汎用シリアル・フラッシュ・インターフェイスのインテル FPGA IP コアのユーザーガイド

インテル® Quartus® Prime 開発デザインスイートの更新情報: **19.3**

IP バージョン: **19.1**

更新情報

フィードバック

UG-20161 | 2019.11.27

最新版をウェブからダウンロード: PDF | HTML

目次

UG-20161 | 2019.11.27

フィードバック

1. Intel® FPGA IP FPGA コンフィグレーションのユーザー・ガイド

汎用シリアル・フラッシュ・インターフェイス Intel® FPGA IP コアは、シリアル・ペリフェラル・インターフェイス(SPI)フラッシュデバイスへのアクセスを提供します。汎用シリアル・フラッシュ・インターフェイス IP は、ASMI Parallel および ASMI Parallel II Intel FPGA IP コアと比較してより効率的な代替手段です。汎用シリアル・フラッシュ・インターフェイス Intel FPGA IP コアは、Intel コンフィグレーション・デバイスとさまざまなベンダーのフラッシュをサポートしています。Intel は、新しいデザインに汎用シリアル・フラッシュ・インターフェイス Intel FPGA IP コアを使用することを推奨します。

汎用シリアル・フラッシュ・インターフェイス IP を使用して、フラッシュデバイスに次のデータを書き込むことができます。

- コンフィグレーション・メモリー(1)Active Serial(AS)コンフィグレーション・スキームのコンフィグレーション・データ
- 汎用メモリー—アプリケーション固有のデータ

汎用シリアル・フラッシュ・インターフェイスの I/O は次の機能をサポートします。

- シングル、デュアル、またはクアッド I/O モード
- コントローラーがフラッシュからコードを直接実行できるようにする、Avalon Memory Mapped (Avalon-MM)スレーブ・インターフェイスを介した直接フラッシュアクセス
- 最大 3 つのマルチ・フラッシュ・デバイスのサポート(インテル® Arria® 10 そして インテル Cyclone® 10 GX デバイスのみ)
- フラッシュコントロール・ステータス・レジスターにアクセスするための汎用制御レジスター
- デバイスクロックの実行時ボーレート変更を備えたプログラマブル・クロック・ジェネレーター
- プログラマブル・チップ・セレクト遅延
- 高周波数で実行する場合のリードデータのキャプチャー・ロジック
- FPGA アクティブ・シリアル・メモリー・インターフェイス(ASMI)は、アクティブシリアル(AS)ピンへのアトム接続をブロックするか、FPGA I/O ピンにエクスポートする

関連情報

- 汎用シリアル・フラッシュ・インターフェイス Intel FPGA IP コアおよびリファレンス・デザイン (12 ページ)
- 汎用シリアル・フラッシュ・インターフェイスのインテル FPGA IP コアのリファレンス・デザイン・ファイル
- How do I enable Micron's MT25Q device support in replacement to End Of Life (EOL) EPCQ(>=256Mb) and EPCQ-L devices?

(1) コンフィグレーション・メモリーでサポートされているフラッシュデバイスは、EPCQ、EPCQ-A、EPCQ-L、および Micron* MT25Q(256Mb から 2Gb)デバイスです。

Intel Corporation.無断での引用、転載を禁じます。Intel、インテル、Intel ロゴ、Altera、ARRIA、CYCLONE、ENPIRION、MAX、NIOS、QUARTUS および STRATIX の名称およびロゴは、アメリカ合衆国および/ またはその他の国における Intel Corporation の商標です。インテルは FPGA 製品および半導体製品の性能がインテルの標準保証に準拠することを保証しますが、インテル製品およびサービスは、予告なく変更される場合があります。インテルが書面にて明示的に同意する場合を除き、インテルはここに記載されたアプリケーション、または、いかなる情報、製品、またはサービスの使用によって生じるいっさいの責任を負いません。インテル製品の顧客は、製品またはサービスを購入する前、および、公開済みの情報を信頼する前には、デバイスの仕様を最新のバージョンにしておくことをお勧めします。
*その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

ISO 9001:2015 登録済

- コンフィグレーション・デバイス
  サードパーティーのフラッシュサポートの詳細を提供します。
- 汎用シリアル・フラッシュ・インターフェイス（ODEVGSFI）トレーニングコースを使用する

1.1. デバイスファミリー・サポート

汎用シリアル・フラッシュ・インターフェイス IP は次のデバイスでサポートされています。

- インテル Agilex™
- インテル Stratix® 10
- インテル Arria 10
- インテル Cyclone 10 GX
- インテル Cyclone 10 LP
- インテル MAX® 10（汎用メモリーのみ）
- Stratix V
- Arria V
- Cyclone V
- Stratix IV
- Cyclone IV
- Arria II

関連情報

コンフィグレーション・デバイス
ユーザー・フラッシュメモリーについての詳しい情報を提供します。

1.2. 信号

図 -1: UFM のブロック図

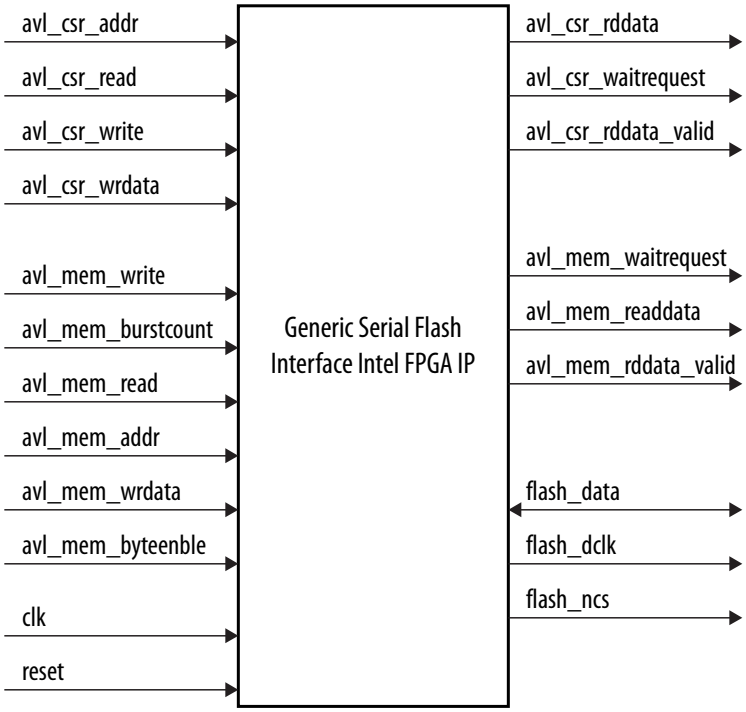


表 1. ポートの説明

信号	幅	入力/出力	説明
CSR 用の Avalon® -MM スレーブ・インターフェイス(avl_csr)			
avl_csr_addr	6	入力	Avalon -MM アドレスバス。アドレスバス幅はワード・アドレッシング単位。
avl_csr_read	1	入力	CSR に対する Avalon -MM リード制御。
avl_csr_rddata	32	出力	CSR から Avalon -MM リード・データ・バス。
avl_csr_write	1	入力	CSR に対する Avalon -MM ライト制御。
avl_csr_wrdata	32	入力	Avalon -MM がデータバスを CSR に書き込みます。
avl_csr_waitrequest	1	出力	CSR からの Avalon -MM waitrequest 制御
avl_csr_rddata_valid	1	出力	CSR リードデータが使用可能であることを示す有効な MM リードデータ。
メモリーアクセス用の Avalon -MM スレーブ・インターフェイス(avl_ mem)			
avl_mem_write	1	入力	メモリーへの Avalon -MM ライト制御
avl_mem_burstcount	7	入力	メモリーの Avalon -MM バーストカウント。値の範囲は 1～64(最大ページサイズ)です。
avl_mem_waitrequest	1	出力	メモリーからの Avalon -MM waitrequest 制御。
avl_mem_read	1	入力	メモリーへの Avalon -MM リード制御

continued...

