

## はじめに

SOPC Builder システム開発ツールは、FPGA の内部または外部に位置するプロセッサ、ペリフェラル、およびメモリに基づくメモリ・マップド・システムを構成する強力なプラットフォームを提供します。アルテラおよびサードパーティ・ベンダから SOPC Builder とともに使用するさまざまなコンポーネントが用意されています。これら既製の SOPC Builder 対応コンポーネントに加えて、独自のカスタム・ペリフェラルを作成し、SOPC Builder を使用するシステムに接続することもできます。カスタム・ペリフェラルを SOPC Builder にインポートすることにより、市販の SOPC Builder コンポーネントと同様に、SOPC Builder で生成されたシステムに容易に接続できます。

この資料では、ペリフェラルを開発し、それらをインポートして SOPC Builder 対応コンポーネントにする方法を示しており、以下の項目で構成されています。

- SOPC Builder および Avalon™ インタフェースの概要
- ユーザ・デザインのコンポーネントを開発し、SOPC Builder にインポートする方法が詳しく説明された実際的なデザイン例

## デザイン例について

本書および関連するデザイン・ファイルは、アルテラ・ウェブサイトの SOPC Builder の資料ページ ([www.altera.co.jp/literature/lit-sop.jsp](http://www.altera.co.jp/literature/lit-sop.jsp)) から入手できます。次の2つのデザイン例が用意されています。

- 単一の Avalon スレーブ・ペリフェラルとして実装されたパルス幅変調器 (PWM) デザイン
- ストリーミング Avalon スレーブ・ペリフェラルとして実装された VGA ビデオ・コントローラ

デザイン例を使用するには、以下について精通していることが必要です。

- アルテラの Quartus® II 開発ソフトウェアおよび SOPC Builder – 詳しくは、「Quartus II ハンドブック」を参照してください。
- Avalon インタフェース – 詳しくは、「Avalon Interface Specification Reference Manual」を参照してください。
- Mentor Graphics® 社の ModelSim® - Altera シミュレーション・ツール

デザイン例は、アルテラから提供される Nios 開発キットを使用して作成されています。Nios 開発キットがなくても、デザイン・ファイルを SOPC Builder にインポートできます。ただし、ハードウェア・システムを再生成したり、システムをハードウェアにダウンロードするには、次のキットのいずれかが必要です。

- Nios 開発キット、Cyclone™ エディション
- Nios 開発キット、Stratix™ エディション
- Nios 開発キット、Stratix プロフェッショナル・エディション

## 概要

SOPC Builder は、ハードウェア・モジュール上のアドレス・ベースのリード/ライト・インタフェースを接続する作業を自動化します。SOPC Builder はモジュールを完全なシステムに自動的に統合するため、ロジックを統合して高性能な system-on-a-programmable-chip (SOPC) デザインを作成する作業が大幅に簡素化されます。SOPC Builder を使用すれば、コンポーネントを指定するだけで、SOPC Builder によりアドレス・デコーディング、データ・パス多重化、ウェイト・ステート生成、割り込みコントローラ、およびデータ幅マッチングなどのインタコネクタ・ロジックが自動的に生成されます。また SOPC Builder は、ModelSim シミュレータでのシミュレーションのためのシミュレーション・プロジェクト・ファイル、テストベンチ、および実行スクリプトも生成します。ソフトウェア・エンジニアの場合、SOPC Builder を使用すると、システム内のペリフェラルにアクセスするプロセッサで使用するためのヘッダ・ファイルおよびドライバ・ルーチンを出力できます。

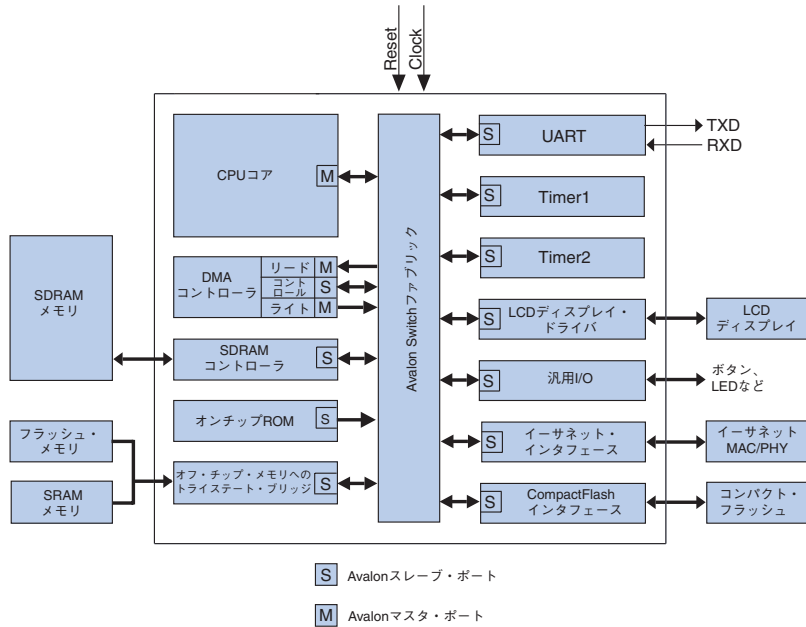
SOPC Builder のハードウェア・コンポーネントは、次の 2 つの基本的なファイルで構成されています。

- コンポーネント・ハードウェアを記述するハードウェア・デザイン・ファイル
- SOPC Builder で使用する **class.ptf** コンポーネント記述子ファイル

**class.ptf** ファイルには、コンポーネントに関する情報のデータベースが含まれており、SOPC Builder がコンポーネントをより大規模なシステムに統合する方法が定義されています。ユーザが定義したロジックのブロックを SOPC Builder コンポーネントに変換するプロセスで、**class.ptf** ファイルが作成され、そのファイルに必要な情報が書き込まれます。

SOPC Builder が出力するハードウェアの最上位階層は、「システム・モジュール」と呼ばれるハードウェア記述言語 (HDL) のデザイン・ファイルです。**システム・モジュール**には、各システム・コンポーネントを定義する HDL デザイン・ファイルと、すべてのコンポーネントを相互に接続する Avalon スイッチ・ファブリックが含まれます。SOPC Builder によって Avalon スイッチ・ファブリックが自動的に生成されるため、設計者はペリフェラル・モジュールを相互に接続するためのグルー・ロジックの作成ではなく、ペリフェラルの設計に専念できます。**3 ページの図 1**に、複数のマスタおよびスレーブ・ペリフェラルを接続するマルチ・マスタ・システム・モジュールの例を示します。

図 1. マルチ・マスタ SOPC Builder のデザイン例



## Avalon インタフェース

Avalon インタフェース規格は、SOPC (system-on-a-programmable chip) 環境のペリフェラル開発に対応するために考案されたものです。この規格はパラメータ化可能なインタフェースで、ペリフェラル・デザイナーがマスタ (マイクロプロセッサ、DMA コントローラなど) およびスレーブ (メモリ、UART、タイマなど) で使用されるアドレス・ベースのリード/ライト・ポートを記述するための基礎になります。Avalon インタフェース規格は、転送をマスタとスレーブの間で直接実行するのではなく、ペリフェラルとインタコネクト・ロジックの間での処理として定義しています。この手法によれば、システムに存在するマスタの種類が事前に分からなくてもスレーブ・ペリフェラルを設計でき、その逆も可能になります。したがって、現在のシステム向けに設計した Avalon ペリフェラルは、将来のシステムで容易に再使用できます。

Avalon インタフェースは、データ転送をサポートする役割を持つ信号タイプのセットで定義される同期インタフェースです。Avalon インタフェース・ポートには、マスタ・ポートとスレーブ・ポートの2つのタイプがあります。Avalon マスタ・ポートは転送を開始します。Avalon スレーブ・ポートは転送要求に応答します。Avalon インタフェースは、address、clk、readdata、writedata、write (ライト・イネーブル信号)、read (リード・イネーブル信号)、chipsselect、およびwaitrequest など、共通のインタフェース信号を提供して、シンプルな転送をサポートします。また Avalon は、レイテンシ付き転送 (ポスト・リード転送としても知られています) やストリーミング転送などのより高度な転送に対する信号もサポートします。

