

## Introduction

The Excalibur™ devices are excellent system development platforms, offering flexibility, performance, and programmability in an integrated package.

This document describes a dual-port SRAM (DPRAM) reference design implemented in the embedded stripe of an EPXA10F1020 device. The reference design consists of a state machine in the PLD portion and the embedded stripe configured to allow the ARM922T™ processor core to access the DPRAM. In addition, the design includes software to transfer data between the embedded processor and the DPRAM.



EXCALIBUR™



Refer to “[Revision History](#)” on [page 11](#) to see the changes made for this version of the document.

For more information on Excalibur devices, refer to the [Excalibur Devices Hardware Reference Manual](#).

## Requirements

The following hardware and software development tools are required to build, simulate, and run the design on the EPXA10 development board:

- The Altera® Quartus® II software, version 2.1 or later
- ARM® Developer Suite (ADS-Lite) software version 1.1 or Red Hat GNUPro Toolkit for ARM
- Model Technology™ ModelSim® software version 5.6
- The EPXA10 development board



When you purchase the EPXA10 Development Kit, you automatically receive a complete solution that contains all the necessary tools for this design, except that ADS-Lite is an evaluation version.

This document assumes that the above development tools have been installed and that you are familiar with them.



For more information on using the tools, refer to the [EPXA10 Development Kit Getting Started](#) manual.

## Overview

This section outlines the hardware characteristics of the DPRAM on an EPXA10 device, and explains how the DPRAM is accessed by the PLD and embedded processor. Subsequent sections describe the reference design in more detail: configuring the DPRAM and how the design works.

## DPRAM Configurations

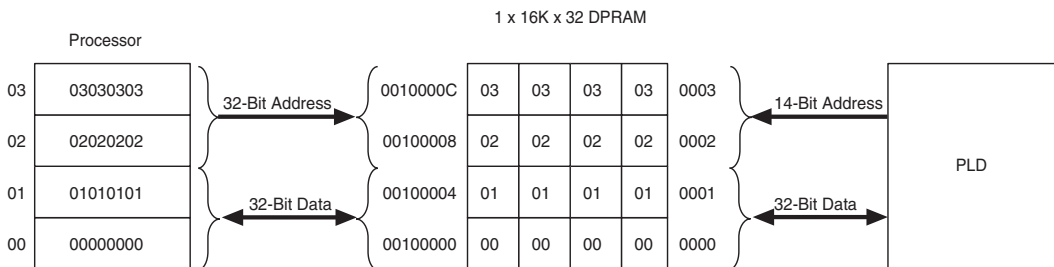
The EPXA10 device contains 128 Kbytes of DPRAM, mapped as two independent 64-Kbyte blocks or four 32-Kbyte blocks. In addition, each block can be configured independently of the other, giving many combinations of block configuration.

Figures 1 through 3 show how the data widths of the DPRAM blocks are configured and how the blocks are accessed by the embedded processor and the PLD.

In this design, the base address of the DPRAM is 00100000H, although you can specify any other valid base address. The embedded processor bus (AHB1) always accesses the DPRAM as 32-bit data because it is a 32-bit wide bus. In C programs, the data read by the embedded processor from the DPRAM is stored in a variable, and its size is declared according to the DPRAM data width (i.e., as an unsigned long integer, unsigned short integer, or unsigned character). When the embedded processor accesses the DPRAM, the variable index is translated to the corresponding AHB1 address. The PLD accesses the DPRAM either as 32-, 16-, or 8-bit data.

