XAPP290 (v2.0) December 3, 2007

Difference-Based Partial Reconfiguration

Author: Emi Eto

Summary

An important feature in the Virtex™ architectures is the ability to reconfigure a portion of the FPGA while the remainder of the design is still operational. Partial reconfiguration is useful for applications that require the flexibility to change portions of a design without having to completely reconfigure the entire device. With this capability, entirely new application areas become possible:

- In-the-field hardware upgrades and updates to remote sites
- Runtime reconfiguration

Other potential benefits include:

- Reduced device count
- Reduced power consumption
- More efficient use of available board space

This application note describes difference-based partial reconfiguration, which is useful for making small on-the-fly changes to design parameters such as logic equations, filter parameters, and I/O standards. This design flow is not recommended for making large changes in the functionality or structure of a design, for example, changing an entire algorithm. When there are sizable changes or the routing has to be modified, the recommended flow is to start from the HDL.

Introduction

Partial reconfiguration of Virtex devices can be accomplished through the SelectMAP, JTAG, or ICAP configuration interfaces. Instead of resetting the device and performing a complete reconfiguration, new data is loaded to reconfigure a specific area of a device, while the rest of the device is still in operation.

The difference-based partial reconfiguration design flow described in this application note allows a designer to make small logic changes using *FPGA_Editor* and generate a bitstream that programs only the difference between the two versions of the design. Switching the configuration of a module from one implementation to another is very quick because the bitstream differences can be much smaller than the entire device bitstream.

This application note does not discuss the methodology of creating reconfigurable regions to implement multiple reconfigurable modules.

Difference-Based Partial Reconfiguration

The main objective for difference-based partial reconfiguration is allow small design changes. For example, perhaps LUT programming or an I/O standard needs to be simultaneously changed and loaded. These changes can be made easily by directly editing the routed NCD file in the Xilinx *FPGA_Editor* application. If block RAM contents need to be modified, the Data2MEM utility be used instead of *FPGA_Editor*, or these changes can be made in *FPGA_Editor*.

After the changes are made, the BitGen program is used to produce a bitstream that only programs the differences between the original design and the new one. Depending on the changes, this partial bitstream can be much smaller than the original bitstream. These bitstreams can be loaded quickly and easily by the software. All that is required is an understanding of how to make logic changes using the *FPGA_Editor* application, and the pertinent options to select in BitGen.

Making Small Design Changes Using *FPGA_Editor*

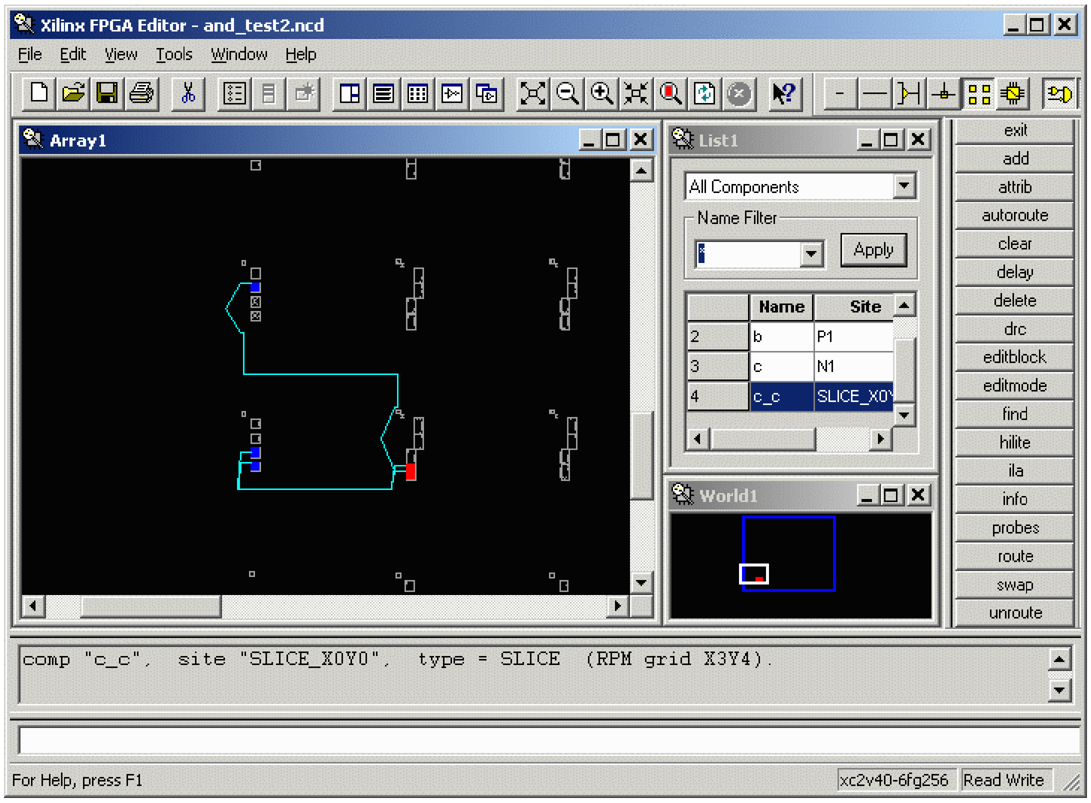
While many different types of changes can be made to an FPGA design, this application note addresses changing I/O standards, block RAM contents, and LUT programming using *FPGA_Editor*. Even though it is possible to change routing information, it is not recommended due to the possibility of internal contention during reconfiguration. If routing changes are desired, using the flow described in this application note is not recommended.

