Requirements Driven Design Verification Flow Tutorial

Ateş BERNA – Managing Partner

Ahmet JORGANXHI – Design & Verification Eng.

ELECTRA IC
Design & Verification

accellera
SYSTEMS INITIATIVE

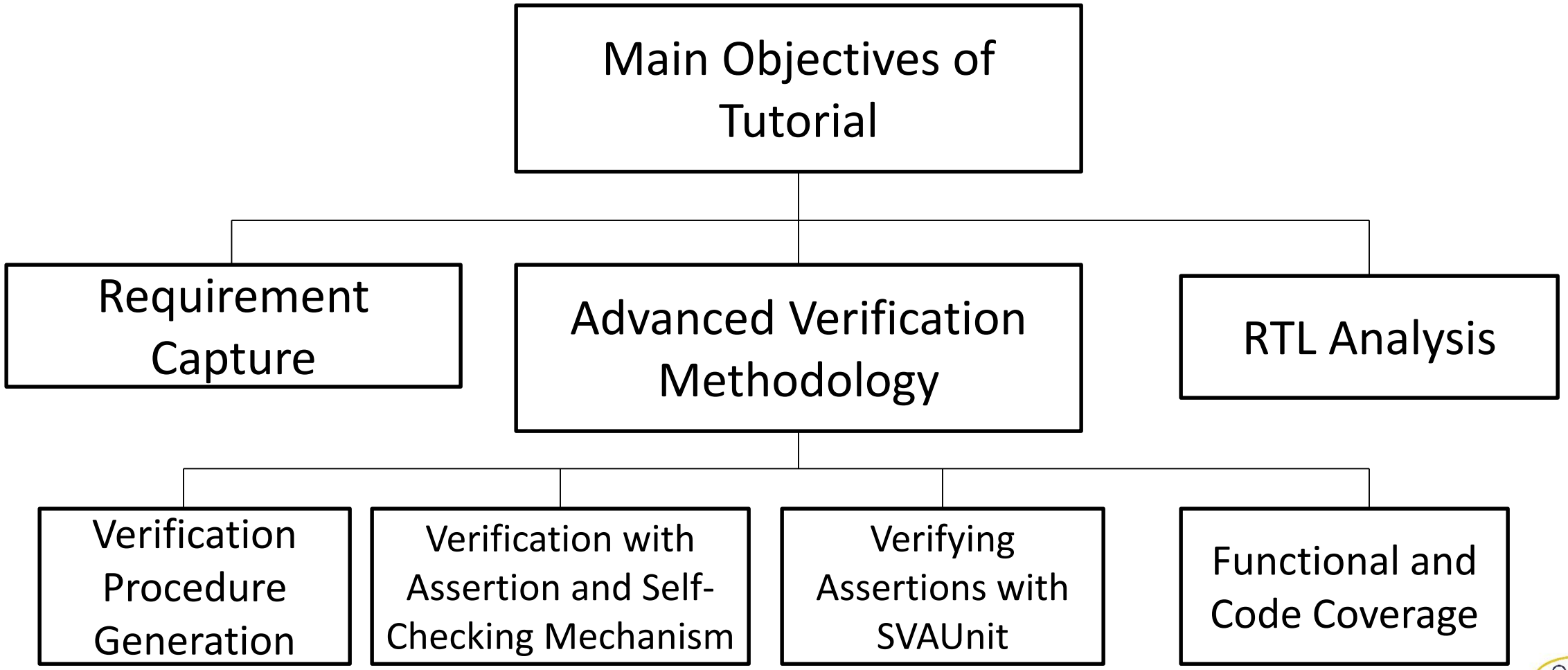
2018
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
EUROPE

About Electra IC

teknoparkistanbul

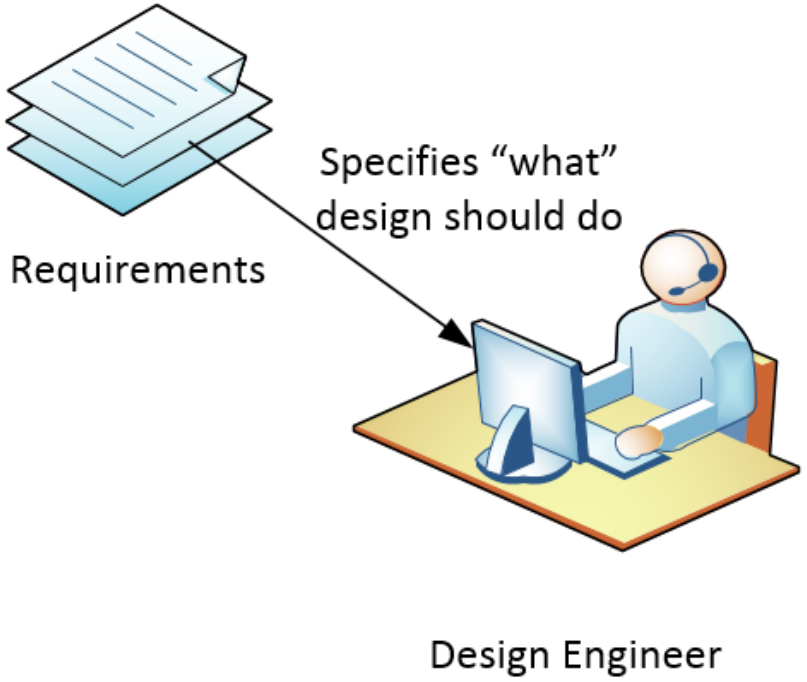
- Founded in 2014
- Headquarter in Istanbul, Turkey
- Branch office in Ankara
- Total 20 people
- ASIC/FPGA D&V Services
- ASIC/FPGA/EmbSys Training Services

