



Qsys システム・デザイン チュートリアル



101 Innovation Drive
San Jose, CA 95134
www.altera.com

TU-01006

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。

ドキュメント・バージョン : 11.0
ドキュメント・デート : 2011 年 4 月



Subscribe

© 2011 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS and STRATIX are Reg. U.S. Pat. & Tm. Off. and/or trademarks of Altera Corporation in the U.S. and other countries. All other trademarks and service marks are the property of their respective holders as described at www.altera.com/common/legal.html. 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. 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.



第1章．はじめに

ソフトウェア要件およびハードウェア要件	1-1
概要	1-2
デザイン例ファイルのダウンロードおよびインストール	1-3
チュートリアル・プロジェクトを開く	1-4

第2章．Qsys システムの作成

データ・パターン・ジェネレータの生成	2-1
クロック・ソース付きの新規 Qsys システムを作成	2-2
パイプライン・ブリッジの追加	2-2
カスタム・パターン・ジェネレータの追加	2-3
PRBS パターン・ジェネレータの追加	2-4
2:1 ストリーミング・マルチプレクサの追加	2-5
メモリ・アドレス・マップの検証	2-6
リセット信号の接続およびアダプタの挿入	2-6
データ・パターン・チェッカの作成	2-6
新規 Qsys システムの作成およびクロック・ソースの設定	2-7
パイプライン・ブリッジの追加	2-7
1:2 ストリーミング・デマルチプレクサの追加	2-8
カスタム・パターン・チェッカの追加	2-8
PRBS パターン・チェッカの追加	2-9
メモリ・アドレス・マップの検証	2-9
リセット信号の接続	2-9

第3章 階層システムの組み立て

階層メモリ・テストの作成	3-1
データ・パターン・ジェネレータの追加	3-4
パターン・チェッカの追加	3-4
メモリ・マスタ・コンポーネントの追加	3-5
リセット信号の接続	3-7
メモリ・アドレス・マップの指定	3-7
トップレベル・システムの完成	3-8
開発ボードにソフトウェアをコンパイルおよびダウンロード	3-10

第4章．システム・コンソールにおけるハードウェアの検証

スクリプトについて	4-1
チュートリアル・プロジェクトを開く	4-1
JTAG-Avalon マスタ・ブリッジの追加	4-2
開発ボードにシステム・コンソールをコンパイルおよび使用	4-3

第5章．カスタム・コンポーネントのシミュレーション

Qsys におけるテストベンチ・システムの生成	5-1
チュートリアル・プロジェクトを開く	5-2
テスト対象のデザインに対して新規の Qsys システムを作成	5-2
テスト対象のデザインのすべてのインタフェースをエクスポート	5-2
Qsys テストベンチ・システムの生成	5-2
Qsys テストベンチ・システムのシミュレーション・モデルの生成	5-3
ModelSim-Altera ソフトウェアにおけるシミュレーションの実行	5-4
シミュレーション環境のセットアップ	5-5
シミュレーションの実行	5-5

追加情報


改訂履歴	Info-1
アルテラへのお問い合わせ	Info-1
表記規則	Info-1

このチュートリアルでは、Quartus® II ソフトウェアに付属の Qsys システム統合ツールについて紹介します。このチュートリアルでは、様々なテスト・パターンを使用して外部メモリ・デバイスをテストするシステムの設計方法を示します。また、このチュートリアルではシステムの設計を中心にして、システム要件の解析、ハードウェア・デザインの作業、およびシステム・パフォーマンスの評価について解説しています。完了時には、Qsys の開発フローを理解し、独自のシステムを設計することができます。

ソフトウェア要件およびハードウェア要件

このチュートリアルでは、以下のソフトウェアが要求されます。

- Altera® Quartus II ソフトウェア v11.0 以降

 システム要件およびインストール手順については、[「Altera Software Installation and Licensing」](#) を参照してください。

- Nios® II EDS v11.0 以降
- **Qsys Tutorial Design Example** ウェブページから入手できる **tt_qsys_design.zip** デザイン例ファイル。
このデザイン例ファイルには、アルテラ開発ボード別で複数のプロジェクト・ファイルが用意されています。

このチュートリアルで説明したシステムは、以下のボード要件を満たすユーザ独自のカスタム・ボードに構築することができます。

- このボードは、アルテラの Arria®、Cyclone®、または Stratix® シリーズの FPGA を持つ必要があります。
- この FPGA は、最低 12K のロジック・エレメント (LE) またはアダプティブ・ルック・アップ・テーブル (ALUT) を持つ必要があります。
- この FPGA は、最低 150K ビットのエンベデッド・メモリを持つ必要があります。
- このボードは、FPGA への JTAG 接続を持つ必要があります。JTAG 接続は、メモリ・テストの進行状況をモニタするために、ホストに接続する通信を提供します。
- このボードは、デザインがテストするメモリを持つ必要があります。例えば、Avalon® Memory-Mapped (Avalon-MM) スレーブ・インタフェース付きの Qsys ベース・コントローラを有する任意のメモリです。

デザイン例ファイルに設定済みのものを除くほかの開発ボードを使用する場合、そのボードのクロック周波数およびピンアウト説明に関する資料を参照してください。アルテラ開発ボードの場合、この情報は該当するリファレンス・マニュアルから入手できます。また、ほかの様々な開発ボードに移植するための手順を含むハードウェア・プロジェクトが用意されています。

概要

このチュートリアルで作成した Qsys システムは、SDRAM をテストします。最終的なシステムには SDRAM コントローラが含まれており、1つの Nios II プロセッサおよび複数のエンベデッド・ペリフェラルが階層的なサブシステムとしてインスタンス化されます。テスト・データの生成、メモリのアクセス、および返されるデータの検証を司る様々な Qsys コンポーネントを追加することで、Qsys システムが完成されます。

最終的なシステムには、下記のコンポーネントが含まれています。

- Nios II/e コアに基づくプロセッサ・サブシステム (Altera Complete Design Suite に付属)
- SDRAM コントローラ (Altera Complete Design Suite に付属)
- 擬似ランダム・バイナリ・シーケンス (PRBS) パターン・ジェネレータおよびチェッカ
- カスタム・パターン・ジェネレータおよびチェッカ
- パターン選択マルチプレクサおよびデマルチプレクサ
- パターン・ライターおよびリーダー
- メモリ・テスト・コントローラ

この最終的なシステムは、ライセンスなしで実機上で使用できます。アルテラの無償 OpenCore Plus 評価機能により、以下の処理を実行することができます。

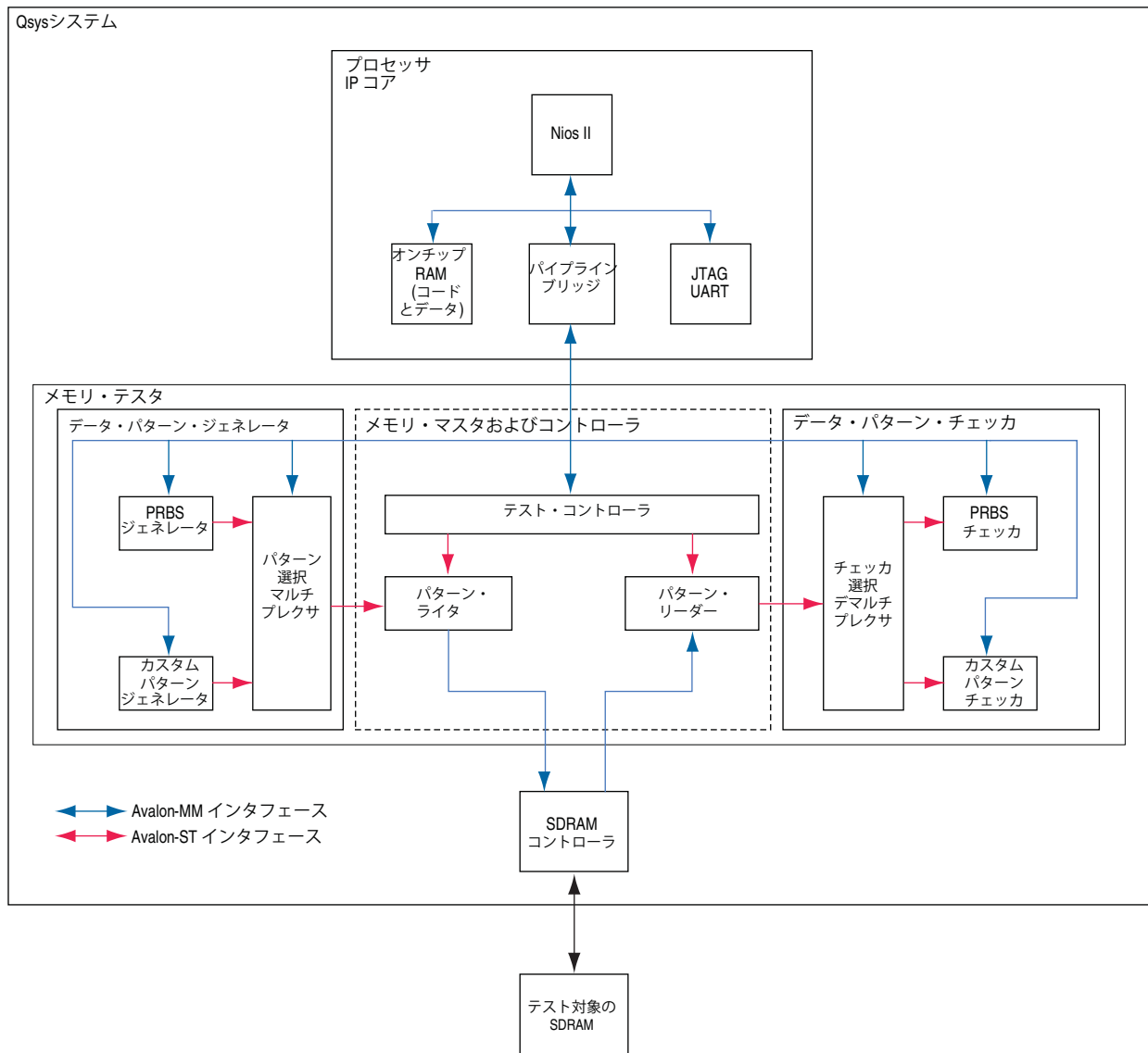
- システム動作のシミュレーションおよび機能の検証を実行できる
- デザインに対して、時間制限付きのデバイス・プログラミング・ファイルを生成する。
- デバイスをプログラムし、デザインを実機上で検証する。

これらのデザイン例ファイルは、任意のデザインで無償で使用できるコンポーネントから構成されています。Nios II 'e' プロセッサ・コアおよび DDR SDRAM IP コアは、Quartus II サブスクリプション・ライセンスを持つ場合は無料で使用できます。開発キット上のメモリ・デバイスに一致させるために、別々の開発キットのデザイン・ファイルには、異なる DDR SDRAM コントローラが使用されます。

- OpenCore Plus について詳しくは、「[AN320: OpenCore Plus Evaluation of Megafunctions](#)」を参照してください。

図 1-1 に、デザイン例の完全なトップレベル・システムを示します。図 1-1 に、メモリ・テスト・システム内のコンポーネントは、点線によってグループされた 3 つの主要なデザイン機能を持つ Qsys システムとして表示されます。このチュートリアルでは、階層的な構造が使用されています。データ・パターン・ジェネレータおよびデータ・パターン・チェッカは個別のシステムにインスタンス化され、その後にメモリ・テスト・システムに導入されます。階層的な構造により、システムをより上位レベルのシステム内のコンポーネントとしてインスタンス化することができます。

図 1-1. トップレベル・システムの構造



デザイン例ファイルのダウンロードおよびインストール

このチュートリアル用のデザイン例ファイルをダウンロードおよびインストールするには、以下のステップに従います。

1. **Qsys Tutorial Design Example** ウェブページから **Qsys Tutorial Design Example (.zip)** ファイルをダウンロードします。
2. ZIP ファイルの内容をコンピュータ上の任意のディレクトリに解凍します。ディレクトリ・パス名にスペースを使用しないでください。

チュートリアル・プロジェクトを開く

このチュートリアルのデザイン例ファイルには、開始点として使用するのに必要なカスタム IP デザイン・ブロックとプロジェクト・ファイル、および部分的に完成している Quartus II プロジェクトと Qsys システムが含まれています。デザイン例ファイルには、下記のプロジェクトが含まれています。

