ORCA Series 4 Quad-Port Embedded Block RAM

August 2002 Technical Note TN1016

Introduction

The ORCA® Series 4 FPGA platform provides embedded block RAM (EBR) macrocells to compliment it's distributed PFU RAM. By using ORCA Series 4 EBR, designers can realize the benefits of system-on-a- chip (SoC) and intellectual property (IP) reuse to quickly deliver their end product to market. The ORCA EBR delivers several configurable blocks of memory based embedded IP. These blocks include quad-port RAM, dual-port RAM, FIFO memory, content addressable memory (CAM), and multipliers. This application note provides the details required to use the ORCA Series 4 EBR as a true quad-port memory.

ORCA Series 4 EBR Quad-Port RAM Features

- Quad port EBR basic configuration is 512 x 18 providing 9 kbits of storage per EBR.
- Embedded logic is provided to optionally connect 2 EBR blocks together with no FPGA based glue logic.
- Multiple (byte wide) write enables per port provides a variety of address depth and data width combinations for each EBR with minimal FPGA based glue logic.
- Up to 18 bit data width per EBR allows two extra bits of storage per data word for control signals.
- Four clocks and four chip selects per EBR allow independent operation of write and read channels for each port.
- Supports write through, read through, or read pipeline SRAM operations.
- Embedded arbitration logic is provided in quad- port EBR to prevent write collisions.

Related Document

ispLEVER™ Libraries Manual, included with the Project Navigator software documentation.

Scalability

Each ORCA Series 4 EBR block is configurable as a 512 x 18 quad-port RAM. Additional embedded logic exists to allow two EBR blocks to be combined together to build a 1024 x 18 quad-port RAM with no additional user logic required.

Byte write enables and chip selects can be used as the extra write address bits to build deeper/narrower quad-port RAM:

- 1024 x 9-consumes one block RAM + user logic
- 2048 x 9-consumes two block RAM + user logic

User logic can be built with traditional look up table (LUT) logic or by using ORCA's supplemental logic interconnect cell (SLIC) logic for fast wide decode functions. Figure 1 shows an example of how to build a 1024 x 9 quad-port RAM using one Series 4 EBR and FPGA glue logic.

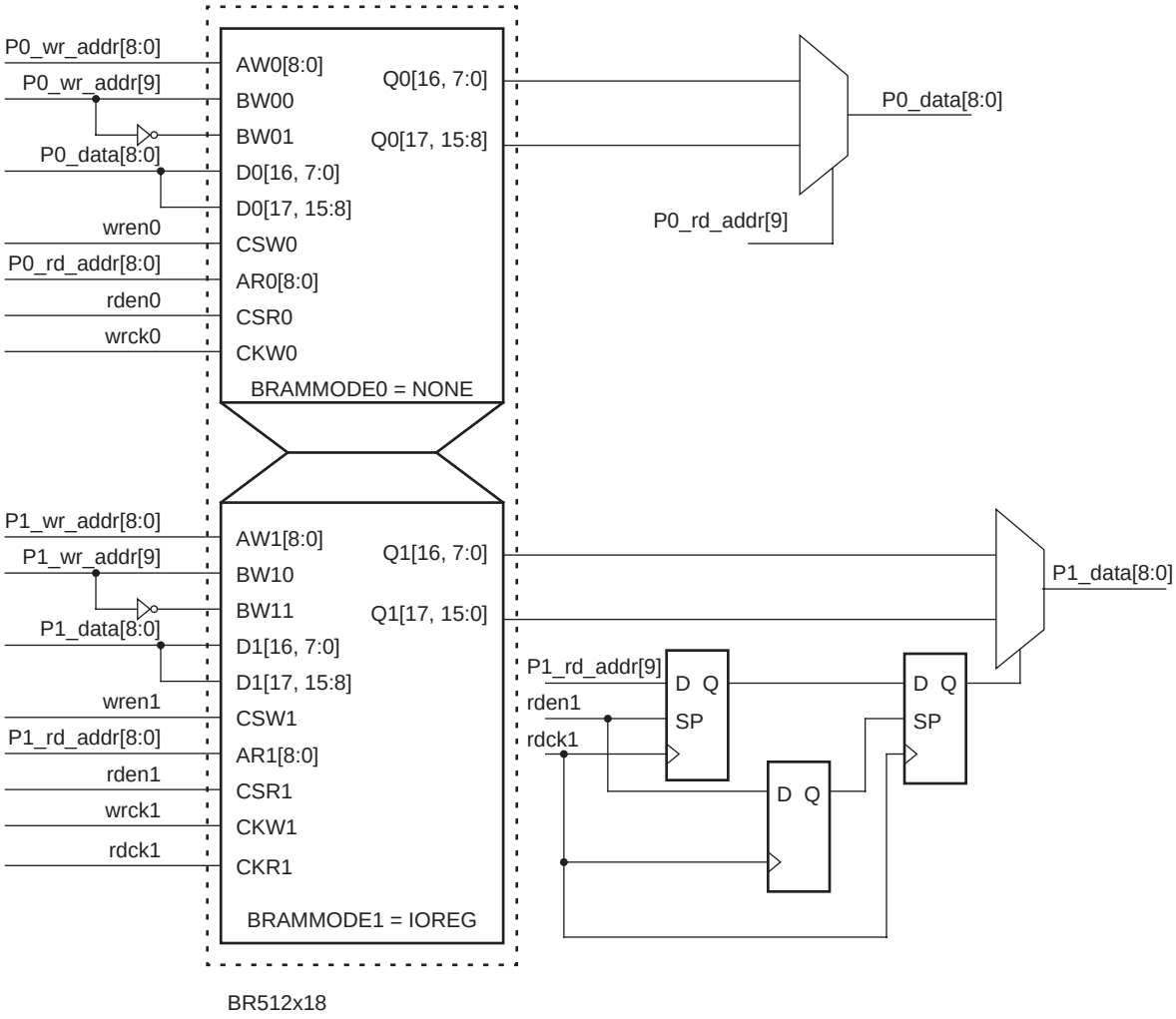
The read path for port 0 is fully asynchronous (controlled by the BRAMMODE0 = NONE attribute). As a result, the required FPGA glue logic is an inverter between port 0 write address bit 9 and the BW01 (upper byte wide write enable port 0) pin. Also required, is a 9 bit wide 2 to 1 multiplexer on the data outputs using port 0 read address bit 9 as the select signal.

The read path for port 1 has registers on the read address inputs and the data outputs (controlled by the BRAMMODE1 = IOREG attribute), thus the FPGA glue logic for port 1 is a little more complicated. An inverter is required between bit 9 of port 1 write address and the BW11 (upper byte-wide write enable port 1) pin. The read path requires 2 registers on port 1 read address bit 9 to make sure that this signal multiplexes the proper data out values. (This accounts for the 2 cycle read path pipeline for port 1 of the quad-port RAM). If the design uses the pipelined feature of the read chip select as well (i.e., using a single cycle chip select pulse as opposed to holding

CSR1 high for the duration of the transaction) then a single pipeline register is also required on the read enable signal so that it can control the clock enable on the second stage pipeline register on the ninth read address bit.

Additionally, a similar arrangement of FPGA glue logic could be used to create a 2048 x 9 using two quad-port EBR configured in 1024 x 18 mode.

Figure 1. 1024 x 9 Quad-Port RAM Using (1) 512 x 18 ORCA Series 4 EBR



Library Elements

There are two quad-port EBR elements in the ispLEVER Project Navigator design libraries.

- BR512x18
- BR1024x18

These elements can be connected together to build wider/deeper RAMs as required for the user's design.

The following is a description of the signal pins and available properties for these elements.

Properties

- BRAMMODE0: NOREG (default), INREG, OUTREG, IOREG
- BRAMMODE1: NOREG (default), INREG, OUTREG, IOREG
- ARBITERMODE: TRUE, FALSE (default)

BRAMMODE[0|1]
This Block RAM Mode (BRAMMODE) property determines how the input and output should be configured for each embedded memory block.

- NOREG-asynchronous read address inputs and data outputs (default)
- INREG-read address inputs latched on rising edge of CKR[0|1] data outputs not registered
- OUTREG-read address inputs not registered data outputs latched on rising edge of CKR[0|1]
- IOREG-read address inputs and data outputs latched on rising edge of CKR[0|1] 2 cycle read access

Arbitermode
This Arbitermode property sets the internal arbitration associated with the entire memory block.

- TRUE-embedded arbiter enabled
- FALSE-embedded arbiter disabled (default)

Signal Definitions

Clock-CK[R|W][0|1]
Each quad-port block RAM provides four clock inputs. This provides synchronous operation of all four ports independent from each other.

These clocks are positive edge triggered only.

Note: Read clock CKR[0|1] signals only function if the optional read address registers and/or dataout registers are enabled by the user.

Chip Select-CS[R|W][0|1]
The chip select pins (four total) act independently to enable or disable read or write functionality for the corresponding port.

These signals are active-high only.

Notes for read chip selects:

1. CSR[0:1] signals only function if the optional read address registers and/or dataout registers are enabled by the user.
2. If both read address and output registers are used, the CSR[0:1] signals are pipelined within the EBR block to allow the user to pulse the read chip select signal at the start of the read cycle. It is not necessary to hold the read chip select high for the entire duration of a two clock cycle pipelined read operation.

Byte Write Enable-BW[0|1](1:0)
Allows user to enable/disable write operations on a byte wide basis for each port. Since the quad-port RAM block is 18 bits wide, each byte write enable signal controls 9 bits as follows:

- BW[0|1]0-write enable for data bits [16, 7, 6, 5, 4, 3, 2, 1, 0]
- BW[0|1]1-write enable for data bits [17, 15, 14, 13, 12, 11, 10, 9, 8]

These signals are active-high only.

