

この資料は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。こちらの日本語版は参考用としてご利用ください。設計の際には、最新の英語版で内容をご確認ください。

Q1155002-8.0.0

コアの概要

オンチップ FIFO メモリ・コアは、データのバッファリングおよびフロー・コントロールを提供するコンフィギュレーション可能なコンポーネントです。FIFO の入出力ポートクロックは、1 つの共通クロックとするか、もしくは 2 つの個別クロックとするかの選択が可能です。

FIFO への入力インタフェースは、Avalon® Memory-Mapped (Avalon-MM) ライト・スレーブまたは Avalon Streaming (Avalon-ST) シンクのいずれかを使用できます。出力インタフェースは、Avalon-ST ソースまたは Avalon-MM リード・スレーブのいずれかです。チャンネル信号、パケット信号、フレーム信号等の信号の値に関係なく、入力インタフェースで受信されたデータは、同一の順序で出力インタフェースに送出されます。

シングル・クロック・モードでは、FIFO のフィル・レベルに関する情報を提供するステータス・インタフェースを追加することが可能です。デュアル・クロック・モードでは、入力インタフェースおよび出力インタフェース用に、個別のステータス・インタフェースを追加することが可能です。ステータス・インタフェースには、割り込みを設定および制御するためのレジスタも含まれています。

オンチップ FIFO メモリ・コアは SOPC Builder に対応しており、SOPC Builder で生成されたとのシステムにも容易に統合できます。デバイス・ドライバが HAL システム・ライブラリで提供されており、ソフトウェアは ANSIC を使用してコアにアクセスすることができます。

この章は、以下の項で構成されています。

- 「機能の説明」
- 15-8 ページ「デバイスおよびツールのサポート」
- 15-8 ページ「SOPC Builder でのコアのインスタンス化」
- 15-11 ページ「ソフトウェア・プログラミング・モデル」
- 15-12 ページ「オンチップ FIFO メモリを使用したプログラミング」
- 15-19 ページ「オンチップ FIFO メモリ・コア API」

機能の説明

オンチップ FIFO メモリには、以下の 4 つのコンフィギュレーションがあります。

- Avalon-MM ライト・スレーブ - Avalon-MM リード・スレーブ
- Avalon-ST シンク - Avalon-ST ソース
- Avalon-MM ライト・スレーブ - Avalon-ST ソース
- Avalon-ST シンク - Avalon-MM リード・スレーブ

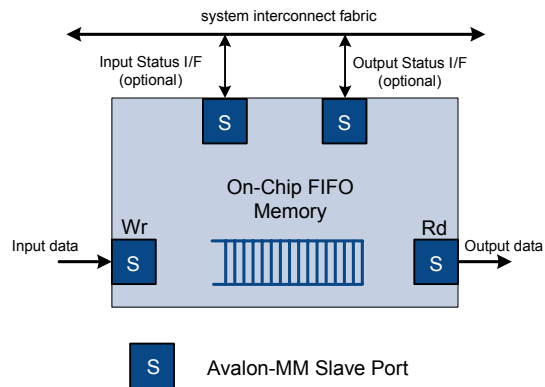
すべてのコンフィギュレーションで、入力インタフェースおよび出力インタフェースは、オプションのバックプレッシャ信号を使用して、アンダーフローおよびオーバーフロー状態を防止できます。Avalon-MM インタフェースの場合、バックプレッシャは `waitrequest` 信号を使用して実装されます。Avalon-ST インタフェースの場合、バックプレッシャは `ready` 信号および `valid` 信号を使用して実装されます。オンチップ FIFO メモリでは、シンクが `ready` をアサートしてからソースが有効データをドライブするまでの遅延 (`READY_LATENCY`) は 1 サイクルです。

Avalon-MM ライト・スレーブ - Avalon-MM リード・スレーブ

このモードでは、FIFO の入力はアドレス幅がゼロの Avalon-MM ライト・スレーブです。Avalon-MM ライト・マスタが、この入力インタフェースに書き込むことでデータが FIFO に書き込まれ、リード・マスタ (おそらく同じマスタ) はその出力インタフェースから読み出してデータを FIFO から取り出します。FIFO の入力データと出力データは同じ幅でなければなりません。

Allow backpressure がオンの場合、`data_in` マスタがフル状態の FIFO への書き込みを試みると、常に `waitrequest` 信号がアサートされます。FIFO 内に新しいトランザクションが完了するのに十分なスペースがある場合に限り、`waitrequest` がデアサートされます。`waitrequest` は、FIFO から読み出すデータがないときに読み出し動作に対してアサートされ、FIFO にデータがあるときはデアサートされます。

図 15-1. Avalon-MM 入力および出力インタフェース付き FIFO



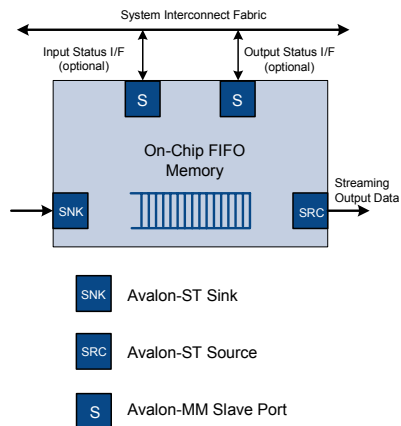
Avalon-ST シンク - Avalon-ST ソース

この FIFO は、図 15-2 に示すように、ストリーミング用の入力インタフェースと出力インタフェースを備えています。シンボルあたりのビット数、ビートあたりのシンボル数、および error 信号と channel 信号の幅を含む Avalon-ST インタフェースの大部分をパラメータ化することができます。入力インタフェースと出力インタフェースは同じ幅でなければなりません。SOPC Builder MegaWizard で、**Allow backpressure** がオンになっている場合、両方のインタフェースとも ready 信号と valid 信号を使用して、FIFO 内にスペースがあることおよび有効データが存在することを示します。