Avalon 規格では、インタフェース信号の数をアプリケーションで要求される転送タイプのサポートに必要な信号数に制限することができます。例えば、デザインによるペリフェラルがデータを返すことがない場合、Avalon インタフェースはデータの読み込みに関連するポートを必要としません。この Avalon インタフェースの独自機能によって、オンチップ・リソースが節約され、適応性の高い SOPC デザインにも十分対応できるようになります。

Avalon スイッチ・ファブリックはインタコネク構造を定義し、このインタコネク構造によって、Avalon マスタ・ポートは、ペリフェラルに存在する信号が事前に分からなくても、すべての Avalon スレーブ・ポートと相互にやりとりすることができます。Avalon スイッチ・ファブリックはシングル・マスタ・システムおよびマルチ・マスタ・システムをサポートします。さらに、データ幅の一致しないマスタ・ポートとスレーブ・ポート間で、シームレスなデータ転送を可能にします。

すでに説明したとおり、SOPC Builder は Avalon スイッチ・ファブリックを自動的に生成します。Avalon ペリフェラルの設計者は、自分のデザインに適合する Avalon インタフェースを設計するだけで、インタコネク・ロジックを作成する必要はありません。設計するペリフェラルは、最終的なシステム構造に関係なく、Avalon 信号タイプの任意のサブセットを採用することができます。

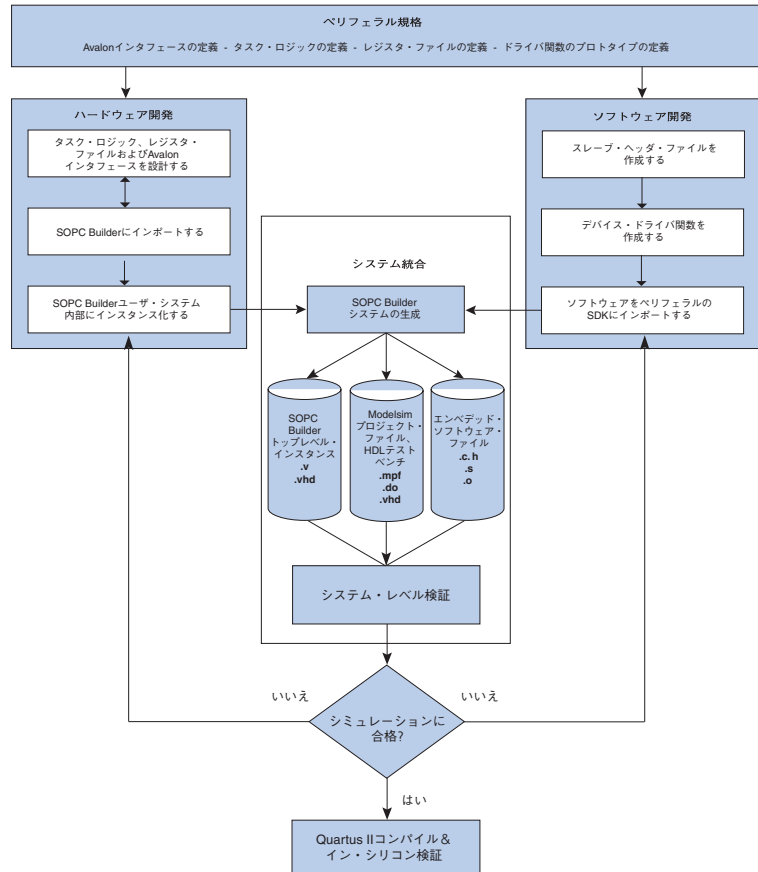


Avalon インタフェースおよび Avalon 転送について詳しくは、「Avalon Interface Specification Reference Manual」を参照してください。

## SOPC Builder のデザイン・ フロー

SOPC Builder システム統合ツールによって、ハードウェアおよびソフトウェアのデザインが容易になります。次のセクションでは、SOPC Builder コンポーネントに対する開発プロセスの概要を説明します。この説明では、コンポーネント・デザインにおけるハードウェア面とソフトウェア面の両方を扱います。本書で説明する手順に従えば、作成するどの Avalon ペリフェラルでも、ペリフェラル・デザイン・ファイルを修正することなく、SOPC Builder で作成された他のシステムに統合できます。5 ページの図 2 に、SOPC Builder で使用するためのペリフェラルを作成するデザイン・フローを示します。

図 2. SOPC Builder 用のペリフェラルを作成するためのデザイン・フロー



SOPC Builder コンポーネントの開発プロセスは、他のロジック・ブロックの開発プロセスと同様です。このプロセスは、ペリフェラルの規格化と要求される機能の定義から始まります。ペリフェラル開発は、ハードウェア・ロジック・デザインとソフトウェア・ドライバ定義の2つの方向で進められます。最終的に、ペリフェラルのハードウェアとソフトウェアは、他のシステム・コンポーネントとともに、1つのSOPCデザインに統合されます。図2に示すように、SOPC Builderを使用してシステムを統合します。SOPC Builderのコンポーネントを開発するときは、通常の開発プロセスで開始し、後でSOPC Builder インポート・ウィザードにより追加情報を提供して、コンポーネントの再利用性を向上させます。

## Avalon スレーブ・ペリフェラルのデザイン例:

このセクションでは、シンプルな Avalon スレーブ・ペリフェラルを作成するデザイン・プロセスについて詳しく説明します。デモンストレーションを目的として、Avalon スレーブ・ポートを備えたパルス幅変調器コンポーネントを作成します。デザイン・ファイルが提供されており、表示、変更、および実験が可能です。デザイン・ファイルについては、18 ページの「PWM コンポーネント・リファレンス・デザインの使用」を参照してください。

### ペリフェラル規格

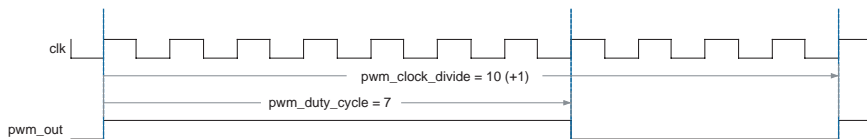
一般的な Avalon スレーブ・ペリフェラルのアーキテクチャは、次の機能ブロックで構成されます。

- ペリフェラル・タスク・ロジック - ペリフェラル・タスク・ロジックはペリフェラルの機能動作を実行します。
- レジスタ・ファイル — レジスタ・ファイルは、ペリフェラル・タスク・ロジックとの間の入力および出力用メモリ・エレメントを提供します。
- Avalon インタフェース — Avalon インタフェースは、レジスタ・ファイルへの標準アドレス・マップド・インタフェースを提供します。このインタフェースは、レジスタ・ファイルへのアクセスに必要なすべての信号タイプを提供し、タスク・ロジックで実装される転送タイプをサポートします。
- ソフトウェア・ドライバ関数 — ソフトウェア・ドライバ関数は、ペリフェラルへのソフトウェア・アプリケーション・インタフェースを提供します。ソフトウェア要件は、ペリフェラルのニーズに応じて変化します。最も一般的なルーチンは、ペリフェラルの初期化、ペリフェラルからのデータの読み出し、またはペリフェラルへのデータの書き込みを行います。

ペリフェラル規格の重要な点は、ペリフェラル・タスク・ロジック要件とソフトウェア要件を定義することです。これらの要件を定義すれば、レジスタ・ファイルと Avalon インタフェースが必要に応じて指定されます。

PWM コンポーネントは、変調されたデューティ・サイクルで方形波を出力します。基本的なパルス幅波形を図 3 に示します。

図 3. 基本的なパルス幅変調 (PWM) 波形



PWM コンポーネントは、次のとおり指定および作成されています。

- ペリフェラル・タスク・ロジックは、単一マスタ・クロックに同期して動作します。
- 適切な範囲の PWM 周期とデューティ・サイクルが得られるように、32 ビット値が使用されます。
- ホスト・プロセッサは、PWM 周期およびデューティ・サイクルの値を設定します。これは、コントロール・ロジックへのリード / ライト・インタフェースが必要であることを意味します。
- PWM 周期およびデューティ・サイクルの値を保持するために、レジスタ・エレメントが定義されます。
- ホスト・プロセッサは、enable コントロール・ビットを使用して PWM 出力を停止することができます。
- デザインの再利用を向上させるために、PWM ペリフェラルはすべてのデータ幅 (8、16、32 ビットなど) に適合しなければなりません。したがって、レジスタへのインタフェースは、個別バイトの読み出し / 書き込みをサポートします。

