RAM Inferencing in Synplify® Software Using Xilinx RAMs

Synplicity®

Simply Better Results

Overview

Many FPGA families provide a mechanism to implement technology-specific RAMs in HDL source code. To take advantage of these optimal RAM implementations, you must manually instantiate the technology-specific RAM cells.

Disadvantages of Instantiation

The following list outlines the disadvantages of instantiating the technology-specific RAM cells.

- The HDL code is no longer technology independent.
- If you use a black box methodology, your synthesis tool might not have access to any timing or area data.

Synplify software version 7.1 addresses these issues by automatically inferring synchronous RAMs directly from your HDL source code. The RTL View of HDL Analyst® then displays the RAM as a simple component, which makes reading the schematic easier. Additionally, the RAM logic is automatically mapped to applicable technology-specific RAM cells. The Synplify software supports synchronous RAMs for Altera, Atmel, Lattice Orca, and Xilinx technology families. This application note specifically covers the RAM inferencing of Xilinx technology families in the Synplify software.

Advantages of Inferencing

RAM inferencing also has the advantages listed below:

- Technology-independent coding style.
- Synplify software provides automatic timing-driven synthesis for RAMs.
- No additional tool dependencies.

The goal for RAM inferencing in the Synplify software is to give you a method that lets you easily specify RAM structures in your HDL source code, while maintaining portability and ensuring that the netlist output after synthesis remains logically correct. Portability across vendors requires that each vendor technology that is mapped has a certain amount of *glue* logic which normally surrounds the technology-specific RAM primitive so that the logic matches the functionality of the specific RAM module in the Synplify HDL-source RAM primitive. Xilinx-specific details regarding *glue* logic are explained in the “Virtex Conflict Resolution” section. The addition of the glue logic can result in a non-optimal RAM implementation. However, if you want a design that most efficiently uses a specific RAM primitive technology, you must instantiate the vendor-specific RAM primitive.

Synplify Tool RAM Inferencing Support

To infer a RAM, the Synplify synthesis tool looks for an assignment to a signal (register in Verilog) that is an array of an array, or a case structure controlled by a clock edge and a write enable. If the address used to index the write-to and read-from RAM is the same, then a single-port RAM is inferred as shown in the example below. If the addresses are different, then a dual-port RAM is inferred.

In addition to this support for inferring RAMs, from the Synplify 7.0 software release forward, new support lets you infer Xilinx block SelectRAMs with new coding styles when the RAM output is registered. The new coding style supports the enable and reset (ssrt in the case of Virtex-II) pins of the block SelectRAM primitive. Different write mode operations are supported for single-port RAM targeted for the Virtex-II technology. For more details on these coding styles refer to *Coding Style Mapped to Single-Port Block SelectRAMs* on page 18.

VHDL Single-Port RAM Example

The following code illustrates an example of a single-port RAM.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

entity ramtest is

port (q : out std_logic_vector(3 downto 0);
       d : in  std_logic_vector(3 downto 0);
       addr : in  std_logic_vector(2 downto 0);
       we         : in  std_logic;
       clk : in  std_logic);
end ramtest;

architecture rtl of ramtest is
type mem_type is array (7 downto 0) of std_logic_vector (3 downto 0);
signal mem : mem_type;
begin

q <= mem(conv_integer(addr));

process (clk, we, addr) begin
if rising_edge(clk) then
if (we = '1') then
      mem(conv_integer(addr)) <= d;
end if;
end if;
end process;
end rtl;
```

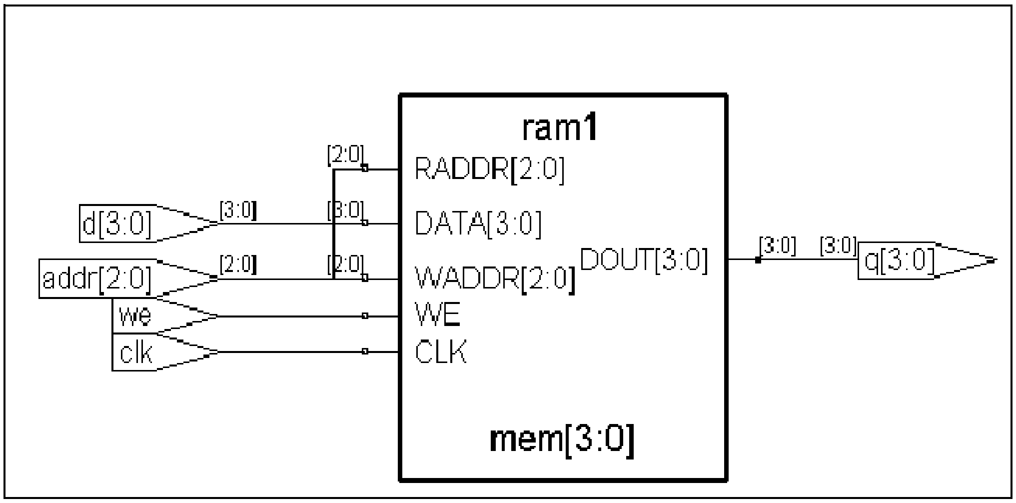

Figure 1: HDL Analyst RTL view of the preceding inferred single-port RAM

Verilog Memory Array

The following code implements a Verilog memory array.

```
module ramtest(z, raddr, d, waddr, we, clk);
output [3:0] z;
input [3:0] d;
input [3:0] raddr, waddr;
input we;
input clk;

reg [3:0] mem [7:0];

assign z = mem[raddr];

always @(posedge clk) begin
if(we) mem[waddr]= d;
end

endmodule
```

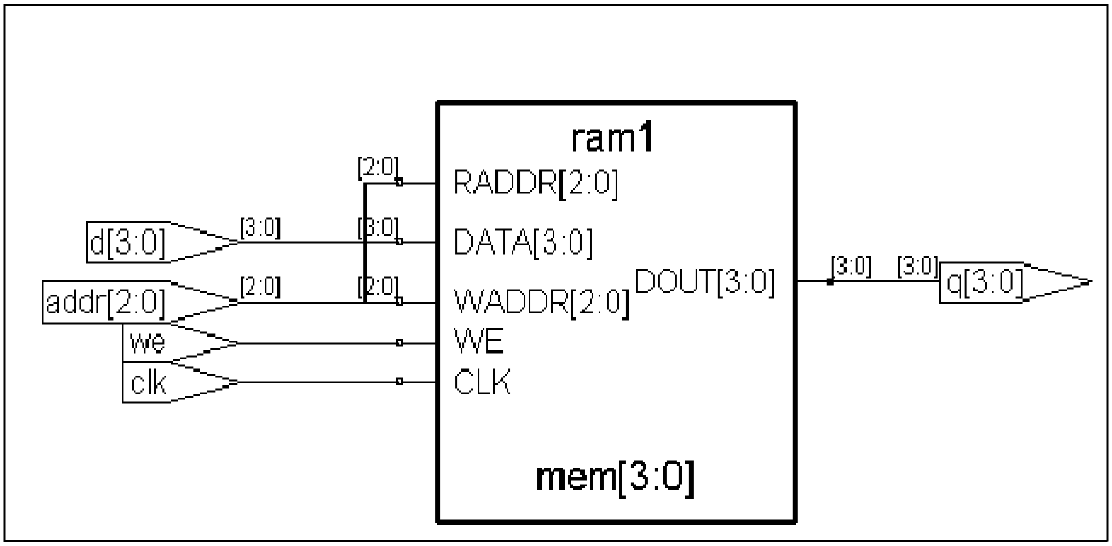

Figure 2: HDL Analyst RTL view of inferred dual -port RAM.

Verilog Code Example of a Dual-Port RAM

The following code illustrates an example of a dual-port RAM.

```
module ram16x8(z, raddr, d, waddr, we, clk);
output [7:0] z;
input [7:0] d;
input [3:0] raddr, waddr;
input we;
input clk;
reg [7:0] z;
reg [7:0] mem0, mem1, mem2, mem3, mem4, mem5, mem6, mem7;
reg [7:0] mem8, mem9, mem10, mem11, mem12, mem13, mem14, mem15;
always @(mem0 or mem1 or mem2 or mem3 or mem4 or mem5 or mem6 or mem7 or
mem8 or mem9 or mem10 or mem11 or mem12 or mem13 or mem14 or mem15 or
raddr)
begin
case (raddr[3:0])
4'b0000: z = mem0;
4'b0001: z = mem1;
4'b0010: z = mem2;
4'b0011: z = mem3;
4'b0100: z = mem4;
4'b0101: z = mem5;
4'b0110: z = mem6;
4'b0111: z = mem7;
4'b1000: z = mem8;
4'b1001: z = mem9;
4'b1010: z = mem10;
4'b1011: z = mem11;
4'b1100: z = mem12;
4'b1101: z = mem13;
4'b1110: z = mem14;
4'b1111: z = mem15;
```