Avalon-ST インタフェース・プロトコルについて詳しくは、「[Avalon Interface Specifications](#)」を参照してください。

図 15-2. Avalon-ST 入力および出力インタフェース付き FIFO



Avalon-MM ライト・スレーブ - Avalon-ST ソース

このモードでは、FIFOの入力は、図 15-3 に示すように、32 ビット幅の Avalon-MM ライト・スレーブです。Avalon-ST 出力（ソース）データ幅も 32 ビットでなければなりません。シンボルあたりのビット数、ビットあたりのシンボル数、および channel 信号と error 信号の幅を含む出力インタフェース・パラメータをコンフィギュレーションすることができます。FIFO は、出力インタフェース・プロトコルに適合するようにエンディアン変換を実行します。

このインタフェースを構成する信号は、Avalon のアドレス空間内のビットにマップされます。**Allow backpressure** がオンの場合、入力インタフェースは waitrequest をアサートして、FIFO にトランザクションを完了するのに十分なスペースがないことを示します。

図 15-3. Avalon-MM 入力インタフェースおよび Avalon-ST 出力インタフェース付き FIFO

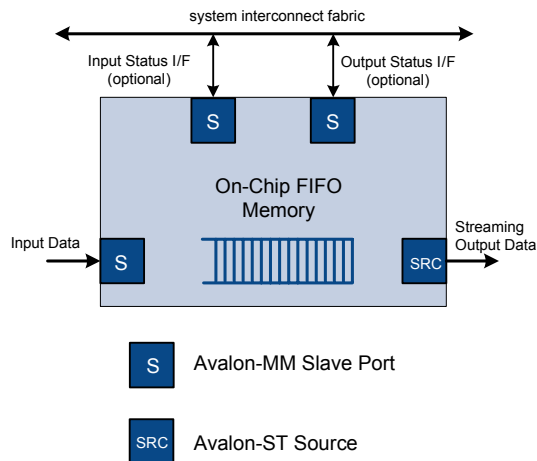


表 15-1 のメモリ・マップは、32 ビット Avalon-MM 入力インタフェースと Avalon-ST 出力インタフェース付き FIFO 用のメモリ・レイアウトを示します。出力インタフェースは、8 ビット・シンボル、5 ビット・チャンネル信号、3 ビット・エラー信号、およびパケット・サポート機能を備えたオンチップ FIFO メモリ・コアの例です。

オフセット	31	24	23	19	18	16	15	13	12	8	7	4	3	2	1	0
base + 0	シンボル 3			シンボル 2			シンボル 1			シンボル 0						
base + 1	reserved			reserved	error	resrvd.	channel		reserved	empty	SOP		EOP		channel	

Enable packet data がオフの場合、Avalon-MM ライト・マスタはアドレス・オフセット 0 にすべてのデータを繰り返し書き込んで、データを FIFO にプッシュします。

Enable packet data がオンの場合、Avalon-MM ライト・マスタは最初にアドレス・オフセット 1 に SOP、error (オプション)、channel (オプション)、EOP、および empty のパケット・ステータス情報を書き込みます。アドレス・オフセット 1 への書き込みでは、FIFO への書き込みは行われません。Avalon-MM マスタは次に、アドレス・オフセット 0 にパケット・データを繰り返し書き込み、8 ビット・シンボルを FIFO に

プッシュします。アドレス・オフセット 0 に有効な書き込みが行われるたびに、データとそれぞれのパケット情報が FIFO にプッシュされます。後続データ書き込み時には、SOP をクリアする必要なく、アドレス・オフセット 0 に書き込むことができます。error と channel に変更が無く、FIFO にプッシュされる後続データがパケットの終了データでない場合には、アドレス・オフセット 1 に再書き込みする必要はありません。

各パケットの終了時に、Avalon-MM マスタはアドレス・オフセット 1 に書き込んで EOP ビットを 1 に設定してから、パケットの最後のシンボルをオフセット 0 に書き込みます。ライト・マスタは、empty フィールドを使用して、最後の転送での未使用シンボル数を示す必要があります。最後のパケット・データがビートあたりのシンボル数と整合しない場合、empty フィールドで最後のパケット・データの空のシンボル数を示します。例えば、Avalon-ST インタフェースのビートあたりのシンボル数が 4 で、最後のパケットが 3 つのシンボルしか持たない場合、empty フィールドを 1 とし、1 つのシンボル（メモリ・マップ内の最下位のシンボル）が空であることを示します。

Avalon-ST シンク - Avalon-MM リード・スレーブ

このモードでは、FIFO の入力 は Avalon-ST シンクで、出力は 32 ビット幅の Avalon-MM リード・スレーブです (図 15-4)。Avalon-ST 入力 (シンク) データ幅も 32 ビットでなければなりません。シンボルあたりのビット数、ビートあたりのシンボル数、および channel 信号と error 信号の幅を含む入力インタフェース・パラメータをコンフィギュレーションすることができます。FIFO は、出力インタフェース・プロトコルに適合するようにエンディアン変換を実行します。

Avalon-MM マスタは、FIFO からデータを読み出します。信号は Avalon のアドレス空間内のビットにマップされます。SOPC Builder MegaWizard で **Allow backpressure** がオンになっている場合、入力 (シンク) インタフェースは ready 信号と valid 信号を使用して、FIFO 内にスペースがあることおよび有効データが存在することを示します。出力インタフェースの場合、waitrequest は、FIFO から読み出すデータがないときに、読み出し動作に対してアサートされます。FIFO に送信するデータがある場合はデアサートされます。

