

## はじめに

DDR/DDR2 SDRAM コントローラ MegaCore® ファンクション v3.2.0 には、SOPC Builder のサポートが含まれています。これにより、SOPC Builder システムで DDR または DDR2 SDRAM コントローラをインスタンス化することができます。

このアプリケーション・ノートでは、SOPC Builder で DDR2 SDRAM コントローラを実装するための手順を説明しています。詳細な手順に基づいて、デザインのコンパイル、デザインのボードへのダウンロード、Nios II プロセッサの組み込み、ハードウェアによるデザイン上でのサンプル・コードの実行を行うことができます。



これらの手順では、Cyclone II デバイス・ファミリをターゲットとするデザインが作成されます。また、これらの手順をサポートされる他のデバイス・ファミリにも使用できます。

## 必要条件

このアプリケーション・ノートでは、以下のハードウェアとソフトウェアが要求されます。

- DDR または DDR2 SDRAM メモリを搭載したアルテラの開発ボード
- DDR/DDR2 SDRAM コントローラ MegaCore ファンクション v3.2.0 以降
- Quartus II ソフトウェア v5.0 以降

このプロジェクトでは、Cyclone II DSP 開発ボードを使用します。



詳細については、「DDR & DDR2 SDRAM Controller Compiler User Guide」を参照してください。

## プロジェクト の設定

### 新規 Quartus II プロジェクトの作成

作業を開始する前に、新規 Quartus II プロジェクトを作成する必要があります。新規プロジェクトを作成するには、以下のステップに従います。



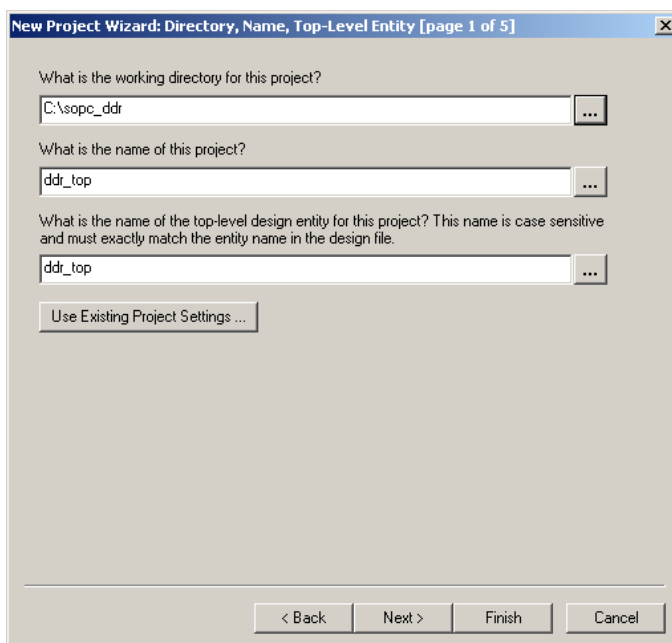
これらの手順は、Windows ベースのシステムを対象としたものです。Solaris および Linux ベースのシステムの場合、ステップは似ていますが、詳細については、「DDR & DDR2 SDRAM Controller Compiler User Guide」を参照してください。

1. Windows の [ スタート ] メニューから、[ プログラム ]、[ Altera ] を順にポイントし、**Quartus II <version>**をクリックして、Quartus IIソフトウェアを実行します。
2. File メニューで、**New Project Wizard** をクリックします。
3. イントロダクション（イントロダクションを以前にオフにしている場合は、表示されません）で、**Next** をクリックします。
4. プロジェクトの作業ディレクトリを指定します。この演習では、ディレクトリ **c:\sopc\_ddr** を使用します。
5. プロジェクトの名前を指定します。この演習では、**ddr\_top** を使用します (図 1)。




プロジェクトとトップ・レベルのデザイン・エンティティには、同じ名前を指定する必要があります。

図 1. New Project Wizard ダイアログ・ボックス

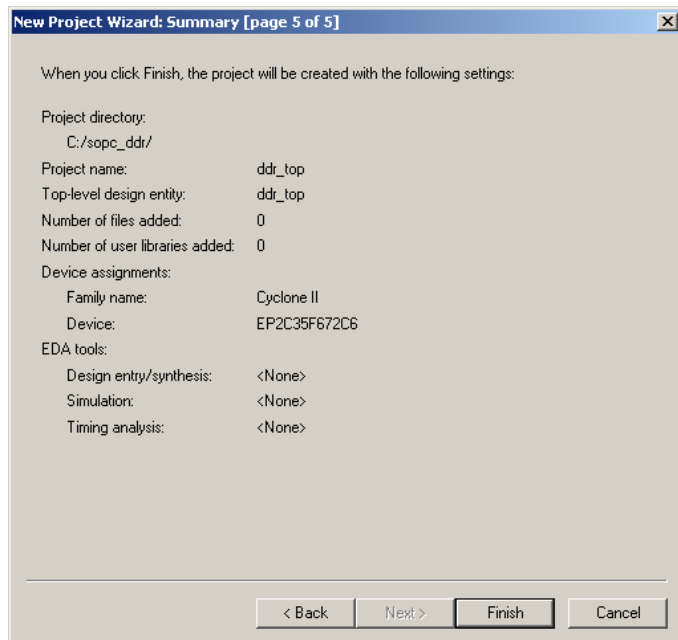


6. **Next** をクリックします。
7. **Next** をクリックします。
8. **Family** リストで、**Cyclone II** を選択します。

 以下の手順では、Cyclone II デバイス・ファミリをターゲットとするデザインが作成されます。サポートされる他のデバイス・ファミリにも、以下の手順を使用できます。

9. **Target device** ボックスで、**Specific device selected in 'Available devices' list** を選択します。
10. **Filters** 内のすべてのフィールドに、デフォルト値 **Any** が表示されるはずですが。
11. **Available Devices** リストで、**EP2C35F672C6** を選択します。
12. **Next** をクリックします。
13. **Next** をクリックします。
14. Summary ページで、すべての情報を正しく入力していることを確認します(図2)。

図 2. Summary ページ



15. **Finish** をクリックします。

新しい Quartus II プロジェクトの作成が終了しました。

## SOPC Builder システムの 作成

この項では、1つの SOPC Builder システムを作成するのに必要なコンポーネントを追加します。

### SOPC Builder の起動

SOPC Builder から IP Toolbench を起動するには、以下のステップに従います。

1. Tools メニューで、**SOPC Builder** をクリックします。

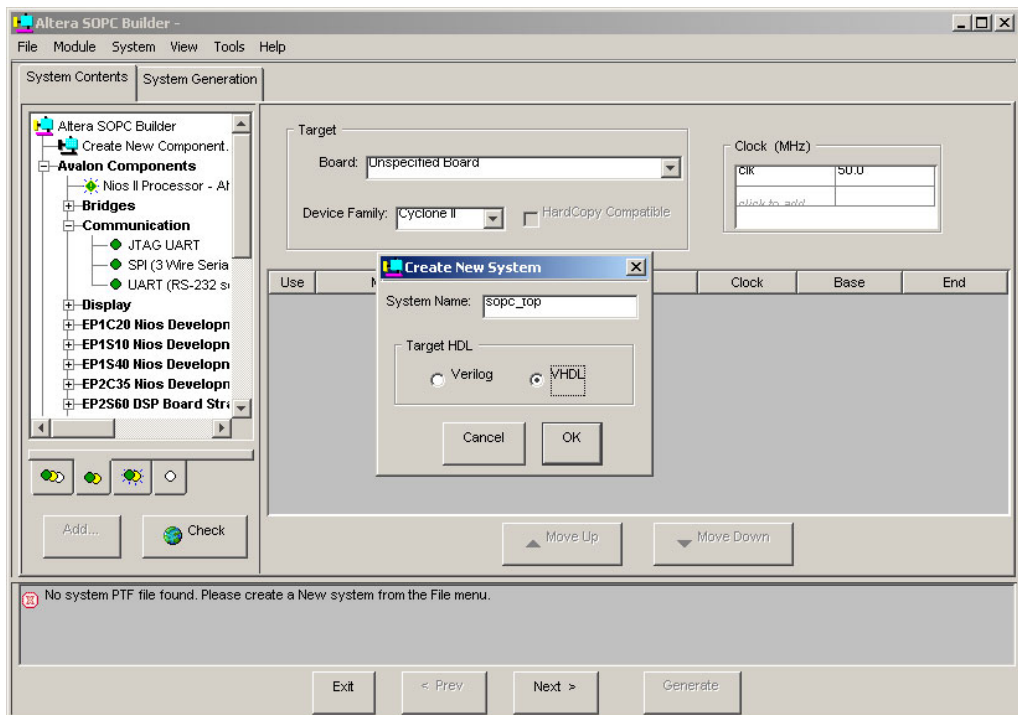
**Altera SOPC Builder** ダイアログ・ボックスが表示されます。



SOPC Builder の使用方法の詳細は、**Quartus II Help** を参照してください。

2. **Create New System** ダイアログ・ボックスの **System Name** ボックスに `sopc_top` を入力し、**Target HDL** で **VHDL** を選択します (図 3)。
3. **OK** をクリックします。

