ISim ハードウェア協調シミュレーション : Spartan-6 メモリ コントローラーおよびオンボード DDR2 メモリ

UG818 (v 13.1) 2011 年 3 月 18 日

XILINX®

本資料は英語版 (v.13.1) を翻訳したもので、内容に相違が生じる場合には原文を優先します。
資料によっては英語版の更新に対応していないものがあります。
日本語版は参考用としてご使用の上、最新情報につきましては、必ず最新英語版をご参照ください。

目次

XILINX®

第 1 章

概要

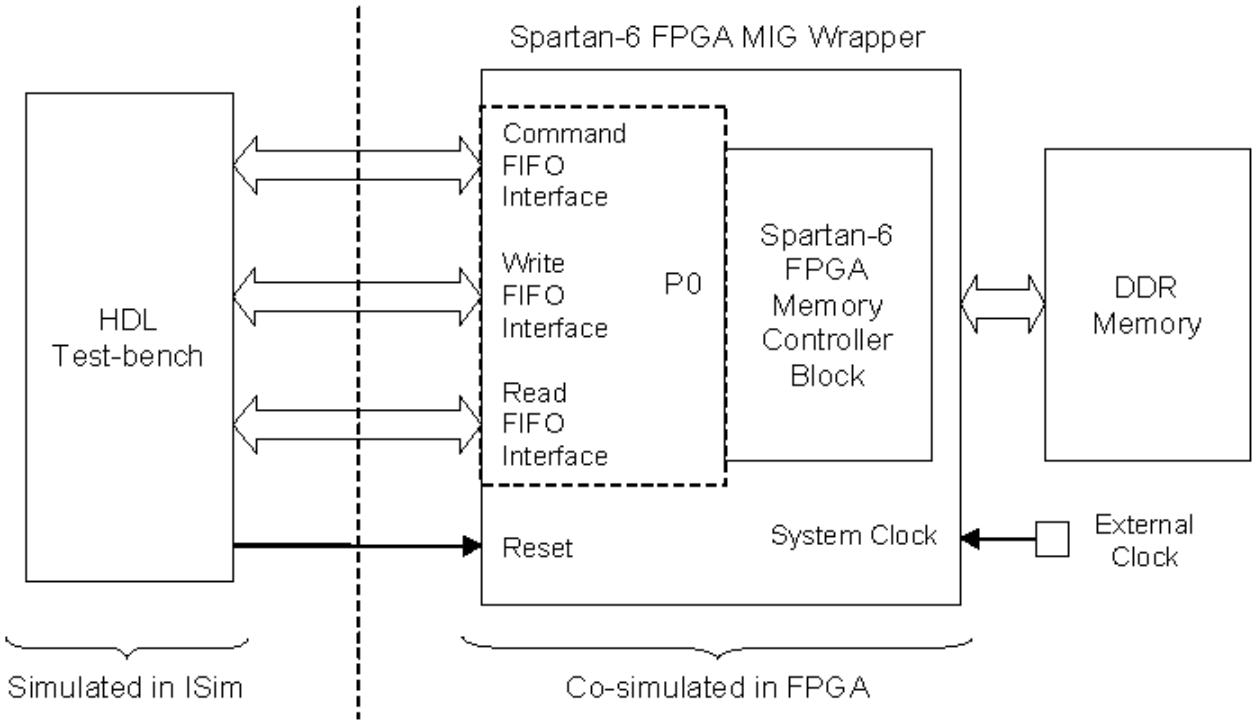
このチュートリアルでは、ISim ハードウェア協調シミュレーションを使用して、Spartan®-6 FPGA メモリ コントローラー ブロック (MCB) と通信し、HDL テストベンチからランタイムで外部 DDR2 メモリを駆動する方法を説明します。

外部メモリは、多量のメモリを必要とするエンベデッド、イメージ、ビデオ処理アプリケーションでよく使用されます。

外部メモリを使用する FPGA デザインを開発する場合、メモリ コントローラーおよび外部メモリ モジュールを含むデザイン全体を検証するのは困難です。従来は、ソフトウェアでデザイン全体をシミュレーションするか、ハードウェアでデザイン全体を実行していました。すべてをソフトウェアでシミュレーションする方法は、デザインを完全に可視化でき、テストベンチまたはデザインを変更して即再検証できるという利点がありますが、外部メモリ モジュールの正確なシミュレーション モデルを得ること、現実的なシミュレーション速度を達成することは困難です。デザインをハードウェアで実行するとこれらの問題は回避されますが、デザインの可視性が低下し、ハードウェアでのテストベンチの設定および変更は複雑です。

ISim ハードウェア協調シミュレーションを使用すると、デザインの一部をハードウェアで実行し、残りの部分をソフトウェアでシミュレーションできます。たとえば、メモリ コントローラーおよび外部メモリをハードウェアで実行すると、これらを正確にモデリングすることができ、シミュレーションを高速に実行できます。テストベンチおよびデザインに含まれるアプリケーション ロジックは、簡単にすばやく変更、検証、デバッグできるようにソフトウェアでシミュレーションします。次の図に、メモリ コントローラーおよび外部メモリを使用するデザインを分割して、ISim ハードウェア協調シミュレーションを利用する方法を示します。

ISim ハードウェア協調シミュレーション用にメモリ デザインを分割する方法



要件

このチュートリアルを実行するには、次のソフトウェアおよびハードウェアが必要です。

- ザイリンクス ISE® Design Suite バージョン 13.1
- Spartan®-6 FPGA SP601 評価キット

チュートリアル ファイル

ファイル	説明
mig_dut.v	MIG コアをインスタンシエートし、MIG コアの c3_p0_cmd_clk、c3_p0_wr_clk、および c3_p0_rd_clk を 1 つの入力クロック c3_clk0 に接続するラッパー
mig_hw_tb.v	MCB に対して読み出しトランザクションおよび書き込みトランザクションを発行する Verilog タスクを含む最上位テストベンチ
mig_dut_hwcosim.ucf	ハードウェア協調シミュレーション用のカスタム制約ファイル。mig_dut モジュールのどのポートを外部 I/O にマップするか、どのポートをテストベンチで制御するかを指定します。
init.tcl	ISim で testmem.tcl を読み込み、シミュレーションを初期化するためのカスタム シミュレーション コマンド ファイル
testmem.tcl	ISim コンソールでテストベンチの test_memory Verilog タスクを実行するのに使用する testmem Tcl コマンドを含むファイル
mig_hw_tb.wcfg	カスタム波形コンフィギュレーション ファイル
mig_hw_tb.prj	コマンド ライン フロー用の ISim プロジェクト ファイル

ファイル	説明
hwcosim.bsp	SP601 ボード上で、ハードウェア協調シミュレーション インターフェイス クロックとして 200MHz の差動クロックの代わりに 27MHz ユーザー クロックを使用するための変更されたハードウェア協調シミュレーション ボード サポート
full_compile.bat	Fuse コマンド ラインを使用してハードウェア協調シミュレーション用にデザインを完全にコンパイルする Windows バッチ ファイル
full_compile.sh	Fuse コマンド ラインを使用してハードウェア協調シミュレーション用にデザインを完全にコンパイルする Linux シェル スクリプト
incr_compile.bat	Fuse コマンド ラインを使用してハードウェア協調シミュレーション用にテストベンチをインクリメンタルにコンパイルする Windows バッチ ファイル
incr_compile.sh	Fuse コマンド ラインを使用してハードウェア協調シミュレーション用にテストベンチをインクリメンタルにコンパイルする Linux シェル スクリプト
run_isim.bat	ISim シミュレーションを起動する Windows バッチ ファイル
run_isim.sh	ISim シミュレーションを起動する Linux シェル スクリプト

メモ： このチュートリアルを実行する際は、すべてのデータ ファイルを作業ディレクトリにコピーしてください。

XILINX®

第 2 章

チュートリアル

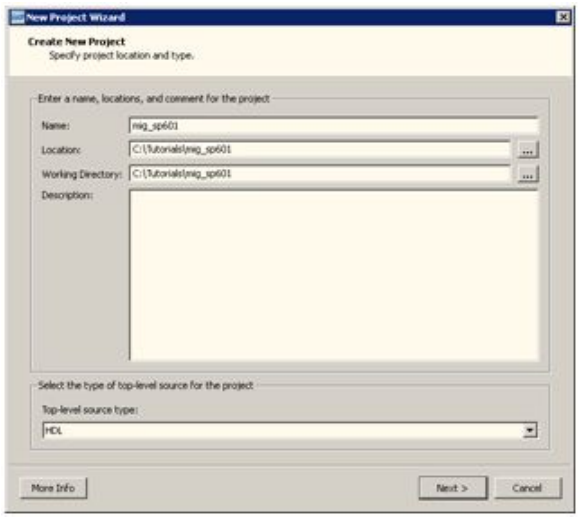
この章では、ISim ハードウェア協調シミュレーションを使用してメモリ デザインを実行する 5 つの手順を実行します。

1. CORE Generator™ の Memory Interface Generator (MIG) ツールを使用して、Spartan®-6 メモリのリファレンス デザインを生成します。
2. メモリのリファレンス デザインを実行するテストベンチを作成します。
3. カスタム制約ファイルを作成して、ISim で制御するポートと外部 I/O にマップするポートを指定します。
4. ハードウェア協調シミュレーションを実行するデザインのテストベンチをコンパイルします。
5. ターゲット FPGA ボードを PC に接続し、ISim シミュレーションを実行します。

