**SYSTEM VERILOG PROGRAMS**

**MELVIN RIJOHN T**

Contents

[Array Types 3](#_Toc182546404)

[Process, Task & Functions 4](#_Toc182546405)

[MUX 2x1 5](#_Toc182546406)

[Data Types 6](#_Toc182546407)

[Events 7](#_Toc182546408)

[Extern Keyword (Task Example) 8](#_Toc182546409)

[Extern Keyword (Function Example) 9](#_Toc182546410)

[Parameterized Class 9](#_Toc182546411)

[Pre-Post Randomization 10](#_Toc182546412)

[Randomization Constraints Enable/Disable 11](#_Toc182546413)

[Constrained Randomization 11](#_Toc182546414)

[Randomize Constraints Distribution 12](#_Toc182546415)

# Array Types

module array\_types();

    int arr[3] = {20,40,34};

    string arr1[3] = {"Hello","World","!"};

    string arr2[];

    int arr3[string];

    initial begin

        arr2 = new[4];

        arr2 = {"Hello","vlsi","world"};

        arr3["RED"] = 128;

        arr3["GREEN"] = 230;

        arr3["BLUE"] = 10;

        $display("/\*\*\*\* Simple Integer Array \*\*\*\*/");

        foreach(arr[i]) begin

            $display("arr[%0d]: %0d",i, arr[i]);

        end

        $display("/\*\*\*\* Simple String Array \*\*\*\*/");

        foreach(arr1[i]) begin

            $display("arr1[%0d]: %0s",i, arr1[i]);

        end

        $display("/\*\*\*\* Dynamic Array \*\*\*\*/");

        foreach(arr2[i]) begin

            $display("arr2[%0d]: %0s",i, arr2[i]);

        end

        $display("/\*\*\*\* Associative Array \*\*\*\*/");

        $display("arr3[RED]: %0d", arr3["RED"]);

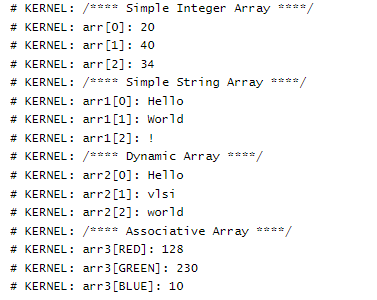
        $display("arr3[GREEN]: %0d", arr3["GREEN"]);

        $display("arr3[BLUE]: %0d", arr3["BLUE"]);

    end

endmodule

OUTPUT



# Process, Task & Functions

module process\_task();

    int a,b,c,sum;

    task t1(int x, int y);

        begin

            #10;

            $display("Sum: %0d", a+b);

        end

    endtask

    task t2(int x, int y);

        begin

            #10;

            $display("Difference: %0d", a-b);

        end

    endtask

    function int f1(int x, int y, int z);

        begin

            f1 = x + (z - y);

        end

    endfunction

    initial begin

        a = 37; b = 8; c = 66; #10;

        $display("/\*\*\*\*\* Initial Values \*\*\*\*/");

        $display("a = %0d b = %0d c = %0d", a,b,c);

        $display("/\*\*\*\*FORK JOIN\*\*\*\*/");

        fork

            t1(a,b);

            t2(a,b);

        join

        $display("Sum & Differnce: %0d", f1(a,b,c));

         $display("/\*\*\*\*FORK JOIN ANY\*\*\*\*/");

        fork

            t1(a,b);

            t2(a,b);

        join\_any

        $display("Sum & Differnce: %0d", f1(a,b,c));

         $display("/\*\*\*\*FORK JOIN NONE\*\*\*\*/");

        fork

            t1(a,b);

            t2(a,b);

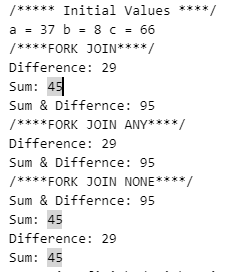
        join\_none

        $display("Sum & Differnce: %0d", f1(a,b,c));

    end

endmodule

OUTPUT



# MUX 2x1

module mux\_2x1 (

    input logic a,

    input logic b,

    input logic s,

    output logic y

);

    assign  y = (s ? b : a);

endmodule

module mux\_2x1\_tb ();

    logic a,b,s,y;

    mux\_2x1 dut(a,b,s,y);

    initial begin

        $dumpfile("out.vcd");