図 15-4. Avalon-ST 入力および Avalon-MM 出力付き FIFO

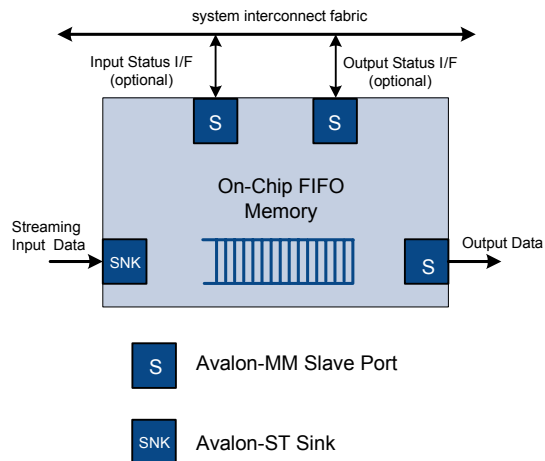


表 15-2 に示すように、Avalon-ST - Avalon-MM スレーブ FIFO 用のメモリ・マップは、Avalon-MM - Avalon-ST FIFO 用のものとまったく同じです。

オフセット	31	24	23	19	18	16	15	13	12	8	7	4	3	2	1	0
base + 0	シンボル 3			シンボル 2			シンボル 1			シンボル 0						
base + 1	reserved			reserved	error	resrvd.	channel		reserved	empty	A L O S		E O P		S O P	

Enable packet data がオフの場合、アドレス・オフセット 0 でデータを繰り返し読み出して、FIFO からデータをポップします。

Enable packet data がオンの場合、Avalon-MM リード・マスタはアドレス・オフセット 0 から読み出しを開始します。読み出しが有効な場合、すなわち FIFO が空でない場合は、データとパケット・ステータス情報の両方が FIFO からポップされます。パケット・ステータス情報は、アドレス・オフセット 1 で読み出すことにより取得されます。アドレス・オフセット 1 から読み出しても、データは FIFO からポップされません。アドレス・オフセット 1 の **error**、**channel**、**SOP**、**EOP**、および **empty** の各フィールドは、アドレス・オフセット 0 から読み出されたパケット・データの状態を確認するために使用してください。

empty フィールドは、データ・フィールド内の空のシンボル数を示します。例えば、Avalon-ST インタフェースのビートあたりのシンボル数が 4 で、最後のパケット・データに 1 つのシンボルしかない場合、empty フィールドは 3 になり、3 つのシンボル（メモリ・マップ内の最下位から 3 つのシンボル）が空であることを示します。

ステータス・インタフェース

FIFO は 2 つのステータス・インタフェースをオプションで提供します。1 つは入力インタフェースに書き込むマスタ用のインタフェース、もう 1 つは出力インタフェースから読み出すリード・マスタ用のインタフェースです。1 つのドメインで動作する FIFO の場合、FIFO の状態をモニタするには 1 つのステータス・インタフェースで十分です。デュアル・クロッキング方式を使用する FIFO の場合は、両方のクロック・ドメインで FIFO の状態を正確にモニタするために、出力クロックを使用するもう 1 つのステータス・インタフェースが必要です。

クロック・モード

シングル・クロック・モードが使用されているとき、使用している FIFO は SCFIFO です。デュアル・クロック・モードが選択されているとき、使用している FIFO は DCFIFO です。デュアル・クロック・モードでは、入力データ・インタフェースおよび書き込み側ステータス・インタフェースは書き込み側クロック・ドメインを使用し、出力データ・インタフェースおよび読み出し側ステータス・インタフェースは読み出し側クロック・ドメインを使用します。

デバイスおよび ツールの サポート

オンチップ FIFO メモリは、Arria™ GX、Stratix® III、Stratix II GX、Stratix II、Stratix GX、Stratix、Cyclone® III、Cyclone II、Cyclone および Hardcopy® II デバイス・ファミリーをサポートしています。

SOPC Builder でのコアの インスタンス化

設計者は、SOPC Builder のオンチップ FIFO メモリ用の MegaWizard® Plug-In Manager を使用して、コアのコンフィギュレーションを指定します。以下の項では、MegaWizard インタフェースで使用可能なオプションについて説明します。

FIFO の設定

以下の項では、FIFO 全般に関する設定について概説します。

Depth

Depth は、Avalon-ST ビートまたは Avalon-MM ワードで表される FIFO の深度を示します。デフォルトの Depth は 16 です。デュアル・クロック・モードを使用するとき、実際の FIFO の深度は (depth-3) になります。これはクロック・クロッシングに起因し、FIFO オーバーフローを回避するためのものです。

クロック設定

シングル・クロック・モードとデュアル・クロック・モードの2つのオプションがあります。シングル・クロック・モードでは、すべてのインタフェース・ポートで同じクロックを使用します。デュアル・クロック・モードでは、入力データ・インタフェースと入力側ステータス・インタフェースは入力クロック・ドメインにあります。出力データ・インタフェースと出力側ステータス・インタフェースは出力クロック・ドメインにあります。

ステータス・ポート

オプションのステータス・ポートは Avalon-MM です。オプションの入力側ステータス・インタフェースを含めるには、SOPC Builder MegaWizard の **Create status interface for input** をオンにします。入力ポートと出力ポートが別々のクロック・ドメインで動作する FIFO の場合、**Create status interface for output** をオンにして、もう1つのステータス・インタフェースを含めることができます。**Enable IRQ for status ports** をオンにすると、ステータス・ポートに割り込み信号が追加されます。

FIFO の実装

このオプションにより、FIFO をレジスタまたはエンベデッド・メモリ・ブロックのどちらを使用して実装するかを選択できます。デフォルトでは、FIFO はエンベデッド・メモリ・ブロックから構築されます。