- 割り当てられた Quartus II プロジェクト I/O ピン・アサインメントおよび指定された .sdc (Synopsys Design Constraints) タイミング・アサインメント
- 開発されたメモリ・テスト・システムを制御するホスト PC と通信するための、パラメータ化された Nios II プロセッサ・コア
- 開発ボード上のメモリを使用するための、パラメータ化された DDR SDRAM

チュートリアル・プロジェクトを開くには、下記のステップに従います。


1. Quartus II ソフトウェアを起動します。
2. ご使用のボードに該当する Quartus II プロジェクト・ファイル (.qpf) ファイルを開きます。
 - a. File メニューで、**Open Project** をクリックします。
 - b. `tt_qsys_design\quartus_ii_projects_for_boards\<development_board>\` ディレクトリに移動します。
 - c. ボード固有の .qpf ファイルを選択して、**Open** をクリックします。

デザインのカスタム・メモリ・テスト・コンポーネントは Verilog HDL コンポーネントであり、付属のハードウェア・コンポーネント記述ファイル (_hw.tcl) が各コンポーネントのインタフェースおよびパラメータ設定を記述します。これらのファイルは `tt_qsys_design\memory_tester_ip` ディレクトリにあります。Qsys でこれらのコンポーネントを表示するには、**Component Library** タブで **Memory Test Microcores** を展開します。IP インデックス (.ipx) ファイルには、これらのメモリ・テスト・コンポーネントを含む `memory_tester_ip` ディレクトリへの参照パスが提供されています。

この章では、コンポーネントのインスタンス化、パラメータ化、および接続で Qsys システムを作成する方法について説明しています。

この章では、[ページ 3](#) の [図 1-1](#) に示す下記のデザイン・ブロックの Qsys システムを作成します。


- データ・パターン・ジェネレータ
- データ・パターン・チェッカ

 既に Qsys システムの作成を理解している場合、この章をスキップし、[3. 階層システムの組み立て](#)に進んでもかまいません。チュートリアル・デザイン・ファイルには、この章で説明した完全なシステムが含まれています。

データ・パターン・ジェネレータは、PRBS シーケンスまたはソフト・プログラマブル・シーケンスとして動作する高速ストリーミング・データを生成します (例：“walking ones”)。このデザインは、Avalon-ST (Avalon-Streaming) 接続でデータをメモリ・マスタおよびコントロール・ロジックのパターン・ライターに転送します。


データ・パターン・ジェネレータは、コントローラ・ロジックによって発行されるコマンドに基づいてデータをメモリに書き込みます。デザインがメモリにデータを書き込む際、パターン・リーダー・ロジックが内容をリードバックし、データ・パターン検証ロジックに転送します。

データ・パターン・チェッカは、Avalon-ST 接続でパターン・リーダーによってリードバックされるデータを受信します。デザインはデータ・パターンを検証し、メモリに書き込まれたパターンがリードバックされたデータと同じであることを確認します。

 システムでコンポーネントを追加および接続する作業が進んでいると、エラーおよび警告メッセージが Qsys の **Messages** タブに表示され、システムを完成させる前に実行しなければならないステップを示します。一部のエラー・メッセージはすぐに解決できず、後の段階で解決されます。

データ・パターン・ジェネレータの生成

この項では、データ・パターン・ジェネレータ・システムを生成します。データ・パターン・ジェネレータ・システムは、テスト・パターンを生成する 2 つのコンポーネント、およびデータをマルチプレックするプロセッサ制御のコンポーネントから構成されています。メモリ・インタフェースの幅に一致するようにデータ・パターン・ジェネレータをコンフィギュレーションすることができます。データ・ジェネレータがクロック・サイクルごとに 1 ワードのデータを提供できるため、コンポーネントをメモリ幅に一致するようにコンフィギュレーションすることで、メモリに迅速にアクセスするのに十分な帯域幅を提供できます。

 この Qsys システムを作成する前には、チュートリアル・ファイルをダウンロードとインストールし、Quartus II プロジェクトを開くようにしてください。(3 ページの「デザイン例ファイルのダウンロードおよびインストール」および 4 ページの「チュートリアル・プロジェクトを開く」)

クロック・ソース付きの新規 Qsys システムを作成

新規の Qsys システムを作成し、クロック・ソースを設定するには、下記のステップに従います。

1. Quartus II ソフトウェアの Tools メニューで、**Qsys** をクリックします。
2. Qsys の File メニューで、**New System** をクリックします。Qsys は新規の空のシステムを表示します。**System Contents** タブで、クロック・ソース・インスタンス、**clk_0** が表示されます。
3. クロック・ソース設定を開くには、**clk_0** を右クリックして **Edit** をクリックするか、またはそのインスタンスをダブル・クリックします。
4. **Clock frequency is known** をオフにします。これで、インスタンスが作成された後、このサブシステムをインスタンス化する上位レベルの階層システムがクロック周波数を提供します。
5. **Finish** をクリックします。
6. システムに名前を付けて、保存します。
 - a. File メニューで、**Save As** をクリックします。
 - b. ファイル名 `pattern_generator_system` を入力して、**Save** をクリックします。チュートリアルのスクリプトではここに記述した名前を使用しているため、システム名を正しく入力するように注意してください。

パイプライン・ブリッジの追加

このシステムを構成するコンポーネントには、いくつかの Avalon-MM スレーブ・インタフェースが含まれています。トップレベルのシステムが 1 つのスレーブ・インタフェースをリードおよびライトすることですべての Avalon-MM スレーブ・インタフェースに接続できるようにするには、すべてのスレーブ・インタフェースを Avalon-MM パイプライン・ブリッジに統合し、1 つの Avalon-MM スレーブ・インタフェースをこのシステムからエクスポートします。また、このブリッジは 1 レベルのパイプラインを追加し、タイミング性能を向上させます。パイプライン・ブリッジを追加するには、下記のステップに従います。

1. **Component Library** タブで、**Bridges and Adapters** を展開し、そして **Memory Mapped** を展開します。または、検索ボックスに `bridge` を入力し、リストをフィルタしてブリッジ・コンポーネントだけを表示させることができます。検索フィルタをクリアするには、検索ボックスの横にある **X** をクリックします。
2. **Avalon-MM Pipeline Bridge** をクリックして、**Add** をクリックします。あるいは、**Avalon-MM Pipeline Bridge** をダブル・クリックします。パラメータ・エディタが表示されます。
3. パラメータ・エディタで、このシステム内のメモリ・マップド・コンポーネントの範囲を調整するために、**Address width** 枠に 11 を入力します。
4. **Finish** をクリックします。デフォルトのブリッジは、**mm_bridge_0** のインスタンス名でシステムに追加されます。

5. **mm_bridge_0** クロック・ドメインを **clk_0** に設定します。
 - **mm_bridge_0** clk インタフェースの **Clock** カラムをクリックして、ドロップダウン・リストから **clk_0** を選択します。
 - あるいは、**Connections** カラムで接続を設定することができます。 **clk_0** clk 出力と **mm_bridge_0** clk 入力間の接続点をクリックします。
 - または、**mm_bridge_0** clk 入力を右クリックし、**mm_bridge_0.clk Connections** をポイントして、**clk_0.clk** を選択します。
6. **mm_bridge_0** s0 インタフェースを **slave** という名前でエクスポートします。**Export** カラムをクリックして、**slave** を入力します。

カスタム・パターン・ジェネレータの追加

カスタム・パターン・ジェネレータは、様々なテスト・パターンを生成するようにコンフィギュレーションできます。このコンポーネントには、パターン・データおよびパターン長がプログラムされています。パターンが終了したら、カスタム・パターン・ジェネレータがパターンの最初のエレメントに戻ります。このコンポーネントは、次のパターンを生成します。


- Walking ones
- Walking zeros
- Low frequency
- Alternating low frequency
- High frequency
- Alternating high frequency
- Synchronous PRBS

Synchronous PRBS パターンは最も長いパターンであり、パターンを繰り返す前のエレメントは 256 個あります。Walking ones と Walking zeros のパターン長は、メモリの幅によって決まります。例えば、32 ビットのメモリをテストするとき、Walking ones または Walking zeros のパターン長は、パターンを繰り返す前に 32 です。High frequency と Low frequency のパターンの場合、パターンを繰り返す前のエレメントは 2 個しかありません。

このカスタム・パターン・ジェネレータは 3 つのインタフェースを持っており、その 2 つが生成されたパターンを制御します。このインタフェースは生成されたカスタム・パターンの動作を制御します。パターン・ライター・コアに送信したカスタム・パターンのエレメントをプログラムするために、プロセッサが書き込み専用の **pattern_access** インタフェースにアクセスします。**st_pattern_output** は、パターン・ライター・コアにデータを送信するストリーミング・ソース・インタフェースです。カスタム・パターン・ジェネレータを追加するには、次のステップに従います。

1. **Component Library** タブで、**Project** の下の **Memory Test Microcores** を展開し、**Custom Pattern Generator** をダブル・クリックします。パラメータ・エディタが表示されます。
2. パラメータ・エディタで **Finish** をクリックして、デフォルトのパラメータを受け入れます。

3. **System Contents** タブで、インスタンスの名前を `custom_pattern_generator` に変更します。
 - a. **Name** カラムで、`custom_pattern_generator_0` を右クリックし、**Rename** を選択します。
 - b. 名前から `_0` キャラクタを削除します。
4. `custom_pattern_generator` クロック・ドメインを `clk_0` に設定します。
5. `custom_pattern_generator` csr インタフェースを `mm_bridge_0` m0 インタフェースに接続させます。
 - **Connections** カラムで、`custom_pattern_generator` csr インタフェースと `mm_bridge_0` m0 インタフェース間の接続点をクリックします。
 - あるいは、`customer_pattern_generator` csr インタフェースを右クリックし、**customer_pattern_generator.csr Connections** をポイントし、`mm_bridge_0.m0` を選択します。
6. `custom_pattern_generator` `pattern_access` インタフェースを `mm_bridge_0` m0 インタフェースに接続させます。
7. `custom_pattern_generator` csr インタフェースを `0400` のベース・アドレスに割り当てます。
 - a. **Base** カラムで、アドレス `0x00000000` をダブル・クリックします。
 - b. ベース・アドレスに対して `400` (16 進フォーマット) を入力します。

 `pattern_access` インタフェースのアドレス・スペースとの衝突を回避するためには、ベース・アドレスがエンド・アドレスを少し上回る値に設定されます。
8. `custom_pattern_generator` `pattern_access` インタフェースのベース・アドレスは、`0000` に維持します。

PRBS パターン・ジェネレータの追加

PRBS パターン・ジェネレータの出力は統計的に定義された PRBS パターンです。パラメータ・エディタで、パターンを繰り返す前のパターン長を指定することができます。パターン長は、「 $2^{\text{データ幅}} - 1$ 」に定義されています。例えば、32 ビットの PRBS パターン・ジェネレータは、4,294,967,295 個の要素を送信した後にはパターンを繰り返します。ボード上のメモリの (ローカル) データ幅に基づいて PRBS ジェネレータの幅を設定してください。

PRBS パターン・ジェネレータには 2 つのインタフェースがあります。csr インタフェースは、生成された PRBS パターンの動作を制御します。`st_pattern_output` ストリーミング・ソース・インタフェースは、パターン・ライター・コンポーネントにデータを送信します。PRBS パターン・ジェネレータを追加するには、下記のステップに従います。

1. **Memory Test Microcores** グループから **PRBS Pattern Generator** をダブル・クリックします。
2. パラメータ・エディタで **Finish** をクリックして、デフォルトのパラメータを受け入れます。
3. インスタンスの名前を `prbs_pattern_generator` に変更します。

4. **prbs_pattern_generator** クロック・ドメインを **clk_0** に設定します。
5. **prbs_pattern_generator** csr インタフェースを **mm_bridge_0** m0 インタフェースに接続させます。
6. **prbs_pattern_generator** csr インタフェースを **0x0420** のベース・アドレスに割り当てます。(custom_pattern_generator csr インタフェースのベース・アドレスの **0x0400** を少々上回る)