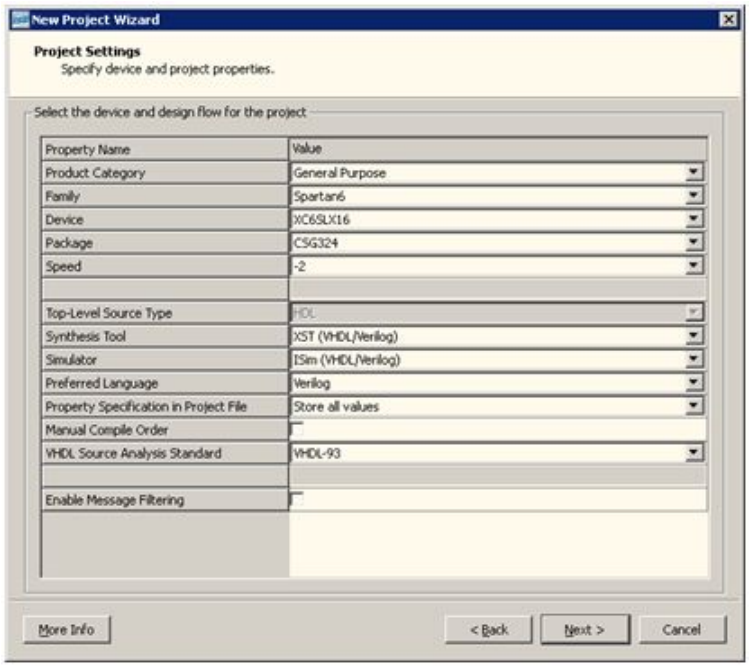
手順 1 : MIG ツールを使用したデザインの生成

Spartan-6 FPGA には、エンベデッド マルチポート メモリ コントローラー ブロック (MCB) が含まれており、外部 DDR メモリとの簡単で確実なインターフェイスを提供します。CORE Generator に含まれる Memory Interface Generator (MIG) ツールを使用すると、MCB のインターフェイスを簡単に作成できます。このチュートリアルでは、MIG ツールで生成したリファレンス デザインを使用し、Spartan-6 FPGA SP601 評価キットで動作する ISim ハードウェア協調シミュレーション テストベンチを作成します。

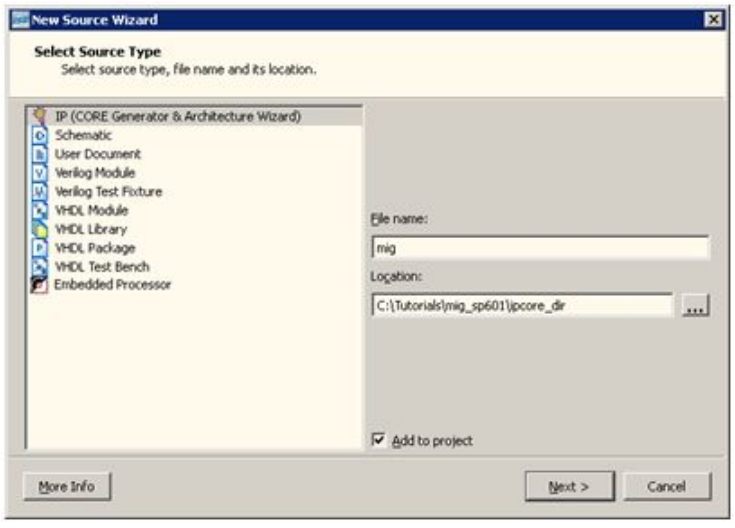
1. ISE® Project Navigator を起動します。
2. [File] → [New Project] をクリックし、New Project Wizard を開きます。プロジェクト名 (mig_sp601) と保存ディレクトリを入力します。[Next] をクリックします。



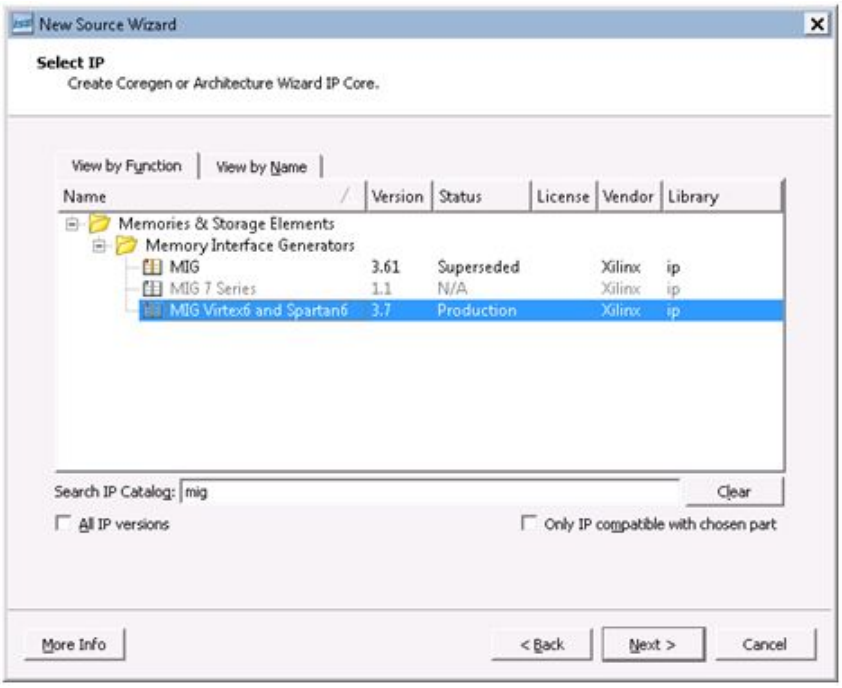
3. [Project Settings] ページで、[Family] に [Spartan6]、[Device] に [XC6SLX16] (SP601 ボードの Spartan-6 デバイス)、[Package] に [CSG324]、[Speed] に [-2] を選択します。[Simulator] に [ISim]、[Preferred Language] に [Verilog] を選択します。[Next] をクリックして [Project Summary] ページで [Finish] をクリックし、プロジェクトの作成を完了します。



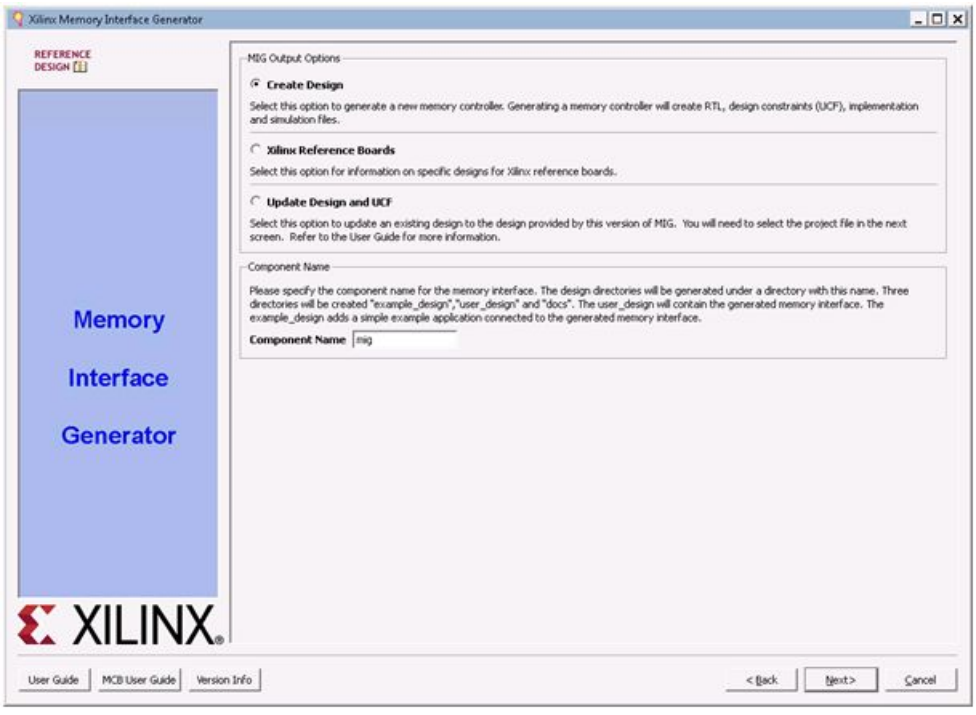
4. [Project] → [New Source] をクリックし、New Source Wizard を開きます。[IP (CORE Generator & Architecture Wizard)] を選択し、[File name] に「mig」と入力します。[Next] をクリックします。