        $dumpvars(0, mux\_2x1\_tb);

        $monitor("a=%0d b=%0d s=%0d y=%0d",a,b,s,y);

        a=0; b=0; s=0; #10;

        a=0; b=1; s=0; #10;

        a=1; b=0; s=0; #10;

        a=1; b=1; s=0; #10;

        a=0; b=0; s=1; #10;

        a=0; b=1; s=1; #10;

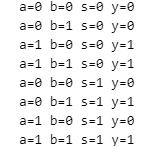
        a=1; b=0; s=1; #10;

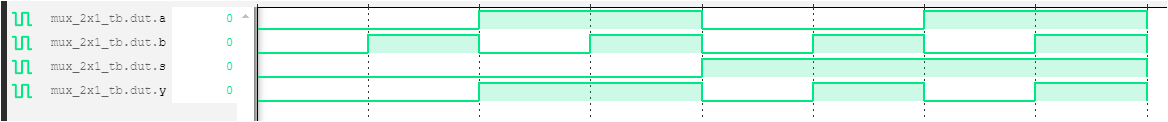
        a=1; b=1; s=1; #10;

    end

endmodule

OUTPUT





# Data Types

module dataTypes\_tb ();

    logic[7:0] a,b;

    logic [7:0] c,d;

    string e,g;

    bit[31:0] f = 128;

    typedef struct packed {

        int RED;

        int GREEN;

        int BLUE;

    } RGB\_color;

    typedef struct{

        int RED;

        int GREEN;

        int BLUE;

        string ALPHA;

    } RGBA\_color;

    typedef union packed {

        int i;

        int s;

    } something;

    class Printer;

        function void log(string msg);

            $display(msg);

        endfunction

    endclass

    RGB\_color rgb; //struct

    RGBA\_color rgba; //unpacked struct

    something some; //union

    Printer console; //class

    initial begin

        a=5; b=10;

        c = a + b;

        d = c - a;

        g = "Hello";

        rgb.RED = 122;

        rgb.GREEN = 233;

        rgb.BLUE = 111;

        rgba.RED = 122;

        rgba.GREEN = 233;

        rgba.BLUE = 111;

        rgba.ALPHA = "120";

        some.i = 0;

        e = $sformatf("%0d", f); //converts bit value to string

        $display("a=%0d b=%0d c=%0d d=%d e=%0s f=0x%0h",a,b,c,d,e,f);

        $display("Len: %0d",e.len());

        $display("RGB: #%0h%0h%0h", rgb.RED, rgb.GREEN, rgb.BLUE);

      $display("RGBA: #%0h%0h%0h%0s", rgba.RED, rgba.GREEN, rgba.BLUE, rgba.ALPHA); //unpacked struct

        $display("union: {i: %0d, s: %0d}", some.i,some.s);

        some.s = 255;

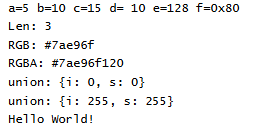
        $display("union: {i: %0d, s: %0d}", some.i,some.s);

        console.log("Hello World!");

    end

endmodule

OUTPUT



# Events

module events\_mgmt ();

    event ev1;

    initial begin

        fork

            begin

                 #60;

                 $display($time,"\t Triggring Event");

                 -> ev1;

            end

            begin

               $display($time,"\t Waitingg for event trigger");

               #20;

              @(ev1);

               $display($time,"\t Event Triggered");

            end

        join

    end

    initial begin

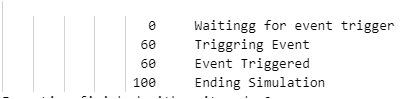
        #100;

      $display($time,"\t Ending Simulation");

    end

endmodule

OUTPUT



# Extern Keyword (Task Example)

//class with extern task

class packet;

  bit [31:0] addr;

  bit [31:0] data;

  extern virtual task display();

endclass

task packet::display();

  $display("ADDRESS: 0x%0h", addr);

  $display("DATA: 0x%0h", data);

endtask

module extern\_class\_tb;

  initial begin

    packet p;

    p = new();

    p.addr = 120;

    p.data = 200;

    p.display();

  end

endmodule

OUTPUT



# Extern Keyword (Function Example)

class packet;

  bit [31:0] addr;

  bit [31:0] data;

  extern virtual function void display();

endclass

function void packet::display();

    $display("ADDRESS: 0x%0h", addr);