信号	幅	入力/出力	説明
avl_mem_addr	N	入力	Avalon -MM アドレスバス。アドレスバスはワード・アドレッシングです。アドレスの幅は、フラッシュメモリーの密度によって異なります。 インテル Arria 10 および インテル Cyclone 10 GX を使用している場合、または複数のフラッシュを備えた汎用 I/O を備えたサポートされているデバイスを使用している場合、CSR を記述してチップセレクトを選択します。このアドレスを介してアクセスされる場合、IP は選択されたフラッシュをターゲットにします。
avl_mem_wrdata	32	入力	メモリーへの Avalon -MM ライト・データ・バス
avl_mem_readddata	32	出力	メモリーからの Avalon -MM リード・データ・バス。
avl_mem_rddata_valid	1	出力	Avalon - メモリー・リード・データが使用可能であることを示す Avalon -MM リードデータ有効。
avl_mem_byteenble	4	入力	メモリーへの Avalon -MM のライト・データ・イネーブル・バス。バスをイネーブルします。バーストモード中、バイトイネーブルバスはロジックハイ、4'b1111 になります。
クロックとリセット			
clk	1	入力	IP コアをクロックする入力クロック。
reset	1	入力	IP コアへの非同期リセット入力です。
割り込み			
Irq	1	出力	不正な書き込みまたは不正な消去があるかどうかを示す割り込み信号。
コンジット・インターフェイス (2)			
flash_data	4	双方向	フラッシュデバイスからデータを供給するための入力ポートまたは出力ポート。
flash_dclk	1	Output	フラッシュデバイスに clock 信号を提供します。
flash_ncs	1/3	出力	フラッシュデバイスに ncs 信号を提供します。

1.3. パラメーター

表 2. パラメーター設定

パラメーター	選択可能な値	説明
Device Density	1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048	Mb で使用されるフラッシュデバイスの密度。
Disable dedicated Active Serial interface	—	信号をデザインの最上位にルーティングします。シリアル・フラッシュ・ローダーのインテル FPGA IP をデザインに含める場合、これをイネーブルします。
Enable SPI pins interface	—	信号を SPI ピン・インターフェイスに変換します。
Number of Chip Select used	1 2 3	フラッシュに接続されているチップセレクトの数を選択します。
Enable flash simulation model	—	シミュレーション・モデルにデバイス内のフラッシュを使用します。

(2) **Enable SPI pins interface** パラメーターをイネーブルすると使用可能になります。

1.4. レジスター・マップ

表 3. レジスター・マップ

- 次の表の各アドレスオフセットは、メモリー・アドレス・スペースの 1 ワードを表します。
- IP_CLK は、IP を駆動するクロックです。
- SCLK は、フラッシュデバイスを駆動するクロックです。

オフセット(16進数)	レジスター名	R/W	フィールド名	ビット	デフォルト値(16進数)	変更内容
0	Control Register	予約済み		RCLK[31..8]	予約	
		R/W	Addressing mode	8	0x0	リードおよびライト動作のアドレッシング・モード: • 0x0:3 バイトのアドレッシング • 0x1:4 バイトのアドレッシング 4 バイトのアドレッシング・モードの場合、フラッシュにコマンドを送信して 4 バイトアドレスをイネーブルする必要があります。 このビットはリードおよびライト動作の両方で、Avalon-MM インターフェイスを介したメモリーへの直接アクセスに影響します。
		R/W	Chip select	7:4	0x0	フラッシュデバイスを選択します。 • 0x0: 最初のデバイスを選択する • 0x1: 2 番目のデバイスを選択する • 0x2: 3 番目のデバイスを選択する
		予約済み		3:1	予約	
		R/W	Enable	0	0x1	このビットを 0 に設定すると、IP の出力が無効になり、すべての出力信号が高インピーダンス状態になります。これは、他のデバイスとバスを共有するために使用できます。
1	SPI Clock Baud-rate Register	予約済み		31:5	予約	
		R/W	Baud rate divisor	4:0	0x10	IP には、フラッシュデバイスに接続するクロックを生成するための内部クロック分周器があります。可能な除数の値は 2～32 で、増分は 2 です。 したがって、フラッシュが実行される最大クロックは IP のクロックの半分です。IP が 100Mhz クロックで実行されている場合、フラッシュのクロックは 50Mhz です。 デフォルトでは、クロックは最も低いクロック(/32)に設定され、ほとんどの場合に IP が機能するようになっています。 除数値: • 0x1: 2 • 0x2: 4 • 0x3: 6 • ... • 0xF: 30 • 0x10: 32
2	CS Delay Setting Register	予約済み		31:8	予約	
		R/W	CS de-assert	7:4	0x0	チップ選択のアサート停止遅延を設定します。

continued...

オフセット(16進数)	レジスター名	R/W	フィールド名	ビット	デフォルト値(16進数)	変更内容
						• 0: チップセレクトは、SCLK の最後の立ち下がりエッジでデアサートされます。 • n:チップセレクトは、SCLK の最後の立ち下がりエッジの n クロック後にアサート解除されます。
		R/W	CS assert	3:0	0x0	チップ・セレクト・アサーション遅延を設定します。 • 0: チップセレクトは、SCLK の最初の立ち上がりエッジの半フラッシュクロック周期前にアサートされます。 • n: チップセレクトは、フラッシュクロック周期の半分に n の IP_CLK を加えたものとしてアサートされます。(3)
3	Read Capturing Register	予約済み		31.4	予約	
		R/W	リード遅延	3:0	0x0	クロックの出力タイミングとボードトレース、I/O ピンのタイミングは、IP ロジックに到達するデータの遅延の大きな値に寄与する可能性があります。遅延キャプチャは、IP が読み取りロジックを遅延させてそれらの遅延を補正する方法を提供します。 IP_CLK サイクルの値だけリードデータロジックを遅延させます。
4	Operating Protocols Setting Register	予約済み		31:18	予約	
		R/W	Read data out transfer mode	17:16	0x0	リードデータ出力の転送モード。
		予約済み		15:14	予約	
		R/W	Read address transfer mode	13:12	0x0	リードアドレス入力の転送モード。ビット 1:0 として記述。
		予約済み		11:10	予約	
		R/W	Write Data in transfer mode	9:8	0x0	ビット 1:0 としてのライトデータ入力説明の転送モード。
		予約済み		7:6	予約	
		R/W	Write address transfer mode	5:4	0x0	ビット 1:0 としてのライトアドレス入力説明の転送モード。
		予約済み		3:2	予約	
		R/W	Instruction transfer mode	1:0	0x0	オペコードの転送モード: • 0x0: Standard SPI mode – コマンド入力は DQ0 で送信される • 0x1: コマンド入力は DQ[1:0]で送信される • 0x2: Quad IO mode – コマンド入力は DQ[3:0]で送信される この設定は、フラッシュ・コマンド・レジスターに影響します。たとえば、このフィールドが 0x1 に設定されている場合、フラッシュ共通動作(たとえば、ID のリード、リードステータス、ライト・ステータス・レジスター)も 0x1 を使用します。

continued...