Address Signals-A[R|W][0|1](n-1:0)
Points to the SRAM location for a read or write operation. n represents the width of the bus based on the number of data words for the RAM.

512 words n = 9

1024 words n = 10

Read address signals can be optionally registered upon entering the quad-port block to allow for pipelining. This feature is configurable per port and controlled by a property attached to the library element (INREG or IOREG).

Data In Signals-D[1|0](17:0)
Data to be written to the RAM location referenced by the corresponding write address signals.

Data Out Signals-Q[1|0](17:0)
Data contained at the RAM address location is pointed to by the corresponding port's read address signals. Registers are optionally available on these signals to allow for pipelining. This feature is configurable per port and controlled by a property attached to the library element (OUTREG or IOREG).

BUSY
Signal provided by the quad-port block RAM's embedded arbiter to indicate a write collision was detected and the RAM was unable to write to port 0. This signal is active-high and is activated asynchronously to the CKW[0|1] clocks. Deactivation is synchronous to CKW0. When the embedded arbiter is turned off, this signal is zero.

Quad-Port Timing Diagrams

Figure 2. Read Timing Diagram BRAMMODE[0|1] = INREG or OUTREG

(1) Indicates the timing if BRAMMODE[0|1] = INREG
(2) Indicates the timing if BRAMMODE[0|1] = OUTREG

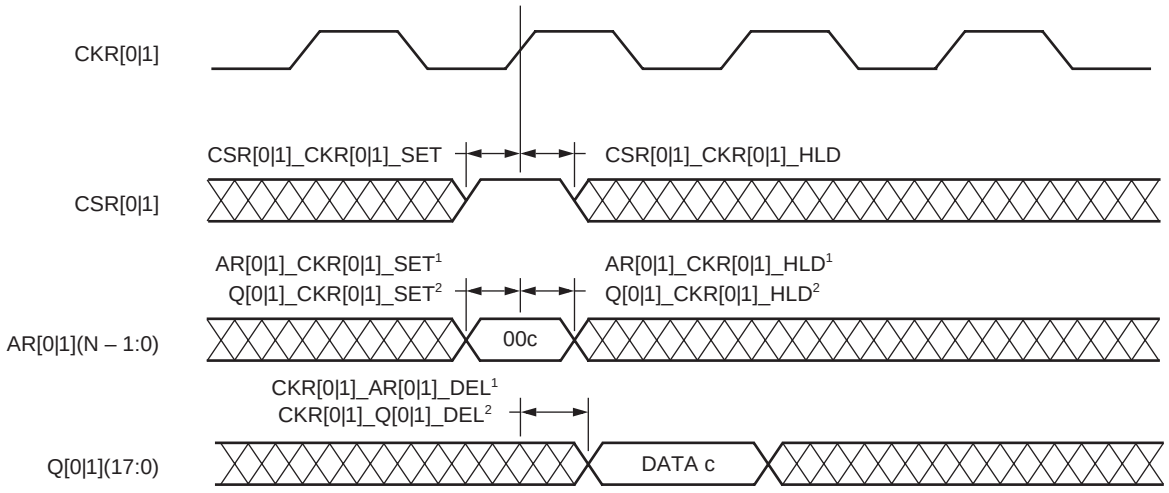


Figure 3. Read Timing Diagram BRAMMODE[0|1] = IOREG

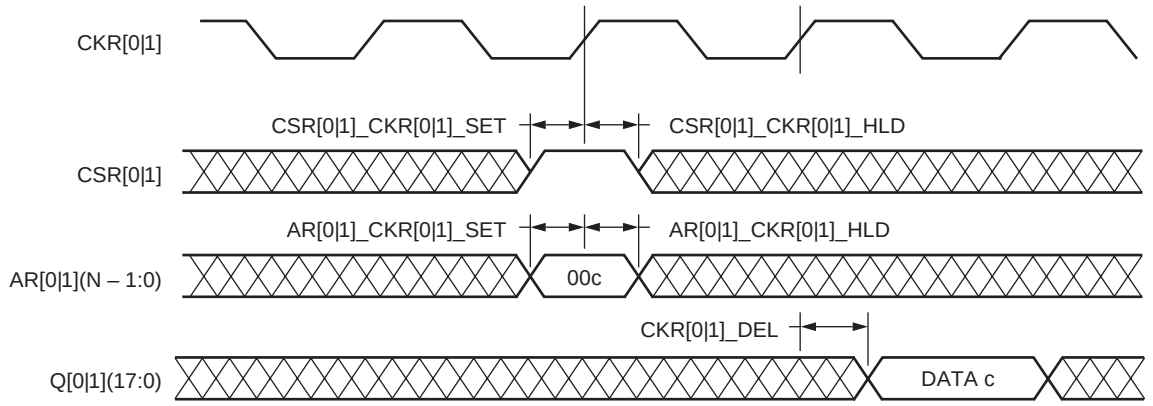
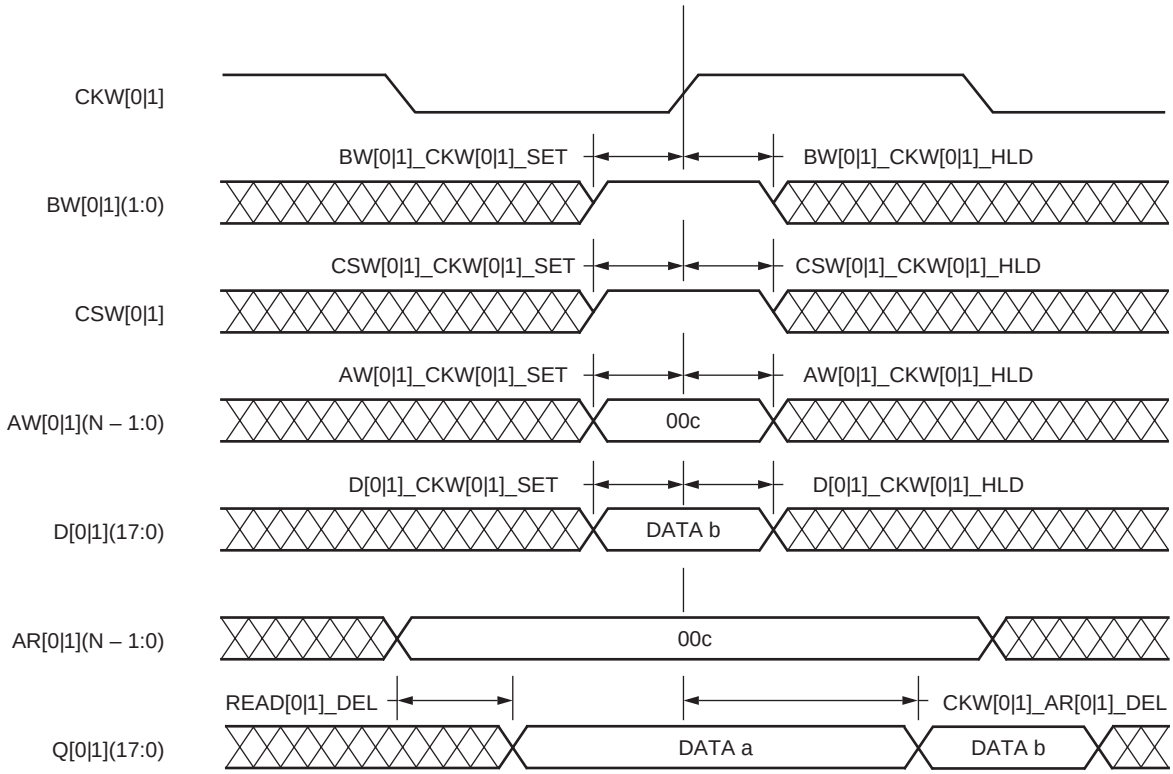


Figure 4. Write Cycle Timing with Write Through (Asynchronous Read) BRAMMODE[0|1] = NONE



Embedded Arbitration Logic

The ORCA Series 4 quad-port EBR contains optionally enabled arbitration logic. This logic is designed to give priority to write operation for port 1 and prevent write operation for port 0 if address collision occurs. The arbiter also provides an active-high BUSY signal which indicates that a collision has occurred and that a write operation was prevented on port 0. Arbitration logic is provided for a warning that a collision occurred and it will also block both ports from writing at the same time. This will be a priority scheme where port 1 always wins. Since port 1 is also the system bus port, if in system bus mode, it will always be able to write. If the arbiter is turned off, both ports can write at the same time and the data will be UNKNOWN. The arbiter will be active when the two write ports have the same address and both CSW signals are high. The BUSY0 is Active High and is the output of the arbiter. A High means that a problem occurred. The BUSY0 is asynchronous with the CKW[1:0] clock.

If two writes collide, the system does not really know what is in the RAM. The arbitration provides some protection so that at least one correct write operation takes place.

