SATRCRules

p340-p507

第九組
吳侑霖 盧睿彬

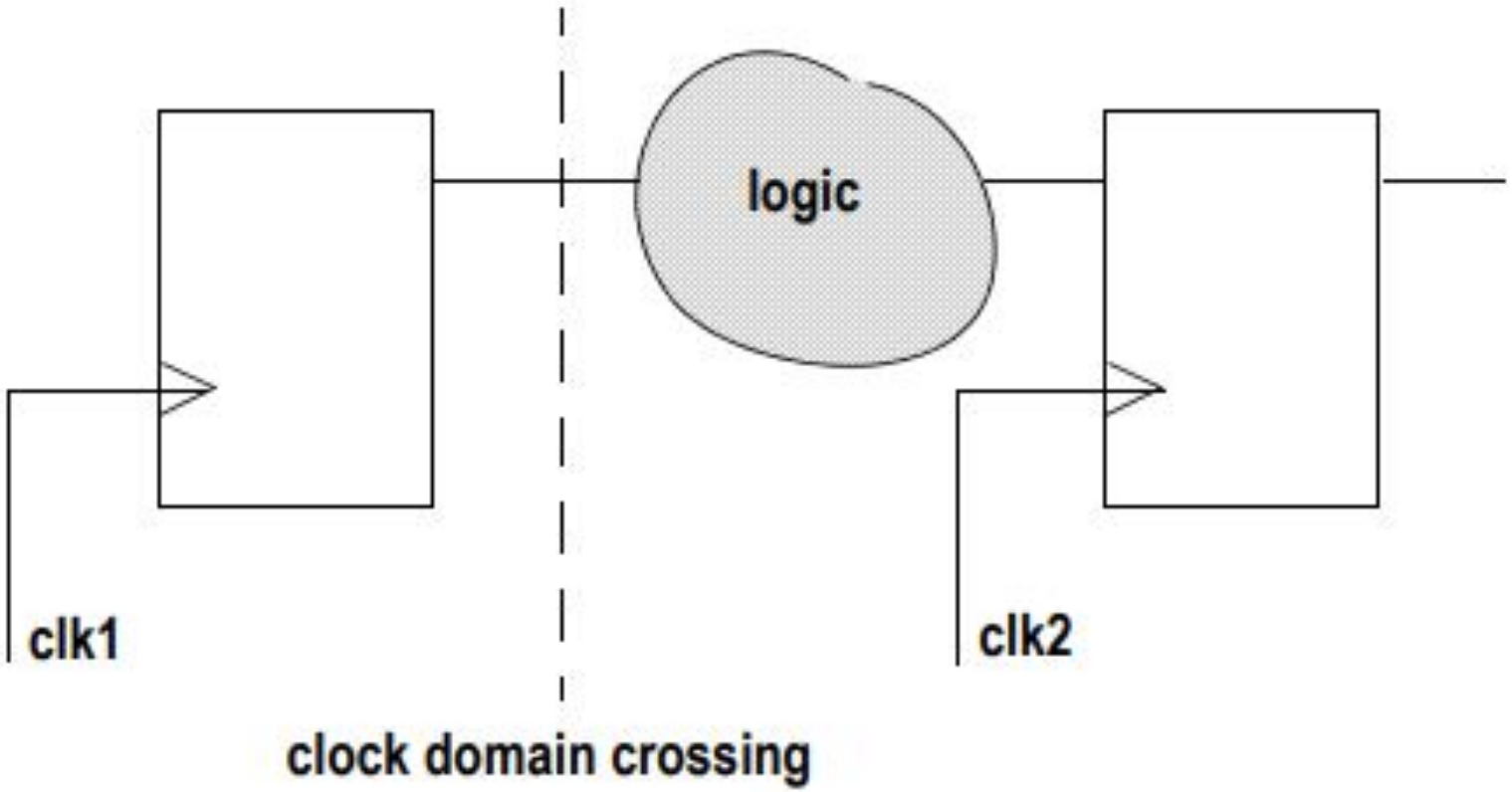
―PART 1―

STARC-1.5.1.1

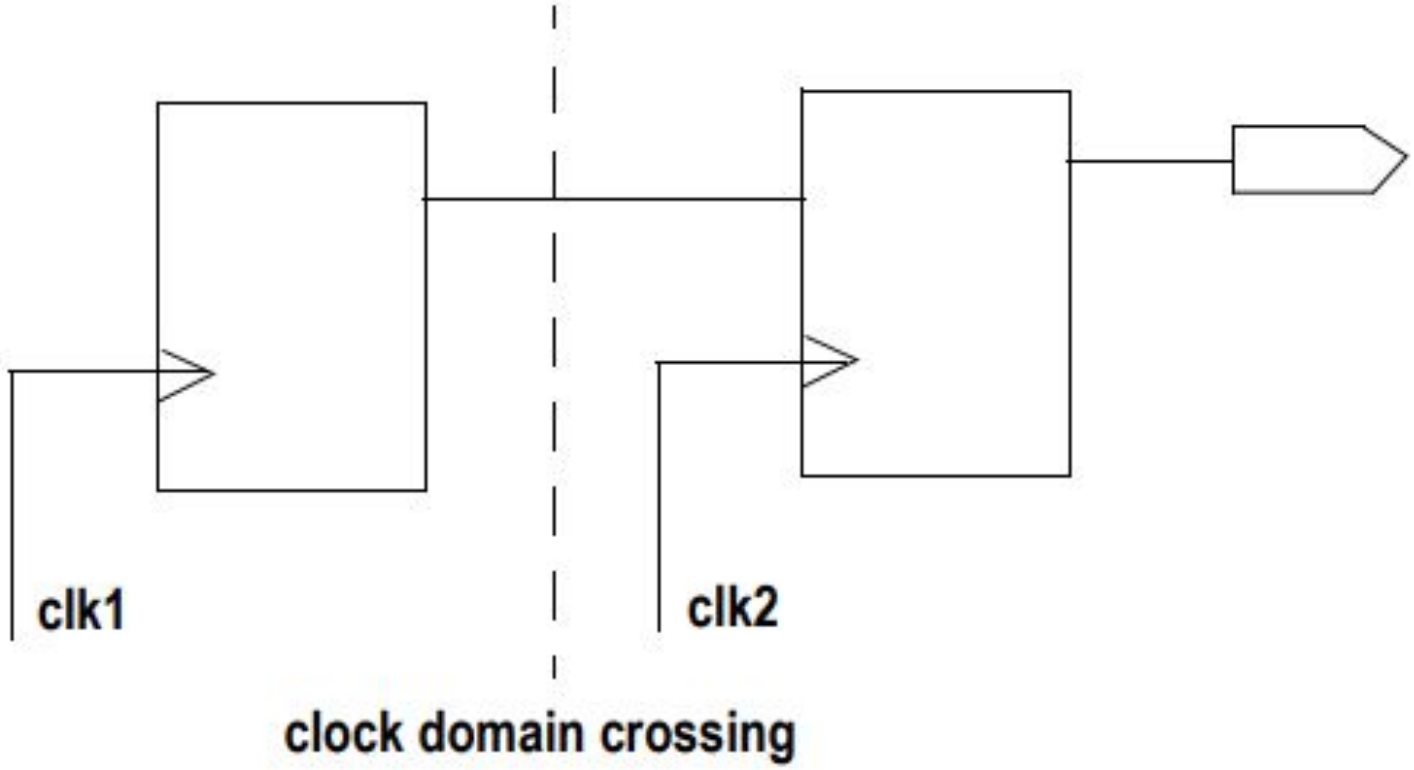
Two-flop synchronization structure must be used for data transfer between two clock domains

- Flaging improperly synchronized data transfer between two clock domains
- Logic exists between a flipflop of the source clock domain and first flipflop of the destination clock domain
- There is no flipflop of the destination clock domain following the first flip-flop of the destination clock domain
- There is a feedback path from the output to the input of the second flip-flop in the destination clock domain