After the placed and routed NCD file is opened in *FPGA_Editor* (by specifying it on the command line or using the **File->Open** menu selection), it should be immediately saved under a different name, so that the original design is not lost. In the first example (Figure 1), **File->Save As** is selected to change the `and_test.ncd` design to `and_test2.ncd`. The latter file will remain open in *FPGA_Editor* after the operation is completed.

After the new design is open, the file becomes available for modification by selecting **File->Main Properties** and changing the **Edit Mode** to **Read Write**.

Changing LUT Equations

The smallest logical element that can be selected is the slice. First, the block must be viewed. An individual slice can be found using the Find button on the right hand side of the window, or the array view can be navigated, and the slice selected by hand. After the slice is selected, (shown as red in Figure 1,) the Editblock button should be clicked on to open the Block Editor toolbar.



	Name	Site
2	b	P1
3	c	N1
4	c_c	SLICE_X0Y

```
comp "c_c",  site "SLICE_X0Y0",  type = SLICE  (RPM grid X3Y4).
```

Figure 1: **Viewing a Block**

To prevent accidental edits, by default, the internals of a slice cannot be edited. Each time a block is opened, to make it editable, the Begin Editing button (the second button from the left in the Block Editor toolbar) must be selected. This changes the window background to black.

To view the LUT equations, the Show/Hide Attributes button must be clicked on. It is the **F=** toolbar button. This opens a panel at the bottom of the window with the slice name, and the two equations. The valid operators are:

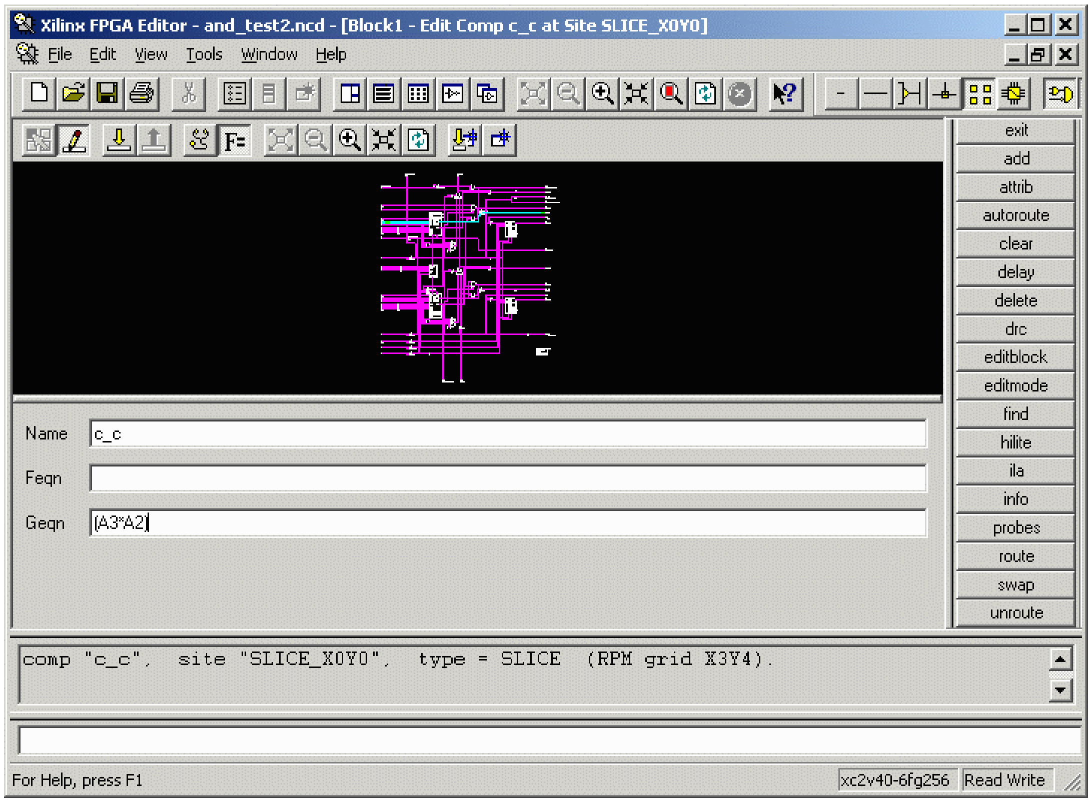
* -> Logical AND

+ -> Logical OR

@ -> Logical XOR

~ -> Unary NOT

Figure 2 shows changing the Geqn from A3*A2 to A3*~A2.



Name: c_c
Feqn:
Geqn: (A3*A2)

```
comp "c_c", site "SLICE_X0Y0", type = SLICE (RPM grid X3Y4).
```

x290_13_042402

Figure 2: **Changing LUT Equations**

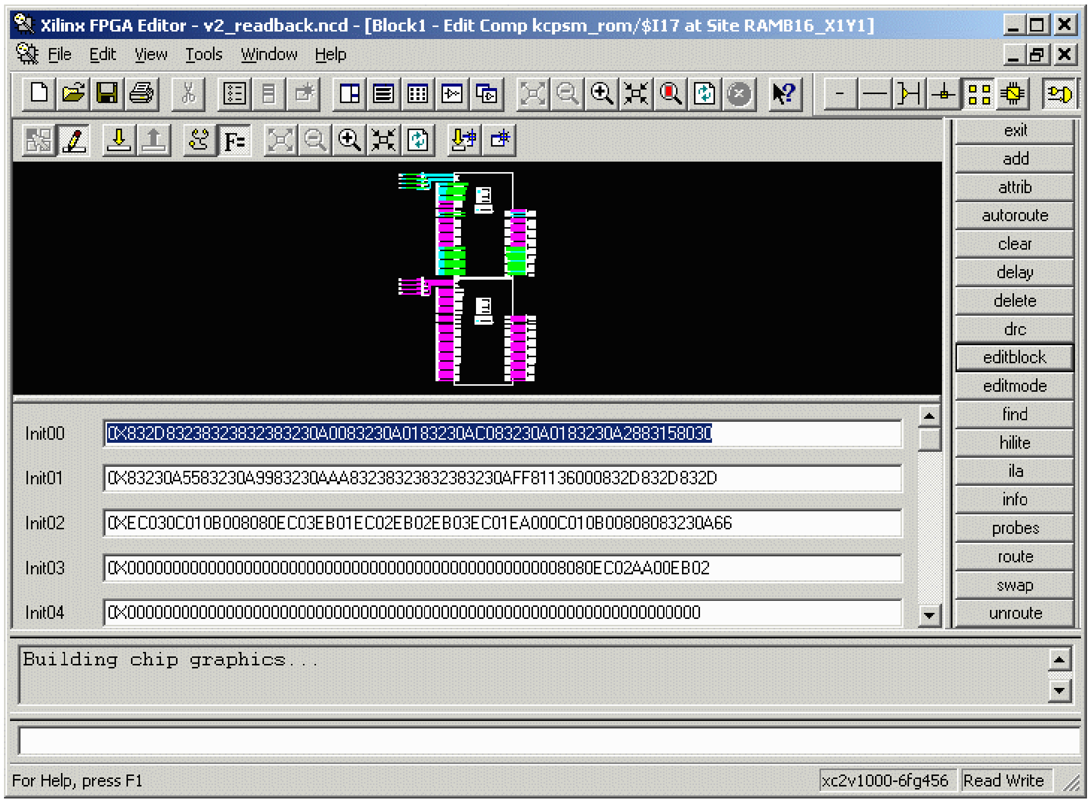
Valid equations values are A1, A2, A3, and A4, representing the four address line inputs to the LUT. Parentheses can also be used to group equation sections, e.g., `(A4 * A1) @ ~A3`. Any other names or operators will produce an error (for example):