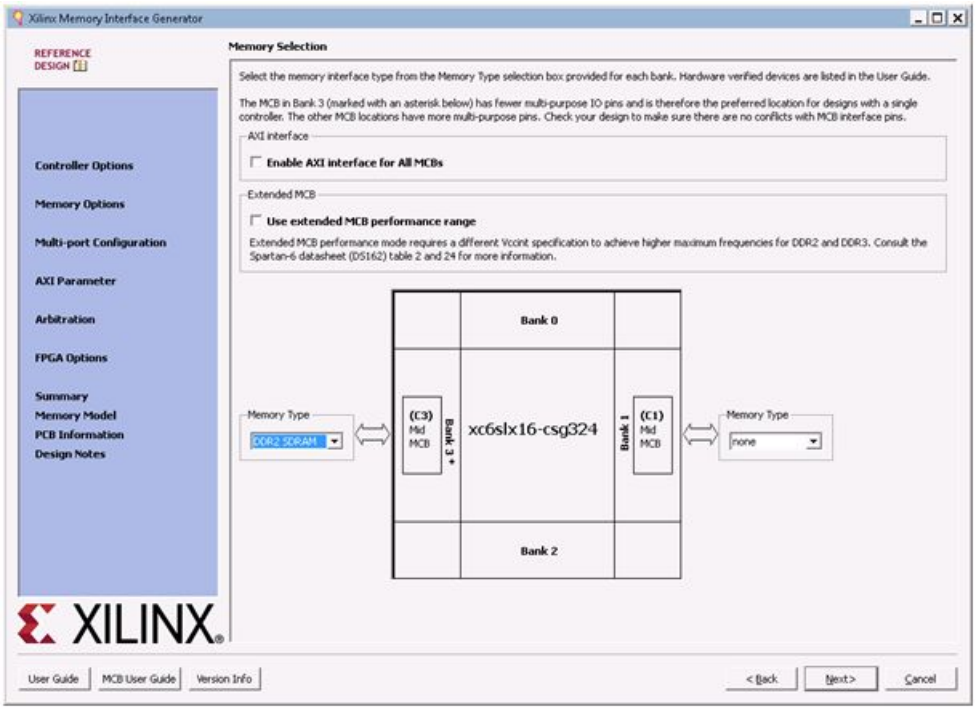
5. IP リストから [MIG] のバージョン 3.7 を選択します。[Next] をクリックし、次のダイアログボックスで [Finish] をクリックします。



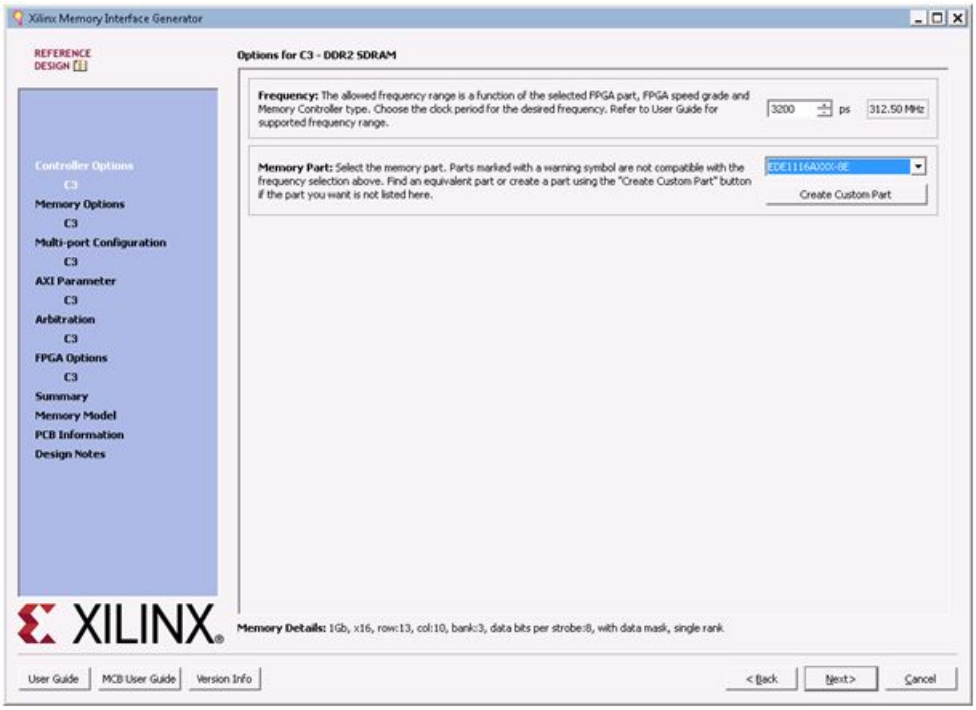
6. MIG の GUI が開いたら、[Create Design] をオンにして新しい MCB ベースのメメモリ インターフェイスを作成します。[Component Name] に「mig」と入力します。[Next] をクリックします。



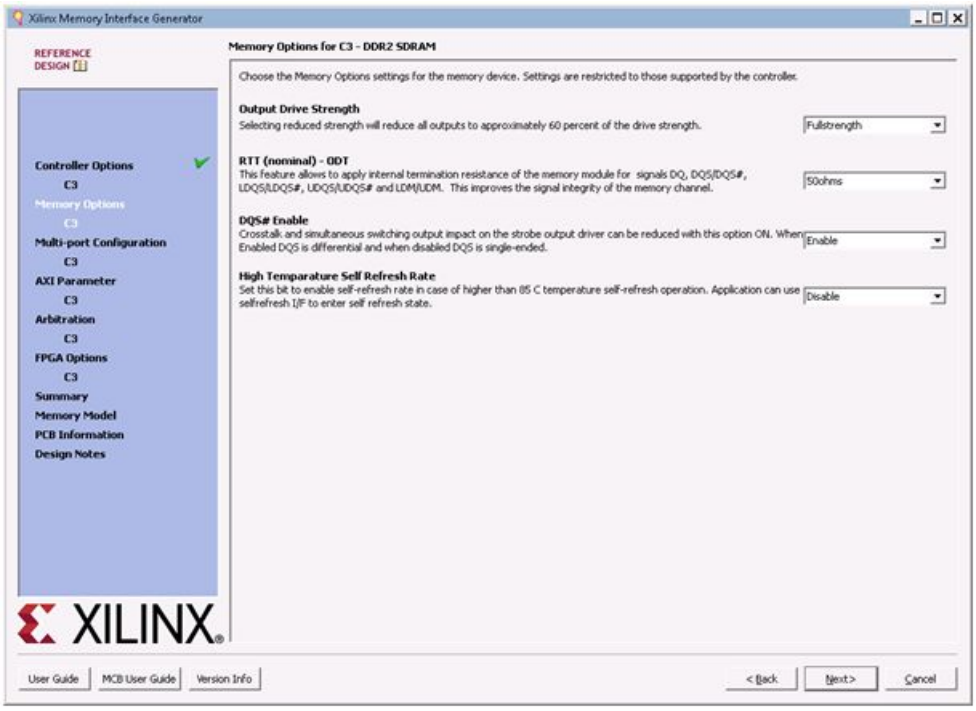
7. バンク 3 の MCB (C3) の [Memory Type] を [DDR2 SDRAM] に設定します。[Next] をクリックします。



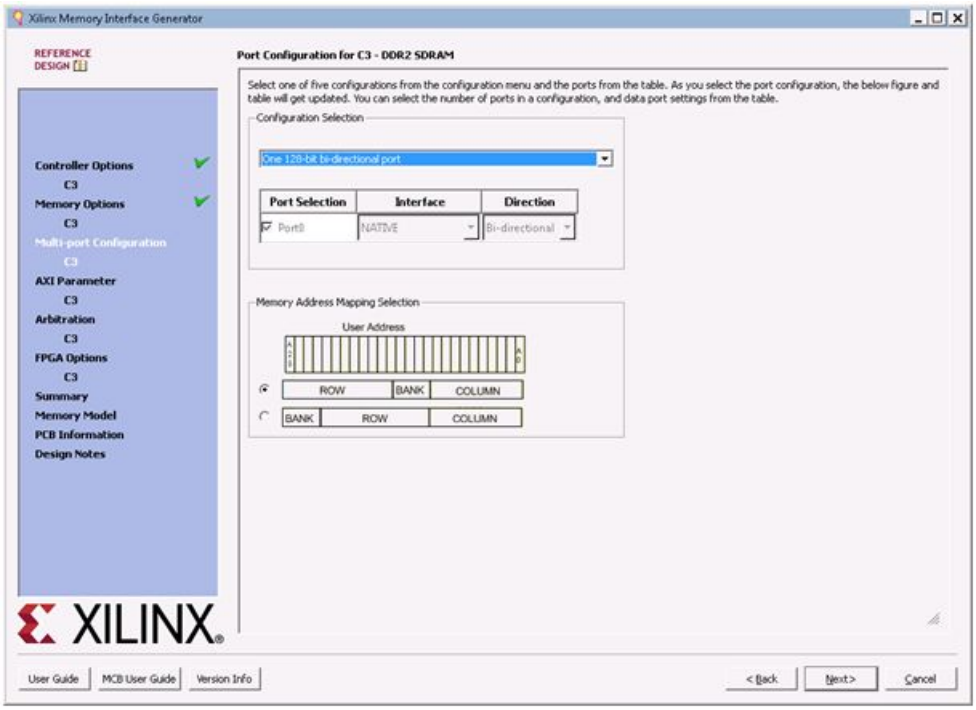
8. [Frequency] を [3200ps] (312.50MHz) に設定します。[Memory Part] を [EDE1116AXXX-8E] に設定します。[Next] をクリックします。