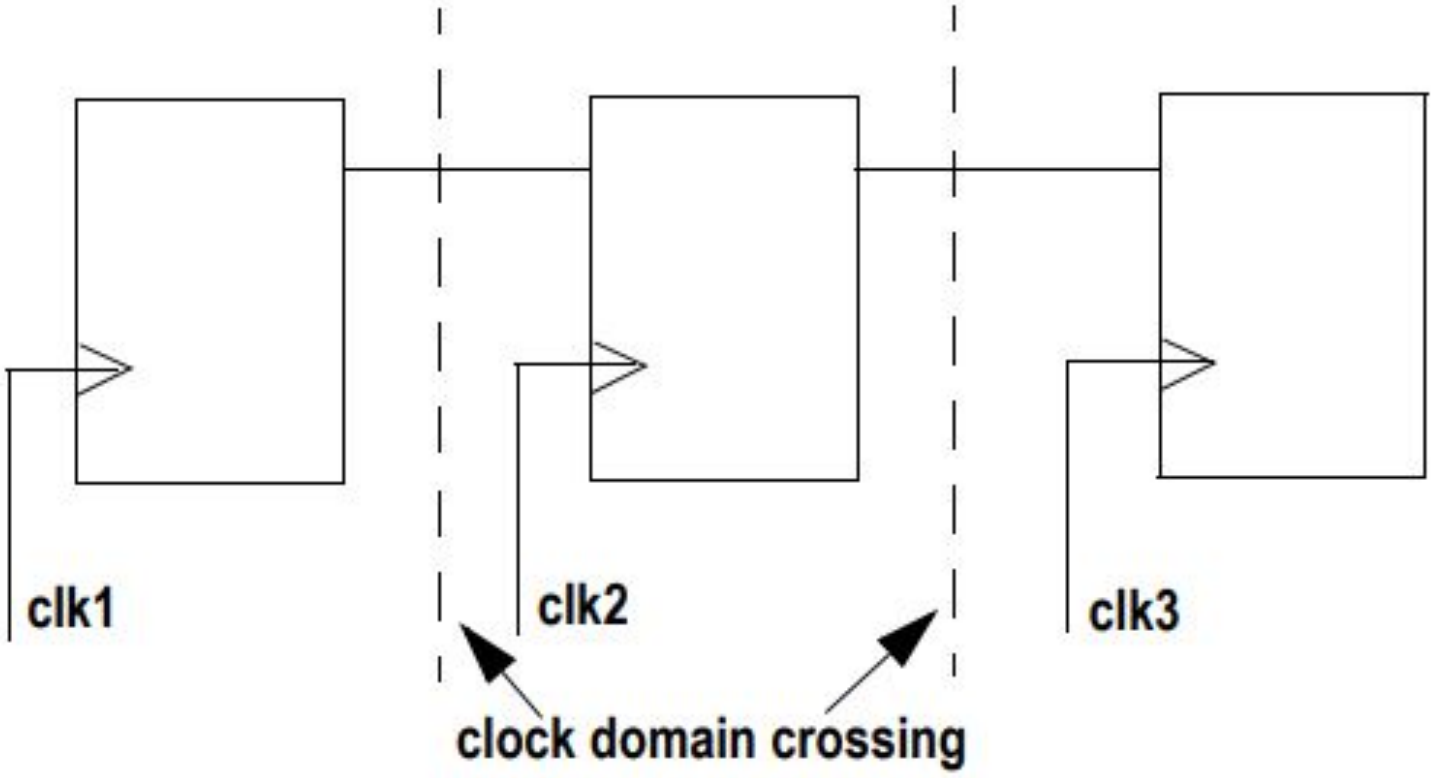
STARC-1.5.1.1



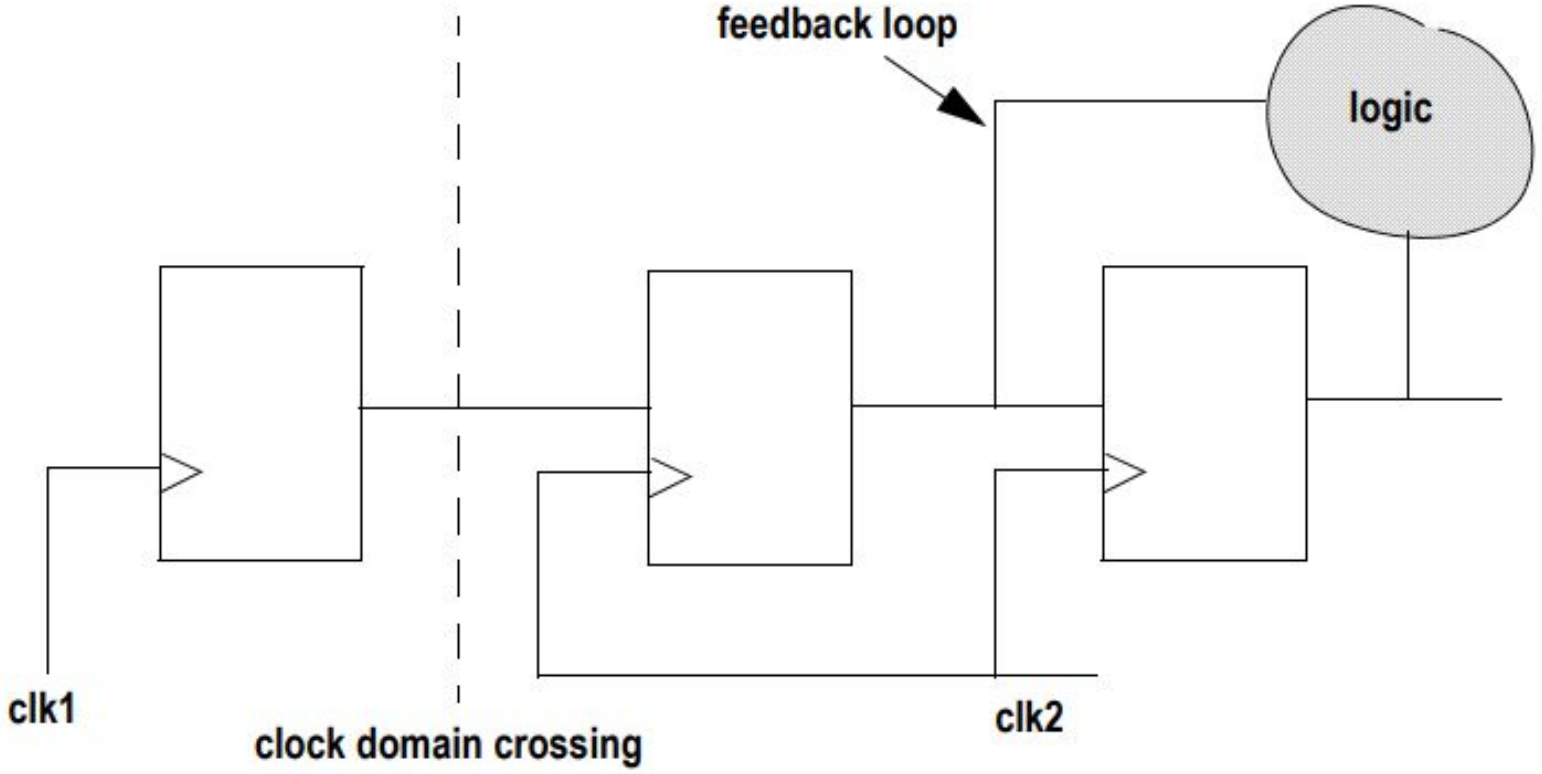
STARC-1.5.1.1



STARC-1.5.1.1



STARC-1.5.1.1



STARC-1.5.1.2

Two-flop synchronization structure must be used for data transfer between two clock domains

STARC-1.5.1.3

Feedback loop must not exist that contains flip-flop one level from a clock domain crossing

