SYSTEM VERILOG PROGRAMS

MELVIN RIJOHN T

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
```

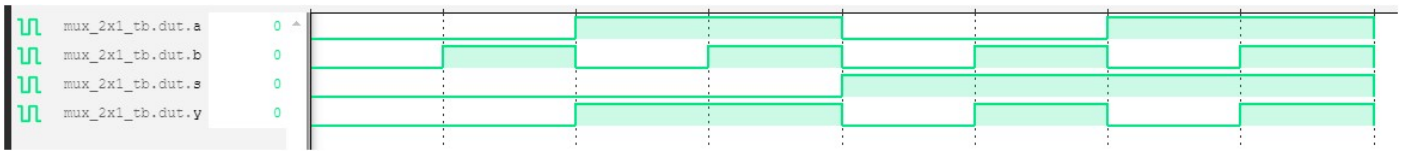
MUX 2x1

```
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!
```

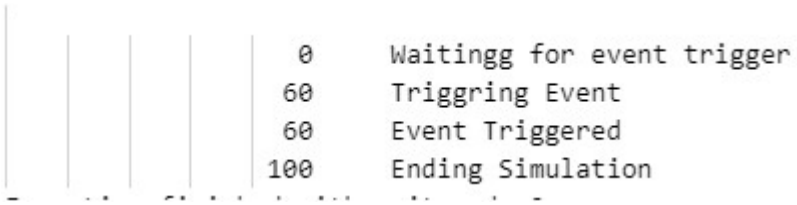
Events

```
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



Deep Copy

```
class first;
    int data = 10;
    function first copy();
        copy = new();
        copy.data = data;
    endfunction
endclass

class second;
    int ds = 56;
    first f1;

    function new();
        f1 = new();
    endfunction

    function second copy();
        copy = new();
        copy.ds = ds;
        copy.f1 = f1.copy();
    endfunction
endclass

module tb();
    second s1, s2;
    initial begin
```

```
        s1 = new();
        s2 = new();
        s1.ds = 34;
        s2 = s1.copy();
      $display("S2_DS: %0d", s2.ds);
        s2.ds = 26;
      $display("S1_DS: %0d", s1.ds);
        s2.f1.data = 68;
      $display("S1_F1_DATA: %0d", s2.f1.data);
    end
endmodule
```

OUTPUT

```
# KERNEL: S2_DS: 34
# KERNEL: S1_DS: 34
# KERNEL: S1_F1_DATA: 68
```

Shallow Copy

```
class first;
   int data = 12;
endclass

class second;
    first f1;
    int ds = 1;
    function new();
        f1 = new();
    endfunction
endclass


module shallow_copy_tb();
    second s1,s2;
    initial begin
        s1 = new();
        s1.ds = 25;
        s2 = new s1;
        $display("S1_DS: %0d", s1.ds);
        s2.ds = 46;
        $display("S1_DS: %0d", s1.ds);
        s2.f1.data = 20;
        $display("S1_DS: %0d, S1_F1_DATA: %0d", s1.ds, s1.f1.data);
    end
endmodule
```

OUTPUT

```
# KERNEL: S1_DS: 25
# KERNEL: S1_DS: 25
# KERNEL: S1_DS: 25, S1_F1_DATA: 20
```

Class Inheritance

```
class Shape;
    string name;

    function new(string name);
        this.name = name;
    endfunction

    function void print();
        $display("Shape: %s", name);
    endfunction
endclass

class Circle extends Shape;
    real radius;

    function new(real radius);
        super.new("Circle");
        this.radius = radius;
    endfunction

    function real calc_area();
        return 3.1416 * radius * radius;
    endfunction
endclass

class Rectangle extends Shape;
    real length, width;

    function new(real length, real width);
        super.new("Rectangle");
        this.length = length;
        this.width = width;
    endfunction

    function real calc_area();
        return length * width;
    endfunction
endclass

module test;
    Circle c;
    Rectangle r;
    initial begin
```

```
        c = new(5.89);
        c.print();
        $display("Area of %s: %0.2f", c.name, c.calc_area());

        r = new(4.25, 7.16);
        r.print();
        $display("Area of %s: %0.2f", r.name, r.calc_area());
    end
endmodule
```

OUTPUT

```
# KERNEL: Shape: Circle
# KERNEL: Area of Circle: 108.99
# KERNEL: Shape: Rectangle
# KERNEL: Area of Rectangle: 30.43
```

Class Polymorphism

```
class first;
    int data = 12;

    virtual function void print();
        $display("FIRST_VAL: %0d", data);
    endfunction
endclass

class second extends first;
    int temp = 34;

    function void add();
      $display("SECOND_VAL_ADD: %0d", super.data + 4);
    endfunction

    function void print();
        $display("SECOND_VAL: %0d", temp);
    endfunction
endclass

module tb ();
    first f;
    second s;
    initial begin
        f = new();
        s = new();
        f = s;
        f.print();
        s.add();
    end
endmodule
```

OUTPUT

```
# KERNEL: SECOND_VAL: 34
# KERNEL: SECOND_VAL_ADD: 16
```