The arbiter activates when port 0 and port 1 write addresses are equal and write chip selects for both ports 0 and 1 are active. The condition is given by

(AW0 == AW1) and CSW[0] and CSW[1]

When active, the embedded arbiter performs the following actions:

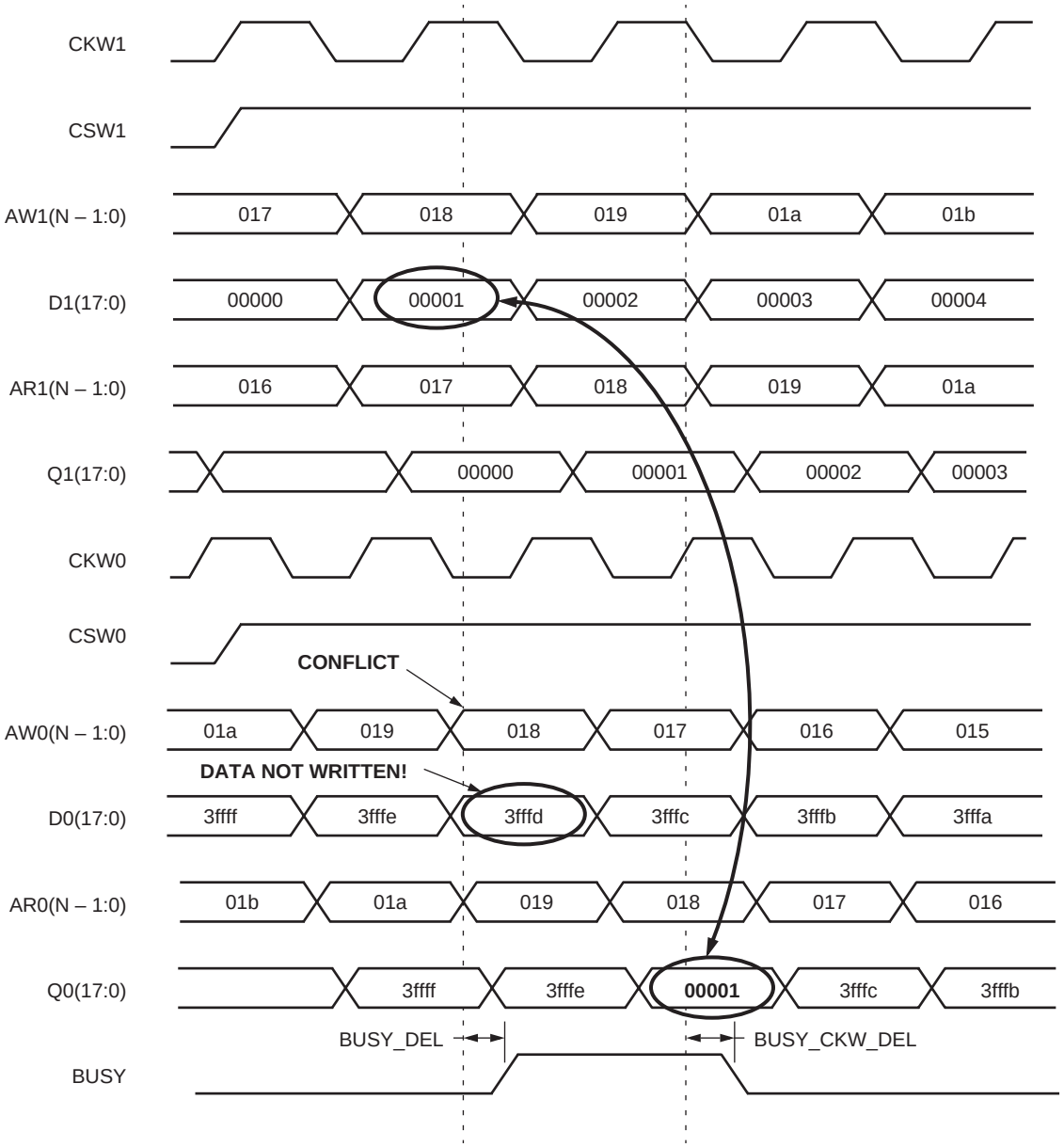
- Executes write as requested by port 1
- Prevents write as requested by port 0
- Sets BUSY to 1 (asynchronously to both CKW rising edges)

The write protection will be removed on the next rising port 0 clock after the above condition is removed. A write on port 0 may occur on the next clock. To avoid address matches during address transitions, it is recommended that CSW[1:0] be disabled during the address changes. During System Bus writes, CSW[1] is always high, so BUSY may falsely disable the port 0 writes if CSW[0] is left active.

NOTE: if CKR[0] is not continuous, then BUSY will not reset until the next time CKR goes high.

The arbitration logic may be disabled by CFG. If the logic is turned off, it is up to the user to avoid write collisions. If both ports write at the same time, data in that word may be corrupted. If the arbiter is turned off, the BUSY signals will always be set to "0".

Figure 5. Arbitration Timing Diagram



Initialization

The ORCA quad-port block RAM can be preloaded with data during device configuration. This is accomplished by attaching the INITVAL_XX property to the library elements in the design netlist.

The INITVAL_XX property uses 5 hex characters to represent the 18-bit data output for each address location. The length of a single INITVAL_XX property is limited to 80 hex characters in length - which means that a single INITVAL_XX property will initialize 80/5 or 16 address locations in descending order.

Example:

INITVAL_00 = "0 x <data_at_address_00f><data_at_address_00e>...<data_at_address_000>"

This works out to a total of 32 INITVAL_XX properties to fully initialize a BR512 x 18 and 64 INITVAL_XX properties to preload a BR1024 x 18 element.

Table 1. RAM Initval Properties

Property	Memory Location
INITVAL_00	15 to 0
INITVAL_01	31 to 16
. . .	. . .
INITVAL_1F	511 to 495
. . .	. . .
INITVAL_3F	1023 to 1007

Note: Initval does not work with using the Microprocessor Interface (MPI) device configuration. Refer to ORCA MPI and System Bus applications note for workaround.

ispLEVER Module/IP Manager

ispLEVER Module/IP Manager is a software tool used to build special configurable function blocks optimized for the ORCA architecture. ispLEVER Module/IP Manager generates structural Verilog, VHDL* and/or electronic design interchange format (EDIF) netlists for instantiation into the user's design. These netlists can be used for simulation using the ORCA simulation libraries and can be synthesized by the major FPGA synthesis vendors. ispLEVER Module/IP Manager provides several options for the user to build various sizes of EBR quad-port RAM. ispLEVER Module/IP Manager supports the initialization of the EBR quad-port RAM by using a memory file. The memory file allows the user to provide initialization data for the RAM in a relatively straight forward manner. ispLEVER Module/IP Manager will then generate the proper INITVAL_XX properties and attach them to the elements in the netlist.

Please refer to the appendix for examples of ispLEVER Module/IP Manager generated Verilog and VHDL reference designs, as well as an example .mem file.

Block RAM Site Naming and Location Conventions

In the ORCA Series 4 FPGA, block RAM sites are organized along the top and the bottom of the device. The number of sites is dependent on the size of the FPGA being used. Table 2 provides the number of block RAM sites for the FPGAs in the Series 4 family.

Table 2. Number of EBR per FPGA

Device	EBR Blocks (Top/Bottom)	EBR kbits
OR4E2	8 (4/4)	74
OR4E4	12 (6/6)	111
OR4E6	16 (8/8)	147

Block RAM Sites

The EBR site names on the top edge of the ORCA Series 4 FPGA/FPSC device are: (left to right)

RAM1024_0-RAM512_1-RAM1024_2-RAM512_3- . . .

The EBR site names on the bottom edge of the ORCA Series 4 FPGA/FPSC device are as follows (left to right):

RAM1024_32-RAM512_33-RAM1024_34-RAM512_35- . . .

For example: an OR4E2 device will have eight sites available for embedded block RAM placement. Those site names and locations are as follows:

Top row (left to right):

RAM1024_0-RAM512_1-RAM1024_2-RAM512_3

Bottom row (left to right):

RAM1024_32-RAM512_33-RAM1024_34-RAM1024_35

Placement Rules

- A 512x18 can be located to any site (RAM1024_x or RAM512_x)
- A 1024x18 can be located in only the RAM1024_x sites. The 1024x18 block will, however, consume two sites: RAM1024_(x) and the adjacent site to the right, RAM512_(x + 1).

Legal and illegal LOCATE preference examples:

Quad-Port 1024x18 blocks

- LOCATE COMP "U1_BR1024x18" SITE "RAM1024_0"; //legal placement
- LOCATE COMP "U1_BR1024x18" SITE "RAM512_1"; //illegal placement

Quad-Port 512x18 blocks

- LOCATE COMP "U1_BR512x18" SITE "RAM1024_32"; //legal placement
- LOCATE COMP "U2_BR512x18" SITE "RAM512_33"; //legal placement

Two Quad-Port blocks (1024x18 and 512x18)

- LOCATE COMP "U1_BR1024x18" SITE "RAM1024_0"; //legal placement (2 sites)
- LOCATE COMP "U2_BR512x18" SITE "RAM512_1"; //illegal placement

The next available site for COMP U2_BR512 x 18 would be RAM1024_2:

- LOCATE COMP "U2_BR512x18" SITE "RAM1024_2"; //legal placement

Note: Locating RAM blocks is not required, the software tool will automatically place the blocks if they are not located.

Conclusion

ORCA Series 4 provides a full-featured, quad-port RAM using its embedded block RAM (EBR) technologies. By using ORCA Series 4 EBR, designers can realize the benefits of system-on-a- chip (SoC) and intellectual property (IP) reuse to quickly deliver their end product to market.

Appendix A. ispLEVER Module/IP Manager Flow

The following windows illustrate how to setup a Quad Port Embedded Block RAM using the ispLEVER Module/IP Manager.