インタフェース・パラメータ

以下の項では、入力インタフェースおよび出力インタフェース用のオプションについて概説します。

入力

使用可能な入力インタフェースは、Avalon-MM ライト・スレーブおよび Avalon-ST シンクです。

出力

使用可能な出力インタフェースは、**Avalon-MM** リード・スレーブおよび **Avalon-ST** ソースです。

バックプレッシャの許可

Allow backpressure がオンのとき、Avalon-MM インタフェースには waitrequest 信号が含まれます。この信号は、マスタがフル状態の FIFO に書き込んだり、空の FIFO から読み出すのを防止するためにアサートされます。Avalon-ST インタフェースには、アンダーフロー状態やオーバーフロー状態を防止するために、ready 信号と valid 信号が含まれています。

Avalon-MM ポートの設定

有効なデータ幅は 8 ビット、16 ビット、および 32 ビットです。

一方のインタフェース用に Avalon-MM が選択され、もう一方のインタフェース用に Avalon-ST が選択された場合、データ幅は 32 ビットに固定されます。

Avalon-MM インタフェースは、一度に 4 バイトのデータにアクセスします。データ幅が 32 ビット以外の場合は、潜在的なオーバーフロー状態とアンダーフロー状態に注意する必要があります。

Avalon-ST ポートの設定

以下のパラメータにより、1 つまたは複数の Avalon-ST ポートのサイズとエラー処理を指定することができます。

- シンボルあたりのビット数
- ビットあたりのシンボル数
- チャンネル幅
- エラー幅

シンボル・サイズが 2 のべき乗でない場合、最も近い 2 のべき乗に切り上げられます。例えば、シンボルあたりのビット数が 10 の場合、シンボルは 16 ビットのメモリ位置にマップされます。10 ビットのシンボルの場合、ビットあたりの最大シンボル数は 2 です。

Enable packet data は、パケット送信用のオプションを提供します。

ソフトウェア・ プログラミング・ モデル

以下の項では、ハードウェアにアクセスするためのレジスタ・マップやソフトウェア宣言など、オンチップ FIFO メモリ・コアのソフトウェア・プログラミング・モデルについて説明します。Nios II プロセッサ・ユーザー用に、アルテラは HAL API を使用したオンチップ FIFO メモリへのアクセスを可能にする HAL システム・ライブラリ・ドライバを提供しています。

HAL システム・ライブラリ・サポート

アルテラが提供するドライバは、Nios II システム用の HAL システム・ライブラリに統合される HAL デバイス・ドライバを実装します。HAL ユーザーは、レジスタに直接アクセスしないで、使い慣れた HAL API を介してオンチップ FIFO メモリにアクセスする必要があります。

ソフトウェア・ファイル

アルテラは、オンチップ FIFO メモリ・コア向けの以下のソフトウェア・ファイルを提供しています。

- **altera_avalon_fifo_regs.h**— コアのレジスタ・マップを定義し、ローレベル・ハードウェアにアクセスするためのシンボル定数を提供します。
- **altera_avalon_fifo_util.h**— オンチップ FIFO メモリ・コア・ハードウェアにアクセスするための関数を定義しています。FIFO の初期化、ステータスの読み出しと書き込み、フラグのイネーブルとイベントの読み出しのためのユーティリティを提供します。
- **altera_avalon_fifo.h**— オンチップ FIFO メモリへのパブリック・インタフェースを提供します。
- **altera_avalon_fifo_util.c**— **altera_avalon_fifo_util.h** にリストされたユーティリティを実装します。

オンチップ FIFO メモリを 使用したプロ グラミング

この項では、オンチップ FIFO メモリ・コア・ハードウェアを操作するローレベル・ソフトウェア構成について説明します。表 15-3 に、使用可能な関数のすべてを示します。

関数名	説明
<code>altera_avalon_fifo_init()</code>	FIFO を初期化します。
<code>altera_avalon_fifo_read_status()</code>	ステータス・レジスタの指定されたビットの値を整数値で返します。すべてのビットを一括して読み出すには、 <code>ALTERA_AVALON_FIFO_STATUS_ALL</code> マスクを使用します。
<code>altera_avalon_fifo_read_ienable()</code>	割り込みイネーブル・レジスタの指定されたビットの値を返します。すべてのビットを一括して読み出すには、 <code>ALTERA_AVALON_FIFO_EVENT_ALL</code> マスクを使用します。
<code>altera_avalon_fifo_read_almostfull()</code>	<code>almostfull</code> レジスタの値を返します。
<code>altera_avalon_fifo_read_almostempty()</code>	<code>almostempty</code> レジスタの値を返します。
<code>altera_avalon_fifo_read_event()</code>	イベント・レジスタの指定されたビットの値を返します。 <code>ALTERA_AVALON_FIFO_STATUS_ALL</code> マスクを使用して、すべてイベント・ビットを一括して読み出すことができます。
<code>altera_avalon_fifo_read_level()</code>	FIFO のフィル・レベルを返します。
<code>altera_avalon_fifo_clear_event()</code>	指定されたビットとイベント・レジスタをクリアし、エラー・チェックを実行します。
<code>altera_avalon_fifo_write_ienable()</code>	<code>interruptenable</code> レジスタの指定されたビットに書き込み、エラー・チェックを実行します。
<code>altera_avalon_fifo_write_almostfull()</code>	<code>almostfull</code> レジスタに指定された値を書き込み、エラー・チェックを実行します。
<code>altera_avalon_fifo_write_almostempty()</code>	<code>almostempty</code> レジスタに指定された値を書き込み、エラー・チェックを実行します。
<code>altera_avalon_fifo_write_fifo()</code>	<code>write_address</code> に指定されたデータを書き込みます。
<code>altera_avalon_fifo_write_other_info()</code>	<code>write_address</code> にパケット・ステータス情報を書き込みます。 Enable packet data がオンのときにのみ使用可能です。

関数名	説明
altera_avalon_fifo_read_fifo()	指定された read_address からデータを読み出します。
altera_avalon_fifo_read__other_info()	指定された read_address からパケット・ステータス情報を読み出します。 Enable packet data がオンのときにのみ使用可能です。