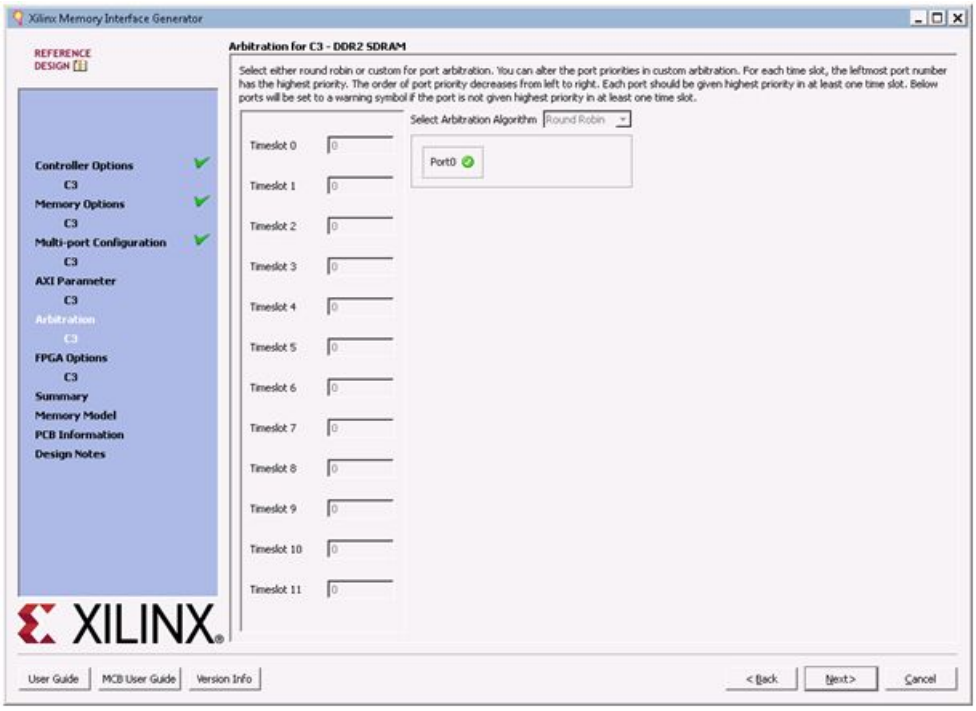
9. [Memory Options] ページの設定はデフォルトのままにします。[Next] をクリックします。



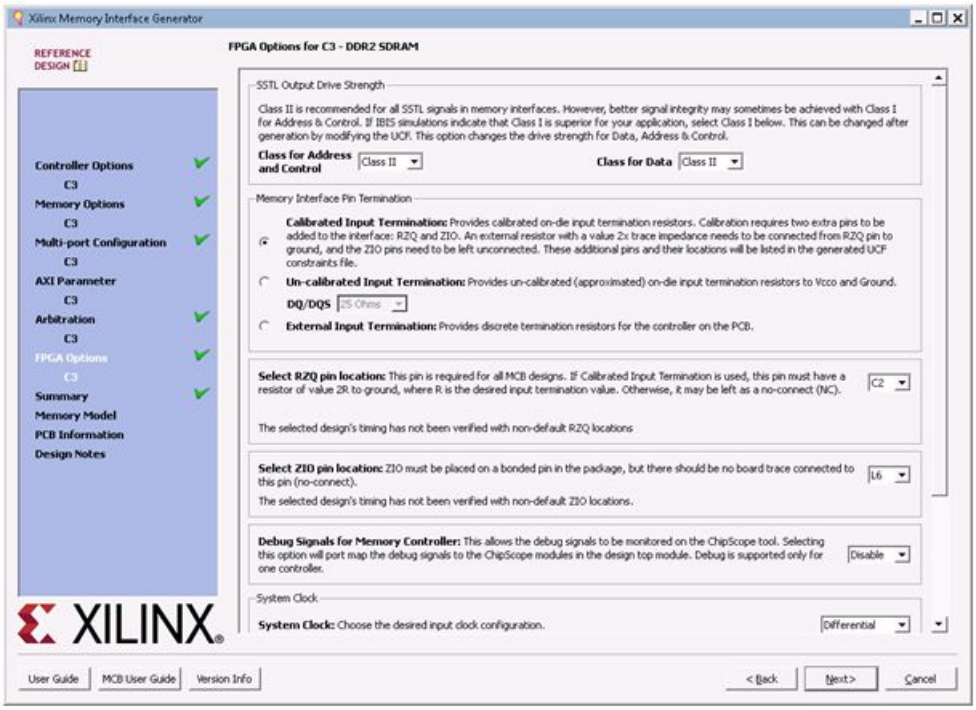
10. [Port Configuration] ページで [One 128-bit bi-directional port] を選択します。[Next] をクリックします。



11. [Arbitration] ページの設定はデフォルトのままにします。[Next] をクリックします。



12. [FPGA Options] ページで [Select RZQ pin location] を [C2] に、[Select ZIO pin location] を [L6] に設定します。[Next] をクリックします。



13. 残りのページで [Next] をクリックし、最後のページで [Generate] をクリックして MIG コアを生成します。

14. MIG コアが生成されたら、生成された `ipcore_dir/mig/users_design/rtl/mig.v` で PLL 設定を変更し、SP601 の 200MHz の差動クロックをシステム クロック

として使用して、MIG コア用の 625MHz クロックが生成されるようにします。mig.v でパラメーター値を次のように変更します。

```
localparam C3_CLKFBOUT_MULT = 25;  // 200 * (25/8) = 625 MHz
localparam C3_DIVCLK_DIVIDE = 8;
```

15. MIG コアをインスタンシエートし、コマンド、読み出し、書き込み FIFO のクロック (c3_p0_cmd_clk、c3_p0_rd_clk、c3_p0_wr_clk) を 1 つのクロック (c3_clk0) に接続する Verilog モジュール mig_dut.v を追加します。このチュートリアルに含まれる完成した mig_dut.v を使用できます。

手順 2 : テストベンチの作成

1. mig_dut インスタンスを駆動する Verilog テストベンチ モジュール mig_hw_tb.v を追加します。このチュートリアルに含まれる完成した mig_hw_tb.v を使用できます。

このチュートリアルでは、ISim Tcl コンソールから MCB および外部メモリにアクセスするためのテストベンチ (mig_hw_tb.v) Tcl コマンド (testmem.tcl) が提供されています。

テストベンチには、外部メモリに書き込むデータを格納する input_data アレイと外部メモリから読み出すデータを格納する output_data アレイが含まれています。MCB では、64 個の 128 ビット ワードまでの書き込み/読み出しバーストがサポートされています。前の手順で、MCB を 128 ビットの双方向ポートを 1 つ使用するようコンフィギュレーションしたので、input_data と output_data のサイズをそれぞれ 64x128 ビットに設定します。テストベンチでは、テストベンチのパラメーターを保存する test_parameters モジュール (params という名前でインスタンシエートされている) も定義されています。次に、これらのパラメーターの使用方法を示します。

テストベンチでは、いくつかの Verilog タスクが定義されています。

- clear_input_output_data : input_data および output_data アレイに 0 を挿入します。
- compare_input_output_data(input nwords) : input_data アレイと output_data アレイのデータの nwords ワードを比較し、不一致をレポートします。
- use_walking_pattern(input b) : input_data アレイに b = 0 の場合はウォーキング ゼロ パターン、b = 1 の場合はウォーキング ワン パターンを挿入します。
- write_data(input start_addr, input burst_size) : input_data アレイから burst_size ワード分のデータを外部メモリのアドレス start_addr から書き込みます。まずデータを MCB 上の書き込み FIFO インターフェイス (c3_p0_wr_*) に送信し、その後コマンド FIFO インターフェイス (c3_p0_cmd_*) に書き込みコマンドを送信します。
- read_data(input start_addr, input burst_size) : 外部メモリのアドレス start_addr から burst_size ワード分のデータを output_data アレイに読み出します。まず PCB 上のコマンド FIFO インターフェイス (c3_p0_cmd_*) に読み出しコマンドを送信し、読み出し FIFO インターフェイス (c3_p0_rd_*) からデータを読み出します。
- test_memory : input_data アレイからのデータを外部メモリの指定領域に書き込み、同じ領域からデータを output_data アレイにリードバックします。メモリ領域は、params.StartAddress および params.EndAddress で指定します。input_data アレイに挿入するデータ パターンは、params.DataPattern で指定します (0 : input_data の現在のデータを使用、1 : ウォーキング ゼロを使用、2 : ウォーキング ワンを使用)。

testmem Tcl コマンドは、params モジュールの StartAddress、EndAddress、および DataPattern を設定し、テストベンチの run_test_trigger 信号をトグルします。run_test_trigger 信号の立ち上がりエッジで test_read_write タスクが呼び出され、外部メモリに対して書き込みおよび読み出しトランザクションが実行され、リードバックと比較して外部メモリにデータが正しく書き込まれたかどうかが確認されます。