```
endcase
end

always @(posedge clk) begin
if(we) begin
case (waddr[3:0])
4'b0000: mem0 = d;
4'b0001: mem1 = d;
4'b0010: mem2 = d;
4'b0011: mem3 = d;
4'b0100: mem4 = d;
4'b0101: mem5 = d;
4'b0110: mem6 = d;
4'b0111: mem7 = d;
4'b1000: mem8 = d;
4'b1001: mem9 = d;
4'b1010: mem10 = d;
4'b1011: mem11 = d;
4'b1100: mem12 = d;
4'b1101: mem13 = d;
4'b1110: mem14 = d;
4'b1111: mem15 = d;
endcase
end
end
endmodule
```

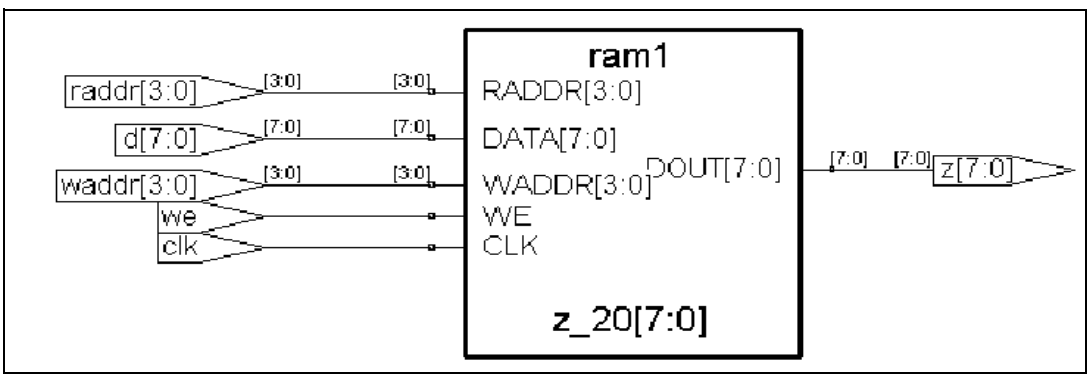

Figure 3: HDL Analyst RTL view of the preceding inferred dual-port RAM.

Inferring Block SelectRAMs in Xilinx

This sections discusses synchronous Xilinx block SelectRAMs and their requirements.

Fully Synchronous RAMs and Registered Address Requirement

Xilinx block SelectRAMs are fully synchronous. To map to a block SelectRAM, one of the following registered conditions must exist:

- Either the read address or the output must be registered
- Both the read address and the output must be registered

Using the syn_ramstyle Attribute for Block SelectRAMs

The `syn_ramstyle="block_ram"` attribute is only required for Xilinx Virtex block SelectRAM and must be set in one of two places to infer block SelectRAMs. You can set the syn_ramstyle attribute on a memory object in the HDL source code, with TCL script, or the SCOPE® interface as follows:

- In your HDL source code on the register signal used to hold the values of the output of the RAM.
- In Tcl (script)/SCOPE interface(GUI) on the output signal of the RAM.

Attribute Usage

The following examples illustrate how to specify the syn_ramstyle attribute in various HDL languages. Tcl script, and the SCOPE interface.

Verilog Example of Specifying the syn_ramstyle Attribute

```
reg [7:0] ram_dout [127:0] /*synthesis syn_ramstyle = "block_ram"*/;
```

VHDL Example of Specifying the syn_ramstyle Attribute

attribute syn_ramstyle of ram_dout : signal is "block_ram";

Tcl Example of Specifying the syn_ramstyle Attribute

define_attribute { ram_dout [127:0]} syn_ramstyle {block_ram}

SCOPE Interface Example of Specifying the syn_ramstyle Attribute

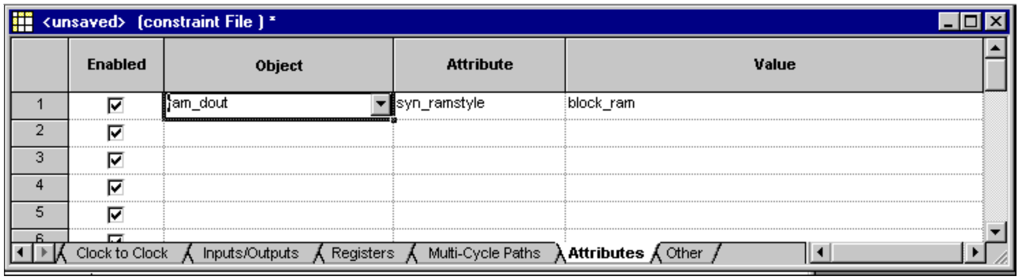

Figure 4: Setting the syn_ramstyle attribute in the SCOPE interface to infer block SelectRAMs

Global Limitations

The following global limitations exist when inferencing RAMs:

- RAM inferencing is only supported for synchronous RAMs.
- Initialization of RAMs is not supported
- *Address wrapping* is not supported. This means that the RAM implemented is assumed to start at address 0 and uses one of the following addressing scenarios.

  Scenario 1

  The required RAM primitive is 16 words deep and has an address range of 0 to 23 (or 24 words deep).The inferred RAM is implemented in 2 RAM cells, leaving address 24 to 31 unused.

  Scenario 2

  The required RAM primitive is 16 words deep and has an address range of 8 to 23 (or 16 words deep). The inferred RAM is implemented in 2 RAM cells, leaving address 0 to 7, and 24 to 31 unused.

Implementation Conventions for Specifying Xilinx Block SelectRAMs

The following conventions are used when specifying block SelectRAMs.

Size Requirement: (RAM width > 1 bit) and (RAM depth > 1 bit) and (RAM width * RAM depth >= 8 bits)

RAM Primitive: Use RAM16X1S for single-port RAMs, RAM16X1D for dual-port RAMs.

Block RAM Primitive: Use one of the following, based on the technology and word width:

- RAMB4_S# for single-port block SelectRAMs and RAMB4_S#_S# for dual-port block SelectRAMs in Virtex/VirtexE where # is the word width of the RAM.
- RAMB16_S# for single-port block SelectRAMs, RAMB16S#_S# for dual-port block SelectRAMs where # is the word width of the RAM.

Inferring Block SelectRAMs in Xilinx Technologies

RAM inferencing in the Synplify tool is limited to the coding styles discussed throughout this application note.

Prior to the Synplify 7.0 release, a block SelectRAM could be inferred only if the read address was registered as shown by the following code example.

Verilog Code Example Inferring Single-Port Block SelectRAM

```
module ram_test(q, a, d, we, clk);

output [7:0] q;
input  [7:0] d;
input  [6:0] a;
input  clk, we;

reg [6:0] read_add;

/* The array of an array register ("mem") the RAM will be inferred from.
*/
reg [7:0] mem [127:0] /* synthesis syn_ramstyle = "block_ram */;

assign q = mem[read_add];

always @(posedge clk) begin
if(we)
/* Register RAM Data */
mem[a] <= d;
/* Register Read Address. Basic RAM support does not
require this address register.*/
read_add <= a;
end

endmodule
```

Dual-Port Block SelectRAM with Registered Read Address

When two addresses are used to do the read and the write operation respectively, and the read address is registered, a dual-port block SelectRAM can be inferred as shown by the following example and illustrated in *HDL Analyst Technology view of an inferred Virtex block SelectRAM* on page 9.

Dual-Port Block SelectRAM with Read Address Registered

```
module dualportram(q, a1, a2, d, we, clk, en);
output [7:0] q;
input [7:0] d;
input[6:0] a1;
input[6:0] a2;
input clk, we, en;

reg [6:0] read_addr;
reg[7:0] mem [127:0] /* synthesis syn_ramstyle="block_ram" */;
```

```
assign q = mem[read_addr];
always @(posedge clk) begin
if (we)
mem[a2] <= d;
read_addr <= a1;
end