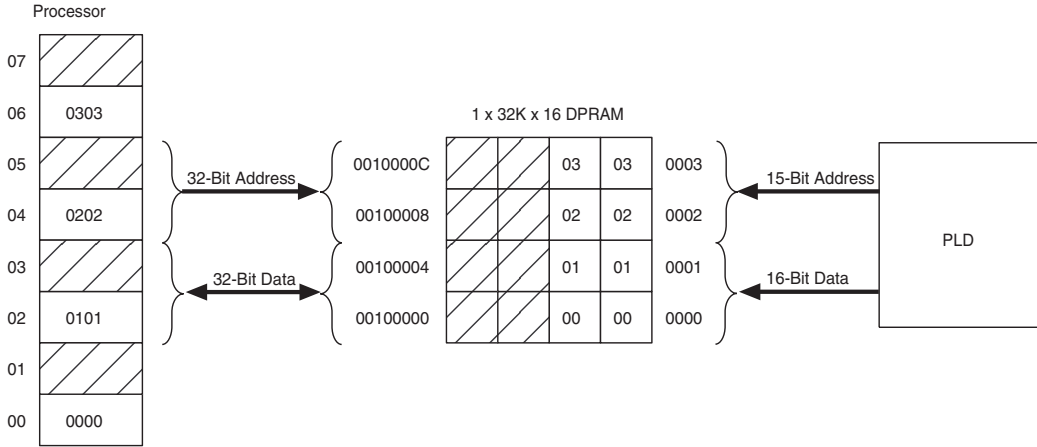
If the PLD writes 01010101H to DPRAM address 0001H in a 32-bit wide configuration, the embedded processor reads the same data at AHB1 address 00100004H and variable index 01H; see Figure 1.

Figure 1. 32-Bit Wide DPRAM



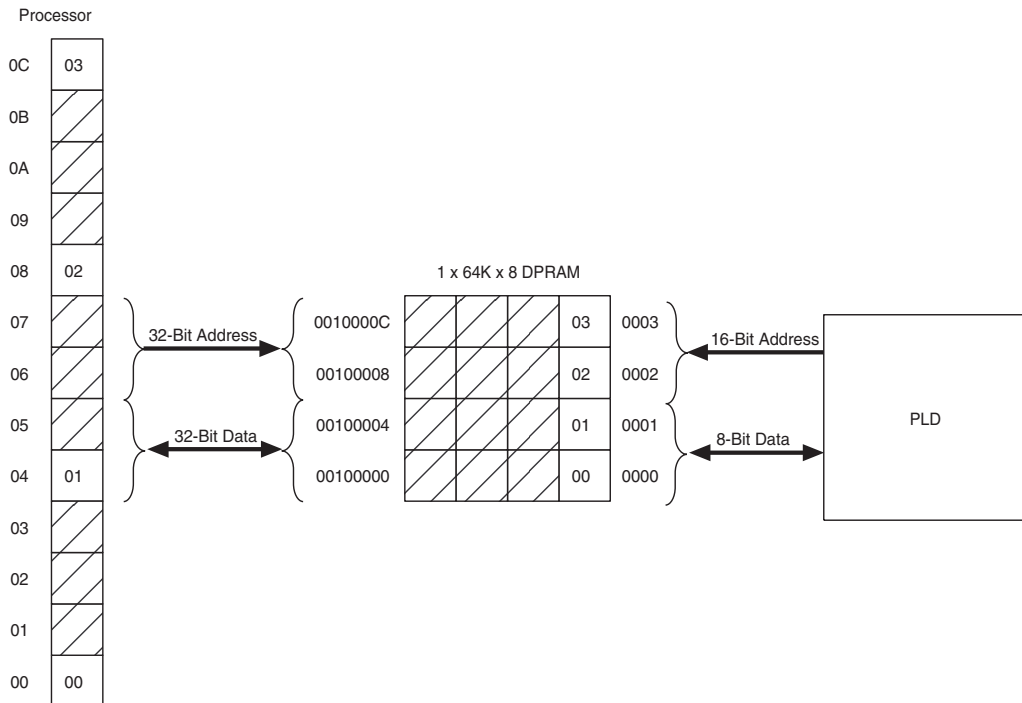
If the PLD writes the half-word value 0101H to DPRAM address 0001H in a 16-bit wide configuration, the embedded processor reads the same data at AHB1 address 00100004H and variable index 0H2; see [Figure 2 on page 3](#).

