唯一のクロックとして使用する完全同期デザインです。すべてのステート エレメントはこのクロックの立ち上がりエッジに同期します。このため、インターフェイスもすべてこのクロックの立ち上がりエッジに同期します。

ステータス インターフェイス

ステータス インターフェイスは、コントローラーからの信号をそのまま出力します。このインターフェイスの詳細は、第2章「ステータス インターフェイス」を参照してください。

クロック インターフェイス

クロック インターフェイスは、システム レベル サンプル デザインにクロックを供給する目的で使用します。クロック信号は内部でグローバル クロック バッファーを経由してすべての同期ロジック セルへ分配されます。

表 5-2 : クロック インターフェイスの詳細

名前	センス	方向	説明
clk	エッジ	入力	システム レベル サンプル デザインのマスター クロックを受信します。

モニター インターフェイス

モニター インターフェイスは常に存在します。システム レベル サンプル デザインに含まれる MON シムは UART です。このシムは、コントローラーが生成したステータス情報 (1 バイトの ASCII コード) をシリアライズしてシリアル送信します。同様に、このシムは入力されたコマンド情報 (ASCII コードのビットストリーム) をデシリアライズしてパラレル データとしてコントローラーに送信します。

MON シムは標準のシリアル通信プロトコルを使用します。このシムは同期およびオーバー サンプリング ロジックを備えており、同じ公称ポー レートで動作する非同期シリアル デバイスをサポートします。詳細は、[第3章「コアを使用するデザイン」](#)を参照してください。

MON シムが提供するインターフェイスには、エンベデッド マイクロコントローラーからデスクトップ コンピューターまで幅広いデバイスを直接接続できます。システム要件によっては外部レベル変換器が必要です。

表 5-3 : モニター インターフェイスの詳細

名前	センス	方向	説明
monitor_tx	LOW	出力	システム レベル サンプル デザインからのシリアル送信データ
monitor_rx	LOW	入力	システム レベル サンプル デザインへのシリアル受信データ

外部インターフェイス

外部インターフェイスは、コントローラーが外部データにアクセスする必要がある場合に存在します。このインターフェイスが存在する場合、システム レベル サンプル デザインの EXT シムの機能は SPI バス マスターに固定されます。EXT シムは、アドレスとバイト数で構成されるコマンドをコントローラーから受け取ります。EXT シムは要求されたデータを外部 SPI フラッシュからフェッチするための SPI バス トランザクションを生成します。フェッチしたデータは、コントローラーが受信できるフォーマットに EXT シムによって変換されます。

EXT シムは標準 SPI バス プロトコルを使用し、「モード 0」と呼ばれる最も一般的なモード (CPOL = 0, CPHA = 0) を実装します。SPI バスのクロック周波数はシステム レベル サンプル デザインのマスター クロックの 1/2 に固定されます。外部タイミング バジレットの詳細は、[第3章「コアを使用するデザイン」](#)を参照してください。

EXT シムが提供するインターフェイスには、幅広い種類の標準 SPI フラッシュを直接接続できます。システム要件によっては外部レベル変換器が必要です。

表 5-4 : 外部インターフェイスの詳細

名前	センス	方向	説明
external_c	エッジ	出力	外部 SPI フラッシュの SPI バス クロック
external_d	HIGH	出力	外部 SPI フラッシュの SPI バス「マスター出力、スレーブ入力」信号
external_s_n	LOW	出力	外部 SPI フラッシュの SPI バス セレクト信号
external_q	HIGH	入力	外部 SPI フラッシュの SPI バス「マスター入力、スレーブ出力」信号

エラー挿入インターフェイス

エラー挿入インターフェイスは、コントローラーがエラー挿入をサポートしており、HID シムが I/O ピンに設定されている場合に存在します。このインターフェイスはすべてのコントローラーにバス接続されます。このインターフェイスの詳細は、第 2 章「エラー挿入インターフェイス」を参照してください。

コントローラーがエラー挿入をサポートしていても HID シムが Vivado Design Suite デバッグ機能に設定されている場合、またはエラー挿入が無効な場合は、このインターフェイスはありません。

デモンストレーション用テストベンチ

サンプル デザインにはシミュレーション テスト ハーネスが付属します。このため、SEM Controller を含むデザインの機能およびタイミング シミュレーションを標準のザイリンクス シミュレーション フローを利用して実行できます。ただしシミュレーションでは SEM Controller の動作は観察できません。これにはハードウェア ベースの評価が必要です。

インプリメンテーション

このサンプル デザインはデフォルトでは生成されません。これはユーザーからの要求によって生成され、Vivado の新規インスタンスで開くことができます。このためユーザー デザインはそのまま、使用されている各種コアのサンプルのみを表示および変更できます。サンプル デザインを生成するには、[Design Sources] の下の XCI ファイルを右クリックして [Open IP Example Design] をクリックします。