図 3. SOPC Builder ダイアログ・ボックス

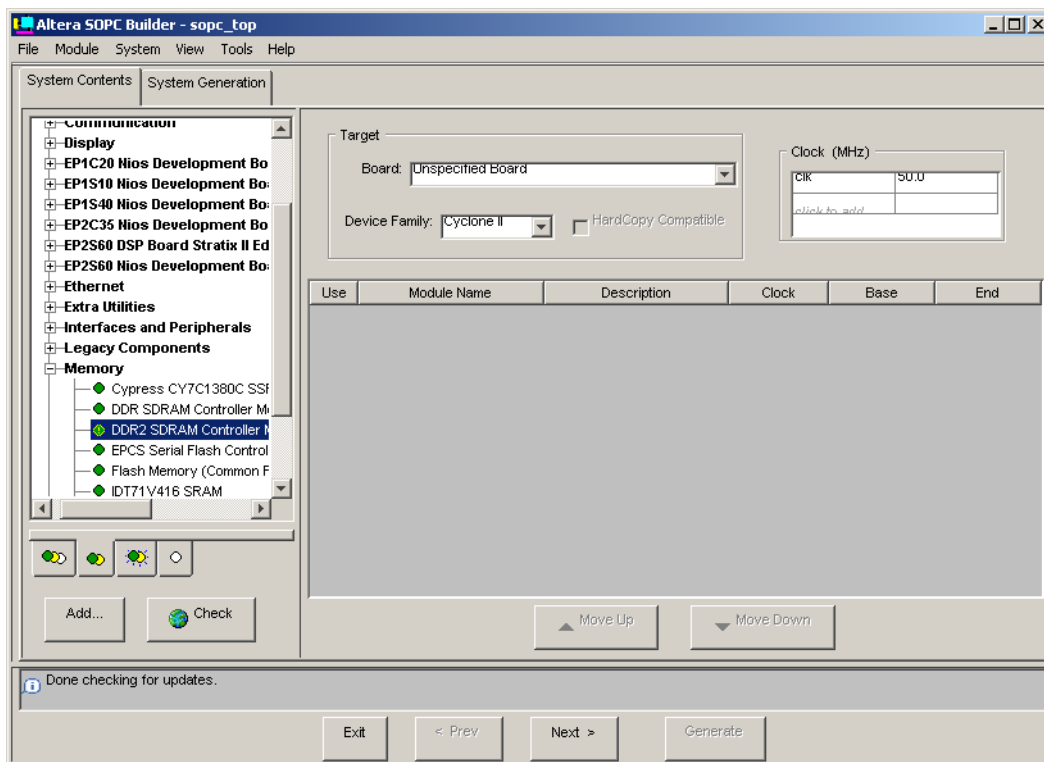


## SOPC Builder システムへの DDR2 SDRAM Controller の追加

1. System Contents タブでコンポーネントを追加して、システムを構築します。

Memory ディレクトリで、**DDR2 SDRAM Controller MegaCore Function – Altera Corporation** を選択します (図 4)。

図 4. SOPC Builder での DDR2 SDRAM Controller の選択



2. **Add** をクリックします。DDR2 SDRAM Controller IP Toolbench が表示されます (図 5)。

図 5. DDR2 SDRAM Controller IP Toolbench



### Step1: Parameterize

DDR2 SDRAMコントローラをパラメータ化するには、以下のステップに従います。

1. **Step 1: Parameterize** をクリックして、カスタム・バリエーションをパラメータ化します。Parameterize ダイアログ・ボックスが表示されます。
2. Presets ボックスで、**Altera Cyclone II EP2C35 DSP Development Board** を選択します。



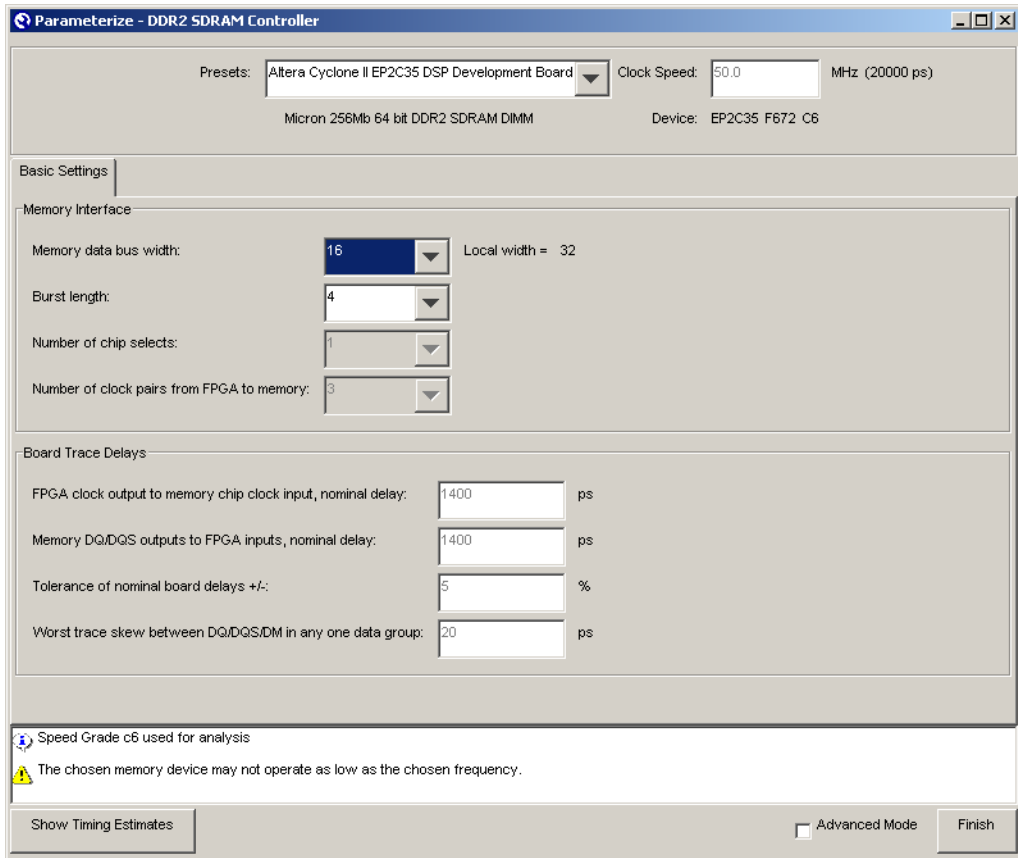
アルテラのボードをターゲットにする場合、Basic Settings タブの設定および **Advanced Mode** の設定はすべてそのままボードに適用できます。

3. **Memory data bus width** を 16 (ローカル幅は自動的に 32 に設定されます) に設定します (図 6)。



ローカル幅は、32 ビット Nios II プロセッサに合わせて 32 ビットに設定されます。

図 6. Parameterize ダイアログ・ボックス



4. **Advanced Mode** をオンにします。各種のタブで利用できるオプションを探します。
5. **Advanced Mode** をオフにします。
6. **Finish** をクリックします。

## Step2: Constraints


デバイスの制約を選択するには、以下のステップに従います。

1. IP Toolbench で **Step 2: Constraints** をクリックします (図 7)。

図 7. IP Toolbench - Constrains

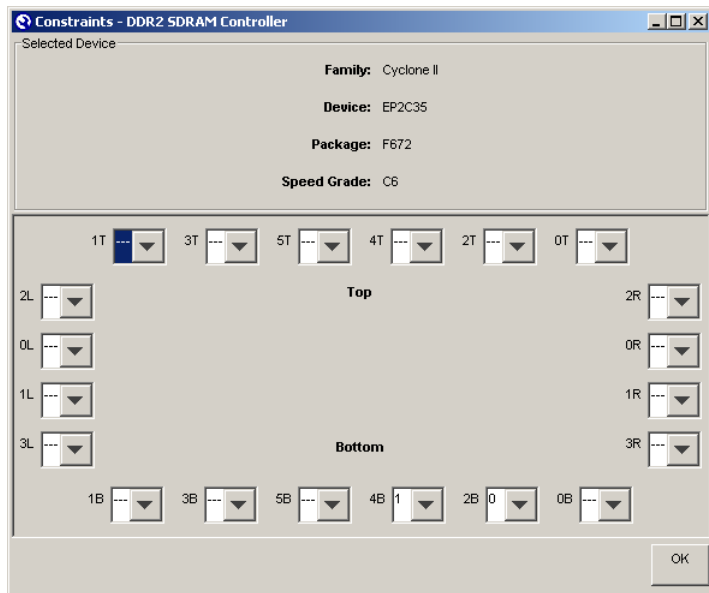


Constraints ダイアログ・ボックスが表示されます (図 8)。

 アルテラのボードをターゲットにする場合、すべての制約の設定はすでにそのボードに適合しています。

2. **OK** をクリックします。

図 8. Constraints ダイアログ・ボックス



### Step3: Add/Update Component

コンポーネントを追加または更新してシステムを生成するには、以下のステップに従います。

1. IP Toolbench で **Step 3: Add/Update Component** をクリックして、SOPC Builder にカスタム・バリエーションを追加します (図 9)。

図 9. IP Toolbench - Add/Update Component



2. IP Toolbench が閉じ、SOPC Builder は使用中の DDR2 SDRAM コントローラのバリエーション名に、デフォルトのモジュール名 (`ddr2_sdram_0`) を使用します。

## SOPC Builder システムの終了

SOPC Builder システムを終了するには、以下のステップを実行します。

1. 以下のコンポーネントを追加します。
  - **Avalon Components** ディレクトリの **Nios II Processor – Altera Corporation**
  - **Memory** ディレクトリの **On-Chip Memory (RAM or ROM)**
  - **Other** ディレクトリの **PIO (Parallel I/O)**
  - **Communication** ディレクトリの **JTAG UART**
2. オンチップ・メモリのサイズを大きくします。
  - a. **onchip\_memory\_0** モジュールをダブル・クリックします。
  - b. On-chip Memory ダイアログ・ボックスで、Total Memory Size を 32 KB に設定します。
  - c. **Finish** をクリックします。