Figure 6. ispLEVER Module/IP Manager Device Select Screen

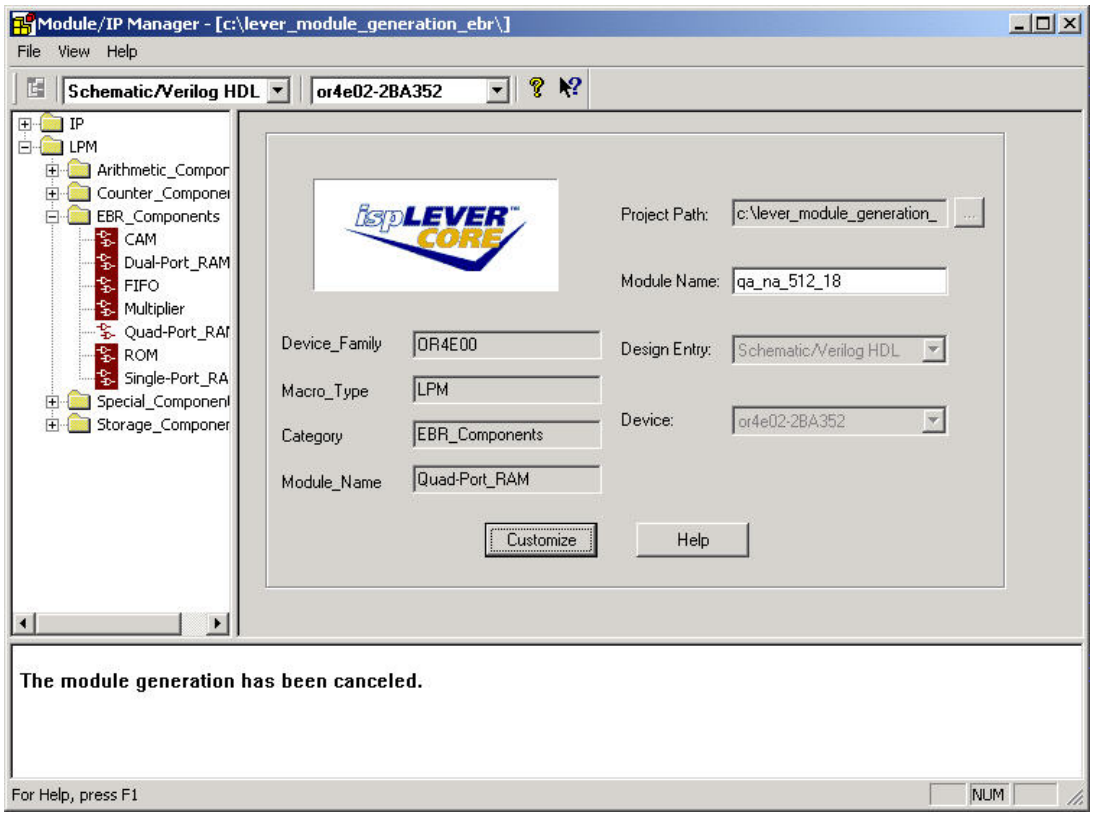


Figure 7. Quad Port Block RAM Setup Screen

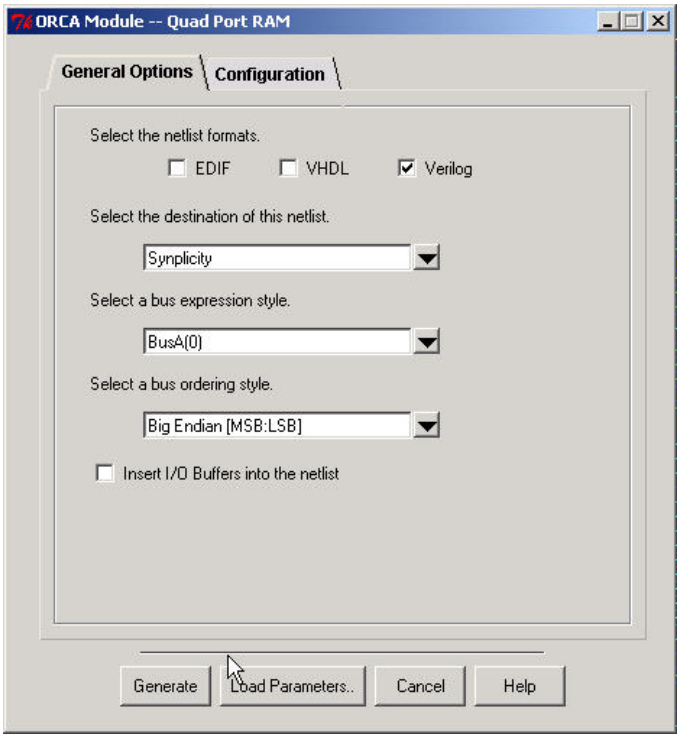
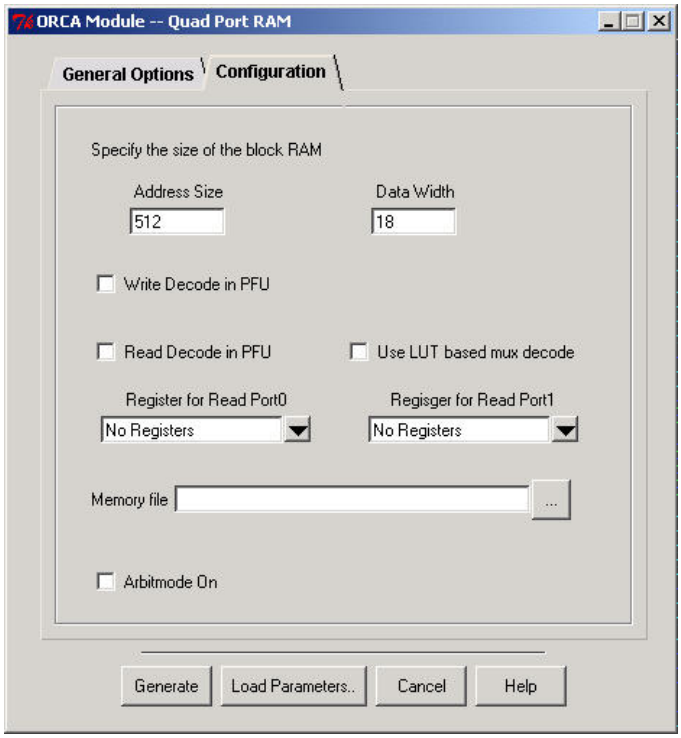


Figure 8. Quad Port RAM Memory Setup



ispLEVER Module/IP Manager Settings

Size of the RAM
The address size is a multiple of 512. This is because the basic library element has a depth of 512 when used as a quad port RAM. The data width may be any value. The size determines the number of block RAMs and PFUs used.

Write Decode in PFU
By default, for sizes that require more than one block RAM, individual write enables for the RAMs are generated using SLIC logic.

Read Decode in PFU
By default, for sizes that require more than one block RAM, individual read enables for the RAMs are generated using SLIC logic.

Dataout Decodes
By default, ispLEVER software generates tristate based decodes for the output data. The enables are generated in SLICs. Selecting the Read Decode in PFU creates these enables using PFU logic which uses LUT based mux decodes. In this case, no tristates are used. These two options are independent of each other. It is recommended that the user selects the Read Decode option.

Register Options
The register options are independent for each port. The option determines the attributes on the library elements and any additional registering required.

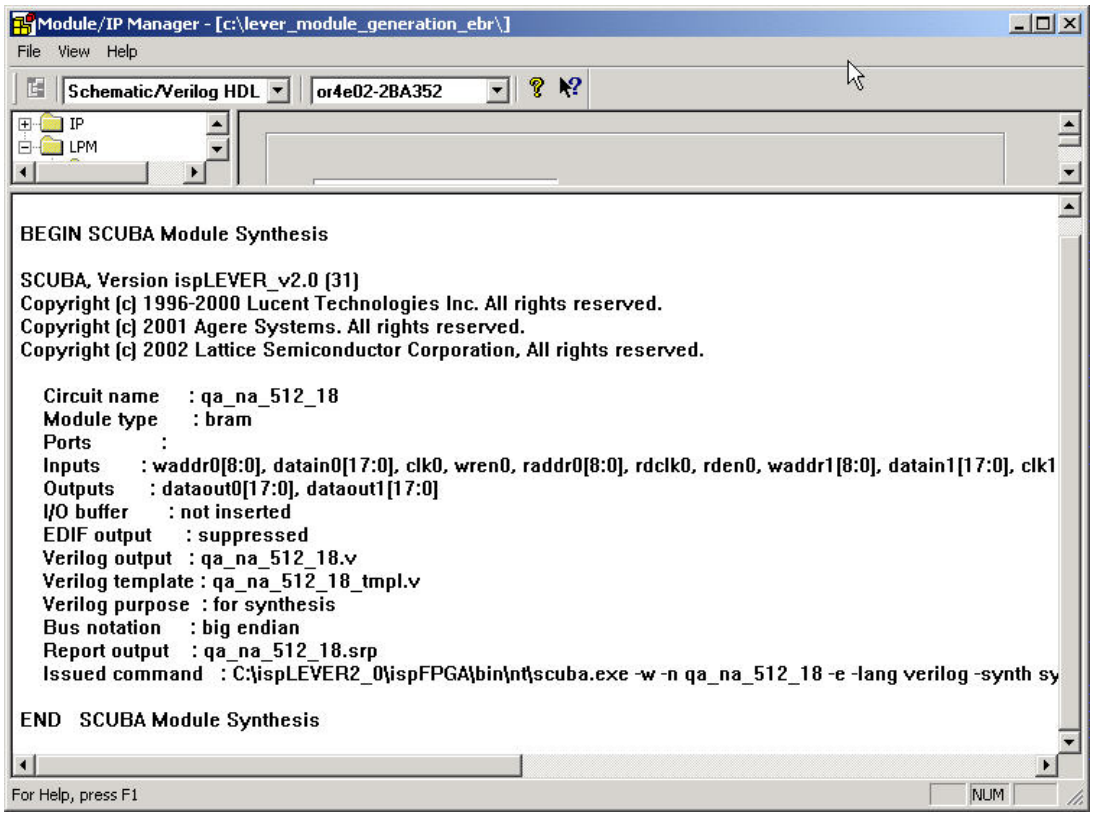
Memory File
An optional memory file (*.mem extension) may be specified that determines the initial value of the block RAM. The syntax of this file is the same as the ispLEVER Module/IP Manager memory file for the PFU based RAM/ROM.

Arbitmode On
By default, the arbitration mode on the quad port is disabled. When selected, a top level busy port is generated and the ARBITERMODE attribute is also set.