合成およびインプリメンテーションの実行

合成とインプリメンテーションは、Flow Navigator の該当するオプションをクリックしてそれぞれ個別に実行できます。

ビットストリームの生成

エラー分類または置換による訂正のオプションを有効にした場合、ビットストリームを生成する前にエッセンシャル ビットを有効にする Tcl プロパティを次のように設定する必要があります。

```
set_property bitstream.seu.essentialbits yes [current_design]
```

この場合、ビットストリームの生成には大容量のシステム RAM が必要です。



ヒント : 64 ビット オペレーティング システムと 16GB 以上のシステム RAM を使用することを推奨します。

外部メモリ プログラミング ファイルの作成 (非 SSI デバイス)

エラー分類または置換によるエラー訂正をサポートするために外部データ ストレージが必要な場合、特別な `write_bitstream` 出力ファイルを SPI フラッシュ プログラミング ファイルにポストプロセスするための Tcl スクリプトを実行します。

サンプル プロジェクトでは `makedata.tcl` スクリプトが生成されます。ビットストリームの生成が完了したら、スクリプトの場所を指定して Tcl コンソールでこのスクリプトを `source` コマンドで実行します。

```
source <path to example project>/<component_name>_example.srscs/sources_1/imports/  
<component_name>/implement/makedata.tcl
```

```
source sem_0_example.srscs/sources_1/imports/sem_0/implement/makedata.tcl
```

次に、インプリメンテーション結果のディレクトリを見つけます。[Design Runs] ウィンドウをクリックし、実行したインプリメンテーションを選択し、[Implementation Run Properties] ウィンドウに表示される Run ディレクトリを確認します。Tcl コンソールでインプリメンテーション結果のディレクトリに移動し、`write_bitstream` 出力ファイルに対して `makedata` スクリプトを実行します。

```
cd <component_name>_example.runs/impl_1
```

```
cd sem_0_example.runs/impl_1
```

エラー分類または置換による訂正を有効にしている場合 (このサンプル フローではこのスクリプトを実行):

```
makedata -ebc <ebc filename> -ebd <ebd filename> datafile
```

```
makedata -ebc sem_0_sem_example.ebc -ebd sem_0_sem_example.ebd datafile
```

置換による訂正を有効にしている場合:

```
makedata -ebc <ebc filename> datafile
```

```
makedata -ebc sem_0_sem_example.ebc datafile
```

エラー分類を有効にしている場合:

```
makedata -ebd <ebd filename> datafile
```

```
makedata -ebd sem_0_sem_example.ebd datafile
```

このコマンドを実行すると VMF、BIN、および MCS ファイルが生成されます。

外部メモリ プログラミング ファイルの作成 (SSI デバイス)

エラー分類または置換によるエラー訂正をサポートするために外部データ ストレージが必要な場合、特別な `write_bitstream` 出力ファイルを SPI フラッシュ プログラミング ファイルにポストプロセスするための Tcl スクリプトを実行します。サンプルプロジェクトでは `makedata.tcl` スクリプトが生成されます。ビットストリームの生成が完了したら、Tcl コンソールでこのスクリプトを `source` コマンドで実行します。

```
source <path to example project>/<component_name>_example.srscs/sources_1/imports/  
<component_name>/implement/makedata.tcl
```

```
source sem_0_example.srscs/sources_1/imports/sem_0/implement/makedata.tcl
```

次に、インプリメンテーション結果のディレクトリを見つけます。[Design Runs] ウィンドウをクリックし、実行したインプリメンテーションを選択し、[Implementation Run Properties] ウィンドウに表示される Run ディレクトリを確認します。Tcl コンソールでインプリメンテーション結果のディレクトリに移動し、`write_bitstream` 出力ファイルに対して `makedata` スクリプトを実行します。

```
cd <component_name>_example.runs/impl_1
```

```
cd sem_0_example.runs/impl_1
```

ビットストリーム生成中にエッセンシャル ビット ファイル (EBC および EBD ファイル) が作成されます。EBC および EBD ファイルはターゲット デバイスの SLR ごとに生成されます。有効にしたオプションに基づいて次に示す適切なコマンドを実行すると、必要なファイルが生成されます。それぞれの例には、汎用パス名とこのサンプルフローで使用する実際のインスタンスの両方を示しています。`-ebc` または `-ebd` スイッチの後に指定するファイルの数は、ターゲット デバイスの SLR の数を指定します。