(3) Intel では、100 MHz で IP クロックを実行している場合、チップセレクトアサーション遅延を 5 に設定することを推奨します。

オフセット(16進数)	レジスター名	R/W	フィールド名	ビット	デフォルト値(16進数)	変更内容
5	Read Instruction Register	予約済み		31:14	予約	
		R/W	Dummy cycles	12:8	0xA	リード動作に使用されるデフォルトのダミーサイクルの数。それぞれのフラッシュデバイスのデータシートを参照してください。
		R/W	Read opcode	7:0	0x03	リード動作のオペコード。転送モード設定に従って正しいオペコードを選択するには、それぞれのフラッシュデバイスのデータシートを参照してください。
6	Write Instruction Register	予約済み		RCLK[31..16]	予約	
		R/W	Polling opcode	RCLK[15..8]	0x05	ライト動作が完了したかどうかを確認するオペコード。ライト動作が完了すると、IP は Avalon-MM インターフェイスの待機要求を解放します。 該当するデバイスでは、ステータスレジスターまたはフラグ・ステータス・レジスターとして設定できます。
		R/W	Write opcode	7:0	0x02	ライト動作のオペコード。転送モード設定に従って正しいオペコードを選択するには、それぞれのフラッシュデバイスのデータシートを参照してください。
7	Flash Command Setting Register(4)	予約済み		31:21	予約	
		R/W	Number of dummy cycles	20:16	0x0	ダミーサイクルの数。動作にダミーサイクルが必要ない場合、0 に設定します。ダミークロックの要件については、それぞれのフラッシュデバイスのデータシートを参照してください。
		R/W	Number of data bytes	15:12	0x08	書き込みまたはリードデータの数。これはビット 11 と連動します。値が 0 に設定されている場合、動作に書き込みまたはリードデータがない場合(書き込み可能など)。
		R/W	Data type	11	0x01	データのタイプを示します(ビット[15:12])。 • 0:[15:12]で宣言されたバイト数は、フラッシュデバイスへのライトデータです。 • 1:[15:12]で宣言されたバイト数は、フラッシュデバイスから読み取られたデータです。
		R/W	Number of address bytes	10:8	0x0	フラッシュデバイスに送信するアドレスバイト数(3または 4 バイト)。 これがゼロに設定されている場合、動作はアドレスバイトを伝送しません。
		R/W	Opcode	7:0	0x05	動作のオペコード。
8	Flash Command Control Register	予約済み		31:1	予約	
		W	Start	0	0x0	動作を開始するには、このビットに 1 を書き込みます。
9	Flash Command Address Register	R/W	Stating address	31:0	31:0	フラッシュコマンドのアドレス。

continued...

(4) デフォルト設定は、リード・ステータス・コマンド用です。

オフセット(16進数)	レジスター名	R/W	フィールド名	ビット	デフォルト値(16進数)	変更内容
A	Flash Command Write Data 0 Register	R/W	Lower 4 bytes write data	RCLK[31..0]	0x0	フラッシュデバイスへのライトデータの最初の 4 バイト。
B	Flash Command Write Data 1 Register	R/W	Upper 4 bytes write data	RCLK[31..0]	0x0	フラッシュデバイスへのライトデータの最後の 4 バイト。
C	Flash Command Read Data 0 Register	R	Lower 4 bytes read data	31:0	0x0	フラッシュデバイスからのリードデータの最初の 4 バイト。
D	Flash Command Read Data 1 Register	R	Upper 4 bytes read data	31:0	0x0	フラッシュデバイスからのリードデータの最後の 4 バイト。

1.5. 汎用シリアル・フラッシュ・インターフェイス IP の使用

コア・インターフェイスは Avalon -MM 準拠です。詳細については、Avalon 仕様を参照してください。

1.5.1. コントロール・ステータス・レジスター動作

コントロール・ステータス・レジスター(CSR)を使用して、特定のアドレスオフセットに対して読み取りまたは書き込みを実行できます。

コントロール・ステータス・レジスターのライトまたはリード動作を実行するには、次の手順を実行します。

1. `avl_csr_waitrequest` 信号が低い間、`avl_csr_write` または `avl_csr_read` 信号をアサートします(`waitrequest` 信号が高い場合、`waitrequest` 信号が低くなるまで `avl_csr_write` または `avl_csr_read` 信号を高く維持する必要がある)。
2. 同時に、`avl_csr_address` バスにアドレス値を設定します。ライト動作の場合、アドレスとともに `avl_csr_writedata` バスに値データを設定します。
3. リード・トランザクションの場合、`avl_csr_readdatavalid` 信号が High にアサートされるまで待機して、リードデータを取得します。

- フラッシュへの書き込み値を必要とする動作の場合、最初にライトイネーブル動作を実行する必要があります。
- 書き込みまたは消去コマンドを発行するたびに、フラグ・ステータス・レジスターを読み取る必要があります。
- 複数のフラッシュデバイスをサポートする場合、特定のフラッシュデバイスに対して動作を実行する前に、チップセレクトレジスターを書き込んで正しいフラッシュデバイスを選択する必要があります。

1.5.2. メモリー動作

フラッシュ・メモリー・アクセス中に、IP コアは次の手順を実行して、直接リードまたはライト動作を実行できるようにします。

- ライト動作のためのライトイネーブル
- フラグ・ステータス・レジスターをチェックして、フラッシュで動作が完了したことを確認します。
- 動作が完了したら、`waitrequest` シグナルを解放します。

メモリーー動作は Avalon -MM 動作に似ています。アドレスバスに正しい値を設定し、ライト・トランザクションの場合はデータを書き込み、単一トランザクションの場合はバーストカウントバス 1 を駆動し、目的のバーストカウント値を書き込み、ライトまたはリード信号をトリガーする必要があります。

注意: 複数のフラッシュデバイスをセットアップする場合、アドレスバスはチップセレクト値を含むように拡張されます。

図 -2: 8 ワードのライトバースト波形の例

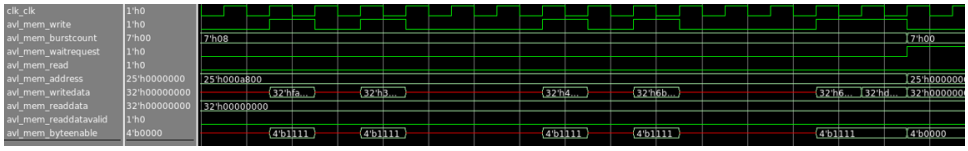


図 -3: 8 ワードのリードバースト波形の例

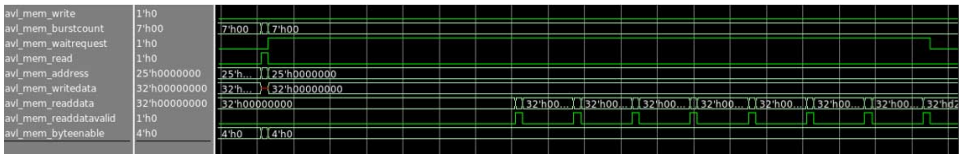
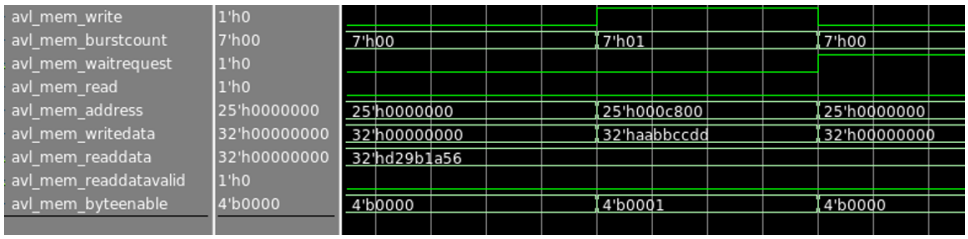


図 -4: 1 バイトのライト `byteenable = 4'b0001` 波形の例



注意: Intel Quartus® Prime プロ・エディションソフトウェアでデザインをコンパイルすると、Generic Flash Serial Interface Intel FPGA IP コアに 2 つの内部制約のないクロックがあります。Intel は次のコマンドを使用してパスを制限することを推奨します。

```
create_generated_clock -name <name_of_generated_clock> -source [get_ports
<input_clock_name>] -divide_by 2 [get_registers
<path_of_the_unconstrained_path>]
```