endmodule
```

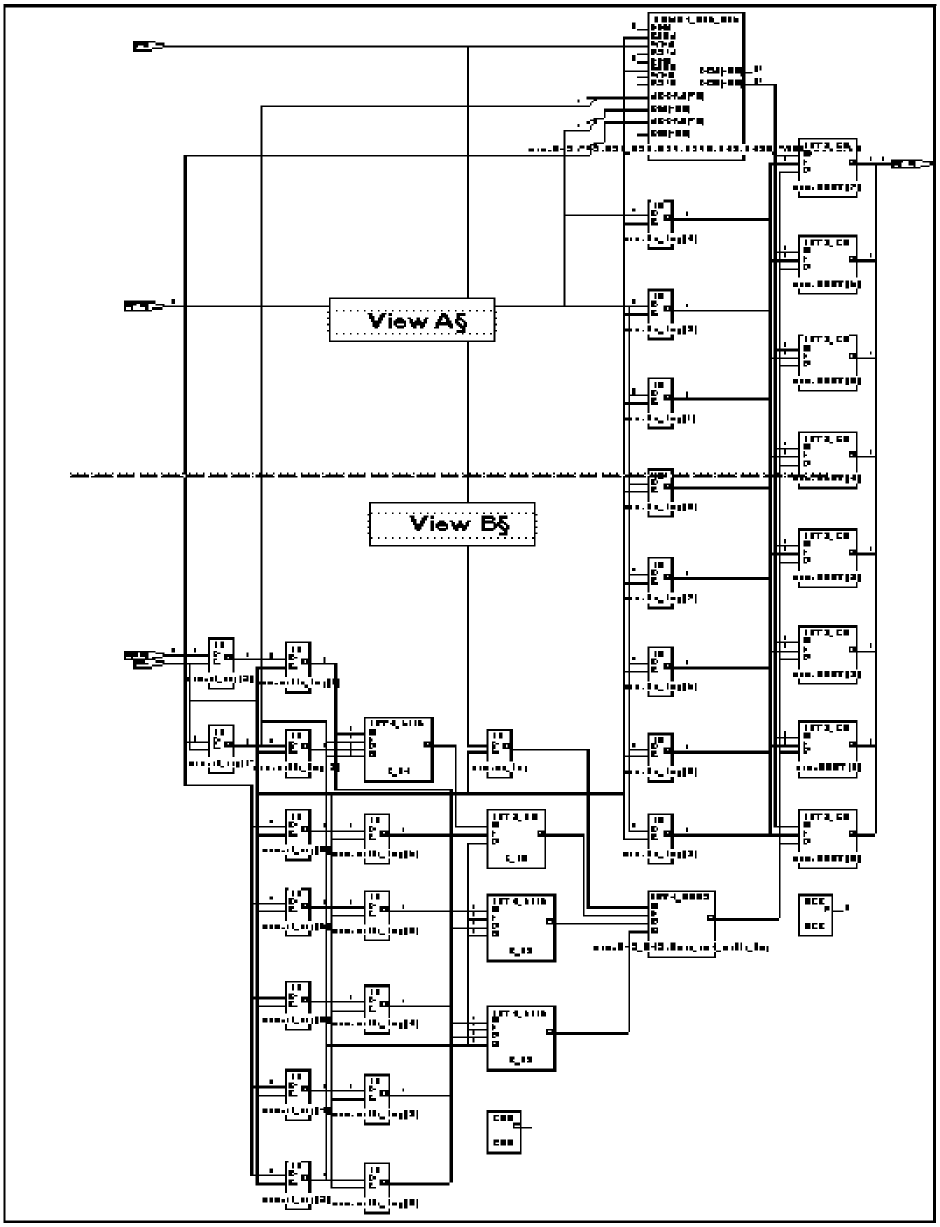

Figure 5: HDL Analyst Technology view of an inferred Virtex block SelectRAM

This figure shows a dual-port RAM inferred by the code in the preceding example, *Dual-Port Block SelectRAM with Registered Read Address* on page 8.

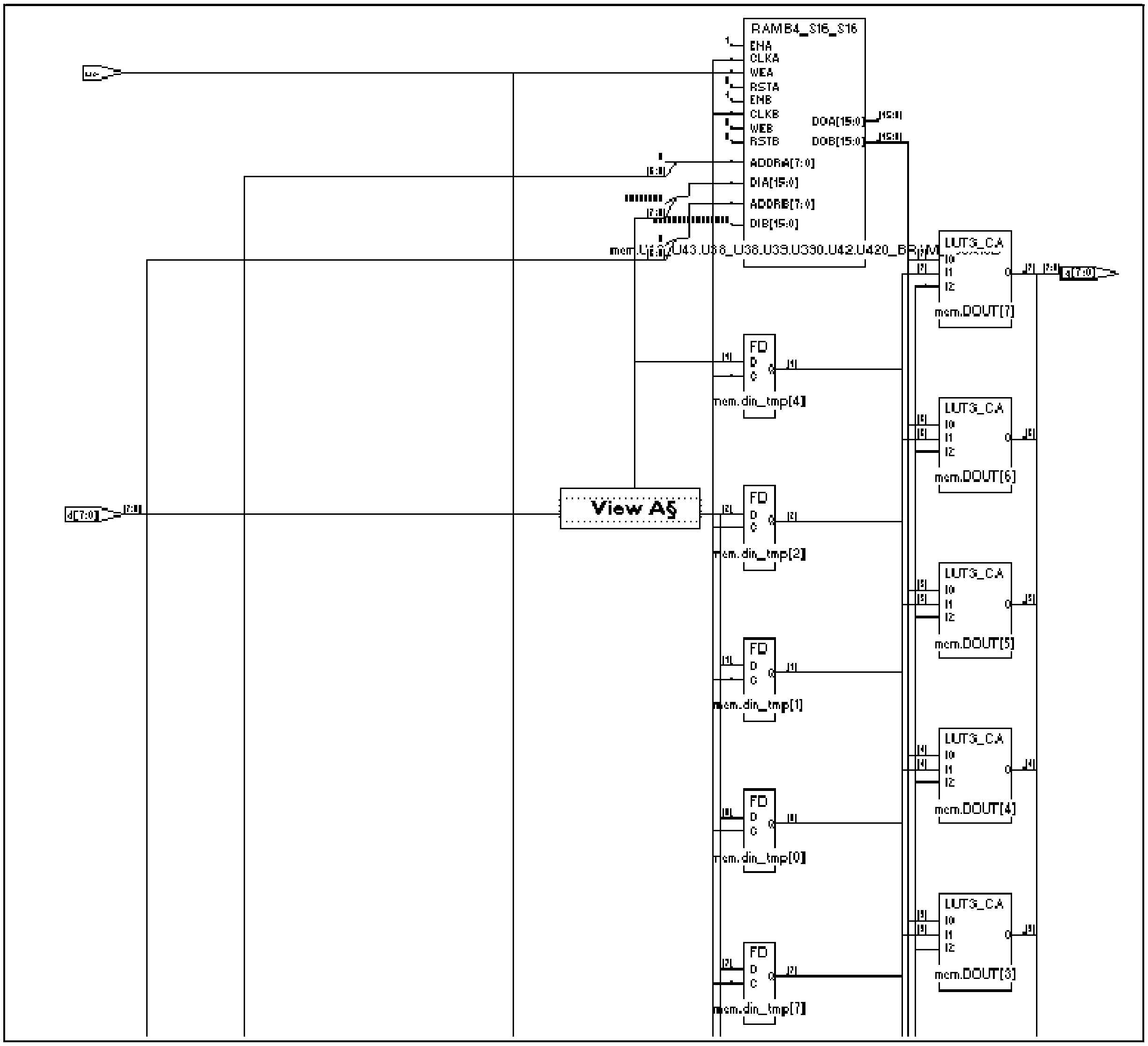

Figure 6: Detailed view A of the inferred dual-port Virtex block SelectRAM shown in its entirety in the *HDL Analyst Technology view of an inferred Virtex block SelectRAM* on page 9.

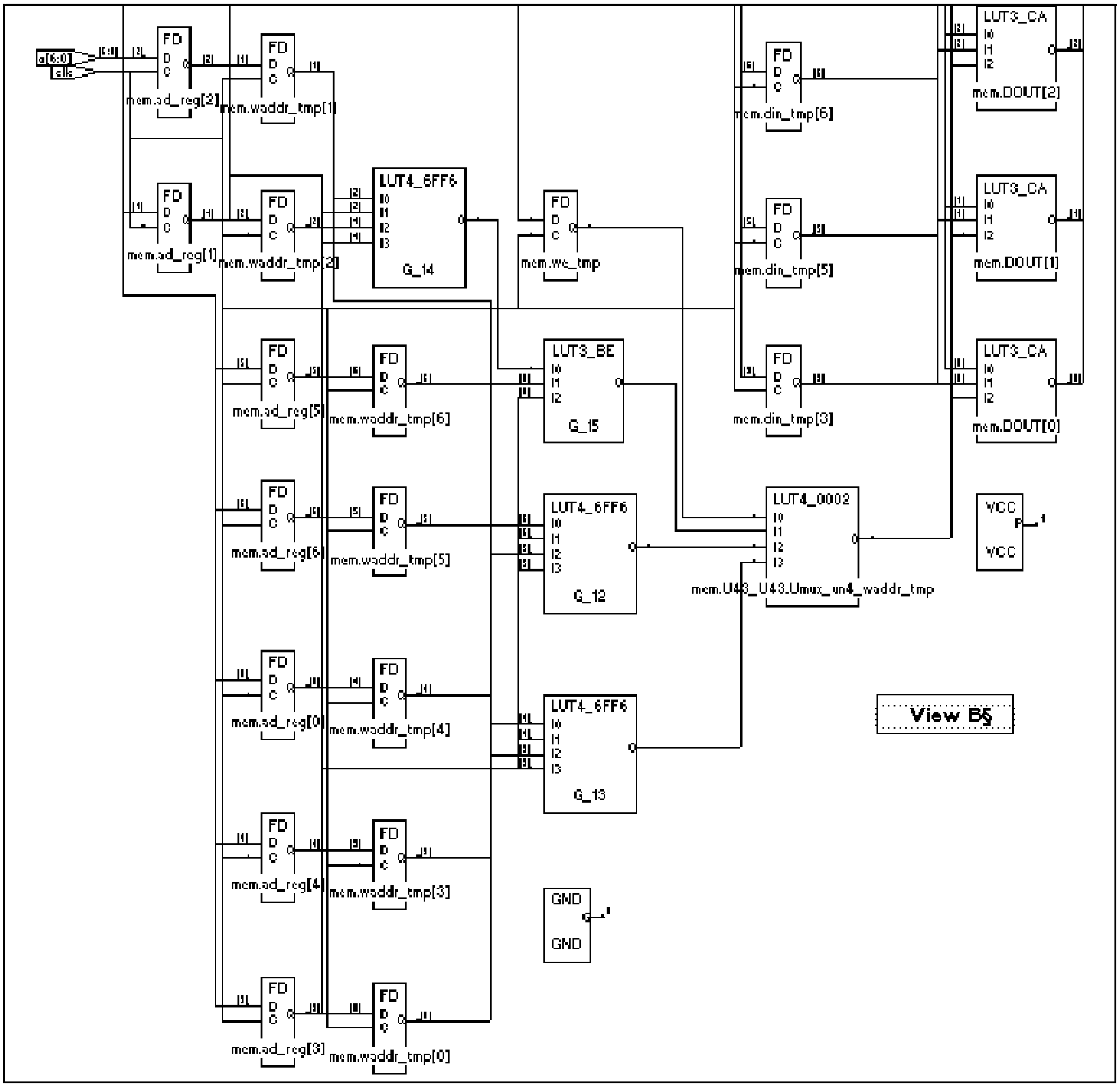

Figure 7: Detailed view of the lower portion of an inferred Virtex block SelectRAM shown in its entirety in the *HDL Analyst Technology view of an inferred Virtex block SelectRAM on page 9*.

Virtex Conflict Resolution

Additionally, the Xilinx application note *XAPP130 October 16, 1998 (Version 1.0)* for Virtex block SelectRAMs specifies conflict resolution behavior (conflicts do not cause any physical damage) as the following two possible flows describe:

1. If both ports write to the same memory cell simultaneously, violating the clock-to-clock setup requirement, consider the data stored as invalid.
2. If one port attempts a read of the same memory cell, the other simultaneously writes, violating the clock-to-clock setup requirement, the following occurs.
   - The write succeeds which subsequently is described as Flow 1.
   - The data out on the writing port accurately reflects the data written which subsequently is described as Flow 2.
   - The data out on the reading port is invalid.

The Synplify software creates *glue* by-pass logic to ensure pre- and post- synthesis simulation results match as shown in the figure below, in which Figure 8: View B shows the block SelectRAM with glue bypass logic on page 12.