ソフトウェア制御

表 15-4 に、status レジスタのレジスタ・マップを示します。入力インタフェースと出力インタフェースの status レジスタのレイアウトは同一です。

オフセット	31	24	23	16	15	8	7	6	5	4	3	2	1	0
base	fill_level													
base + 1												i_status		
base + 2												event		
base + 3												interruptenable		
base + 4	almostfull													
base + 5	almostempty													

表 15-5 では、status レジスタの各フィールドの使用法について概説しています。

フィールド	タイプ	説明
fill_level	RO	その時点での FIFO のフィル・レベル。Avalon-ST FIFO の場合はシンボル、Avalon-MM FIFO の場合はワード単位で示されます。
i_status	RO	その時点での FIFO の状態を示す 6 ビット・レジスタ。各ビット・フィールドの意味については、表 15-6 を参照してください。
event	RW1C	i_status と同一フィールドを持つ 6 ビット・レジスタ。i_status レジスタのビットが設定されると、event レジスタの同じビットが設定されます。event レジスタのビットは、ソフトウェアがそのビットに 1 を書き込んだときのみクリアされます。

フィールド	タイプ	説明
interruptenable	RW	eventレジスタおよびi_statusレジスタと同一フィールドを持つ6ビット割り込みイネーブル・レジスタ。eventレジスタのビットが0から1に変わり、かつinterruptenableの対応するビットがセットされている場合、マスタへの割り込みが発生します。
almostfull	RW	割り込みおよびステータスに使用されるスレッシュヨルド・レベル。Avalon-MMステータス・マスタにより、いつでも書き込みが可能です。DCFIFOのデフォルトのスレッシュヨルド値は (Depth-4) です。SCFIFOのデフォルトのスレッシュヨルド値は (Depth-1) です。スレッシュヨルド値の有効範囲は1からデフォルト値までです。1よりも小さい値の書き込みを試みる場合は1が使用されます。デフォルト値よりも大きい値の書き込んだ場合には、デフォルト値が使用されます。
almostempty	RW	割り込みおよびステータスに使用されるスレッシュヨルド・レベル。Avalon-MMステータス・マスタにより、いつでも書き込みが可能です。DCFIFOのデフォルトのスレッシュヨルド値は1です。SCFIFOのデフォルトのスレッシュヨルド値は1です。スレッシュヨルド値の有効範囲は、1からalmostfullスレッシュヨルドに設定可能な最大値までです。1よりも小さい値の書き込みを試みる場合は1が使用されます。許容最大almostfullスレッシュヨルドよりも大きい値の書き込みを試みる場合は、許容最大almostfullスレッシュヨルドが使用されます。

表 15-6 では、瞬間的なステータスのビットについて説明しています。

ビット	名前	説明
0	FULL	FIFO が現在フル状態の場合は 1 になります。
1	EMPTY	FIFO が現在空の場合は 1 になります。
2	ALMOSTFULL	FIFO のフィル・レベルが almostfull 値よりも大きい場合は 1 になります。
3	ALMOSTEMPTY	FIFO のフィル・レベルが almostempty 値よりも小さい場合は 1 になります。
4	OVERFLOW	FIFO がオーバーフローするたびに、1 サイクルの間 1 に設定されます。Avalon ライト・マスタがフル状態の FIFO に書き込むと、FIFO がオーバーフローします。OVERFLOW は、 Allow backpressure がオフの場合にのみ有効です。
5	UNDERFLOW	FIFO がアンダーフローするたびに、1 サイクルの間 1 が設定されます。Avalon リード・マスタが空の FIFO から読み出すと、FIFO がアンダーフローします。UNDERFLOW は、 Allow backpressure がオフのときにのみ有効です。

表 15-7 に、event レジスタのビット・フィールドを示します。これらのフィールドは、status レジスタのフィールドと同じであり、同時にセットされます。ただし、これらのフィールドはソフトウェアから 1 を書き込んだ (W1C) ときのみクリアされます。event フィールドを使用して、特定のイベントが発生したかどうかを確認することができます。

ビット	名前	説明
1	E_FULL	FIFO がフル状態になっており、かつソフトウェアでビットがクリアされていない場合は 1 になります。
0	E_EMPTY	FIFO が空になっており、かつソフトウェアでビットがクリアされていない場合は 1 になります。
3	E_ALMOSTFULL	FIFO のフィル・レベルが almostfull スレッショルド値よりも大きくなっており、かつソフトウェアでビットがクリアされていない場合は 1 になります。
2	E_ALMOSTEMPTY	FIFO のフィル・レベルが almostempty 値よりも小さくなっており、かつソフトウェアでビットがクリアされていない場合は 1 になります。
4	E_OVERFLOW	FIFO がオーバーフローし、かつソフトウェアでビットがクリアされていない場合は 1 になります。
5	E_UNDERFLOW	FIFO がアンダーフローしており、かつソフトウェアでビットがクリアされていない場合は 1 になります。

表 15-8 に、6 つの STATUS フィールド用のマスクを示します。event レジスタのビットが 0 から 1 に変わり、かつ interruptenable レジスタの対応するビットが設定された場合、マスタへの割り込みが発生します。

ビット	名前	説明
1	IE_FULL	FIFO が現在フル状態の場合は、割り込みをイネーブルします。
0	IE_EMPTY	FIFO が現在空の場合は、割り込みをイネーブルします。
3	IE_ALMOSTFULL	FIFO のフィル・レベルが almostfull レジスタの値よりも大きい場合、割り込みをイネーブルします。
2	IE_ALMOSTEMPTY	FIFO のフィル・レベルが almostempty レジスタの値よりも小さい場合、割り込みをイネーブルします。