Introduction



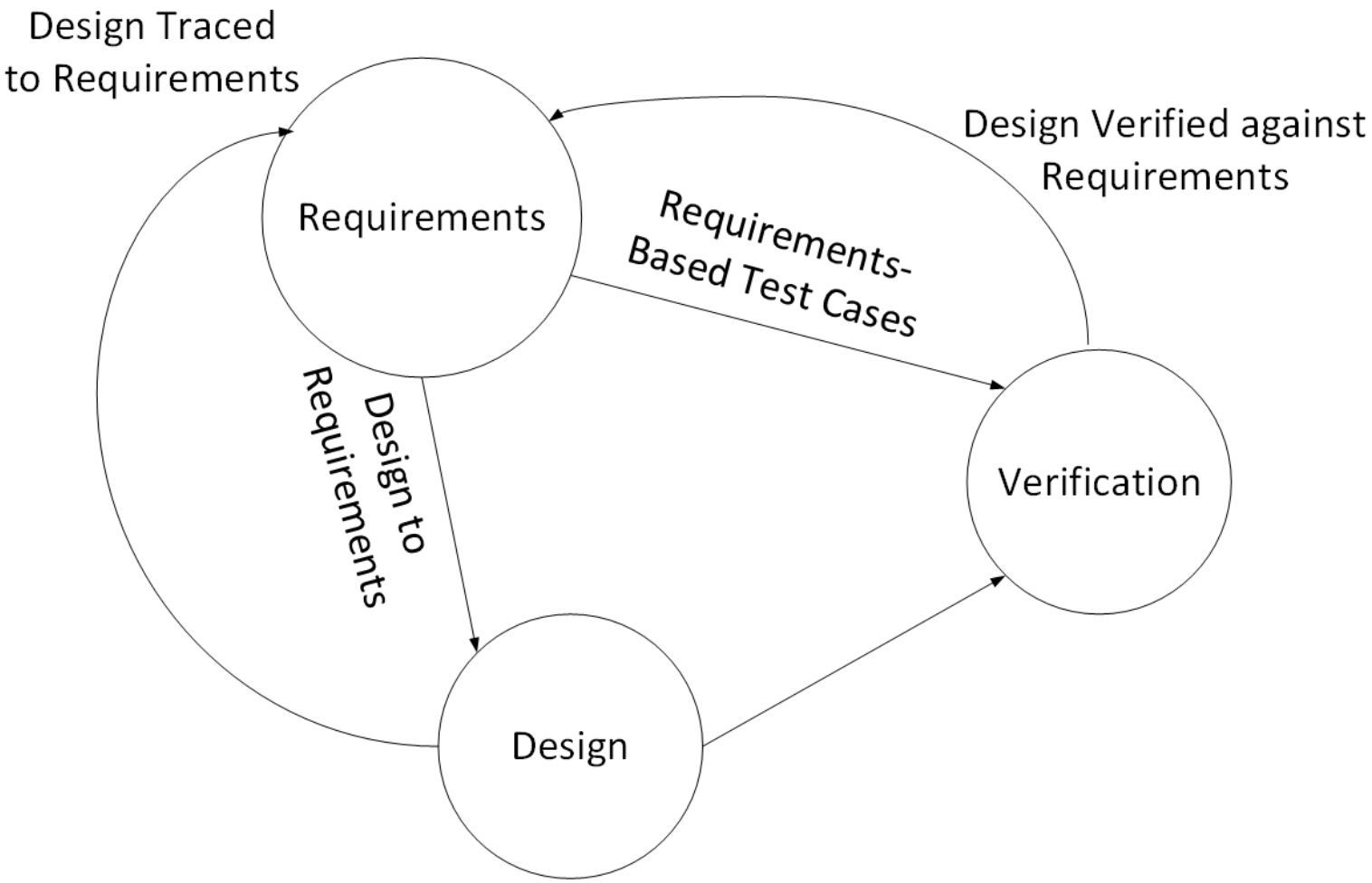
Requirement

Requirements are definitions of “what” hardware must do.



Requirement

Source:
Airborne Electronic Hardware Design Assurance
R. Fullton & R.Vandermolen, 2015



Properly Captured Requirement Format

The {output or verifiable aspect}
shall
{always, unconditionally, only}
{assert, deassert, set to value}
{before, after, when, during, within}
{xnsec, the next rising edge of a clock, read/write asserts low}
when {inputs are set to a combination of high/low,
a sequence of events has occurred or
a timed period elapses}

Source:
Airborne Electronic Hardware Design Assurance
R. Fullton & R.Vandermolen, 2015

Properly Captured Requirement Sample

EIC_IP_CORE_FR_001:

