SYSTEM VERILOG PROGRAMS

MELVIN RIJOHN T

Contents

INDEX .... **Error! Bookmark not defined.**

Array Types 3

Process, Task & Functions 4

MUX 2x1 5

Data Types 6

Events 8

Extern Keyword (Task Example) 8

Extern Keyword (Function Example) 9

Parameterized Class 10

Pre-Post Randomization 10

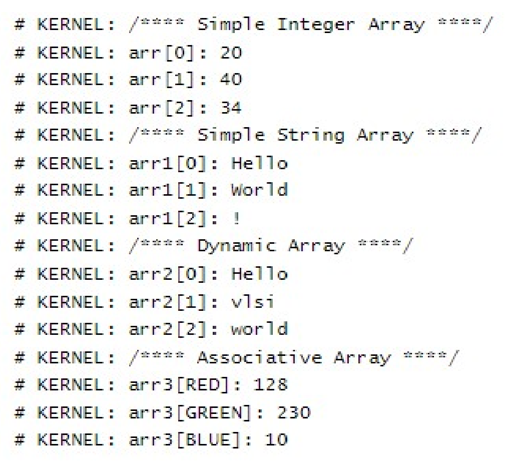
Randomization Constraints Enable/Disable 11

Constrained Randomization 12

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



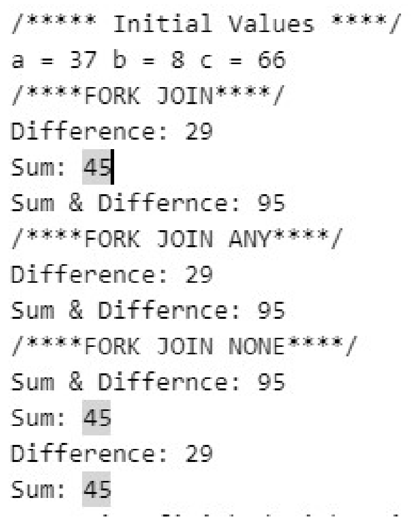
```
# KERNEL: /**** Simple Integer Array ****/
# KERNEL: arr[0]: 20
# KERNEL: arr[1]: 40
# KERNEL: arr[2]: 34
# KERNEL: /**** Simple String Array ****/
# KERNEL: arr1[0]: Hello
# KERNEL: arr1[1]: World
# KERNEL: arr1[2]: !
# KERNEL: /**** Dynamic Array ****/
# KERNEL: arr2[0]: Hello
# KERNEL: arr2[1]: vlsi
# KERNEL: arr2[2]: world
# KERNEL: /**** Associative Array ****/
# KERNEL: arr3[RED]: 128
# KERNEL: arr3[GREEN]: 230
# KERNEL: arr3[BLUE]: 10
```

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



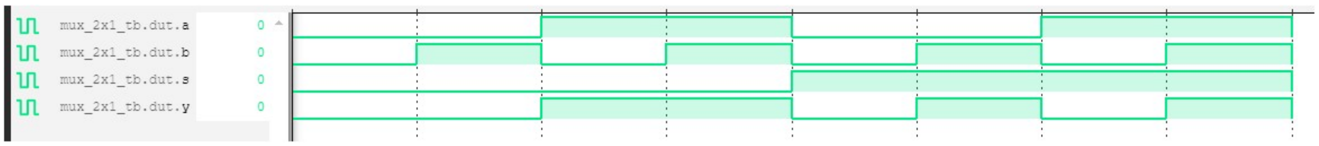
```
/***** Initial Values ****/
a = 37 b = 8 c = 66
/****FORK JOIN****/
Difference: 29
Sum: 45
Sum & Differnce: 95
/****FORK JOIN ANY****/
Difference: 29
Sum & Differnce: 95
/****FORK JOIN NONE****/
Sum & Differnce: 95
Sum: 45
Difference: 29
Sum: 45
```