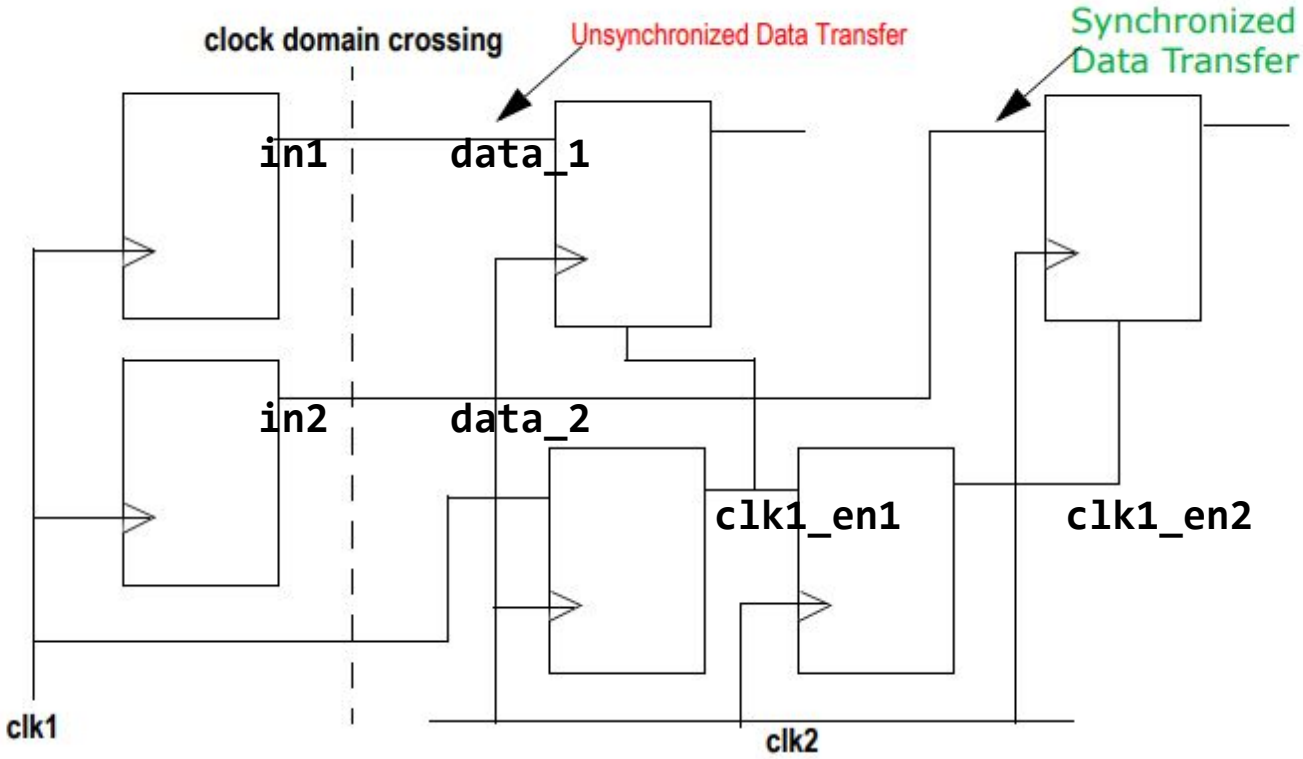
EXAMPLE

```
module test (in1, in2, clk1, clk2, out);
    input in1, in2, clk1, clk2;
    output out;

    wire sig1;
    reg q1;

    always @(posedge clk1)
        q1 <= in1;

    assign sig1 = q1 && in2;

    always @(posedge clk2)
        out <= sig1;
endmodule
```