表 15-8. InterruptEnable ビット・フィールドの説明 (2 / 2)

ビット	名前	説明
4	IE_OVERFLOW	FIFO がオーバーフローすると、割り込みをイネーブルします。Avalon ライト・マスタがフル状態の FIFO に書き込むと、FIFO がオーバーフローします。
5	IE_UNDERFLOW	FIFO がアンダーフローすると、割り込みをイネーブルします。Avalon リード・マスタが空の FIFO から読み出すと、FIFO がアンダーフローします。
6	ALL	割り込みのための6つのステータス条件をすべてイネーブルします。

すべてのレジスタにアクセスするためのマクロは `altera_avalon_fifo_regs.h` に定義されています。例えば、このファイルには `status` レジスタにアクセスするための以下のマクロが含まれています。

```
#define ALTERA_AVALON_FIFO_LEVEL_REG      0
#define ALTERA_AVALON_FIFO_STATUS_REG    1
#define ALTERA_AVALON_FIFO_EVENT_REG     2
#define ALTERA_AVALON_FIFO_IENABLE_REG   3
#define ALTERA_AVALON_FIFO_ALMOSTFULL_REG 4
#define ALTERA_AVALON_FIFO_ALMOSTEMPTY_REG 5
```



オンチップ FIFO ハードウェアにアクセスするための定義済みマクロおよびユーティリティの完全なリストは、`<install_dir>\quartus\sopc_builder\components\altera_avalon_fifo\HAL\inc\alatera_avalon_fifo.h` および `<install_dir>\quartus\sopc_builder\components\altera_avalon_fifo\HAL\inc\alatera_avalon_fifo_util.h` を参照してください。

ソフトウェア例：

オンチップ FIFO メモリ向けの詳細なプログラミング例は、Quartus II 資料ページの本資料の隣に記載されています。www.altera.com/literature/quartus2/lit-qts-peripherals.jsp をご覧ください。

例 15-1. オンチップ FIFO メモリ向けサンプル・コード

```

/*****
//Includes
#include "altera_avalon_fifo_regs.h"
#include "altera_avalon_fifo_util.h"
#include "system.h"
#include "sys/alt_irq.h"
#include <stdio.h>
#include <stdlib.h>

#define ALMOST_EMPTY 2
#define ALMOST_FULL OUTPUT_FIFO_OUT_FIFO_DEPTH-5

volatile int input_fifo_wrclk_irq_event;

void print_status(alt_u32 control_base_address)
{
    printf("-----\n");
    printf("LEVEL = %u\n",
altera_avalon_fifo_read_level(control_base_address) );
    printf("STATUS = %u\n",
altera_avalon_fifo_read_status(control_base_address,
ALTERA_AVALON_FIFO_STATUS_ALL) );
    printf("EVENT = %u\n",
altera_avalon_fifo_read_event(control_base_address,
ALTERA_AVALON_FIFO_EVENT_ALL) );
    printf("IENABLE = %u\n",
altera_avalon_fifo_read_ienable(control_base_address,
ALTERA_AVALON_FIFO_IENABLE_ALL) );
    printf("ALMOSTEMPTY = %u\n",
altera_avalon_fifo_read_almostempty(control_base_address) );
    printf("ALMOSTFULL = %u\n\n",
altera_avalon_fifo_read_almostfull(control_base_address));
}

static void handle_input_fifo_wrclk_interrupts(void* context, alt_u32 id)
{
    /* Cast context to input_fifo_wrclk_irq_event's type.It is important
    * to declare this volatile to avoid unwanted compiler optimization.
    */
    volatile int* input_fifo_wrclk_irq_event_ptr = (volatile int*) context;

    /* Store the value in the FIFO's irq history register in *context. */

```

```
    *input_fifo_wrclk_irq_event_ptr =
altera_avalon_fifo_read_event(INPUT_FIFO_IN_CSR_BASE,
ALTERA_AVALON_FIFO_EVENT_ALL);
    printf("Interrupt Occurs for %#x\n", INPUT_FIFO_IN_CSR_BASE);
    print_status(INPUT_FIFO_IN_CSR_BASE);

    /* Reset the FIFO's IRQ History register. */
    altera_avalon_fifo_clear_event(INPUT_FIFO_IN_CSR_BASE,
ALTERA_AVALON_FIFO_EVENT_ALL);
}

/* Initialize the fifo */
static int init_input_fifo_wrclk_control()
{
    int return_code = ALTERA_AVALON_FIFO_OK;

    /* Recast the IRQ History pointer to match the alt_irq_register()
function
    * prototype. */
    void* input_fifo_wrclk_irq_event_ptr = (void*)
&input_fifo_wrclk_irq_event;
    /* Enable all interrupts. */

    /* Clear event register, set enable all irq, set almostempty and
almostfull threshold */
    return_code = altera_avalon_fifo_init(INPUT_FIFO_IN_CSR_BASE,
                                          0, // Disabled interrupts
                                          ALMOST_EMPTY,
                                          ALMOST_FULL);

    /* Register the interrupt handler. */
    alt_irq_register( INPUT_FIFO_IN_CSR_IRQ,
input_fifo_wrclk_irq_event_ptr, handle_input_fifo_wrclk_interrupts );
    return return_code;
}
```

オンチップ FIFO メモリ・ コア API

この項では、オンチップ FIFO メモリ・コア用のアプリケーション・プログラミング・インタフェース (API) について説明します。

altera_avalon_fifo_init()

プロトタイプ宣言: `int altera_avalon_fifo_init(alt_u32 address, alt_u32 ienable, alt_u32 emptymark, alt_u32 fullmark)`

スレッド・セーフ: ×

ISR からの利用: ×

インクルード: `<altera_avalon_fifo_regs.h>`, `<altera_avalon_fifo_utils.h>`

引数: `address`— FIFO コントロール・スレーブのベース・アドレス。
`ienable`— interruptenable レジスタに書き込む値。
`emptymark`— almost empty スレッシュولد・レベルの値。
`fullmark`— almost full スレッシュولد・レベルの値。