1.6. 汎用シリアル・フラッシュ・インターフェイス Intel FPGA IP コアおよびリファレンス・デザイン

リファレンス・デザインは、汎用シリアル・フラッシュ・インターフェイス Intel FPGA IP 次の汎用メモリー動作を実行します。

- デバイス ID の読み出し
- セクター保護のイネーブル
- セクター消去の実行
- フラッシュデバイスとの間でデータの読み出しと書き込み

関連情報

- Intel FPGA IP FPGA コンフィグレーションのユーザー・ガイド (3 ページ)
- 汎用シリアル・フラッシュ・インターフェイスのインテル FPGA IP コアのリファレンス・デザイン・ファイル

1.6.1. ハードウェアおよびソフトウェア要件

デザイン例のハードウェアおよびソフトウェア要件は次のとおりです。

- Cyclone V E FPGA Development Kit
- インテル Quartus Prime software version 18.0 with Nios® II Software Build Tools for Eclipse
- インテル FPGA ダウンロード・ケーブル II
- Tested flash devices:
  - Cypress* S70FL01G
  - Micron MT25Q01G
  - Micron MT25Q512
  - EPCQ256

1.6.2. 機能の説明

1.6.2.1. リファレンス・デザインのコンポーネント

図 -5: デザインのブロック図

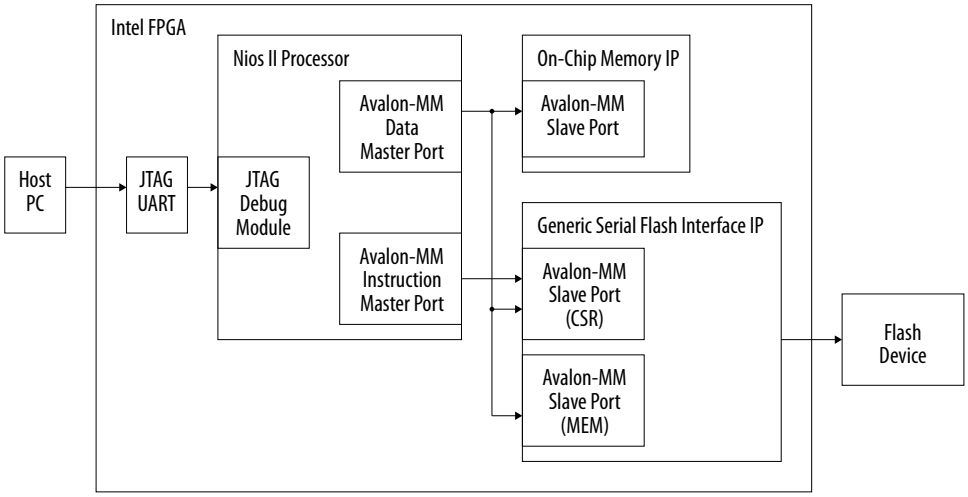
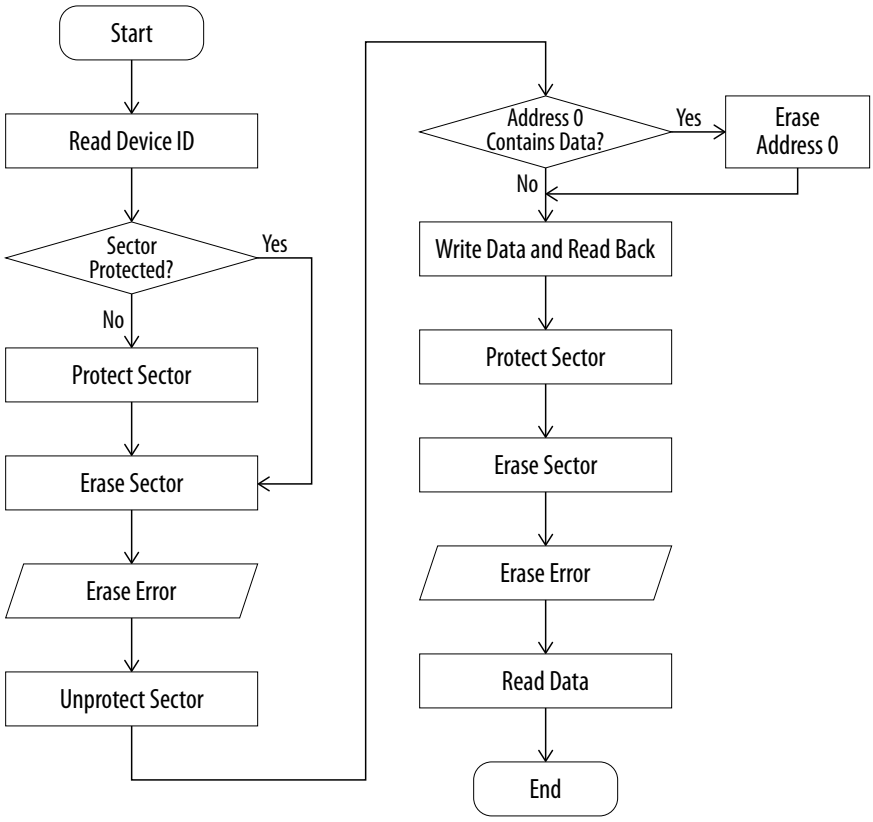


表 4. リファレンス・デザインのコンポーネントの説明

コンポーネント	説明
JTAG UART Intel FPGA IP	Nios II プロセッサーとホスト・コンピューター間の通信をイネーブルします。
Nios II Processor	データと命令を実行してアプリケーション・プログラムを実行します。
On-Chip Memory Intel FPGA IP	コードおよびデータをストアします。
汎用シリアル・フラッシュ・インターフェイス Intel FPGA IP	ベンダーに依存しないフラッシュデバイスを制御してフラッシュ・インタラクションを実行します。

1.6.2.2. リファレンス・デザイン・アプリケーションのプログラム

図 -6: リファレンス・デザイン・アプリケーションのプログラムのフロー図



フロー・ダイアグラム・シーケンスの説明:

1. アプリケーション・プログラムは、FPGA に接続されているフラッシュデバイスを識別することから始まります。

   注意: フラッシュデバイスは、このリファレンス・デザインのみを示すサンプルとして機能します。

2. アプリケーション・プログラムは、セクター保護を実行し 、保護されたセクターを消去します。
   a. セクター保護を実行するには、アプリケーション・プログラムは:
      i. ライト・イネーブル・コマンドを実行します。
      ii. ライト・ステータス・レジスター・コマンドを実行して、ブロック保護(BP)ビットとトップ/ボトム(TB)ビットを設定します。
      iii. 0(準備完了)を返すまで、書き込み中(WIP)ビット(ステータスレジスターのビット 0)をポーリングします。
      iv. ライト・ステータス・レジスター・コマンドを実行して、セクター保護動作が成功したか失敗したかを確認します。
   b. セクター消去を実行するために、アプリケーション・プログラムは以下を実行します。

i. セクター消去コマンドを実行します。
ii. 0(準備完了)を返すまで、書き込み中(WIP)ビット(ステータスレジスターのビット 0)をポーリングします。
iii. ステータスレジスターの読み取りを実行して、消去動作が成功したか失敗したかを確認します。
iv. セクターが保護されているため、消去エラーが発生します。