2:1 ストリーミング・マルチプレクサの追加

システムには2つのパターン・ソースがありますが、パターン・ライター・コンポーネントは1つのストリーミング・ソースからのデータしか受信できません。そのため、パターン・ジェネレータおよびパターン・ライターの間に2:1 ストリーミング・マルチプレクサが追加されています。ストリーミング・ソフト・プログラマブルな2:1 マルチプレクサ IP コアにより、プロセッサはどのパターンがパターン・ライター・コンポーネントに送信されるかを選択できます。このコンポーネントには、次のインタフェースが含まれています。

- 2つのストリーミング入力 — st_input_A および st_input_B
- 1つのストリーミング出力 — st_output
- 1つの csr スレーブ・インタフェース — プロセッサは、入力Aまたは入力Bがストリーミング出力に送信されるかを制御します。

カスタム・パターン・ジェネレータはA入力に接続され、PRBSパターン・ジェネレータはB入力に接続されます。2:1 ストリーミング・マルチプレクサを追加するには、下記のステップに従います。

1. **Memory Test Microcores** グループから **Two-to-one Streaming Mux** をダブル・クリックします。パラメータ・エディタが表示されます。
2. パラメータ・エディタで **Finish** をクリックして、デフォルトのパラメータを受け入れます。
3. インスタンスの名前を two_to_one_st_mux に変更します。
4. **two_to_one_st_mux** クロック・ドメインを **clk_0** に設定します。
5. **two_to_one_st_mux** st_input_A インタフェースを **custom_pattern_generator** st_pattern_output インタフェースに接続させます。
6. **two_to_one_st_mux** st_input_B インタフェースを **prbs_pattern_generator** st_pattern_output インタフェースに接続させます。
7. **two_to_one_st_mux** csr インタフェースを **mm_bridge_0** m0 インタフェースに接続させます。
8. **two_to_one_st_mux** st_output インタフェースを st_data_out という名前でエクスポートします。
9. **two_to_one_st_mux** csr インタフェースを **0x0440** のベース・アドレスに割り当てます。(prbs_pattern_generator csr インタフェースのベース・アドレスの **0x0420** を少々上回る)

メモリ・アドレス・マップの検証

作成されたシステムのメモリ・マップがチュートリアルのほかのセクションに使用されるメモリ・マップに一致することを確保するために、システムのベース・アドレスを検証します。**Address Map** タブをクリックして、表内のエントリが表 2-1 の値に一致することを確認します。赤い感嘆符は、アドレス範囲がオーバーラップしていることを示します。アドレスがオーバーラップしていない、そしてマップがこのチュートリアルのガイドラインに一致するようにアドレス・マップを訂正します。

表 2-1. アドレス・マップ

コンポーネント	アドレス
custom_pattern_generator.csr	0x00000400 – 0x0000040f
custom_pattern_generator.pattern_access	0x00000000 – 0x0000003f
prbs_pattern_generator.csr	0x00000420 – 0x0000043f
two_to_one_st_mux.csr	0x00000440 – 0x00000447

リセット信号の接続およびアダプタの挿入

Messages タブ内の一部のエラー・メッセージを排除するには、すべてのリセット信号を接続させる必要があります。**Qsys** では、システムに複数のリセット・ドメインまたは 1 つのリセット信号を使用できます。このデザインで **Qsys** の自動接続機能を使用するには、すべてのリセット信号を入力リセット信号に接続する必要があります。すべてのリセット信号を接続させるには、**System** メニューで、**Create Global Reset Network** を選択します。

Messages タブ内の残りのエラー・メッセージは、パターン・ジェネレータおよびマルチプレクサ間のレディ・レイテンシの不一致に関連します。ストリーミング・ソースおよびシンク・タイミング特性間の不一致を排除するには、**System** メニューで **Insert Avalon-ST Adapters** を選択します。これで、**Qsys** は適切なデータパスに自動的にストリーミング・タイミング・アダプタを挿入します。

Qsys には、エラー・メッセージが残っていません。**Messages** タブにエラー・メッセージが残っている場合、このシステムの作成手順を見直し、すべてのステップを再確認してください。リセット接続およびタイミング・アダプタは、**System Contents** タブで確認できます。

次に、システムを保存します。**File** メニューで、**Save** をクリックします。

これで、データ・パターン・ジェネレータを含むシステムが完成しました (ページ 3 の図 1-1)。2:1 ストリーミング・マルチプレクサは、カスタム・パターン・ジェネレータまたは PRBS パターン・ジェネレータのいずれかからのパターン・データをフル・システム内のパターン・ライターに転送します。2:1 ストリーミング・マルチプレクサからのデータは、クロック・サイクルごとに 1 ワードのスループットを達成します。

データ・パターン・チェッカの作成

このセクションでは、データ・パターン・チェッカのシステムを作成します。このシステムは、データ・パターン・ジェネレータに非常に似ています。このシステムは、**SDRAM** からのパターンをリードバックし、パターン・チェッカに転送して、データ・パターン・ジェネレータからのパターンに対して検証します。パターン・リーダーはデータを 1:2 ストリーミング・デマルチプレクサに送信し、そしてデマ

ルチプレクサはデータをカスタム・パターン・チェッカまたは PRBS パターン・チェッカのいずれかに転送します。1:2 ストリーミング・デマルチプレクサはソフト・プログラマブルであり、プロセッサはどのパターン・チェッカ IP コアがパターン・レーダーに読み出されるデータを検証するかを選択できます。また、カスタム・パターン・チェッカもソフト・プログラマブルであり、カスタム・パターン・ジェネレータと同様なパターンに一致するようにコンフィギュレーションされます。

新規 Qsys システムの作成およびクロック・ソースの設定

新規の Qsys システムを作成し、クロック・ソースを設定するには、下記のステップを実行します。

1. **File** メニューで、**New System** をクリックします。Qsys は新規の空のシステムを表示します。**System Contents** タブで、クロック・ソース・インスタンス、**clk_0** が表示されます。
2. インスタンスをダブル・クリックして、クロック・ソースの設定を編集します。
3. **Clock frequency is known** をオフにします。これで、インスタンスが作成された後、このサブシステムをインスタンス化する上位レベルの階層システムがクロック周波数を提供します。
4. **Finish** をクリックします。
5. パターン・チェッカ・システムを保存します。
 - a. **File** メニューで、**Save As** をクリックします。
 - b. ファイル名 `pattern_checker_system` を入力して、**Save** をクリックします。

パイプライン・ブリッジの追加

スレーブ・インタフェースを統合するパイプライン・ブリッジを追加するには、これらのステップに従います。

1. **Component Library** タブで、**Bridges and Adapters** を展開し、**Memory Mapped** を展開します。
2. **Avalon-MM Pipeline Bridge** コンポーネントをクリックし、**Add** をクリックします。パラメータ・エディタが表示されます。
3. パラメータ・エディタで、このシステム内のメモリ・マップド・コンポーネントの範囲を調整するために、**Address width** 枠に 11 を入力します。
4. **Finish** をクリックします。デフォルトのインスタンス名は **mm_bridge_0** です。
5. **mm_bridge_0** クロック・ドメインを **clk_0** に設定します。
6. `slave` という名前で **mm_bridge_0** s0 インタフェースをエクスポートします。

1:2 ストリーミング・デマルチプレクサの追加

1:2 ストリーミング・デマルチプレクサは、2:1 ストリーミング・マルチプレクサと反対の動作を実行しています。1:2 ストリーミング・デマルチプレクサは、パターン・リーダーからデータを受信する 1 つのストリーミング・デマルチプレクサ (st_input)、およびカスタム・パターン・ジェネレータと PRBS パターン・ジェネレータを接続させるストリーミング出力インタフェース (st_output_A と st_output_B) を備えています。データがコンポーネントを経由するルートのプロセッサによってプログラムするために、システムにはスレーブ・インタフェースの csr が含まれています。1:2 ストリーミング・デマルチプレクサを追加するには、次のステップに従います。

1. **Memory Test Microcores** グループから **One-to-two Streaming Demux** をダブル・クリックします。
2. パラメータ・エディタで **Finish** をクリックして、デフォルトのパラメータを受け入れます。
3. **System Contents** タブで、インスタンスの名前を one_to_two_st_demux に変更します。
4. **one_to_two_st_demux** クロック・ドメインを **clk_0** に設定します。
5. st_data_in という名前前で **one_to_two_st_demux** st_input インタフェースをエクスポートします。
6. **one_to_two_st_demux** csr インタフェースを **mm_bridge_0** m0 インタフェースに接続させます。
7. **one_to_two_st_demux** csr インタフェースを 0x0400 のベース・アドレスに割り当てます。

カスタム・パターン・チェッカの追加

カスタム・パターン・チェッカは、カスタム・パターン・ジェネレータと反対の動作を実行しています。カスタム・パターン・チェッカは、1:2 ストリーミング・デマルチプレクサからデータを受信する 1 つのストリーミング入力インタフェース (st_pattern_input) を備えています。プロセッサがコンポーネントの制御に使用させる Avalon-MM スレーブ・インタフェース (csr) を備えています。また、プロセッサがカスタム・パターン・ジェネレータ・コンポーネントと同様なパターンをプログラムするためのメモリ・マップド・スレーブ・インタフェース (pattern_access) も備えています。カスタム・パターン・チェッカを追加するには、次のステップに従います。

1. **Memory Test Microcores** グループから、**Custom Pattern Checker** をダブル・クリックします。
2. パラメータ・エディタで **Finish** をクリックして、デフォルトのパラメータを受け入れます。
3. インスタンスの名前を custom_pattern_checker に変更します。
4. **custom_pattern_checker** クロック・ドメインを **clk_0** に設定します。
5. **custom_pattern_checker** の csr インタフェースおよび pattern_access インタフェースを **mm_bridge_0** m0 インタフェースに接続させます。
6. **custom_pattern_checker** st_pattern_input インタフェースを **one_to_two_st_demux** st_output_A インタフェースに接続させます。

7. **custom_pattern_checker** csr インタフェースを 0x0420 のベース・アドレスに割り当てます。
8. **custom_pattern_checker** pattern_access インタフェースを 0x0000 のベース・アドレスに割り当てます。

PRBS パターン・チェッカの追加

PRBS パターン・チェッカは、PRBS パターン・ジェネレータと反対の動作を実行しています。PRBS パターン・チェッカは、プロセッサがコンポーネントを制御するメモリ・マップド・スレーブ・インタフェース (csr) を備えています。また、1:2 ストリーミング・デマルチプレクサからデータを受信する st_pattern_input ストリーミング入力 (st_pattern_input) を備えています。PRBS パターン・チェッカを追加するには、次のステップに従います。

1. **Memory Test Microcores** グループから **PRBS Pattern Checker** をダブル・クリックします。パラメータ・エディタが表示されます。
2. パラメータ・エディタで **Finish** をクリックして、デフォルトのパラメータを受け入れます。
3. インスタンスの名前を prbs_pattern_checker に変更します。
4. **prbs_pattern_checker** クロック・ドメインを **clk_0** に設定します。
5. **prbs_pattern_checker** csr インタフェースを **mm_bridge_0** m0 インタフェースに接続させます。
6. **prbs_pattern_checker** st_pattern_input インタフェースを **one_to_two_st_demux** st_output_B インタフェースに接続させます。
7. **prbs_pattern_checker** csr インタフェースを 0x0440 のベース・アドレスに割り当てます。

メモリ・アドレス・マップの検証

作成されたシステムのメモリ・マップがチュートリアルほかのセクションに使用されるメモリ・マップに一致することを確保するために、システム内のベース・システムを検証します。**Address Map** タブをクリックして、表内のエントリが表 2-2 の値に一致することを確認します。

表 2-2. アドレス・マップ

コンポーネント	アドレス
one_to_two_st_demux.csr	0x00000400 - 0x00000407
custom_pattern_checker.csr	0x00000420 - 0x0000042f
custom_pattern_checker.pattern_access	0x00000000 - 0x0000003f
prbs_pattern_checker.csr	0x00000440 - 0x0000045f

リセット信号の接続

すべてのリセット信号を接続しなければなりません。すべてリセット信号を接続させるには、**System** メニューで、**Create Global Reset Network** を選択します。

Qsys には、エラー・メッセージが残っていません。**Messages** タブにエラー・メッセージが残っている場合、このシステムの作成手順を見直し、すべてのステップを再確認してください。リセット接続およびタイミング・アダプタは、**System Contents** タブで確認できます。