エラー分類または置換による訂正を有効にしている場合 (このサンプル フローではこのスクリプトを実行):

```
makedata -ebc <file0> <file1> <file2> <file3> -ebd <file0> <file1> <file2> <file3>  
datafile
```

```
makedata -ebc sem_0_sem_example_0.ebc sem_0_sem_example_1.ebc  
sem_0_sem_example_2.ebc sem_0_sem_example_3.ebc -ebd  
sem_0_sem_example_0.ebd sem_0_sem_example_1.ebd  
sem_0_sem_example_2.ebd sem_0_sem_example_3.ebd datafile
```

置換による訂正を有効にしている場合:

```
makedata -ebc <file0> <file1> <file2> <file3> datafile
```

```
makedata -ebc sem_0_sem_example_0.ebc sem_0_sem_example_1.ebc  
sem_0_sem_example_2.ebc sem_0_sem_example_3.ebc datafile
```

エラー分類を有効にしている場合:

```
makedata -ebd <file0> <file1> <file2> <file3> datafile
```

```
makedata -ebd sem_0_sem_example_0.ebd sem_0_sem_example_1.ebd  
sem_0_sem_example_2.ebd sem_0_sem_example_3.ebd datafile
```

このコマンドを実行すると VMF、BIN、および MCS ファイルが生成されます。

外部メモリ プログラミング ファイル

置換によるエラー訂正を有効にした場合は、コンフィギュレーション データのイメージが必要です。エラー分類を有効にした場合は、エッセンシャル ビットのルックアップ テーブルのイメージが必要です。このため、これらデータセットのいずれか、または両方が必要になります。これらのデータセットのサイズはどちらも同じで、ターゲット デバイスによって決まります。データセットは `write_bitstream` アプリケーションによって生成されます。

データのフォーマットはバイナリで、`write_bitstream` によって生成されたすべてのデータセットを使用する必要があります。外部ストレージはバイト アドレス指定可能なものが必要です。`fetch_tbladdr[31:0]` によって SEM Controller に指定されたアドレスには、小規模なテーブルが必要です。デフォルトでは、`fetch_tbladdr[31:0]` は 0 です。

非 SSI デバイスのテーブルフォーマットは次のとおりです。

- バイト 0: 置換データ バイト 0 (LSB) の先頭を示す 32 ビット ポインター
- バイト 1: 置換データ バイト 1 の先頭を示す 32 ビット ポインター
- バイト 2: 置換データ バイト 2 の先頭を示す 32 ビット ポインター
- バイト 3: 置換データ バイト 3 (MSB) の先頭を示す 32 ビット ポインター
- バイト 4: エッセンシャルビット データ バイト 0 (LSB) の先頭を示す 32 ビット ポインター
- バイト 5: エッセンシャルビット データ バイト 1 の先頭を示す 32 ビット ポインター
- バイト 6: エッセンシャルビット データ バイト 2 の先頭を示す 32 ビット ポインター
- バイト 7: エッセンシャルビット データ バイト 3 (MSB) の先頭を示す 32 ビット ポインター
- 残りのバイトは予約バイトで、すべて 1 を書き込みます。

SSI デバイスのテーブルフォーマットは次のとおりです。

- バイト 0: ハードウェア SLR0 (マスター) 置換データ バイト 0 (LSB) の先頭を示す 32 ビット ポインター
- バイト 1: ハードウェア SLR0 (マスター) 置換データ バイト 1 の先頭を示す 32 ビット ポインター
- バイト 2: ハードウェア SLR0 (マスター) 置換データ バイト 2 の先頭を示す 32 ビット ポインター
- バイト 3: ハードウェア SLR0 (マスター) 置換データ バイト 3 (MSB) の先頭を示す 32 ビット ポインター
- バイト 4: ハードウェア SLR0 (マスター) エッセンシャルビット データ バイト 0 (LSB) の先頭を示す 32 ビット ポインター
- バイト 5: ハードウェア SLR0 (マスター) エッセンシャルビット データ バイト 1 の先頭を示す 32 ビット ポインター
- バイト 6: ハードウェア SLR0 (マスター) エッセンシャルビット データ バイト 2 の先頭を示す 32 ビット ポインター
- バイト 7: ハードウェア SLR0 (マスター) エッセンシャルビット データ バイト 3 (MSB) の先頭を示す 32 ビット ポインター
- バイト 8: ハードウェア SLR1 (オプション スレーブ) 置換データ バイト 0 (LSB) の先頭を示す 32 ビット ポインター
- バイト 9: ハードウェア SLR1 (オプション スレーブ) 置換データ バイト 1 の先頭を示す 32 ビット ポインター
- バイト 10: ハードウェア SLR1 (オプション スレーブ) 置換データ バイト 2 の先頭を示す 32 ビット ポインター
- バイト 11: ハードウェア SLR1 (オプション スレーブ) 置換データ バイト 3 (MSB) の先頭を示す 32 ビット ポインター
- バイト 12: ハードウェア SLR1 (オプション スレーブ) エッセンシャルビット データ バイト 0 (LSB) の先頭を示す 32 ビット ポインター