## ハードウェア開発

5 ページの図 2 に示すように、SOPC Builder コンポーネントのハードウェア開発は、以下の内容で構成されます。

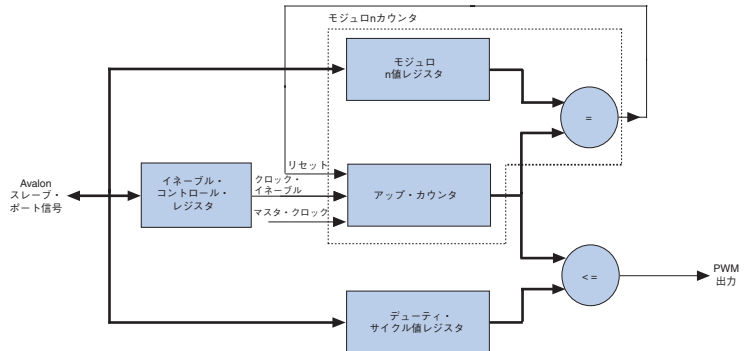
- タスク・ロジック、レジスタ・ファイル、および Avalon インタフェースの設計
- SOPC Builder へのデザイン・ファイルのインポート
- SOPC Builder システムへのコンポーネントのインスタンス化

どのロジック・デザイン・プロセスでもそうですが、SOPC Builder コンポーネントの開発は規格化フェーズ後に開始されます。コーディング・プロセスは、規格に対してHDLロジックを記述および検証する反復のプロセスです。SOPC Builder は開発プロセスを円滑にするための機能を提供します。あるいは、SOPC Builder 開発フローとはまったく異なるフローで、ペリフェラルの開発と検証を行うことも可能です。

### PWM タスク・ロジック

PWM タスク・ロジックは、入力クロック (clk)、出力信号 (pwm\_out)、イネーブル・ビット、32 ビット・モジュロ n カウンタ、および 32 ビット・コンパレータ回路で構成されます。clk は 32 ビット・モジュロ n カウンタをドライブして、pwm\_out 信号の周期を確立します。コンパレータは、モジュロ n カウンタの現在の値をデューティ・サイクルの値と比較して、pwm\_out 信号の出力を決定します。現在のカウンタ値がデューティ・サイクルの値以下であれば、pwm\_out 信号はロジック値 0 をドライブし、デューティ・サイクルの値以下でなければ、pwm\_out 信号はロジック値 1 をドライブします。基本的なタスク・ロジック構造を 8 ページの図 4 に示します。

図 4. 基本的な PWM ペリフェラル構造



### レジスタ・ファイル

レジスタ・ファイルで、enable ビット、モジュロ n 値、およびデューティ・サイクル値にアクセスできます。便宜上、これらのレジスタにはそれぞれリード・アクセス権とライト・アクセス権が与えられています。つまり、ソフトウェアは以前にレジスタに書き込まれていた値をリード・バックできます。これは設計者の選択によるものです。同様にすべてのレジスタがライト・オンリーになるように設計することもできました。この場合、オンチップ・ロジック・リソースは節約できますが、ソフトウェアが現在のレジスタ値をリード・バックすることはできなくなります。設計者は各レジスタを固有のアドレスにマップします。

レジスタ・ファイルおよびアドレス・マッピングを表 1 に示します。3 個のレジスタをサポートするには、2 ビットのアドレス・エンコーディングが必要なことに注意してください。この場合、4 番目の「予約」レジスタが発生します。

| レジスタ名        | オフセット | アクセス    | 説明                                  |
|--------------|-------|---------|-------------------------------------|
| clock_divide | 00    | リード/ライト | PWM 出力 1 サイクル中にカウントされるクロック・サイクル数です。 |
| duty_cycle   | 01    | リード/ライト | PWM 出力が Low になるクロック・サイクル数です。        |

表 1. レジスタ・ファイルおよびアドレス・マッピング ( 2 / 2 )

| レジスタ名    | オフセット | アクセス    | 説明   |
|----------|-------|---------|--|
| enable   | 10    | リード/ライト | PWM出力をイネーブル/ディセーブルします。ビットを0から1に設定すると、PWMがイネーブルになります。 |
| Reserved | 11    | N/A     | N/A  |

レジスタを読み出すまたは書き込むには、1 クロック・サイクルが必要です。13 ページの「Timing タブ」で説明するように、これは Avalon インタフェースのウェイト・ステートを指定する方法に影響を与えます。Avalon スレーブ・ポートは、任意の固定ウェイト・ステート・サイクル数を使用するように指定でき、また可変クロック・サイクル数を使用することもできます。いずれの場合も、SOPC Builder は適切なインタコネクタ・ロジックを生成して、マスタ・ポートとスレーブ・ポート間で確実に適切な動作が実行されるようにします。

### Avalon スレーブ・ポート

Avalon スレーブ・ポートは、スイッチ・ファブリックとペリフェラルの間のインタフェースです。Avalon スレーブ・ポートは、コンフィギュレーション可能なインタフェースであり、設計者がペリフェラル・ロジックを構築する際の柔軟性が大幅に向上します。入力および出力セットが固定された厳密なバス規格へのインタフェースを設計する代わりに、Avalon インタフェースを利用すれば、設計者はペリフェラルがサポートする特定の Avalon 転送タイプに必要な入力と出力だけを含めるようにすることができます。

SOPC Builder は、すべての Avalon インタフェース信号タイプとその役割を認識します。SOPC Builder で生成された Avalon スwitch・ファブリックは、各マスタ・スレーブ・ペア間での転送機能を調和させます。

PWM ペリフェラルの Avalon インタフェースは、レジスタへのシンプルなリードおよびライト・アクセスを処理するための限定された信号セットのみ必要とします。Avalon インタフェース規格を検討すると、次の2つの基本転送タイプが必要なことが明らかです。

- ウェイト・ステートが固定された Avalon スレーブ・リード転送
- ウェイト・ステートが固定された Avalon スレーブ・ライト転送

表 2 に、これらの転送タイプの実装に必要な Avalon 信号タイプを、HDL デザイン・ファイルで定義される各信号の名前とともに示します。

| HDL での信号名  | Avalon 信号タイプ | ビット幅 | 方向 | 注                                  |
|------------|--------------|------|----|------------------------------------|
| clk        | clk          | 1    | 入力 | データ転送およびペリフェラル・タスク・ロジック用プライマリ・クロック |
| write_n    | write        | 1    | 入力 | ライト・イネーブル信号 (アクティブ Low)            |
| write_data | writedata    | 32   | 入力 | 32 ビットのライト・データ値                    |
| byte_en_n  | byteenable   | 4    | 入力 | 特定のバイト・レーンをイネーブルする信号               |
| chip_sel   | chipselect   | 1    | 入力 | チップ・セレクト信号                         |
| addr       | address      | 2    | 入力 | 2 ビット・アドレス (3 つのエンコーディングのみ使用)      |
| reset_n    | reset        | 1    | 入力 | リセット信号 (アクティブ Low)                 |
| read_data  | readdata     | 32   | 出力 | 32 ビットのリード・データ値                    |

これらの信号のタイプと幅で、Avalon マスタ・ポートはデータを PWM コンポーネントに転送できます。PWM ペリフェラルは、オプションのバイト・イネーブル信号をサポートしているため、マスタ・ポートによって 32 ビット未満のデータ幅でアドレス指定できます。Avalon インタフェースでは、信号のネーミング規則は定義されていません。インポート・プロセスの一部として、SOPC Builder は各信号名を Avalon 信号タイプに対応させます。この情報は、コンポーネントの **class.ptf** コンポーネント記述子ファイルに追加されます。

### ペリフェラルの検証

ペリフェラルの検証は、一般に各ステージで完了します。ペリフェラル・タスク・ブロックは、標準の動作シミュレーション・ツールおよび RTL シミュレーション・ツールとテスト・ベンチを使用して検証できます。このシンプルなペリフェラルのレジスタ・ファイルと Avalon 信号タイプを追加した結果は、多くの場合、テスト・ベンチをわずかに変更するだけで検証できます。ペリフェラルを SOPC Builder にインポートした後で、アルテラ Nios エンベデッド・プロセスまたは他のマスタ・ペリフェラルを使用して、シミュレーションまたはハードウェアのペリフェラルでリード/ライト転送を実行することもできます。Nios プロセスは、カスタム・ペリフェラルの検証をカスタム・ペリフェラルへの転送を開始する C コードの記述程度にする、堅牢なシミュレーション環境を提供します。