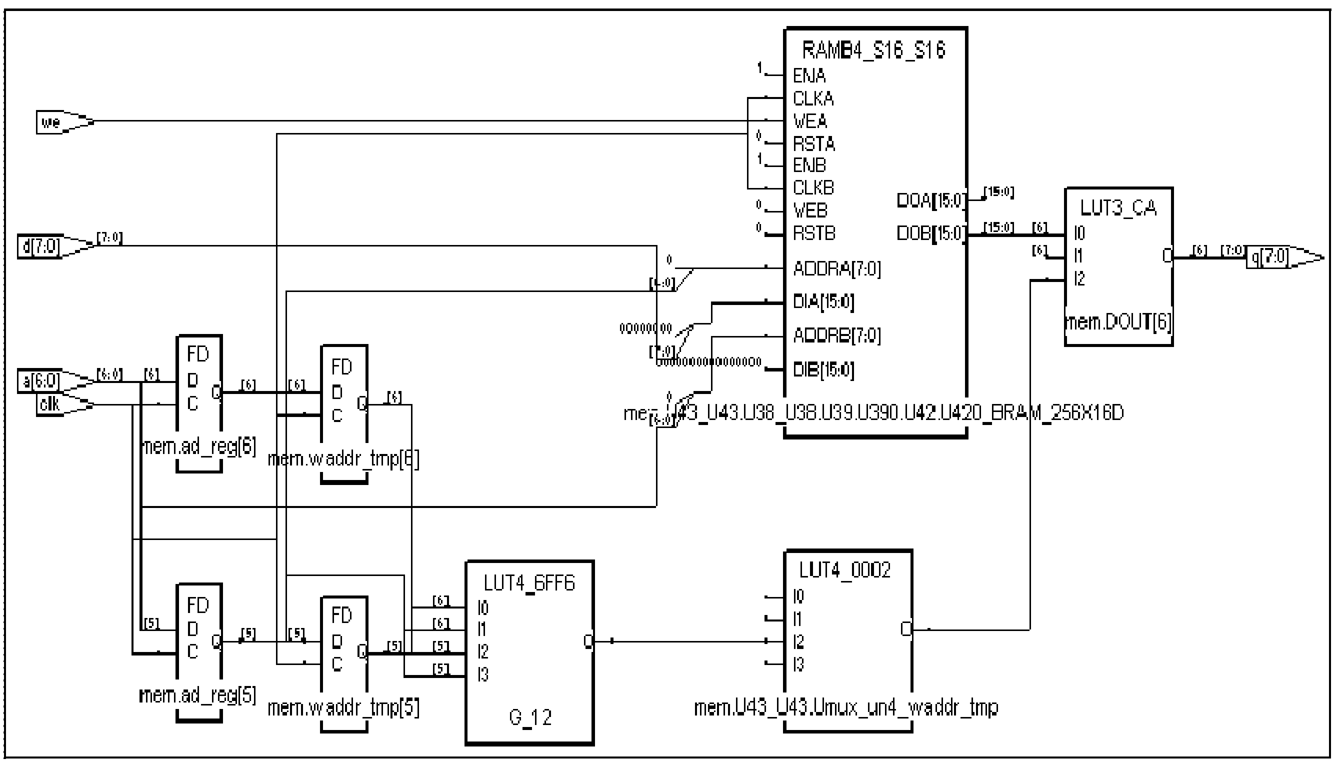

Figure 8: View B shows the block SelectRAM with *glue* bypass logic

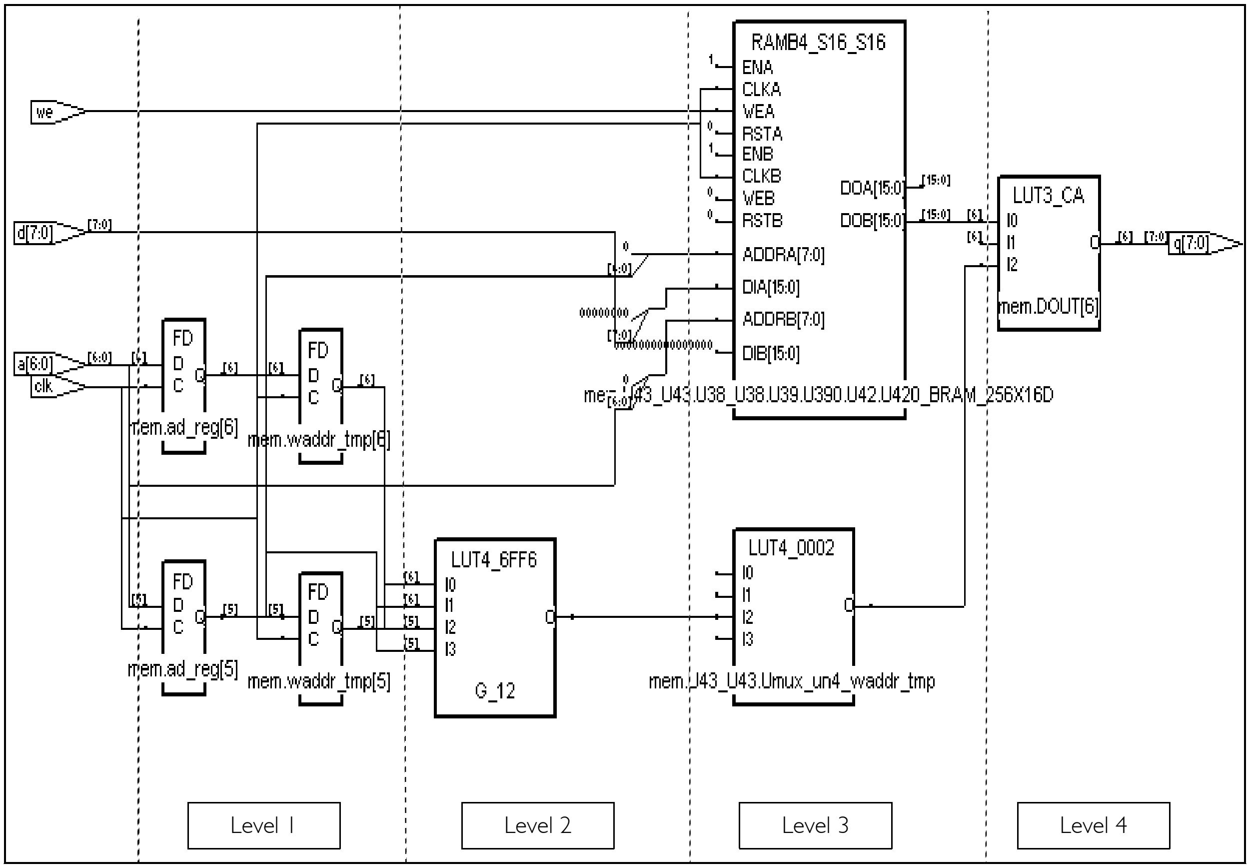

Figure 9: Flow 1 overview showing block SelectRAM with bypass logic

The following series of diagrams zoom in on the logic created by the Synplify tool to let you closely examine the *glue* logic created for the inferred RAM as shown in the overview diagram depicting *View B shows the block SelectRAM with glue bypass logic* on page 12.

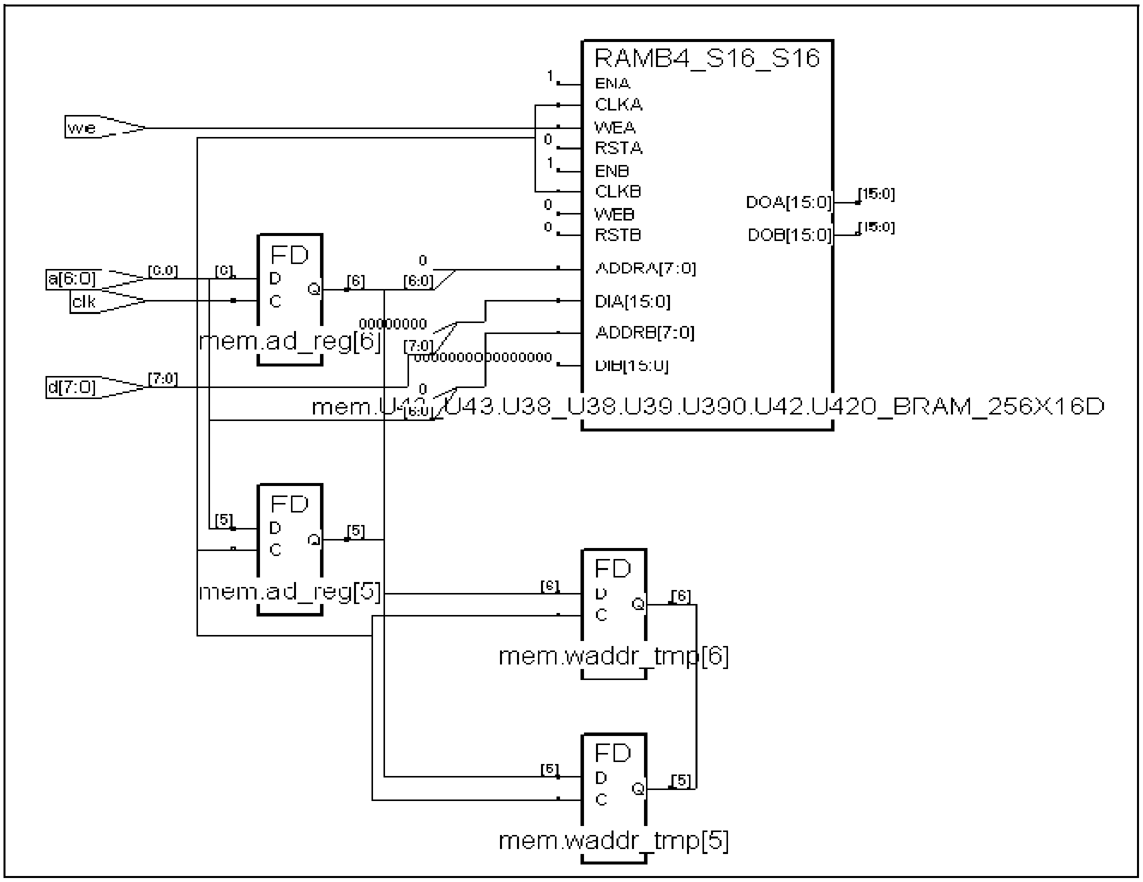

Figure 10: Block SelectRAM with bypass logic

- Level 1 in the above figure shows the two parallel FD registers on the left. This stage stores memory read and memory write addresses as shown in the figure Figure 11: Filtered RTL view of Level 1 on page 15.
- Level 2 in in the above figure shows the LUT4_6FF6. This stage compares the values of the read and write addresses and checks for equality in Figure 12: Gate-level view of Level 2 (after traversing component LUT4_6FF6 G12) on page 15.
- Level 3 in in the above figure shows the LUT4_002. This stage produces the logical AND output which controls the select line for the MUX in Stage 3 as shown in the figure Figure 13: Gate-level view of Level 3, read/write address compare (after pushing down into LUT4_002 G_6). on page 16.
- Level 4 in in the above figure is LUT3_CA. This stage contains the MUX that controls the output of the RAM data shown in the figure Figure 14: Gate-level view of Level 4, read/write address compare output ANDed with the write enable (after pushing down into LUT3_CA). on page 16.

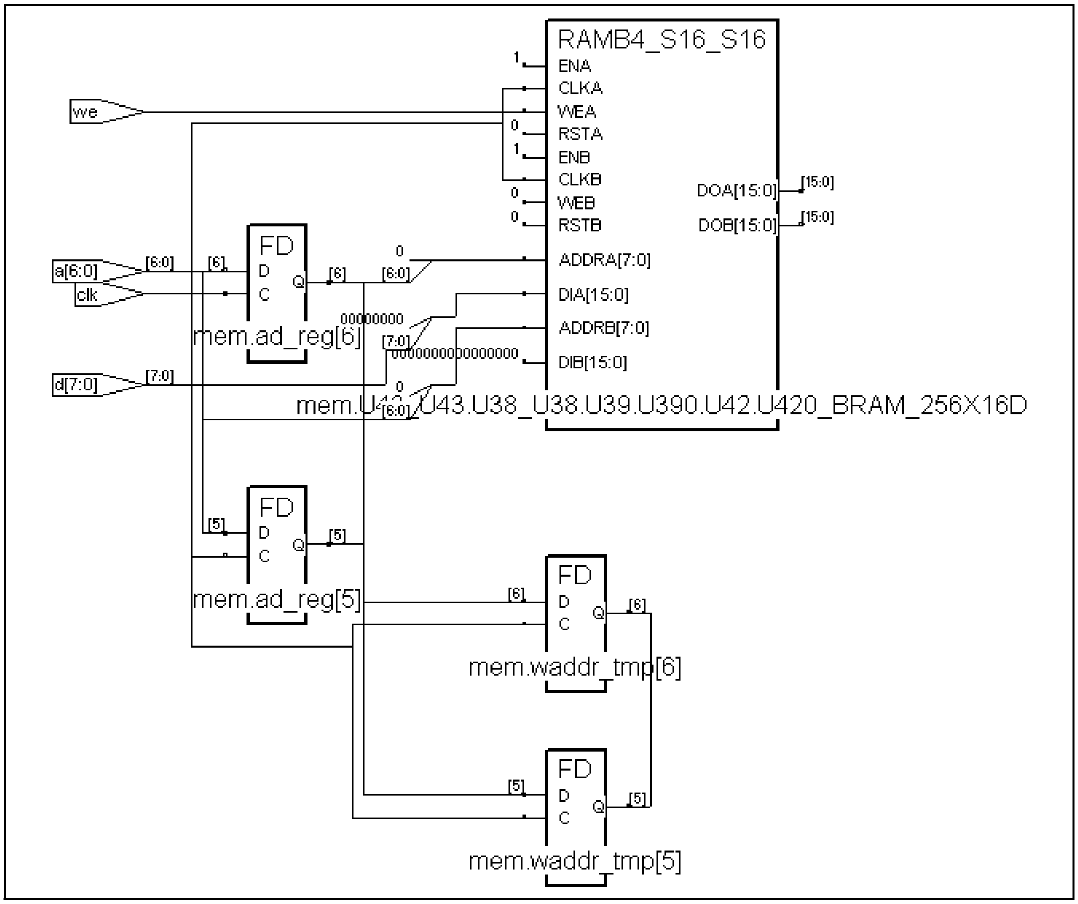