- d. これまで、メモリ・サイズを 32 KB に増やすと、メモリに対して生成されるベース・アドレスにエラーが発生していました。

これを解決するには、System メニューで **Auto-Assign Base Addresses** をクリックします。

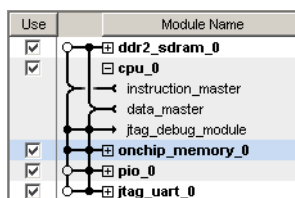
3. バス・リンクを設定します (図 10 を参照)。

- a. **ddr2\_sdram\_0** モジュールと **cpu\_0/instruction\_master** モジュール間のリンクを切断します。

この例では、DDR2 SDRAM メモリはデータ専用で、命令コードは含まれません。

- b. **ddr2\_sdram\_0** モジュールと **cpu\_0/data\_master** モジュール間にリンクを作成します。

図 10. バス・リンクの設定



4. クロック周波数を 125 MHz にリセットします。

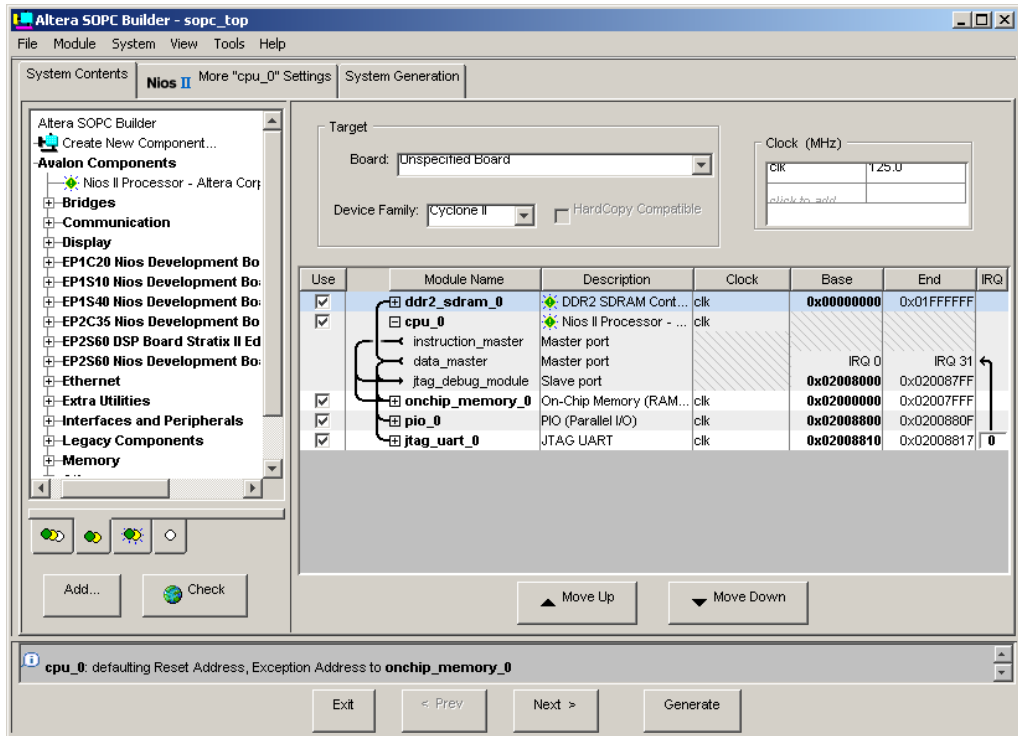
- a. SOPC Builder で、クロック周波数を 125 MHz に変更します。

125 MHz を使用するのには、DDR2 DRAM メモリの  $f_{MIN}$  が 125 MHz であるためです (SOPC Builder のデフォルトは 50 MHz です)。

- b. **ddr2\_sdram\_0** モジュールをダブル・クリックします。IP Toolbench が起動します。
- c. **Step 1: Parameterize** をクリックします。
- d. Parameterize ウィンドウでは、クロック速度は自動的に 125 MHz に設定されています。 **Finish** をクリックします。
- e. IP Toolbench で、 **Step 3: Add/Update Component** をクリックします。

図 11 に、完成した SOPC Builder システムを示します。

図 11. SOPC Builder システム



## システムの生成

システムを生成するには、以下のステップを実行します。

1. SOPC Builder で、**Next** をクリックします。
2. **Next** をクリックします。
3. **Simulation. Create simulator project files** をオフにします。
4. **Generate** をクリックします。

生成が完了すると、次のメッセージが表示されます。

SUCCESS: SYSTEM GENERATION COMPLETED.

5. **Exit** をクリックします。

## トップレベル・ファイルの作成

システムを生成すると以下の 2 つのファイルが作成され、作業ディレクトリ `c:\sopc_ddr` に置かれます。

- **ddr2\_sdrām\_0\_debug\_design.vhd** - このトップレベルのファイルには、SOPC Builder コンポーネントは格納されません。
- **sopc\_top.vhd** - このファイルには、SOPC Builder システムのコンポーネント宣言が格納されます。

次のステップでは、PLL と SOPC Builder コンポーネントを格納するトップレベル・ファイルを作成します。この場合、例のドライバと **ddr2\_sdrām\_0\_debug\_design.vhd** の DDR/DDR2 SDRAM コントローラを、**sopc\_top.vhd** の SOPC Builder コンポーネントに置き換えて、2 つのファイルをマージします。図 12 は、現在のファイル **ddr2\_sdrām\_0\_debug\_design.vhd** を表したものです。図 13 は、**ddr\_top.vhd** と呼ばれる新しいトップレベル・ファイルを表したものです。

図 12. 現在のファイルの表示

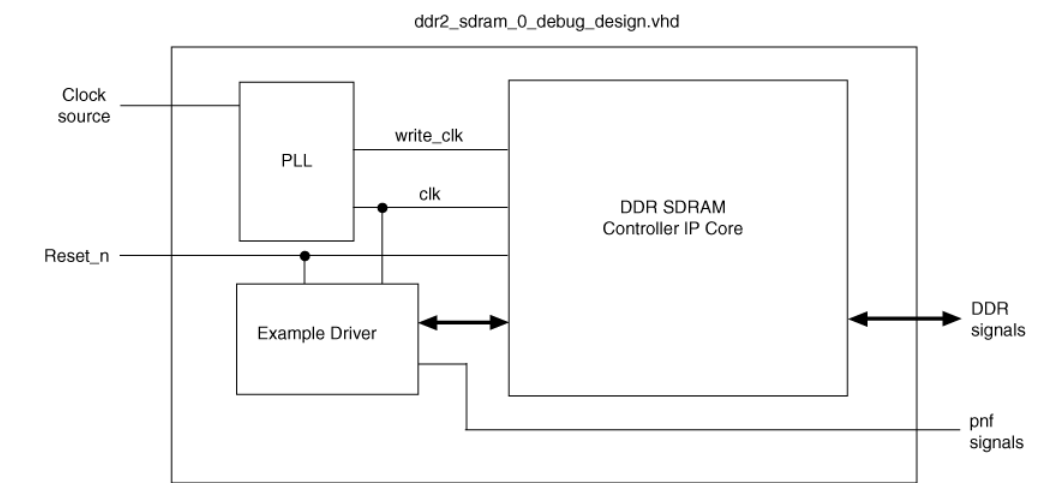
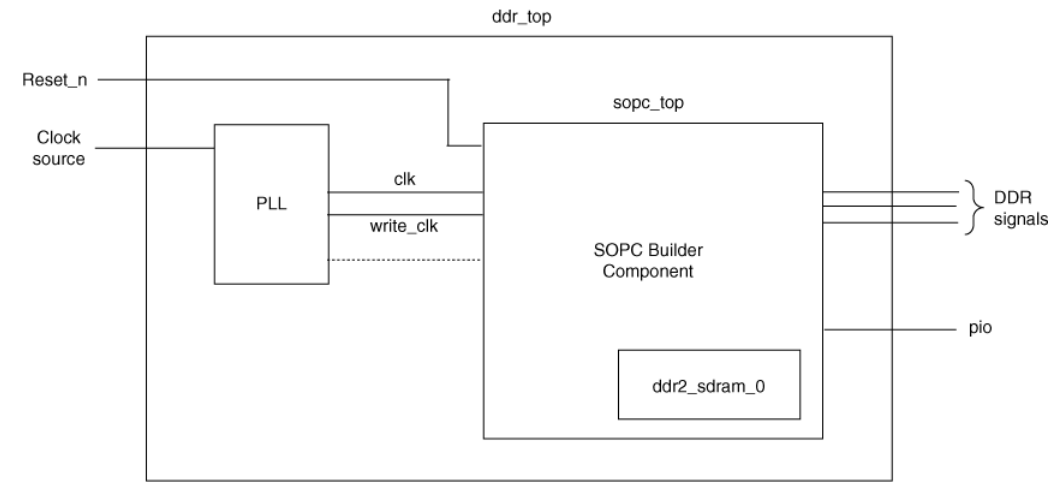


図 13. 新しいトップレベル・ファイルの表示



新しいトップレベル・ファイル `ddr_top.vhd` を作成するには、page 20 以降のコードをカット・アンド・ペーストするか、以下のステップに従って手動で変更します。

1. テキスト・エディタで `ddr2_sdram_0_debug_design.vhd` を開きます。最初の `port` 定義に移動します (図 14)。このコンポーネントは、デバッグ・シェルの信号を定義します。ステップ 2～6 では、コンポーネントを SOPC Builder 互換にするのに必要な変更を説明しています。修正されたコンポーネントを図 15 に示します。

図 14. オリジナルのポート定義

```
ddr2_sdr2m_0_debug_design.vhd - Notepad
File Edit Format View Help
entity ddr2_sdr2m_0_debug_design is
  port (
    -- inputs:
    signal clock_source : IN STD_LOGIC;
    signal reset_n : IN STD_LOGIC;

    -- outputs:
    signal clk_to_sdr2m : OUT STD_LOGIC_VECTOR (2 DOWNTO 0);
    signal clk_to_sdr2m_n : OUT STD_LOGIC_VECTOR (2 DOWNTO 0);
    signal ddr2_a : OUT STD_LOGIC_VECTOR (12 DOWNTO 0);
    signal ddr2_ba : OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
    signal ddr2_cas_n : OUT STD_LOGIC;
    signal ddr2_cke : OUT STD_LOGIC_VECTOR (0 DOWNTO 0);
    signal ddr2_cs_n : OUT STD_LOGIC_VECTOR (0 DOWNTO 0);
    signal ddr2_dm : OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
    signal ddr2_dq : INOUT STD_LOGIC_VECTOR (15 DOWNTO 0);
    signal ddr2_dqs : INOUT STD_LOGIC_VECTOR (1 DOWNTO 0);
    signal ddr2_odt : OUT STD_LOGIC_VECTOR (0 DOWNTO 0);
    signal ddr2_ras_n : OUT STD_LOGIC;
    signal ddr2_we_n : OUT STD_LOGIC;
    signal pnf : OUT STD_LOGIC;
    signal pnf_per_byte : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
    signal test_complete : OUT STD_LOGIC
  );
end entity ddr2_sdr2m_0_debug_design;
```

