

Problem :

Any procedure which has nested procedures or functions and there is a wait statement in any of the procedures will result in compilation errors.

Workaround:

If a procedure has a wait statement, do not use nested procedures or functions, but instead declare them outside the procedure.

Problem:

If a procedure is only called from another procedure and never called from within a process or concurrent statement, a compilation error will result.

Workaround:

Call all procedures from a process wrapped from the IF statement that is always false.

```
process is
begin
    if false then
        procedure1;
        procedure2;
    end if;
    wait;
end process;
```

Problem:

Default value of a generic is a function call and the argument of this function call is another function call.

Example:

```
generic (
    g1 : integer := func1(func2(16))
);
```

Workaround:

Write a function which is used as the generics default value which calls the original two functions.

```
function func3 (
    constant arg1 : integer)
return integer is
begin
```

```
        return func1(func2(arg1));
    end function func3;

    generic (
        g1 : integer := func3(16)
    );
```

Problem:

Using complex static expressions as a case choice can cause compilation errors.

Example:

```
    case i is
        when SUB_TYPE('1','0') => v := false;
        when others => v := true;
    end case;
```

Workaround:

First assign static expression to a constant and use that as the case choice.

```
    constant const1 : SUB_TYPE := SUB_TYPE('1','0');

    case i is
        when const1 => v := false;
        when others => v := true;
    end case;
```

Problem :

Using type conversion, qualified expressions, function calls or an aggregate in a concatenation expression can cause compilation errors.

Example:

```
    var1 := s2 & not unsigned(s1);
```

Workaround:

Use a temporary variable to store the results of the expression, and then use the temp variable in the concatenation expression.

```
    var2 := not unsigned(s1);
    var1 := s2 & var2;
```

Problem:

Using complex static expressions as argument to function can cause compilation errors.

Example:

```
s0 <= f1(const1+100/const3 + 63 rem 8 , const3 ** 4 - const1 +
abs(const1));
```

Workaround:

Assign expression to a temporary constant and use temporary constant as argument to function

```
constant const4 : integer := const1+100/const3;
constant const5 : integer := const3 ** 4 - const1 + abs(const1);

s0 <= f1(const4, const5);
```

Problem :

Block port map with Array Param can cause compilation errors.

Example:

```
B : block
  port (p0 : T := (0, 1, 2));
  port map (p0 => (2, 4, 6));
begin
end block;
```

Workaround:

Assign array to a temporary constant, and use the temporary constant as the actual.

```
constant const0 : T := (2, 4, 6);

B : block
  port (p0 : T := (0, 1, 2));
  port map (p0 => const0);
begin
end block;
```

Problem:

When the variable name of variable declared inside a sub program is an extended identifier, a compilation error will result.

Example:

```
procedure procedure1 is
    variable \name _\ : integer range 0 to datawidth := 0;
begin
end procedure procedure1;
```

Workaround:

Don't use extended identifiers in variable names.

Problem :

Changing the library name of the work library will cause fuse errors if there are any direct entity instantiations.

Example:

```
UUT: entity work.e(a) port map(p0 => s0);
and fuse is run as:
fuse -top top -work mylib
```

Workaround:

Don't change the work library, or don't use direct entity instantiation.