3. アプリケーション・プログラムは、エラーフラグをクリアします。
   - フラグ・ステータス・レジスターのクリアコマンド(EPCQ-L または Micron)。
   - ステータスレジスターのクリアコマンド(Cypress)。
4. アプリケーション・プログラムは、セクター保護を無効にします。
   a. 書き込み可能コマンドを実行します。
   b. ステータスレジスター書き込みコマンドを実行して、BP ビットと TB ビットをクリアします。
   c. 0(準備完了)を返すまで、WIP ビット(ステータスレジスターのビット 0)をポーリングします。
   d. ステータスレジスターのリードコマンドを実行して、BP ビットと TB ビットがクリアされたかどうかを確認します。
5. セクターが保護されていない場合、アプリケーション・プログラムはフラッシュデバイスのプログラミングを実行します。アプリケーション・プログラム:
   a. メモリーが空のアドレスにメモリーを書き込みます。
   b. WIP ビット(ステータスレジスターのビット 0)をポーリングし、0(準備完了)が返されるまでアドレスのメモリーをリードバックし、アドレスがプログラムされたことを確認します。
   c.
6. 手順 2 を繰り返し、アドレスのメモリーを読み戻します。セクターが保護されているため、メモリーは消去されません。

1.6.3. Nios II ハードウェアシステムの作成

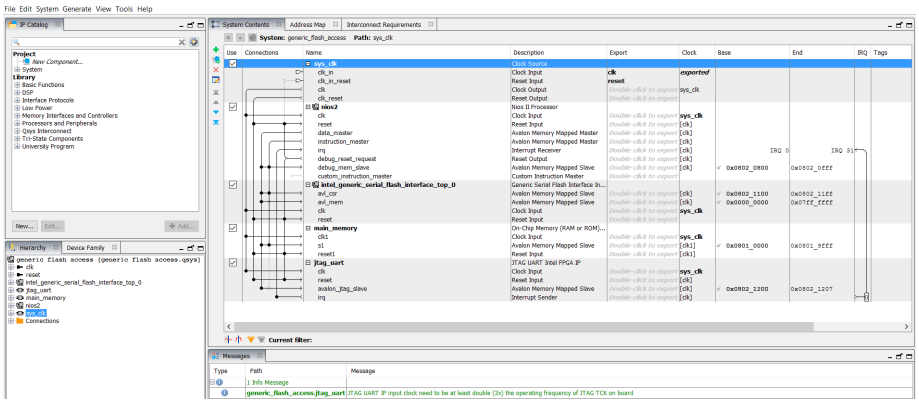
1. インテル Quartus Prime ソフトウェアでは、**File** > **New Project Wizard** に移動します。
2. 新しいを作成します インテル Quartus Prime 新しいディレクトリにある `generic_flash_access` という Prime プロジェクトで、**Cyclone V E 5CEFA7F3117** デバイスを選択します。
3. **Tools** > **Platform Designer** を選択し、ファイルを `generic_flash_access.qsys` として保存します。
4. クロックソース `clk_0` をダブルクリックし、クロック**周波数**を **100000000 Hz** (100Mhz)に変更します。
5. `clk_0` を右クリックし、名前を `sys_clk` に変更します 。
6. Nios II プロセッサーを追加します。
   a. **Processor and Peripherals** > **Embedded Processors** > **Nios II Processor** に移動し、**Add** をクリックします。
   b. **Finish** をクリックして Nios II プロセッサーをデザインに追加し、名前を `nios2` に変更します 。

   注意: まだ指定されていないパラメーターに関するメッセージは無視してください。
7. 汎用シリアル・フラッシュ・インターフェイス IP を追加します。

a. **Basic Functions** > **Configuration and Programming** > **Generic Serial Flash Interface Intel FPGA IP** を選択し、**Add** をクリックします。このコンポーネントの名前を `intel_generic_serial_flash_interface_top0` に変更します。
b. デバイス密度を設定します。
   注意: このリファレンス・デザインでは、1024MB のフラッシュデバイス密度を使用しています。
c. `avl_mem` と `avl_csr` にプロセッサーの `data_master` を接続します。そして、このコンポーネントの唯一 `avl_csr` にプロセッサーの `instruction_master` を接続します。

8. オンチップメモリーを追加します。
   a. **Basic Functions** > **On Chip Memory** > **On-Chip Memory (RAM or ROM) Intel FPGA IP** を選択します。
   b. **Total memory size** を **40960 Bytes** に設定します。
   c. **Finish** をクリックし、`main_memory` に名前を変更します。
   d. スレーブをプロセッサーの `data_master` および `instruction_master` に接続します。
9. JTAG UART を追加します。
   a. **Interface Protocols** > **Serial** > **JTAG UART Intel FPGA IP** に移動し **Add** をクリックします。
   b. **Finish** をクリックして、名前を `jtag_uart` に変更します。
   c. その `avalon_jtag_slave` ポートは、プロセッサーの `data_master` ポートに接続します。
   d. **IRQ** カラムで、割り込み送信側ポートを `Avalon_jtag_slave` ポートからプロセッサーの割り込み受信側ポートに接続し、`0` を入力します。
10. `sys_clk` のクロック入力を他のすべてのコンポーネントのクロック入力に接続します。
11. Platform Designer システムを生成する前にすべての Nios II プロセッサーのエラーメッセージを解決します。
    a. Nios II プロセッサー `nios2` をダブルクリックします。
    b. **Vectors** をクリックし、**Reset vector memory** と **Exception vector memory** の両方を **main_memory.s1** に変更します。
    c. **System** タブをクリックし、ドロップ・ダウン・メニューの **System** をクリックして、すべてのコンポーネントの自動割り当てのベースアドレスに **Assign Base Address** をクリックしてください。
    d. 同じメニューで、**Create Global Reset Network** をクリックしてリセット信号を接続し、グローバル・リセット・ネットワークを形成します。

図 -7: 完成しました Platform Designer 接続



12. システムを生成します。
    a. ウィンドウの下部にある **Generate HDL** をクリックします。
    b. 完了すると、Platform Designer は `Generate:Completed successfully` と表示されます。

1.6.4. インテル Quartus Prime プロジェクトへのモジュールの統合

1. インテル Quartus Prime ソフトウェアで、**Assignment** > **Settings** を選択します。
2. **Settings** ウィンドウで、合成フォルダーにある `generic_flash_access.qys` ファイルを追加し、**Apply** をクリックします。
3. `generic_flash_access.qys` ファイルは、`Files` ディレクトリの下に表示されます。ファイルを右クリックし、**Set as Top-Level Entity** を選択します。
4. ハードウェアシステムが入力ピンと出力ピンを決定できるようにするには、**Processing** > **Start** > **Start Analysis and Elaboration** に進みます。
5. **Assignments** > **Pin Planner** に移動してピン割り当てを開始し、`PIN_L14` は `clk_clk` として、`PIN_AA26` は `reset_reset_n` として割り当てます。
6. **Assignments** > **Device** > **Device and Pin Options** > **Configuration** に移動し、**Configuration scheme** を **Active Serial x1** に変更します。
7. ハードウェアシステムの完全なコンパイルを実行するには、**Processing** > **Start** > **Start Analysis and Synthesis** に進みます。

1.6.5. .sof ファイルのプログラミング

1. インテル Quartus Prime Programmer では、**Hardware setup** をクリックし、FPGA を接続する正しい USB チェーンを選択します。
2. **Auto Detect** をクリックすると、5CEFA7F31 が表示され、ファイルを `top.sof` に変更します。
3. **Enable Program/ Configure** そして **Start** をクリックします。

1.6.6. Nios II Software Build Tools を使用したアプリケーション・ソフトウェア・システムの構築

1. インテル Quartus Prime で **Tools** > **Nios II Software Build Tools for Eclipse** に進みます。
2. ワークスペース・ディレクトリーを参照します。
3. **Nios II Software Build Tools for Eclipse** で、**File** > **New** > **Nios II Application and BSP from Template** に移動します。
4. **SOPC Information File name** フィールドで、プロジェクト・ディレクトリーから `generic_flash_access.sopcinfo` を選択し、**Open** をクリックします。
5. **Project Name** で `generic_flash_access` に設定し、`Hello World Small` プロジェクトテンプレートを選択して、**Finish** をクリックします。
6. `generic_flash_access` プロジェクト・ディレクトリーで、`hello_world_small.c` ファイルを、リファレンス・デザインに添付されている `main.c` および `operation.c` ファイルに置き換えます。
7. `main.c` ファイルを選択し **Project** > **Build Project to create the generic_flash_access.elf** ファイルに進みます。
8. `generic_flash_access.elf` ファイルを選択し **Run** > **Run As** > **Nios II Hardware** に進みます。
9. **Nios II Console** は、次の結果を出力します。