戻り値: 成功の場合は 0 (`ALTERA_AVALON_FIFO_OK`)、クリア・エラーの場合は `ALTERA_AVALON_FIFO_EVENT_CLEAR_ERROR`、割り込みイネーブル書き込みエラーの場合は `ALTERA_AVALON_FIFO_IENABLE_WRITE_ERROR`、almostfull レジスタおよびalmostemptyレジスタへの書き込みのエラーの場合は `ALTERA_AVALON_FIFO_THRESHOLD_WRITE_ERROR` を返します。

説明: `event` レジスタをクリアし、`interruptenable` レジスタに書き込み、`almostfull` レジスタと `almostempty` レジスタを設定します。

altera_avalon_fifo_read_status()

プロトタイプ宣言: `int altera_avalon_fifo_read_status(alt_u32 address, alt_u32 mask)`

スレッド・セーフ: ×

ISR からの利用: ×

インクルード: `<altera_avalon_fifo_regs.h>`, `<altera_avalon_fifo_utils.h>`

引数: `address`— FIFO コントロール・スレーブのベース・アドレス。
`mask`— ステータス・レジスタからの読み出し値をマスクします。

戻り値: ステータス・レジスタの指定されたビット値を返します。

説明: アドレス指定された FIFO のステータス・レジスタの値と `mask` の論理積を返します。

altera_avalon_fifo_read_ienable()

プロトタイプ宣言: `int altera_avalon_fifo_read_ienable(alt_u32 address, alt_u32 mask)`

スレッド・セーフ: ×

ISR からの利用: ×

インクルード: `<altera_avalon_fifo_regs.h>`, `<altera_avalon_fifo_utils.h>`

引数: `address`— FIFO コントロール・スレーブのベース・アドレス。
`mask`— interruptenable レジスタからの読み出し値をマスクします。

戻り値: interruptenable レジスタと `mask` の論理積を返します。

説明: interruptenable レジスタと `mask` の論理積を取得します。

altera_avalon_fifo_read_almostfull()

プロトタイプ宣言: `int altera_avalon_fifo_read_almostfull(alt_u32 address)`
スレッド・セーフ: ×
ISR からの利用: ×
インクルード: `<altera_avalon_fifo_regs.h>`, `<altera_avalon_fifo_utils.h>`
引数: `address`— FIFO コントロール・スレーブのベース・アドレス。
戻り値: `almostfull` レジスタの値を返します。
説明: `almostfull` レジスタの値を取得します。

altera_avalon_fifo_read_almostempty()

プロトタイプ宣言: `int altera_avalon_fifo_read_almostempty(alt_u32 address)`
スレッド・セーフ: ×
ISR からの利用: ×
インクルード: `<altera_avalon_fifo_regs.h>`, `<altera_avalon_fifo_utils.h>`
引数: `address`— FIFO コントロール・スレーブのベース・アドレス。
戻り値: `almostempty` レジスタの値を返します。
説明: `almostempty` レジスタの値を取得します。

altera_avalon_fifo_read_event()

プロトタイプ宣言: `int altera_avalon_fifo_read_event(alt_u32 address, alt_u32 mask)`

スレッド・セーフ: ×

ISR からの利用: ×

インクルード: `<altera_avalon_fifo_regs.h>`, `<altera_avalon_fifo_utils.h>`

引数: `address`— FIFO コントロール・スレーブのベース・アドレス。
`mask`— イベント・レジスタからの読み出し値をマスクします。

戻り値: `event` レジスタと `mask` の論理積を返します。

説明: `event` レジスタと `mask` の論理積を取得します。イベント・レジスタの1ビットを読み出すには、`ALTERA_AVALON_FIFO_FIFO_EVENT_F_MSK` などのシングル・ビット・マスクを使用します。イベント・レジスタ全体を読み出すには、フル・マスク: `ALTERA_AVALON_FIFO_EVENT_ALL` を使用します。

altera_avalon_fifo_read_level()

プロトタイプ宣言: `int altera_avalon_fifo_read_level(alt_u32 address)`
スレッド・セーフ: ×
ISR からの利用: ×
インクルード: `<altera_avalon_fifo_regs.h>`, `<altera_avalon_fifo_utils.h>`
引数: `address`— FIFO コントロール・スレーブのベース・アドレス。
戻り値: FIFO のフィル・レベルを返します。
説明: FIFO のフィル・レベルを取得します。

altera_avalon_fifo_clear_event()

プロトタイプ宣言: `int altera_avalon_fifo_clear_event(alt_u32 address, alt_u32 mask)`

スレッド・セーフ: ×

ISR からの利用: ×

インクルード: `<altera_avalon_fifo_regs.h>`, `<altera_avalon_fifo_utils.h>`

引数: `address`— FIFO コントロール・スレーブのベース・アドレス。
`mask`— ビットのクリアに使用するマスク(1はこのビットをクリアすること、0はクリアしないことを意味する)。

戻り値: 成功の場合は 0 (`ALTERA_AVALON_FIFO_OK`)、失敗の場合は `ALTERA_AVALON_FIFO_EVENT_CLEAR_ERROR` を返します。

説明: `event` レジスタの指定されたビットをクリアします。

altera_avalon_fifo_write_ienable()

プロトタイプ宣言: `int altera_avalon_fifo_write_ienable(alt_u32 address, alt_u32 mask)`

スレッド・セーフ: ×

ISR からの利用: ×

インクルード: `<altera_avalon_fifo_regs.h>`, `<altera_avalon_fifo_utils.h>`

引数: `address`— FIFO コントロール・スレーブのベース・アドレス。
`mask`— interruptenable レジスタに書き込む値。個々の割り込みビット・マスクについては、`altera_avalon_fifo_regs.h` を参照してください。