- バイト 13 : ハードウェア SLR1 (オプション スレーブ) エッセンシャル ビット データ バイト 1 の先頭を示す 32 ビット ポインター
- バイト 14 : ハードウェア SLR1 (オプション スレーブ) エッセンシャル ビット データ バイト 2 の先頭を示す 32 ビット ポインター
- バイト 15 : ハードウェア SLR1 (オプション スレーブ) エッセンシャル ビット データ バイト 3 (MSB) の先頭を示す 32 ビット ポインター
- バイト 16 : ハードウェア SLR2 (オプション スレーブ) 置換データ バイト 0 (LSB) の先頭を示す 32 ビット ポインター
- バイト 17 : ハードウェア SLR2 (オプション スレーブ) 置換データ バイト 1 の先頭を示す 32 ビット ポインター
- バイト 18 : ハードウェア SLR2 (オプション スレーブ) 置換データ バイト 2 の先頭を示す 32 ビット ポインター
- バイト 19 : ハードウェア SLR2 (オプション スレーブ) 置換データ バイト 3 (MSB) の先頭を示す 32 ビット ポインター
- バイト 20 : ハードウェア SLR2 (オプション スレーブ) エッセンシャル ビット データ バイト 0 (LSB) の先頭を示す 32 ビット ポインター
- バイト 21 : ハードウェア SLR2 (オプション スレーブ) エッセンシャル ビット データ バイト 1 の先頭を示す 32 ビット ポインター
- バイト 22 : ハードウェア SLR2 (オプション スレーブ) エッセンシャル ビット データ バイト 2 の先頭を示す 32 ビット ポインター
- バイト 23 : ハードウェア SLR2 (オプション スレーブ) エッセンシャル ビット データ バイト 3 (MSB) の先頭を示す 32 ビット ポインター
- バイト 24 : ハードウェア SLR3 (オプション スレーブ) 置換データ バイト 0 (LSB) の先頭を示す 32 ビット ポインター
- バイト 25 : ハードウェア SLR3 (オプション スレーブ) 置換データ バイト 1 の先頭を示す 32 ビット ポインター
- バイト 26 : ハードウェア SLR3 (オプション スレーブ) 置換データ バイト 2 の先頭を示す 32 ビット ポインター
- バイト 27 : ハードウェア SLR3 (オプション スレーブ) 置換データ バイト 3 (MSB) の先頭を示す 32 ビット ポインター
- バイト 28 : ハードウェア SLR3 (オプション スレーブ) エッセンシャル ビット データ バイト 0 (LSB) の先頭を示す 32 ビット ポインター
- バイト 29 : ハードウェア SLR3 (オプション スレーブ) エッセンシャル ビット データ バイト 1 の先頭を示す 32 ビット ポインター
- バイト 30 : ハードウェア SLR3 (オプション スレーブ) エッセンシャル ビット データ バイト 2 の先頭を示す 32 ビット ポインター
- バイト 31 : ハードウェア SLR3 (オプション スレーブ) エッセンシャル ビット データ バイト 3 (MSB) の先頭を示す 32 ビット ポインター
- 残りのバイトは予約バイトで、すべて 1 を書き込みます。

データ ブロックが存在しない場合はポインターの値を 0xFFFFFFFF とします。エッセンシャル ビット データと置換データは任意のアドレスに配置できますが、各データ ブロックを連続して配置してバースト読み出しできるようにしておく必要があります。デバイス境界を越えた読み出しバーストをサポートしない SPI フラッシュの場合、デバイス境界をまたがないようにデータ ブロックを配置する必要があります。たとえば集積度 256Mb を超える SPI フラッシュの多くは、256Mb 境界を越えた読み出しバーストをサポートしていません。

write_bitstream 出力ファイルをポストプロセスする Tcl スクリプトにより、次の 3 つのファイルが生成されます。

- SPI フラッシュ デバイス プログラミング用の Intel Hex データ ファイル (MCS)
- SPI フラッシュ デバイス プログラミング用のバイナリ データ ファイル (BIN)
- SPI フラッシュ シミュレーション モデルをロードするための初期化ファイル (VMF)

検証、互換性、相互運用性

SEM Controller とサンプル デザインは、自動ハードウェア テストベンチやハードウェア協調シミュレーション ツールなどいくつかの方法で一緒に検証されています。これらは、加速粒子を照射してバリデーションが行われており、自然に発生したランダム エラー イベントにソリューションが正しく対処できることが確認されています。