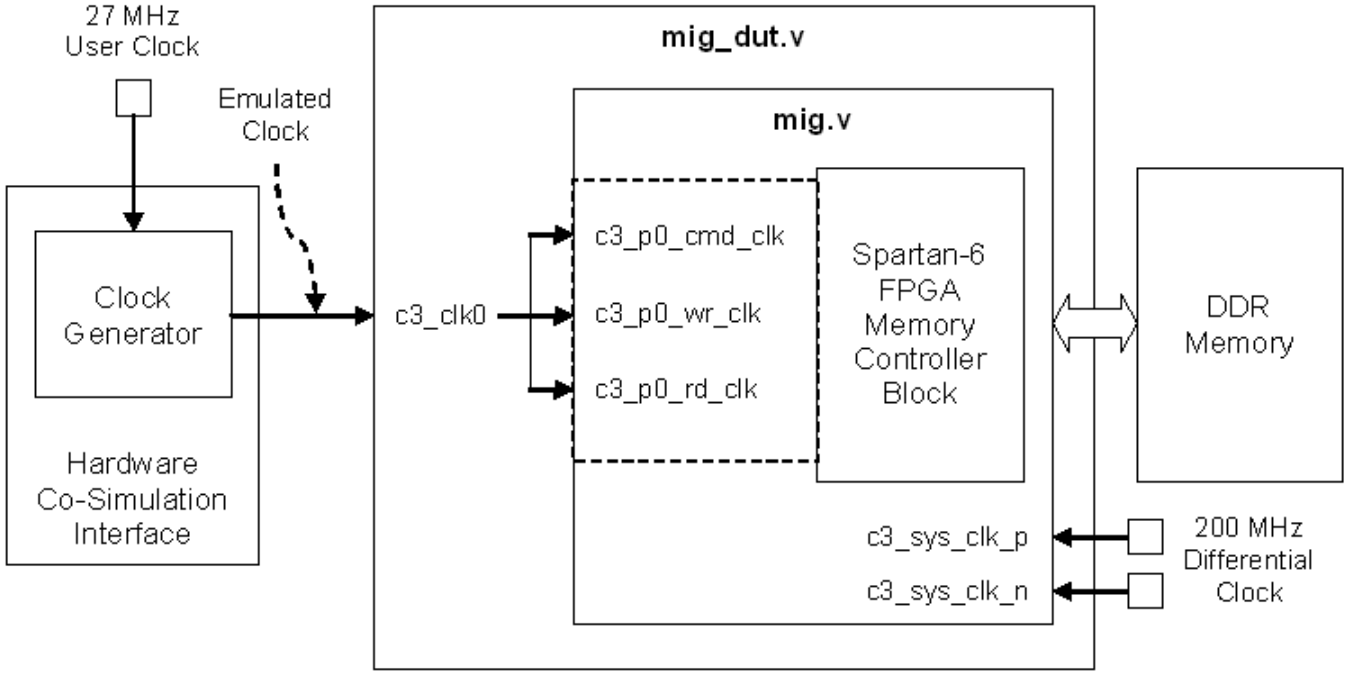
手順 3 : カスタム制約ファイルの作成

デザインをロックステップ部分とフリーランニング部分に分割

このチュートリアルで重要なのは、デザインを次の 2 つの部分に分割するということです。

- Spartan®-6 MCB を介して外部メモリとインターフェイスするフリーランニング部分。外部 I/O とクロックを接続し、メモリ クロックのフル スピードで動作します。
- ISim を使用して HDL テストベンチで駆動するロックステップ部分。ISim シミュレーションに同期化されており、ハードウェア協調シミュレーション インターフェイスからスティミュラスおよびクロック イベントを受信します。そのため、動作速度はかなり遅くなります。

次の図に、ハードウェア協調シミュレーションでの MIG デザインへのクロック供給方法を示します。ハードウェア協調シミュレーション インターフェイスは、コンパイル中に自動的に挿入されます。ハードウェア協調シミュレーション インターフェイスは、SP601 ボード上の 27MHz のユーザー クロックに基づいてエミュレート クロックを生成します。エミュレート クロックは、テストベンチの c3_clk0 信号のクロック イベントに対応し、ハードウェアで動作している mig_dut の c3_clk0 ポートを駆動します。MIG コアのシステム クロックは、SP601 ボード上の 200MHz 差動クロックから生成されます。



ポートを外部 I/O およびクロックに割り当て

カスタム制約ファイル (ザイリンクスの UCF フォーマット) を使用して、ハードウェア協調シミュレーションでどのポートを FPGA IOB にマップするか、HDL テストベンチでどのポートを制御するかを指定できます。ISim コンパイラにより、UCF ファイルに含まれる LOC 制約が検索されます。LOC 制約が設定されているポートは、対応する FPGA IOB にマップされます。LOC 制約が設定されていないポートは、ハードウェア協調シミュレーション インターフェイスにマップされ、HDL テストベンチでアクセスできます。

デザインのフリーランニング部分とロックステップ部分への分割は、クロック ポートのマップ方法により決定されます。LOC 制約を使用してクロック ポートを FPGA IOB にマップすると、このクロックで駆動されるロジックはフリーランニング部分に含まれます。クロック ポートに LOC 制約が設定されていない場合、テストベンチで対応するクロック イベントが発生したときに、ハードウェア協調シミュレーション インターフェイスによりこのポートの値がトグルされます。このクロックで駆動されるロジックは、ロックステップ部分に含まれます。

フリーランニング部分とロックステップ部分は異なるクロックを使用して異なる速度で動作するので、この 2 つの部分の間でのクロック ドメインの切り替わりをデザインで適切に処理する必要があります。ISim ハードウェア協調シミュレーションのコンパイルではデザインの内部は変更されないので、2 つの部分の間の速度差と同期化をデザインで適切に処理できることを前提としています。

次の表に、mig_dut モジュールの外部 I/O にマップされるポートと、テストベンチで制御されるポートを示します。

mig_dut モジュールのポートの分割

外部 I/O にマップされるポート	テストベンチで制御されるポート
c3_sys_clk_p c3_sys_clk_n mcb3_dram_dq mcb3_dram_a mcb3_dram_ba mcb3_dram_ras_n mcb3_dram_cas_n mcb3_dram_we_n mcb3_dram_odt mcb3_dram_cke mcb3_dram_ck mcb3_dram_ck_n mcb3_dram_dqs mcb3_dram_dqs_n mcb3_dram_udqs mcb3_dram_udqs_n mcb3_dram_udm mcb3_dram_dm rzq3 zio3	c3_sys_rst_n c3_clk0 c3_rst0 c3_calib_done c3_p0_cmd_en c3_p0_cmd_instr c3_p0_cmd_bl c3_p0_cmd_byte_addr c3_p0_cmd_empty c3_p0_cmd_full c3_p0_wr_en c3_p0_wr_mask c3_p0_wr_data c3_p0_wr_full c3_p0_wr_empty c3_p0_wr_count c3_p0_wr_underrun c3_p0_wr_error c3_p0_rd_en c3_p0_rd_data c3_p0_rd_full c3_p0_rd_empty c3_p0_rd_count c3_p0_rd_overflow c3_p0_rd_error

MIG ツールにより、サンプル UCF ファイル ipcore_dir/mig/user_design/par/mig.ucf が生成されます。このチュートリアルでは、これをテンプレートとして使用して、ハードウェア協調シミュレーション用のカスタム制約ファイルを作成します。

1. ipcore_dir/mig/user_design/par/mig.ucf を mig_dut.v が含まれる ISim プロジェクト ディレクトリにコピーします。コピーしたファイルの名前を mig_dut_hwcosim.ucf に変更します。
2. mig_dut_hwcosim.ucf ファイルを次のように SP601 ボード用に変更します。TS_SYS_CLK3 の PERIOD 制約を 5ns に変更します。このチュートリアルでは、SP601 の 200MHz の差動クロックをシステム クロックとして使用します。

   ```
   TIMESPEC "TS_SYS_CLK3" = PERIOD "SYS_CLK3" 5 ns HIGH 50 %;
   ```

   c3_sys_clk_n の LOC 制約を K16 に、c3_sys_clk_p の LOC 制約を K15 に変更し、SP601 のピン割り当てに一致させます。

   ```
   NET  "c3_sys_clk_n"  LOC = "K16";
   NET  "c3_sys_clk_p"  LOC = "K15";
   ```

3. ISim ハードウェア協調シミュレーションの要件に応じて mig_dut_hwcosim.ucf ファイルを変更します。

   次の制約で階層パスの最初にワイルドカード文字「*」を追加します。これは、サブモジュールがハードウェア協調シミュレーション用にコンパイルされる際に mig_dut がラッパーに含まれるからです。

```
NET "*memc?_wrapper_inst/mcb_ui_top_inst/mcb_raw_wrapper_inst/selfrefresh_mcb_mode" TIG;
NET "*c?_pll_lock" TIG;
NET "*memc?_wrapper_inst/mcb_ui_top_inst/mcb_raw_wrapper_inst/gen_term_calib.mcb_soft_calibration_top_inst/
  mcb_soft_calibration_inst/CKE_Train" TIG;                              ##This path exists for DDR2
NET "*memc3_infrastructure_inst/sys_clk_ibufg" TNM_NET = "SYS_CLK3";
```

error、calib_done、および c3_sys_rst_n の制約 (特に LOC) をコメントアウトします。これらはテストベンチで制御されます。

```
#NET  "error"          IOSTANDARD = LVCMOS18 ;
#NET  "calib_done"     IOSTANDARD = LVCMOS18 ;
#NET  "calib_done"     LOC = "B2" ;
#NET  "error"          LOC = "A2" ;

#NET  "c3_sys_rst_n"   IOSTANDARD = LVCMOS18;
#NET  "c3_sys_rst_n"   LOC = "M8" ;
```

