1. *Constant Definition*

A variable shall be interpreted as a constant if the value of the expression (rhs) can be computed statically. Example:

```
reg  [15:0] r;

initial begin
  r = 16'h000A;
  r2 = 3* r;
end
```

2. *Modeling Read-Only Memories (ROM)*

Asynchronous ROMs objects may 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). The case statement shall be within an *always* block.
2. *Initial* statement of a memory array with values defined in Verilog code (section 5.6.2). A ROM object may be generated when the memory array is read from within a procedural block. The ROM object shall not be written in any other procedural block or continuous assignment.
3. *Initial* statement with memory initialized with *$readmemb* or $*readmemh*. ROM data shall be in a text file (section 5.6.3). A ROM object may be generated when the memory array is read from within a procedural block. The ROM object shall not be written in any other procedural block or continuous assignment.

The ROM object may be specified from within a function or a task. However, the definition of the ROM values shall be specified within an *initial* block.
The rom_block attribute shall be used to help guide the synthesis process. This attribute shall precede the declaration of the ROM object.

5.6.1 ROM Using One-Dimensional Array with case Statement

The data values of a ROM shall be defined within a case statement. All the values of the ROM shall be defined within the case statement. The value assigned to each ROM address shall be a static expression that can be evaluated at compile time.
Example:

```
module rom_case(z, a);
  output [3:0] z; // 4 wide
  input  [2:0] a; // address- 8 deep memory
  (* synthesis, rom_block = "ROM_CELL XYZ01" *)
  reg    [3:0] z;

  always @* begin // @(a)
    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 within Initial Block

The values of the ROM may be defined within a constant defined as a two-dimensional array. A *rom_block* or *logic_block* attribute may be necessary to guide the synthesis style of the synthesis tool. Uninitialized values shall have a don't care assignment, and can be assigned by the compiler to a value of 1 or 0 (always true in synthesis).

Example:

```
module rom_2dimarray_inital (z, a);
  output [3:0] z;
  input  [2:0] a;
  (* synthesis, rom_block = "ROM_CELL XYZ01" *)
  reg    [3:0] rom[0:7];
  // declares a memory rom of 8 4-bit registers. The indices are 0 to 7

  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: If combinational logic is desired, replace the (* synthesis, rom_block = "ROM_CELL XYZ01" *) attribute with the (* synthesis, logic_block *) attribute in the above example.

```
module rom_array_function (z, q, a, b);
  output [3:0] z, q;
  input  [2:0] a, b;
  reg    [3:0] z;
  wire   [3:0] q;

  function [3:0] lookup;
    input [2:0] i;
    (* synthesis, rom_block = "ROM_CELL XYZ01" *)
    reg   [3:0] rom[0:7];
    begin
      lookup = rom[i];
    end
  endfunction // lookup

  initial begin
    lookup.rom[0] = 4'b1011;
    lookup.rom[1] = 4'b0001;
    lookup.rom[2] = 4'b0011;
    lookup.rom[3] = 4'b0010;
    lookup.rom[4] = 4'b1110;
    lookup.rom[5] = 4'b0111;
    lookup.rom[6] = 4'b0101;
    lookup.rom[7] = 4'b0100;
  end

  always @(a)  z = lookup(a);

  assign q = lookup(b);
endmodule
```

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

  task lookup;
    input [2:0] i;
    (* synthesis, rom_block = "ROM_CELL XYZ01" *)
    reg   [3:0]  rom[0:7];
    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;
      z      = rom[i];
    end
  endtask // lookup

  initial begin
    lookup.rom[0] = 4'b1011;
    lookup.rom[1] = 4'b0001;
    lookup.rom[2] = 4'b0011;
```

```
    lookup.rom[3] = 4'b0010;
    lookup.rom[4] = 4'b1110;
    lookup.rom[5] = 4'b0111;
    lookup.rom[6] = 4'b0101;
    lookup.rom[7] = 4'b0100;
  end

  always @(a) lookup(a);
endmodule
```

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

The values of a ROM may be defined within a two-dimensional array using the *initial* statement, functions, and tasks 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. The ROM shall be defined as a two-dimensional array.
2. An attribute may be necessary to identify the ROM style (*synthesis, rom_block *) or (*synthesis, logic_block *).
3. Uninitialized values shall have a don't care assignment, and can be assigned by the synthesis tool a value of 1 or 0.

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;
  // declares a memory rom of 8 4-bit registers.
  //The indices are 0 to 7
  (* synthesis, rom_block = "ROM_CELL XYZ01" *)
  reg    [3:0] rom[0:7]
  // NOTE:  To infer combinational logic instead of a ROM, use
  // (* synthesis, logic_block *)


  initial $readmemb("rom.data", rom);

  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 may be used to help guide the synthesis process. The following restrictions shall apply:

1. The RAM shall be defined as a two-dimensional array.
2. An attribute may be necessary to identify the RAM style (* synthesis, ram_block *) for RAM cells, or (* synthesis, reg_block *) for registers.

Example:

```
module ram_test(q, a, d, we, clk);
  output [7:0] q;
  input  [7:0] d;
  input  [6:0] a;
  input        clk, we;
  (* synthesis, ram_block *)
  reg    [7:0] mem [127:0];


  always @(posedge clk) if(we) mem[a] <= d;

  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
  (* synthesis, ram_block *)
  reg    [7:0] mem [127:0];
  // memory 128 deep, 8 wide

  always @* if(we) mem[a] <= d;

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