The **{dscrt_out}**

shall

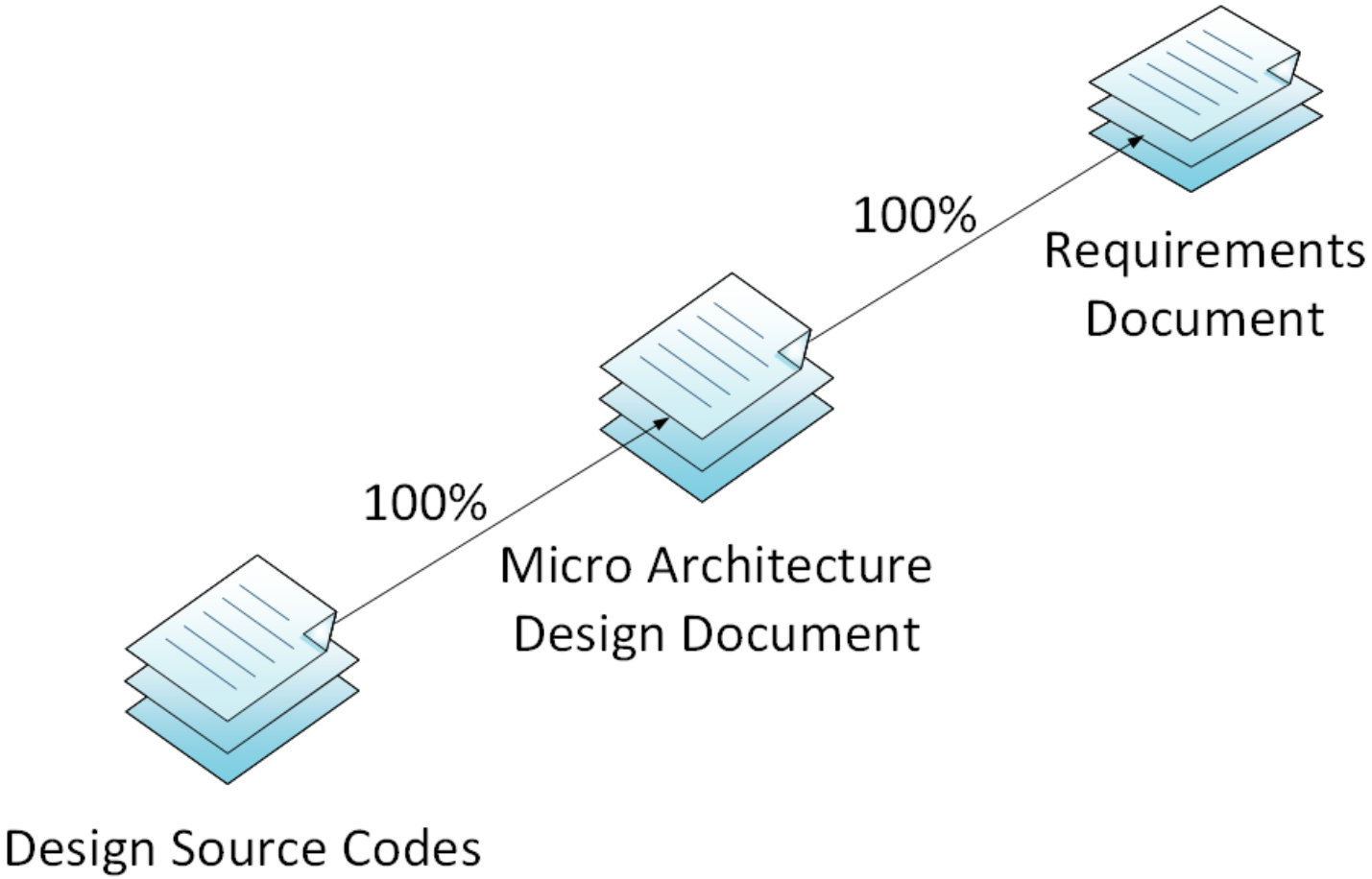
{always}

{assert to logic HIGH}

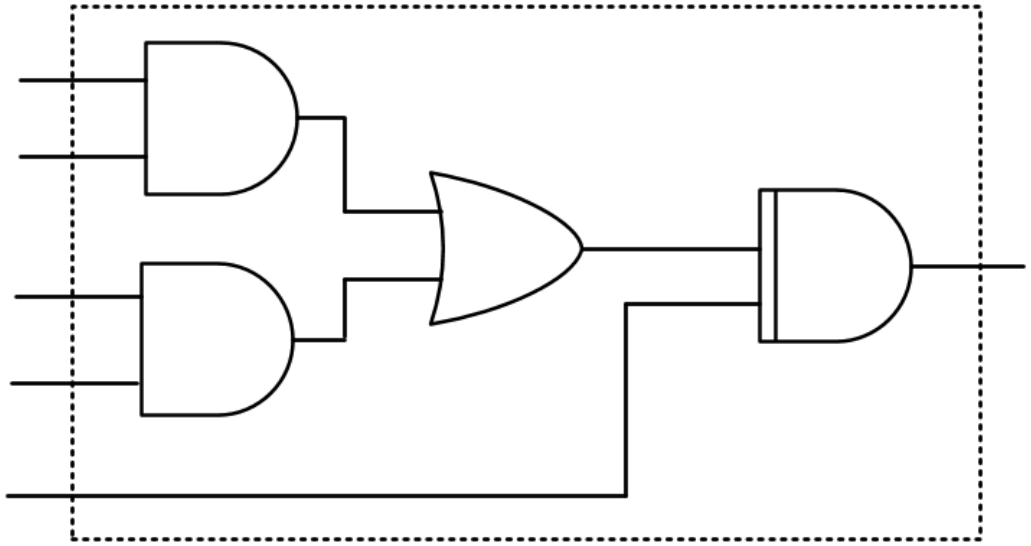
{within 40 nanoseconds}

when **{dscrt_in is asserted to logic HIGH}**

Requirement Tracing

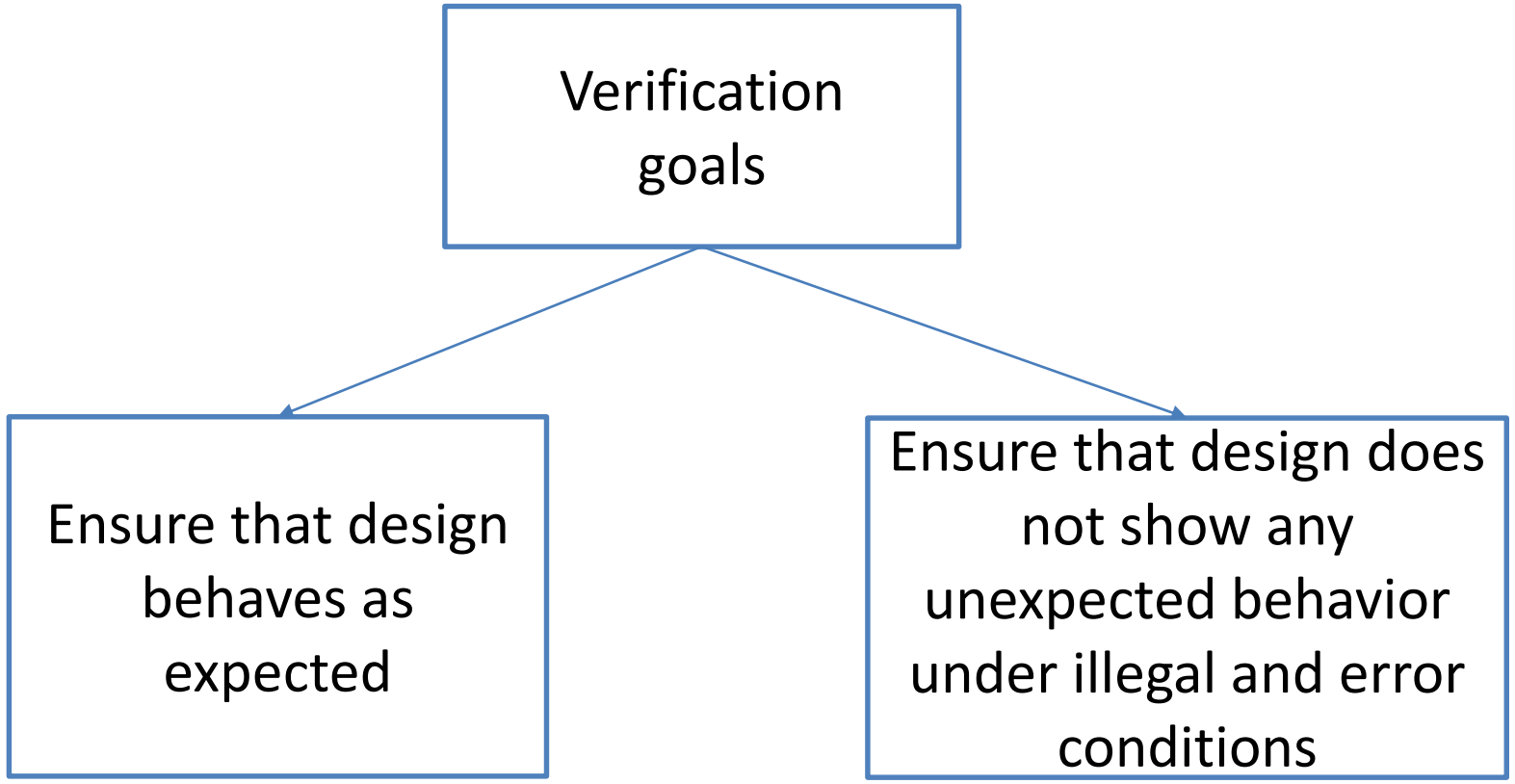


Introduction to Verification



Design **U**nder **T**est

Goals of Verification



Verification Methodologies Comparison

Features	Classic VHDL	SV/UVM	UVVM/OSVVM
OOP	-	+	-
Ease of use for whose who knows VHDL	+	-	+
Ease of use for whose who knows Verilog	-	+	-
Code Coverage	+	+	+
Functional Coverage	-	+	+
Development Tool Advantages	+	-	+
IEEE Standard	+	+	-
Extensive Verification IP Support	-	+	-

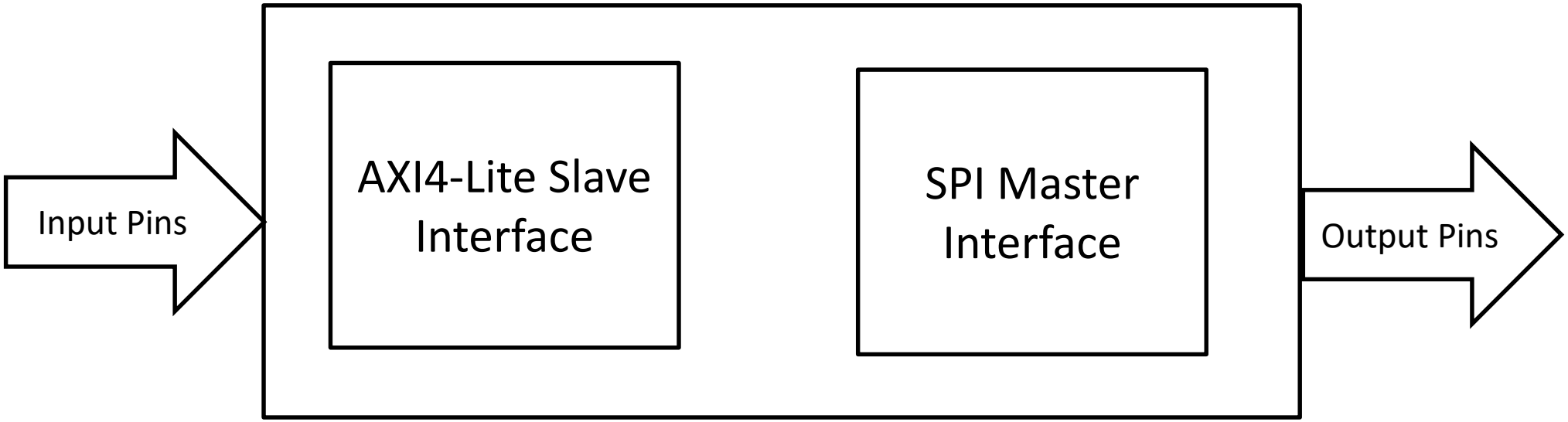
SV/UVM Verification Methodology

- Sequence Methodology
- Factory Mechanism
- Config Mechanism
- UVM Phase
- Modularity and Re-Usability