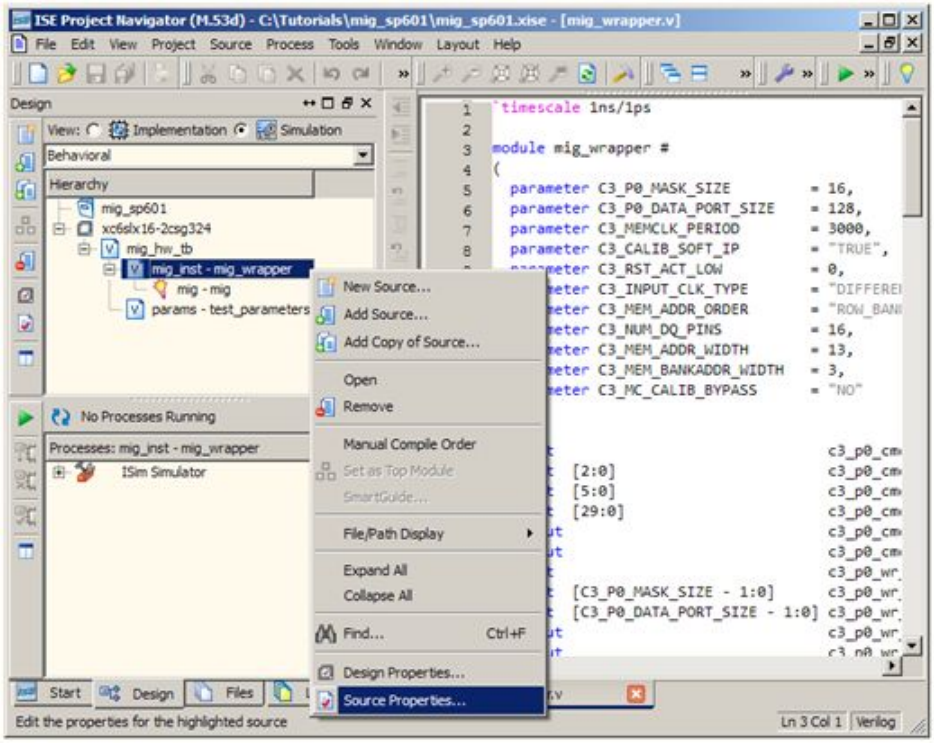
メモ : mig_dut 上の c3_clk0、c3_rst0、c3_calib_done、c3_p0_* ポートもテストベンチで制御されるので、制約されていません。

手順 4 : ハードウェア協調シミュレーション用のデザインのコンパイル

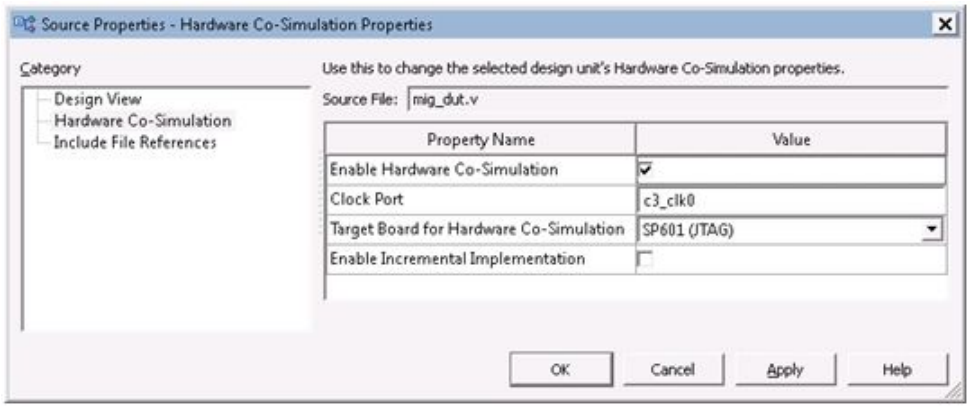
テストベンチとカスタム制約ファイルを作成したら、ISim コンパイラを使用して、デザインをハードウェア協調シミュレーション用にコンパイルします。これは、Project Navigator でデザインの選択したインスタンスでハードウェア協調シミュレーションをイネーブルにすると実行できます。選択したインスタンスとそれに含まれるサブモジュールは、ISim シミュレーション時にハード

ウェアで協調シミュレーションされます。その他のモジュールは、ソフトウェアでシミュレーションされます。

1. Project Navigator の [View] ペインで [Simulation] をオンにします。[Hierarchy] ペインで [mig_inst - mig_dut] インスタンスを右クリックし、[Source Properties] をクリックします。



2. [Category] で [Hardware Co-Simulation] を選択します。[Enable Hardware Co-Simulation] チェック ボックスをオンにします。[Clock Port] を [c3_clk0] に設定します。[Target Board for Hardware Co-Simulation] を [SP601 (JTAG)] に設定します。

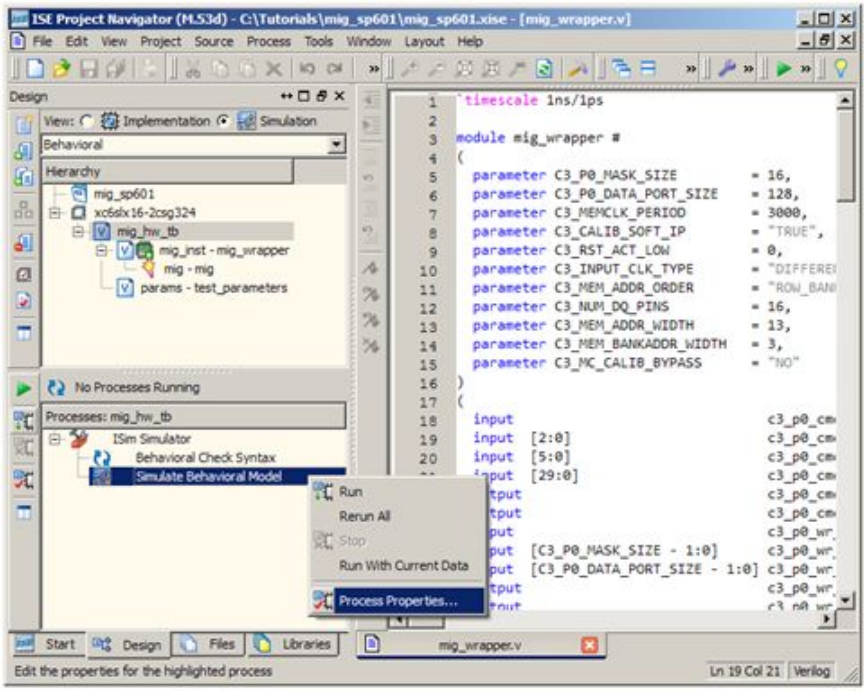


メモ : ハードウェア協調シミュレーションをイネーブルにしたインスタンスには、 アイコンが付きます。

デザインをハードウェア協調シミュレーション用に一度コンパイルしたら、[Enable Incremental Implementation] を使用できます。ハードウェア協調シミュレーション用に選択されたインスタンスがその後の実行で変更されない場合、このオプションをオンにすると、ハードウェア協調シミュレーション用の合成、インプリメンテーション、ビットストリーム生成がスキップ

されます。このオプションを使用すると、ソフトウェアでシミュレーションする部分をすばやく変更し、再シミュレーションできます。

3. [Hierarchy] ペインで [mig_hw_tb] インスタンスを選択します。[Processes] ペインで [Simulate Behavioral Model] を右クリックし、[Process Properties] をクリックします。



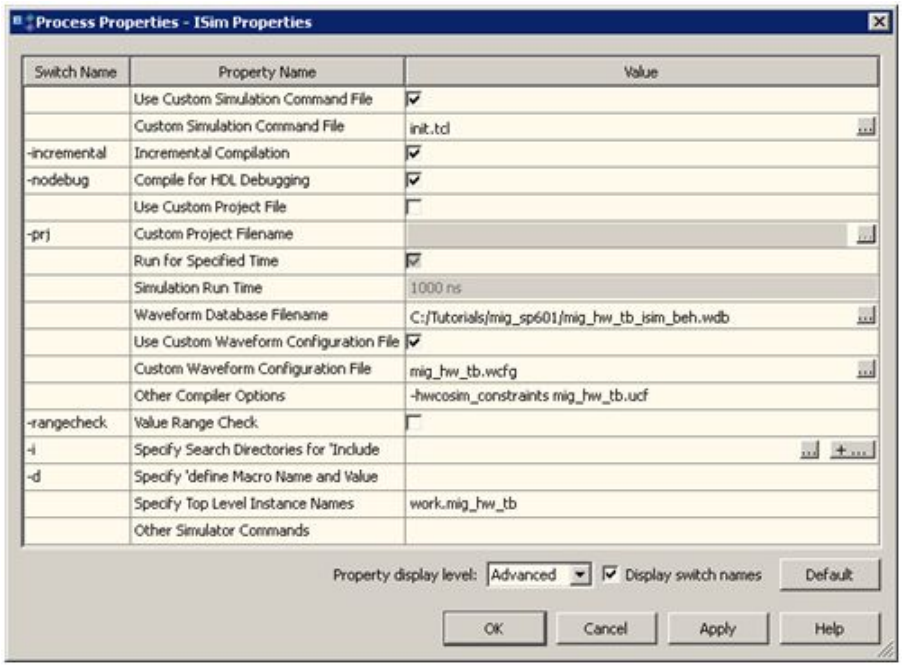
4. [Process Properties] ダイアログ ボックスで [Property display level] を [Advanced] に変更します。[Simulate Behavioral Model] プロセスの次のプロパティを設定します。