Nios プロセッサ・ベース・システムのシミュレーションに関する情報は、「AN 189: Simulating Nios Embedded Processor Designs」を参照してください。

## SOPC Builder へのデザイン・ファイルのインポート

ペリフェラルを SOPC Builder にインポートするには、ポートのタイプと動作を記述する **class.ptf** コンポーネント記述子ファイルを作成する必要があります。SOPC Builder では、**Interface to User Logic** と呼ばれる使いやすいインポート・ウィザードが利用でき、ユーザはこのウィザードを使用して、**class.ptf** ファイルを自動的に生成できます。**Interface to User Logic** ウィザードには、デザイン・ファイルのインポートおよびシステムへの接続のためのグラフィカル・ユーザ・インタフェース (GUI) が用意されています。**class.ptf** ファイルは任意のテキスト・エディタで編集できますが、**Interface to User Logic** ウィザードを使用することをお勧めします。

**Interface to User Logic** ウィザードでは、以下のことができます。

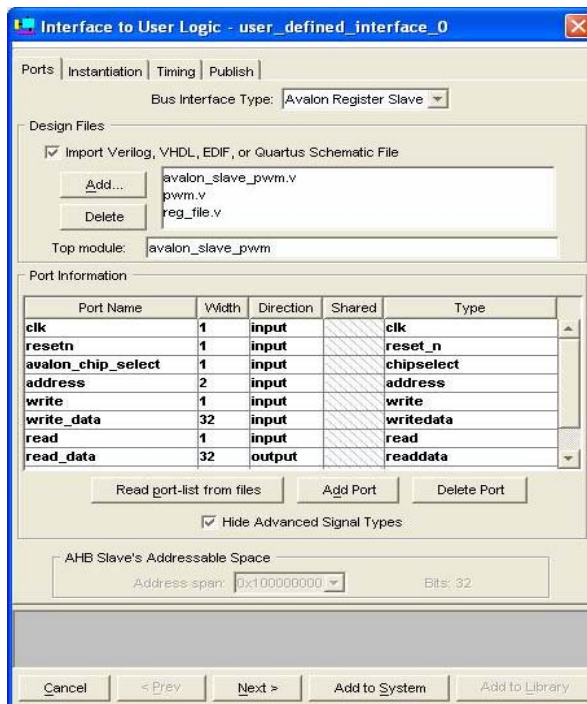
- Avalon マスタまたは Avalon スレーブなど、コンポーネント上のインタフェース・ポートのタイプを定義する
- コンポーネント・デザイン・ファイルのリストをインポートする
- 入力 / 出力信号名について、デザイン・ファイルを自動的にスキャンする
- 信号名を Avalon 信号タイプにマップする
- セットアップ・タイムとホールド・タイプ、またはウェイト・ステートなどのタイミング要求を指定する
- オプションで、利用可能なモジュールの SOPC Builder プールにペリフェラルを公表する

全体のシステムに統合された PWM ペリフェラルを探すには、付属のデザイン・ファイルを調べることができます。このセクションの残りでは、PWM をデザイン全体に統合するための作業について説明します。

## Ports タブ

図 5 に、PWM ペリフェラルへの接続に使用する **Interface to User Logic** ウィザードの **Ports** タブを示します。

図 5. PWM ペリフェラルに対する Ports タブ



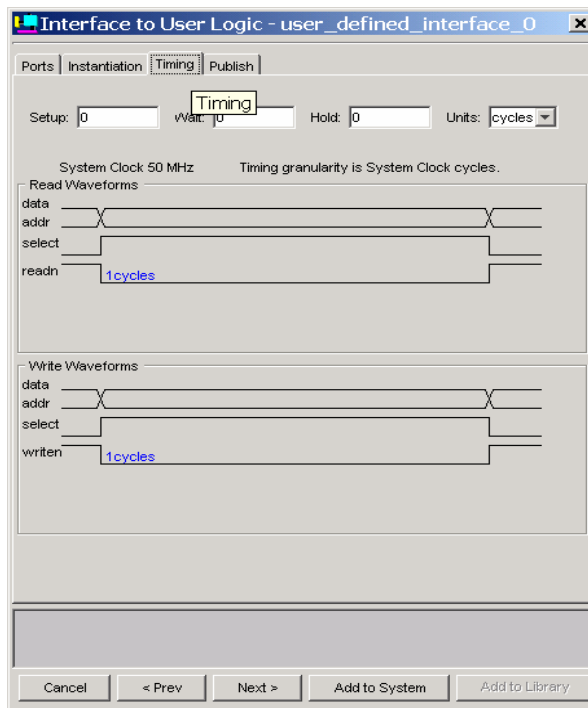
**Interface to User Logic** ウィザードを使用して PWM ペリフェラルをインポートする場合、最初のステップは **Bus Interface Type** を選択することです。PWM ペリフェラルについて、**Avalon Register Slave** を選択しています。Avalon Register Slave は、レジスタ・マップド・ペリフェラルをシステムに接続するのに選択します。レジスタ・マップド・スレーブ・ペリフェラルを使用する場合は、SOPC Builder がネイティブ・アドレス・アラインメントを使用して Avalon スレーブ・ポートを接続する必要があります。言い換えれば、Avalon スイッチ・ファブリックはスレーブ・ポートに対してダイナミック・バス・サイジングを実行しないため、スレーブ・ポートへの 1 回のマスタ転送で、1 回のスレーブ転送が発生します。ネイティブ・アドレス・アラインメントおよびダイナミック・バス・サイジングについて詳しくは、「Avalon Interface Specification Reference Manual」を参照してください。

**Bus Interface Type** を選択した後に、デザイン・ファイルをインポートします。ウィザードで、**Import Verilog, VHDL, EDIF or Quartus Schematic File** オプションをオンにすると、コンポーネントをシステム・モジュールの階層構造内に配置し、それを Avalon スイッチ・ファブリックに配線するよう SOPC Builder に指示します。このオプションをオフにすると、SOPC Builder は単に Avalon スレーブ・インタフェースをシステム・モジュール上に引き出すだけで、設計者は手動でペリフェラルを接続する必要があります。**Add** ボタンをクリックすると、デザイン・ファイルがあるディレクトリに移動できます。この場合、**avalon\_slave\_pwm** と呼ばれるトップレベル・モジュールとともに、1つのファイル **avalon\_slave\_pwm.v** がインポートされます。**Read port-list from files** ボタンをクリックすると、入力 / 出力ポートのデザイン・ファイルがスキャンされ、トップレベル・モジュール上に存在する信号名が自動的にポート・リストに入力されます。最後のステップでは、選択した信号名を **Port Information** テーブルの Avalon 信号タイプにマップします。**pwm\_out** などの非 Avalon 信号は、タスク・ロジックの一部として機能するもので、タイプ **Export** にマップする必要があります。SOPC Builder は、これらの信号をトップレベル・システム・モジュール上のポートとして引き出します。

### Timing タブ

図 6 に、PWM ペリフェラルの接続に使用する **Interface to User Logic** ウィザードの **Timing** タブを示します。

図 6. PWM ペリフェラルに対する Timing タブ



**Timing** タブでは、SOPC Builder が Avalon スイッチ・ファブリックの生成に使用するタイミング・パラメータを設定することができます。PWM ペリフェラルの場合、レジスタ・ファイルへのリードおよびライト転送は、セットアップ・タイム、ホールド・タイム、またはウェイト・ステートを必要としない同期インタフェースとして設計されました。**Read Waveforms** 図および **Write Waveforms** 図は、設定に基づいて変化して Avalon 転送を示します。Avalon マスタ・ペリフェラルに対しては、**Timing** タブはディセーブルされます。

この時点で、**Add to System** をクリックすると、システムにペリフェラルが追加されます。後でパラメータの変更が必要な場合は、ウィザードを再び開いて値を変更し、**Add to System** をクリックして変更を保存します。

コンポーネントをシステムに追加した後、SOPC Builder **System Contents** ページで基本アドレスと IRQ の設定を行うことができます。マスタ・コンポーネントをシステムに追加すると、システム生成を実行できます。SOPC Builder は、少なくとも 1 つのマスタ・ポートと 1 つのスレーブ・ポートがなければ、システムを生成できません。生成プロセスの間に、SOPC Builder はペリフェラルの HDL をプロジェクト・ディレクトリにコピーし、適切な Avalon スイッチ・ファブリックを生成して、このペリフェラルをシステムに接続します。