**Figure 2. 16-Bit Wide DPRAM**



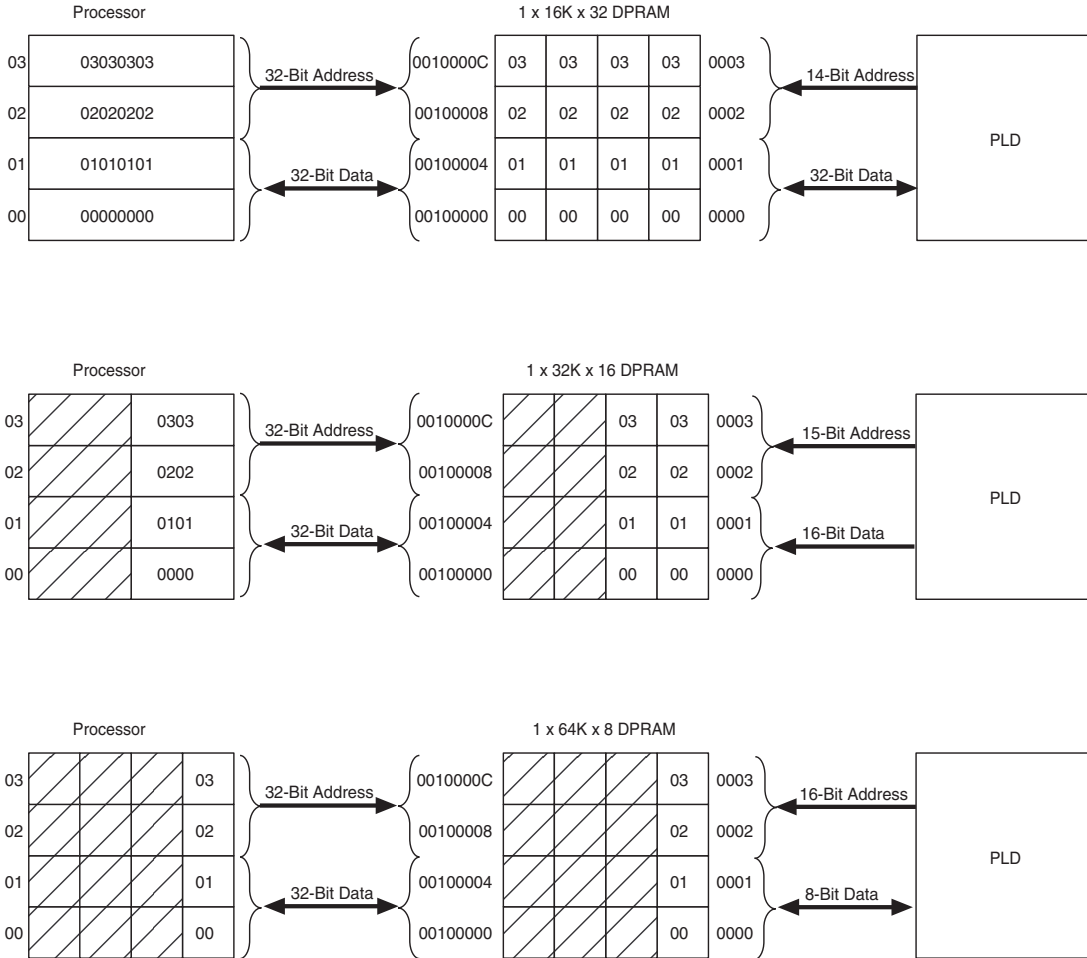
If the PLD writes the byte value 01H to DPRAM address 0001H in an 8-bit wide configuration, the embedded processor reads the same data at AHB1 address 00100004H and variable index 04H; see [Figure 3 on page 4](#).

**Figure 3. 8-Bit Wide DPRAM**



Instead of indexing the variable as shown in [Figure 1](#) through [Figure 3](#), you can declare the variable as a 32-bit data variable (i.e., unsigned long integer). The variable index then matches the DPRAM address from the PLD side; see [Figure 4](#) on page 5.