   - [Use Custom Simulation Command File] : オン
   - [Custom Simulation Command File] : init.tcl
   - [Use Custom Waveform Configuration] : オン
   - [Custom Waveform Configuration File] : mig_hw_tb.wcfg
   - [Other Compiler Options] : -hwcosim_constraints mig_dut_hwcosim.ucf

   ISim シミュレーションを開始すると、init.tcl スクリプトが実行されます。このスクリプトは、testmem.tcl スクリプトから Tcl コマンド testmem を読み込みます。testmem.tcl は、このチュートリアルの後の方でシミュレーションを実行するのに使用されます。

   mig_hw_tc.wcfg ファイルは、このチュートリアル用にカスタマイズされた波形コンフィギュレーション ビューを提供します。

   メモ : ハードウェア協調シミュレーション用のカスタム制約ファイルは、-hwcosim_constraints オプションで指定します。このオプションは、現在のところ Project Navigator GUI にプロパティとして含まれていないので、[Other Compiler Options] で指定します。



- mig_hw_tb インスタンスの [Simulate Behavioral Model] プロセスをダブルクリックしてシミュレーションを実行します。

Fuse コマンド ライン ツールの使用

ISim コンパイラは、Fuse コマンド ライン ツールで起動できます。完全なソフトウェア シミュレーション フローと同様に、プロジェクト ファイル、デザインの最上位モジュール、およびリンクするライブラリやライブラリ検索パスなどの引数を指定して、Fuse を実行します。ハードウェア協調シミュレーション用にデザインをコンパイルするには、次に示す引数を指定する必要があります。

```
fuse -prj <project file> <top level modules>
     -hwcosim_instance <instance>
     -hwcosim_clock <clock>
     -hwcosim_board <board>
     -hwcosim_constraints <constraint file>
     -hwcosim_incremental <0|1>
```

- -hwcosim_instance : ハードウェアで協調シミュレーションするためにインスタンスの完全階層パスを指定します。
- -hwcosim_clock : インスタンスのクロック入力のポート名を指定します。
  - これはロックステップ部分のクロックで、テストベンチで制御されます。
  - 複数のクロックを使用するデザインでは、このオプションで最高速のクロックを指定し、ISim でシミュレーションが最適化されるようにします。その他のクロック ポートは、通常のデータ ポートとして処理されます。
- -hwcosim_board : 協調シミュレーションに使用するハードウェア ボードを指定します。デフォルトでは、次の 2 つの Spartan®-6 ボードがサポートされています。
  - sp601-jtag : ザイリンクス SP601 評価プラットフォーム
  - sp605-jtag : ザイリンクス SP605 評価プラットフォーム
- -hwcosim_constraints (オプション) : ハードウェア協調シミュレーション用にインスタンスをインプリメントするための追加制約を含むカスタム制約ファイルを指定します。この制約ファイルでは、インスタンスのどのポートを外部 I/O またはクロックにマップするかも指定します。
- -hwcosim_incremental (オプション) : Fuse で前回生成されたハードウェア協調シミュレーション ビットストリームを再利用し、インプリメンテーション フローをスキップするよう指定します。

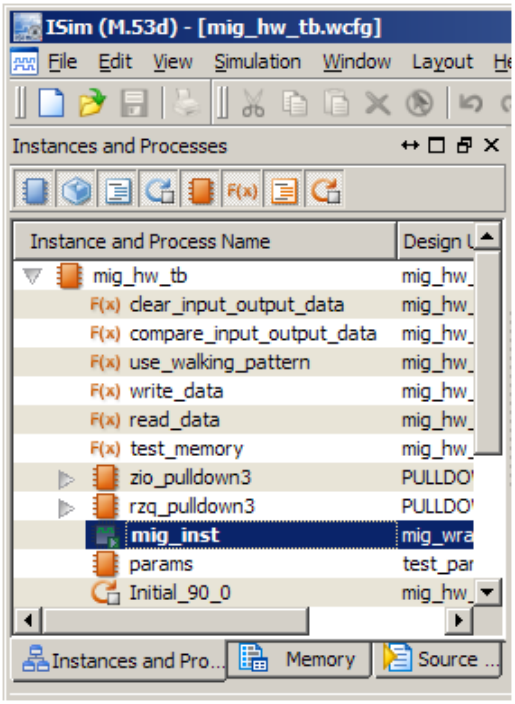
たとえば、このチュートリアルの EMAC デザインをコンパイルするには、次のようにコマンド ラインに入力して Fuse を実行できます。

```
fuse -prj mig_hw_tb.prj mig_hw_tb glbl
     -L unisims_ver -L secureip
     -o mig_hw_tb.exe
     -hwcosim_instance /mig_hw_tb/mig_inst
     -hwcosim_clock c3_clk0
     -hwcosim_board sp601-jtag
     -hwcosim_constraints mig_dut_hwcosim.ucf
```

手順 5 : ISim ハードウェア協調シミュレーションの実行

ISim コンパイラで生成されるシミュレーション実行ファイルは、完全なソフトウェア シミュレーションおよびハードウェア協調シミュレーション フローの両方で同様に使用できます。コンパイルが終了すると、Project Navigator によりシミュレーション実行ファイルが GUI モードで実行されます。

ハードウェア協調シミュレーションに選択されたインスタンスは、[Instances and Processes] パネルで アイコンで示されます。ハードウェアでインスタンスを実行すると、その内部信号およびサブモジュールをモニターすることはできません。

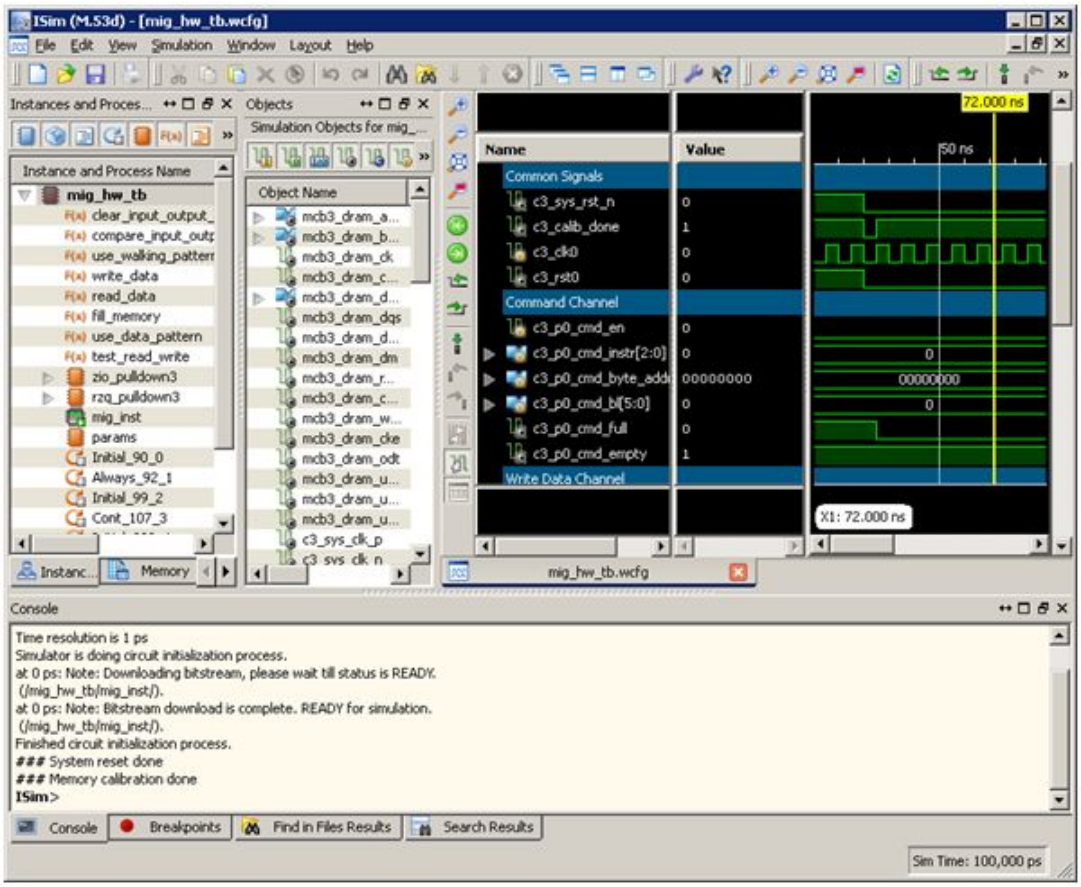


シミュレーションを開始する前に、ハードウェア協調シミュレーション用に生成されたビットストリームで FPGA がプログラムされます。ISim の [Console] パネルに、ビットストリームをダウンロード中であることを示すメッセージ「Downloading bitstream, please wait till status is READY」が表示されます。FPGA がコンフィギュレーションされると、ビットストリームのダウンロードが完了し、シミュレーションの準備ができたことを示すメッセージ「Bitstream download is complete. READY for simulation」が表示されます。この時点で、ソフトウェア シミュレーション フローと同様に、ISim GUI でシミュレーションを実行できます。