戻り値: 成功の場合は0(`ALTERA_AVALON_FIFO_OK`)、失敗の場合は`ALTERA_AVALON_FIFO_IENABLE_WRITE_ERROR` を返します。

説明: interruptenable レジスタの指定されたビットに書き込みます。

altera_avalon_fifo_write_almostfull()

プロトタイプ宣言: `int altera_avalon_fifo_write_almostfull(alt_u32 address, alt_u32 data)`

スレッド・セーフ: ×

ISR からの利用: ×

インクルード: `<altera_avalon_fifo_regs.h>`, `<altera_avalon_fifo_utils.h>`

引数: `address`— FIFO コントロール・スレーブのベース・アドレス。
`data`— `almost full` スレッシュホールド・レベルの値。

戻り値: 成功の場合は `0` (`ALTERA_AVALON_FIFO_OK`)、失敗の場合は `ALTERA_AVALON_FIFO_THRESHOLD_WRITE_ERROR` を返します。

説明: `almostfull` レジスタに `data` を書き込みます。

altera_avalon_fifo_write_almostempty()

プロトタイプ宣言: `int altera_avalon_fifo_write_almostempty(alt_u32 address, alt_u23 data)`

スレッド・セーフ: ×

ISR からの利用: ×

インクルード: `<altera_avalon_fifo_regs.h>`, `<altera_avalon_fifo_utils.h>`

引数: `address`— FIFO コントロール・スレーブのベース・アドレス。
`data`— almost empty スレッシュヨルド・レベルの値。

戻り値: 成功の場合は `ALTERA_AVALON_FIFO_OK`、失敗の場合は `ALTERA_AVALON_FIFO_THRESHOLD_WRITE_ERROR` を返します。

説明: `almostempty` レジスタに `data` を書き込みます。

altera_avalon_write_fifo()

プロトタイプ宣言: `int altera_avalon_write_fifo(alt_u32 write_address, alt_u32 ctrl_address, alt_u32 data)`

スレッド・セーフ: ×

ISR からの利用: ×

インクルード: `<altera_avalon_fifo_regs.h>`, `<altera_avalon_fifo_utils.h>`

引数: `write_address`— FIFO ライト・スレーブのベース・アドレス。
`ctrl_address`— FIFO コントロール・スレーブのベース・アドレス。
`data`— Avalon-MM - Avalon-ST 転送の場合はアドレス・オフセット 0 に書き込む値。
Avalon-MM 間転送の場合は使用可能な唯一のアドレスに書き込む値。データのオーダリングについては、[\[Avalon Interface Specifications\]](#) を参照してください。

戻り値: 成功の場合は 0 (`ALTERA_AVALON_FIFO_OK`)、失敗の場合は `ALTERA_AVALON_FIFO_FULL` を返します。

説明: FIFO がフル状態でない場合は、指定されたアドレスに `data` を書き込みます。

altera_avalon_write_other_info()

プロトタイプ宣言: `int altera_avalon_write_other_info(alt_u32 write_address, alt_u32 ctrl_address, alt_u32 data)`

スレッド・セーフ: ×

ISR からの利用: ×

インクルード: `<altera_avalon_fifo_regs.h>`, `<altera_avalon_fifo_utils.h>`

引数: `write_address`— FIFO ライト・スレーブのベース・アドレス。
`ctrl_address`— FIFO コントロール・スレーブのベース・アドレス。
`data`— Avalon インタフェースのアドレス・オフセット 1 に書き込むパケット・ステータス情報。パケット・ステータス情報のオーダリングについては、[「Avalon Interface Specifications」](#) を参照してください。

戻り値: 成功の場合は 0 (`ALTERA_AVALON_FIFO_OK`)、失敗の場合は `ALTERA_AVALON_FIFO_FULL` を返します。

説明: `write_address` にパケット・ステータス情報を書き込みます。Enable packet data がオンのときにのみ有効です。

altera_avalon_fifo_read_fifo()

プロトタイプ宣言: `int altera_avalon_fifo_read_fifo(alt_u32 read_address, alt_u32 ctrl_address)`

スレッド・セーフ: ×

ISR からの利用: ×

インクルード: `<altera_avalon_fifo_regs.h>`, `<altera_avalon_fifo_utils.h>`

引数: `read_address`— FIFO リード・スレーブのベース・アドレス。
`ctrl_address`— FIFO コントロール・スレーブのベース・アドレス。

戻り値: アドレス・オフセット 0 からのデータを返します。FIFO が空の場合は 0 を返します。

説明: `read_address` でアドレス指定されたデータを取得します。

altera_avalon_fifo_read_other_info()

プロトタイプ宣言: `int altera_avalon_fifo_read_other_info(alt_u32 read_address)`
スレッド・セーフ: ×
ISR からの利用: ×
インクルード: `<altera_avalon_fifo_regs.h>`, `<altera_avalon_fifo_utils.h>`
引数: `read_address`— FIFO リード・スレーブのベース・アドレス。
戻り値: Avalon インタフェースのアドレス・オフセット 1 からパケット・ステータス情報を返します。パケット・ステータス情報のオーダリングについては、「[Avalon Interface Specifications](#)」を参照してください。
説明: 指定された `read_address` からパケット・ステータス情報を読み出します。**Enable packet data** がオンのときにのみ有効です。

参考資料

この章は、「[Avalon Interface Specifications](#)」を引用しています。

改訂履歴

表 15-9 に、本資料の改訂履歴を示します。

日付および ドキュメント・ バージョン	変更内容	概要
2008年5月 v8.0.0	前バージョンからの内容の変更はありません。	—
2007年10月 v7.2.0	前バージョンからの内容の変更はありません。	—
2007年5月 v7.1.0	初版	—