The Ram Settings are illustrated in Figure 8.

ispLEVER Module/IP Manager File Output
Depending on the nelist format selection (see Figure 7), the output can be edif and/or HDL source code with a template file for instantiation. A report file is always generated after each netlist run. The generated report file has an srp, ispLEVER Module/IP Manager Report, extension.

Figure 9. Module Generation Output



Appendix B. Verilog Example of 512x18 Quad-Port EBR RAM With Mem File

```
/* Verilog netlist generated by SCUBA ORCA Foundry 2001 Production (114) */
/* C:\ORCA\bin\nt\scuba -w -lang verilog -n qp_na_512x18 -synth synplify -bus_exp 1 -bb -type bram -
wp 11 -rp 1111 -num_rows 512 -data_width 18 -read_reg0 noreg -read_reg1 noreg -memfile
C:\EBR\source\memory_files\qp_512x18_ram.mem  */
/* Wed Mar 06 13:57:22 2002 */

`timescale 1 ns / 100 ps
module qp_na_512x18 (waddr0, datain0, clk0, wren0, raddr0, rdclk0, rden0,
    waddr1, datain1, clk1, wren1, raddr1, rdclk1, rden1, dataout0,
    dataout1);
    input [8:0] waddr0;
    input [17:0] datain0;
    input clk0;
    input wren0;
    input [8:0] raddr0;
    input rdclk0;
    input rden0;
    input [8:0] waddr1;
    input [17:0] datain1;
    input clk1;
    input wren1;
    input [8:0] raddr1;
    input rdclk1;
    input rden1;
    output [17:0] dataout0;
    output [17:0] dataout1;


    VHI scuba_vhi_inst (.Z(scuba_vhi));

    BR512X18 qp_na_512x180_0_0 (.AW18(waddr1[8]), .AW17(waddr1[7]), .AW16(waddr1[6]),
        .AW15(waddr1[5]), .AW14(waddr1[4]), .AW13(waddr1[3]), .AW12(waddr1[2]),
        .AW11(waddr1[1]), .AW10(waddr1[0]), .AW08(waddr0[8]), .AW07(waddr0[7]),
        .AW06(waddr0[6]), .AW05(waddr0[5]), .AW04(waddr0[4]), .AW03(waddr0[3]),
        .AW02(waddr0[2]), .AW01(waddr0[1]), .AW00(waddr0[0]), .D117(datain1[17]),
        .D116(datain1[16]), .D115(datain1[15]), .D114(datain1[14]), .D113(datain1[13]),
        .D112(datain1[12]), .D111(datain1[11]), .D110(datain1[10]), .D19(datain1[9]),
        .D18(datain1[8]), .D17(datain1[7]), .D16(datain1[6]), .D15(datain1[5]),
        .D14(datain1[4]), .D13(datain1[3]), .D12(datain1[2]), .D11(datain1[1]),
        .D10(datain1[0]), .D017(datain0[17]), .D016(datain0[16]), .D015(datain0[15]),
        .D014(datain0[14]), .D013(datain0[13]), .D012(datain0[12]), .D011(datain0[11]),
        .D010(datain0[10]), .D09(datain0[9]), .D08(datain0[8]), .D07(datain0[7]),
        .D06(datain0[6]), .D05(datain0[5]), .D04(datain0[4]), .D03(datain0[3]),
        .D02(datain0[2]), .D01(datain0[1]), .D00(datain0[0]), .BW11(scuba_vhi),
        .BW10(scuba_vhi), .BW01(scuba_vhi), .BW00(scuba_vhi), .AR18(raddr1[8]),
        .AR17(raddr1[7]), .AR16(raddr1[6]), .AR15(raddr1[5]), .AR14(raddr1[4]),
        .AR13(raddr1[3]), .AR12(raddr1[2]), .AR11(raddr1[1]), .AR10(raddr1[0]),
        .AR08(raddr0[8]), .AR07(raddr0[7]), .AR06(raddr0[6]), .AR05(raddr0[5]),
        .AR04(raddr0[4]), .AR03(raddr0[3]), .AR02(raddr0[2]), .AR01(raddr0[1]),
        .AR00(raddr0[0]), .CKW1(clk1), .CKW0(clk0), .CKR1(rdclk1), .CKR0(rdclk0),
        .CSW1(wren1), .CSW0(wren0), .CSR1(rden1), .CSR0(rden0), .Q117(dataout1[17]),
        .Q116(dataout1[16]), .Q115(dataout1[15]), .Q114(dataout1[14]), .Q113(dataout1[13]),
        .Q112(dataout1[12]), .Q111(dataout1[11]), .Q110(dataout1[10]), .Q19(dataout1[9]),
        .Q18(dataout1[8]), .Q17(dataout1[7]), .Q16(dataout1[6]), .Q15(dataout1[5]),
        .Q14(dataout1[4]), .Q13(dataout1[3]), .Q12(dataout1[2]), .Q11(dataout1[1]),
        .Q10(dataout1[0]), .Q017(dataout0[17]), .Q016(dataout0[16]), .Q015(dataout0[15]),
        .Q014(dataout0[14]), .Q013(dataout0[13]), .Q012(dataout0[12]), .Q011(dataout0[11]),
```

```
        .Q010(dataout0[10]), .Q09(dataout0[9]), .Q08(dataout0[8]), .Q07(dataout0[7]),
        .Q06(dataout0[6]), .Q05(dataout0[5]), .Q04(dataout0[4]), .Q03(dataout0[3]),
        .Q02(dataout0[2]), .Q01(dataout0[1]), .Q00(dataout0[0]), .BUSY()) /* synthesis
INITVAL_1F="0x301ff301fe301fd301fc301fb301fa301f9301f8301f7301f6301f5301f4301f3301f2301f1301f0" */ /* synthesis
INITVAL_1E="0x301ef301ee301ed301ec301eb301ea301e9301e8301e7301e6301e5301e4301e3301e2301e1301e0" */ /* synthesis
INITVAL_1D="0x301df301de301dd301dc301db301da301d9301d8301d7301d6301d5301d4301d3301d2301d1301d0" */ /* synthesis
INITVAL_1C="0x301cf301ce301cd301cc301cb301ca301c9301c8301c7301c6301c5301c4301c3301c2301c1301c0" */ /* synthesis
INITVAL_1B="0x301bf301be301bd301bc301bb301ba301b9301b8301b7301b6301b5301b4301b3301b2301b1301b0" */ /* synthesis
INITVAL_1A="0x301af301ae301ad301ac301ab301aa301a9301a8301a7301a6301a5301a4301a3301a2301a1301a0" */ /* synthesis
INITVAL_19="0x3019f3019e3019d3019c3019b3019a30199301983019730196301953019430193301923019130190" */ /* synthesis
INITVAL_18="0x3018f3018e3018d3018c3018b3018a30189301883018730186301853018430183301823018130180" */ /* synthesis
INITVAL_17="0x2017f2017e2017d2017c2017b2017a20179201782017720176201752017420173201722017120170" */ /* synthesis
INITVAL_16="0x2016f2016e2016d2016c2016b2016a20169201682016720166201652016420163201622016120160" */ /* synthesis
INITVAL_15="0x2015f2015e2015d2015c2015b2015a20159201582015720156201552015420153201522015120150" */ /* synthesis
INITVAL_14="0x2014f2014e2014d2014c2014b2014a20149201482014720146201452014420143201422014120140" */ /* synthesis
INITVAL_13="0x2013f2013e2013d2013c2013b2013a20139201382013720136201352013420133201322013120130" */ /* synthesis
INITVAL_12="0x2012f2012e2012d2012c2012b2012a20129201282012720126201252012420123201222012120120" */ /* synthesis
INITVAL_11="0x2011f2011e2011d2011c2011b2011a20119201182011720116201152011420113201122011120110" */ /* synthesis
INITVAL_10="0x2010f2010e2010d2010c2010b2010a20109201082010720106201052010420103201022010120100" */ /* synthesis
INITVAL_0F="0x100ff100fe100fd100fc100fb100fa100f9100f8100f7100f6100f5100f4100f3100f2100f1100f0" */ /* synthesis
INITVAL_0E="0x100ef100ee100ed100ec100eb100ea100e9100e8100e7100e6100e5100e4100e3100e2100e1100e0" */ /* synthesis
INITVAL_0D="0x100df100de100dd100dc100db100da100d9100d8100d7100d6100d5100d4100d3100d2100d1100d0" */ /* synthesis
INITVAL_0C="0x100cf100ce100cd100cc100cb100ca100c9100c8100c7100c6100c5100c4100c3100c2100c1100c0" */ /* synthesis
INITVAL_0B="0x100bf100be100bd100bc100bb100ba100b9100b8100b7100b6100b5100b4100b3100b2100b1100b0" */ /* synthesis
INITVAL_0A="0x100af100ae100ad100ac100ab100aa100a9100a8100a7100a6100a5100a4100a3100a2100a1100a0" */ /* synthesis
INITVAL_09="0x1009f1009e1009d1009c1009b1009a10099100981009710096100951009410093100921009110090" */ /* synthesis
INITVAL_08="0x1008f1008e1008d1008c1008b1008a10089100881008710086100851008410083100821008110080" */ /* synthesis
INITVAL_07="0x0007f0007e0007d0007c0007b0007a00079000780007700076000750007400073000720007100070" */ /* synthesis
INITVAL_06="0x0006f0006e0006d0006c0006b0006a00069000680006700066000650006400063000620006100060" */ /* synthesis
INITVAL_05="0x0005f0005e0005d0005c0005b0005a00059000580005700056000550005400053000520005100050" */ /* synthesis
INITVAL_04="0x0004f0004e0004d0004c0004b0004a00049000480004700046000450004400043000420004100040" */ /* synthesis
INITVAL_03="0x0003f0003e0003d0003c0003b0003a00039000380003700036000350003400033000320003100030" */ /* synthesis
INITVAL_02="0x0002f0002e0002d0002c0002b0002a00029000280002700026000250002400023000220002100020" */ /* synthesis
INITVAL_01="0x0001f0001e0001d0001c0001b0001a00019000180001700016000150001400013000120001100010" */ /* synthesis
INITVAL_00="0x0000f0000e0000d0000c0000b0000a00009000080000700006000050000400003000020000100000" */ /* synthesis ARBITER-
MODE="FALSE" */;
endmodule
```