## SOPC Builderコンポーネントとしてのペリフェラルの公開

Interface to User Logic 機能を使用すれば、オプションでペリフェラルを公開して、SOPC Builder の利用可能モジュールのプールに表示させることができます。公開プロセスでは、パラメータ値がロックされ、デザイン・ファイルが 1 つのディレクトリにパッケージされます。これによって、コンポーネントを他のシステムで再利用可能な単一ユニットとして配布できるようになります。すべての設定がロックされるため、他のユーザに Avalon インタフェースのコンフィギュレーション方法を説明する必要はありません。公開後には、パラメータは変更できません。したがって、まずペリフェラルをシステムに正しく追加し、システムが正常に動作することをテストした後で公開する必要があります。

ペリフェラルを公開すると、情報がこのコンポーネントの **class.ptf** ファイルに記録され、同じペリフェラルの名前でサブフォルダ内に生成スクリプトが作成されます。表 3 にサンプルの PWM コンポーネントを使用して、公開されたコンポーネントのディレクトリ構造を示します。

| ファイル名                            | 説明  |
|----------------------------------|---|
| ...\user_logic_altera_avalon_pwm | コンポーネント・ディレクトリ  |
| altera_avalon_pwm.v              | ペリフェラルの HDL デザイン・ファイル                                 |
| class.ptf                        | コンポーネント記述子ファイル  |
| mk_user_logic_altera_pwm.pl      | SOPC Builder がシステム生成時に、ペリフェラルをシステムに統合するために使用する生成スクリプト |

## ソフトウェア開発

SOPC Builder を使用すると、ペリフェラル設計者はソフトウェア・ファイルをペリフェラルと関連付けることができます。これらのファイルは、SOPC Builder で生成されたシステム全体に対するソフトウェア開発キット (SDK) ディレクトリの一部となります。関連のソフトウェア・ファイルは、以下で構成されます。

- ペリフェラルに関連するソフトウェア構造および定数を宣言するヘッダ・ファイル
- ペリフェラルにアクセスするペリフェラル・ドライバ関数

エンベデッド・システムで使用される大部分のペリフェラルと同様に、SOPC Builder コンポーネントは、ハードウェアのソフトウェア・ビューを定義するヘッダ・ファイルを 1 つ (または複数) インクルードする必要があります。ヘッダ・ファイルには、ペリフェラルのベース・アドレスに関連するレジスタ宣言、プロトタイプ関数宣言、定数、およびその他のコンフィギュレーション情報が含まれます。

ソフトウェア・ファイルを供給する方法はシンプルです。各 SOPC Builder コンポーネント・ディレクトリには、SDK ディレクトリを含めることができます。このディレクトリは、SOPC Builder で生成された完全なシステムに対する SDK の構造を反映しています。システム生成時に SOPC Builder は、コンポーネントの SDK ディレクトリからシステム全体で対応する SDK の場所にファイルをコピーします。さらに、SOPC Builder は、コンポーネントのヘッダ・ファイル (1 つまたは複数) からシステム全体のヘッダ・ファイル (**excalibur.h**) に宣言をコピーします。

現時点では、SDK ディレクトリとそのサブディレクトリを手動で作成して、ペリフェラルのソフトウェア・サポートを追加する必要があります。

### PWM コンポーネントの SDK

PWM の例を使用して、公開された SOPC Builder コンポーネントのディレクトリ構造を [図 7](#) に示します。手動で作成するディレクトリは、太字で示してあります。

図 7. PWM コンポーネントの SDK ディレクトリ構造



上図に示すように、SDK ディレクトリには次のディレクトリが含まれています。

- **inc** – ヘッダ・ファイルのインクルード・ディレクトリ
- **lib** – ペリフェラル・ドライバ関数を格納するライブラリ・ディレクトリ
- **src** – アプリケーションでのペリフェラル・ドライバ関数の使用法を示すソース・コード例を格納するソース・ディレクトリ

この例では、**inc** ディレクトリ内のヘッダ・ファイルには、PWM ドライバ関数のレジスタ・マップ情報と関数プロトタイプが含まれています。

**lib** ディレクトリ内のファイルには、初期化、PWM 出力のイネーブルとディセーブル、そして PWM 出力のデューティ・サイクと周期を変更する低レベル・ペリフェラル関数が含まれています。

**src** ディレクトリ内のファイルには、LED をドライブする PWM をデモンストレーションするアプリケーション例が含まれています。

PWM ペリフェラルでは、以下の関数が提供されます。

- `int init_Altera_Avalon_PWM(unsigned int address, unsigned int clock_divider, unsigned int duty_cycle);`
- `int enable_Altera_Avalon_PWM(unsigned int address);`
- `int disable_Altera_Avalon_PWM(unsigned int address);`
- `int change_Altera_Avalon_PWM_Duty_Cycle(unsigned int address, unsigned int duty_cycle);`

各関数はパラメータ・アドレスを受け取ります。このアドレスは、PWM コンポーネントの特定インスタンスのベース・アドレスです。

## システム・インテグレーション

PWMコンポーネントを公開した後、複数のPWMコンポーネントをSOPC Builderシステムに追加することができます。システム生成時に、SOPC Builderは、システム内のすべてのコンポーネントに対して **class.ptf** システム記述子ファイルを読み出し、コンポーネントのインタコネクットのニーズに合わせて Avalon スイッチ・ファブリックを構築します。さらに SOPC Builder は、SDK ディレクトリのほかに、システム・レベル検証用のテストベンチを備えた ModelSim プロジェクト・ファイルも構築します。

## システム・レベル検証

SOPC Builder には、ModelSim を使用したシステム・レベル検証のサポートが組み込まれています。SOPC Builder はシステム検証用のテストベンチを作成しますが、シミュレーション環境の能力は、システムに含まれるペリフェラルによって決まります。システム内に Nios プロセッサが存在する場合、ペリフェラルをドライブするアプリケーション・ソフトウェアを容易にシミュレートできます。



詳しくは、「AN189: Simulating Nios Embedded Processor Designs」を参照してください。

## PWM コンポーネント・リファレンス・デザインの使用

PWM コンポーネントのリファレンス・デザイン・ファイルは、次のソフトウェアおよびハードウェアを使用して作成されました。

- Quartus II バージョン 4.0
- Nios 開発キット、バージョン 3.1

このリファレンス・デザインは、「現状」でも Nios 開発ボード、Stratix Edition、Stratix Professional Edition、および Cyclone Edition 上で動作しますが、デザイン・ファイルを修正すればユーザー独自の開発ボード上でも使用できます。PWM リファレンス・デザインでは、Nios プロセッサを使用して Avalon PWM コンポーネントをドライブします。PWM 出力は、Nios 開発ボードの LED に接続されます。LED をドライブする信号を変調すると、LED の輝度変化します。

Nios 開発キットがインストールされていない場合、次のステップを実行しても正しく機能しません。デザイン例を Nios 開発ボードにダウンロードして、サンプル・コードを実行するには、これらのステップに従います。

1. 以下に示すように、使用するNios開発ボード用の付属のzipファイルを解凍します。
  - Cyclone エディション - Avalon\_PWM\_1C20
  - Stratix エディション - Avalon\_PWM\_1S10
  - プロフェッショナル・エディション - Avalon\_PWM\_1S40

zip ファイルには、PWM ペリフェラルを搭載した既製システムの Quartus II プロジェクト・ディレクトリが含まれています。

2. **Quartus II Programmer** ツールを使用して、トップレベルの Quartus II プロジェクト・ディレクトリにある SOF ファイルでアルテラ FPGA をコンフィギュレーションします。

3. Nios SDK Shell を開き、次のように入力して sdk/src ディレクトリにアクセスします。

```
cd <デザイン例のインストール・ディレクトリ>/cpu_sdk/src
```

4. プロンプトで次のように入力して、デザイン例のソフトウェア・コードをコンパイルします。

```
nios-build hello_avalon_slave_pwm.c
```

5. プロンプトで次のように入力して、ソフトウェア・コード例をダウンロードおよびコンパイルします。

```
nios run hello_avalon_slave_pwm.srec
```