    $display("DATA: 0x%0h", data);

endfunction

module extern\_method;

  initial begin

    packet p;

    p = new();

    p.addr = 110;

    p.data = 240;

    p.display();

  end

endmodule

OUTPUT



# Parameterized Class

class packet #(parameter int WIDTH,int DEPTH);

    function void print();

        $display("WIDTH: %0d", WIDTH);

        $display("DEPTH: %0d", DEPTH);

    endfunction

endclass

module tb;

    initial begin

        packet #(16, 8) pkt = new(); // Ensure this is at the correct scope

        pkt.print();

    end

endmodule

OUTPUT



# Pre-Post Randomization

class generator;

    rand bit [4:0] a, b;

    bit [3:0] y;

    // Constructor

    function new();

        y = 4'b1111;

    endfunction

    function void pre\_randomize();

        y = 4'b1111;

    endfunction

    function void post\_randomize();

        if (b >= 16) begin

            b = b % 16;

        end

    endfunction

endclass

module tb;

    generator g;

    int i = 0;

    initial begin

        g = new();

        for (i = 0; i < 5; i = i + 1) begin

            if (g.randomize()) begin

                $display("Value of a: %0d, b: %0d, and y: %0d", g.a, g.b, g.y);

            end else begin

                $display("Randomization failed at iteration %0d", i);

            end

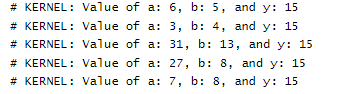
            #10;

        end

    end

endmodule

OUTPUT



# Randomization Constraints Enable/Disable

class myClass;

    rand int a;

    rand int b;

    constraint a\_con { a > 5; }

    constraint b\_con { b < 10; }

    function void dis\_con();

        a\_con.constraint\_mode(0);

    endfunction

    function void en\_con();

        a\_con.constraint\_mode(1);

    endfunction

endclass

module testbench;

  myClass obj;

    initial begin

        obj = new();

        obj.dis\_con();

        if (obj.randomize()) begin

            $display("Randomization successful.");

            if(!obj.a\_con.constraint\_mode())

                $display("Constraints Disabled!");

        end else begin

            $display("Randomization failed.");

        end

    end

endmodule

OUTPUT



# Constrained Randomization

class packet;

    rand bit [7:0] addr;

    rand bit [7:0] start\_addr;

    rand bit [7:0] end\_addr;

    constraint con2 { start\_addr inside {0:5, 8, 10:12}; }

    constraint con1 { !(end\_addr inside {[0:20]}); }

    constraint con3 { !(addr inside {[start\_addr:end\_addr]}); }

endclass

module constr\_inside;

    initial begin

        packet pkt;

        pkt = new();

        repeat(3) begin

            pkt.randomize();

            $display("\tstart\_addr = 0x%0h,end\_addr = 0x%0h",pkt.start\_addr,pkt.end\_addr);

            $display("\taddr = 0x%0h",pkt.addr);

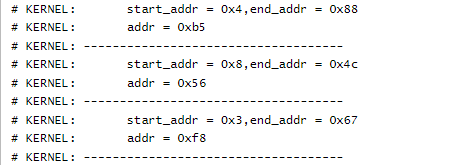
            $display("------------------------------------");

        end

    end

endmodule

OUTPUT



# Randomize Constraints Distribution

class myClass;

  rand bit[1:0] var1, var2;

  rand bit var3, var4, var5, var6, var7, var8;

    constraint data {

        var1 dist {0:=30, [1:3]:=90};

        var2 dist {0:/30, [1:3]:/90};

        (var3 == 0) -> (var4 == 0);

        (var5 == 0) <-> (var6 == 1);

    }

    constraint d {

        if (var7==1) {

            var8 == 0;

        } else {

            var8 == 1;

        }

    }

endclass //myClass

module tb ();

    myClass c;

    initial begin

        c = new();

            $display("====================================================");

        repeat(3) begin

            c.randomize();

            $display("var1: %0d var2: %0d //Same weight for specified val", c.var1, c.var2);

            $display("var3: %0d var4: %0d //Equally disribute weight", c.var3, c.var4);

            $display("var5: %0d var6: %0d //implicit operator", c.var5, c.var6);

            $display("var7: %0d var8: %0d //mutally Exclusive", c.var7, c.var8);

            $display("====================================================");

        end

    end

endmodule