図 15. 修正されたポート定義

```

ddr2_s dram_0_debug_design.vhd - Notepad
File Edit Format View Help
entity ddr_top is
  port (
    -- inputs:
    signal clock_source : IN STD_LOGIC;
    signal reset_n : IN STD_LOGIC;

    -- outputs:
    signal clk_to_s dram : OUT STD_LOGIC_VECTOR (2 DOWNTO 0);
    signal clk_to_s dram_n : OUT STD_LOGIC_VECTOR (2 DOWNTO 0);
    signal ddr2_a : OUT STD_LOGIC_VECTOR (15 DOWNTO 0);
    signal ddr2_ba : OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
    signal ddr2_cas_n : OUT STD_LOGIC;
    -- signal ddr2_cke : OUT STD_LOGIC_VECTOR (0 DOWNTO 0);
    signal ddr2_cke : OUT STD_LOGIC;
    -- signal ddr2_cs_n : OUT STD_LOGIC_VECTOR (0 DOWNTO 0);
    signal ddr2_cs_n : OUT STD_LOGIC;
    signal ddr2_dm : OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
    signal ddr2_dq : INOUT STD_LOGIC_VECTOR (15 DOWNTO 0);
    signal ddr2_dqs : INOUT STD_LOGIC_VECTOR (1 DOWNTO 0);
    -- signal ddr2_odt : OUT STD_LOGIC_VECTOR (0 DOWNTO 0);
    signal ddr2_odt : OUT STD_LOGIC;
    signal ddr2_ras_n : OUT STD_LOGIC;
    signal ddr2_we_n : OUT STD_LOGIC;
    signal pio_0 : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
    -- signal pnf : OUT STD_LOGIC;
    -- signal pnf_per_byte : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
    -- signal test_complete : OUT STD_LOGIC
  );
end entity ddr_top;

```

- 最初のポート定義の名前を、`ddr2_s dram_0_debug_design` から現在のトップレベル名 `ddr_top` に変更します。
- 信号 `ddr2_a` のアドレス範囲を `15 DOWNTO 0` に拡張します。
- SOPC Builder により、一部のバスが `STD_LOGIC_VECTOR [0 DOWNTO 0]` から `STD_LOGIC` に変更されます。図 15 に示される 3 つの信号を、各ラインの下に示される新しい名前に変更します。
- `pio_0` 信号を挿入します。この信号は LED をトグルします。
- `pnf`、`pnf_per_byte`、および `test_complete` の各信号を削除します。図 15 では、これらの信号はコメント・アウトされています。
- `sopc_top.vhd` を開きます。ファイルの一番下までスクロールしてから、少し上の `test_bench` コンポーネントまでスクロールします (図 16)。

図 16. テスト・ベンチ・コード

```
sopc_top.vhd - Notepad
File Edit Format View Help
architecture europa of test_bench is
component socp_top is
  port (
    -- 1) global signals:
    signal clk : IN STD_LOGIC;
    signal reset_n : IN STD_LOGIC;

    -- the_dds2_sdram_0
    signal clk_to_sdram_from_the_dds2_sdram_0 : OUT STD_LOGIC_VECTOR (2 DOWNTO 0);
    signal clk_to_sdram_n_from_the_dds2_sdram_0 : OUT STD_LOGIC_VECTOR (2 DOWNTO 0);
    signal ddr2_a_from_the_dds2_sdram_0 : OUT STD_LOGIC_VECTOR (12 DOWNTO 0);
    signal ddr2_ba_from_the_dds2_sdram_0 : OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
    signal ddr2_cas_n_from_the_dds2_sdram_0 : OUT STD_LOGIC;
    signal ddr2_cke_from_the_dds2_sdram_0 : OUT STD_LOGIC;
    signal ddr2_cs_n_from_the_dds2_sdram_0 : OUT STD_LOGIC;
    signal ddr2_dm_from_the_dds2_sdram_0 : OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
    signal ddr2_dq_to_and_from_the_dds2_sdram_0 : INOUT STD_LOGIC_VECTOR (15 DOWNTO 0);
    signal ddr2_dqs_to_and_from_the_dds2_sdram_0 : INOUT STD_LOGIC_VECTOR (1 DOWNTO 0);
    signal ddr2_odt_from_the_dds2_sdram_0 : OUT STD_LOGIC;
    signal ddr2_ras_n_from_the_dds2_sdram_0 : OUT STD_LOGIC;
    signal ddr2_we_n_from_the_dds2_sdram_0 : OUT STD_LOGIC;
    signal write_clk_to_the_dds2_sdram_0 : IN STD_LOGIC;

    -- the_pio_0
    signal out_port_from_the_pio_0 : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
  );
end component socp_top;
```

8. この test\_bench コンポーネントを ddr2\_sdram\_0\_debug\_design.vhd にコピーし、dds2\_sdram\_0 コンポーネントおよび ddr2\_sdram\_0\_example\_driver コンポーネントと置き換えます。
9. アーキテクチャ文の test\_bench 名を ddr\_top に変更します。
10. ddr2\_sdram\_0\_debug\_design.vhd の ddr\_pll\_cycloneii コンポーネントの直後にある、internal\_ から始まる信号を削除します。
11. ddr2\_sdram\_0\_debug\_design.vhd の begin 文の直後に、以下の行を挿入します。

```
dds2_a(15 DOWNTO 13) <= (others => ('0'));
```

この文により、dds2\_a レジスタ（このデザインでは未使用）の最上位ビットがゼロに設定されます。

12. socp\_top.vhd で、DUT Instantiation に進みます（図 17）。

図 17. DUT コード

```
sopc_top.vhd - Notepad
File Edit Format View Help
--Set us up the Dut
DUT : socp_top
port map(
  clk_to_sDRAM_from_the_ddr2_sDRAM_0 => clk_to_sDRAM_from_the_ddr2_sDRAM_0,
  clk_to_sDRAM_n_from_the_ddr2_sDRAM_0 => clk_to_sDRAM_n_from_the_ddr2_sDRAM_0,
  ddr2_a_from_the_ddr2_sDRAM_0 => ddr2_a_from_the_ddr2_sDRAM_0,
  ddr2_ba_from_the_ddr2_sDRAM_0 => ddr2_ba_from_the_ddr2_sDRAM_0,
  ddr2_cas_n_from_the_ddr2_sDRAM_0 => ddr2_cas_n_from_the_ddr2_sDRAM_0,
  ddr2_cke_from_the_ddr2_sDRAM_0 => ddr2_cke_from_the_ddr2_sDRAM_0,
  ddr2_cs_n_from_the_ddr2_sDRAM_0 => ddr2_cs_n_from_the_ddr2_sDRAM_0,
  ddr2_dm_from_the_ddr2_sDRAM_0 => ddr2_dm_from_the_ddr2_sDRAM_0,
  ddr2_dq_to_and_from_the_ddr2_sDRAM_0 => ddr2_dq_to_and_from_the_ddr2_sDRAM_0,
  ddr2_dqs_to_and_from_the_ddr2_sDRAM_0 => ddr2_dqs_to_and_from_the_ddr2_sDRAM_0,
  ddr2_odt_from_the_ddr2_sDRAM_0 => ddr2_odt_from_the_ddr2_sDRAM_0,
  ddr2_ras_n_from_the_ddr2_sDRAM_0 => ddr2_ras_n_from_the_ddr2_sDRAM_0,
  ddr2_we_n_from_the_ddr2_sDRAM_0 => ddr2_we_n_from_the_ddr2_sDRAM_0,
  out_port_from_the_pio_0 => out_port_from_the_pio_0,
  clk => clk,
  reset_n => reset_n,
  write_clk_to_the_ddr2_sDRAM_0 => write_clk_to_the_ddr2_sDRAM_0
);
```

13. この DUT Instantiation を `ddr2_sDRAM_0_debug_design.vhd` にコピーし、START MEGAWIZARD INSERT WRAPPER\_NAME で始まり、END MEGAWIZARD INSERT EXAMPLE\_DRIVERで終了するセクションと置き換えます。
14. DUT Instantiation で、ポート名をエンティティ宣言で使用する短い名前に変更します (図 18 を参照)。
15. DUT Instantiation で、`sopc_top` のアドレス出力ポート `ddr2_a_from_the_ddr2_sDRAM_0` を、オフチップになるアドレスの下位ビットにマップします。  
  
`ddr2_a_from_the_ddr2_sDRAM_0 => ddr2_a(12 DOWNTO 0),`

図 18. 修正された DUT コード

```

ddr2_sdrām_0_debug_design.vhd - Notepad
File Edit Format View Help
--Set up the DUT
DUT : socp_top
port map(
  clk_to_sdrām_from_the_ddr2_sdrām_0 => clk_to_sdrām,
  clk_to_sdrām_n_from_the_ddr2_sdrām_0 => clk_to_sdrām_n,
  ddr2_a_from_the_ddr2_sdrām_0 => ddr2_a(12 DOWNTO 0),
  ddr2_ba_from_the_ddr2_sdrām_0 => ddr2_ba,
  ddr2_cas_n_from_the_ddr2_sdrām_0 => ddr2_cas_n,
  ddr2_cke_from_the_ddr2_sdrām_0 => ddr2_cke,
  ddr2_cs_n_from_the_ddr2_sdrām_0 => ddr2_cs_n,
  ddr2_dm_from_the_ddr2_sdrām_0 => ddr2_dm,
  ddr2_dq_to_and_from_the_ddr2_sdrām_0 => ddr2_dq,
  ddr2_dqs_to_and_from_the_ddr2_sdrām_0 => ddr2_dqs,
  ddr2_odt_from_the_ddr2_sdrām_0 => ddr2_odt,
  ddr2_ras_n_from_the_ddr2_sdrām_0 => ddr2_ras_n,
  ddr2_we_n_from_the_ddr2_sdrām_0 => ddr2_we_n,
  out_port_from_the_pio_0 => pio_0,
  clk => clk,
  reset_n => reset_n,
  write_clk_to_the_ddr2_sdrām_0 => write_clk
);