検証

SEM Controller の検証目標は、製品の機能仕様をもとに決定しています。高い製品品質を確保するため、ハイブリッドアプローチで検証を実行しています。テストはハードウェア検証を重点的に実施し、協調シミュレーション テストベンチを補完的に実行しました。使用した手法とツールは次のとおりです。

- ハードウェア テストベンチによるダイナミック チェック
 - 機能カバレッジ: 期待されるビヘイビアと実際のデザインのビヘイビアを比較
- 協調シミュレーション テストベンチによるダイナミック チェック
 - 機能カバレッジ: 期待されるビヘイビアと実際のデザインのビヘイビアを比較
 - コード カバレッジ: コントローラーの FSM の実行トレースを記録して解析
- チェック ツールスイートによるスタティック チェック
 - リント
 - クロック ドメイン クロッシング (クロック乗せ換え)

ハードウェア検証プラットフォームで使用した SPI フラッシュ デバイスは次のとおりです。

- M25P128 (ST Microelectronics/Numonyx)
- M25L25635E (Macronix)
- N25Q512 (Micron)
- N25Q00 (Micron)

バリデーション

ハードウェアバリデーションは、製品リリース前の最後の関門となる重要なテストの 1 つです。ハードウェアバリデーションでは次のテストを実施して付加価値を高めています。

- 外部インターフェイスの評価
 - 外部メモリ システムに対するタイミング バジェット評価
- 統合およびインプリメンテーション
 - 生成される可能性のあるすべてのコア ネットリストのハードウェア テスト
- 動作環境の堅牢性
 - サポートされるデバイス リストのすべてのサブ ファミリのサンプル テスト

適合性検査

業界標準の認証試験は定義されていません。生成したコア ネットリストは、加速粒子を照射した状態で検査する必要があります。この検査により、次の点を確認します。

- エラー挿入とは関係なく検出、訂正、分類が正しく行われること。この検証手法はソリューション自体によるエラー挿入に基づいており、エラー挿入と関係ない検出、訂正、分類プロセスはテストしません。加速粒子照射テスト中のエラーは、ソリューション自体によるエラー挿入とは関係なく発生します。
- エラーの検出、訂正、分類を含め、ソリューションが通常の予想される動作を示すこと。

移行およびアップグレード

この付録では、新しいバージョンの IP コアへのアップグレードについて説明します。

Vivado Design Suite への移行

Vivado Design Suite への移行方法については、『ISE から Vivado Design Suite への移行ガイド』(UG911) [\[参照 11\]](#) を参照してください。

Vivado Design Suite でのアップグレード

このセクションでは、コアのバージョンをアップグレードする際に必要なユーザー ロジックおよびポートの変更点について説明します。

パラメーターの変更点

[Error Injection Shim] の [chipscope] オプションが [Vivado lab tools] オプションに変更されました。[Error Injection Shim] に [Vivado lab tools] を選択した場合、VIO LogiCORE IP が生成され、SEM v4.1 IP サンプル デザインに含まれます。

ポートの変更点

このバージョンでは、ポートの変更はありません。

デバッグ

この付録では、ザイリンクス サポート ウェブサイトより入手可能なリソースおよびデバッグ ツールについて説明します。



ヒント : IP の生成にエラーが発生し停止した場合、ライセンスに問題がある可能性があります。詳細は、[第 1 章の「ライセンスチェッカー」](#)を参照してください。

ザイリンクス ウェブサイト

SEM コアを使用した設計およびデバッグでヘルプが必要な場合は、[ザイリンクス サポート ウェブ ページ](#)から製品の資料、リリース ノート、アンサーなどを参照するか、テクニカル サポートでケースを開いてください。

資料

この製品ガイドは SEM コアに関する主要資料です。このガイド並びに全製品の設計プロセスをサポートする資料はすべて、ザイリンクス サポート ウェブ ページ (<http://japan.xilinx.com/support>) またはザイリンクスの Documentation Navigator から入手できます。

Documentation Navigator は[ダウンロード ページ](#)からダウンロードできます。このツールの詳細および機能は、インストール後にオンライン ヘルプを参照してください。

アンサー

アンサーには、よく発生する問題についてその解決方法、およびザイリンクス製品に関する既知の問題などの情報が記載されています。アンサーは、ユーザーが該当製品の最新情報にアクセスできるよう作成および管理されています。

このコアに関するアンサーの検索には、[ザイリンクス サポート ウェブ ページ](#)にある検索ボックスを使用します。よりの確な検索結果を得るには、次のようなキーワードを使用してください。

- 製品名
- ツールで表示されるメッセージ
- 問題の概要

検索結果は、フィルター機能を使用してさらに絞り込むことができます。

SEM コアに関するマスター アンサー :

AR : [54642](#)