Figure 11: Filtered RTL view of Level 1

Level 1 is the two parallel FD registers on the left. This stage stores memory read and memory write addresses.

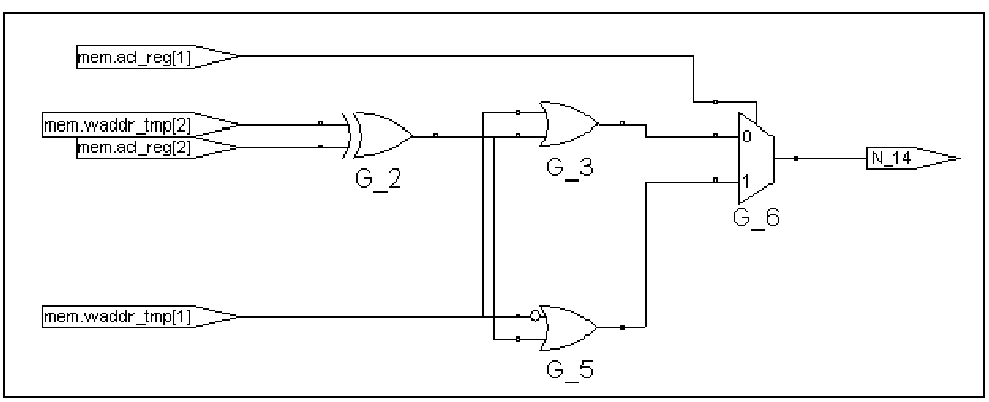

Figure 12: Gate-level view of Level 2 (after traversing component LUT4_6FF6 G12)

Level 2 is the LUT4_6FF6. This stage compares the values of the read and write addresses and checks for equality.

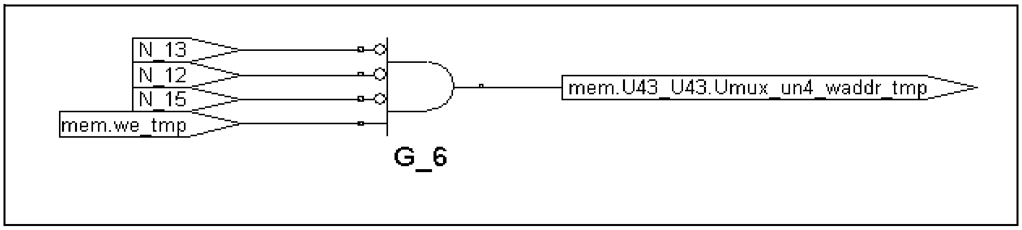

Figure 13: Gate-level view of Level 3, read/write address compare (after pushing down into LUT4_002 G_6).

Level 3 is LUT4_002. This stage produces the logical AND output that controls the select line for the MUX in Stage 4.

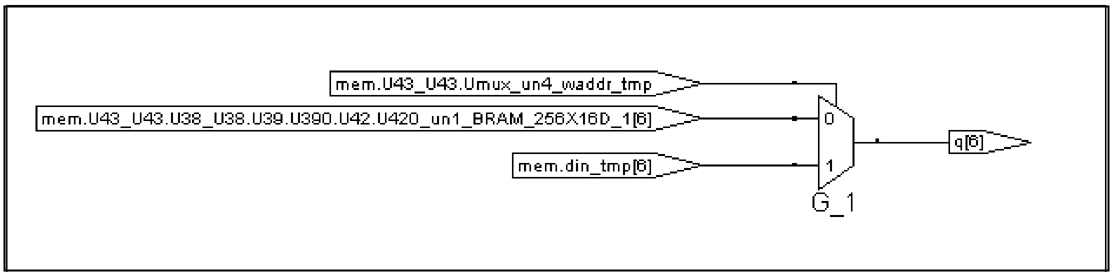

Figure 14: Gate-level view of Level 4, read/write address compare output ANDed with the write enable (after pushing down into LUT3_CA).

Level 4 is LUT3_CA. This stage contains the MUX that controls the output of the RAM data.

The following series of diagrams illustrate Flow 2, defined by the second bullet in the section *Virtex Conflict Resolution* on page 12 in which the data out on the writing port accurately reflects the data written. The read/write address compare and control path for the block SelectRAM in this figure and subsequent figures illustrate the four stages that comprise the bypass logic for the block SelectRAM.

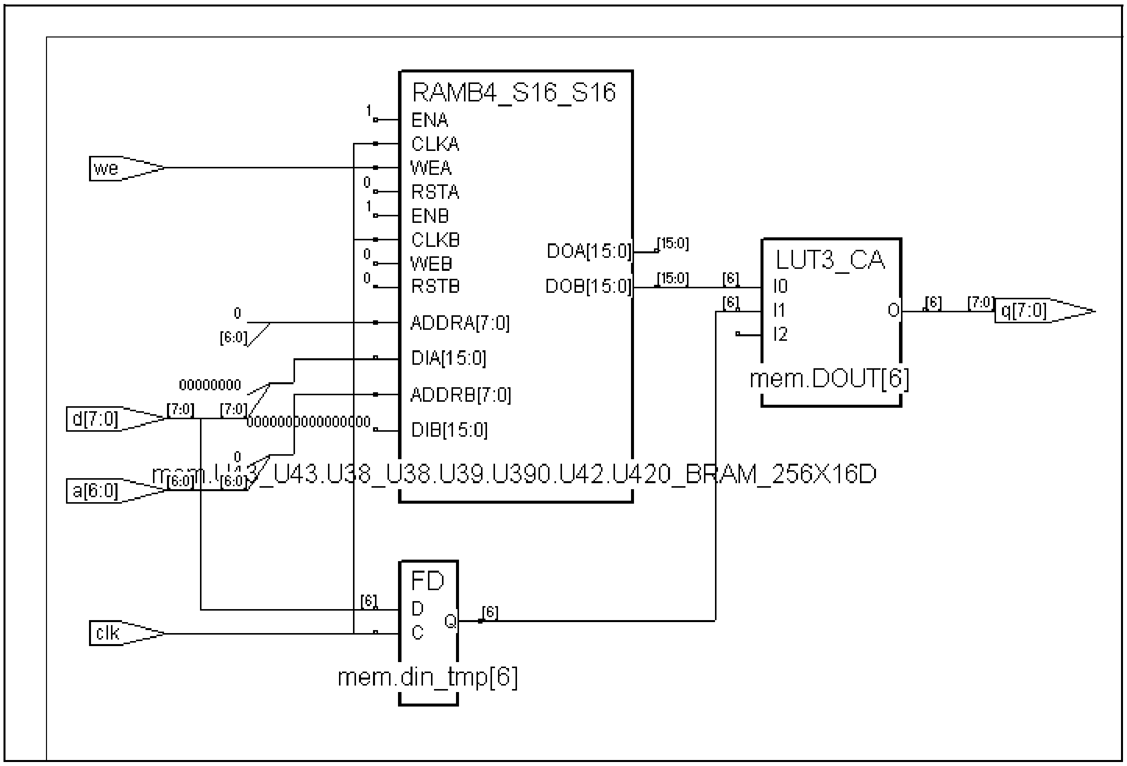

Figure 15: Flow 2: Xilinx block SelectRAM with by-pass logic

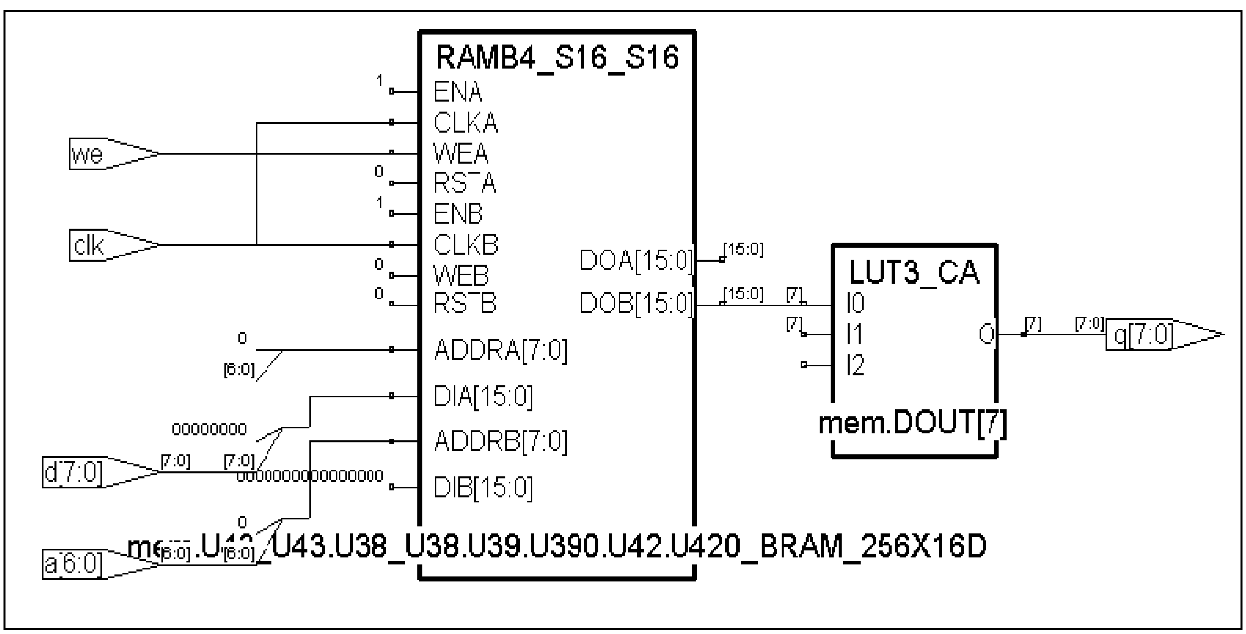

Figure 16: Flow 2: Level 1. Filtered view of registered data output path when READ-WRITE conflict occurs.

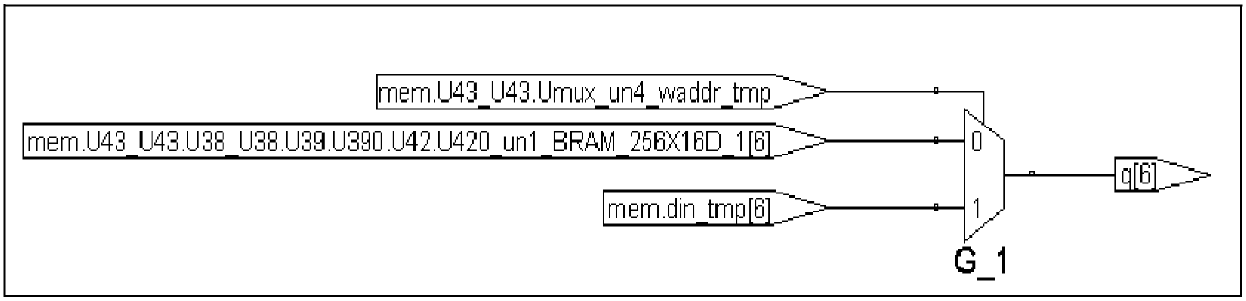