`ERROR:FPGAEDITOR:24 - "(A3*~A2 + mynet)` is not a valid value for the Geqn attribute.

After the attributes are changed, the **Saves Changes and Closes Window** button should be selected to close the Block Editor.

Changing Block RAM Contents

The Block Editor for block RAMs (Figure 3) is similar to the slice Block Editor. While in the Block Editor mode, the Show/Hide Attributes button should be selected to display the contents of the RAM. The format of the data is the same as an INIT constraint in a UCF file. See the *Libraries Guide* for details on the INIT constraint. Once the changes have been made, the **Saves Changes and Closes Window** button should be selected to close the window and return to the Array view.



```
Init00  0X832D8323832383238323 0A0083230A0183230AC083230A0183230A2883158030
Init01  0X83230A5583230A9983230AAA832383238323 83230AFF81136000832D832D832D
Init02  0XEC030C010B008080EC03EB01EC02EB02EB03EC01EA000C010B00808083230A66
Init03  0X0000000000000000000000000000000000000000000000008080EC02AA00EB02
Init04  0X0000000000000000000000000000000000000000000000000000000000000000
```

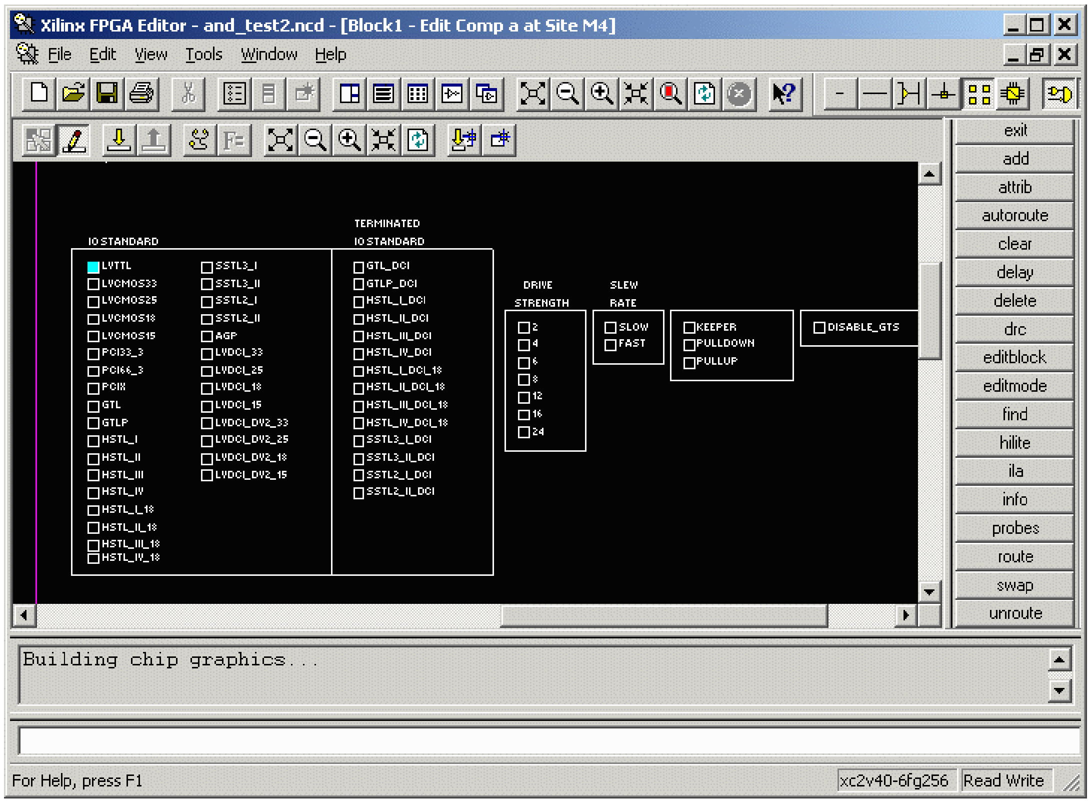
```
Building chip graphics...
```

Figure 3: **Changing Block RAM Contents**

Changing I/O Standards

To change the I/O standards, enter the Block Editor the same way as a slice or block RAM. The I/O standards are in a box in the upper-right corner of the window (Figure 4). To change the I/O standard, select the checkbox next to the desired I/O standard. There are also *Drive Strength* and *Slew Rate* checkboxes. Only select these when applicable. See the *Libraries Guide* and the specific FPGA data sheet for details on which I/O standards have selectable slew rate and drive strength.

I/O standards must match the V_{REF} voltages (or the absence of a V_{REF} voltage) with the other I/Os in the bank or the changed I/Os will not function properly. For example, it is not possible to change an LVTTL I/O in the middle of a bank of LVTTL I/Os to the GTL standard; GTL requires V_{REF} voltages, while LVTTL does not.



Building chip graphics...

Figure 4: **Changing I/O Standards**

Other Changeable Elements

A number of muxs and changeable properties in slices, IOBs, and block RAMs are eligible for a difference-based partial reconfiguration flow. Some changeable properties are: muxs that invert polarity, flip-flop initialization and reset values, pull-ups or pull-downs on external pins, or block RAM write modes. All of these properties can be modified in the actual slice, IOB, or block RAM as appropriate. Changing any property or value that would impact routing is not recommended due to the risk of internal contention.