Quartus II 開発ソフトウェアおよび SOPC Builder を使用すれば、プロジェクト・ハードウェアの再構築、他のペリフェラルの追加、およびアプリケーションの変更を行うことも可能です。

## デザイン例: ストリーミング 転送をサポート する Avalon スレーブ・ポート

このセクションでは、Avalon スレーブ・ポートの異なるコンフィギュレーションをデモンストレーションする、別のデザイン例を示します。このセクションのデザインは、Avalon ストリーミング転送モードをサポートする Avalon スレーブ・ポートを使用する VGA ディスプレイ・コントローラです。デザイン・ファイルが提供されており、表示、変更、および実験が可能です。このデザイン・ファイルについて詳しくは、[27 ページの「リファレンス・デザインの使用」](#)を参照してください。

### Avalon ストリーミング転送

Avalon インタフェース規格には、帯域幅を変更して性能を向上させたり、ペリフェラルのデザインを簡略化する高度な転送機能が含まれています。Avalon ストリーミング転送は、マスター・スレーブ・ペア間におけるオープン・チャンネルとして定義されており、フロー制御信号を使用した連続データ転送を可能にします。ストリーミング転送モードを使用すれば、メモリとペリフェラル間でのバルク・データ転送を管理するシンプルなマスター・ペリフェラル (DMA コントローラなど) での、システムのデザインが簡略化されます。マスター・スレーブ・ペア間にあるこのオープン・チャンネルによって、データは利用可能になると流れるため、マスターがスレーブ・ペリフェラル内のステータス・レジスタを継続的にポーリングして、データの送信または受信が可能かどうかを判断する必要はありません。ストリーミング転送によって、マスター・スレーブ・ペア間のスループットが最大になると同時に、スレーブ・ペリフェラルでのデータのオーバーフローやアンダーフローを回避します。

Avalon インタフェース規格は、ストリーミング転送をサポートしていないシステム内のストリーミング Avalon ペリフェラルに対しても、デザインの再利用をサポートしています。ストリーミング転送を実現するには、マスタ・ポートとスレーブ・ポートの両方がストリーミング転送をサポートしていることが必要です。マスタ・ポートまたはスレーブ・ポートのいずれかがストリーミング転送をサポートしていない場合、データ転送は基本的な Avalon リードおよびライト転送に戻ります。ストリーミングをサポートする、またはサポートしないマスタ・ポートとスレーブ・ポートのどのような組み合わせに対しても、SOPC Builder は自動的に適切な Avalon スイッチ・ファブリックを生成します。

## ペリフェラル規格

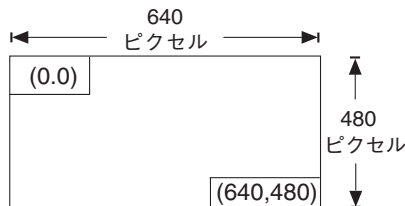
ストリーミングをサポートする一般的な Avalon スレーブ・ペリフェラルのアーキテクチャは、以下の機能ブロックで構成されます。

- ペリフェラル・タスク・ロジック
- レジスタ・ファイル
- Avalon インタフェース
- ソフトウェア・ドライバ関数

ペリフェラル規格の重要な点は、ペリフェラル・タスク・ロジック要件とソフトウェア要件を定義することです。これらの要件を定義すれば、次にレジスタ・ファイルと Avalon インタフェースが指定されます。

標準 VGA モニタは、水平および垂直同期信号で制御する所定の方法で画面をリフレッシュします。図 8 に示すように、画面は 480 行と 1 行あたり 640 ピクセルで構成されるピクセルのグリッドで構成されています。各ピクセルの色は、赤 (R)、緑 (G)、青 (B) の 3 つのアナログ信号の値に基づいています。

図 8. VGA モニタのピクセル構成

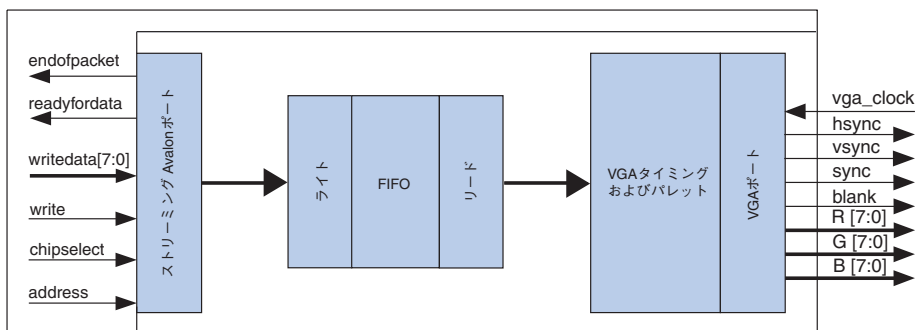


各画面のリフレッシュ・サイクルは、画面の左上隅のピクセルの更新で始まり、この位置は、X-Y 面の原点として扱うことができます。最初のピクセルがリフレッシュされた後、モニタはその行の残りのピクセルをリフレッシュします。モニタは水平同期でパルスを受け取ると、次の行のピクセルをリフレッシュし始めます。この処理はモニタが画面の下部に達するまで繰り返されます。モニタが最後の行およびその行の最後のピクセル（つまり、[640, 480]）に達すると、垂直同期パルスによって、モニタは最初の行の最初のピクセル（つまり、[0, 0]）に戻ります。画面のリフレッシュは連続かつ VGA 規定のタイミングで実行されることが必要です。

ペリフェラル・タスク・ロジックは、8 ビットのピクセル値を受け取り、それらの値を個別の 8 ビット R、G、B コンポーネント値に変換し、外部デジタル-アナログ (D/A) コンバータに供給するように設計されています。さらにこのコントローラは、水平および垂直同期信号を供給するように設計されています。信号のタイミングは、モジュールへの 25 MHz 入力クロックに基づいています。8 ビット R、G、および B 信号は通常デバイス・ピンに接続され、外部 D/A コンバータに供給されます。

1024 バイト FIFO はローカル・データ・バッファを提供します。これは 1.6 ロウ分のピクセル・データに相当します。コントローラ FIFO バッファは、バッファが 1000 バイトを超えるとバッファ・フル信号を供給します。またバッファは、フレームの最後のピクセルが FIFO に書き込まれたことを示す信号も出力します。コントローラには、バッファ・フル信号および最終ピクセル信号以外に、データ・フローを制御するための他の入力や出力はありません。図 9 に、基本的なハードウェア構造を示します。

図 9. VGA ディスプレイ・ドライバ・ハードウェアのブロック図



このペリフェラルのソフトウェア要件は、ペリフェラルのベース・アドレスのためのヘッダ・ファイルをインクルードすることだけです。マスタ・コンポーネントは、ピクセル・データをベース・アドレスに書き込むことしか必要ありません。

## ハードウェア開発

次のセクションでは、ハードウェア・デザインの構造について説明します。

### VGA コントローラ・タスク・ロジック

VGA コントローラ・タスク・ロジックのトップレベル・モジュールは、**vga\_controller\_stream** という名前で、ファイル **vga\_controller\_stream.v** に含まれています。トップレベル・モジュールは、フレームの先頭と末尾に同期し、さらに **vga\_timing** および **vga\_pixel\_fifo** の 2 つのサブモジュールをインスタンス化します。

**vga\_timing** モジュールは、ファイル **vga\_timing.v** 内にあり、水平および垂直同期信号のタイミングを制御します。またこのモジュールは、ルック・アップ・テーブルを使用して、8 ビットのピクセル値を 8 ビットの R、G、および B 値に変換します。色のパレットは、16 色と黒および白に制限されます。この範囲外のピクセル値の場合は、黒が表示されます。

**vga\_pixel\_fifo** モジュールは、**vga\_pixel\_fifo.v** ファイルに含まれており、**dcfifo** と呼ぶアルテラのデュアル・クロック FIFO メガファンクションをインスタンス化したものです。**dcfifo** メガファンクションは、アルテラ・デバイスで利用可能なデュアル・ポート・メモリ構造を利用して、FIFO の書き込みと読み出しのための独立したクロック・ドメインを提供します。**wrusedw[ ]** 信号は、FIFO に格納されているワード数を表し、上記のバッファ・フル信号の生成に使用されます。