Figure 17: Flow 2: Level 1. Stage 4: Mux select is controlled by the previous AND gate stage driving RAM data out READ-WRITE conflict resolved (zoom of LUT3_CA).

Coding Style Mapped to Single-Port Block SelectRAMs

To infer a single-port block SelectRAM, *all* of the following conditions must be true:

- The read and write clocks must be the same
- The read and write addresses must be the same
- The enable signals are the same
- The write enable signals are the same

In addition to the support for block SelectRAMs in Virtex/VirtexE, Virtex-II block SelectRAM supports three modes:

- WRITE_FIRST
- READ_FIRST
- NO_CHANGE

These modes determine output of the RAM when write enable is active.

WRITE_FIRST refers to the behavior that when write enable (WE) is active, data output (DO) uses the value of data input (DI).

READ_FIRST refers to the behavior that when write enable (WE) is active, data output (DO) uses the value of the memory content.

NO_CHANGE refers to the behavior that when write enable (WE) is active, data output (DO) remains the same.

Note: These modes are passed as the WRITE_MODE property in the EDIF file. The RAM also can be reset by any pattern other than 0. Whatever the specified reset pattern is, the reset pattern is passed to the EDIF as a SRVAL property to Xilinx.

For Virtex/Virtex E, only WRITE_FIRST mode is supported.

Here are some examples of the new RAM coding styles supported from the Synplify Pro 7.0 release forward. The examples cover three modes of the Virtex block SelectRAM along with the extended support of the reset and enable signals from the Synplify Pro 7.0 release forward.

WRITE_FIRST Mode Example

The following example of the WRITE_FIRST mode has both enable and reset, with enable taking precedence. (Virtex/VirtexE, Virtex-II)