システムを保存します。**File** メニューで、**Save** をクリックします。

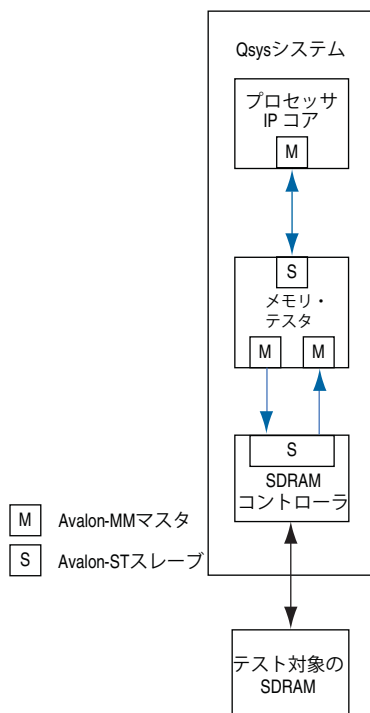
これで、メモリからリードバックされるデータの検証用のシステムが完成しました。トップレベル・システムは、パターン・リーダー・コンポーネントからのデータをストリーミング・インタフェースを介して送信します。1:2 ストリーミング・デマルチプレクサに入ったデータは、後ほどカスタム・パターン・チェッカまたは PRBS パターン・チェッカのいずれかに転送されます。

メモリ・テスト・デザインの下位レベル・システムが完成しました。これで、**3. 階層システムの組み立て**に進んで、これらのシステムを階層的なシステム・デザインに使用することができます。

このチュートリアルでは、Qsysによる階層システム・デザインについて説明します。階層システムにより、再利用可能なモジュラー・システム・コンポーネントの作成が可能になり、大規模なシステムをより小さいサブシステムに分解することで、大規模システムの構造を容易に把握できます。

このチュートリアルでは、2. Qsys システムの作成からのシステム（またはデザイン・ファイルに含まれる完全なシステム）をメモリ・テスト・システム内のサブシステムとして使用します。その後、プロセッサ・システムおよび SDRAM コントローラを含めて、トップレベル・システムでメモリ・テスト・システムをインスタンス化します。図 3-1 に、トップレベル・システムの上位レベルのインタフェースを示します。

図 3-1. システムの上位レベルのインタフェース



階層メモリ・テストの作成

図 3-2 に、メモリ・テスト・インタフェースを示します。

図 3-2. 階層的なメモリ・テスト・インタフェース

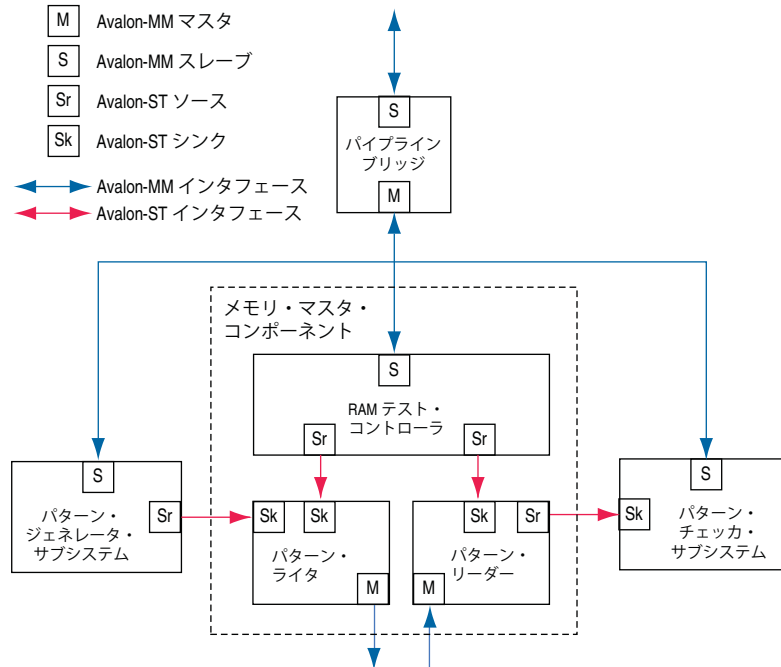


図 3-2 に示すように、メモリ・テストは、2. Qsys システムの作成で作成された次の Qsys サブシステムを含みます。

- データ・パターン・ジェネレータ —Avalon-ST データを生成して、メモリ・マスタ・コンポーネントに転送します。
- データ・パターン・チェッカ —メモリ・マスタ・コンポーネントからの Avalon-ST データを受信し、検証を行います。

第 2 章をスキップした場合、次のステップを実行して Quartus II プロジェクトをセットアップしてください。

1. チュートリアル・ファイルをダウンロードして、インストールします (3 ページの「デザイン例ファイルのダウンロードおよびインストール」を参照)。
2. `tt_qsys_design\completed_subsystems` ディレクトリからの 2 つの完全なシステム (`pattern_checker_system.qsys` と `patter_generator_system.qsys`) をご使用のボードに該当する `tt_qsys_design\quartus_ii_projects_for_boards\ ディレクトリにコピーします。`
3. 開発ボードに該当する Quartus II プロジェクトを開きます (4 ページの「チュートリアル・プロジェクトを開く」を参照)。
4. Quartus II ソフトウェアの Tools メニューで、**Qsys** をクリックします。

メモリ・テストを作成するには、次のステップに従います。

1. Qsys の **File** メニューで、**New System** をクリックします。Qsys は、新規の空のシステムを表示します。**System Contents** タブで、クロック・ソース・インスタンス、**clk_0** が表示されます。
2. クロック・ソース設定を開くには、**clk_0** を右クリックして **Edit** をクリックするか、またはそのインスタンスをダブル・クリックします。
3. **Clock frequency is known** をオフにします。これで、インスタンスが作成されたら、このサブシステムをインスタンス化する上位レベルの階層システムがクロック周波数を提供します。
4. **Finish** をクリックします。
5. システムを保存します。
 - a. **File** メニューで、**Save As** をクリックします。
 - b. ファイル名 `memory_tester_system` を入力して、**Save** をクリックします。

メモリ・テストには、いくつかの Avalon-MM スレーブ・インタフェースが含まれています。ただし、メモリ・テストでは、それらのインタフェースがトップレベル・システムに 1 つの Avalon-MM スレーブ・インタフェースをエクスポートする Avalon-MM パイプライン・ブリッジに纏められます。この手法では、トップレベルのシステムが 1 つのスレーブ・インタフェースをリードおよびライトすることですべての Avalon-MM スレーブ・インタフェースに接続できます。また、このブリッジは 1 レベルのパイプラインを追加し、タイミング性能を向上させます。パイプライン・ブリッジを追加するには、下記のステップに従います。

1. **Component Library** タブで、**Bridges and Adapters** を展開し、そして **Memory Mapped** を展開します。または、検索ボックスに `bridge` を入力し、リストをフィルタしてブリッジ・コンポーネントだけを表示させることができます。検索フィルタをクリアするには、検索ボックスの横にある **X** をクリックします。
3. **Avalon-MM Pipeline Bridge** をクリックして、**Add** をクリックします。あるいは、**Avalon-MM Pipeline Bridge** をダブル・クリックします。パラメータ・エディタが表示されます。
4. パラメータ・エディタで、このシステム内のメモリ・マップド・コンポーネントの範囲を調整するために、**Address width** 枠に 13 を入力します。
5. **Finish** をクリックします。デフォルトのインスタンス名は `mm_bridge_0` となります。
6. **System Contents** タブで、`mm_bridge_0` クロック・ドメインを `clk_0` に設定します。
 - `mm_bridge_0` clk インタフェースの **Clock** カラムをクリックして、ドロップダウン・リストから `clk_0` を選択します。
 - あるいは、**Connections** カラムで接続を設定することができます。`clk_0` clk 出力と `mm_bridge_0` clk 入力間の接続点をクリックします。
 - または、`mm_bridge_0` clk 入力を右クリックし、`mm_bridge_0.clk` **Connections** をポイントして、`clk_0.clk` を選択します。
7. `mm_bridge_0` s0 インタフェースを `slave` という名前でエクスポートします。**Export** カラムをクリックして、`slave` を入力します。


データ・パターン・ジェネレータの追加

2. Qsys システムの作成からのデータ・パターン・ジェネレータ・システムは、Avalon-ST ソース・インタフェースを介して、パターン・データのストリームを提供します。サブシステム内の各コンポーネントに割り当てられるメモリ位置をアクセスすることによって、システムを制御できます。このシステムには、次のコンポーネントが含まれています。

- パイプライン・ブリッジ
- カスタム・パターン・ジェネレータ
- PRBS パターン・ジェネレータ
- 2:1 ストリーング・マルチプレクサ
- ストリーミング・タイミング・アダプタ

メモリ・テストにデータ・パターン・ジェネレータを追加するには、次のステップに従います。

1. **Component Library** タブで、**Project** の下の **System** を展開し、**pattern_generator_system** をダブル・クリックします。パラメータ・エディタが表示されます。
2. **Finish** をクリックします。
3. インスタンスの名前を `pattern_generator_subsystem` に変更します。
 - a. **Name** カラムで、**system_0** を右クリックし、**Rename** を選択します。
 - b. インスタンス名に `pattern_generator_subsystem` を入力します。
4. **pattern_generator_subsystem** クロック・ドメインを **clk_0** に入力します。
5. **pattern_generator_subsystem** slave を **mm_bridge_0** m0 に接続させます。
6. **pattern_generator_subsystem** reset インタフェースを **clk_0** `clk_reset` インタフェースに接続させます。

 reset インタフェースが `reset_0` にエクスポートされるため、インタフェースを右クリックして接続を設定することはできません。接続は、**Connections** カラムで実行しなければなりません。**pattern_generator_subsystem** reset インタフェースを `clk_0` `clk_reset` インタフェース間の接続点をクリックしてください。

パターン・チェッカの追加

2. Qsys システムの作成からのパターン・チェッカ・システムは、Avalon-ST シンク・インタフェースを介して受信されるデータを検証します。サブシステム内の各コンポーネントに割り当てられるメモリ位置をアクセスすることによって、システムを制御できます。このシステムは、すべてのスレーブ・ポートをパイプライン・ブリッジに接続し、外部のコンポーネントに接続できるようにします。このシステムには、次のコンポーネントが含まれています。

- パイプライン・ブリッジ
- カスタム・パターン・チェッカ
- PRBS パターン・チェッカ
- 1:2 デマルチプレクサ

メモリ・テストにデータ・パターン・チェッカを追加するには、次のステップに従います。

1. **System** グループから **pattern_checker_system** をダブル・クリックします。
2. **Finish** をクリックします。
3. インスタンス名を **pattern_checker_subsystem** に変更します。
4. **pattern_checker_subsystem** クロック・ドメインを **clk_0** に変更します。
5. **pattern_checker_subsystem** slave インタフェースを **mm_bridge_0** m0 インタフェースに接続させます。
6. **pattern_checker_subsystem** reset インタフェースを **clk_0** clk_reset インタフェースに接続させます。

メモリ・マスタ・コンポーネントの追加

この項では、メモリ・マスタおよびRAMテスト・コントローラを追加します。メモリ・マスタは、メモリにテスト・パターンを書き込んでパターンをリードバックして検証を実行することでSDRAMコントローラにアクセスします。RAMテスト・コントローラはプロセッサからコマンドを受信し、メモリ・マスタを制御します。コマンドごとに、開始アドレス、テスト長（バイト単位）、およびメモリ・ブロックのサイズ（バイト単位）などの情報が含まれています。RAMテスト・コントローラは、コマンドをより小さいブロック転送に分割し、ストリーミング接続を介してリード&ライト・マスタに独立して割り当てます。

パターン・リーダーまたはパターン・ライターがブロック転送を完了したとき、次のコマンドを送受信する準備ができたという信号をRAMテスト・コントローラに転送します。RAMテスト・コントロールは、ブロックのサイズのコマンドを独立して発行し、メモリ転送間のアイドル・サイクルを最低限に抑えます。また、RAMテスト・コントローラは、パターン・リーダーがテスト中のメモリ位置に対してパターン・ライターより優先していないことを保証します。そうしないと、データが破壊されます。