Creating Difference-Based Partial Reconfiguration Bitstreams

The `-g ActiveReconfig:Yes` switch is required for active partial reconfiguration, meaning that the device remains in full operation while the new partial bitstream is being downloaded. If `ActiveReconfig:Yes` is not specified (or `-g ActiveReconfig:No` is specified), then the partial bitstream contains the Shutdown and AGHIGH commands used to deassert DONE. Additionally, the `-g Persist:Yes` switch is required when utilizing partial reconfiguration through the SelectMAP mode. This switch allows the SelectMAP pins to persist after the device is configured, which allows the SelectMAP interface to be used for partial reconfiguration. The `-g Persist:Yes` setting is also required for the *initial* bitstream. The `-g security:none` setting must also be set for the initial bitstream.

A partial reconfiguration bitstream can be created with any other BitGen option, including the `-b` option (create `.rbt` file) or any `-g` options specifying configuration options except for encryption. A device that has been configured with an encrypted bitstream cannot be partially reconfigured. Similarly, a device cannot be partially reconfigured with an encrypted bitstream.

A difference-based partial reconfiguration bitstream can be created with the BitGen utility using the `-r` switch. This switch produces a bitstream that contains only the differences between the input `.ncd` file and the original bit file.

Examples

Generic Example:

```
bitgen -g ActiveReconfig:Yes -g Persist:yes -r <original.bit> <new.ncd>
<new.bit>
```

Test Example:

```
bitgen -g ActiveReconfig:Yes -g Persist:Yes -r and_test.bit and_test2.ncd
and_test2_partial.bit
```

Create a Partial Bitstream to Restore the Original Design:

```
bitgen -g ActiveReconfig:Yes -g Persist:yes -r and_test2.bit and_test.ncd
and_test_partial.bit
```

These files produce a configuration file (`and_test2_partial.bit`) that only configures the frames that are different between *and_test* and *and_test2*. When downloading this file, the and_test configuration file **MUST** already be programmed into the device. It is advisable to run DRC to be alerted of any violations. No additional steps are necessary to run DRC. BitGen automatically performs the Design Rule Checker unless the `-d option` has been toggled.