```

16. `ddr2_sdrām_0_debug_design.vhd` では、MEGAWIZARD INSERT PLL で囲まれたセクションは残しますが、このセクションの後からコードの最後の行 `end europa` を除く部分をすべて削除します。
17. ファイル名をトップレベルの Quartus II プロジェクト名、`ddr_top.vhd` に変更して、保存します。この新しいファイルは、以下のようになります（ほとんどのコメントは省略されています）。

```
--Legal Notice: (C)2005 Altera Corporation. All rights reserved.
```

```
library altera_mf;
use altera_mf.all;
```

```
library altera_vhdl_support;
use altera_vhdl_support.altera_vhdl_support_lib.all;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
```

```
-----
--This confidential and proprietary software may be used only as authorized by
--a licensing agreement from Altera Corporation.
--(C) COPYRIGHT 2004 ALTERA CORPORATION
--ALL RIGHTS RESERVED
--The entire notice above must be reproduced on all authorized copies and any
--such reproduction must be pursuant to a licensing agreement from Altera.
--This module instantiates your configured Altera DDR SDRAM Controller,
```

```

--some example driver logic, and a suitably configured PLL and DLL (where needed).
-----

entity ddr_top is
  port (
    -- inputs:
    signal clock_source : IN STD_LOGIC;
    signal reset_n : IN STD_LOGIC;

    -- outputs:
    signal clk_to_sdram : OUT STD_LOGIC_VECTOR (2 DOWNTO 0);
    signal clk_to_sdram_n : OUT STD_LOGIC_VECTOR (2 DOWNTO 0);
    signal ddr2_a : OUT STD_LOGIC_VECTOR (15 DOWNTO 0);
    signal ddr2_ba : OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
    signal ddr2_cas_n : OUT STD_LOGIC;
    -- signal ddr2_cke : OUT STD_LOGIC_VECTOR (0 DOWNTO 0);
    signal ddr2_cke : OUT STD_LOGIC;
    -- signal ddr2_cs_n : OUT STD_LOGIC_VECTOR (0 DOWNTO 0);
    signal ddr2_cs_n : OUT STD_LOGIC;
    signal ddr2_dm : OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
    signal ddr2_dq : INOUT STD_LOGIC_VECTOR (15 DOWNTO 0);
    signal ddr2_dqs : INOUT STD_LOGIC_VECTOR (1 DOWNTO 0);
    -- signal ddr2_odt : OUT STD_LOGIC_VECTOR (0 DOWNTO 0);
    signal ddr2_odt : OUT STD_LOGIC;
    signal ddr2_ras_n : OUT STD_LOGIC;
    signal ddr2_we_n : OUT STD_LOGIC;
    signal pio_0 : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
    -- signal pnf : OUT STD_LOGIC;
    -- signal pnf_per_byte : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
    -- signal test_complete : OUT STD_LOGIC
  );
end entity ddr_top;

architecture europa of ddr_top is
  component socp_top is
    port (
      -- 1) global signals:
      signal clk : IN STD_LOGIC;
      signal reset_n : IN STD_LOGIC;

      -- the_dds2_sdram_0
      signal clk_to_sdram_from_the_dds2_sdram_0 : OUT STD_LOGIC_VECTOR (2
DOWNTO 0);
      signal clk_to_sdram_n_from_the_dds2_sdram_0 : OUT STD_LOGIC_VECTOR (2
DOWNTO 0);
      signal ddr2_a_from_the_dds2_sdram_0 : OUT STD_LOGIC_VECTOR (12 DOWNTO 0);
      signal ddr2_ba_from_the_dds2_sdram_0 : OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
      signal ddr2_cas_n_from_the_dds2_sdram_0 : OUT STD_LOGIC;
      signal ddr2_cke_from_the_dds2_sdram_0 : OUT STD_LOGIC;
      signal ddr2_cs_n_from_the_dds2_sdram_0 : OUT STD_LOGIC;
      signal ddr2_dm_from_the_dds2_sdram_0 : OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
      signal ddr2_dq_to_and_from_the_dds2_sdram_0 : INOUT STD_LOGIC_VECTOR
(15 DOWNTO 0);
      signal ddr2_dqs_to_and_from_the_dds2_sdram_0 : INOUT STD_LOGIC_VECTOR
(1 DOWNTO 0);
      signal ddr2_odt_from_the_dds2_sdram_0 : OUT STD_LOGIC;
      signal ddr2_ras_n_from_the_dds2_sdram_0 : OUT STD_LOGIC;
    );
  end component socp_top;
end architecture europa;

```

```

        signal ddr2_we_n_from_the_ddr2_sdram_0 : OUT STD_LOGIC;
        signal write_clk_to_the_ddr2_sdram_0 : IN STD_LOGIC;

        -- the_pio_0
        signal out_port_from_the_pio_0 : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
    );
end component socp_top;

component ddr_pll_cycloneii is
PORT (
    signal c0 : OUT STD_LOGIC;
    signal c1 : OUT STD_LOGIC;
    signal c2 : OUT STD_LOGIC;
    signal inclk0 : IN STD_LOGIC
);
end component ddr_pll_cycloneii;
    signal clk : STD_LOGIC;
    signal ddr2_local_addr : STD_LOGIC_VECTOR (23 DOWNTO 0);
    signal ddr2_local_be : STD_LOGIC_VECTOR (3 DOWNTO 0);
    signal ddr2_local_burstbegin : STD_LOGIC;
    signal ddr2_local_col_addr : STD_LOGIC_VECTOR (9 DOWNTO 0);
    signal ddr2_local_cs_addr : STD_LOGIC;
    signal ddr2_local_rdata : STD_LOGIC_VECTOR (31 DOWNTO 0);
    signal ddr2_local_rdata_valid : STD_LOGIC;
    signal ddr2_local_read_req : STD_LOGIC;
    signal ddr2_local_ready : STD_LOGIC;
    signal ddr2_local_refresh_req : STD_LOGIC;
    signal ddr2_local_size : STD_LOGIC_VECTOR (1 DOWNTO 0);
    signal ddr2_local_wdata : STD_LOGIC_VECTOR (31 DOWNTO 0);
    signal ddr2_local_write_req : STD_LOGIC;
    signal dedicated_resynch_or_capture_clk : STD_LOGIC;
    -- signal internal_clk_to_sdram : STD_LOGIC_VECTOR (2 DOWNTO 0);
    -- signal internal_clk_to_sdram_n : STD_LOGIC_VECTOR (2 DOWNTO 0);
    -- signal internal_ddr2_a : STD_LOGIC_VECTOR (12 DOWNTO 0);
    -- signal internal_ddr2_ba : STD_LOGIC_VECTOR (1 DOWNTO 0);
    -- signal internal_ddr2_cas_n : STD_LOGIC;
    -- signal internal_ddr2_cke : STD_LOGIC_VECTOR (0 DOWNTO 0);
    -- signal internal_ddr2_cs_n : STD_LOGIC_VECTOR (0 DOWNTO 0);
    -- signal internal_ddr2_dm : STD_LOGIC_VECTOR (1 DOWNTO 0);
    -- signal internal_ddr2_odt : STD_LOGIC_VECTOR (0 DOWNTO 0);
    -- signal internal_ddr2_ras_n : STD_LOGIC;
    -- signal internal_ddr2_we_n : STD_LOGIC;
    -- signal internal_pnf : STD_LOGIC;
    -- signal internal_pnf_per_byte : STD_LOGIC_VECTOR (3 DOWNTO 0);
    -- signal internal_test_complete : STD_LOGIC;
    signal write_clk : STD_LOGIC;

begin
    ddr2_a(15 DOWNTO 13) <= (others => ('0'));
    ddr2_local_refresh_req <= std_logic('0');

    --Set up the Dut
    DUT : socp_top
    port map(
        clk_to_sdram_from_the_ddr2_sdram_0 => clk_to_sdram,
        clk_to_sdram_n_from_the_ddr2_sdram_0 => clk_to_sdram_n,
        ddr2_a_from_the_ddr2_sdram_0 => ddr2_a(12 DOWNTO 0),

```

```

    ddr2_ba_from_the_ddr2_sdram_0 => ddr2_ba,
    ddr2_cas_n_from_the_ddr2_sdram_0 => ddr2_cas_n,
    ddr2_cke_from_the_ddr2_sdram_0 => ddr2_cke,
    ddr2_cs_n_from_the_ddr2_sdram_0 => ddr2_cs_n,
    ddr2_dm_from_the_ddr2_sdram_0 => ddr2_dm,
    ddr2_dq_to_and_from_the_ddr2_sdram_0 => ddr2_dq,
    ddr2_dqs_to_and_from_the_ddr2_sdram_0 => ddr2_dqs,
    ddr2_odt_from_the_ddr2_sdram_0 => ddr2_odt,
    ddr2_ras_n_from_the_ddr2_sdram_0 => ddr2_ras_n,
    ddr2_we_n_from_the_ddr2_sdram_0 => ddr2_we_n,
    out_port_from_the_pio_0 => pio_0,
    clk => clk,
    reset_n => reset_n,
    write_clk_to_the_ddr2_sdram_0 => write_clk
);

--<< START MEGAWIZARD INSERT PLL
g_cyclonepll_ddr_pll_inst : ddr_pll_cycloneii
  port map(
    c0 => clk,
    c1 => write_clk,
    c2 => dedicated_resynch_or_capture_clk,
    inclk0 => clock_source
  );

--<< END MEGAWIZARD INSERT PLL

end europa;

```

## プロジェクト のコンパイル

### プロジェクト設定の更新

プロジェクト設定を正しい値にしてから、システムのコンパイルを開始しなければなりません。

最良のタイミング結果を得るには、Fitter を **Auto** ではなく **Standard** に設定します。以下のステップを実行します。

1. Assignments メニューから **Settings** をクリックします。
2. Category リストで、**Fitter Settings** を選択します。
3. Fitter エフォート・ボックスで、**Standard Fit** を選択します。
4. **OK** をクリックします。