Figure 4. 32-Bit Variable

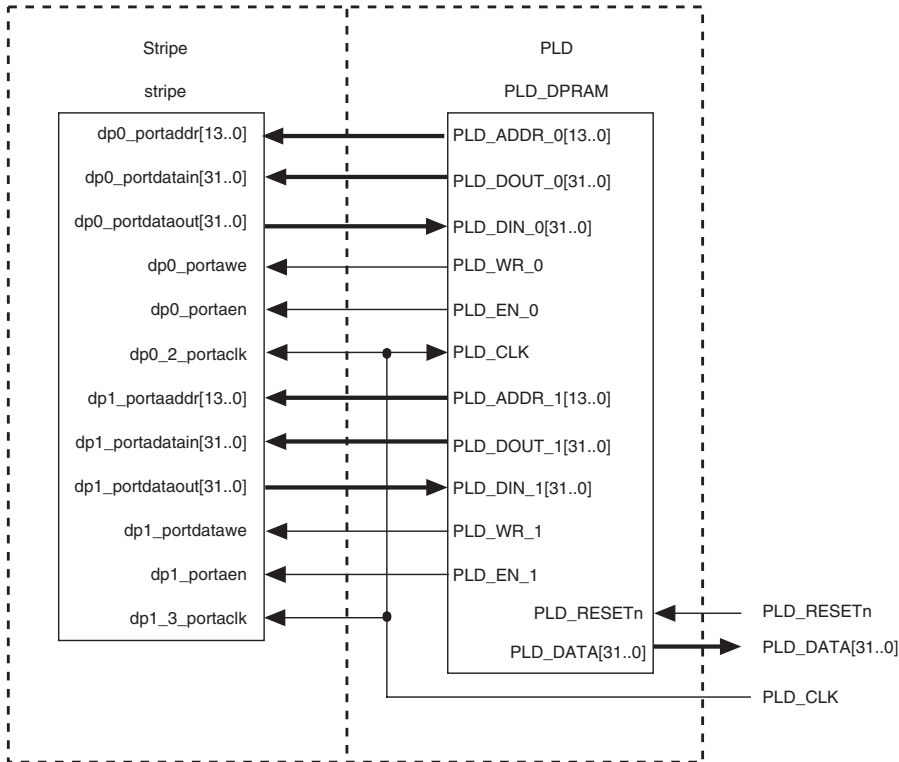


## State Machine

The PLD state machine interfaces with the stripe’s DPRAM, providing the necessary control signals to read and write to it. The state machine is initiated by setting the `PLD_RESETh` signal low. The `PLD_CLK` signal provides the clock source. Data is written from the PLD to DPRAM port 0, which is in turn read by the embedded processor. Then the embedded processor stores the same data back to DPRAM port 1, which is read by the PLD and latched to the `PLD_DATA` signal. The next section describes the data flow in more detail.

Figure 5 on page 6 shows the DPRAM reference design.

Figure 5. DPRAM Reference Design Block Diagram



## Sequence of Events

You can run the DPRAM reference design using either the **dpram\_asm** project or the **dpram\_c** project. The assembly program performs 8-word bursts of read and write transactions from the DPRAM to the embedded processor, while the C program only performs single read and write transactions. The following list describes the sequence of events for the DPRAM reference design:

1. The PLD\_DPRAM state machine writes a burst of 8 words (ranging from 01010101H to 08080808H) to the stripe's DPRAM port 0, starting at address 1.
2. The state machine sets status flag 0 (located at DPRAM port 0, address 0).

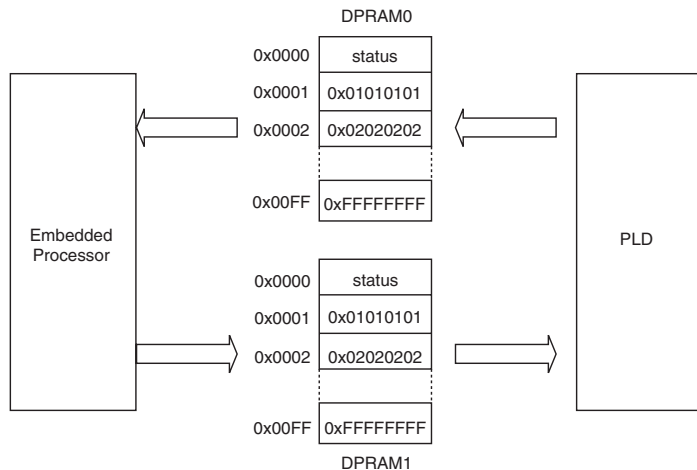


Status flag 0 is cleared if data has not been written to DPRAM port 0.