テクニカル サポート

ザイリンクスは、製品資料の説明に従って使用されている LogiCORE™ IP 製品に対するテクニカル サポートを japan.xilinx.com/support で提供しています。次のいずれかに該当する場合、タイミング、機能、製品サポートは保証されません。

- 資料で定義されていないデバイスにインプリメントした場合
- 資料で定義されている許容範囲を超えてカスタマイズした場合
- 「DO NOT MODIFY」とされているデザイン セクションに変更を加えた場合

テクニカル サポートへのお問い合わせは、<http://japan.xilinx.com/support> にアクセスしてください。

デバッグ ツール

SEM デザインの問題に対応できるツールは多数あります。さまざまな状況をデバッグするのに有益なツールを理解しておくことが重要です。

Vivado Design Suite のデバッグ機能

Vivado® Design Suite のデバッグ機能では、ユーザー デザインに Logic Analyzer および Virtual I/O コアを直接挿入します。デバッグ機能を使用すると、トリガー条件を設定して、ハードウェアでアプリケーションおよび統合ブロックのポート信号をハードウェアに取り込むことができます。取り込まれた信号は、その後解析できます。Vivado IDE のこの機能は、ザイリンクス デバイスで実行されるデザインの論理デバッグおよびバリデーションに使用されます。

Vivado ロジック解析は次の LogiCORE IP ロジック デバッグ コアと共に使用されます。

- ILA 2.0 (およびそれ以降のバージョン)
- VIO 2.0 (およびそれ以降のバージョン)

詳細は、『Vivado Design Suite ユーザー ガイド : プログラムおよびデバッグ』(UG908) [参照 4] を参照してください。

リファレンス ボード

SEM コアはさまざまなザイリンクス開発ボードでサポートされています。これらのボードを使用してデザインのプロトタイプを作成し、コアがシステムと通信できるようにします。

- ザイリンクスの評価ボード :
 - AC701
 - KC705
 - VC707
 - ZC702
 - ZC706

ハードウェア デバッグ

ハードウェアの問題は、リンク立ち上げ時の問題から、何時間ものテストの後に発生する問題までさまざまです。ここでは、一般的な問題のデバッグ手順を説明します。Vivado Design Suite のデバッグ機能は、ハードウェア デバッグに有益なリソースです。次の各セクションに示す信号を Vivado Design Suite のデバッグ機能でプローブすることで、個々の問題をデバッグできます。

汎用チェック

コアに対するタイミング制約がサンプル デザインからすべて適切に取り込まれていること、さらにインプリメンテーション時にこれらの制約がすべて満たされていることを確認します。

クロック管理ブロックを使用するデザインでは、ブロックのステータスを監視してロックが完了したか確認してください。

第 3 章の「その他の注意事項」を参照して、サポートされない設定や機能が使用されていないかチェックしてください。



推奨: ザイリンクスは、SEM IP コアをなるべく早い段階で (理想的にはプロジェクトの最初から) 統合することを推奨しています。詳細は、81 ページの「[統合およびバリデーション](#)」を参照してください。

インターフェイスのデバッグ

モニター インターフェイス

モニター インターフェイスの使用は必須ではありませんが、モニター インターフェイスを接続する手段を確保しておくことを強く推奨します。モニター インターフェイスからは、潜在的な問題のデバッグやトラブルシューティングに役立つ重要な情報が得られます。システム レベル サンプル デザインに含まれる MON シムは UART で、標準 RS232 ポートに接続するか、USB-to-UART を介して USB に接続できます。

SEM Controller が正しく動作しているかは、SEM Controller がモニター インターフェイスから出力する初期化レポートで確認します。通常、このレポートのフォーマットは次のとおりです。

```
X7_SEM_V4_1
SC 01
FS 02
ICAP OK
RDBK OK
INIT OK
SC 02
O>
```

1 行目はデバイスとコア バージョンを示しています。3 行目は SEM Controller の機能セットを示しており、コア生成時に選択した SEM コントローラー コアのオプションの要約が表示されます。

MON シムを使用していて初期化レポートに正常な文字が表示されない場合は、第 3 章の「[モニター インターフェイス](#)」を参照して端末プログラムの通信設定が正しいかどうか確認してください。また、実際に SEM Controller に供給されるクロック周波数と MON シムの V_ENABLETIME パラメーター値の組み合わせで標準ボーレートが得られるか、また端末プログラムの通信設定がビット レートと一致しているかを確認してください。詳細は [式 3-1](#) および [式 3-2](#) を参照してください。

SEM Controller が ICAP (内部コンフィギュレーション アクセス ポート) プリミティブ経由で FPGA コンフィギュレーション ロジックと通信できない場合、初期化レポートは「ICAP」の行で止まり、「OK」が表示されません。この場合、次のような初期化レポートが表示されます。

```
X7_SEM_V4_1
SC 01
FS 02
ICAP
```