未使用のピンを入力トライステートに設定します。このように設定していない場合は、SignalTap® ロジック・アナライザが正しく動作しないことがあります。以下のステップを実行します。

1. Assignments メニューから **Settings** をクリックします。
2. Category リストで、**Device** を選択します。

3. Device ダイアログ・ボックスで、**Device & Pin Options** をクリックします。
4. Unused Pins タブで、**As inputs, tri-stated** を選択します。
5. **OK** をクリックします。
6. Device ダイアログ・ボックスで、**OK** をクリックします。

### PLL の編集

PLL のデフォルトのクロック比は 1:1 です。ただし、使用するボードの入力クロックが 33 MHz または 100 MHz になる可能性がある場合は、別の出力クロックが必要になります。

Cyclone II DSP ボードに対しては、以下のステップを実行してください。

1. Tools メニューで、**MegaWizard Plug-In Manager** をクリックします。
2. MegaWizard Plug-In Manager [page 1] ダイアログ・ボックスで、**Edit an existing custom megafunction variation** を選択し、**Next** をクリックします。
3. MegaWizard Plug-In Manager [page 2b] ダイアログ・ボックスで、**ddr\_pll\_cycloneii.vhd** を選択し、**Next** をクリックします。
4. MegaWizard Plug-In Manager – ALTPLL [page 3 of 9] ダイアログ・ボックスで、**inclock0** 入力の周波数を 100 MHz に設定します。
5. **Next** を 3 回クリックします。
6. MegaWizard Plug-In Manager – ALTPLL [page 6 of 9] ダイアログ・ボックスで、出力クロック **c0** のパラメータを設定します。

**Clock multiplication factor** を 5 に設定します。

**Clock division factor** を 4 に設定します。

これらのパラメータにより、**c0** の周波数は 125 MHz に設定されます。

7. **Next** をクリックします。
8. MegaWizard Plug-In Manager – ALTPLL [page 7 of 9] ダイアログ・ボックスで、出力クロック **c1** のパラメータを設定します。

**Clock multiplication factor** を 5 に設定します。

**Clock division factor** を 4 に設定します。

これらのパラメータにより、**c1** の周波数は 125 MHz に設定されます。

9. **Next** をクリックします。

10. MegaWizard Plug-In Manager – ALTPLL [page 8 of 9] ダイアログ・ボックスで、出力クロック c2 のパラメータを設定します。

**Clock multiplication factor** を 5 に設定します。

**Clock division factor** を 4 に設定します。

これらのパラメータにより、c2 の周波数は 125 MHz に設定されます。

11. **Next** をクリックします。
12. **Finish** をクリックします。
13. 現在のファイルを上書きするかどうかを尋ねるダイアログ・ボックスが表示される場合、**OK** をクリックします。

MegaWizard Plug-In Manager に戻り、PLL の自動アップデート・オプションをオフに設定しない場合、PLL 設定は上書きされます。

1. Tools メニューで **SOPC Builder** をクリックして、SOPC Builder を起動します。

**Altera SOPC Builder** ダイアログ・ボックスが表示されます。

2. **ddr2\_sdram\_0** モジュールをダブル・クリックします。IP Toolbench が表示されます (7 ページの図 5 を参照)。
3. **Step 1: Parameterize** をクリックします。Parameterize ダイアログ・ボックスが表示されます。
4. **Advanced Mode** をオンにします。
5. Hierarchy タブで、**Automatically generate the PLL** をオフにします。



DDR/DDR2 SDRAM コントローラ MegaCore ファンクション v3.3.0 では、このタブは Project Settings と呼ばれます。

6. **Finish** をクリックします。

## システムのコンパイル

Processing メニューから **Start Compilation** をクリックします。

プリコンパイル処理により、自動的に制約が追加されます。制約が追加されたことを確認するには、Assignments メニューに移動して、**Assignment Editor** をクリックします。

制約が追加されていない場合、Tools メニューから **Tcl Scripts** をクリックし、**add\_constraints** スクリプトを実行します。

## ポスト・コンパイルの変更

### タイミングの確認

タイミング検証スクリプトは、システムをコンパイルすると自動的に実行されます。スクリプトの実行が成功すると、回路は予期したとおりコンパイルされます。一部でも最適化されていない場合、タイミング検証スクリプトは失敗します。

タイミング検証結果を確認します。Cyclone II 開発ボードをターゲットにしたときは、再同期化が失敗する場合があります。失敗したタイミングは青でハイライトされます。

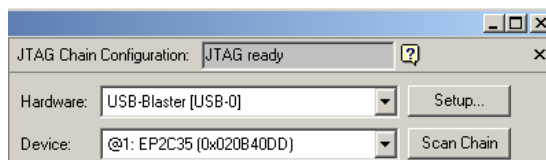
Timing Analyzer レポートで、内部  $f_{MAX}$  が要求に適合するかどうかも確認します。

### SignalTap のハードウェアのインストール

Tools メニューから **SignalTap II Logic Analyzer** をクリックします。SignalTap® ダイアログ・ボックスが表示されます。Hardware フィールドで、ドロップダウン・メニューから **USB-Blaster (USB-0)** を選択します (図 19)。

必要なハードウェア・ドライバがない場合は、アルテラのウェブサイトアクセスして、適切なドライバをダウンロードします。

図 19. Hardware Installation in SignalTap II ダイアログ・ボックス

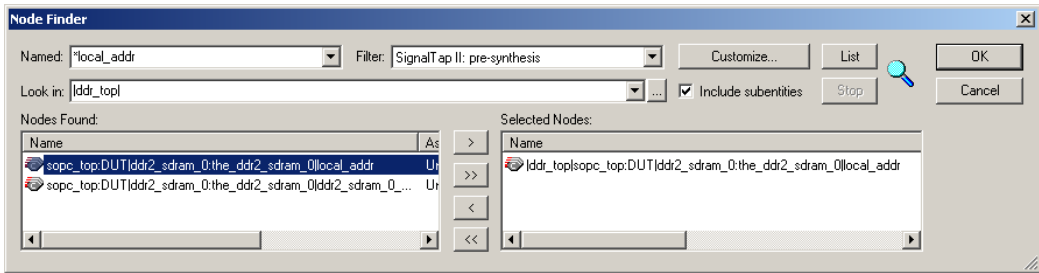


### SignalTap の信号追加

SignalTap II ロジック・アナライザに信号を追加するには、以下のステップに従います。

1. ノード選択ボックス内をダブル・クリックして、Node Finder ダイアログ・ボックスを表示します (図 20)。

図 20. SignalTap Node Finder



2. local\_addr 信号を追加するには、**Named** ボックスに \*local\_addr を入力して、**List** をクリックします。
3. **Nodes Found** リストで、  
sopc\_top:DUT|ddr2\_sdram\_0:the\_ddr2\_sdram\_0|local\_addr  
local\_addr を選択し、> をクリックして **Selected Nodes** リストに信号を追加します (図 20)。
4. ステップ 2～3 を繰り返して、以下の信号を追加します。
  - local\_rdata
  - local\_rdata\_valid
  - local\_read\_req
  - local\_ready
  - local\_wdata
  - local\_write\_req
5. **OK** をクリックします。
6. local\_addr、local\_rdata、および local\_wdata の各信号で、トリガ・イネーブルをオフにします。これにより、必要な SignalTap トリガ・ロジックが減少します。
7. local\_write\_req 信号の **Trigger Levels** セルを右クリックして、**Rising Edge** トリガを選択します。



ddr インタフェース信号は追加しないでください。この信号を追加すると、信号の負荷が増加し、タイミング解析に影響を与えます。

図 21 に、SignalTap II ロジック・アナライザの完全なノード・セクションを示します。クロック信号は、次のステップで追加します。

図 21. SignalTap II ロジック・アナライザに追加された信号

trigger: 2005/10/10 19:24:41 #0		Lock mode:  Allow all changes						
Node			Incremental Route	Debug Port Out	Data Enable		Trigger Enable	Trigger Levels
Type	Alias	Name			92/Auto	4/Auto	1 <input checked="" type="checkbox"/> Basic	
		..._ddr2_sdr2m_0jlocal_addr	<input type="checkbox"/>		<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
		..._ddr2_sdr2m_0jlocal_rdata	<input type="checkbox"/>		<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
		..._ddr2_sdr2m_0jlocal_rdata_valid	<input type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
		..._ddr2_sdr2m_0jlocal_read_req	<input type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
		..._the_ddr2_sdr2m_0jlocal_ready	<input type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
		..._ddr2_sdr2m_0jlocal_wdata	<input type="checkbox"/>		<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
		..._ddr2_sdr2m_0jlocal_write_req	<input type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	

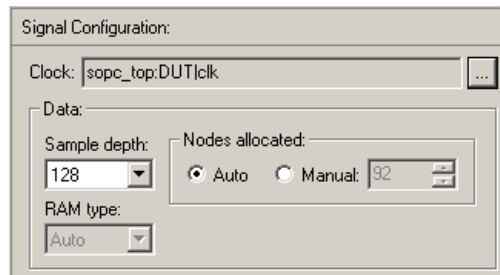
## クロック信号の追加

Signal Configuration ボックスにクロック信号を追加するには、以下のステップを実行します。

1. Signal Configuration ボックスの **Browse Node Finder** ボタンをクリックして、Node Finder ダイアログ・ボックスを表示します (図 20)。
2. **Named** ボックスに \*clk を入力し、**List** をクリックします。
3. **Nodes Found** リストで sopc\_top:DUT|clk を選択し、> をクリックして **Selected Nodes** リストに信号を追加します。
4. **OK** をクリックします。
5. ファイルを保存します。
6. 現在のプロジェクトで、SignalTap II file “stp1.stp” をイネーブルにするかどうかを尋ねるメッセージが表示された場合は、**Yes** をクリックします。