3. The embedded processor polls on status flag 0.
4. If status flag 0 is set, the embedded processor does the following:
  - a. Reads a burst of 8 words of data from DPRAM port 0 using the load multiple (LDMIB) assembly instruction.
  - b. Clears status flag 0.
  - c. Writes a burst of 8 words of the same set of data to DPRAM port 1 using the store multiple assembly instruction.
  - d. Sets status flag 1 (located at DPRAM port 1, address 0).
5. The state machine polls on status flag 1.
6. If status flag 1 is set, the state machine does the following:
  - a. Reads a burst of 8 words of data from DPRAM port 1, starting at address 1.
  - b. Clears the status flag 1.
7. Repeats steps 1 to 6 with the data and address incremented by 8 until it reaches 256.

Figure 6 shows the data flow from the PLD to DPRAM port 0 to the embedded processor, and then back to DPRAM port 1 and to the PLD. After running the design, both DPRAMs have the data contents shown in the figure.

**Figure 6. DPRAM Data Flow**



## Project Description

The DPRAM reference design contains two projects: **dpram\_asm** and **dpram\_c**. **dpram\_asm** is used for validating the design through simulation, and **dpram\_c** is used for validating the design on the EPXA10 development board. Each project contains two directories, **ads** and **gnu**, depending on which compiler you are using.

### DPRAM\_ASM Project

This project contains the necessary hardware source files and a software assembly file to simulate the hardware design. The purpose of the software assembly file is to keep the software code small in order to simulate the hardware design faster; it provides the minimum processor code initialization and performs load/store instructions to read/write data between the processor and DPRAM.

### DPRAM\_C Project

This project contains the hardware source files and C programs to run the design on the EPXA10 development board. The C programs provide a more comprehensive processor code initialization and perform a read/write to the DPRAM. The project outputs the design results to a terminal window and to the LEDs of the EPXA10 development board.

## Configuring the DPRAM

The example design has already been pre-built. To modify the DPRAM configuration, follow the steps below:

1. Run the Quartus II software.
2. Choose **Open Project** (File menu) and select the **dpram\_ref.quartus** project file either in the **ads** or **gnu** directory, depending on which compiler you are using.
3. Choose **Open** (File menu) and select **arm\_top.bdf** to open the block diagram of the top-level design.
4. Double-click on the **stripe** module to invoke the MegaWizard® Plug-In.
5. Click the **Next** button on every screen of the wizard until you reach page 4 of 5.
6. Place the cursor in the address field of DPRAM0 or DPRAM1 to change its settings. For example, to change DPRAM0 to a 64K × 8 configuration, choose **1 × single port 64K × 8** from the **PLD Access** drop-down list.

## Building the Project

- Click **Finish** to create the software header files and the verilog instantiation of the stripe.

This section describes the procedure for either building either the `dpram_asm` or `dpram_c` project.

### DPRAM\_ASM

To build the `dpram_asm` project, follow the steps below:

- Run the Quartus II software.
- Open `dpram_ref.quartus` in either the `dpram\asm\ads` or `dpram\asm\gnu` directory, depending on which compiler you are using.
- Choose **Software Mode** (Processing menu).
- Choose **Start Software Build** (Processing menu) to build the software application and run a batch file, `sdb2sim.bat`. The batch file creates the memory initialization files (`memory.regs`, `memory.sram`, and `memory.dpram`) and copies them to the `simulation\modelsim` directory.



The `memory.regs` file contains stripe register settings and the `memory.sram` file contains the RAM image of the assembly program.

Alternatively, you can run the makefile in a Command Prompt window by typing `make` in the `dpram\asm\ads` or `dpram\asm\gnu` directory. Run the batch file, `sdb2sim.bat`, after building the software application.