STARC-1.5.1.5

For data transfers from one domain to another, latch the source clock signal and use it as enable signal for destination domain

STARC-1.5.1.5


clock domain crossing
Unsynchronized Data Transfer
Synchronized
Data Transfer
in1
data_1
in2
data_2
clk1_en1
clk1_en2
clk1
clk2

EXAMPLE

```
module test (in1, in2, clk1,
    clk2, out1, out2);
    input in1, in2, clk1, clk2;
    output out1, out2;

    reg data_1, data_2, clk1_en1, clk1_en2;

    always @(posedge clk1)
        begin
            data_1 <= in1;
            data_2 <= in2;
        end

    always @(posedge clk2)
        clk1_en1 <= clk1;

    always @(posedge clk2)
        clk1_en2 <= clk1_en1;

    always @(posedge clk2)
        begin
            if ( clk1_en1 )
                out1 <= data_1;
            if ( clk1_en2 )
                out2 <= data_2;
        end
endmodule
```

STARC-1.6.1.1

A basic block should not contain more than 10,000 gates and 200 I/Os

STARC-1.6.1.2

Modules instantiating basic blocks must not contain logic gates

STARC-1.6.1.4

Clock generation, reset generation, RAM, Setup/Hold ensure buffers, and I/O cells must be a module at top-level

STARC-1.6.2.1

Top-level module outputs should be directly driven by flip-flops

EXAMPLE

```
module test (OUT, IN);
    input IN;
    output OUT;

    inst1 inst (OUT, IN);
endmodule

module inst (OUT, IN);
    input IN;
    output OUT;

    assign OUT = IN;
endmodule
```

```
module test (OUT, IN, CLK);
    input IN, CLK;
    output OUT;

    inst1 inst (OUT, IN, CLK);
endmodule

module inst (OUT, IN, CLK);
    input IN, CLK;
    output OUT;

    reg OUT;

    always @(posedge CLK)
        OUT = IN;
endmodule
```

STARC-1.6.2.2

Combinatorial Path should not span more than specified number of modules

- Intermediate modules are not considered
- Renamed by a continuous assignment are also not considered.
- Modules where the signal is inverted are considered.
- Originating module is also taken into consideration, even if the flip-flop output goes directly to the module output without any intervening logic on the way. The same is true of the destination module.

STARC-1.6.2.2a

Timing path should not span more than two basic blocks

STARC-1.6.3.1

Keep timing paths within two sub-blocks inside each basic block

STARC-1.6.3.2

Keep timing paths within three sub-blocks inside each basic block

STARC-1.6.6.2

Do not use set_dont_touch directive directly on cells

STARC-1.6.6.3

Do not instantiate library cells in the design. (Verilog)
Do not directly instantiate cells in the design (VHDL)

EXAMPLE

```
FF1 myFF1(.Din(in), .Clk(clk), .Q(out1), .QB(out2));

LFD2Q dff (.CP(clk),.RN(rst),.D(q),.Q(r1));
```

STARC-2.1.1.1

Combinational logic should consistently be described using either always constructs or function and assign constructs. (Verilog)

Package IEEE.std_logic_1164 must be included in each entity (VHDL)

STARC-2.1.1.2

A function description must assign return values to all possible states of the function. (Verilog)

At least one of the packages IEEE.std_logic_arith and IEEE.std_logic_unsigned must be included in each entity (VHDL)

EXAMPLE

```
function [4:0] Adder;
    input [3:0] first;
    input [3:0] second;
    input select;

    begin
        while(select == 1'b0)
            Adder = first + second;
    end
endfunction
```

STARC-2.1.1.4

reg declarations in always blocks should be commented. (Verilog)

STARC-2.1.2.1

Use std_logic or std_logic_vector data types to describe ports of an entity (VHDL)

STARC-2.1.2.2

always blocks should not have function calls. (Verilog)

Use std_logic, std_logic_vector, integer, Boolean, unsigned, signed, time or user-defined data type inside the architecture (VHDL)

STARC-2.1.2.3

Do not use nonblocking assignment in function description. (Verilog)

Use range specification for integer types (VHDL)

STARC-2.1.2.4

Global signals must not be read in a function description. (Verilog)

Do not use bit and bit_vector data types in the design (VHDL)