この場合、ICAP が応答しない理由を特定する必要があります。一般的な確認事項は次のとおりです。

- 使用するデバイスに合わせて ICAP が正しくインスタンス化されているか確認する。
 - サンプル デザインを使用して正しいインスタンス化を得る。
 - Zynq-7000 デバイスへのインプリメンテーションでは、SEM Controller の icap_grant 入力に対するハードウェアおよびソフトウェア制御を見直す。
- ほかのプロセスによって ICAP がブロックされていないか確認する。
 - JTAG アクセスが発生していないこと、SelectMAP の persist を設定していないことを確認する。
- XAPP517 で説明している ICAP 共有を使用しない場合、SEM Controller と ICAP を直接接続する必要があります。SEM Controller と ICAP の間にパイプライン処理を追加することは避けてください。

第 3 章の「その他の注意事項」で説明したように、SEM Controller はビットストリーム認証をサポートしていません。また、POST_CRC、POST_CONFIG_CRC、またはその他の関連する制約と併用することはできません。初期化レポートが ICAP、RDBK、または INIT の行で止まる場合、これらが使用中でないことを確認してください。

クロッキング

SEM Controller のクロックはオシレーターからピンを経由して直接 SEM Controller へ供給することを推奨します。内部の PLL または DCM でクロックを生成すると、クロック生成に関するコンフィギュレーション セルが SEU イベントの影響を受ける可能性があります。その確率は非常にわずかですが、信頼性を低下させる要因はなるべく排除しておく必要があります。どうしても PLL/DCM 出力またはその他のロジックを使用してクロックを生成する必要がある場合は、デザインのスタートアップ中や PLL/DCM がロックするまでの間を含め、クロックが SEM Controller の最小周期要件に違反しないよう注意が必要です。

クロック管理ブロックを使用する場合、クロックが安定するまで SEM Controller へのクロックのトグルを抑制してください。たとえば BUFGMUX または BUFGCE を使用して、PLL/DCM がロックするまで SEM Controller のクロックがトグルしないようにします。

その他のリソースおよび法的通知

ザイリンクス リソース

アンサー、資料、ダウンロード、フォーラムなどのサポート リソースは、[ザイリンクス サポート サイト](#)を参照してください。

参考資料

次の資料は、この製品ガイドの補足資料として役立ちます。

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. 『7 シリーズ FPGA コンフィギュレーション ユーザー ガイド』(UG470 : [英語版](#)、[日本語版](#))
2. 『ザイリンクス デバイス信頼性レポート』([UG116](#))
3. 『Zynq-7000 All Programmable SoC テクニカル リファレンス マニュアル』(UG585 : [英語版](#)、[日本語版](#))
4. 『Vivado Design Suite ユーザー ガイド：プログラムおよびデバッグ』(UG908 : [英語版](#)、[日本語版](#))
5. 『Vivado Design Suite ユーザー ガイド：IP インテグレーターを使用した IP サブシステムの設計』(UG994 : [英語版](#)、[日本語版](#))
6. 『Vivado Design Suite ユーザー ガイド：IP を使用した設計』(UG896 : [英語版](#)、[日本語版](#))
7. 『Vivado Design Suite ユーザー ガイド：入門』(UG910 : [英語版](#)、[日本語版](#))
8. 『Vivado Design Suite ユーザー ガイド：ロジック シミュレーション』(UG900 : [英語版](#)、[日本語版](#))
9. Udipi, Sujeeth『Comprehensive Full-Chip Methodology to Verify EM and Dynamic Voltage Drop on High Performance FPGA Designs in the 20nm Technology』(DesignCon 2014 でのプレゼンテーション。最終アクセス日 2015 年 6 月 19 日)
http://japan.xilinx.com/events/designcon2014/1_TH6Paper_ComprehensiveFull_ChipMethodologytoVerifyElectromigration_v2.pdf
10. 『Vivado Design Suite ユーザー ガイド：インプリメンテーション』(UG904 : [英語版](#)、[日本語版](#))
11. 『ISE から Vivado Design Suite への移行ガイド』(UG911 : [英語版](#)、[日本語版](#))
12. 『ディープ サブミクロン IC への大気中性子の影響に関する継続実験』([WP286](#))