Appendix C. VHDL Example of 512x18 Quad-Port EBR RAM With Mem File

Read Address Registers Enabled for Port 1

Read Address and Data out Registers Enabled for Port 0

```
-- VHDL netlist generated by SCUBA ORCA Foundry 2001 Production (3)
--c:\ORCA\bin\nt\scuba -w -e -lang vhdl -n qp_na_512x18 -synth synplify -bus_exp 1 -bb -type bram -wp
11 -rp 1111 -num_rows 512 -data_width 18 -read_reg0 ioreg -read_reg1 inreg -memfile
C:\EBR\source\memory_files\qp_512x18_ram.mem

-- Wed Mar 06 10:56:44 2002

library IEEE;
use IEEE.std_logic_1164.all;

entity qp_na_512x18 is
    port (waddr0: in  std_logic_vector(8 downto 0);
        datain0: in  std_logic_vector(17 downto 0); clk0: in  std_logic;
        wren0: in  std_logic; raddr0: in  std_logic_vector(8 downto 0);
        rdclk0: in  std_logic; rden0: in  std_logic;
        waddr1: in  std_logic_vector(8 downto 0);
        datain1: in  std_logic_vector(17 downto 0); clk1: in  std_logic;
        wren1: in  std_logic; raddr1: in  std_logic_vector(8 downto 0);
        rdclk1: in  std_logic; rden1: in  std_logic;
        dataout0: out  std_logic_vector(17 downto 0);
        dataout1: out  std_logic_vector(17 downto 0));
end qp_na_512x18;

architecture Structure of qp_na_512x18 is

    -- internal signal declarations
    signal scuba_vhi: std_logic;

    -- local component declarations
    component BR512X18
        port (AW18: in  std_logic; AW17: in  std_logic;
            AW16: in  std_logic; AW15: in  std_logic;
            AW14: in  std_logic; AW13: in  std_logic;
            AW12: in  std_logic; AW11: in  std_logic;
            AW10: in  std_logic; AW08: in  std_logic;
            AW07: in  std_logic; AW06: in  std_logic;
            AW05: in  std_logic; AW04: in  std_logic;
            AW03: in  std_logic; AW02: in  std_logic;
            AW01: in  std_logic; AW00: in  std_logic;
            D117: in  std_logic; D116: in  std_logic;
            D115: in  std_logic; D114: in  std_logic;
            D113: in  std_logic; D112: in  std_logic;
            D111: in  std_logic; D110: in  std_logic; D19: in  std_logic;
            D18: in  std_logic; D17: in  std_logic; D16: in  std_logic;
            D15: in  std_logic; D14: in  std_logic; D13: in  std_logic;
            D12: in  std_logic; D11: in  std_logic; D10: in  std_logic;
            D017: in  std_logic; D016: in  std_logic;
            D015: in  std_logic; D014: in  std_logic;
            D013: in  std_logic; D012: in  std_logic;
            D011: in  std_logic; D010: in  std_logic; D09: in  std_logic;
            D08: in  std_logic; D07: in  std_logic; D06: in  std_logic;
            D05: in  std_logic; D04: in  std_logic; D03: in  std_logic;
            D02: in  std_logic; D01: in  std_logic; D00: in  std_logic;
```

```
        BW11: in  std_logic; BW10: in  std_logic;
        BW01: in  std_logic; BW00: in  std_logic;
        AR18: in  std_logic; AR17: in  std_logic;
        AR16: in  std_logic; AR15: in  std_logic;
        AR14: in  std_logic; AR13: in  std_logic;
        AR12: in  std_logic; AR11: in  std_logic;
        AR10: in  std_logic; AR08: in  std_logic;
        AR07: in  std_logic; AR06: in  std_logic;
        AR05: in  std_logic; AR04: in  std_logic;
        AR03: in  std_logic; AR02: in  std_logic;
        AR01: in  std_logic; AR00: in  std_logic;
        CKW1: in  std_logic; CKW0: in  std_logic;
        CKR1: in  std_logic; CKR0: in  std_logic;
        CSW1: in  std_logic; CSW0: in  std_logic;
        CSR1: in  std_logic; CSR0: in  std_logic;
        Q117: out  std_logic; Q116: out  std_logic;
        Q115: out  std_logic; Q114: out  std_logic;
        Q113: out  std_logic; Q112: out  std_logic;
        Q111: out  std_logic; Q110: out  std_logic;
        Q19: out  std_logic; Q18: out  std_logic;
        Q17: out  std_logic; Q16: out  std_logic;
        Q15: out  std_logic; Q14: out  std_logic;
        Q13: out  std_logic; Q12: out  std_logic;
        Q11: out  std_logic; Q10: out  std_logic;
        Q017: out  std_logic; Q016: out  std_logic;
        Q015: out  std_logic; Q014: out  std_logic;
        Q013: out  std_logic; Q012: out  std_logic;
        Q011: out  std_logic; Q010: out  std_logic;
        Q09: out  std_logic; Q08: out  std_logic;
        Q07: out  std_logic; Q06: out  std_logic;
        Q05: out  std_logic; Q04: out  std_logic;
        Q03: out  std_logic; Q02: out  std_logic;
        Q01: out  std_logic; Q00: out  std_logic;
        BUSY: out  std_logic);
end component;
component VHI
    port (Z: out  std_logic);
end component;
attribute INITVAL_1F : string;
attribute INITVAL_1E : string;
attribute INITVAL_1D : string;
attribute INITVAL_1C : string;
attribute INITVAL_1B : string;
attribute INITVAL_1A : string;
attribute INITVAL_19 : string;
attribute INITVAL_18 : string;
attribute INITVAL_17 : string;
attribute INITVAL_16 : string;
attribute INITVAL_15 : string;
attribute INITVAL_14 : string;
attribute INITVAL_13 : string;
attribute INITVAL_12 : string;
attribute INITVAL_11 : string;
attribute INITVAL_10 : string;
attribute INITVAL_0F : string;
attribute INITVAL_0E : string;
attribute INITVAL_0D : string;
attribute INITVAL_0C : string;
attribute INITVAL_0B : string;
```

```
    attribute INITVAL_0A : string;
    attribute INITVAL_09 : string;
    attribute INITVAL_08 : string;
    attribute INITVAL_07 : string;
    attribute INITVAL_06 : string;
    attribute INITVAL_05 : string;
    attribute INITVAL_04 : string;
    attribute INITVAL_03 : string;
    attribute INITVAL_02 : string;
    attribute INITVAL_01 : string;
    attribute INITVAL_00 : string;
    attribute ARBITERMODE : string;
    attribute BRAMMODE1 : string;
    attribute BRAMMODE0 : string;
attribute INITVAL_1F of qp_na_512x180_0_0 : label is
"0x301ff301fe301fd301fc301fb301fa301f9301f8301f7301f6301f5301f4301f3301f2301f1301f0";
attribute INITVAL_1E of qp_na_512x180_0_0 : label is
"0x301ef301ee301ed301ec301eb301ea301e9301e8301e7301e6301e5301e4301e3301e2301e1301e0";
attribute INITVAL_1D of qp_na_512x180_0_0 : label is
"0x301df301de301dd301dc301db301da301d9301d8301d7301d6301d5301d4301d3301d2301d1301d0";
attribute INITVAL_1C of qp_na_512x180_0_0 : label is
"0x301cf301ce301cd301cc301cb301ca301c9301c8301c7301c6301c5301c4301c3301c2301c1301c0";
attribute INITVAL_1B of qp_na_512x180_0_0 : label is
"0x301bf301be301bd301bc301bb301ba301b9301b8301b7301b6301b5301b4301b3301b2301b1301b0";
attribute INITVAL_1A of qp_na_512x180_0_0 : label is