```
module ram_test(data_out, data_in, addr, clk, rst, en, we);

output [7:0]data_out;
input [7:0]data_in;
input [6:0]addr;
input clk, en, rst, we;

reg [7:0] mem [127:0] /* synthesis syn_ramstyle = “block_ram”*/;

always@(posedge clk)
if(en)
 if(rst == 1)
 data_out = 0;
 else
 if(we == 1)
  data_out = data_in;
  else
  data_out = mem[addr];

always @(posedge clk)
    if (en & we) mem[addr] = data_in;

endmodule
```

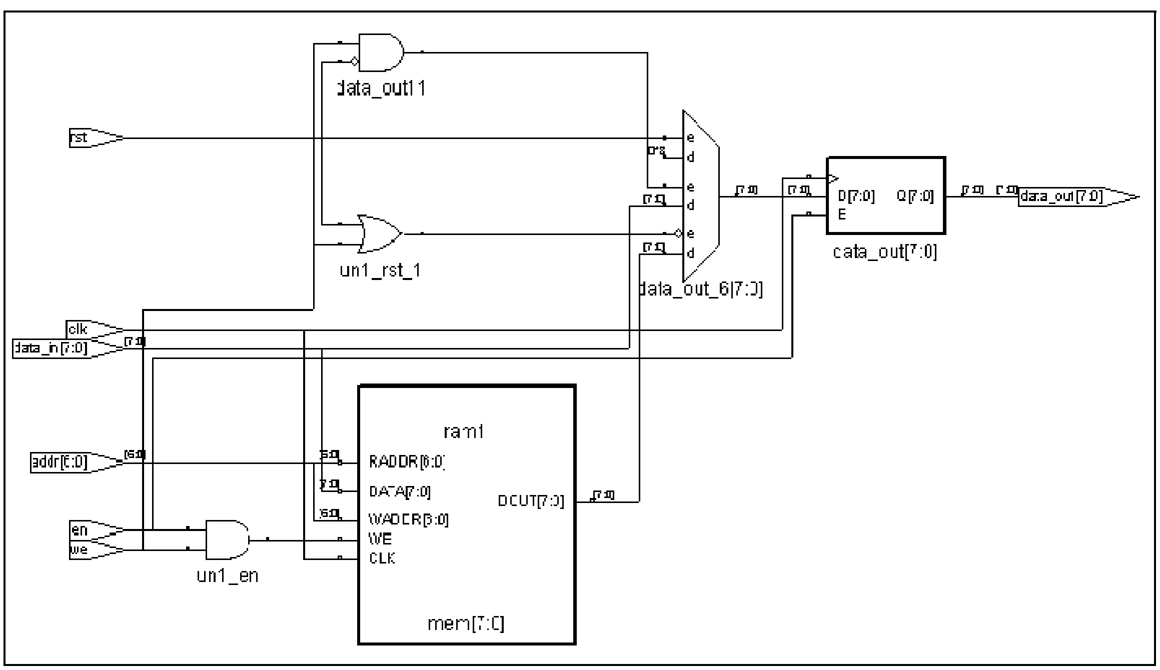

Figure 18: HDL Analyst RTL view of WRITE_FIRST Mode RAM with output registered, enable and reset inferred

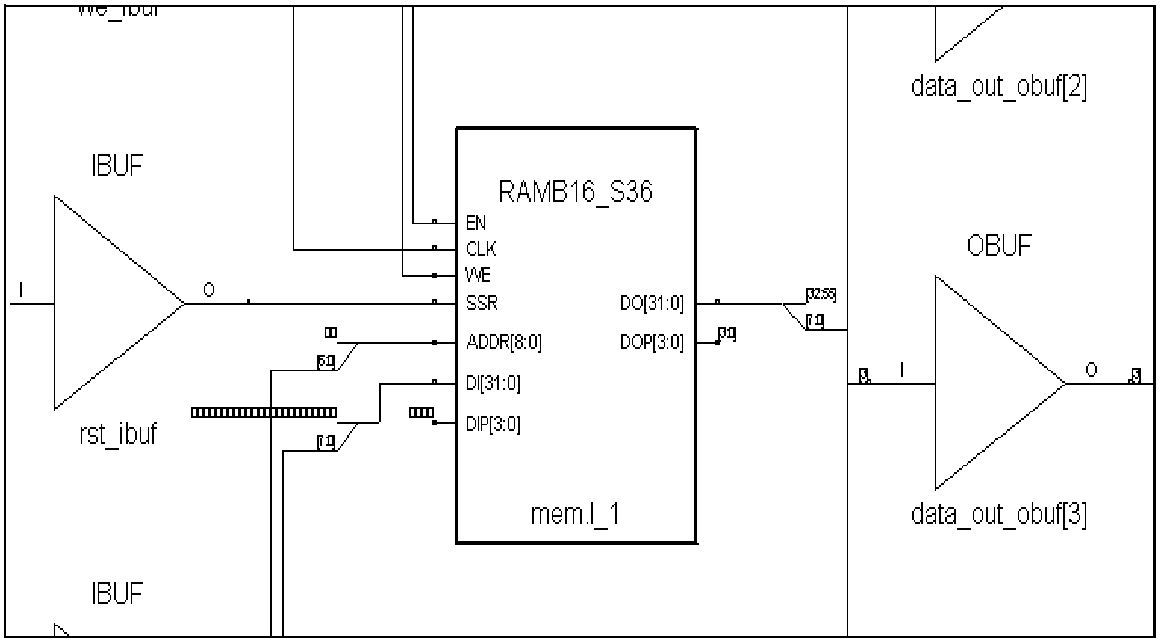

Figure 19: HDL Analyst Technology view of the inferred RAM mapped to block SelectRAM

READ_FIRST Mode Example

The following example of READ_FIRST mode with both enable and reset, has reset taking precedence.

This example is for Virtex-II only.

```
module ram_test(data_out, data_in, addr, clk, rst, en, we);

output [7:0]data_out;
input [7:0]data_in;
input [6:0]addr;
input clk, en, rst, we;

reg [7:0] mem [127:0] /* synthesis syn_ramstyle = "block_ram" */;
reg [7:0] data_out;

always@(posedge clk)
  if(rst == 1)
    data_out = 0;
  else begin
    if(en) begin
      data_out = mem[addr];
    end
  end

always @(posedge clk)
  if (en & we) mem[addr] = data_in;

endmodule
```

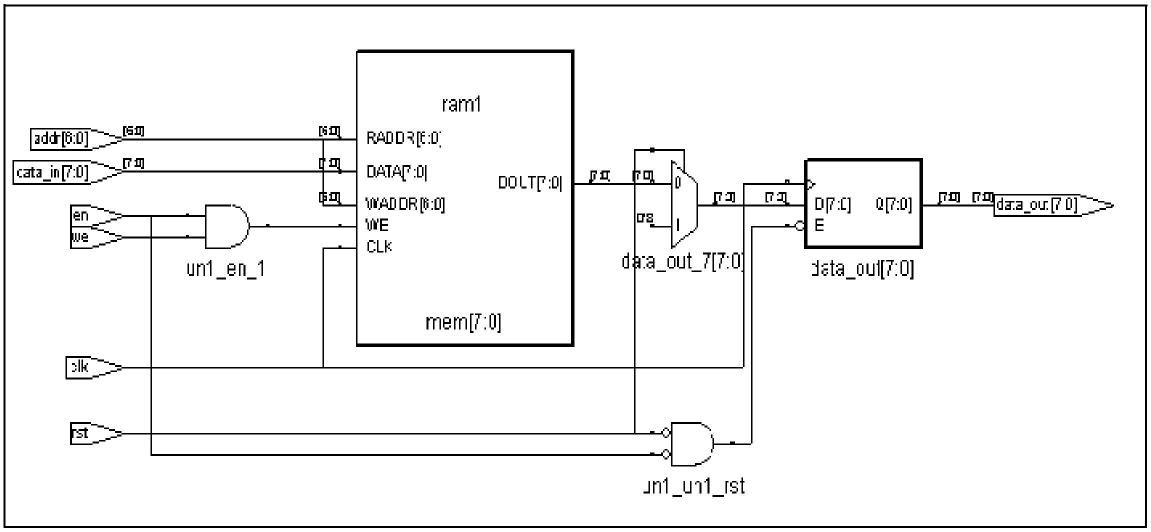

Figure 20: HDL Analyst RTL view of READ_FIRST Mode RAM with output registered, reset and enable inferred

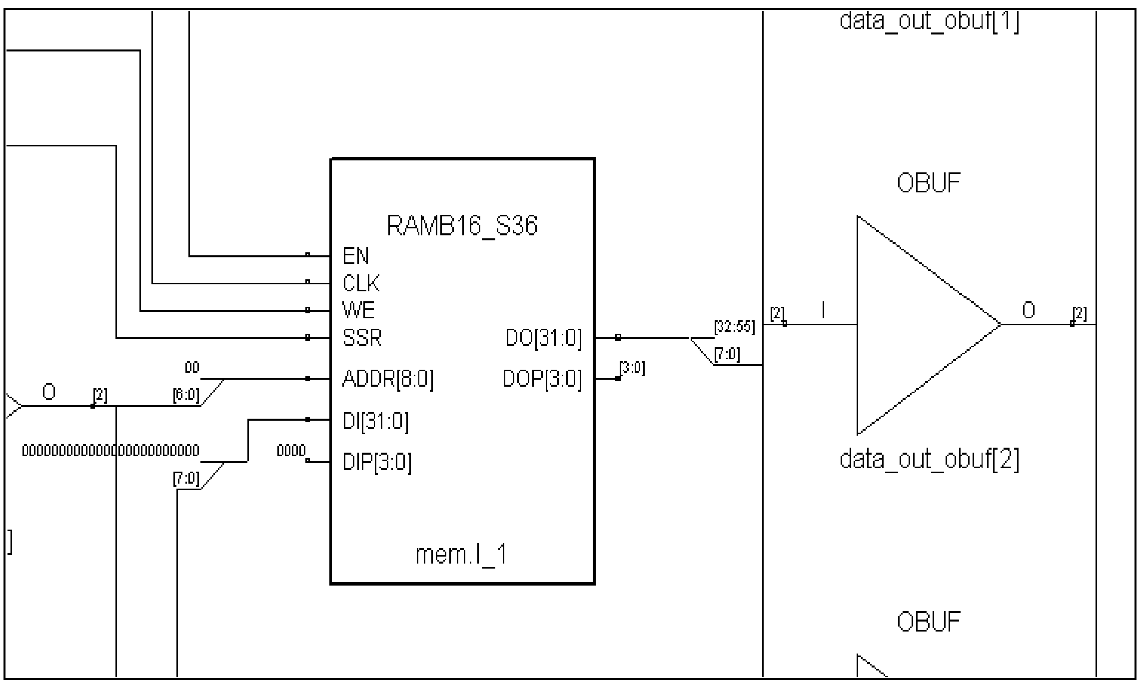

Figure 21: HDL Analyst Technology view of inferred RAM mapped to block SelectRAM

NO_CHANGE Mode Example

The following NO _CHANGE mode example has neither enable nor reset.

This is for Virtex-II only.