このデザイン内のSDRAMコントローラは、2のローカル最大バースト長を使用するようにパラメータ化されています。また、メモリ帯域幅を最大化するために、パターン・リーダーおよびパターン・ライターはバースト長に一致するようにコンフィギュレーションされます。

パターン・ライター・コンポーネントの追加

パターン・ライター・コンポーネントは、commandストリーミング・インタフェースによってRAMテスト・コントローラからメモリ転送コマンドを受信します。st_dataストリーミング・インタフェースは、デザインのパターン・ジェネレータに提供されるデータを受信します。mm_dataメモリ・マップド・インタフェースはパターン・データをSDRAMコントローラに書き込みます。システムにパターン・ライター・コンポーネントを追加するには、次の手順に従います。

1. **Memory Test Microcores** グループから、**Pattern Writer** コアをダブル・クリックします。パラメータ・エディタが表示されます。
2. **Burst Enable** サポートをオンにします。
3. **Maximum Burst Count** が **2** であることを確認します。
4. **Enable Burst Re-alignment** がオンであることを確認します。

5. パラメータ・エディタで **Finish** をクリックして、デフォルトのパラメータを受け入れます。
6. インスタンスの名前を `pattern_writer` に変更します。
7. **pattern_writer** クロック・ドメインを **clk_0** に設定します。
8. **pattern_writer** `st_data` インタフェースを **pattern_generator_subsystem** `st_data_out` インタフェースに接続させます。
9. `write_master` という名前で **pattern_writer** `mm_data` インタフェースをエクスポートします。

パターン・リーダー・コンポーネントの追加

パターン・リーダー・コンポーネントは、`command` ストリーミング・インタフェースによって RAM テスト・コントローラからメモリ転送コマンドを受信します。`mm_data` Avalon-MM インタフェースは SDRAM コントローラからパターン・データを読み出します。`st_data` Avalon-ST インタフェースは、メモリから読み出されるデータをデザインのパターン・チェッカに送信します。システムにパターン・リーダー・コンポーネントを追加するには、これらのステップに従います。

1. **Memory Test Microcores** グループから **Pattern Reader** コアをダブル・クリックします。パラメータ・エディタが表示されます。
2. **Burst Enable** サポートをオンにします。
3. **Maximum Burst Count** が **2** であることを確認します。
4. **Enable Burst Re-alignment** がオンであることを確認します。
5. パラメータ・エディタで **Finish** をクリックして、デフォルトのパラメータを受け入れます。
6. インスタンスの名前を `pattern_reader` に変更します。
7. **pattern_reader** クロック・ドメインを **clk_0** に設定します。
8. **pattern_reader** `st_data` インタフェースを **pattern_checker_subsystem** `st_data_in` インタフェースに接続させます。
9. `read_master` という名前で **pattern_reader** `mm_data` インタフェースをエクスポートします。

RAM テスト・コントローラの追加

RAM テスト・コントローラは、パターン・リーダー・コンポーネントおよびパターン・ライター・コンポーネントにコマンドを送信する 2 つのインタフェースを備えています。これらのストリーミング・インタフェースは、`write_command` および `read_command` です。Avalon-ST インタフェースは、低レイテンシかつシンプルなハンドシェイク・プロトコルを提供するため、これらのストリーミング・インタフェースは効率的にコマンドを発行します。また、プロセッサはコントローラにコマンドを書き込むためのスレーブ・ポート (`csr`) にアクセスしています。

システムに RAM テスト・コントローラを追加するには、次のステップに従います。

1. **Memory Test Microcores** グループから、**RAM Test Controller** をダブル・クリックします。パラメータ・エディタが表示されます。
2. パラメータ・エディタで **Finish** をクリックして、デフォルトのパラメータを受け入れます。

3. **System Contents** タブで、インスタンスの名前を `ram_test_controller` に変更します。
4. **ram_test_controller** クロック・ドメインを `clk_0` に設定します。
5. **ram_test_controller** `write_command` インタフェースを **pattern_writer** `command` インタフェースに接続させます。
6. **ram_test_controller** `read_command` インタフェースを **pattern_reader** `command` インタフェースに接続させます。
7. **ram_test_controller** `csr` インタフェースを **mm_bridge_0** `m0` インタフェースに接続させます。



すべてのサブシステムを含むトップレベル全体に対してファイルを生成する必要があるため、この時点では **Generation** タブを使用してこれらのサブシステムの HDL コードを生成してはいけません。デバイスをプログラムするのに提供されるバッチ・スクリプトは、プロジェクト・ディレクトリで1つのシステムのみが生成されることを要求します。トップ・レベル・デザインには Nios II サブシステムが含まれており、Nios II ソフトウェア・ビルド・ツールは **.sopcinfo** ファイルがトップレベルのデザインに対して生成されることを要求します。複数の **.sopcinfo** ファイルが存在する場合、デザインをプログラムするバッチ・スクリプトはソフトウェア・ビルド・ツールのエラーのため失敗します。

リセット信号の接続

すべてのリセット信号を接続する必要があります。System メニューで、**Create Global Reset Network** を選択します。

メモリ・アドレス・マップの指定

この項では、**Address Map** タブを使用して、このチュートリアルのほかの項に使用されるメモリ・マップに一致するようにシステムのメモリ・マップ内のアドレスを設定します。ベース・アドレスを設定するには、次のステップに従います。

1. **Address Map** タブをクリックします。現在、すべてのスレーブ・アドレス・マップがアドレス `0x0` から開始するため、赤い感嘆符はアドレスのオーバーラップを示します。
2. 各インタフェースの横にある **mm_bridge_0.m0** カラムをダブル・クリックし、アドレス範囲を編集します。現在のアドレスを削除し、表 3-1 を参照して正確なベース・アドレスを入力します。

表 3-1. メモリ・テストのアドレス・マップ

コンポーネント名	ベース・アドレス	結果として得られるアドレス範囲
<code>mm_bridge_0.s0</code>	N/A	N/A
<code>pattern_generator_subsystem.slave</code>	<code>0x0</code>	<code>0x00000000 – 0x000007ff</code>
<code>pattern_checker_subsystem.slave</code>	<code>0x1000</code>	<code>0x0001000 – 0x000017ff</code>
<code>ram_test_controller.scsr</code>	<code>0x800</code>	<code>0x00000800 – 0x0000081f</code>


3. セルの外をクリックすると、Qsys は結果として得られるアドレス範囲を表示します。テーブル内の結果アドレス範囲が表 3-1 に示す値に一致することを確認します。


これで、エラー・メッセージまたは警告メッセージは残っていません。メモリ・テスト・システムを保存します。

トップレベル・システムの完成

この項では、トップレベルを完成させる方法について説明します。メモリ・テスト・システムを追加してトップレベル・システムを完成させるには、次のステップに従います。

1. **tt_qsys_design\quartus_ii_projects_for_boards\<development_board>** ディレクトリからの **top_system.qsys** ファイルを Qsys で開きます。
2. このシステムは、ご使用の開発ボードに対してセットアップされ、1つの外部クロック・ソース、1つのプロセッサ・システムおよび1つの SDRAM コントローラを備えています。**Clock Settings** タブでこのトップレベル・システム内のクロックを表示することができます。**System Contents** タブで部分的に完成しているシステム・コンポーネントを表示することができます。
3. **System** グループから **memory_tester_system** をダブル・クリックします。
4. **Finish** をクリックします。メモリ・テスト・システムがトップレベル・システムに追加されています。
5. **System Contents** タブで、システムの名前を **memory_tester_subsystem** に変更します。
6. **System Contents** タブで、**memory_test_subsystem** を **cpu_subsystem** と **sdram** の間に移動させます。**cpu_subsystem** は **memory_tester_subsystem** を制御し、そして **memory_tester_subsystem** は **sdram** を制御するため、このステップにより、システムの把握が簡単になります。**memory_test_subsystem** を選択して、上向きの矢印を1回クリックします。
7. **memory_tester_subsystem** クロック・ドメインを次のように設定します。
 - ALTMEMPHY ベースのデザインの場合、**sdram_sysclk** に設定
 - UniPHY ベースのデザインの場合、**sdram_afi_clk** に設定

 一部のボードは、アルテラの ALTMEMPHY 付きの DDR または DDR2 SDRAM コントローラを使用する FPGA および SDRAM デバイスを備えています。その他のボードはアルテラの UniPHY 付きの DDR3 SDRAM コントローラを使用しています。
8. **memory_tester_subsystem** reset インタフェースを、**ext_clk** clk_reset インタフェースおよび **cpu_subsystem** cpu_jtag_debug_reset インタフェースに接続させます。


 このデザインは、`cpu_jtag_debug_reset` というインタフェース名で、Nios II プロセッサ JTAG デバッグ・リセット出力インタフェース (`jtag_debug_module_reset`) を `cpu_subsystem` からエクスポートします。デザインは、この Nios II リセット出力を、Nios II プロセッサ・コードまたは JTAG インタフェースによってリセットされる必要のあるコンポーネント・リセット入力、および Nios II プロセッサのリセット入力インタフェースに接続させる必要があります。`cpu_subsystem` `cpu_reset` インタフェースは Nios II プロセッサのリセット入力インタフェースに接続します。`top_level.qsys` ファイルは、`cpu_jtag_debug_reset` インタフェースを `cpu_reset` インタフェースに接続させます。

9. `memory_tester_subsystem` `write_master` および `read_master` インタフェースを次のように接続させます。

- ALTMEMPHY ベースのデザインの場合、`sdram s1` インタフェースに接続
- UniPHY ベースのデザインの場合、`sdram av1` インタフェースに接続

10. `memory_tester_subsystem` `slave` インタフェースを `cpu_subsystem` `master` インタフェースに接続させます。

11. ALTMEMPHY ベースのデザインの場合、`memory_tester_subsystem` `slave` インタフェースおよび `sdram s1` インタフェースを `0x0` に維持するようにします。UniPHY ベースのデザインの場合、`memory_tester_subsystem` `slave` インタフェースおよび `sdram av1` インタフェースを `0x0` に維持するようにします。

 2つのスレーブ・インタフェースは異なるマスタによって制御されるため、同じアドレスを使用することができます。`cpu_subsystem` `master` インタフェースはメモリ・テスト・サブシステムを制御し、`memory_tester_subsystem` `write_master` および `read_master` インタフェースは `sdram` インタフェースを制御します。

これで、デザインが完成しました。**Messages** タブにエラー・メッセージが残っている場合、このシステムの作成手順を見直し、すべてのステップを再確認してください。

システムを保存します。また、次のタブをクリックすることができます。

- **System Inspector** タブは階層システムを表示します。デフォルトのビューでは、トップレベル・システムのプロジェクト設定を表示します。**Submodules** を展開すると、下位レベルのシステムおよびコンポーネントの情報が表示されます。
- **HDL Example** タブは Qsys システムの入力信号と出力信号を表示します。このタブは、このシステムのインスタンス化の例の HDL を HDL ファイルで示し、システム内のエクスポートされるインタフェースからのすべての信号の一覧を示します。信号名は、エクスポートされたインタフェースの名前に続くアンダースコア、そしてコンポーネントまたは IP コアで指定される信号名の組合せとなります。この場合、ほとんどの信号はテスト対象の外部 SDRAM デバイスに接続しません。
- **Generation** タブでは、Qsys からデザイン・ファイルを生成することができます。また、生成完了後に Quartus II ソフトウェアでハードウェア・デザインをコンパイルし、ボードにイメージおよびソフトウェアをダウンロードすることができます (10 ページの「開発ボードにソフトウェアをコンパイルおよびダウンロード」を参照)。

開発ボードにソフトウェアをコンパイルおよびダウンロード

アルテラは、デザインを完成させ、ボードのインタフェースをテストするために、開発ボードにシステムおよびソフトウェアをダウンロードすることを推奨しています。開発ボードを持っていない場合、ほかの FPGA デバイスまたはボードにデザインを移植する方法については、付属の **readme.txt** ファイルで提供された手順を参照してください。

アルテラ提供のソフトウェアは、様々なテスト・パラメータおよびパターンを使用してメモリをテストします。このソフトウェアは、ボードにコンパイルおよびダウンロードするために書かれています。トップレベル・システムを開発ボードにダウンロードするには、次のステップに従います。