1.6.6.1. リファレンス・デザインのインストール

Cypress S70FL01G:

```
Flash Device: Cypress flash S70FL01G
Device ID: 4d210201
All sectors in this flash device is not protected
Now performing sector protection...
All sectors in this flash device is now successfully protected
Trying to erase sector 0...
ERASE ERROR as sector is protected!
Now perform sector unprotect...
Sector unprotect successfully! :)
Reading data at address 0...
Memory content at address 0: abcd1234
Trying to erase sector 0...
Sector erase successfully. Sector 0 is now empty.
Writing data to address 0...
Read back data from address 0...
Current memory in address 0: abcd1234
Read data match with data written. Write memory successful.Now performing
sector protection...
All sectors in this flash device is now successfully protected
Trying to erase sector 0...
ERASE ERROR as sector is protected!
Current memory in address 0: abcd1234
Read data match with data written previously. Sector erase does not perform
during sector is protected.
```

Micron MT25Q01G:

```
Flash Device: Micron flash MT25Q01G
Device ID: 1021ba20
All sectors in this flash device is not protected
Now performing sector protection...
All sectors in this flash device is now successfully protected
Trying to erase sector 0...
```

```
Erase Error as erase is not allow during sector is protected!
Now perform sector unprotect...
Sector unprotect successfully! :)
Reading data at address 0...
Memory content at address 0: abcd1234
Address 0 containing data, it is not empty.
Trying to erase sector 0...
Sector erase successfully. Sector 0 is now empty.
Memory not containing data...
Writing data to address 0...
Read back data from address 0...
Current memory in address 0: abcd1234
Read data match with data written. Write memory successful.
Now performing sector protection...
All sectors in this flash device is now successfully protected
Trying to erase sector 0...
ERASE ERROR as sector is protected!
Current memory in address 0: abcd1234
Read data match with data written previously. Sector erase does not perform
during sector is protected.
```

Micron MT25Q512:

```
Flash Device: Micron flash MT25Q512
Device ID: 1020ba20
All sectors in this flash device is not protected
Now performing sector protection...
All sectors in this flash device is now successfully protected
Trying to erase sector 0...
Erase Error as erase is not allow during sector is protected!
Now perform sector unprotect...
Sector unprotect successfully! :)
Reading data at address 0...
Memory content at address 0: abcd1234
Address 0 containing data, it is not empty.
Trying to erase sector 0...
Sector erase successfully. Sector 0 is now empty.
Memory not containing data...
Writing data to address 0...
Read back data from address 0...
Current memory in address 0: abcd1234
Read data match with data written. Write memory successful.
Now performing sector protection...
All sectors in this flash device is now successfully protected
Trying to erase sector 0...
ERASE ERROR as sector is protected!
Current memory in address 0: abcd1234
Read data match with data written previously. Sector erase does not perform
during sector is protected.
```

EPCQ256:

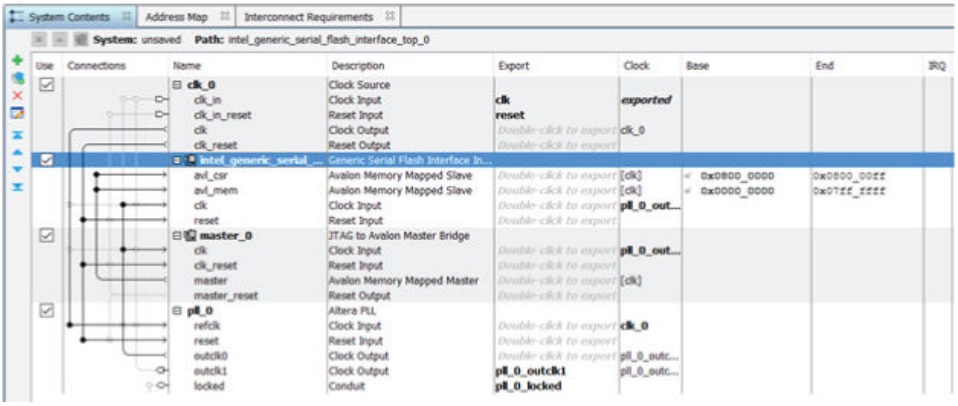
```
Flash Device: EPCQ256
Device ID: 1019ba20
All sectors in this flash device is not protected
Now performing sector protection...
All sectors in this flash device is now successfully protected
Trying to erase sector 0...
Erase Error as erase is not allow during sector is protected!
Now perform sector unprotect...
Sector unprotect successfully! :)
Reading data at address 0...
Memory content at address 0: abcd1234
Address 0 containing data, it is not empty.
Trying to erase sector 0...
Sector erase successfully. Sector 0 is now empty.
Memory not containing data...
Writing data to address 0...
Read back data from address 0...
Current memory in address 0: abcd1234
Read data match with data written. Write memory successful.
```

```
Now performing sector protection...
All sectors in this flash device is now successfully protected
Trying to erase sector 0...
ERASE ERROR as sector is protected!
Current memory in address 0: abcd1234
Read data match with data written previously. Sector erase does not perform
during sector is protected.
```

1.7. 汎用シリアル・フラッシュ・インターフェイス Intel FPGA IP コアを使用したフラッシュアクセス

このセクションでは、この Intel FPGA IP コアのレジスタを使用してフラッシュアクセスを実行する方法について説明します。最初に、以下に示すように、いくつかのコンポーネント(`clock`、`jtag master`、`pll`、およびこの Intel FPGA IP コア)で Platform Designer システムを構築します。次に、次の例のフラッシュ動作を使用します。

図 -8: **汎用シリアル・フラッシュ・インターフェイス Intel FPGA IP コアを使用して Flash アクセスを作成する例**



注意: FPGA デバイスの MSEL ピンを AS コンフィギュレーションモードに設定する必要があります。インテル MAX 10 デバイスの場合、この Intel FPGA IP コアの **Enable SPI Pins Interface** パラメーターをイネーブルする必要があります。

フラッシュ動作はいくつかのカテゴリに分類されます。動作の例、使用するレジスター、および各カテゴリのサンプル `.tcl` スクリプトが提供されています。

1.7.1. 動作コードを必要とするフラッシュ動作

次のフラッシュ動作には、動作コードが必要です。

- Write enable
- Enter 4-byte addressing mode
- Exit 4-byte addressing mode
- Clear flag status register
- Clear status register

以下のレジスターは、動作コードを必要とする動作に使用されます。

- Flash command setting register
- Flash command control register

例-1: フラッシュのライトイネーブル動作を実行する

```
proc write_enable { } {
global mp flash_cmd_setting flash_cmd_ctrl flash_cmd_write_data_0
master_write_32 $mp $flash_cmd_setting 0x00000006
master_write_32 $mp $flash_cmd_ctrl 0x1
}
```

フラッシュのライトイネーブル動作を実行するには、次の手順を実行します。

1. グローバル変数を定義します。
2. フラッシュコマンド設定レジスターに書き込むことにより、ライトイネーブル動作をカスタマイズします。
   a. 06h はライトイネーブル動作の動作コードであるため、このレジスターのビット[7:0]を 06 に設定します。
3. フラッシュコマンド制御レジスターのビット 0 に 1 を書き込んで、ライトイネーブル動作を開始します。

