5.6 Modeling Read-Only Memories (ROM)

Asynchronous ROMs shall be modeled as combinational logic using any of the following styles:

1. CASE statement of address with assignment to vector output (section 5.6.1).
2. Initial statement of a memory array with values defined in Verilog code (section 5.6.2).
3. Initial statement with memory initialized with $readmemb or $readmemh. ROM data shall be in a text file (section 5.6.3).

Synchronous ROM with output data provided at a specific clock edge shall be modeled in two separate blocks, one block for the asynchronous ROM model, and another block for the registering of the asynchronous ROM output.

Two ROM style attributes shall be used to help guide the synthesis process. The recommended attributes shall be:

/* syn_romstyle = block_rom */	Infers ROM cells
/* syn_romstyle = logic */	Infers combinational logic instead of ROM

(Cliff Cummings will provide a new definition for the attributes per LRM. The ones defined here are used as examples only)

5.6.1 ROM Using One-Dimensional Array with CASE Statement

The values of ROM shall be defined within a CASE statement. A single CASE statement within an always block shall be allowed to define a ROM. In addition, the assigned ROM output signal shall not be assigned a value in any other block.

Example:

```
module rom_case(z, a);
 output [3:0] z; // 4 wide
 input [2:0]  a;  // address.   8 deep memory
 reg [3:0]    z /* syn_romstyle = block_rom */;
 // NOTE:  To infer combinational logic instead of ROM, use
 // /* syn_romstyle = logic */
 always @(a) begin
   case (a)
     3'b000: z = 4'b1011;
     3'b001: z = 4'b0001;
     3'b100: z = 4'b0011;
     3'b110: z = 4'b0010;
     3'b111: z = 4'b1110;
     default: z = 4'b0000;
   endcase
 end
endmodule // rom_case
```

5.6.2 ROM Using Two-Dimensional Array with Data in Initial Statement

The values of the ROM shall be defined within a two-dimensional array using the "INITAL" statement to identify the values. The following restrictions shall apply:

1. A single INITIAL statement shall be used to specify the ROM, and no other assignments shall be included within that INITIAL statement.
2. The ROM shall be defined as a two-dimensional array.
3. The assigned ROM output signal shall not be assigned a value in any other block.
4. An attribute shall be necessary to identify the ROM style (*block_rom* or *logic*).

Example:

```
module rom_2dimarray_inital (z, a);
 output [3:0] z;
 input [2:0]  a;

 reg [3:0]    rom[0:7] /* syn_romstyle = block_rom */;
 // declares a memory  rom of 8 4-bit registers. The indices are 0 to 7

 // NOTE:  To infer combinational logic instead of ROM, use
 // /* syn_romstyle = logic */
 initial begin
   rom[0] = 4'b1011;
   rom[1] = 4'b0001;
   rom[2] = 4'b0011;
   rom[3] = 4'b0010;
   rom[4] = 4'b1110;
   rom[5] = 4'b0111;
   rom[6] = 4'b0101;
   rom[7] = 4'b0100;
 end

 assign z = rom[a];
endmodule
```

Note to group: It seems that the style can be used to define plain constants. Do we need that the ROM be a two-dimensional array? For example:
reg [3:0] add / syn_romstyle = logic */;*
reg [3:0] plus / syn_romstyle = logic */;*
initial begin
add = 4'b1010;
plus = 4'1000;
end

assign w = (opcode == add) ? plus : 4'b0000;
Do we want to restrict this?

5.6.3 ROM Using Two-Dimensional Array with Data in Text File

The values of ROM shall be defined within a two-dimensional array using the INITAL statement and with the values defined in text files. The system functions *$readmemb* or *$readmemh* shall be used to identify the file. The format of the file shall be as defined in the *$readmemb* or *$readmemh* system functions. The following restrictions shall apply:

1. A single INITIAL statement shall be used to specify the ROM, and no other assignments shall be included within that INITIAL statement.
2. The ROM shall be defined as a two-dimensional array.
3. The assigned ROM output signal shall not be assigned a value in any other block.
4. An attribute shall be necessary to identify the ROM style (*block_rom* or *logic*).

Example:

```
// ROM module using two dimensional arrays with
// memory defined in text file with $readmemb or $readmemh
// NOTE: This style can lead to simulation/synthesis mismatch
//       if the content of data file changes after synthesis
module rom_2dimarray_initial_readmem (z, a);
 output [3:0] z;
 input [2:0]  a;
 reg [3:0]    rom[0:7] /* syn_romstyle = block_rom */;
 // declares a memory rom of 8 4-bit registers. The indices are 0 to 7

 // NOTE:  To infer combinational logic instead of ROM, use
 // /* syn_romstyle = logic */
 initial begin
   $readmemb("rom.data", rom);
   //
 end

 assign z = rom[a];

endmodule
```

5.7 Modeling Random Access Memories (RAM)

Synchronous RAMs shall be modeled as a two-dimensional array. The data entry into the RAM shall be modeled as a clocked block, and the asynchronous READ shall be modeled as either an assign statement or a combinational block.

Two RAM style attributes shall be used to help guide the synthesis process. If a RAM is defined as a set of primitives with latches, and if the RAM style infers a LATCH implementation method, then NO LATCH warnings shall be emitted by the synthesis tool *(do we really want this lack of warnings?).*

The following restrictions shall apply:

1. A single ALWAYS statement shall be used to specify the RAM, and no other assignments shall be included within that ALWAYS statement.
2. The RAM shall be defined as a two-dimensional array.
3. The assigned RAM output signal shall not be assigned a value in any other block.
4. An attribute shall be necessary to identify the RAM style (*block_ram* or *register*).

The recommended attributes shall be:

```
/* syn_ramstyle = block_ram */  Infers RAM cells
/* syn_ramstyle = registers */    Infers registers instead of RAM

module ram_test(q, a, d, we, clk);
output [7:0] q;
input  [7:0] d;
input  [6:0] a;
input  clk, we;
reg [7:0] mem [127:0] /* synthesis syn_ramstyle="registers" */;
always @(posedge clk) begin
   if(we)
       mem[a] <= d;
   end

assign q = mem[a];
endmodule


//Ram implemented with latches
module ramlatch (q, a, d, we);
 output [7:0] q;  // output
 input [7:0]  d;  // data input
 input [6:0]  a;  // address
 input        we; // clock and write enable
 reg [7:0]    mem [127:0] /* syn_ramstyle = block_ram */;
 // memory 128 deep, 8 wide
 always @(/*AUTOSENSE*/a or d or we) begin
   if(we)
     mem[a] <= d;
 end

 assign q = mem[a];
endmodule
```