### DPRAM\_C

To build the `dpram_c` project, follow the steps below:

- Run the Quartus II software.
- Open `dpram_ref.quartus` in either the `dpram\c\ads` or `dpram\c\gnu` directory, depending on which compiler you are using.
- Choose **Software Mode** (Processing menu).

4. Choose **Start Software Build** (Processing menu) to build the software application.

Alternatively, you can run the makefile in a Command Prompt window by typing `make` in the `dpram\c\ads` or `dpram\c\gnu` directory.

## Simulating the Design

To run an RTL or timing simulation, follow the steps below:

1. Start the ModelSim simulation tool.
2. Click on **File > Open > Project** and select either `dpram\asm\ads\simulation\modelsim\modelsim.mpf` or `dpram\asm\gnu\simulation\modelsim\modelsim.mpf`, depending on which compiler you are using.
3. Choose **Execute Macro** (Macro menu).
4. Select `compile_and_run_rtl_fullmodel.do` and click **Open** to perform RTL simulation. Alternatively, Select `compile_and_run_timing_fullmodel.do` and click **Open** to perform a timing simulation.

## Running the Design on the EPXA10 Development Board

The following procedure lists the steps necessary to run the reference design on the EPXA10 development board:

1. Configure the following jumpers on the EPXA10 board as shown in [Table 1](#).

**Table 1. Jumper Settings**

| Jumper     | Position | Description  |
|------------|----------|--|
| MSEL0      | 1-2      | Boot from 16-bit flash   |
| MSEL1      | 1-2      | Boot from 16-bit flash   |
| DEBUG_EN   | 2-3      | Enable debug mode  |
| BOOT_FLASH | 2-3      | Boot from flash  |
| JP40       | 1-2      | Select the external 32.768 MHz oscillator on the EPXA10 board for the PLD Clock 0 source |
| JP51       | 2-3      | Enable reference clock for EPXA10 stripe   |
| JP54       | 2-3      | Enable PLD Clock 0   |

2. Start up a HyperTerminal window with the following settings: 38400 baud rate, 8 data bits, no parity, 1 stop bit, and no flow control.
3. Configure the PLD with the reference design by running the **prog\_hw.bat** DOS batch file. The software application displays the following messages on the HyperTerminal window and lights up the EPXA10 board LEDs with the corresponding data value:

```
*****
* DPRAM Reference Design *
*****

Processor Read from DPRAM0: Addr = 100004, Data = 1010101
Processor Read from DPRAM0: Addr = 100008, Data = 2020202
Processor Read from DPRAM0: Addr = 10000c, Data = 3030303
Processor Read from DPRAM0: Addr = 100010, Data = 4040404
Processor Read from DPRAM0: Addr = 100014, Data = 5050505
Processor Read from DPRAM0: Addr = 100018, Data = 6060606
Processor Read from DPRAM0: Addr = 10001c, Data = 7070707
Processor Read from DPRAM0: Addr = 100020, Data = 8080808
Processor Write to DPRAM1: Addr = 140004, Data = 1010101
Processor Write to DPRAM1: Addr = 140008, Data = 2020202
Processor Write to DPRAM1: Addr = 14000c, Data = 3030303
Processor Write to DPRAM1: Addr = 140010, Data = 4040404
Processor Write to DPRAM1: Addr = 140014, Data = 5050505
Processor Write to DPRAM1: Addr = 140018, Data = 6060606
Processor Write to DPRAM1: Addr = 14001c, Data = 7070707
Processor Write to DPRAM1: Addr = 140020, Data = 8080808
```

## Revision History

Table 2 shows the document revision history.

| <b>Table 2. Revision History</b> |  |
|----------------------------------|--|
| <b>Date</b>                      | <b>Description</b>   |
| October 2001                     | Version 2.0 first publication  |
| April 2002                       | Modification to acknowledge Quartus 2.0                                  |
| July 2002                        | Rewrite to remove duplicate information and accommodate the GNU compiler |
| August 2002                      | Directory structure modification   |



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

Copyright © 2002 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, mask work 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.