1. Qsys で、**Generation** タブをクリックします。
2. **Create HDL design files for synthesis** がオンであることを確認します。このステップは、ほかのオプションを必要としません。
3. **Generate** をクリックします。Qsys は、システムの HDL ファイル、および Quartus II コンパイル作業に必要な HDL ファイルのリストを提供する **.qip** (Quartus IP ファイル) ファイルを生成します。
4. 生成完了後に、**Close** をクリックします。Qsys を閉じてかまいません。
5. Quartus II ソフトウェアの Project メニューで、**Add/Remove Files in Project** をクリックして、プロジェクトがこのシステムの **.qip** ファイルを含むことを確認します。また、このプロジェクトには、**top_system.qsys** システムおよび SDC タイミング制約ファイル (**my_constraints.sdc**) をインスタンス化するトップレベルの Verilog HDL ラッパー・ファイル (**top_level.v**) が含まれています。
6. Processing メニューで、**Start Compilation** をクリックします。
7. サポートされるプログラミング・ケーブルに開発ボードを接続させます。
8. Linux または Nios II コマンド・シェルを起動します。Qsys の Tools メニューで、**Nios II Command Shell [gcc4]** をクリックします。また、Windows で Nios II コマンド・シェルを起動します。Start メニューで、**Altera** をポイントして、**Nios II EDS** をポイントし、そして **Nios II EDS Command Shell** をクリックします。Linux の場合、Linux ターミナルを起動します。
9. **quartus_ii_projects_for_boards\<development_board>\software** ディレクトリに移動します。
10. コンパイルの完了後に、**OK** をクリックします。
11. Nios II コマンド・シェルまたは Linux ターミナルで次のコマンドを入力します。


```
./batch_script.sh.
```

バッチ・スクリプトは Nios II ソフトウェアをコンパイルし、**.sof** プログラミング・ファイルを FPGA にダウンロードします。テスト・ソフトウェアは、次のパラメータを検証することで、SDRAM にスイープ・テストを実行します。

- パターンの種類
- メモリ・ブロックのサイズ
- メモリ・ブロック間の距離 (パターン・リーダーとパターン・ライター間のブロックの数)
- テストされたメモリの容量

 プロジェクト・ディレクトリに複数セットの生成されたシステム・ファイルがないように確認してください。複数の **.sopcinfo** ファイルが存在する場合、デザインをプログラムするバッチ・スクリプトはソフトウェア・ビルド・ツールのエラーのため失敗します。

メモリがテストされている間に、メモリのスループット値はコマンド・ターミナルに表示されます。これらの値は **16 進数** で表示され、**SDRAM アドレス・スパン全体** をテストするのに必要なクロック・サイクル数を表します。ターミナルにキャラクタを出力するためのソフトウェア・ライブラリが非常に小さいため、出力が **16 進数** に制限されます。メモリ・テスト・システムがメモリに書き込み、そしてリードバックをするため、メモリ・テスト・システムがアクセスしてトランスクリプト・ウィンドウで表示されるバイト数はメモリ・スパンの **2 倍** になります。この値は、メモリ・デバイスに対してテストされるメモリ・スパンによって異なります。メモリ・インタフェースのデータ幅、転送されるバイト数、およびクロック・サイクル数によって、メモリ・アクセスの効率を算出できます (**式 3-1**)。

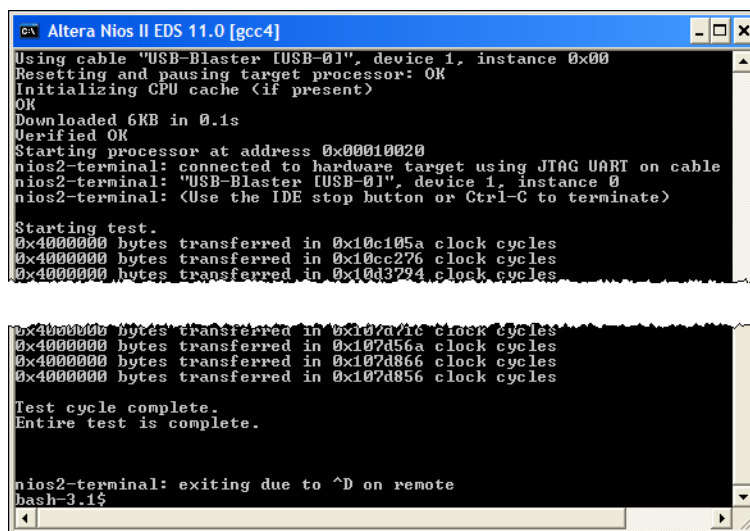
 このチュートリアルの場合、トップレベルの **Qsys** システム内の **SDRAM** コントローラは **32 ビット** のローカル・インタフェース幅を持っているため、**式 3-1** の「**バイト単位のメモリ・データ幅**」は **4 バイト** になります。

式 3-1. メモリ効率を算出する式

効率 = $100 \times \frac{\text{転送されたバイトの合計}}{\text{バイト単位のメモリ・データ幅} \times \text{合計クロック・サイクル}}$

 **図 3-3** に、コマンド・ウィンドウで表示される出力の例を示します。

図 3-3. ターミナルからの出力の例



```
Altera Nios II EDS 11.0 [gcc4]
Using cable "USB-Blaster [USB-01]", device 1, instance 0x00
Resetting and pausing target processor: OK
Initializing CPU cache (if present)
OK
Downloaded 6KB in 0.1s
Verified OK
Starting processor at address 0x00010020
nios2-terminal: connected to hardware target using JTAG UART on cable
nios2-terminal: "USB-Blaster [USB-01]", device 1, instance 0
nios2-terminal: <Use the IDE stop button or Ctrl-C to terminate>

Starting test.
0x4000000 bytes transferred in 0x10c105a clock cycles
0x4000000 bytes transferred in 0x10cc276 clock cycles
0x4000000 bytes transferred in 0x10d3794 clock cycles
0x4000000 bytes transferred in 0x10d7a76 clock cycles
0x4000000 bytes transferred in 0x107d56a clock cycles
0x4000000 bytes transferred in 0x107d866 clock cycles
0x4000000 bytes transferred in 0x107d856 clock cycles

Test cycle complete.
Entire test is complete.

nios2-terminal: exiting due to ^D on remote
bash-3.1$
```

テストは、デザインがメモリ全体に対するテストを終了するまでに続きます。テストを中断するには、コマンド・ウィンドウで **Ctrl+C** を入力します。

図 3-3 に示す最後のスループット値の効率を算出するには、16 進数の値を 10 進数に変換します。効率の計算は次のようになります。

転送された 0x4000000 バイトは、10 進数で 0d67108864 バイトになります。

0x107d856 クロック・サイクルは、10 進数で 0d17291350 クロック・サイクルになります。

従って、この例の効率は次のようになります。

$$100 \times 67108864 / (4 \times 17291350) = 97.0\%$$

メモリ・テストが正常に完了していない場合、ターミナルでは障害メッセージが表示されます。メモリ・テストで障害メッセージが発生する場合、この前の項のすべての指示を正常に実行したかどうか確認してください。1 つの接続ミスやメモリ・アドレス・アサインメントの誤りによって、ボードでのテスト・デザインが失敗することがあります。アルテラは、結果をチェックするために、完全なシステムを提供します。別の名前で完全なシステムをプロジェクト・ディレクトリにコピーすることで、2 つのシステムを横並びにして比較することができます。また、システムをアルテラ提供の完全なシステムに置き換えて、メモリ・テスト・デザインを正常に実行することができます。完全なシステムは、次のディレクトリにあります。

- `tt_qsys_design\completed_subsystems\pattern_checker_system.qsys`
- `tt_qsys_design\completed_subsystems\pattern_generator_system.qsys`
- `tt_qsys_design\completed_subsystems\completed_memory_tester_system\memory_tester_system.qsys`
- `\tt_qsys_design\quartus_ii_projects_for_boards\<development_board>\backup_and_completed_top_system\completed_top_system\top_system.qsys`

このチュートリアルでは、Quartus II ソフトウェアで使用可能なシステム・コンソールを使用してシステム・デザインを検証する方法について説明します。デザイン例ファイルには、システム・コンソールの Tcl コマンドによってシステムをエキササイズするためのスクリプトが含まれています。このシステムは、3. 階層システムの組み立てで説明したシステムに似ています。ただし、このシステムは、Nios II プロセッサ・システムの代わりに、JTAG-Avalon マスタ・ブリッジ・コンポーネントを使用してすべてのスレーブ・コンポーネントをドライブします。

スクリプトについて

`\quartus_ii_projects_for_boards\<development_board>\system_console` ディレクトリには、`run_sweep.tcl`、`base_address.tcl` および `test_cases.tcl` スクリプトが含まれています。これらのスクリプトにより、様々な開発ボード・プロジェクトでメモリ・テストをセットアップおよび実行することができます。スクリプトを表示することは、スレーブ・コンポーネント・レジスタをドライブするシステム・コンソール・コマンドを理解するのに役立ちます。同じ Qsys システム構造を維持する限り、このスクリプトは任意のボードで動作できます。

`run_sweep.tcl` ファイルはメイン・スクリプトであり、ほかの 2 つのスクリプトを呼び出します。`base_address.tcl` ファイルは、前章で使用されるスレーブ・コンポーネントのベース・アドレスに関するすべての情報を含みます。スレーブ・コンポーネントのベース・アドレスを変更する場合、それらを `base_address.tcl` ファイルで変更しなければなりません。`test_cases.tcl` ファイルには、メモリ・スパン、メモリ・ブロック・サイズ、およびメモリ・ブロック間の距離の設定が含まれています。

`run_sweep.tcl` ファイルには、次の動作のためのシステム・コンソール Tcl コマンドが含まれています。

- コンポーネントの初期化
- テスト・パラメータの調整
- PRBS パターン・チェッカ、PRBS パターン・ジェネレータおよび RAM コントローラの起動
- PRBS チェッカでのストップおよびフェイル・ビットを連続してポーリング

チュートリアル・プロジェクトを開く

3. 階層システムの組み立てを完了していない場合、Quartus II プロジェクトをセットアップするのに次のステップに従います。

1. チュートリアル・ファイルをダウンロードおよびインストールします（3 ページの「デザイン例ファイルのダウンロードおよびインストール」を参照）。

2. 次の完全なシステムをご使用のボードに該当する
tt_qsys_design\quartus_ii_projects_for_boards\<development_board> ディレクトリにコピーします。
 - **tt_qsys_design\completed_subsystems** ディレクトリからの2つの完全なシステム — **pattern_checker_system.qsys** および **patter_generator_system.qsys**
 - **tt_qsys_design\completed_subsystems\completed_memory_tester_system** ディレクトリからの完全なシステム — **memory_tester_system.qsys**
 - **tt_qsys_design\quartus_ii_projects_for_boards\<development_board>\back_up_and_completed_top_system\completed_top_system** ディレクトリからの完全なトップレベル・システム — **top-system.qsys**

 これらのシステムの作成については、2. Qsys システムの作成 および 3. 階層システムの組み立てを参照してください。
3. ご使用の開発ボードに該当する Quartus II プロジェクトを開きます（4 ページの「チュートリアル・プロジェクトを開く」を参照）。
 トップレベル Qsys ファイルを開くには、次のステップに従います。
 1. Tools メニューで、**Qsys** をクリックします。
 2. Qsys の File メニューで、プロジェクト・ディレクトリ内の **top_system.qsys** ファイルを開きます。

JTAG-Avalon マスタ・ブリッジの追加

JTAG-Avalon マスタ・ブリッジは、JTAG インタフェースおよびシステムのメモリ・テスト間のブリッジとしての機能を果たします。このブリッジをトップレベル・システムに追加するには、これらのステップに従います。

1. **Component Library** タブで、**Bridges and Adapters** を展開し、**Memory Mapped** を展開します。
2. **JTAG to Avalon Master Bridge** をクリックして、**Add** をクリックします。パラメータ・エディタが表示されます。
3. パラメータ・エディタで **Finish** をクリックして、デフォルトのパラメータを受け入れます。
4. **System Contents** タブで、インスタンスの名前を **jtag_to_avalon_bridge** に変更します。
5. **jtag_to_avalon_bridge master** インタフェースを **memory_tester_subsystem slave** インタフェースに接続させます。
6. **jtag_to_avalon_bridge** クロック・ドメインを **sdram_sysclk** に設定します。
7. ALTMEMPHY ベースのデザインの場合、**jtag_avalon_bridge clk_reset** インタフェースを **ext_clk clk_reset** インタフェースおよび **sdram reset_request_n** インタフェースに接続させます。
 UniPHY ベースのデザインの場合、**jtag_avalon_bridge clk_reset** インタフェースを **ext_clk clk_reset** インタフェースおよび **sdram afi_reset** インタフェースに接続させます。