Using Bitstreams and Programming the FPGA

Partial reconfiguration supports either the parallel slave SelectMAP, JTAG, or ICAP programming options. The following FPGA configuration documents are useful when programming with these options:

Virtex-E FPGAs

DS022-2:*Virtex-E 1.8V FPGA Detailed Functional Description* (Configuration Section)

Virtex-II Platform FPGAs

UG002: *Virtex-II Platform FPGA User Guide* (Chapter 4: Configuration)

Virtex-II Pro FPGAs

UG012: Virtex-II Pro and Virtex-II Pro X FPGA User Guide

Virtex-4 FPGAs

ug071: Virtex-4 FPGA Configuration Guide

Virtex-5 Platform FPGAs

ug191: Virtex-5 FPGA Configuration User Guide

The Xilinx configuration application, iMPACT, can be used in conjunction with any Xilinx download cable to interface to target device(s) for configuration testing. Alternatively, designers can create board-level functions to control device configuration at a system level.

Because the device cannot distinguish partial bitstreams from full bitstreams, designers must be careful to correctly sequence the application of these partial bitstreams to the target devices. The iMPACT software can identify a partial bitstream but cannot determine if it is being applied in the correct sequence order. When downloading a device using a partial bitstream, iMPACT software displays a message indicating that a partial bitstream is being used and that care should be taken to ensure correct sequencing:

1. Load the full bitstream
2. Load the partial bitstream to change the design.
3. If desired, load a subsequent partial bitstream to restore the original design.

Where there are two partial bitstreams, the proper sequence is:

1. Load the full bitstream.
2. Load the a partial bitstream to change the design.
3. Load the full bitstream.
4. Load the other partial bitstream to change the design.

When targeting a partial bitstream to a Xilinx configuration PROM using the iMPACT PROM file formatting capabilities, no special options are needed. The formatting of the PROM data is identical regardless of the bitstream contents. End users should be aware that when targeting Xilinx configuration PROMs, these devices do not allow selective loading of configuration data contents. Instead, all data is transmitted to the attached FPGAs. End users looking for a solution to provide bitstream selectability should consider either the SelectMAP or JTAG port through a processor or an ICAP-based solution.

Initially on device power-up, a full bitstream must be loaded into the device prior to any partial bitstreams. Only after that time can a partial bitstream be loaded to reconfigure a partially reconfigurable module. The states of the flip-flops are preserved during the reconfiguration process. Fixed portions of the design that are not being reconfigured remain fully operational during the reconfiguration process. New partial bitstreams can be subsequently loaded to change functionality.

Bitstream Length and Reprogramming Speed

The bitstream length and reprogramming time of a particular partial bitstream are directly proportional.

Conclusion

This application note is useful when trying to partially reconfigure Xilinx FPGAs with small changes to logic or I/O standards.