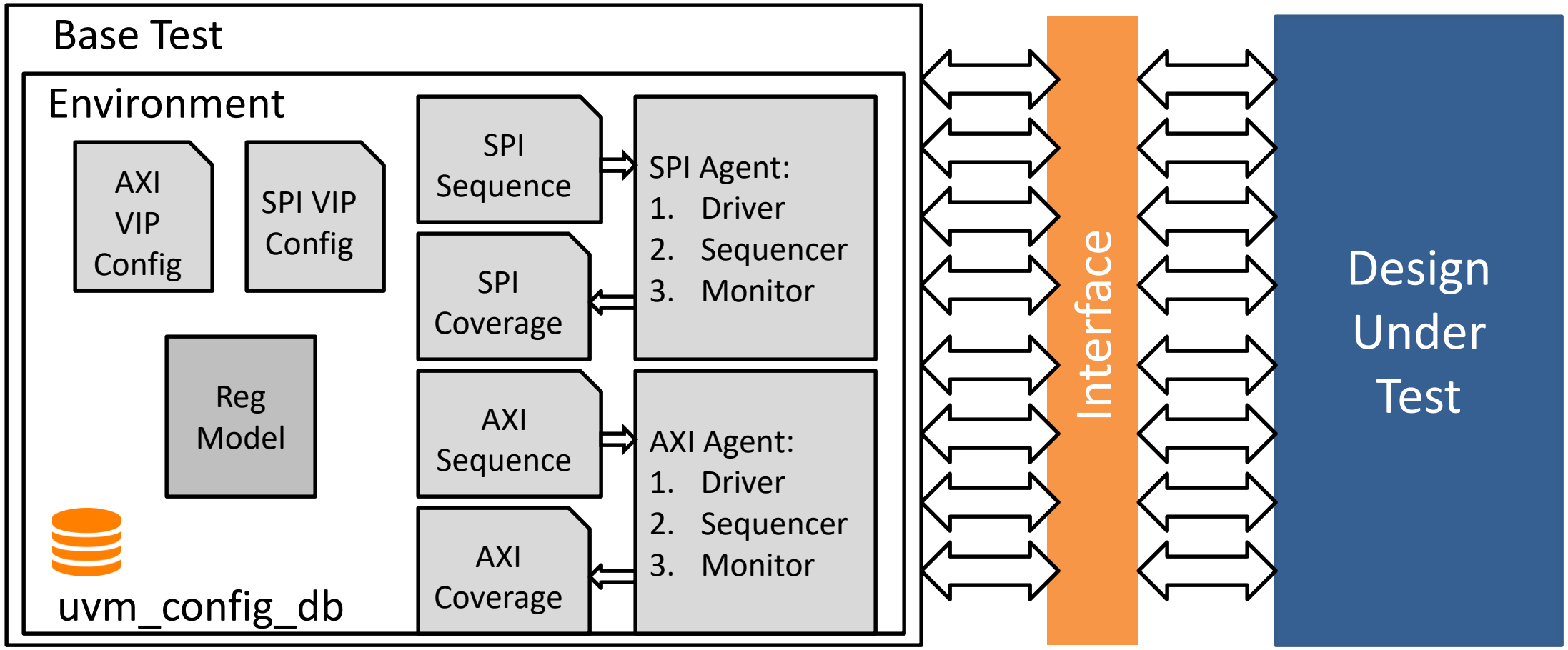
Verification Procedure Document

- Verification Environment
- For each testcase:
  - Description
  - Coverpoints
  - Test Steps

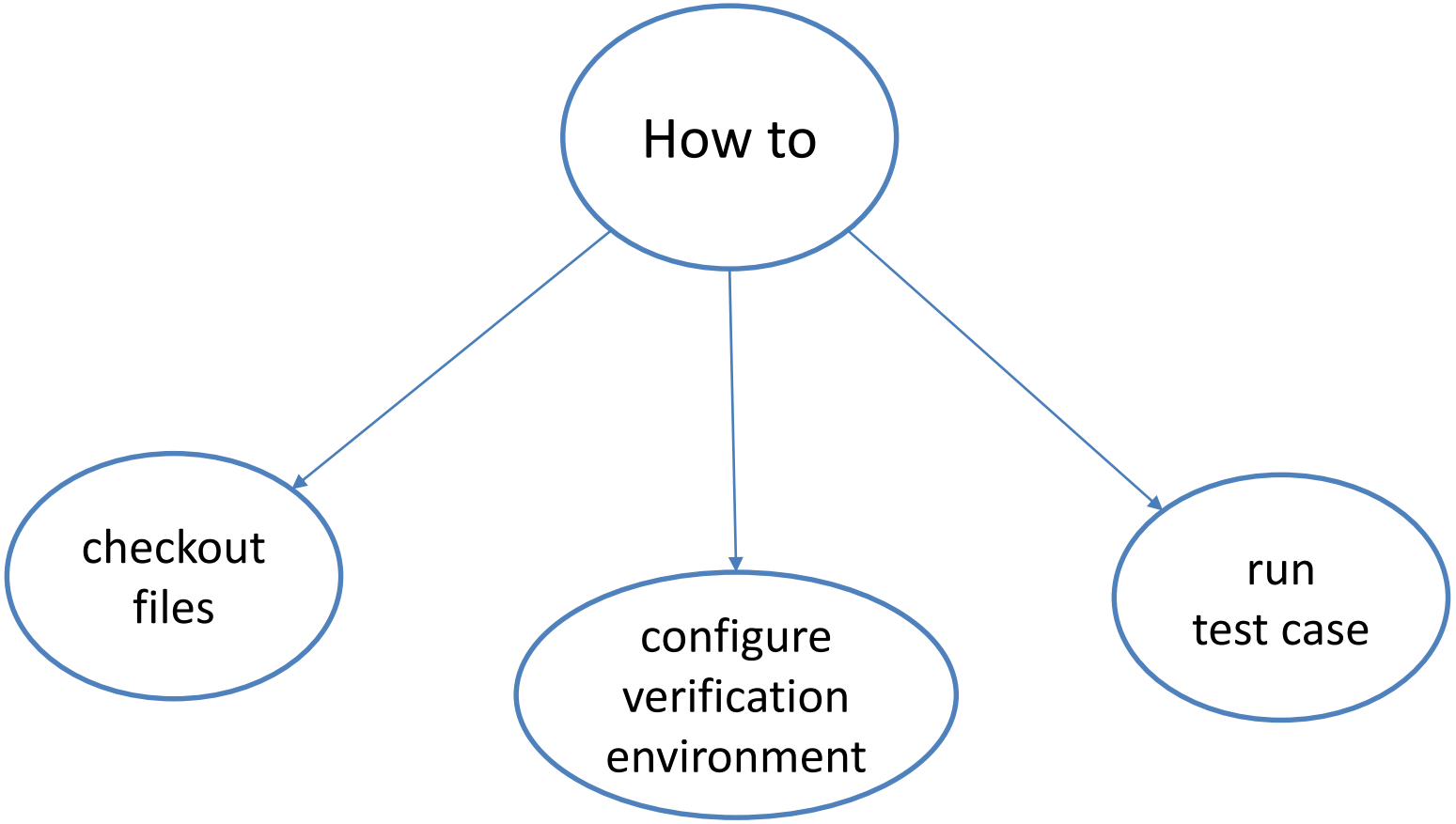
Verification Procedure Document



Verification Procedure Document



Verification Procedure Document



Verification Procedure Document

[EIC_IP_CORE_FR_002] *spi_cs_n* shall assert to logic LOW and remain for $24.5*T_{spi\_sclk} \pm 1\%$ when any of the Active State conditions of *spi_mosi* are satisfied. (Tspi_sclk = 10 MHz)

[EIC_IP_CORE_FR_005] *spi_sclk* shall assert to clock signal with the period 10 MHz when any of the Active State conditions of *spi_mosi* are satisfied.