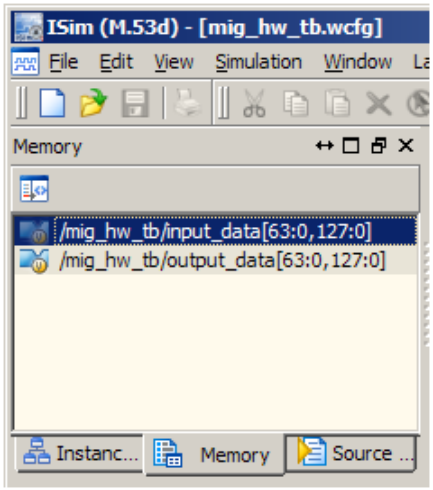
まずテストベンチにより c3_sys_rst_n 信号がアサートされてシステムがリセットされ、メモリ キャリブレーション プロセスが開始します。ISim の [Console] パネルに次のようなメッセージが表示されます。

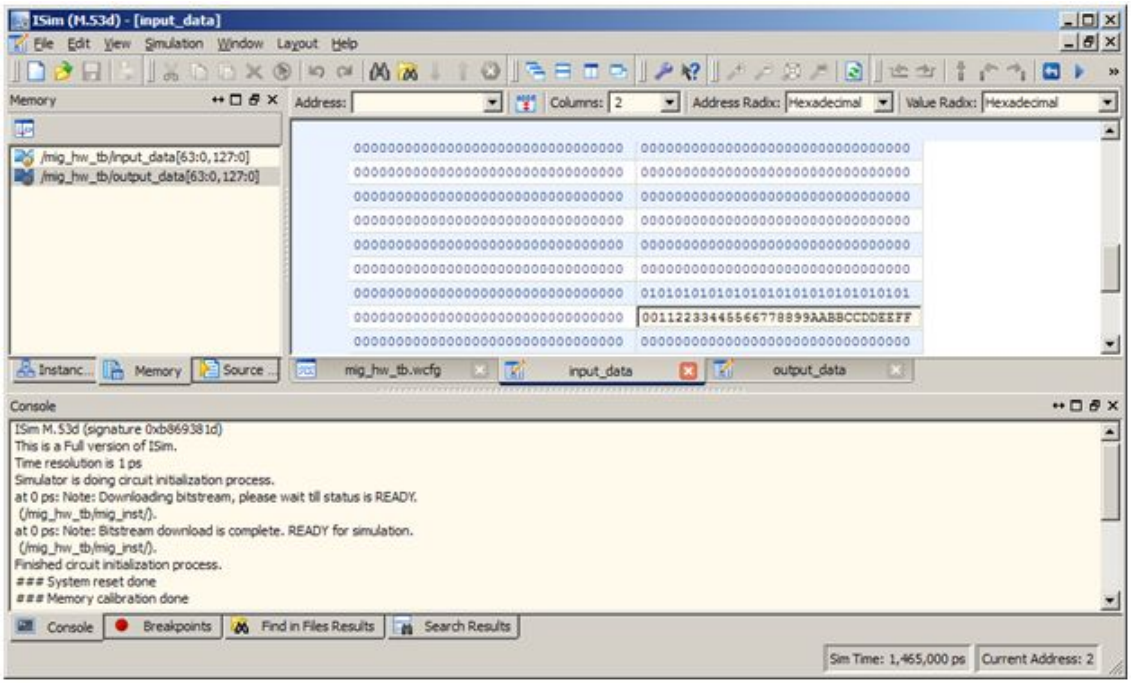
```
### System reset done
### Memory calibration done
```

reset 信号がディアサートされた直後に、c3_calib_done 信号が Low から High に遷移します。これは、メモリ キャリブレーション プロセスがハードウェアでフル スピードで実行されるからです。キャリブレーション プロセスをソフトウェアでシミュレーションすると、もっと時間がかかります。



[Memory] パネルをクリックします。input_data アレイをダブルクリックしてメモリ エディターを開きます。[Address Radix] および [Value Radix] を [Hexadecimal] に変更します。input_data の内容を編集し、それを使用して外部メモリに書き込むことができます。

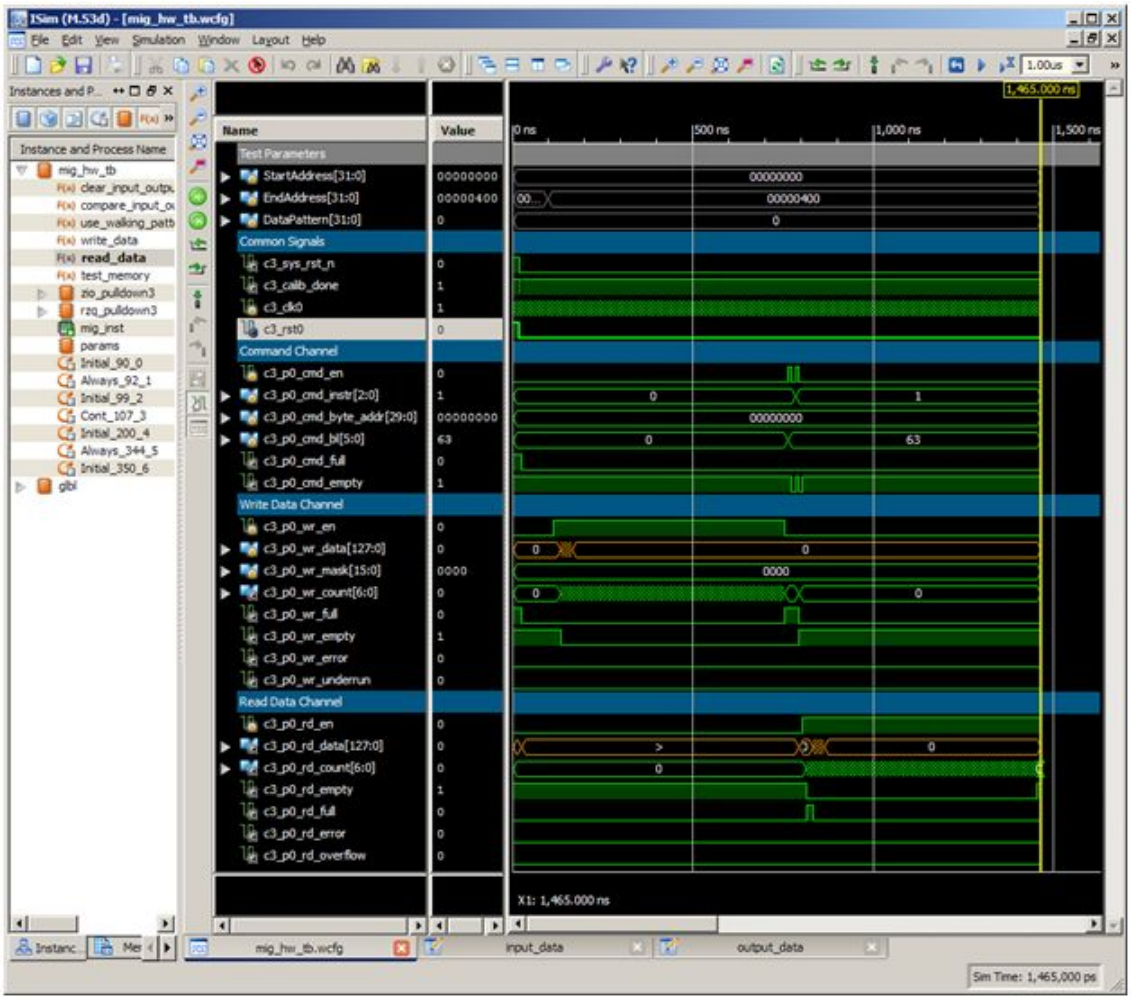




ISim の [Console] パネルから「testmem 0 1024 0」を実行します。これによりテストベンチの test_memory タスクが実行され、input_data のデータが外部メモリのアドレス 0 ～ 1024 に書き込まれ、同じメモリ領域のデータが output_data にリードバックされます。ISim の [Console] パネルに次のようなメッセージが表示されます。

```
### Read/write test started
... Writing 00000000
... Reading 00000000
==> No data mismatches found.
### Read/write test done
```

c3_p0_cmd_*、c3_p0_wr_*、および c3_p0_rd_* の波形を観察し、メモリの書き込み/読み出しトランザクションでのテストベンチと MCB 間の関係、MCB のステータス信号 (c3_p0_wr_count など) の変化を確認します。



testmem コマンドを使用して、異なるデータ パターン、開始アドレスおよび終了アドレスを試します。次に例を示します。

- testmem 0 2048 1 :アドレス 0 ～ 2048 のメモリ領域にウォーキング ゼロ パターンを使用します。
- testmem 1024 4096 2 :アドレス 1024 ～ 4096 のメモリ領域にウォーキング ワン パターンを使用します。

XILINX®

付録

その他のリソース

- **用語集：** http://japan.xilinx.com/support/documentation/sw_manuals/glossary.pdf
- **ザイリンクス マニュアル：** http://japan.xilinx.com/support/documentation
- **ザイリンクス サポート：** http://japan.xilinx.com/support