8. ALTMEMPHY ベースのデザインの場合、**jtag_avalon_bridge** master_reset インタフェースを **memory_tester_subsystem** reset インタフェースおよび **sdram** soft_reset_n インタフェースに接続させます。
UniPHY ベースのデザインの場合、**jtag_avalon_bridge** master_reset インタフェースを **memory_tester_subsystem** reset インタフェースおよび **sdram** soft_reset インタフェースに接続させます。
9. **cpu_subsystem** の機能はブリッジおよびシステム・コンソールに置き換えられるため、**cpu_subsystem** をディセーブルしてシステムから削除します。**Use** カラムで、**Use** をオフにします。
10. File メニューで、**Save** をクリックします。

開発ボードにシステム・コンソールをコンパイルおよび使用

デザインのスクリプト例は、様々なブロック・サイズに対してメモリを繰り返してテストします。ブロック・サイズは、バック・ツー・バック・リード/ライトの単一インスタンスにグループされるバイト数です。プログラミング・ファイルを開発ボードにダウンロードするには、次のステップに従います。

1. Qsys で、**Generation** タブをクリックします。
2. **Create HDL design files for synthesis** がオンであることを確認します。このステップは、ほかのオプションを必要としません。
3. **Generate** をクリックします。Qsys は、システムの HDL ファイル、および Quartus II コンパイル作業に必要な HDL ファイルのリストを提供する **.qip** ファイルを生成します。
4. 生成完了後に、**Close** をクリックします。
5. Quartus II ソフトウェアの Project メニューで、**Add/Remove Files in Project** をクリックして、プロジェクトがこのシステムの **.qip** ファイルを含むことを確認します。
6. Processing メニューで、**Start Compilation** をクリックします。
7. サポートされるプログラミング・ケーブルに開発ボードを接続させます。
8. Tools メニューで、**Programmer** をクリックします。
9. **Programmer** が正しいプログラミング・ハードウェアを表示することを確認します。そうでない場合は、**Hardware Setup** をクリックして、正しいプログラミング・ハードウェアを選択し、**Close** をクリックします。
10. コンパイルの完了後に、**OK** をクリックします。
11. **Start** をクリックすることで、デバイスをプログラムします。
12. IQsys の Tools メニューで、**System Console** をクリックします。
13. システム・コンソールでスクリプトを実行する前に、Tcl スクリプトが存在するディレクトリに移動しなければなりません。Tcl Console ウィンドウで次のコマンドを入力して、ディレクトリを変更します。

```
cd system_console
```
14. File メニューで、**Execute Script** をクリックします。

15. メモリ・テストを開始するには、
`tt_qsys_design\quartus_ii_projects_for_boards\<development_board>\system_console` からの `run_sweep.tcl` ファイルを実行します。表 4-1 に、`run_sweep.tcl` スクリプトがシステム・コンソールを通じて実行するタスクを示します。

表 4-1. `run_sweep.tcl` スクリプト・タスク

スクリプト・タスク	説明
コンポーネントの初期化	<ol style="list-style-type: none"> 1. <code>csr</code> インタフェースにアクセスすることで、マルチプレクサおよびデマルチプレクサを入力/出力 B に切り換えます。 2. PRBS ジェネレータ・コアおよび PRBS チェッカ・コアの <code>csr</code> スレーブ・インタフェースに書き込み、そして次のタスクを実行します。 <ol style="list-style-type: none"> a. 無限ペイロード・サイズをディセーブルします。 b. デザインのメモリ・スパンと同様なペイロード・サイズを書き込みます。 3. 次のものを RAM テスト・コントローラの <code>csr</code> スレーブ・インタフェースに書き込みます。 <ol style="list-style-type: none"> a. テストの開始のための RAM ベース・アドレス b. デザインのメモリ・スパンと同様な転送長 c. 同時リード/ライト・イネーブル これは、メモリ・ブロックにアクセスするリード作業とライト作業が同時に実行されるかどうかを決めます。この設定を無効にすることは、最大のスループットを提供しますが、リードおよびライト間の切り換えをテストしません。.
テスト・パラメータの調整	<ul style="list-style-type: none"> ■ 1 ループ内で様々なブロック・サイズを書き込みます。 ■ 1 ループ内で様々なブロック間距離を書き込みます。
テストを開始する	<ol style="list-style-type: none"> 1. 開始ビットを PRBS ジェネレータ・コアの <code>csr</code> スレーブ・インタフェースに書き込みます。 2. 開始ビットを PRBS チェッカ・コアの <code>csr</code> スレーブ・インタフェースに書き込みます。 3. RAM コントローラ <code>csr</code> インタフェースの開始ビットに書き込みます。
PRBS チェッカのストップおよび障害ビットを連続してポーリングする	<ol style="list-style-type: none"> 1. PRBS チェッカ・コアの <code>csr</code> スレーブ・インタフェースをポーリングして、障害ビットがイネーブルされたかどうかを決定し、そしてランビット（並んだビット）をポーリングして、テストがいつ完了するかを決めます。 2. テスト完了時に、異なるメモリ・ブロック間距離およびメモリ・ブロック・サイズを使用してプロセスを繰り返します。

`run_sweep.tcl` スクリプトを実行した後、Messages ウィンドウではテストの進行状況が表示されます。テストは、異なるメモリ・ブロック・サイズおよびメモリ・ブロック間距離を使用して、SDRAM にスイープ・テストを実行します。テストが正常に終了した後、Tcl コンソールでは次のメッセージが表示されます。

```
.... All tests have finished without any Failures ...
```

この章では、Qsys および Avalon 検証 IP スイートによってカスタム・コンポーネントを検証する方法を示します。Qsys を使用してテスト対象のデザインに対してテストベンチ・システムを生成し、そして ModeSim シミュレータを使用して機能シミュレーションを実行します。Qsys が生成したテストベンチは、Avalon 検証 IP スイート・コンポーネントを使用します。


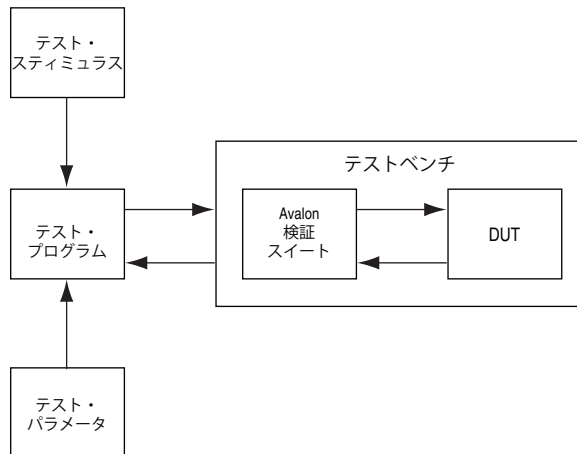
 Avalon 検証 IP スイートについて詳しくは、「[Avalon Verification IP Suite User Guide](#)」を参照してください。

図 5-1 に、一般的なテスト環境のブロック図を示します。

図 5-1. 一般的なテスト環境のブロック図



Qsys におけるテストベンチ・システムの生成

この項では、Qsys でテスト対象のデザインに対してテストベンチ・システムを生成します。この章で使用されるテスト対象のデザインは、ほかの章からのカスタム・パターン・ジェネレータです。このコンポーネントはメモリ・デバイスをテストするために、高速ストリーミング・データを生成します。このソフト・プログラマブルなカスタム・パターン・ジェネレータは複数のテスト・パターンを生成できます。このコンポーネントでは、パターン・データおよびパターン長がプログラムされています。パターンが終了したら、カスタム・パターン・ジェネレータはパターンの最初のエレメントに戻ります。

Qsys が生成したテストベンチ・システムを使用しない場合、Avalon 検証スイートの BFM またはユーザー独自のモデルをシミュレーションに追加することで、ユーザー独自の Qsys テストベンチ・システムを作成できます。また、デザインまたはテスト対象の Qsys システムに対して Qsys シミュレーション・モデルを生成し、そして独自のカスタム HDL・テストベンチによってシミュレーション・ステイミュラスを提供します。

チュートリアル・プロジェクトを開く

チュートリアル・デザイン例ファイルには、作業環境をセットアップする Quartus II プロジェクトが含まれています。プロジェクトを開くには、次のステップに従います。

1. **はじめに**で説明したように、チュートリアル・デザイン・ファイルをダウンロードします。
2. Quartus II ソフトウェアで、`\simulation_tutorial` ディレクトリからの `.pqfqsys_sim_tutorial.pqf` ファイルを開きます。

テスト対象のデザインに対して新規の Qsys システムを作成

テスト対象のデザインに対して新規の Qsys システムを作成するには、次のステップに従います。

1. Quartus II ソフトウェアの **File** メニューで、**New** をクリックします。
2. **Qsys System File** を選択して、**OK** をクリックします。
3. **Initializing Complete** ウィンドウ上の **Close** をクリックします。
4. 新しいシステムは 1 つのクロック・ソースをインスタンス化します。ただし、システムにはクロック・ソースが必要ないため、それを削除します。**clk_0** インスタンスをクリックして、**X** アイコンをクリックするか、または右クリックして **Remove** を選択します。
5. **Component Library** タブで、**Memory Test Microcores** を展開します。
6. **Custom Pattern Generator** を選択して、**Add** をクリックします。
7. **Finish** をクリックします。
8. インスタンスの名前を `pg` に変更します。
 - a. **Module** カラムで、**custom_pattern_generator_0** を右クリックし、**Rename** を選択します。
 - b. インスタンス名に対して `pg` を入力します。

テスト対象のデザインのすべてのインタフェースをエクスポート

テスト対象のデザインのインタフェースをエクスポートするには、次のステップに従います。

1. **System Contents** タブの **Export** カラムで、インタフェースごとに **Click to export** をクリックします。デフォルトのエクスポート名を保持します。
2. システムを名前を付けて保存します。
 - a. **File** メニューで、**Save As** をクリックします。
 - b. ファイル名に `pattern_generator` を入力して **Save** をクリックします。

Qsys テストベンチ・システムの生成

テスト対象のデザインに対してテストベンチ・システムを生成するには、次のステップに従います。

1. **Generation** タブをクリックします。
2. **Simulation** で、**Create testbench Qsys system** に対して **Standard, BFM for standard Avalon interfaces** を選択します。
3. **Synthesis** で、**Create HDL design files for synthesis** および **Create block symbol file (.bsf)** をオフにします。
4. **Generate** をクリックします。
5. Qsys がテストベンチを生成した後、メッセージ・ウィンドウ上の **Close** をクリックします。



Qsys は、このテストベンチ・システムを `\simulation_tutorial\pattern_generator\testbench` ディレクトリに生成します。

Create testbench simulation model をオンにすることで、Qsys テストベンチ・システム用のシミュレーション・モデルを同時に生成することができます。ただし、Qsys が生成したテストベンチ・システムのコンポーネント名は自動的に割り当てられ、BFM に対するテスト・プログラムを容易に実行できるよう、インスタンス名を制御する場合があります。このチュートリアルでは、シミュレーション・モデルを生成する前に Qsys テストベンチ・システムを編集します。

Qsys テストベンチ・システムのシミュレーション・モデルの生成


この項では、生成された Qsys テストベンチ・システムを開いて、チュートリアル・デザイン・ファイルとともに提供されるテスト・プログラムに一致するように BFM コンポーネントのインスタンス名を変更します。テストベンチのシミュレーション・モデルも生成されます。BFM のインスタンス名を変更し、シミュレーション・モデルを生成するには、次のステップに従います。

1. Qsys で、`simulation_tutorial\pattern_generator\testbench` ディレクトリからテスト・ベンチシステムの `pattern_generator_tb.qsys` を開きます。
2. **System Contents** タブで、表 5-1 に示すようにインスタンスの名前を変更します。