Verification Procedure Document

[EIC_IP_CORE_FR_008] *spi_mosi* shall assert to opcode and data value where;

- Opcode : 0x04
- Data : SPI CONF DATA REG 2(15:0)

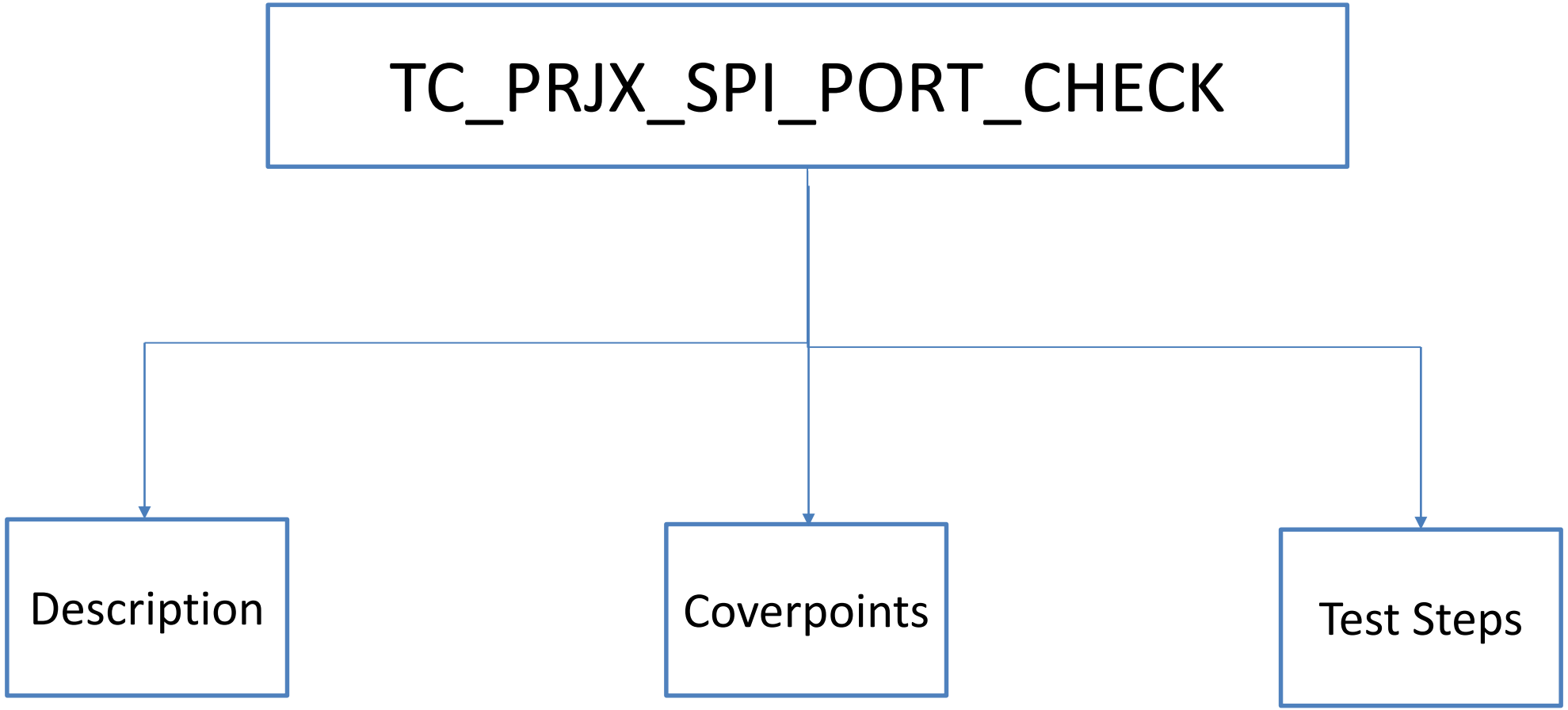
within 500*$T_{s\_axi\_aclk}$ ± 1% when SPI CONF DATA REG 2(31) is logic HIGH. ($T_{s\_axi\_aclk}$ = 100 MHz, SPI CONF DATA REG is register at address 0x8)

Verification Procedure Document

TC_ProjectName_FunctionalElement_Feature_CHECK

TC_PRJX_SPI_PORT_CHECK

Verification Procedure Document



Verification Procedure Document

Description

This test case verifies active state of output ports of SPI Master Interface which are listed below.

- spi_cs_n
- spi_sclk
- spi_mosi

As test scenario after applied reset, AXI4 Write Transfer to SPI CONF DATA REG 2 register which is located at address 0x8 with 31st of Write Data set to 1 and (30:0) set to a random value will be initiated. Then, at required time, active states of spi_cs_n, spi_sclk and spi_mosi output ports are verified.

Pass/Fail Criteria: Assertions and checkers of requirement checking mechanism of test class should pass.

Verification Procedure Document

Coverpoints

spi_master_cg: spi_cs_cp
spi_master_cg: spi_opcode_cp
spi_master_cg: spi_trans_type_cp
axi4_slave_cg: axi4_trans_type_cp
reg_cg: spi_conf_reg2_cp

Verification Procedure Document

Action	Expected Result	Assertion	Requirement Links
1. Apply clock signal with period of 100 MHz to sys_clk input.			
2. Set reset_n to logic LOW.			
3. Wait for 4 to 10 sys_clk cycles.			
4. Set reset_n to logic HIGH			

Verification Procedure Document

Action	Expected Result	Assertion	Requirement Links
5. Initiate AXI4 Write Transfer to address 0x8 with 31st bit of write data set to 1 and (30:0) bits of write data set to random value.			

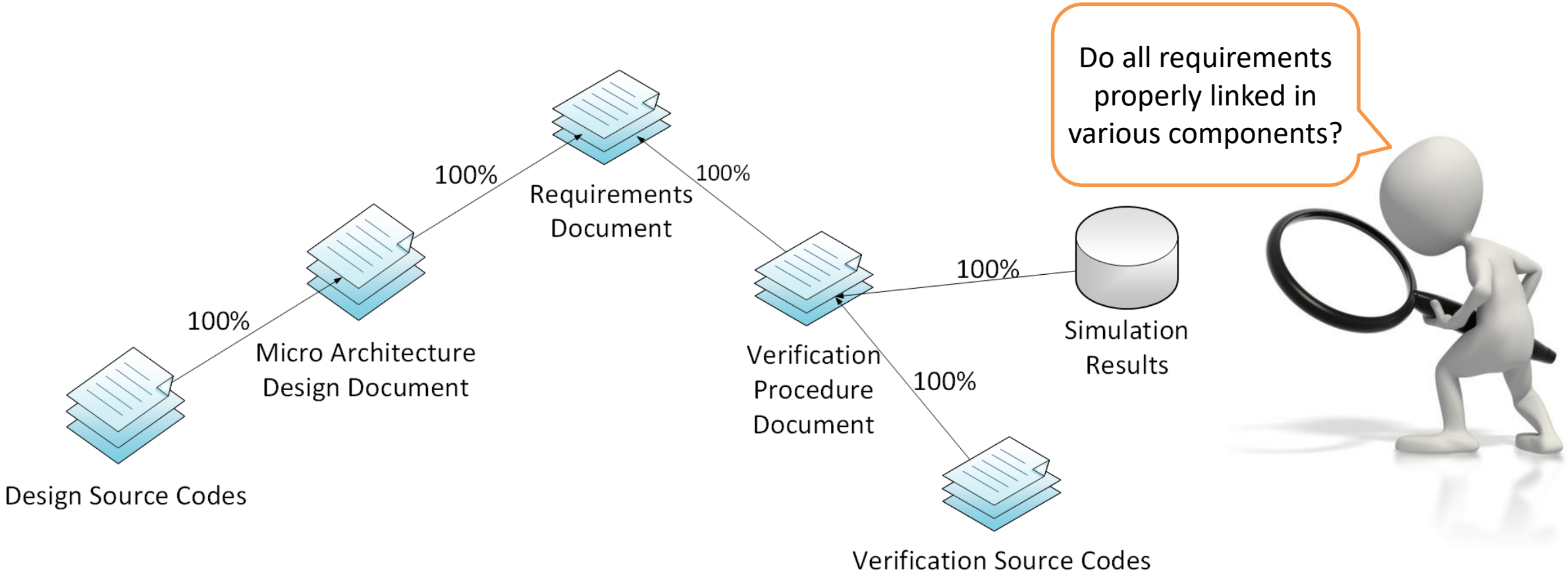
Verification Procedure Document

Action	Expected Result	Assertion	Requirement Links
6. Perform steps 6.1-6.3 in parallel manner			
6.1 Within 500*sys_clk± 1% cycles check spi_mosi.	spi_mosi should assert respectively to opcode and data byte in order: Opcode : 0x04 Data Byte : (15:0) bits of write data initiated in step 5.		EIC_IP_CORE_FR_008