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
2015年9月30日	4.1	<ul style="list-style-type: none"> 表 2-4 を更新。 第 4 章「デザインフローの手順」に「ユーザー パラメーター」と「統合およびバリデーション」のセクションを追加。
2014年11月19日	4.1	<ul style="list-style-type: none"> 新しい Artix-7 (XC7A15T) および Zynq-7000 (XC7Z035) デバイスのリソース使用量とパフォーマンス メトリクスを追加。 ビットストリーム暗号化がサポートされていることを明確化。 status_heartbeat のハートビート パルス最大間隔の仕様値を 150 サイクルに訂正。
2014年4月2日	4.1	<ul style="list-style-type: none"> コアを v4.1 にアップデート。 従来の ChipScope サポートを Vivado ラボ ツールのサポートに変更。 新しい Artix-7 および Zynq-7000 デバイスのリソース使用量とパフォーマンス メトリクスを追加。 makedata.tcl スクリプトの実行に関する記述を更新。 「合成およびインプリメンテーション」のセクションを追加。
2013年6月19日	4.0	<ul style="list-style-type: none"> 第 5 章の「インプリメンテーション」で makedata.tcl スクリプトの実行に関する記述を更新。 第 2 章の「リソース使用状況」で Virtex-7 SSI デバイスのリソース使用量を更新。 第 2 章の「ソリューションのレイテンシ」で Zynq-7000 7Z100 デバイスのソリューションレイテンシを追加。 第 3 章「コアを使用するデザイン」でシステム レベルの監視機能に関する情報を追加。
2013年3月20日	4.0	<ul style="list-style-type: none"> コアを v4.0 にアップデート。 ISE Design Suite のサポートを終了。 一部の 7 シリーズ デバイスで最大クロック周波数を 100MHz に向上。
2012年12月18日	3.0	<ul style="list-style-type: none"> コア バージョン v3.4、ISE Design Suite 14.4、Vivado Design Suite 2012.4 にアップデート。 プリプロダクションの Zynq-7000、Artix-7、Virtex-7 SSI デバイスのサポートを追加。 コンフィギュレーション リードバックに関する新しいガイダンスに基づき、Spartan-6 ソリューションの設計を変更。
2012年7月25日	2.0	Vivado Design Suite のサポートを追加。
2012年4月24日	1.0	初版。DS796『LogiCORE IP Soft Error Mitigation Controller データシート』と UG764『LogiCORE IP Soft Error Mitigation Controller ユーザー ガイド』を置き換え。

法的通知

本通知に基づいて貴殿または貴社（本通知の被通知者が個人の場合には「貴殿」、法人その他の団体の場合には「貴社」。以下同じ）に開示される情報（以下「本情報」といいます）は、ザイリンクスの製品を選択および使用することのためにのみ提供されます。適用される法律が許容する最大限の範囲で、(1) 本情報は「現状有姿」、およびすべて受領者の責任で (with all faults) という状態で提供され、ザイリンクスは、本通知をもって、明示、黙示、法定を問わず（商品性、非侵害、特定目的適合性の保証を含みますがこれらに限られません）、すべての保証および条件を負わない（否認する）ものとし、(2) ザイリンクスは、本情報（貴殿または貴社による本情報の使用を含む）に関し、起因し、関連する、いかなる種類・性質の損失または損害についても、責任を負わない（契約上、不法行為上（過失の場合を含む）、その他のいかなる責任の法理によるかを問わない）ものとし、当該損失または損害には、直接、間接、特別、付随的、結果的な損失または損害（第三者が起した行為の結果被った、データ、利益、業務上の信用の損失、その他あらゆる種類の損失や損害を含みます）が含まれるものとし、それは、たとえ当該損害や損失が合理的に予見可能であったり、ザイリンクスがそれらの可能性について助言を受けていた場合であったとしても同様です。ザイリンクスは、本情報に含まれるいかなる誤りも訂正する義務を負わず、本情報または製品仕様のアップデートを貴殿または貴社に知らせる義務も負いません。事前の書面による同意のない限り、貴殿または貴社は本情報を再生産、変更、頒布、または公に展示してはなりません。一定の製品は、ザイリンクスの限定的保証の諸条件に従うこととなるので、<http://japan.xilinx.com/legal.htm#tos> で見られるザイリンクスの販売条件を参照してください。IP コアは、ザイリンクスが貴殿または貴社に付与したライセンスに含まれる保証と補助的条件に従うこととなります。ザイリンクスの製品は、フェイルセーフとして、または、フェイルセーフの動作を要求するアプリケーションに使用するために、設計されたり意図されたりしていません。そのような重大なアプリケーションにザイリンクスの製品を使用する場合のリスクと責任は、貴殿または貴社が単独で負うものです。<http://japan.xilinx.com/legal.htm#tos> で見られるザイリンクスの販売条件を参照してください。

© Copyright 2012–2015 Xilinx, Inc. Xilinx、Xilinx のロゴ、Artix、ISE、Kintex、Spartan、Virtex、Vivado、Zynq、およびこの文書に含まれるその他の指定されたブランドは、米国およびその他の各国のザイリンクス社の商標です。すべてのその他の商標は、それぞれの所有者に帰属します。

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。フィードバックは日本語で入力可能です。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。