

## Problem

Hierarchical reference to named events.

## Workaround

Avoid accessing named events hierarchically. Instead capture the named event's activity in a reg. Try accessing this reg hierarchically.

---

## Problem

Alias port using concatenation operation.

```
module func(.aOut({a0[0], outreg[1]}));  
output reg [20] a0;  
output reg [20] outreg = 3'b010;  
...
```

## Workaround

Capture the concatenation into another *reg* and then use that *reg* as port. The following code snippet shows how to do this.

```
module func(aOut);  
reg [20] a0;  
reg [20] outreg = 3'b010;  
output reg [10] aOut;  
always @(a0[0], outreg[1])  
    aOut = {a0[0], outreg[1]};  
...
```

---

## Problem

Module arrays are unsupported

## Workaround

Expand the array instantiation using generate statement.

---

## Problem

Hierarchical expression on the left hand side of assignment.

## Workaround

Export the *reg/wire* which needs to be assigned hierarchically as a port and then assign to the port

---

## Problem

Net declaration assignment where the initial value is an expression with operators.

```
wire [AddrBits-10] AddrSize = -1;
```

## Workaround

Capture the expression in a parameter and then initialize the wire with the parameter.

```
parameter pp = -1;  
wire [AddrBits-10] AddrSize = pp;
```

---

## Problem

Parameter initialization with conditional operator.

```
localparam lp = (p1 > p2) ? 10 20;
```

## Workaround

Capture the condition expression in another parameter and use the new parameter as the condition in the operator.

```
localparam lpp = p1 > p2;  
localparam lp = lpp ? 10 20;
```

---

## Problem

An initial/always block with fork/join inside generate statement.

```
generate
if (width == 1)
beginabcd
initial
beginabc
fork mod_1
reg x;
#1 mod_2.x = 1'b1;
join
fork mod_2
reg x;
mod_1.x = 1;
join
end
end
endgenerate
```

## Workaround

Move the initial/always block out of the generate statement and restructure the initial body to have an equivalent effect.

```
initial
if (width == 1)
beginabc
```

```
fork mod_1
reg x;
#1 mod_2.x = 1'b1;
join
fork mod_2
reg x;
mod_1.x = 1;
join
end
```

---

### **Problem**

System calls as parameters of other system calls.

```
$display($itor(a +2.0));
```

### **Workaround**

Capture the value of nested system call in a variable and then pass the value to the outer system call.

---

### **Problem**

Using *genvar* as actual in port connection.

---

### **Problem**

Using hierarchical expression as range of a part select.

```
Mreg[1:4] = Preg[a.b.c +: 4];
```

### **Workaround**

Capture the hierarchical expression in a *reg* and use the *reg* in the part select.

---

## **Problem**

CR 417164

Using concatenation expression in a conditional operator

```
force w = r0 ? { r1, r1 } : r1;
```

## **Workaround**

Capture the result of concatenation operation in another variable and pass the new variable as operand to the conditional operator.

---

## **Problem**

Passing complex expression as operands to system calls.

## **Workaround**

Capture the value of complex expression in a variable and pass the new variable as argument to system call.