```
module ram_test(data_out, data_in, addr, clk, rst, en, we);

output [7:0]data_out;
input [7:0]data_in;
input [6:0]addr;
input clk, en, rst, we;

reg [7:0] mem [127:0] /* synthesis syn_ramstyle = "block_ram" */;
reg [7:0] data_out;

always@(posedge clk)
  if(we == 1)
    data_out = data_out;
  else
    data_out = mem[addr];

always @(posedge clk)
  if (we) mem[addr] = data_in;

endmodule
```

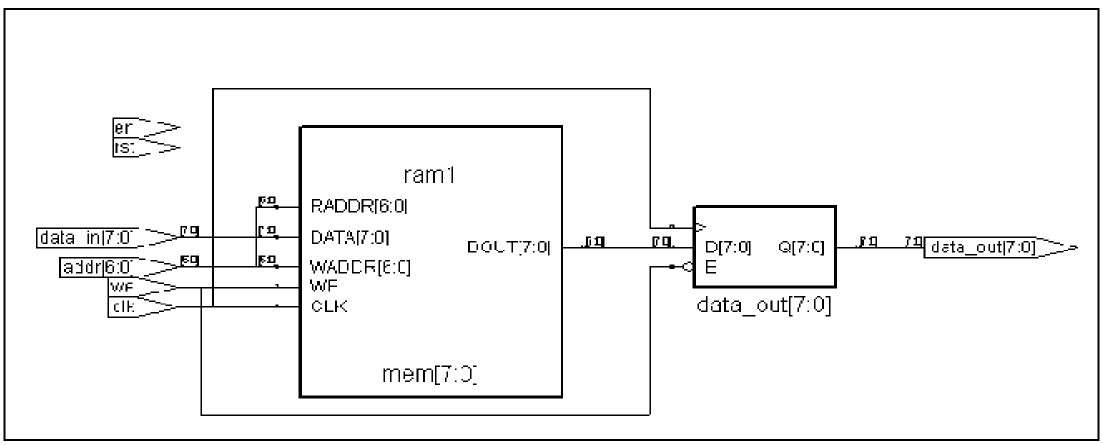

Figure 22: HDL Analyst RTL view of No_Change Mode – RAM with no reset or enable inferred

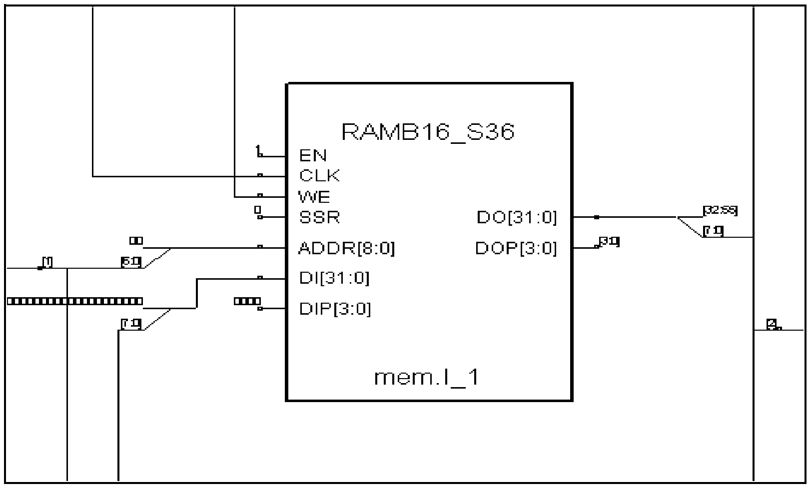

Figure 23: HDL Analyst Technology view of No_Change Mode inferred RAM mapped to block SelectRAM

Note: block SelectRAM also is inferred if both the read address and the output address are registered.

Dual-Port RAM Styles Mapped to Dual-Port Block SelectRAMs (Virtex-II)

One of the following conditions must exist for the various coding styles to infer dual-port block SelectRAMs:

- read and write addresses are different
- read and write clocks can be different
- enable signals can be different.

WRITE_FIRST mode is the only mode supported for dual-port block SelectRAMs in Virtex-II.

Dual-Port Block SelectRAM Example

The following dual-port block SelectRAM example has the output registered and the read port has both enable and reset with enable taking precedence.

```
module (data_out, data_in, addr_out, addr_in, clk_r, clk_w,en_w, we,
rst, en_r);

output [7:0]data_out;
input [7:0] data_in;
input [6:0] addr_in, addr_out;
input en_r, en_w, we, rst, clk_r, clk_w;

reg [7:0] mem [127:0] /* synthesis syn_ramstyle = "block_ram" */;
reg [7:0]data_out;

always@(posedge clk_r)
  if(en_r)
    if(rst == 0)
      data_out = mem[addr_out];
    else
      data_out = 0;

always @(posedge clk_w)
if (en_w & we) mem[addr_in] = data_in;

endmodule
```

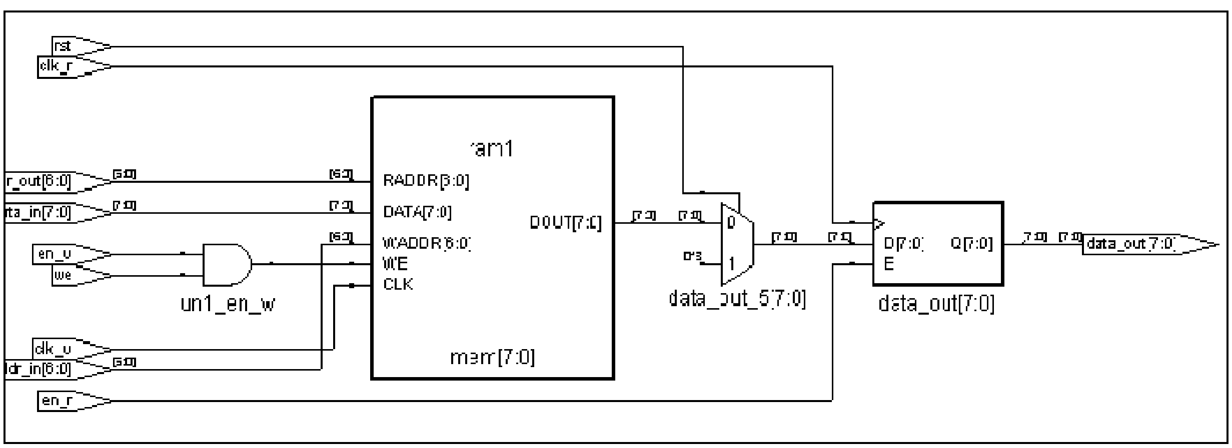

Figure 24: HDL Analyst RTL view of WRITE_FIRST Mode of dual-port SelectRAM with output registered, reset, and enable

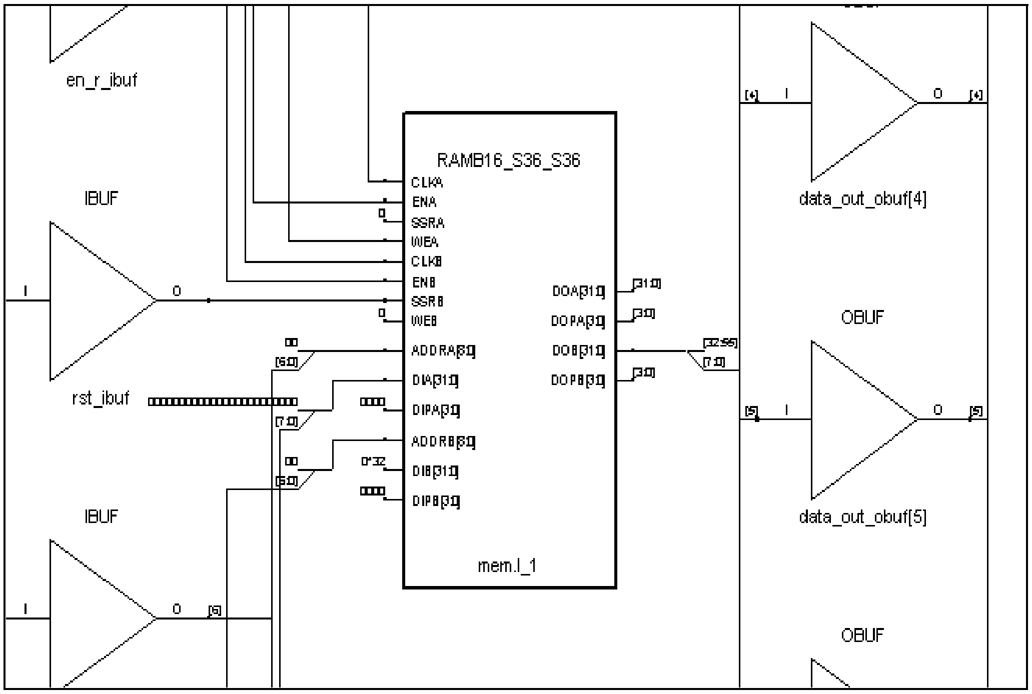

Figure 25: HDL Analyst Technology view of an inferred RAM mapped to dual-port block SelectRAM

Synplicity, Inc.
935 Stewart Drive, Sunnyvale, CA 94085 USA
Phone: (U.S.) +1 408 215-6000, Fax: (U.S.) +1 408 990-0290
www.synplicity.com

Simply Better Results