STARC-2.1.2.5

task constructs should not be used in the design. (Verilog)

Do not use attribute enum_encoding (VHDL)

STARC-2.1.2.6

Clock edges should not be used in a task description. (Verilog)

STARC-2.1.3.1

Bit-width of function arguments must match bit-width of the corresponding function inputs. (Verilog)

Linkage or buffer ports must not be described (VHDL)

STARC-2.1.3.2

Bit-width of function return value must match the bit-width of assignment destination signal. (Verilog)

Do not use buffer and linkage (VHDL)

STARC-2.1.3.3

Port mode must be explicitly specified (VHDL)

STARC-2.1.3.4

Return value assignment statement must be the last statement in function description. (Verilog)

Input port must not be described with initial value (VHDL)

STARC-2.1.3.5

Global signals must not be assigned in function description. (Verilog)

STARC-2.1.4.1

Parenthesis should be used in expressions with multiple operators. (Verilog)

STARC-2.1.4.2

Shift operator should not be used in the design (VHDL)

STARC-2.1.4.3

Bit-wise operators should be used instead of logic operators in single-bit operations. (Verilog)

Do not use shift operator (VHDL)

STARC-2.1.4.6

Use bit-wise operators instead of logic operators in multi-bit operations.

―PART 2―

STARC-2.1.4.7

Reduction of single-bit expressions or large expressions should not be performed. (Verilog)

STARC-2.1.4.7a

Reduction of a single-bit expression should not be performed (Verilog) (ex: & , |))

STARC-2.1.4.7b

Reduction of Large expression should not be performed (Verilog)

STARC-2.1.5.1

Conditional operators (?:) should not be nested (Verilog)
When-else nesting in conditional signal assignment statement must not exceed 4 levels (VHDL)

EXAMPLE

```
assign out = en1?en2?1'b1:in:1'b0;
assign out = en1?1'b0:en2?1'b1:in;
out = en1?in1:(en2?1'b1:(en3?in2:1'b0));
```

STARC-2.1.5.3

Ensure that conditional expressions evaluate to a scalar.

EXAMPLE

```
module test(out1, out2, in1, in2, in3, in4, en1, en2);
input in1, in2, in3, in4;
output out1, out2;
input [1:0] en1, en2;
reg out1, out2;
always @ (en1 or in1 or in2)
if (en1)
out1 <= in1;
else
out1 <= in2;
always @ (en2 or in3 or in4)
out2 <= (en2) ? in3 : in4;
endmodule
```

STARC-2.1.6.1

Array specification should follow recommended convention.
(Verilog)
Specification of one-dimensional array should be downto (VHDL)
[MSB:LSB] format where LSB is less than MSB.

STARC-2.1.6.2

LSB of vectors/memory should be zero. (Verilog)
LSB of one-dimensional arrays should be zero. (VHDL)

EXAMPLE

```
reg [7:0] memory [0:15] //memory size [0:15] should
//be [15:0]
reg [7:1] memory [15:0] //index specification [7:1]
//should be [7:0]
```

STARC-2.1.6.3

Expressions should not be used as array indexes. (Verilog)
Expression should not be used as array indices. (VHDL)
(simple signal name, NOT expression)

STARC-2.1.6.4

Index variable width should not be short (Verilog)

EXAMPLE

```
module test(out, in, clk);
input [3:0] in;
input clk;
output [3:0] out;
reg [3:0] out;
wire i;
always @ (posedge clk)
out[i] = in[i];
endmodule
```

STARC-2.1.7.1

Direct assignment must be used for aggregates (VHDL)

EXAMPLE

```
SIG <=(IN1,others =>IN2);{Good}
SIG <=IN1 &(1 =>IN2, 2 => IN3);{Bad}
```

STARC-2.1.7.3

Bit position based and others based assignment should not be used together for aggregate specification (VHDL)

EXAMPLE

```
SIG <= (2 => IN1, others => '0');{Bad}
SIG <= "00" & IN1;{Good}
```

STARC-2.1.8.1

Constrained arrays should not be used as sub-program arguments (VHDL)

Definition: A constrained array's index range is explicitly define

STARC-2.1.8.2

Specify range for return values in function description when return type is array (VHDL)

STARC-2.1.8.5a

Do not use nested sub-program description in the design (VHDL)

STARC-2.1.8.5b

Do not call a sub-program within a subprogram (VHDL)

STARC-2.1.8.6

Global signals must not be read in a subprogram description (VHDL)

EXAMPLE (1)

```
library ieee;
use ieee.std_logic_1164.all;
package MyPkg is
shared variable GLOBAL_VAR : integer:=1;
impure function myFunc( input : std_logic ) return
std_logic;
end MyPkg;
```