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

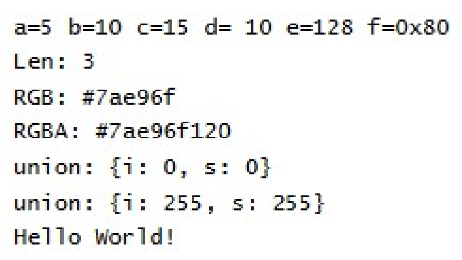
```
a=0 b=0 s=0 y=0
a=0 b=1 s=0 y=0
a=1 b=0 s=0 y=1
a=1 b=1 s=0 y=1
a=0 b=0 s=1 y=0
a=0 b=1 s=1 y=1
a=1 b=0 s=1 y=0
a=1 b=1 s=1 y=1
```



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

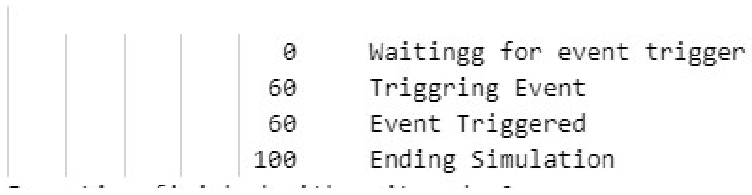


```
a=5 b=10 c=15 d= 10 e=128 f=0x80
Len: 3
RGB: #7ae96f
RGBA: #7ae96f120
union: {i: 0, s: 0}
union: {i: 255, s: 255}
Hello World!
```

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

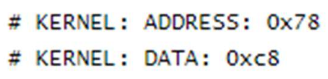


```
  0     Waitingg for event trigger
 60     Triggring Event
 60     Event Triggered
100     Ending Simulation
```

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



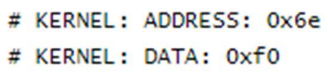
```
# KERNEL: ADDRESS: 0x78
# KERNEL: DATA: 0xc8
```

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

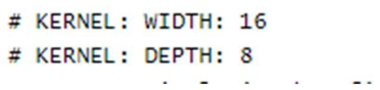


```
# KERNEL: ADDRESS: 0x6e
# KERNEL: DATA: 0xf0
```

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

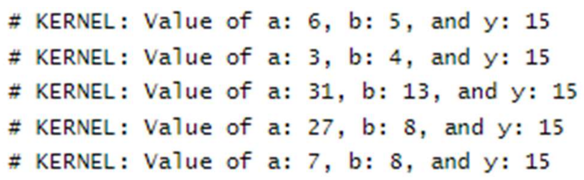


```
# KERNEL: WIDTH: 16
# KERNEL: DEPTH: 8
```

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

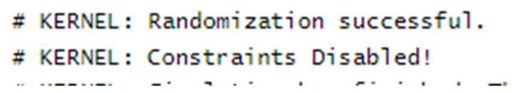


```
# KERNEL: Value of a: 6, b: 5, and y: 15
# KERNEL: Value of a: 3, b: 4, and y: 15
# KERNEL: Value of a: 31, b: 13, and y: 15
# KERNEL: Value of a: 27, b: 8, and y: 15
# KERNEL: Value of a: 7, b: 8, and y: 15
```

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

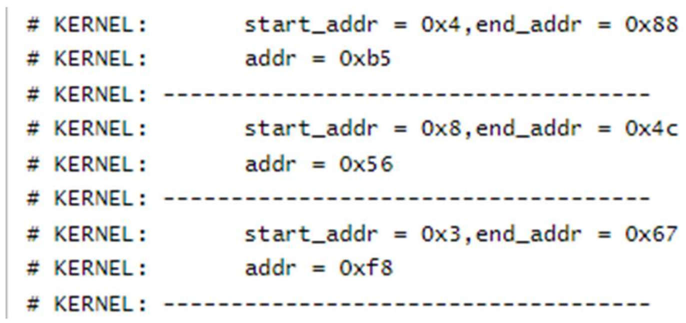


```
# KERNEL: Randomization successful.
# KERNEL: Constraints Disabled!
```

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



```
# KERNEL:       start_addr = 0x4,end_addr = 0x88
# KERNEL:       addr = 0xb5
# KERNEL: ------------------------------------
# KERNEL:       start_addr = 0x8,end_addr = 0x4c
# KERNEL:       addr = 0x56
# KERNEL: ------------------------------------
# KERNEL:       start_addr = 0x3,end_addr = 0x67
# KERNEL:       addr = 0xf8
# KERNEL: ------------------------------------
```