Verification Procedure Document

Action	Expected Result	Assertion	Requirement Links
6.2 Within 500*sys_clk± 1% check spi_cs_n. Keep checking for 24.5*Tspi_sclk± 1%.	spi_cs_n should assert and remain at logic LOW.	eic_ip_core_fr_003_spi_cs_n_check	EIC_IP_CORE_FR_003
6.3 Within 500*sys_clk± 1% check spi_sclk.	spi_sclk should assert to clock signal with period of 10 MHz.	eic_ip_core_fr_005_spi_sclk_check	EIC_IP_CORE_FR_005

Requirement Tracing

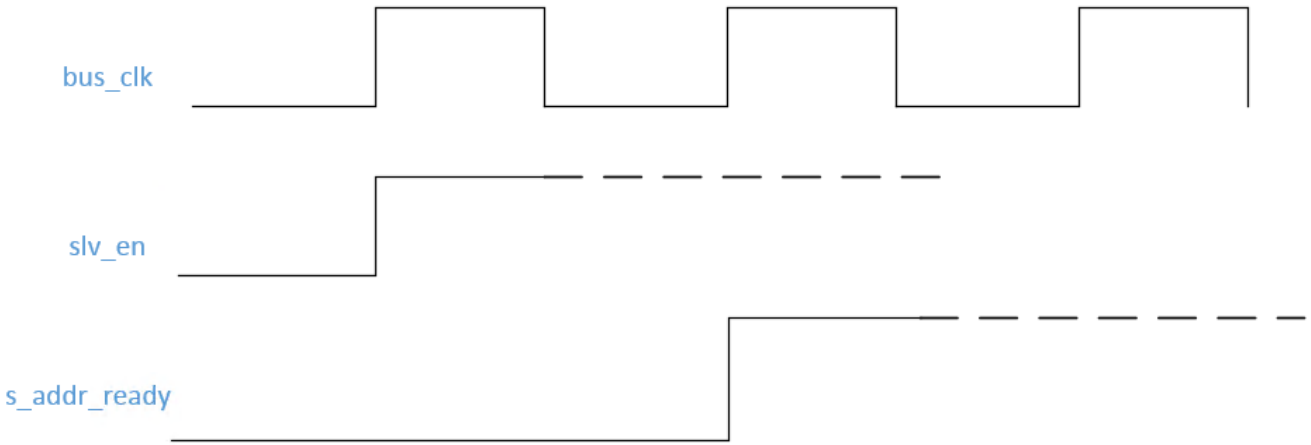


Requirement Verification

Verification with:

- Assertion
- Self-Checking

Verification with Assertion



EIC_IP_CORE_FR_100

"s_addr_ready shall assert to 0x1 after one clock cycle slv_en is asserted to 0x1"

Verification with Assertion

```
cover property (S_ADDR_READY_ASSERT);
```

```
property S_ADDR_READY_ASSERT;
 @(posedge bus_clk)
 ($rose(slv_en)) |=> (s_addr_ready);
 endproperty
```

```
assert property (S_ADDR_READY_ASSERT)
else $error(“EIC_IP_CORE_FR_100 has failed.");
```

Verification with Self-Checking Mechanism

EIC_IP_CORE_FR_102

"i2c_sda shall assert to I2C Write Transfer Sequence within 5us after reset_n is set to logic HIGH with the following conditions:"

- *Slave Address : 7b1101010*
- *Register Address : 0xC4*
- *Write Data Byte in order: 0xD0, 0xC1 and 0xAA*

Verification with Self-Checking Mechanism

Task which verifies:

1. Time when i2c_sda is deasserted after reset.
2. Asserted value of the i2c_sda

Verification with Self-Checking Mechanism

"i2c_sda shall assert to I2C Write Transfer Sequence within 5us after ***reset_n is set to logic HIGH*** *with the following conditions:"*

- *Slave Address : 7b1101010*
- *Register Address : 0xC4*
- *Write Data Byte in order: 0xD0, 0xC1 and 0xAA*

```
wait(dut_if.reset_n == 1);
```

"i2c_sda shall assert to I2C Write Transfer Sequence ***within 5us*** *after reset_n is set to logic HIGH with the following conditions:"*

- *Slave Address : 7b1101010*
- *Register Address : 0xC4*
- *Write Data Byte in order: 0xD0, 0xC1 and 0xAA*

```
tinit = $time;
```

"i2c_sda shall assert to I2C Write Transfer Sequence within 5us after reset_n is set to logic HIGH with the following conditions:"

- ***Slave Address : 7b1101010***
- ***Register Address : 0xC4***
- ***Write Data Byte in order: 0xD0, 0xC1 and 0xAA***

```
check_i2c_write
(EIC_IP_CORE_FR_102,7b1101010, 8hC4, 8hD0, 8hC1, 8hAA, tinit, 5us)
```

Verification with Self-Checking Mechanism

check_i2c_write
(Req. ID, Slave Addr., Register Addr., Write Data, Starting Time, Req. Time)

"i2c_sda shall assert to I2C Write Transfer Sequence ***within 5us*** *after reset_n is set to logic HIGH with the following conditions:"*

- *Slave Address : 7b1101010*
- *Register Address : 0xC4*
- *Write Data Byte in order: 0xD0, 0xC1 and 0xAA*

```
wait(m_i2c_seq.ev0.triggered);
tfinal = $time;
```

Verification with Self-Checking Mechanism

"i2c_sda shall assert to I2C Write Transfer Sequence within 5us after reset_n is set to logic HIGH with the following conditions:"

- ***Slave Address : 7b1101010***
- *Register Address : 0xC4*
- *Write Data Byte in order: 0xD0, 0xC1 and 0xAA*

```
wait(m_i2c_seq.ev0.triggered);
slv_addr = m_i2c_seq.data_buf;
wait(m_i2c_seq.ev0.triggered);
```

Verification with Self-Checking Mechanism

"i2c_sda shall assert to I2C Write Transfer Sequence within 5us after reset_n is set to logic HIGH with the following conditions:"

- *Slave Address : 7b1101010*
- ***Register Address : 0xC4***
- ***Write Data Byte in order: 0xD0, 0xC1 and 0xAA***

- Register Address → reg_addr
- First Write Data Byte → wr_data[0]
- Second Write Data Byte → wr_data[1]
- Third Write Data Byte →wr_data[2]

Verification with Self-Checking Mechanism

"i2c_sda shall assert to I2C Write Transfer Sequence ***within 5us after reset_n is set to logic HIGH*** *with the following conditions:"*

- ***Slave Address : 7b1101010***
- ***Register Address : 0xC4***
- ***Write Data Byte in order: 0xD0, 0xC1 and 0xAA***

```
if(({Slave Addr, 1b0} == slv_addr) && (Register Addr == reg_addr) &&
(Write Data [0] == wr_data[0]) && (Write Data [1] == wr_data[1]) &&
(Write Data [2] == wr_data[2]) && (tfinal – tinit <= Req. Time))
`uvm_info("req_pass", $sprint("Requirement: %s has passed", Req.ID), UVM_LOW)
else
`uvm_error("req_fail", $sprint("Requirement: %s has failed", Req. ID))
```

Verification with Self-Checking Mechanism

EIC_IP_CORE_FR_103

Control Register should assert to ***write data byte of the AHB Write Transfer within 2 clk cycles****, after all of the following conditions are satisfied:*

- *reset_n is set to logic HIGH*
- *AHB Write Transfer to Control Register address is completed*

Verification with Self-Checking Mechanism

Register Class

```
class Control_Reg extends uvm_reg;
function new(string name = "Control_Reg")
        super.new(name, 16, UVM_NO_COVERAGE)
endfunction

virtual function void build();
        Control_Reg = uvm_reg_field::type_id::create("Control_Reg");
        Control_Reg.configure(this, 16, 0, "RW", 0, 16'h0000, 1, 1, 1);
endfunction
```