EXAMPLE (2)

```
package body MyPkg is
impure function myFunc( input : std_logic ) return
std_logic is
begin
assert GLOBAL_VAR = 1;
report "This is STARC VHDL";
return not input;
end myFunc;
end package body MyPkg;
```

EXAMPLE (3)

```
library ieee;
use ieee.std_logic_1164.all;
use work.MyPkg.all;
entity ent2161_Test2 is
port ( in_ent2161_Test2 : in std_logic;
out_ent2161_Test2 : out std_logic );
end ent2161_Test2;
architecture structure_ent2161_Test2 of
ent2161_Test2 is
begin
out_ent2161_Test2 <= myFunc(in_ent2161_Test2);
end structure_ent2161_Test2;
```

STARC-2.1.8.7

No other statement should follow the return statement in function description (VHDL)

STARC-2.1.8.8 (covered by STARC-2.1.10.5)

Do not use procedure construct with RTL description. (VHDL)

STARC-2.1.8.9

A function must have the return statement and return a valid value in all possible states of the function. (VHDL)

STARC-2.1.9.3 (covered by STARC-2.3.1.2b)

Avoid using stable attribute for FF inference. (VHDL)

STARC-2.1.9.4

Do not use built-in attribute except range, length, left, right, high, low, reverse_range, event (VHDL)

STARC-2.1.9.5

Do not use user-defined attributes in the design (VHDL)

STARC-2.1.10.1

Do not use block statement in the design (VHDL)

```
A1: OUT1 <= '1' after 5 ns;
LEVEL1 : block
begin
  A2: OUT2 <= '1' after 5 ns;
  A3: OUT3 <= '0' after 4 ns;
end block LEVEL1;
A1: OUT1 <= '1' after 5 ns;
A2: OUT2 <= '1' after 5 ns;
A3: OUT3 <= '0' after 4 ns;
```

STARC-2.1.10.2

Do not use object of type record (VHDL)
(like structure in C)

STARC-2.1.10.3

Do not use shared variable in the design (VHDL)

Definition: a feature of the programming language APL which allows APL programs running on one processor to share information with another processor.

STARC-2.1.10.4

Do not use while-loop statement in the design (VHDL)

STARC-2.1.10.5

Do not use procedure in the design (VHDL)

```
with a select b <=
    "1000" when "00",
    "0100" when "01",
    "0010" when "10",
    "0001" when "11";
```

STARC-2.1.10.6

Do not use selected signal assignment in the design (VHDL)

STARC-2.1.10.7 (covered by STARC-1.1.6.5)

Configuration construct must not be used in the design. (VHDL)

STARC-2.1.10.8

Do not use attributes, which are defined in Synopsys library (VHDL)

Rules for always Constructs of Combinational Logic

STARC-2.2.1.3

Design should not have latches

EXAMPLE

```
module test (OUT, IN, EN);
output [1:0] OUT;
input [1:0] IN, EN;
reg [1:0] OUT;
always @ (EN or IN)
case (EN)
2'b00 : OUT[0] <= 1'b1;
2'b01 : OUT[0] <= 1'b0;
2'b10 : OUT[0] <= IN[1];
endcase
endmodule
```

No EN = 2’b11 !
No default branch !

STARC-2.2.2.1

Signals read inside a combinational always block must be present in the sensitivity list (Verilog)
Signals read inside a combinational process block must be present in the sensitivity list (VHDL)

STARC-2.2.2.2

A variable specified in the sensitivity list must be read in the contained block (Verilog)
A signal should not be included in sensitivity list of a process if it is not read in that process (VHDL)

STARC-2.2.2.4 (covered by STARC-2.2.2.1)

Sensitivity list of a combinational always block must not be incomplete.

STARC-2.2.3.1

Non-blocking assignment should not be used in combinational always blocks (may lead to shoot through). (Verilog)

*順序不影響

STARC-2.2.3.2

Do not use nonblocking assignments for assigning initial values in always blocks. (Verilog)

STARC-2.2.3.3

Do not assign over the same signal in an always construct for sequential circuits

EXAMPLE 1

```
module test (out, in1, in2, clk, reset);
output out;
input in1, in2, clk, reset;
reg out, out1;
always @ (posedge clk)
begin
out <= 1'b0;
out <= in1;
end
always @ (posedge clk or negedge reset)
begin
out1 <= 1'b1;
if (!reset)
out1 <= 1'b0;
else
out1 <= in2;
end
endmodule
```

EXAMPLE 2

```
always @ (posedge clk or negedge reset)
begin
out1 <= 1'b1; // Direct assignment
if (!reset)
out1 <= in1; //Violation only when the strict parameter is set to yes
```

EXAMPLE 3

```
always @ (posedge clk or negedge reset)
begin
if (!reset)
begin
out1 <= 1'b0; // Initialization inside if block
if (en)
out1 <= in; //Violation only when the strict parameter is set to yes
end
end
```

