Inferring RAM in Synplify

by Allen Drost, Corporate Applications Engineering Group Manager, allen@synplicity.com

Synplicity has added automatic RAM inferencing to Synplify version 5.0.5. Now, you no longer need to manually instantiate RAM as a black box or Xilinx-specific primitive; you can make designs that are truly technology independent.

This article describes how to successfully incorporate RAM into your next VHDL or Verilog design with Synplify. Synplify integrates both RAM timing estimates and regular timing constraints to efficiently optimize your next design, and includes:

- Automatic synchronous RAM inferencing
- SelectRAM™ or register implementations
- Timing estimates on RAM blocks
- Flexible coding styles

RAM Implementation

Synplify v5.0.5 can automatically infer RAM structures when coded as an indexed array or as a CASE statement. However, it will infer only synchronous RAMs; asynchronous RAMs are not supported. See the following examples for a suggestion on coding styles for RAM blocks.

When a RAM block is recognized, Synplify will automatically implement the circuit using RAM16X1S, RAM32X1S, and RAM16X1D Xilinx RAM primitives. If a RAM block is more complex than a single 16x1 primitive, Synplify creates the necessary write enable and data multiplexing logic to implement the circuit using multiple RAM blocks. HDL Analyst displays a RAM block in the RTL view, making the schematic view easier to read. To see how the large RAM blocks are implemented, use the technology view in HDL Analyst, as shown in Figure 1.

If you want to map your RAM into standard logic and registers, you can disable the usage of Xilinx SelectRAM™, by setting the syn_ramstyle attribute to "registers." This attribute can be applied directly in the HDL source code or through Synplify's Synthesis Constraint OPtimization Environment (SCOPE™).

To effectively optimize designs that incorporate RAM, the synthesis tool must understand the timing delays through the RAM blocks for the critical path optimization. Synplify understands the timing characteristics of the RAM primitives, and includes all RAM delays in the analysis and optimization of critical paths.

"Inferring RAM is the most effective method of designing memory into your FPGA design. Synplify provides a full suite of features to implement, analyze, and optimize your RAM design."

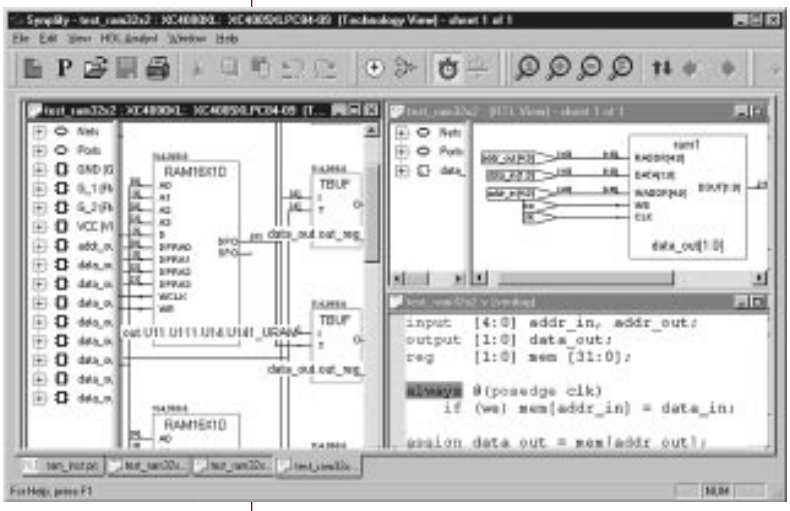

Figure 1: The Technology View in HDL Analyst

Examples

(**Note:** Synplify supports inferencing of single and dual ported RAM in VHDL and Verilog, using either indexed arrays or case statements)

Example 1: Single-ported VHDL

```
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;
```

Example 2: Dual-ported VHDL

```
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    : in std_logic_vector(2 downto 0);
addr_out: 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_out));
  process (clk, we, addr_in) begin
    if (rising_edge (clk)) then
      if (we = '1') then
        mem(conv_integer(addr_in)) <= d;
      end if;
    end if;
  end process;
end rtl;
```

Example 3: Single-ported Verilog

```
module test_ram32x2 (clk, we, addr, data_in,
 data_out);
input clk, we;
input [1:0] data_in;
input [4:0] addr;
output [1:0] data_out;
reg [1:0] mem [31:0];

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

assign data_out = mem[addr];
endmodule
```

Example 4: Dual-ported Verilog

```
module ram4x4(z, raddr, d, waddr, we, clk);
output [3:0] z;
input [3:0] d;
input [1:0] raddr, waddr;
input we;
input clk;
reg [3:0] z;

reg [3:0] mem0, mem1, mem2, mem3;

always @(mem0 or mem1 or mem2 or mem3 or
 raddr)
begin
case (raddr[1:0])
  4'b00: z = mem0;
  4'b01: z = mem1;
  4'b10: z = mem2;
  4'b11: z = mem3;
  endcase
end

always @(posedge clk) begin
  if(we) begin
    case (waddr[1:0])
    4'b00: mem0 = d;
    4'b01: mem1 = d;
    4'b10: mem2 = d;
    4'b11: mem3 = d;
    endcase
  end
end

endmodule
```

Conclusion

Inferring RAM is the most effective method of designing memory into your FPGA design. Synplify provides a full suite of features to implement, analyze, and optimize your RAM design.

For more information please see the Synplicity website at www.synplicity.com.