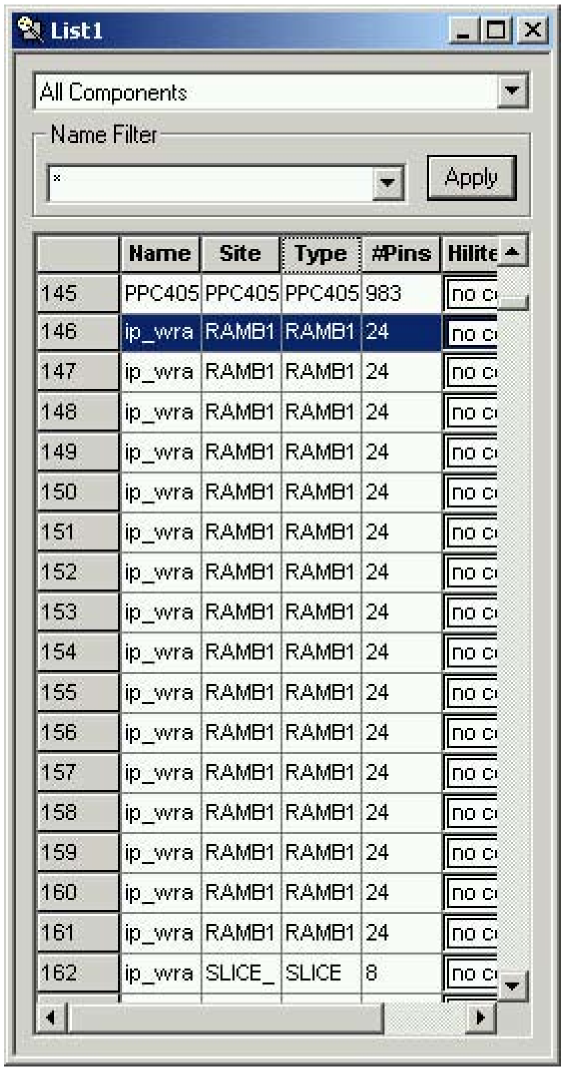
Appendix A

Saving Block RAM Contents with SaveData

In a normal reconfiguration process, block RAM contents are overwritten by the bitstream. This behavior is the default for partial reconfiguration, but can be changed through the use of the SaveData feature. SaveData mode prevents block RAM data from being overwritten during device reconfiguration.

Use the following procedure to create a SaveData bitstream:

1. Open the design in *FPGA_Editor* and use the List window to sort the components by type (see Figure 5).



	Name	Site	Type	#Pins	Hilite
145	PPC405	PPC405	PPC405	983	no c
146	ip_wra	RAMB1	RAMB1	24	no c
147	ip_wra	RAMB1	RAMB1	24	no c
148	ip_wra	RAMB1	RAMB1	24	no c
149	ip_wra	RAMB1	RAMB1	24	no c
150	ip_wra	RAMB1	RAMB1	24	no c
151	ip_wra	RAMB1	RAMB1	24	no c
152	ip_wra	RAMB1	RAMB1	24	no c
153	ip_wra	RAMB1	RAMB1	24	no c
154	ip_wra	RAMB1	RAMB1	24	no c
155	ip_wra	RAMB1	RAMB1	24	no c
156	ip_wra	RAMB1	RAMB1	24	no c
157	ip_wra	RAMB1	RAMB1	24	no c
158	ip_wra	RAMB1	RAMB1	24	no c
159	ip_wra	RAMB1	RAMB1	24	no c
160	ip_wra	RAMB1	RAMB1	24	no c
161	ip_wra	RAMB1	RAMB1	24	no c
162	ip_wra	SLICE_	SLICE	8	no c

x290_16_091903

Figure 5: **List Window**

2. Select and double-click on a RAMB to open it in the block viewer (see Figure 6).

   Note: Turn off "Routes" to make it easier to view the block RAMs.