1.7.2. フラッシュレジスターを読み出すフラッシュ動作

フラッシュレジスターを読み出すには、次のフラッシュ動作が使用されます。

- Read device ID
- Read status register
- Read flag status register
- Read configuration register
- Read bank register
- Read enhanced volatile configuration register

次のレジスターは、レジスターのステータスを読み取るために使用されます。

- Flash command setting register
- Flash command control register
- Flash command read data 0 register

例-2: デバイス ID のリード動作を実行する

```
proc read_device_id { } {
global mp flash_cmd_setting flash_cmd_ctrl flash_cmd_read_data_0
master_write_32 $mp $flash_cmd_setting 0x000048 9F
master_write_32 $mp $flash_cmd_ctrl 0x1
set device_id [master_read_32 $mp $flash_cmd_read_data_0 1]
puts $device_id
}
```

リードデバイス ID 動作を行うには、以下のステップを実行します。

1. グローバル変数を定義します。
2. フラッシュコマンド設定レジスターに書き込むことにより、リードデバイス ID 動作をカスタマイズします。
   a. 9Fh はリードデバイス ID 動作の動作コードであるため、このレジスタのビット[7:0]を 9F に設定します。
   b. この動作はアドレスバイトを伝送しないため、ビット[10:8]を 0 に設定します。
   c. ビット[15:12]で宣言されたバイト数がフラッシュデバイスからのリードデータであるため、ビット 11 を 1 に設定します。
   d. フラッシュから 4 バイトのデバイス ID データを読み出すため、ビット[15:12]を 4 に設定します。
3. フラッシュコマンド制御レジスターのビット 0 に 1 を書き込むと、デバイス ID のリード動作が開始されます。
4. フラッシュコマンドのリードデータ 0 レジスターからデバイス ID を読み出します。

1.7.3. フラッシュレジスターを書き込むフラッシュ動作

フラッシュレジスターの書き込みには、次のフラッシュ動作が使用されます。

- Write enhanced volatile configuration register
- Write bank register
- Write status register
- Write configuration register

注意: これらの動作を開始する前に、ライトイネーブル動作を実行する必要があります。

次のレジスターは、レジスターのステータスを書き込むために使用されます。

- Flash command setting register
- Flash command control register
- Flash command write data 0 register

例-3: フラッシュのセクターを保護するためのステータスレジスターライト動作の実行

```
proc write_status_register { } {
 global mp flash_cmd_setting flash_cmd_write_data_0 flash_cmd_ctrl
master_write_32 $mp $flash_cmd_setting 0x00001001
master_write_32 $mp $flash_cmd_write_data_0 0x0000007c
master_write_32 $mp $flash_cmd_ctrl 0x1
}
```

ライト・ステータス・レジスター動作を実行するには、以下の手順に従います。

1. グローバル変数を定義します。
2. フラッシュコマンド設定レジスターに書き込むことにより、ステータスレジスターの書き込み動作をカスタマイズします。
   a. 01h はライト・ステータス・レジスター動作の動作コードであるため、このレジスターのビット[7:0]を 01 に設定します。
   b. この動作はアドレスバイトを伝送しないため、ビット[10:8]]を 0 に設定します。
   c. ビット[15:12]で宣言されたバイト数がフラッシュデバイスへのライトデータであるため、ビット 11 を 0 に設定します。
   d. 1 バイト(8 ビット)のデータをステータス・レジスターに書き込むため、ビット[15:12]を 1 に設定します。
3. フラッシュコマンドのライトデータ 0 レジスターにセクター保護を設定するデータを書き込みます。
   a. ステータスレジスターのビット 6 およびビット[4:2]はブロック保護ビットで、ビット 5 はトップ/ボトムビットです。この例では、メモリーアレイの下からのすべてのセクターに保護が必要です。詳細については、それぞれのフラッシュデータシートを参照してください。
4. フラッシュコマンド制御レジスターのビット 0 に 1 を書き込むと、セクター保護動作の書き込みステータスレジスターが開始されます。

1.7.4. アドレスが必要なフラッシュ動作

次のフラッシュ動作にはアドレスが必要です。

- Sector erase
- Bulk erase
- Die erase

注意: これらの動作を開始する前に、ライトイネーブル動作を実行する必要があります。

次のレジスターは、アドレスを必要とする動作に使用されます。

- Flash command setting register
- Flash command control register
- Flash command address register

例-4: フラッシュセクターの消去動作を実行する

```
proc erase_sector { } {
global mp flash_cmd_setting flash_cmd_ctrl flash_cmd_addr_register
master_write_32 $mp $flash_cmd_setting 0x000004D8
master_write_32 $mp $flash_cmd_addr_register 0x00001000
master_write_32 $mp $flash_cmd_ctrl 0x1
}
```

セクター消去ド動作を実行するには、以下の手順に従います。

1. グローバル変数を定義します。
2. フラッシュコマンド設定レジスターに書き込むことにより、セクター消去動作をカスタマイズします。
   a. D8h はセクター消去動作の動作コードであるため、このレジスターのビット[7:0]を D8 に設定します。
   b. 4 バイトのアドレスがフラッシュデバイスに送信されるため、ビット[10:8]を 4 に設定します。
   c. ビット[15:12]で宣言されたバイト数がフラッシュデバイスへのライトデータであるため、ビット 11 を 0 に設定します。
3. 消去するセクター内のアドレスを指定して、フラッシュ・コマンド・アドレス・レジスターに書き込みます。
   a. この例では、アドレス 00001000 に対してセクター消去動作を実行しています。
4. フラッシュコマンド制御レジスターのビット 0 に 1 を書き込んで、セクター消去動作を開始します。

この Intel FPGA IP コアは、拡張、デュアル、およびクアッド I/O プロトコルでフラッシュをサポートします。現在、これでサポートされているプロトコル Intel FPGA IP コアは単一転送速度(STR)のみです。この Intel FPGA IP コアは、3 バイトと 4 バイトの両方のアドレス指定モードをサポートしています。以下のセクションでは、メモリーおよびプログラム動作を読み取るためのさまざまなプロトコルとアドレス指定モードについて説明します。

1.7.5. フラッシュからメモリーを読み取る

リード・メモリーの実行には、次のレジスターが使用されます。

- Operating protocols setting register
- Control register
- Read instruction register

例-5: Perform the Read Memory (Extended Mode)

```
proc read { } {
global mp operating_protocols_setting control_register read_instr
master_write_32 $mp $operating_protocols_setting 0x00000000
master_write_32 $mp $control_register 0x00000001
master_write_32 $mp $read_instr 0x00000003
master_read_32 $mp 0x0100000 0x1
}
```

拡張モードのリード・メモリーを実行するには、次の手順を実行します。

1. グローバル変数を定義します。
2. 動作プロトコル設定レジスターに書き込み、メモリー読み出し動作の転送モードを設定します。この例では、読み取りの転送モードは(1-1-1)です。
   a. 命令転送モード[1:0]を 0 に設定し 、リードアドレス転送モード[13:12]を 0 に設定し、読み出しデータ転送モード[17:16]を 0 に設定します。
3. 制御レジスターに書き込み、リード・メモリー動作のバイトアドレッシング・モードを選択します。
   a. この例では、3 バイトのアドレス指定モードを使用しています。ビット 8 を 0 に設定します 。
4. 読み取り命令レジスターに書き込み、リード・メモリー動作をカスタマイズします。
   a. 03h は読み取り用の動作コードであるため、リード動作コード[7:0]を 03 に設定します。
   b. リード動作にはダミーサイクルが含まれないため、ダミーサイクル[12:8]を 0 に設定します。
5. レジスターを設定した後、アドレスのメモリー内容の読み取りを実行できます。
   a. この例では、1 ワードのデータがアドレス 0x01000000 から読み取られます。