図 22 に、SignalTap II ロジック・アナライザの完成した信号コンフィギュレーション・セクションを示します。

図 22. Signal Configuration ダイアログ・ボックス



## ピン配置ファイルのソーシング

Tools メニューから、**Tcl Scripts** をクリックします。Tcl Scripts ダイアログ・ボックスが表示されます。

ピン配置ファイルをソースするには、以下のステップに従います。

1. Libraries ボックスで、C:/MegaCore/ddr\_dds2\_sdram-v3.2.0/lib/ ディレクトリにスクロールして、**cycloneii\_dsp\_rev\_a\_pins** を選択します。
2. **Run** をクリックします。

実行に成功すると、以下のメッセージが表示されます。

```
Info: Successfully loaded and ran Tcl Script File
"C:\MegaCore\ddr_dds2_sdram-v3.2.0\lib\
cycloneii_dsp_rev_a_pins.tcl"
```

## ピンの変更

Assignments メニューから **Assignment Editor** をクリックします。Assignment Editor ダイアログ・ボックスが表示されます。

ピン・アサインメントを変更するには、以下のステップに従います。

1. Assignment Editor ダイアログ・ボックスで、**To** バーをクリックして、ピンをアルファベット順に並べます。
2. user\_dipsw[0] から user\_dipsw[7] の 8 本のピンを削除します。
3. dig\_1\_a など、dig\_1 から始まるピンを削除します。

ddr2\_ckeピンとddr2\_cs\_nピンの名前を変更する必要があります。図23に現在の名前を示します。

図 23. アサインメント・エディタのオリジナルのピン名

	From	To ▲	Assignment Name	Value	Enabled
91		ddr2_cas_n	Output Pin Load	24	Yes
92		ddr2_cke[0]	Fast Output Register	On	Yes
93		ddr2_cke[0]	Location	PIN_AE21	Yes
94		ddr2_cke[0]	I/O Standard	SSTL-18 Class I	Yes
95		ddr2_cke[0]	Output Pin Load	24	Yes
96		ddr2_cke[1]	Location	PIN_AC19	Yes
97		ddr2_cs_n[0]	Fast Output Register	On	Yes
98		ddr2_cs_n[0]	Location	PIN_AF22	Yes
99		ddr2_cs_n[0]	I/O Standard	SSTL-18 Class I	Yes
100		ddr2_cs_n[0]	Output Pin Load	24	Yes
101		ddr2_cs_n[1]	Location	PIN_AB18	Yes
102		ddr2_dm	Output Enable Group	1	Yes

- ピン名 ddr2\_cke[0] を ddr2\_cke[0] のすべてのインスタンスを表す ddr2\_cke に変更します。ddr2\_cke が PIN\_AE21 の位置にあることを確認します。
- ピン ddr2\_cke[1] を削除します。
- ピン名 ddr2\_cs\_n[0] を ddr2\_cs\_n[0] のすべてのインスタンスを表す ddr2\_cs\_n に変更します。ddr2\_cs\_n が PIN\_AF22 の位置にあることを確認します。
- ピン ddr2\_cs\_n[1] を削除します。

図 24 にこれらのピンの新しい名前を示します。

図 24. アサインメント・エディタによるピンの変更

	From	To ▲	Assignment Name	Value	Enabled
91		ddr2_cas_n	Output Pin Load	24	Yes
92		ddr2_cke	Fast Output Register	On	Yes
93		ddr2_cke	Location	PIN_AE21	Yes
94		ddr2_cke	I/O Standard	SSTL-18 Class I	Yes
95		ddr2_cke	Output Pin Load	24	Yes
96		ddr2_cs_n	Fast Output Register	On	Yes
97		ddr2_cs_n	Location	PIN_AF22	Yes
98		ddr2_cs_n	I/O Standard	SSTL-18 Class I	Yes
99		ddr2_cs_n	Output Pin Load	24	Yes
100		ddr2_dm	Output Enable Group	1	Yes

8. ピン名 `ddr2_odt[0]` を `ddr2_odt[0]` のすべてのインスタンスを表す `ddr2_odt` に変更します。`ddr2_odt` が `PIN_AF21` の位置にあることを確認します。
9. ピン `ddr2_odt[1]` を削除します。
10. ピン `ddr2_a[13]`、`ddr2_a[14]`、および `ddr2_a[15]` に1つずつ、3つの新しいアサインメントを作成します。
  - a. **To** 列の下の <<new>> フィールドをダブル・クリックします。
  - b. ドロップダウン・メニューから、**Node Finder** をクリックします。
  - c. **Node Finder** ダイアログ・ボックスで、**Named** ボックスに `ddr2_a*` を入力し、**List** をクリックします。
  - d. **Nodes Found** リストで、`ddr2_a[13]`、`ddr2_a[14]`、および `ddr2_a[15]` を選択し、> をクリックして、**Selected Nodes** リストにピンを追加します。
  - e. **OK** をクリックします。
11. 新しいアサインメントのそれぞれに、**Assignment Name** フィールドと **Value** フィールドを設定します。フィールドを設定するには、セルをダブル・クリックします。**Assignment Name** ドロップダウン・メニューから **I/O Standard** を、**Value** ドロップダウン・メニューから **SSTL-18 Class I** を選択します。
12. ピン名 `user_led[0]` を `pio_0[0]` に変更します。ピン名 `user_led[1]` を `pio_0[1]` に変更し、`user_led[7]` まで同様に変更します。
13. ファイルを保存します。

## 修正ファイルの実行



修正ファイルは、DDR/DDR2 SDRAM コントローラ MegaCore ファンクション v3.2.0 以前を使用している場合のみ要求されます。DDR/DDR2 SDRAM コントローラ MegaCore ファンクション v3.3.0 以降を使用している場合、修正ファイルを実行する必要はありません。

ポストアンブル・レジスタは **DQS** ピンの近くに配置されていましたが、Cyclone II のデザインでは、クロック MUX の近くに配置する必要があります。したがって、十分近接していないために、ポストアンブル・バスのいずれかがタイミング制約に適合しない可能性があります。レジスタを正しい位置に移動するには、以下のステップを実行します。

1. **Tools** メニューから、**Tcl Scripts** をクリックします。**Tcl Scripts** ダイアログ・ボックスが表示されます。

2. Libraries ボックスで、C:/MegaCore/ddr\_ddr2\_sdram-v3.2.0/lib/ ディレクトリにスクロールして、**fix\_ep2c35f672\_postamble\_constraints.tcl** を選択します。
3. **Run** をクリックします。

実行に成功すると、以下のメッセージが表示されます。

```
Info: Successfully loaded and ran Tcl Script File  
"C:\MegaCore\ddr_ddr2_sdram-v3.2.0\lib\  
fix_ep2c35f672_postamble_constraints.tcl"
```

MegaWizard Plug-In Manager に戻って、自動制約追加オプションをオフにします。これによって、システムを再コンパイルする場合、Quartus II ソフトウェアは制約ファイルが実行されません。

1. Tools メニューで **SOPC Builder** をクリックして、SOPC Builder を起動します。  
**Altera SOPC Builder** ダイアログ・ボックスが表示されます。
2. **ddr2\_sdram\_0** モジュールをダブル・クリックします。IP Toolbench が表示されます (7 ページの図 5 を参照)。
3. **Step 1: Parameterize** をクリックします。Parameterize ダイアログ・ボックスが表示されます。
4. **Advanced Mode** をオンにします。
5. Hierarchy タブで、**Automatically run add constraints script** をオフにします。



DDR/DDR2 SDRAM コントローラ MegaCore ファンクション v3.3.0 では、このタブは Project Settings と呼ばれます。

6. **Finish** をクリックします。

## プロジェクトの再コンパイル

Processing メニューから **Start Compilation** をクリックして、プロジェクトを再コンパイルします。

### タイミング結果の確認

再同期タイミング・エラーがないことを確認します。失敗したタイミングは、Quartus II メッセージ・ウィンドウの System タブで青でハイライトされます。

## SRAM オブジェクト・ファイルのダウンロード

Tools メニューから **SignalTap II Logic Analyzer** をクリックします。SignalTap ダイアログ・ボックスが表示されます。

正しいファイル **.sof** がインストールされていることを確認します。SOF Manager ボックスには、ファイル **ddr\_top.sof** が含まれているはずですが、このファイルが含まれていない場合、以下のステップを実行します。


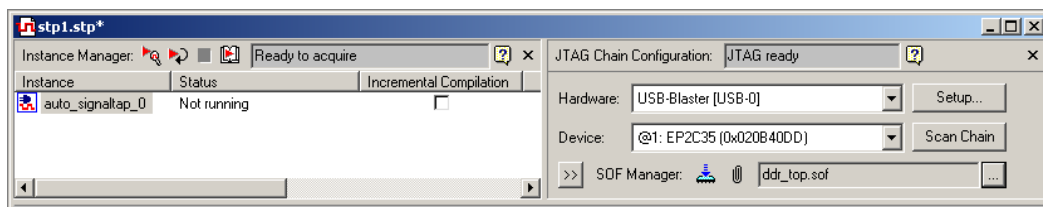

1. Browse Program Files ダイアログ・ボックスを開きます。
2. **ddr\_top.sof** を選択します。
3. **Open** をクリックします。
4. ファイルをダウンロードするには、**Program Device** ボタン  をクリックします (図 25)。

図 25. Installing the SRAM Object File In the SignalTap II ダイアログ・ボックス



## Nios II IDE の組み込み

この演習の最後のパートでは、プロジェクトに Nios II プロセッサを追加し、このプロセッサで簡単なプログラムを実行します。

 メモリ・テストを実行する場合、キャッシュ・メモリが使用されていないことを確認してください。キャッシュ・メモリの使用を避けるには、以下の3つの方法があります。

- キャッシュ・メモリを内蔵しない Nios II プロセッサを使用する
- **alt\_remap\_uncached** ファンクションを使用する
- 0x80000000 など、アドレスを 31 ビット超に設定する

## Nios II IDE の起動

Nios II IDE を起動するには、以下のステップに従います。

1. Tools メニューから、**SOPC Builder** をクリックします。

2. **System Generation** タブをクリックします。
3. **Run Nios II IDE** をクリックします。
4. Workspace Launcher ダイアログ・ボックスで、Workspace を **c:\temp** に設定して、**OK** をクリックします。