Register width

size

lsb

access type

volatility

reset

has_reset

is_rand

individually access

Verification with Self-Checking Mechanism

In the test class register block is defined

REG block m_reg_block;

Control Register should assert to write data byte of the AHB Write Transfer within 2 clk cycles, after all of the following conditions are satisfied:

- ***reset_n is set to logic HIGH***
- ***AHB Write Transfer to Control Register address is completed***

```
wait(dut_if.reset_n == 1);
m_reg_block.Control_Register_h.write(status, write_value);
```

Verification with Self-Checking Mechanism

*Control Register should assert to **write data byte** of the AHB Write Transfer within 2 clk cycles, after all of the following conditions are satisfied:*

- *reset_n is set to logic HIGH*
- *AHB Write Transfer to Control Register address is completed*

```
get_value = m_reg_block.Control_Register_h.get(status);
```

*Control Register should assert to write data byte of the **AHB Write Transfer within 2 clk cycles**, after all of the following conditions are satisfied:*

- *reset_n is set to logic HIGH*
- *AHB Write Transfer to Control Register address is completed*

```
repeat(2) @(posedge bus_if.clk);
m_reg_block.Control_Register_h.read(status, read_value);
```

Verification with Self-Checking Mechanism

Control Register should assert to ***write data byte of the AHB Write Transfer within 2 clk cycles****, after all of the following conditions are satisfied:*

- *reset_n is set to logic HIGH*
- *AHB Write Transfer to Control Register address is completed*

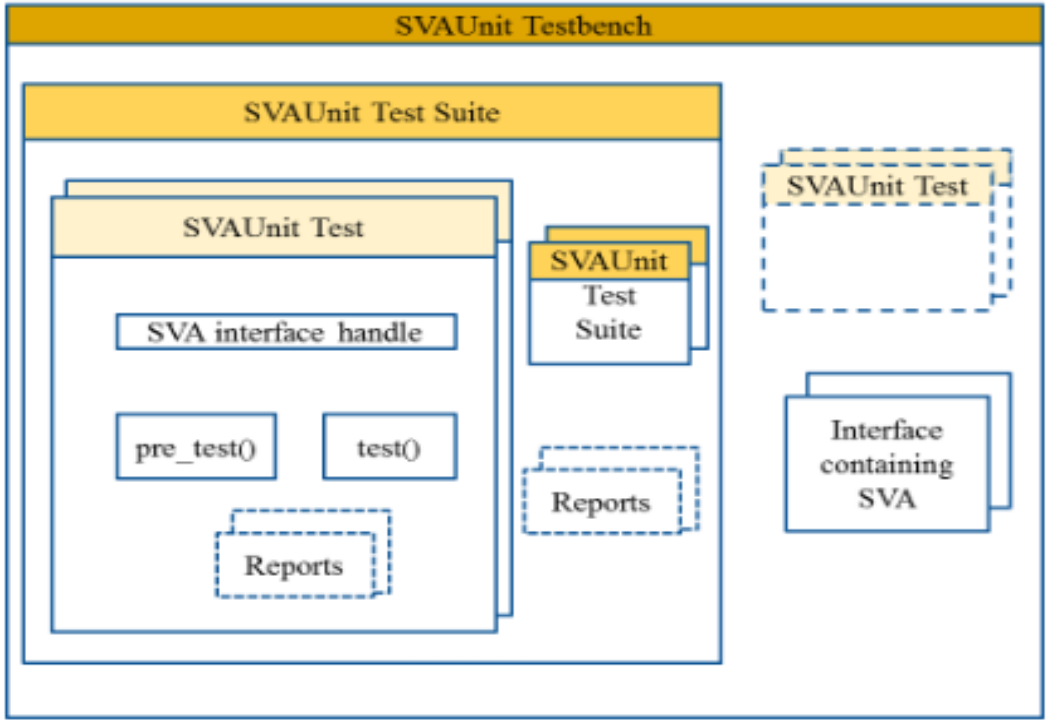
```
If(read_value == get_value)
        `uvm_info("req_pass", $sprint("Requirement EIC_IP_CORE_FR_103
 has passed"), UVM_LOW)
else
        `uvm_error("req_fail", $sprint("Requirement: EIC_IP_CORE_FR_103
 has failed", Req. ID))
```

accellera SYSTEMS INITIATIVE

2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Verification of Assertion with SVA Unit

Source:
Socianu.A & Ciocirlan.I (2015, April 29)
SystemVerilog Assertion Verification with SVAUnit
Retrieved from: https://www.amiq.com



Verification of Assertion with SVA Unit

Is assertion working as specified?

Verification of Assertion with SVA Unit

		SVAUnit checks			
		CHECK_SVA_EXISTS	CHECK_SVA_ENABLED	CHECK_SVA_PASSED	CHECK_SVA_FAILED
SVAs	I2C_CLK_SVA	✓	✓	✓	✓

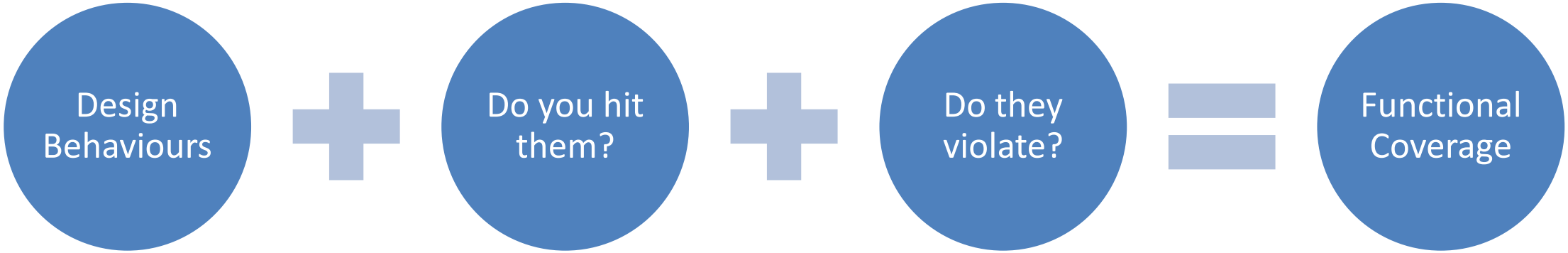
Coverage Analysis

We need metrics for…

Coverage Analysis

- Functional Coverage
- Code Coverage
- Other Coverage Types
  - Linting (quality of RTL code)
  - Clock Domain Crossing (metastability)

Functional Coverage



Functional Coverage

```
cover property (S_ADDR_READY_ASSERT);
```

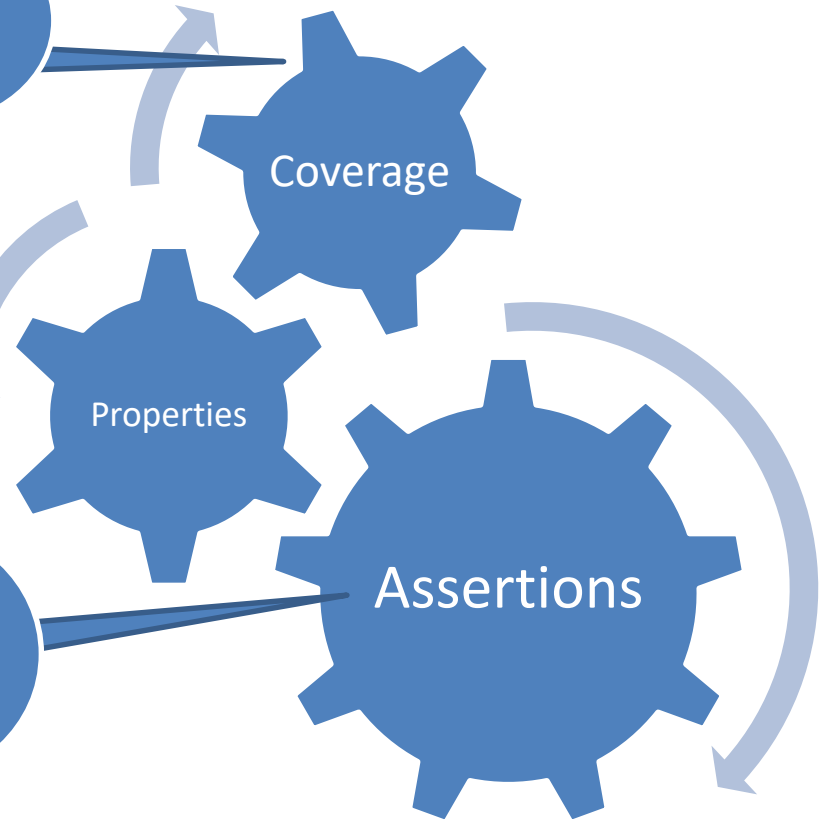
Do you hit them?

```
property S_ADDR_READY_ASSERT;
 @(posedge bus_clk)
 ($rose(slv_en)) |=> (s_addr_ready);
 endproperty
```