例-6: デュアル出力高速読み出しの実行(デュアル SPI モード)

```
proc dual_output_fast_read { } {
global mp operating_protocols_setting control_register read_instr
master_write_32 $mp $operating_protocols_setting 0x00011001
master_write_32 $mp $control_register 0x00000101
master_write_32 $mp $read_instr 0x00000A3B
master_read_32 $mp 0x00000100 0x1
}
```

デュアル出力高速読み取りモードを実行するには、次の手順を実行します。

1. グローバル変数を定義します。
2. 動作プロトコル設定レジスターに書き込み、メモリー読み出し動作の転送モードを設定します。この例では、読み取りの転送モードは(2-2-2)です。

a. 命令転送モード[1:0]を 1 に 、アドレス転送モード[13:12]を 1 に 、データ転送モード[17:16]を 1 に設定します。

3. 制御レジスターに書き込み、リード・メモリー動作のバイトアドレッシング・モードを選択します。
   a. この例では、4 バイトのアドレス指定モードを使用しています。ビット 8 を 1 に設定します。
4. 読み取り命令レジスターに書き込み、リード・メモリー動作をカスタマイズします。
   a. 3Bh はデュアル出力高速読み取りの動作コードであるため、リード動作コード[7:0]を 3B に設定します。
   b. デュアル出力高速リード動作には 10 ダミーサイクルが含まれるため、ダミーサイクル[12:8]を A に設定します。
5. レジスターを設定した後、アドレスのデュアル出力高速リード・メモリーコンテンツを実行できます。
   a. この例では、メモリーの内容はアドレス 0x00000100 から読み取られます。

1.7.6. プログラムフラッシュ

次のレジスターは、プログラムフラッシュの実行に使用されます。

- Operating protocols setting
- Control register
- Write instruction

例-7: ページプログラムの実行(拡張モード)

```
proc page_program { } {
global mp operating_protocols_setting control_register write_instr
master_write_32 $mp $operating_protocols_setting 0x00000000
master_write_32 $mp $control_register 0x00000001
master_write_32 $mp $write_instr 0x00007002
master_write_32 $mp 0x00001000 0x1234abcd
 }
```

拡張モードのページプログラムを実行するには、次の手順を実行します。

1. グローバル変数を定義します。
2. 動作プロトコル設定レジスターに書き込み、プログラム動作の転送モードを設定します。この例では、読み取りの転送モードは(1-1-1)です。
   a. 命令転送モード[1:0]を 1 に 、アドレス転送モード[5:4]を 1 に 、転送モード[9:8]でデータを 1 に設定します。
3. 制御レジスターに書き込み、ライト動作のバイトアドレッシング・モードを選択します。
   a. この例では、3 バイトのアドレス指定モードを使用しています。ビット 8 を 0 に設定します。
4. 書き込み命令レジスターに書き込み、プログラムの動作をカスタマイズします。
   a. 02h はページプログラムのオペレーションコードであるため、ライト・オペレーション・コード[7:0]を 02 に設定します。

a. 70h はリード・フラグ・ステータス・レジスターの動作コードであるため、ポーリング動作コード [15:8]を 70 に設定します。ライト動作が完了すると、Intel FPGA IP コアは Avalon – MM インターフェイスの待機要求を解放します。フラッシュにフラグステータスレジスターを読み込むには、リード・ステータス・レジスター(05h)を使用できます。

5. レジスターを設定した後、メモリーをアドレスにプログラムし始めることができます。
   a. この例では、1234abcdh がメモリーアドレス 0x00001000 に書き込まれます。

例-8: 4 バイトのクワッド入力高速プログラムを実行(クワッド SPI モード)

```
proc fourbyte_quad_input_fast_program { } {

global mp operating_protocols_setting control_register write_instr

master_write_32 $mp $operating_protocols_setting 0x00000222

master_write_32 $mp $control_register 0x00000101

master_write_32 $mp $write_instr 0x00007034

master_write_32 $mp 0x00002000 0xabcd1234

}
```

4 バイトのクワッド入力高速プログラムを実行するには、次の手順を実行します。

1. グローバル変数を定義します。
2. 動作プロトコル設定レジスターに書き込み、プログラム動作の転送モードを設定します。この例では、4 バイトのクワッド入力高速プログラムの転送モードは(4-4-4)です。
   a. 命令転送モード[1:0]を 2 に 、書き込みアドレス転送モード[5:4]を 2 に 、転送モード[9:8]でデータを 2 に設定します。
3. 制御レジスターに書き込み、ライト動作のバイトアドレッシング・モードを選択します。
   a. この例では、4 バイトのアドレス指定モードを使用しています。ビット 8 を 1 に設定します。
4. 書き込み命令レジスターに書き込み、プログラムの動作をカスタマイズします。
   a. 34h は 4 バイトのクワッド入力高速プログラムのオペレーションコードであるため、ライト・オペレーション・コード[7:0]を 34 に設定します。
   b. 70h はリード・フラグ・ステータス・レジスターの動作コードであるため、ポーリング動作コード [15:8]を 70 に設定します。ライト動作が完了すると、Intel FPGA IP コアは Avalon - MM インターフェイスの待機要求を解放します。フラッシュにフラグステータスレジスターを読み込むには、リード・ステータス・レジスター(05h)を使用できます。
5. レジスターを設定した後、メモリーをアドレスにプログラムし始めることができます。
   a. この例では、1234abcdh がメモリーアドレス 0x00002000 に書き込まれます。

1.8. 汎用シリアル・フラッシュ・インターフェイス Intel FPGA IP コアのユーザーガイドのアーカイブ

IP バージョンは、v19.1 までの インテル Quartus Prime デザインスイートソフトウェア・バージョンと同じです。インテル Quartus Prime デザインスイートソフトウェア・バージョン 19.2 以降では、IP コアに新しい IP バージョン管理スキームがあります。

IP コアのバージョンが記載されていない場合には、以前の IP コアバージョン向けのユーザーガイドが当てはまります。

IP コアバージョン	ユーザーガイド
18.1	汎用シリアル・フラッシュ・インターフェイス Intel FPGA IP コアのユーザー・ガイド
18.0	汎用シリアル・フラッシュ・インターフェイス Intel FPGA IP コアのユーザー・ガイド

1.9. 汎用シリアル・フラッシュ・インターフェイス Intel FPGA IP コアのユーザーガイドの改訂履歴

ドキュメント・バージョン	インテル Quartus Prime バージョン	IP のバージョン	変更内容
2019.11.27	19.3	19.1	*メモリー動作*のトピックに説明を追加。
2019.09.30	19.3	19.1	• インテル Agilex デバイスのサポートを追加。 • *デバイス・ファミリー・サポート*のトピックを更新。 • ドキュメント全体を通したテキストのマイナー・チェンジ
2018.11.09	18.1	18.1	• *汎用シリアル・フラッシュ・インターフェイス Intel FPGA IP コアを使用するフラッシュアクセス*のセクションを追加。 • *汎用シリアル・フラッシュ・インターフェイス Intel FPGA IP コアのユーザーガイドのアーカイブ*のセクションを追加。 • *汎用シリアル・フラッシュ・インターフェイス Intel FPGA IP コアのユーザーガイド*のセクションを更新し、汎用シリアル・フラッシュ・インターフェイス Intel FPGA IP コアの詳細を提供。 • *信号ブロック図の図*の信号名を更新。 • *ポートの説明*の表のコンジット・インターフェイス信号名を更新。 • *レジスターマップ*のテーブルの書き込み命令レジスターのライト・オペコード・フィールド名の説明を更新。
2018.05.16	18.0	18.0	• *ジェネリックシリアル・フラッシュ・インターフェイスのインテル FPGA IP コアリファレンス・デザイン・ファイル*のリンクを更新。 • *レジスターマップ*に フラッシュ・コマンド・アドレス・レジスターを追加。
2018.05.07	18.0	18.0	初版。