EXAMPLE 4

```
always @ (posedge clk)
begin
out <= 1'b0;
// First assignment
out <= in1;
// Second assignment in same

// block

end
always @(posedge clk) begin
if (rst) begin
q<=1'b0; // First assignment
q<=1'b1; // Second assignment inside same block(if)
end
else begin
q<=data;
//First assignment in the else block
end
end
```

Rules for Flip-Flop Inference

STARC-2.3.1.1

Blocking assignment must not be used in sequential always blocks. (Verilog)
Variable assignment statement must not be used in flip-flop description (VHDL) *順序會影響

STARC-2.3.1.2

Do not use unsynthesizable flip-flop description styles

STARC-2.3.1.2a

Do not use quasi-continuous assignments. (Verilog)
Do not use rising_edge or falling_edge of clock for flip-flop description (VHDL)

EXAMPLE

```
if (clk’event and clk = ’1’) --{Good}
if (rising_edge(clk)) then --{Bad}
```

STARC-2.3.1.2b

Do not use deassign statements. (Verilog)
Do not use stable attribute for flip-flop description (VHDL)

EXAMPLE

```
if (clk’event and clk = ’0’) then ... {Good}
if (not clk’stable and clk = ’1’) then ...{Bad}
```

STARC-2.3.1.2c

Do not use unsynthesizable User-Defined Primitives (UDPs). (Verilog)
Do not use non-synthesizable clocking styles for flip-flop description (VHDL)

EXAMPLE (non-synthesizable)

```
if (clk'event) then
if (not clk'stable) then
if (clk'stable) then
if (clk'stable and clk = '1') then
if (clk'event and clk'stable) then
-- If edge is not specified
if (not clk'event and clk = '0') then
if (clk'event and clk = "00") then
-- Multi-bit clock-signal
elsif (clk'event) then
elsif (not clk'stable) then
elsif (clk'stable) then
elsif (clk'stable and clk = '1') then
elsif (clk'event and clk'stable) then
-- If edge is not specified
elsif (not clk'event and clk = '0') then
elsif (clk'event and clk = "00") then
-- Multi-bit clock-signal
```

EXAMPLE (synthesizable)

```
wait until clk = ’1’;
if (clk’event and clk = ’1’) then
if (not clk’stable and clk = ’1’) then
wait until clk’event and clk = ’1’;
if (falling_edge(clk)) then
if (rising_edge(clk) then
if ((clk’event and not clk’stable) and clk = ’0’) then
wait until (clk’event and not clk’stable) and clk = ’0’;
```

STARC-2.3.1.3

Flip-flop data paths should have delays

EXAMPLE

```
always @(posedge CLK)
begin
Q <= D;
end
```

FIX

```
always @(posedge CLK)
begin
Q <= #5 D;
end
```

STARC-2.3.1.4

Paths other than flip-flop data paths must not have delays

EXAMPLE 1

```
entity add8 is
port(
a,b : in bit;
s,carry : out bit
);
end add8;
architecture struct_add8 of add8 is
begin
process ( a , b )
begin
s <= a xor b after 1 ns;
carry <= a and b after 2 ns;
end process;
end struct_add8;
```

EXAMPLE 2

```
entity latch is
port(
g,data : in bit;
q : out bit
);
end latch;
architecture struct_latch of latch is
begin
process ( g , data )
begin
if ( g ='1') then
q <= data after 2 ns;
end if;
end process;
end struct_latch;
```

EXAMPLE 3

```
module top (c,out, in, en,a,b);
output out,c;
input in, en;
input a,b;
reg out;
wire a, b;
assign #15 c = ~(a & b);
always @ (en or a or b or in)
if (en)
#4 out = in & a;
else
out = #10 in | b;
endmodule
```

STARC-2.3.1.5

Delay values must be integral and non-negative

STARC-2.3.1.5a

Delay values must be integral

STARC-2.3.1.5b

Ensure that the delay values are non-negative.

EXAMPLE

```
module mod_2_3_1_5test2(in1, clk, out1);
input [3:0] in1;
input clk;
output [3:0] out1;
reg [3:0] out1;
always @(posedge clk)
begin
out1 = #(-10) in1
end
endmodule
```

STARC-2.3.1.6

Check the logic level of the reset signal as specified in the sensitivity list of the always block.

EXAMPLE

```
module test(RESET,Q);
output Q;
input RESET;
reg Q,CLK,D;
always @(posedge CLK or negedge RESET)
if(RESET==1)
Q<= 1’b0;
else
Q<=D;
endmodule
```

STARC-2.3.1.7 (covered by STARC-1.3.1.7)

Flip-flops must not have both asynchronous set and asynchronous reset

—THE END—