インストールした Nios II IDE と SOPC Builder のバージョンが、同じ改訂レベルであることを確認します。Help メニューを使用して、バージョン・レベルを確認します。

5. File メニューから **New** をポイントし、**C/C++ Application** をクリックします。New Project ダイアログ・ボックスが表示されます。
6. Select Project Template ボックスで、**Hello World** を選択します。
7. **Finish** をクリックします。

## テスト・コードの実装

テスト・コードを実装するには、以下のステップに従います。

1. **hello\_world.c** プログラムのコードを以下のプログラムに置き換えます。
2. **hello\_world.c** ファイルを保存します。

新しいファイルは、以下のようになります。

```
#include <stdio.h>
#include "sys/alt_dma.h"
#include "sys/alt_cache.h"

int to_hex(char* pkt)
{
    int value[8];
    int value1;
    int q;
    for (q=0; q <= 7; q++)
    {
        value[q] = (pkt[q] > 0x39) ? (pkt[q] - 0x37) : (pkt[q] - 0x30);
        if (q == 0)
            value1 = (0 + value[q]);
        else
            value1 = ((value[q-1] << 4) + value[q]);
    }
    return value1;
}

int main()
{
```

```
unsigned int *led_address;
unsigned long *DDR_address; //long is 32 bits.
int led_value;
int addr;
int datar;
int ddr_data_out;

char packet[32];

DDR_address = (unsigned long *)0x80000000; //make non-cache
led_address = (unsigned int *)0x84008800; //make non-cache

iprintf("DDR test installed \n");

while (1)
{
    gets(packet);

    switch(packet[0])
    {
    case 'A':
        led_value = to_hex(&packet[1]);
        iprintf("LED test %x \n", led_value);
        *led_address = led_value;
        break;

    case 'B': //write operation
        addr = to_hex(&packet[1]);
        datar = to_hex(&packet[9]);
        DDR_address[addr] = datar;
        iprintf("Written %08x to address %08x \n", datar, addr);
        break;

    case 'C': //Read operation
        addr = to_hex(&packet[1]);
        ddr_data_out = DDR_address[addr];
        iprintf("Read %08x from address %08x \n", ddr_data_out, addr);
        break;

    default:
        iprintf("Error: Unexpected command. Switch was - %C \n", packet[0]);
        break;
    }

}

return 0;
}
```

このプログラムは、JTAG UART のコマンドを取り込んでそれらを実行する簡単なループを構成しています。Nios/C コードに送信されるコマンドは、以下の3つの形式のいずれかでなければなりません。

- スイッチ A は LED を制御します。コマンドの形式は以下のとおりです。

### Switch | Data

例：A000000FF。最後の2バイトは、8個のLEDをすべて制御します。

- スイッチ B は DDR/DDR2 SDRAM メモリに書き込みを行います。コマンドの形式は以下のとおりです。

### Switch | Address | Data

例えば、B0000000000000001 は、メモリ・アドレス 00000000 に 1 を格納します。

- スイッチ C は DDR/DDR2 SDRAM メモリから読み出しを行います。コマンドの形式は以下のとおりです。

### Switch | Address

例えば、C00000000 はメモリ・アドレス 00000000 の内容を読み出します。



スイッチ A、B、C は大文字で入力する必要があります。

## プロジェクト設定のセットアップ

Nios II プロジェクト設定をセットアップするには、以下のステップに従います。

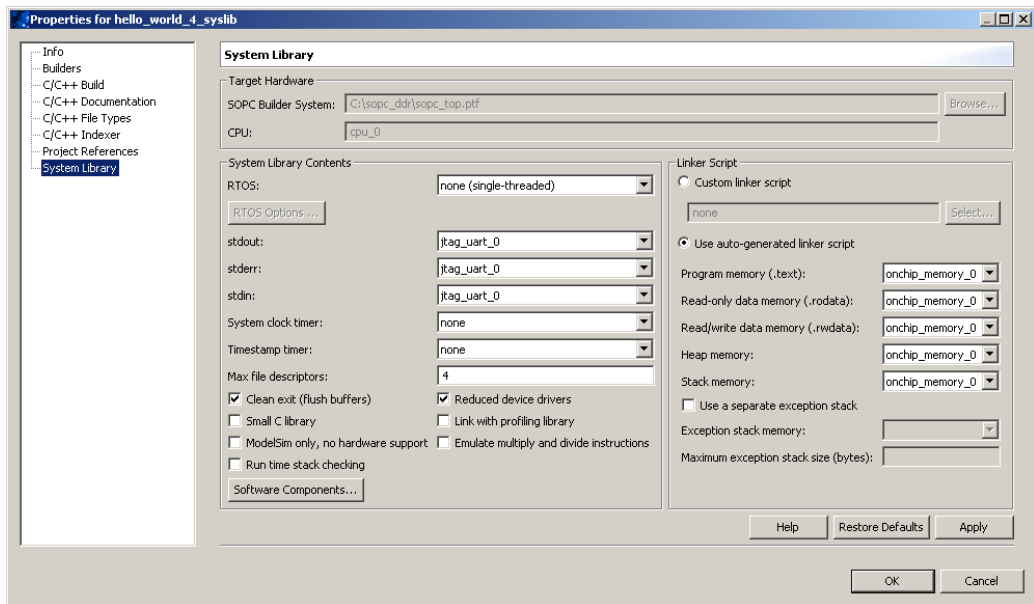
1. C/C++ Projects タブの **hello\_world\_0** を右クリックし、メニューから **System Library Properties** をクリックします。
2. Properties ダイアログ・ボックスの左側のリストで、**System Library** をクリックします。
3. System Library Contents ボックスで、**Reduced device drivers** をオンにします。
4. Linker Script ボックスで、すべてのメモリ・オプションを **onchip\_memory\_0** に設定します。
5. **Max file descriptors** を 4 に設定します。

図 26 に、オプションが選択された Properties ダイアログ・ボックスを示します。

6. **OK** をクリックします。

7. Project メニューから、**Build Project** をクリックします。

図 26. Nios II プロジェクト設定



## Nios II プロジェクトの実行

Run メニューから、Run As をポイントし、**Nios II Hardware** をクリックします。

## システムのテスト

以下のテストを実行して、システムが正しくセットアップされていることを確認します。

### LED への書き込み

コマンド・コンソールで以下を入力します。

```
A00000FE
```

Cyclone II DSP ボードで 1 個を除くすべての LED がオフになります (LED はボードではアクティブ Low です)。

コマンド・コンソールに、以下のメッセージが表示されます。

```
LED test fe
```

以下のコマンドでテストを繰り返します。

```
A000000AA
```

他の LED はすべてオフになります。

## DDR/DDR2 SDRAM メモリへの書き込み

以下のステップを実行して、DDR/DDR2 SDRAM メモリをテストします。

1. メモリ・アドレス 00000000 の内容を確認するには、Nios II コマンド・コンソールで以下を入力します。

```
C00000000
```

2. 次に、同じアドレスに書き込みます。

```
B00000000000000001
```

このコマンドにより、メモリ・アドレス 00000000 に 1 が格納されます。

3. 最後にメモリ・アドレス 00000000 の内容を読み戻し、1 があることを確認します。以下のコマンドを入力します。

```
C00000000
```


## SignalTap によるアクティビティのキャプチャ

SignalTap を使用して、読み出しおよび書き込みアクティビティを表示します。書き込みアクティビティを表示するには、以下のステップを実行します。

1. SignalTap ウィンドウで、local\_write\_req 信号が書き込み要求で立ち上がるようにトリガを設定します (図 27)。

図 27. local\_write\_req 信号のトリガの設定



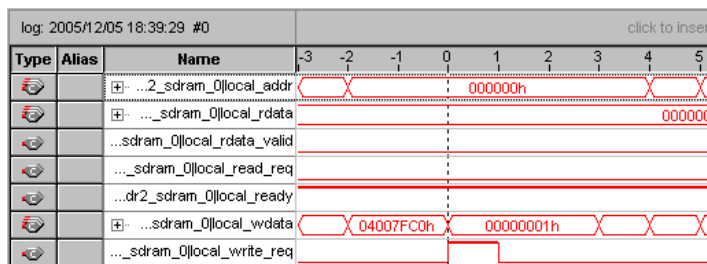
2. **Run Analysis** ボタンをクリックして、書き込み要求を1回キャプチャします。 

3. Nios II コマンド・コンソールで、以下を入力します。

```
B00000000000000001
```

- SignalTap ウィンドウに戻り、`local_write_req` 信号が時間 0 で信号の立ち上がりを示していることを確認します。`local_wdata` 信号は時間 0 で 1 の値を受け取ります (図 28)。

図 28. 書き込み信号の波形



読み出しアクティビティを表示するには、以下のステップを実行します。

- `local_write_req` 信号のトリガを **Don't care** に設定します。
- `local_read_req` 信号のトリガを読み出し要求での立ち上がりを設定します (図 29)。

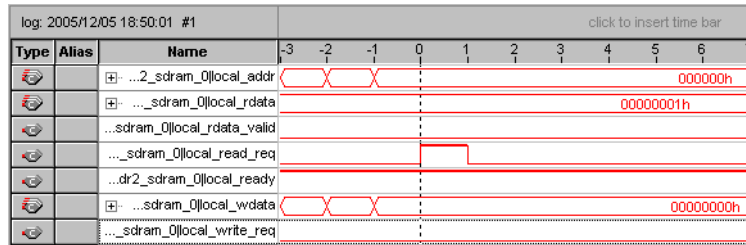
図 29. `local_read_req` 信号のトリガの設定



- Run Analysis** ボタンをクリックして、読み出し要求を 1 回キャプチャします。
- Nios II コマンド・コンソールで、以下を入力します。
 

```
C00000000
```
- SignalTap ウィンドウに戻り、`local_read_req` 信号が時間 0 で信号の立ち上がりを示していることを確認します。`local_rdata` 信号は時間 0 で 1 の値を受け取ります (図 30)。

図 30. 読み出し信号の波形



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。



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 © 2006 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. 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.