"0x301af301ae301ad301ac301ab301aa301a9301a8301a7301a6301a5301a4301a3301a2301a1301a0";
attribute INITVAL_19 of qp_na_512x180_0_0 : label is
"0x3019f3019e3019d3019c3019b3019a3019930198301973019630195301943019330192301913019 0";
attribute INITVAL_18 of qp_na_512x180_0_0 : label is
"0x3018f3018e3018d3018c3018b3018a3018930188301873018630185301843018330182301813018 0";
attribute INITVAL_17 of qp_na_512x180_0_0 : label is
"0x2017f2017e2017d2017c2017b2017a2017920178201772017620175201742017320172201712017 0";
attribute INITVAL_16 of qp_na_512x180_0_0 : label is
"0x2016f2016e2016d2016c2016b2016a2016920168201672016620165201642016320162201612016 0";
attribute INITVAL_15 of qp_na_512x180_0_0 : label is
"0x2015f2015e2015d2015c2015b2015a2015920158201572015620155201542015320152201512015 0";
attribute INITVAL_14 of qp_na_512x180_0_0 : label is
"0x2014f2014e2014d2014c2014b2014a2014920148201472014620145201442014320142201412014 0";
attribute INITVAL_13 of qp_na_512x180_0_0 : label is
"0x2013f2013e2013d2013c2013b2013a2013920138201372013620135201342013320132201312013 0";
attribute INITVAL_12 of qp_na_512x180_0_0 : label is
"0x2012f2012e2012d2012c2012b2012a2012920128201272012620125201242012320122201212012 0";
attribute INITVAL_11 of qp_na_512x180_0_0 : label is
"0x2011f2011e2011d2011c2011b2011a2011920118201172011620115201142011320112201111011 0";
attribute INITVAL_10 of qp_na_512x180_0_0 : label is
"0x2010f2010e2010d2010c2010b2010a2010920108201072010620105201042010320102201012010 0";
attribute INITVAL_0F of qp_na_512x180_0_0 : label is
"0x100ff100fe100fd100fc100fb100fa100f9100f8100f7100f6100f5100f4100f3100f2100f1100f0";
attribute INITVAL_0E of qp_na_512x180_0_0 : label is
"0x100ef100ee100ed100ec100eb100ea100e9100e8100e7100e6100e5100e4100e3100e2100e1100e0";
attribute INITVAL_0D of qp_na_512x180_0_0 : label is
"0x100df100de100dd100dc100db100da100d9100d8100d7100d6100d5100d4100d3100d2100d1100d0";
attribute INITVAL_0C of qp_na_512x180_0_0 : label is
"0x100cf100ce100cd100cc100cb100ca100c9100c8100c7100c6100c5100c4100c3100c2100c1100c0";
attribute INITVAL_0B of qp_na_512x180_0_0 : label is
"0x100bf100be100bd100bc100bb100ba100b9100b8100b7100b6100b5100b4100b3100b2100b1100b0";
attribute INITVAL_0A of qp_na_512x180_0_0 : label is
"0x100af100ae100ad100ac100ab100aa100a9100a8100a7100a6100a5100a4100a3100a2100a1100a0";
```

```
attribute INITVAL_09 of qp_na_512x180_0_0 : label is
"0x1009f1009e1009d1009c1009b1009a10099100981009710096100951009410093100921009110090";
attribute INITVAL_08 of qp_na_512x180_0_0 : label is
"0x1008f1008e1008d1008c1008b1008a10089100881008710086100851008410083100821008110080";
attribute INITVAL_07 of qp_na_512x180_0_0 : label is
"0x0007f0007e0007d0007c0007b0007a0007900078000770007600075000740007300072000710070";
attribute INITVAL_06 of qp_na_512x180_0_0 : label is
"0x0006f0006e0006d0006c0006b0006a0006900068000670006600065000640006300062000610060";
attribute INITVAL_05 of qp_na_512x180_0_0 : label is
"0x0005f0005e0005d0005c0005b0005a0005900058000570005600055000540005300052000510050";
attribute INITVAL_04 of qp_na_512x180_0_0 : label is
"0x0004f0004e0004d0004c0004b0004a0004900048000470004600045000440004300042000410040";attribute
INITVAL_03 of qp_na_512x180_0_0 : label is
"0x0003f0003e0003d0003c0003b0003a0003900038000370003600035000340003300032000310030";
attribute INITVAL_02 of qp_na_512x180_0_0 : label is
"0x0002f0002e0002d0002c0002b0002a0002900028000270002600025000240002300022000210020";
attribute INITVAL_01 of qp_na_512x180_0_0 : label is
"0x0001f0001e0001d0001c0001b0001a0001900018000170001600015000140001300012000110010";
attribute INITVAL_00 of qp_na_512x180_0_0 : label is
"0x0000f0000e0000d0000c0000b0000a0000900008000070000600005000040000300002000010000";
attribute ARBITERMODE of qp_na_512x180_0_0 : label is "FALSE";
attribute BRAMMODE1 of qp_na_512x180_0_0 : label is "INREG";
attribute BRAMMODE0 of qp_na_512x180_0_0 : label is "IOREG";

begin
    -- component instantiation statements
    scuba_vhi_inst: VHI
        port map (Z=>scuba_vhi);

    qp_na_512x180_0_0: BR512X18
        port map (AW18=>waddr1(8), AW17=>waddr1(7), AW16=>waddr1(6),
        AW15=>waddr1(5), AW14=>waddr1(4), AW13=>waddr1(3),
        AW12=>waddr1(2), AW11=>waddr1(1), AW10=>waddr1(0),
        AW08=>waddr0(8), AW07=>waddr0(7), AW06=>waddr0(6),
        AW05=>waddr0(5), AW04=>waddr0(4), AW03=>waddr0(3),
        AW02=>waddr0(2), AW01=>waddr0(1), AW00=>waddr0(0),
        D117=>datain1(17), D116=>datain1(16), D115=>datain1(15),
        D114=>datain1(14), D113=>datain1(13), D112=>datain1(12),
        D111=>datain1(11), D110=>datain1(10), D19=>datain1(9),
        D18=>datain1(8), D17=>datain1(7), D16=>datain1(6),
        D15=>datain1(5), D14=>datain1(4), D13=>datain1(3),
        D12=>datain1(2), D11=>datain1(1), D10=>datain1(0),
        D017=>datain0(17), D016=>datain0(16), D015=>datain0(15),
        D014=>datain0(14), D013=>datain0(13), D012=>datain0(12),
        D011=>datain0(11), D010=>datain0(10), D09=>datain0(9),
        D08=>datain0(8), D07=>datain0(7), D06=>datain0(6),
        D05=>datain0(5), D04=>datain0(4), D03=>datain0(3),
        D02=>datain0(2), D01=>datain0(1), D00=>datain0(0),
        BW11=>scuba_vhi, BW10=>scuba_vhi, BW01=>scuba_vhi,
        BW00=>scuba_vhi, AR18=>raddr1(8), AR17=>raddr1(7),
        AR16=>raddr1(6), AR15=>raddr1(5), AR14=>raddr1(4),
        AR13=>raddr1(3), AR12=>raddr1(2), AR11=>raddr1(1),
        AR10=>raddr1(0), AR08=>raddr0(8), AR07=>raddr0(7),
        AR06=>raddr0(6), AR05=>raddr0(5), AR04=>raddr0(4),
        AR03=>raddr0(3), AR02=>raddr0(2), AR01=>raddr0(1),
        AR00=>raddr0(0), CKW1=>clk1, CKW0=>clk0, CKR1=>rdclk1,
        CKR0=>rdclk0, CSW1=>wren1, CSW0=>wren0, CSR1=>rden1, CSR0=>rden0,
        Q117=>dataout1(17), Q116=>dataout1(16), Q115=>dataout1(15),
        Q114=>dataout1(14), Q113=>dataout1(13), Q112=>dataout1(12),
```

```
        Q111=>dataout1(11), Q110=>dataout1(10), Q19=>dataout1(9),
        Q18=>dataout1(8), Q17=>dataout1(7), Q16=>dataout1(6),
        Q15=>dataout1(5), Q14=>dataout1(4), Q13=>dataout1(3),
        Q12=>dataout1(2), Q11=>dataout1(1), Q10=>dataout1(0),
        Q017=>dataout0(17), Q016=>dataout0(16), Q015=>dataout0(15),
        Q014=>dataout0(14), Q013=>dataout0(13), Q012=>dataout0(12),
        Q011=>dataout0(11), Q010=>dataout0(10), Q09=>dataout0(9),
        Q08=>dataout0(8), Q07=>dataout0(7), Q06=>dataout0(6),
        Q05=>dataout0(5), Q04=>dataout0(4), Q03=>dataout0(3),
        Q02=>dataout0(2), Q01=>dataout0(1), Q00=>dataout0(0), BUSY=>open);

end Structure;
```