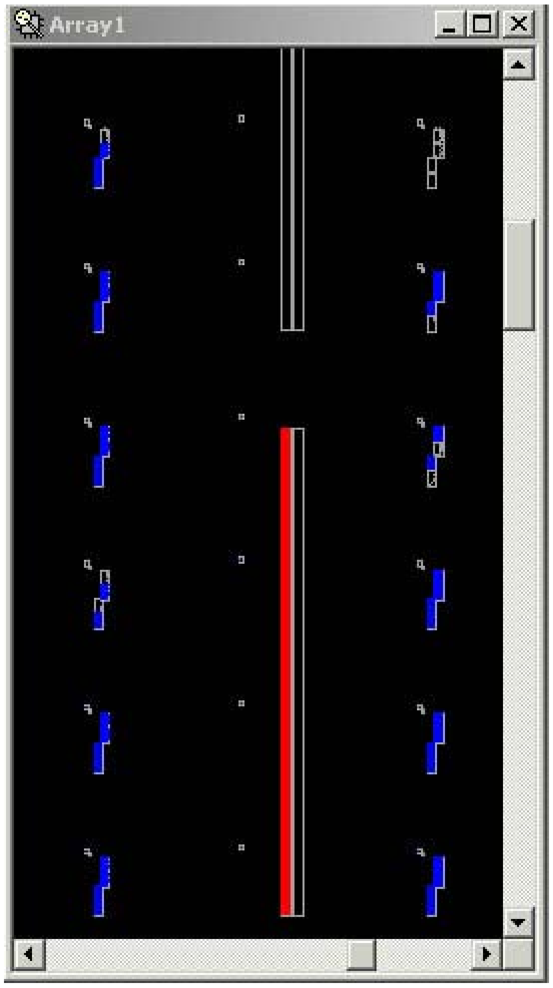
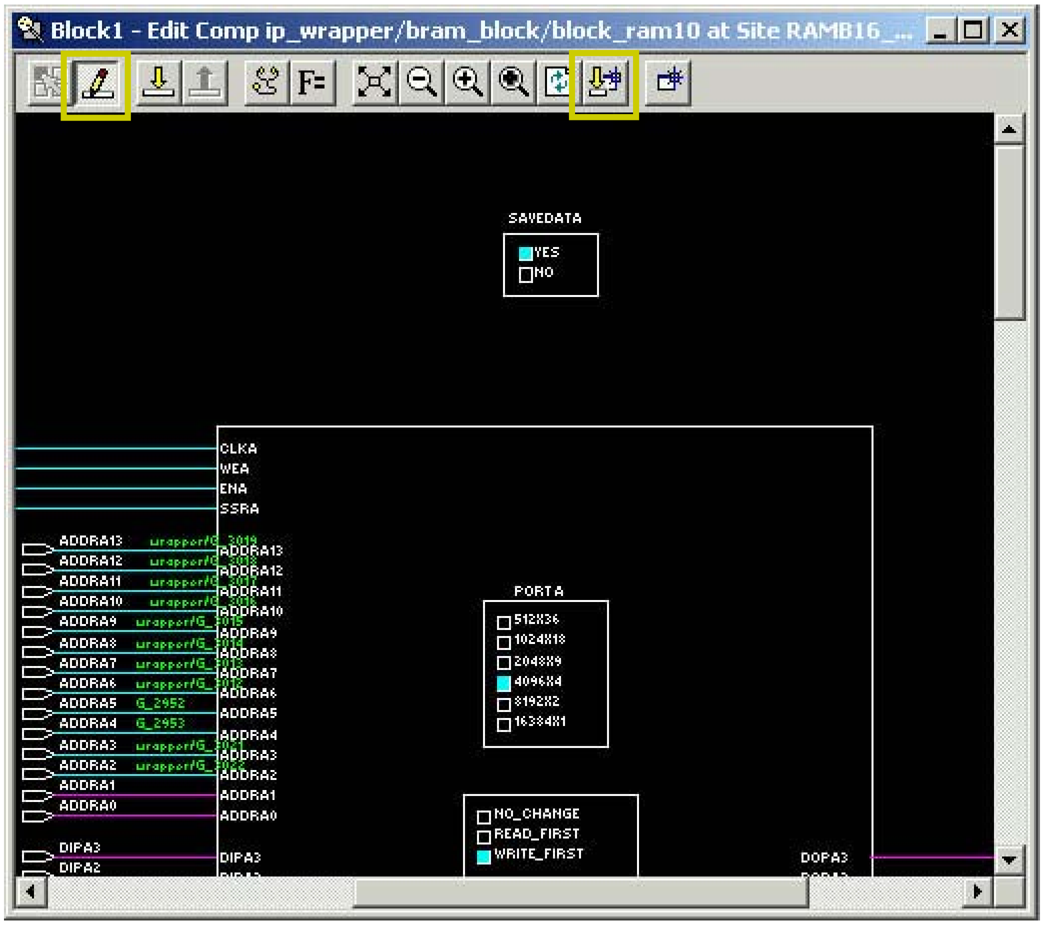


Figure 6: **Selected Block**

3. Click on the “Begin Editing” Button (see Figure 7).
4. Set the SaveData bit to “yes” (see Figure 7).
5. Click on the “Saves Changes and Closes Window” Button (see Figure 7).



Block1 - Edit Comp ip_wrapper/bram_block/block_ram10 at Site RAMB16_...

SAVEDATA
[✓ YES] NO

PORTA
512X36
1024X18
2048X9
[✓ 4096X4]
8192X2
16384X1

NO_CHANGE
READ_FIRST
[✓ WRITE_FIRST]

Figure 7: **Edit Screen**

6. Exit *FPGA_Editor*.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
05/17/02	1.0	Initial Xilinx release.
11/25/03	1.1	Updated constraints for ISE 6.1i. Renamed Small-bit Manipulation to Difference-Based. Added “Appendix A” (SaveData). Restructured Bitstream Generation and Usage sections.
09/09/04	1.2	Corrected errors in Steps 5 and 6 of the “Checklist for Initial Budgeting (Floorplanned and Other .ucf Constraints)” in “Appendix C”.
12/03/07	2.0	Obsoleted and removed module-based partial reconfiguration.