### レジスタ・ファイル

ペリフェラルは、単一 8 ビット・メモリ位置と同様に動作します。8 ビット FIFO へのライト・ポートは、ペリフェラルのベース・アドレスにマップされます。8 ビット・データ・ストリームは入力専用なので、このペリフェラルは 1 つの書き込み専用アドレスしか必要ありません。レジスタを必要とするコンフィギュレーション・オプションはありません。ペリフェラルからの唯一のステータス信号（つまり、バッファ・フル）は、Avalon ストリーミング・インタフェースのフロー制御信号で処理されます。したがって、ペリフェラルは 1 つの 8 ビット・メモリアクセスのように動作します。表 4 にシンプルなレジスタ・マップを示します。

| 表 4. レジスタ・ファイルおよびアドレス・マッピング |       |        |  |
|-----------------------------|-------|--------|--|
| レジスタ名                       | オフセット | アクセス   | 説明   |
| vga_data                    | 0x0   | 書き込み専用 | このレジスタに書き込むと、8 ビット値が FIFO ライン・バッファに格納されます。 |

### ストリーミング転送をサポートする Avalon スレーブ・ポート

スループットを向上させ、ピクセル・データの供給に付随するオーバヘッドを小さくするために、Avalon ストリーミング転送モードを使用することにしました。Avalon 信号の `readyfordata` はデータのフローを制御し、`endofpacket` は VGA コントローラがフレームをすべて受信したことを知らせます。このシステムでは、Avalon DMA コントローラ (SOPC Builder でインクルードされる) は、メモリから VGA パリフェラルへのバルク・データ転送を実行します。DMA コントローラは、ストリーミング転送をサポートしているため、`readyfordata` および `endofpacket` 信号を認識します。これらの信号で行うハンドシェイクによって、インテリジェント・ホストが新しいデータを送信する前に、VGA パリフェラルのステータスをチェックする必要はありません。

VGA コントローラには書き込む位置が 1 つしかないため、アドレス・ラインは不要です。`chipsselect` 信号だけで、Avalon 転送が発生していることを確認するのに十分です。タスク・ロジックの一部として機能する非 Avalon 信号 (R、G、B、同期出力など) は、**Export** ポート・タイプにマップする必要があります。SOPC Builder は、これらの信号をトップレベル・システム・モジュールのポートとして引き出します。

図 5 に、Avalon 信号タイプに対するパリフェラルのポート名の全マッピングを示します。

| ポート名                       | Avalon 信号タイプ              | ビット幅 | 方向 | 説明  |
|----------------------------|---------------------------|------|----|---|
| <code>clock</code>         | <code>clk</code>          | 1    | 入力 | FIFO への書き込み用入力クロック                                |
| <code>reset</code>         | <code>reset</code>        | 1    | 入力 | パリフェラル・リセット                                       |
| <code>cs</code>            | <code>chipsselect</code>  | 1    | 入力 | チップ・セレクト  |
| <code>write</code>         | <code>write</code>        | 1    | 入力 | ライト・イネーブル信号                                       |
| <code>fifo_data</code>     | <code>writedata</code>    | 8    | 入力 | 8 ビットのライト・データ                                     |
| <code>fifo_not_full</code> | <code>readyfordata</code> | 1    | 出力 | 新しいデータが受け取られたことを示すストリーミング転送信号                     |
| <code>lastpixel</code>     | <code>endofpacket</code>  | 1    | 出力 | フレームの最後のピクセルを受信したことを示すストリーミング転送信号                 |
| <code>vga_clock</code>     | Export                    |      | 入力 | VGA タイミングおよび FIFO からのデータの読み出し用入力クロック              |
| <code>hsync</code>         | Export                    | 1    | 出力 | 水平同期信号 (出力)                                       |
| <code>vsync</code>         | Export                    | 1    | 出力 | 垂直同期信号 (出力)                                       |
| <code>sync</code>          | Export                    | 1    | 出力 | <code>hsync</code> と <code>vsync</code> の論理積 (出力) |
| <code>blank</code>         | Export                    | 1    | 出力 | 水平および垂直同期中のブランキング信号 (出力)                          |
| R                          | Export                    | 8    | 出力 | 赤色 (出力)   |
| G                          | Export                    | 8    | 出力 | 緑色 (出力)   |
| B                          | Export                    | 8    | 出力 | 青色 (出力)   |

## SOPC Builder へのデザイン・ファイルのインポート

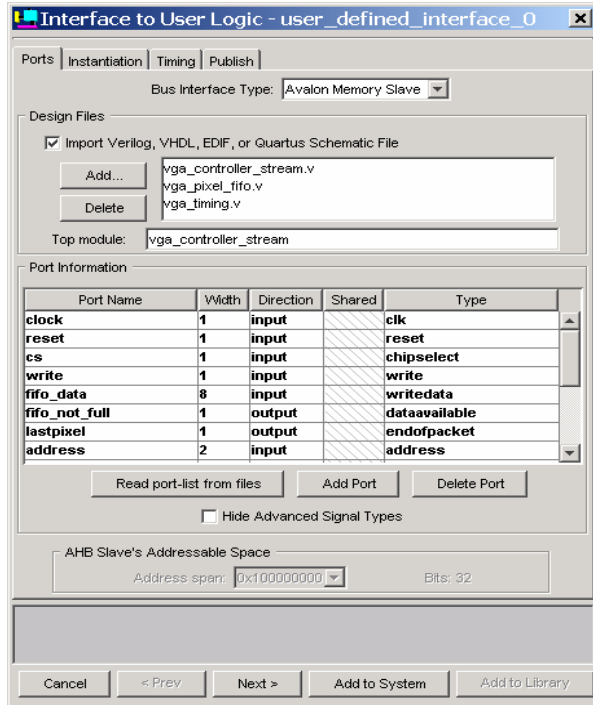
次の項では、**Interface to User Logic** ウィザードでの設定について説明します。

### Ports タブ

SOPC Builder の Systems Contents ページにある **Interface to User Logic** ウィザードを使用して、VGA コントローラ・デザイン・ファイルを Avalon メモリ・スレーブ・コンポーネントとして、SOPC Builder にインポートしました。これは、**Interface to User Logic** ウィザードのファイルのインポート結果を示します。25 ページの図 10 を参照してください。ペリフェラルを Avalon メモリ・スレーブとして指定すると、SOPC Builder はメモリ・デバイスに対応するアドレスを指定して Avalon スイッチ・ファブリックを構築します。具体的には、SOPC Builder はダイナミック・バス・サイジング・ロジックを生成します。ダイナミック・バス・サイジングを使用すれば、Avalon マスタ・ポートはペリフェラルのデータ幅より大きなデータを 1 回の Avalon 転送で送信することができます。Avalon スイッチ・ファブリックは、転送を適当な回数の別々の転送に分割して、すべてのデータをスレーブ・ポートに送信します。ダイナミック・バス・サイジングについては、「Avalon Interface Specification Reference Manual」を参照してください。

3 つの Verilog HDL ファイルには、VGA コントローラ・デザインのすべてが含まれ、各ファイルはデザイン・ファイルのリストに追加する必要があります。信号名を Avalon 信号タイプにマッピングするときには、**Hide Advanced Signal Types** オプションをオフにして、readyfordata および endofpacket 信号を表示させる必要があります。このコンポーネントに対する **Interface to User Logic** ウィザードの **Ports** タブを、図 10 に示します。

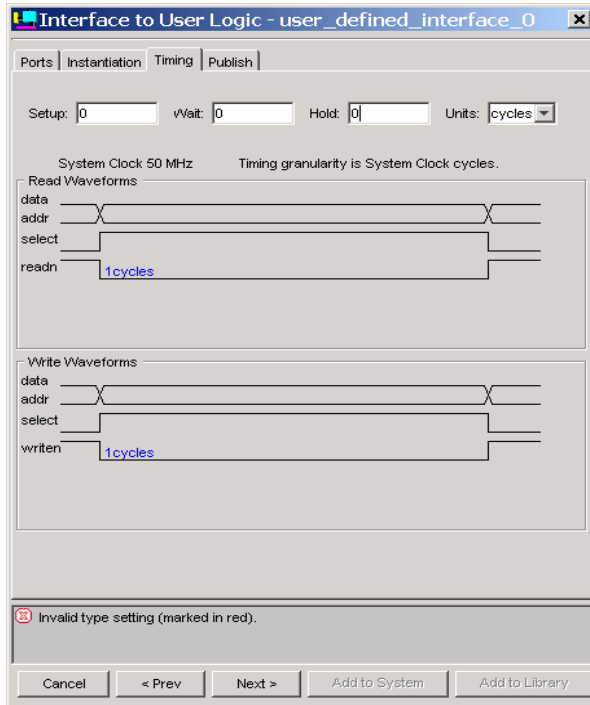
図 10. VGA コントローラ・ペリフェラルに対する Ports タブ



### Timing タブ

VGA コントローラ・ペリフェラルの場合、FIFO ライト・ポートは同期式シングル・サイクル・ライト・ポートなので、余分なウェイト・サイクルを追加する必要はありません。VGA コントローラ・ペリフェラルの接続に使用する **Interface to User Logic** ウィザードの **Timing** タブを、26 ページの図 11 に示します。