Appendix D.

Using Synplicity Synthesis Attributes with Verilog Source Code

Verilog with Asynchronous Read for Ports 0 and 1:

```
Asynchronous Read for Ports 0 and 1.
/* Verilog netlist generated by SCUBA ORCA Foundry 2001 Production (114) */
/* C:\Program Files\ORCA\bin\nt\scuba -w -e -lang verilog -n qp_na_512x18 -synth synplify -bus_exp 1
-bb -type bram -wp 11 -rp 1111 -num_rows 512 -data_width 18 -read_reg0 noreg -read_reg1 noreg  */
/* Wed Feb 13 15:45:02 2002 */

`timescale 1 ns / 100 ps
module qp_na_512x18 (waddr0, datain0, clk0, wren0, raddr0, rdclk0, rden0,
    waddr1, datain1, clk1, wren1, raddr1, rdclk1, rden1, dataout0,
    dataout1);
    input [8:0] waddr0;
    input [17:0] datain0;
    input clk0;
    input wren0;
    input [8:0] raddr0;
    input rdclk0;
    input rden0;
    input [8:0] waddr1;
    input [17:0] datain1;
    input clk1;
    input wren1;
    input [8:0] raddr1;
    input rdclk1;
    input rden1;
    output [17:0] dataout0;
    output [17:0] dataout1;

VHI scuba_vhi_inst (.Z(scuba_vhi));

BR512X18 qp_na_512x180_0_0 (.AW18(waddr1[8]), .AW17(waddr1[7]),
.AW16(waddr1[6]), .AW15(waddr1[5]), .AW14(waddr1[4]), .AW13(waddr1[3]),
.AW12(waddr1[2]), .AW11(waddr1[1]), .AW10(waddr1[0]), .AW08(waddr0[8]),
.AW07(waddr0[7]), .AW06(waddr0[6]), .AW05(waddr0[5]), .AW04(waddr0[4]),
.AW03(waddr0[3]), .AW02(waddr0[2]), .AW01(waddr0[1]), .AW00(waddr0[0]),
.D117(datain1[17]), .D116(datain1[16]), .D115(datain1[15]),
.D114(datain1[14]), .D113(datain1[13]), .D112(datain1[12]),
.D111(datain1[11]), .D110(datain1[10]), .D19(datain1[9]),
.D18(datain1[8]), .D17(datain1[7]), .D16(datain1[6]), .D15(datain1[5]),
.D14(datain1[4]), .D13(datain1[3]), .D12(datain1[2]), .D11(datain1[1]),
.D10(datain1[0]), .D017(datain0[17]), .D016(datain0[16]),
.D015(datain0[15]), .D014(datain0[14]), .D013(datain0[13]),
.D012(datain0[12]), .D011(datain0[11]), .D010(datain0[10]),
.D09(datain0[9]), .D08(datain0[8]), .D07(datain0[7]), .D06(datain0[6]),
.D05(datain0[5]), .D04(datain0[4]), .D03(datain0[3]), .D02(datain0[2]),
.D01(datain0[1]), .D00(datain0[0]), .BW11(scuba_vhi), .BW10(scuba_vhi),
.BW01(scuba_vhi), .BW00(scuba_vhi), .AR18(raddr1[8]), .AR17(raddr1[7]),
.AR16(raddr1[6]), .AR15(raddr1[5]), .AR14(raddr1[4]), .AR13(raddr1[3]),
.AR12(raddr1[2]), .AR11(raddr1[1]), .AR10(raddr1[0]), .AR08(raddr0[8]),
.AR07(raddr0[7]), .AR06(raddr0[6]), .AR05(raddr0[5]), .AR04(raddr0[4]),
.AR03(raddr0[3]), .AR02(raddr0[2]), .AR01(raddr0[1]), .AR00(raddr0[0]),
.CKW1(clk1), .CKW0(clk0), .CKR1(rdclk1), .CKR0(rdclk0), .CSW1(wren1),
.CSW0(wren0), .CSR1(rden1), .CSR0(rden0), .Q117(dataout1[17]),
.Q116(dataout1[16]), .Q115(dataout1[15]), .Q114(dataout1[14]),
```

```
.Q113(dataout1[13]), .Q112(dataout1[12]), .Q111(dataout1[11]),
.Q110(dataout1[10]), .Q19(dataout1[9]), .Q18(dataout1[8]),
.Q17(dataout1[7]), .Q16(dataout1[6]), .Q15(dataout1[5]),
.Q14(dataout1[4]), .Q13(dataout1[3]), .Q12(dataout1[2]),
.Q11(dataout1[1]), .Q10(dataout1[0]), .Q017(dataout0[17]),
.Q016(dataout0[16]), .Q015(dataout0[15 .Q014(dataout0[14]),
.Q013(dataout0[13]), .Q012(dataout0[12]), .Q011(dataout0[11
.Q010(dataout0[10]), .Q09(dataout0[9]), .Q08(dataout0[8]),
.Q07(dataout0[7]), .Q06(dataout0[6]), .Q05(dataout0[5]),
.Q04(dataout0[4]), .Q03(dataout0[3]), .Q02(dataout0[2]),
.Q01(dataout0[1]), .Q00(dataout0[0]), .BUSY())
/* synthesis ARBITERMODE="FALSE" */;
endmodule

module BR512X18 (AW18, AW17, AW16, AW15, AW14, AW13, AW12, AW11, AW10,
                 AW08, AW07, AW06, AW05, AW04, AW03, AW02, AW01, AW00,
                 D117, D116, D115, D114, D113, D112, D111, D110, D19,
                 D18, D17, D16, D15, D14, D13, D12, D11, D10,
                 D017, D016, D015, D014, D013, D012, D011, D010, D09,
                 D08, D07, D06, D05, D04, D03, D02, D01, D00,
                 BW11, BW10, BW01, BW00,
                 AR18, AR17, AR16, AR15, AR14, AR13, AR12, AR11, AR10,
                 AR08, AR07, AR06, AR05, AR04, AR03, AR02, AR01, AR00,
                 CKW1, CKW0, CKR1, CKR0, CSW1, CSW0, CSR1, CSR0,
                 Q117, Q116, Q115, Q114, Q113, Q112, Q111, Q110, Q19,
                 Q18, Q17, Q16, Q15, Q14, Q13, Q12, Q11, Q10,
                 Q017, Q016, Q015, Q014, Q013, Q012, Q011, Q010, Q09,
                 Q08, Q07, Q06, Q05, Q04, Q03, Q02, Q01, Q00, BUSY) /* synthesis syn_black_box */;

input AW08, AW07, AW06, AW05, AW04, AW03, AW02, AW01, AW00;
input AR08, AR07, AR06, AR05, AR04, AR03, AR02, AR01, AR00;
input AW18, AW17, AW16, AW15, AW14, AW13, AW12, AW11, AW10;
input AR18, AR17, AR16, AR15, AR14, AR13, AR12, AR11, AR10;
input D017, D016, D015, D014, D013, D012, D011, D010, D09;
input D08, D07, D06, D05, D04, D03, D02, D01, D00;
input D117, D116, D115, D114, D113, D112, D111, D110, D19;
input D18, D17, D16, D15, D14, D13, D12, D11, D10;
input BW00, BW01, CKW0, CKR0, CSW0, CSR0,BW10, BW11, CKW1, CKR1, CSW1, CSR1;

output Q017, Q016, Q015, Q014, Q013, Q012, Q011, Q010, Q09, Q08, Q07, Q06;
output Q05, Q04, Q03, Q02, Q01, Q00, Q117, Q116, Q115, Q114, Q113, Q112, Q111;
output Q110, Q19, Q18, Q17, Q16, Q15, Q14, Q13, Q12, Q11, Q10, BUSY;

endmodule
```

Note: At the bottom of the ispLEVER Module/IP Manager generated code, the VHI and BR512X18 module definitions need to be included along with the /* synthesis syn_black_box */ comment tag. This procedure can be performed by copying the needed module definitions to the code or by including the orca4_synplify.v file to the synthesis hierarchy.

Appendix E.

Instantiation Templates for Verilog and VHDL Source Code

Verilog Instantiation template

```
/* Verilog module instantiation template generated by SCUBA ORCA Foundry 2001 Production (114) */
/* Wed Mar 06 10:05:05 2002 */

/* parameterized module instance */
module quad_port (waddr0,
                  datain0,
                  clk0,
                  wren0,
                  raddr0,
                  rdclk0,
                  rden0,
                  waddr1,
                  datain1,
                  clk1,
                  wren1,
                  raddr1,
                  rdclk1,
                  rden1,
                  dataout0,
                  dataout1
                  );

    input [8:0] waddr0;
    input [17:0] datain0;
    input clk0;
    input wren0;
    input [8:0] raddr0;
    input rdclk0;
    input rden0;
    input [8:0] waddr1;
    input [17:0] datain1;
    input clk1;
    input wren1;
    input [8:0] raddr1;
    input rdclk1;
    input rden1;
    output [17:0] dataout0;
    output [17:0] dataout1;


qp_na_512x18 qp_ram (.waddr0(waddr0),
                     .datain0(datain0),
                     .clk0(clk0),
                     .wren0(wren0),
                     .raddr0(raddr0),
                     .rdclk0(rdclk0),
                     .rden0(rden0),
                     .waddr1(waddr1),
                     .datain1(datain1),
                     .clk1(clk1),
                     .wren1(wren1),
                     .raddr1(raddr1),
                     .rdclk1(rdclk1),
```

```
                    .rden1(rden1),
                    .dataout0(dataout0),
                    .dataout1(dataout1)
                    );

endmodule

VHDL Instantiation Template
Read Address Registers Enabled for Port 1
Read Address and Data out Registers Enabled for Port 0
-- VHDL module instantiation generated by SCUBA ORCA Foundry 2001 Production (3)
-- Wed Mar 06 10:56:44 2002

-- parameterized module component declaration
library ieee;
use ieee.std_logic_1164.all;

entity quad_port is
    port (waddr0: in  std_logic_vector(8 downto 0);
          datain0: in  std_logic_vector(17 downto 0);
          clk0: in  std_logic;
          wren0: in  std_logic;
          raddr0: in  std_logic_vector(8 downto 0);
          rdclk0: in  std_logic;
          rden0: in  std_logic;
          waddr1: in  std_logic_vector(8 downto 0);
          datain1: in  std_logic_vector(17 downto 0);
          clk1: in  std_logic;
          wren1: in  std_logic;
          raddr1: in  std_logic_vector(8 downto 0);
          rdclk1: in  std_logic;
          rden1: in  std_logic;
          dataout0: out  std_logic_vector(17 downto 0);
          dataout1: out  std_logic_vector(17 downto 0));

end quad_port;

architecture behavior of quad_port is

component qp_na_512x18
    port (waddr0: in  std_logic_vector(8 downto 0);
          datain0: in  std_logic_vector(17 downto 0);
          clk0: in  std_logic;
          wren0: in  std_logic;
          raddr0: in  std_logic_vector(8 downto 0);
          rdclk0: in  std_logic;
          rden0: in  std_logic;
          waddr1: in  std_logic_vector(8 downto 0);
          datain1: in  std_logic_vector(17 downto 0);
          clk1: in  std_logic;
          wren1: in  std_logic;
          raddr1: in  std_logic_vector(8 downto 0);
          rdclk1: in  std_logic;
          rden1: in  std_logic;
          dataout0: out  std_logic_vector(17 downto 0);
          dataout1: out  std_logic_vector(17 downto 0));
end component;

begin
```

```
-- parameterized module component instance
quad_port_ram : qp_na_512x18
    port map (waddr0   => waddr0,
              datain0  => datain0,
              clk0     => clk0,
              wren0    => wren0,
              raddr0   => raddr0,
              rdclk0   => rdclk0,
              rden0    => rden0,
              waddr1   => waddr1,
              datain1  => datain1,
              clk1     => clk1,
              wren1    => wren1,
              raddr1   => raddr1,
              rdclk1   => rdclk1,
              rden1    => rden1,
              dataout0 => dataout0,
              dataout1 => dataout1
             );

end behavior;
```