

Example 8—A module of memory dimm

```

module dimm (adr, ba, rasx, casx, csx, wex, dqm, cke, data, clk,
              dev_id);

    parameter [31:0] MEM_SIZE = 8, // in mbytes
              MEM_WIDTH = 16;

    input [10:0] adr;
    input      ba;
    input      rasx, casx, csx, wex;
    input [ 7:0] dqm;
    input      cke;
    inout [63:0] data;
    input      clk;
    input [ 4:0] dev_id;

    genvar      i;

    generate
    case ({MEM_SIZE, MEM_WIDTH})
    {32'd8, 32'd16}: // 8Meg x 16 bits wide.
    begin
        // The generated instance names are word[3].p, word[2].p,
        // word[1].p, word[0].p, and the task read_mem
        for (i=0; i<4; i=i+1) begin:word
            sms_16b216t0 p(.clk(clk), .csb(csx), .cke(cke),
                          .ba(ba[0]), .addr(adr[10:0]), .rasb(rasx),
                          .casb(casx), .web(wex), .udqm(dqm[2*i+1]),
                          .ldqm(dqm[2*i]), .dqi(data[15+16*i:16*i]),
                          .dev_id(dev_id[4:0]));
        end
        task read_mem;
            input [31:0] address;
            output [63:0] data;
            begin
                word[3].p.read_mem(address, data[63:48]);
                word[2].p.read_mem(address, data[47:32]);
                word[1].p.read_mem(address, data[31:16]);
                word[0].p.read_mem(address, data[15: 0]);
            end
        endtask
    end

```

```

{32'd16, 32'd8}: // 16Meg x 8 bits wide.
begin
  // The generated instance names are byte[7].p, byte[6].p,
  // byte[5].p, byte[4].p, byte[3].p, byte[2].p, byte[1].p,
  // byte[0].p and the task read_mem
  for (i=0; i<8; i=i+1) begin:byte
    sms_16b208t0 p(.clk(clk), .csb(csx), .cke(cke),
                  .ba(ba[0]), .addr(adr[10:0]), .rasb(rasx),
                  .casb(casx), .web(wex), .dqm(dqm[i]),
                  .dqi(data[7+8*i:8*i]), .dev_id(dev_id[4:0]));
  end
  task read_mem;
  input  [31:0] address;
  output [63:0] data;
  begin
    byte[7].p.read_mem(address, data[63:56]);
    byte[6].p.read_mem(address, data[55:48]);
    byte[5].p.read_mem(address, data[47:40]);
    byte[4].p.read_mem(address, data[39:32]);
    byte[3].p.read_mem(address, data[31:24]);
    byte[2].p.read_mem(address, data[23:16]);
    byte[1].p.read_mem(address, data[15: 8]);
    byte[0].p.read_mem(address, data[ 7: 0]);
  end
endtask
end
// Other memory cases ...
endcase
endgenerate
endmodule

```

12.2 Overriding module parameter values

There are two different ways that parameters can be defined. The first is the *module_parameter_port_list* (see 12.1), and the second is as a *module_item* (see 3.11). A module declaration can contain parameter definitions of either or both types, or no parameter definitions.

A module parameter can have a type specification and a range specification. The effect of parameter overrides on a parameter's type and range shall be in accordance with the following rules:

- A parameter declaration with no type or range specification shall default to the type and range of the final override value assigned to the parameter.
- A parameter with a range specification, but with no type specification, shall be the range of the parameter declaration and shall be unsigned. An override value shall be converted to the type and range of the parameter.
- A parameter with a type specification, but with no range specification, shall be of the type specified. An override value shall be converted to the type of the parameter. A signed parameter shall default to the range of the final override value assigned to the parameter.
- A parameter with a signed type specification and with a range specification shall be a signed, and shall be the range of its declaration. An override value shall be converted to the type and range of the parameter.

Examples:

```

module generic_fifo
  #(parameter MSB=3, LSB=0, DEPTH=4) // These parameters can be
  overridden

```