図 11. VGA コントローラ・ペリフェラルに対するタイミング・タブ



### Publish タブ

このペリフェラルを SOPC Builder 対応のコンポーネントとして公開するプロセスは、PWM ペリフェラルの例と同じです。詳しくは、15 ページの「[SOPC Builder コンポーネントとしてのペリフェラルの公開](#)」を参照してください。

## ソフトウェア 開発

インタフェースが非常にシンプルなため、このペリフェラルに提供されるソフトウェア機能はありません。このデザインは、Avalon DMA コントローラ・コンポーネントを使用して、メモリから VGA コントローラにイメージ・データをコピーします。したがって、VGA コントローラを使用するには、DMA コントローラ・コンポーネントで提供されるドライバ・ルーチンを使用して、DMA 転送をコンフィギュレーションし、開始します。

詳しくは、「[Nios DMA Data Sheet](#)」を参照してください。



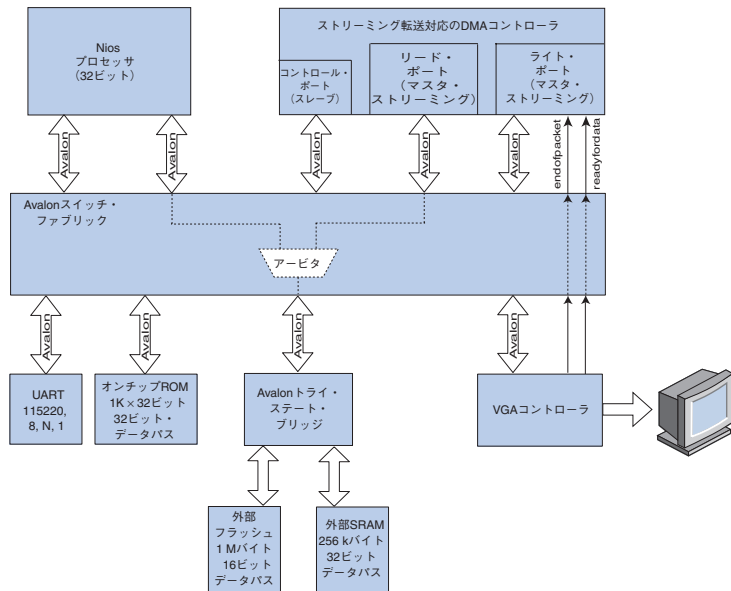
## リファレンス・デザインの使用

次のセクションでは、VGA リファレンス・デザインを使用する方法について説明します。

### 概要

図 12 に、マスタ - スレーブ・ペアを備えたシステム例のブロック図を示します。このマスタ - スレーブ・ペアは、ストリーミングをサポートする Avalon DMA コントローラおよびVGA コントローラで形成されています。また、システムはNios プロセッサも搭載しているため、メモリへのイメージ・データのダウンロードおよび DMA コントローラのコンフィギュレーションが可能です。ただし、ハードウェア・アーキテクチャに関しては、Nios プロセッサに限定されないため、他のどのプロセッサでも使用できます。

図 12. ストリーミング Avalon スレーブを搭載したシステム例



このシステムは、Nios 開発キットに付属の **standard\_32** リファレンス・デザインに基づいています。Avalon DMA コントローラがデザインに追加され、DMA レンダリング・レジスタ・セットの幅が 19 ビットに設定されているため、1 つの VGA イメージを一度に転送することが可能です。これによって、DMA コントローラは、1 つの完全な VGA フレームを一度 (640 x 480 <sup>2</sup>) に転送できます。DMA コントローラのストリーミング・サポートは、ストリーミング・スレープ・ポートに接続されるとイネーブルされます。endofpacket 信号は、DMA コントローラ内のコントロール・レジスタをコンフィギュレーションするとイネーブルされます。リファレンス・デザインでは、VGA コントローラが endofpacket 信号をアサートすると、DMA コントローラがプロセッサへの割り込みを生成します。IRQ が発生すると、Nios プロセッサの割り込みサービス・ルーチンは次の処理を実行します。

- 割り込みをクリアする
- ピクセル・データ・ソースを更新する
- DMA コントローラ・レジスタを設定して、別の DMA 転送をコンフィギュレーションする
- DMA 転送を開始する

### デザイン・ファイルの使用

VGA コントローラ・ペリフェラルは、Lancelot VGA 開発キットのドータ・ボードを使用してハードウェアでテストされています。標準 VGA モニタは Lancelot ドータ・ボードに接続され、このボードが Nios 開発ボードに接続されます。このデザイン例では、Lancelot ドータ・ボードは Nios 開発ボードの J8、J9、および J10 に接続されます。Lancelot VGA 開発キットのオーダー情報は、アルテラ Web サイト ([www.altera.com](http://www.altera.com)) の開発キット・セクションに記載されています。

Lancelot VGA ドータ・カードを実装した Nios 開発ボードでデザイン例を実行するには、次のステップに従ってください。

1. 以下に示すように、使用する Nios 開発ボード用の付属の zip ファイルを解凍します。
  - Cyclone エディション - streaming\_slave\_1C20
  - Stratix エディション - streaming\_slave\_1S10
  - プロフェッショナル・エディション - streaming\_slave\_1S40

zip ファイルには、VGA コントローラを搭載した既製システムの Quartus II プロジェクト・ディレクトリが含まれています。

2. Quartus II Programmer を使用して、トップレベルの Quartus II プロジェクト・ディレクトリにある SOF ファイルでアルテラ FPGA をコンフィギュレーションします。
3. Nios SDK Shell を開き、次のように入力してディレクトリを変更します。

```
cd <デザイン例のインストール・ディレクトリ>/cpu_sdk/src/image
```

4. プロンプトで次のように入力して、アドレス 0x810400 にある SRAM に 256 色 (8 ビット) 640x480 の画像をロードします。

```
nios-run car_256color.srec
```

また、`nios-run` コマンドを使用して、ユーザ独自のイメージの内容が格納された **Intel-Hex** ファイルまたは **srec** ファイルをダウンロードすることもできます。Lancelot VGA 開発キットには、ビットマップ (**.bmp**) イメージを S レコード・フォーマットに変換するソフトウェア・アプリケーションが含まれています。

5. プロンプトで次のように入力して、`.../cpu_sdk/src` ディレクトリにあるアプリケーション・ソフトウェアを再構築します。

```
cd..  
  
nios-build streaming.c
```

6. プロンプトで次のように入力して、ソフトウェア・コードの例をダウンロードおよびコンパイルします。

```
nios-run streaming.srec
```

これらのステップを完了すれば、他のペリフェラルの追加、SOPC Builder プロジェクトの再構築、Quartus II プロジェクトの再構築、アプリケーション・ソフトウェアの編集や再構築を行うことによって、デザイン・ファイルを実験することができます。



101 Innovation Drive  
San Jose, CA 95134  
(408) 544-7000  
[www.altera.com](http://www.altera.com)  
Applications Hotline:  
(800) 800-EPLD  
Literature Services:  
[lit\\_req@altera.com](mailto:lit_req@altera.com)

Copyright © 2004 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, スタイル化されたアルテラのロゴ、各製品名、商標またはサービス・マーク表示されている単語およびロゴは、特に指定のない限り、Altera Corporation の米国およびその他の国における商標またはサービス・マークです。その他の製品およびサービス名はそれぞれの所有者に帰属しています。Altera products are protected under numerous U.S. and foreign patents and pending applications, maskwork rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

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。



I.S. EN ISO 9001