表 5-1. 新しいインスタンス名

Qsys が生成したコンポーネントの名前	新しいインスタンス名
pattern_generator_inst	DUT
pattern_generator_inst_pg_clock_bfm	clock_source
pattern_generator_inst_pg_clock_reset_bfm	reset_source
pattern_generator_inst_pg_csr_bfm	csr_master
pattern_generator_inst_pg_pattern_access	pattern_master
pattern_generator_inst_pg_pattern_output_bfm	pattern_sink

3. BFM コンポーネントのいずれかをダブル・クリックし、パラメータ・エディタを開いて設定を表示します。これらのコンポーネントは、コンポーネント・ライブラリ内の **AvalonVerification Suite** グループにある BFM です。必要に応じて、デザインに十分なテスト・カバレッジを確保するように、パラメータ・エディタで設定を変更することができます。

 Qsys が生成したテストベンチは、挿入された BFM を、ドライブするデザインからのエクスポートされたインタフェースに一致させます。BFM にステイミュラスを提供するテスト・プログラムは、一致するインタフェースを考慮する必要があります。例えば、エクスポートされた Avalon-MM スレーブ・インタフェース（ワード・アラインメントされたアドレスを予期している）は、Avalon マスタのデフォルトのバイトまたはシンボル・アドレスの代わりに、Avalon マスタ BFM に接続されています。

4. **Cancel** をクリックして、変更を行わずにパラメータ・エディタを閉じます。
5. **Generation** タブで、**Simulation** の下の **Create simulation model** に対して、**Verilog** を選択します。
6. **Create testbench Qsys system** および **Create testbench simulation model** に対して、**None** を選択します。
7. **Synthesis** セクションで、すべてのオプションをオフにします。
8. システムを保存します。
9. **Generate** をクリックします。
10. Qsys がテストベンチを生成した後、メッセージ・ウィンドウ上の **Close** をクリックします。

 Qsys は、テストベンチ・システムのシミュレーション・モデルを `\simulation_tutorial\pattern_generator\testbench\pattern_generator_t\b\simulation` ディレクトリに生成します。

Qsys は、シミュレーション・モデルおよび ModelSim シミュレーション・スクリプト (`msim_setup.tcl`) を生成します。これらのファイルは、シミュレーションに必要なファイルをコンパイルし、ModelSim シミュレータ内のシミュレーションをロードするためのコマンドをセットアップします。この ModelSim スクリプトを ModelSim-Altera で実行して、シミュレーションに必要なすべてのものをコンパイル、調整またはロードすることができます。このチュートリアルは、シミュレーション・ティミュラスを提供する外部テスト・プログラムを含みます。チュートリアル・デザイン・ファイルには、トップレベル・シミュレーション・ファイルおよびテスト・プログラムをコンパイルし、必要なファイルをコンパイルする Qsys が生成したスクリプトを呼び出すためのシミュレーション・スクリプト、`load_sim.tcl` が含まれています。

ModelSim-Altera ソフトウェアにおけるシミュレーションの実行

この項では、ModelSim-Altera ソフトウェアで、作成されたテストベンチに対してシミュレーションを実行します。このシミュレーションを完成させるには、デザイン・ファイルで提供されるテスト・プログラムを使用します。テスト・プログラムは以下の動作を実行します。

- パターン・ファイルを読み出す
- パターン・マスタ BFM を介してテスト対象のデザインにパターンを書き込む
- CSR マスタ BFM を介して、テスト対象のデザインの様々なオプションを設定
- テスト対象のデザインのパターン生成を開始する
- テスト対象のデザインに生成されたデータを収集する
- 結果をオリジナルのパターン・ファイルに比較する

テストは、テスト対象のデザインに **Walking Ones** パターンを書き込むと開始されます。

シミュレーション環境のセットアップ

このチュートリアルは、Qsys が生成したテストベンチおよび ModelSim シミュレーション・スクリプトとともに使用できるテスト・プログラム・ファイルを含みます。Qsys シミュレーションのサポートについて詳しくは、シミュレーション・スクリプトの `\simulation_tutorial\load_sim.tcl` を参照してください。


`load_sim.tcl` スクリプトは、Qsys が生成したシミュレーション・モデルおよび ModelSim スクリプトに正確な階層パスをセットアップするために、シミュレーションの変数を設定します。さらに、このスクリプトは、シミュレーション用のトップレベル・インスタンスの名前を識別し、Qsys が生成したファイルの位置へのパスを提供します。メモリ初期化などの一部の機能は、シミュレーション・モデルでの正しい階層パス名に依存します。次に、スクリプトは下記の動作を実行します。

- Qsys が生成した ModelSim シミュレーション・スクリプト、`msim_setup.tcl` を使用する
 - `msim_setup.tcl` スクリプトで定義されているコマンド・エイリアスを使用して、Qsys テストベンチ・シミュレーション・モデル用のファイルをコンパイルおよび調整する。
 - チュートリアル用の追加のシミュレーション・ファイル（テスト・プログラムを初期化するテスト・プログラム・ファイルおよびトップレベル・シミュレーション・ファイル）をコンパイルおよび調整する
 - ModelSim 波形ビューに信号を提供する `wave.do` ファイルをロードする
- 変更を行わずに `load_sim.tcl` スクリプトを閉じます。


シミュレーションの実行

シミュレーションを実行するには、次のステップを実行します。

1. ModelSim-Altera ソフトウェアを起動します。
2. File メニューで、**Change Directory** をクリックし、`\simulation_tutorial` ディレクトリに移動し、**OK** をクリックします。
3. Compile メニューで、**Compile Options** をクリックします。
4. **Verilog & SystemVerilog** タブをクリックし、**Use SystemVerilog** を選択して **OK** をクリックします。
5. File メニューで、**Load** をクリックします。

 ModelSim-Altera の Transcript ウィンドウをアクティブにしてください。そうしないと、**Load** 機能がディセーブルされます。

6. **load_sim.tcl** スクリプトを選択して、**Open** をクリックします。

 警告メッセージは、ALTSYNCRAM メガファンクションにおける未使用の接続に関連します。これらのポートを使用しないため、警告メッセージを無視しても問題ありません。

7. 40 μ s の間シミュレーションを実行します。シミュレーションを実行するには、ModelSim-Altera の Transcript ウィンドウで次のコマンドを入力します。

```
run 40us
```

 h コマンドを実行して、**msim_setup.tcl** スクリプトに使用可能なオプションを表示することができます。

8. 結果を観測します。例 2 に、表示されるメッセージを示します。

例 2. シミュレーションの結果

```
INFO: top.tb.reset_source.reset_deassert: Reset deasserted
INFO: top.pgm: Starting test walking_ones.hex
INFO: top.pgm.read_file: Read file walking_ones.hex success
INFO: top.pgm.read_file: Read file walking_ones_rev.hex success
INFO: top.pgm: Test walking_ones.hex passed
```

9. **Low Frequency** パターンでテストを実行するには、**\simulation_tutorial\test_include.svh** を変更します。(表 5-2)

表 5-2. Low Frequency パターン・テストの値

マクロ	新しい値
PATTERN_POSITION	0
NUM_OF_PATTERN	2
NUM_OF_PAYLOAD_BYTES	256
FILENAME	low_freq.hex
FILENAME_REV	low_freq_rev.hex

10. **load_sim.tcl** スクリプトをリロードし、40 μ s の間にシミュレーションを実行し、Transcript ウィンドウで結果を観測します。例 3 に、表示されるメッセージを示します。

例 3. Low Frequency パターン・テストの Transcript メッセージ

```
INFO: top.pgm: Starting test low_freq.hex
INFO: top.pgm.read_file: Read file low_freq.hex success
INFO: top.pgm.read_file: Read file walking_ones_rev.hex success
INFO: top.pgm: Test low_freq.hex passed
```

11. 乱数パターン・テストの実行するには、`\simulation_tutorial\test_include.svh` を変更します。(表 5-3)

表 5-3. 乱数パターン・テストの値

マクロ	新しい値
PATTERN_POSITION	32
NUM_OF_PATTERN	64
NUM_OF_PAYLOAD_BYTES	1024
FILENAME	random_num.hex
FILENAME_REV	random_num_rev.hex

12. `load_sim.tcl` スクリプトをリロードし、40 μ s の間にシミュレーションを実行すると、例 4 に示す結果が表示されます。

例 4. 乱数パターン・テストの Transcript メッセージ

```
INFO: top.pgm: Starting test random_num.hex
INFO: top.pgm.read_file: Read file random_num.hex success
INFO: top.pgm.read_file: Read file random_num_rev.hex success
INFO: top.pgm: Test random_num.hex passed
```

この章では、カスタム・パターン・ジェネレータに対してシミュレーション環境をセットアップし、BFM テスト・コードを使用してシミュレーションを実行します。この方法でユーザー独自のカスタム Qsys コンポーネントをテストし、完全なシステムに統合する前に機能を検証することができます。また、この方法で完全な Qsys システムに対してテストベンチ・システムを作成し、そして BFM でトップレベル・システムの動作をテストすることができます。

この章では、本資料およびアルテラについての追加情報を提供します。

改訂履歴

下表に、本資料の改訂履歴を示します。

日付	バージョン	変更内容
2011年4月	2.0	Qsys v11.0 のための更新
2010年12月	1.1	メンテナンス・リリース。

アルテラへのお問い合わせ

アルテラの製品に関する最新の情報については、次の表を参照してください。









お問い合わせ先 (Note 1)	お問い合わせ方法	アドレス
技術的なご質問	ウェブサイト	www.altera.com/support
技術トレーニング	ウェブサイト	www.altera.com/training
	電子メール	custrain@altera.com
アルテラの資料に関するお問い合わせ	ウェブサイト	www.altera.com/literature
一般的なお問い合わせ	電子メール	nacomp@altera.com
ソフトウェア・ライセンスに関するお問い合わせ	電子メール	authorization@altera.com

注：

(1) 詳しくは、日本アルテラまたは販売代理店にお問い合わせください。

表記規則

本資料では、以下の表に示す表記規則を使用しています。

書体	意味
太字かつ文頭が大文字	コマンド名、ダイアログ・ボックス・タイトル、ダイアログ・ボックス・オプション、およびその他の GUI ラベルを表します。 例： Save As ダイアログ・ボックス
太字	ディレクトリ名、プロジェクト名、ディスク・ドライブ名、ファイル名、ファイルの拡張子、およびソフトウェア・ユーティリティ名を表します。 例： %qdesigns ディレクトリ、 d: ドライブ、および chiptrip.gdf ファイル
斜体かつ文頭が大文字	資料のタイトルを表します。例： <i>AN 519: Stratix III デザイン・ガイドライン</i>
斜体	変数を表します。例： <i>n + 1</i> 変数名は、山括弧 (< >) で囲んでいます。例： <file name> および <project name>.pof ファイル
文頭が大文字	キーボード・キーおよびメニュー名を表します。 例： Delete キー、 Options メニュー
「小見出しタイトル」	かぎ括弧は、資料内の小見出しおよび Quartus II Help トピックのタイトルを表します。例：「表記規則」
Courier フォント	信号、ポート、レジスタ、ビット、ブロック、およびプリミティブ名を表します。例： data1、tdi、および input。 アクティブ Low 信号は、サフィックス n で表されています。例： resetn コマンドライン・コマンド、および表示されているとおりに入力する必要があるものを表します。例： c:\qdesigns\tutorial\chiptrip.gdf また、Report ファイルのような実際のファイル、ファイルの構成要素（例：AHDL キーワードの SUBDESIGN）、ロジック・ファンクション名（例：TRI）も表します。
	矢印は、Enter キーを押すことを示しています。
1.、2.、3.、および a.、b.、c.、など。	順など項目の順序が重要なものは、番号が付けられリスト形式で表記されています。
	順など項目の順序が重要なものは、番号が付けられリスト形式で表記されています。
	指差しマークは、要注意箇所を表しています。
	疑問符は、関連情報を持つソフトウェア・ヘルプ・システムへの参照先を示しています。
	足跡マークは、詳細情報の参照先を示しています。
	注意は、製品または作業中のデータに損傷を与えたり、破壊したりするおそれのある条件や状況に対して注意を促します。
	警告は、ユーザーに危害を与えるおそれのある条件や状況に対して注意を促します。
	封筒は、アルテラ・ウェブサイトの Email Subscription Management Center へのリンクです。ここでサインアップすることで、アルテラ資料の更新通知を受信することができます。