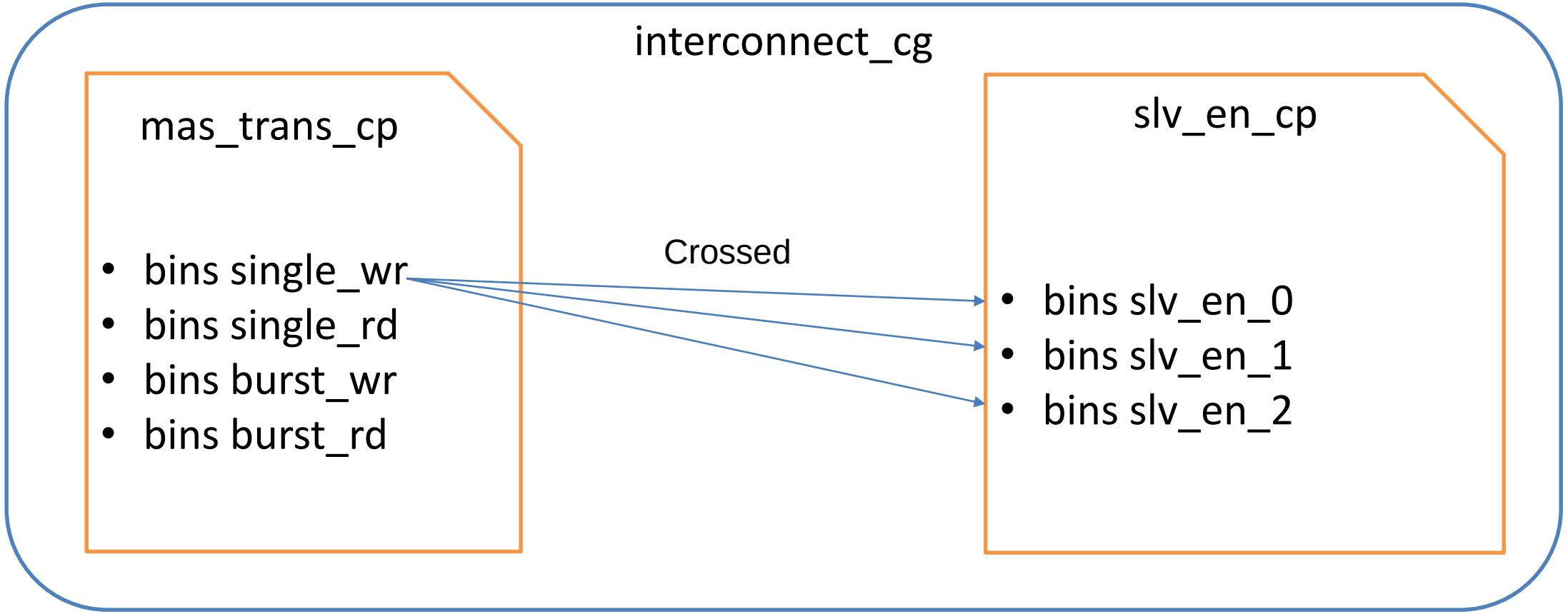
Design Behaviours

```
assert property (S_ADDR_READY_ASSERT)
else $error("EIC_IP-CORE_FR_100 is
failed.");
```

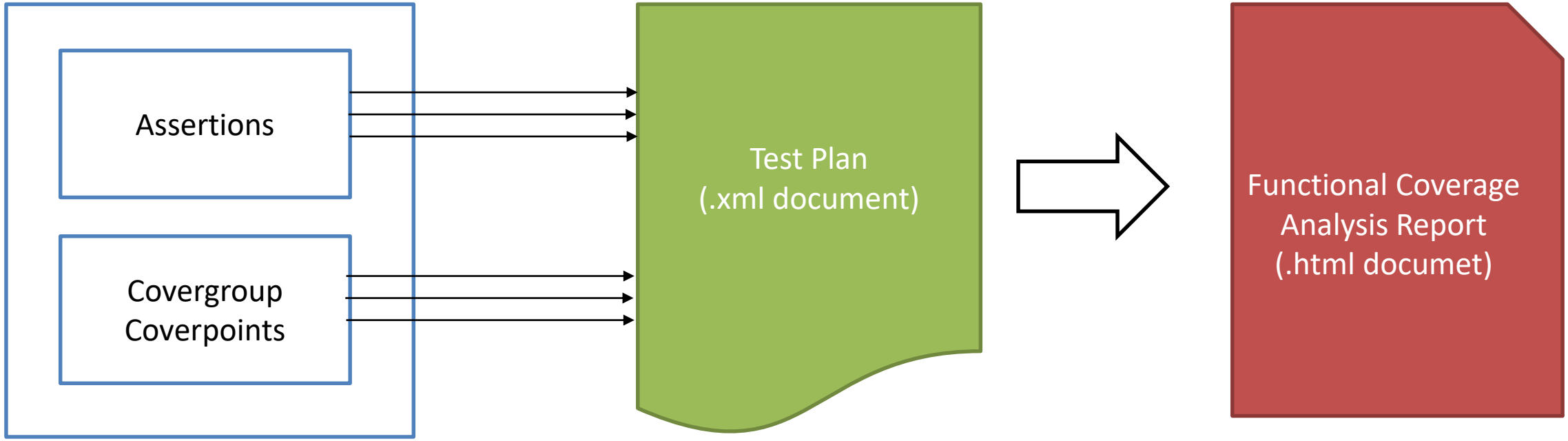
Do they violate?



Functional Coverage



Functional Coverage



Functional Coverage

3.2.1	**TC_PRJX_SPI_PORT_CHECK**				**1**	**100**
3.2.1.1	TC_PRJX_SPI_PORT_CHECK_1cp		spi_master_cg:spi_cs_cp spi_master_cg:spi_opcode_cp spi_master_cg:spi_trans_type_cp axi4_slave_cg:axi4_trans_type_cp reg_cg:spi_conf_reg2_cp	CoverPoint	1	100
3.2.1.2	TC_PRJX_SPI_PORT_CHECK_2a		eic_ip_core_fr_003_spi_cs_n_check eic_ip_core_fr_005_spi_sclk_check	Assertion	1	100
3.2.1.3	TC_PRJX_SPI_PORT_CHECK_3cd		eic_ip_core_fr_003_spi_cs_n_cov eic_ip_core_fr_005_spi_sclk_cov	Directive	1	100

Coverage Summary by Testplan Section:

Scope	Coverage	% of Goal
3.2.1 TC_PRJX_SPI_PORT_CHECK	100.00%	100.00%
3.2.1.1 TC_PRJX_SPI_PORT_CHECK_1cp	100.00%	100.00%
3.2.1.2 TC_PRJX_SPI_PORT_CHECK_2a	100.00%	100.00%
3.2.1.3 TC_PRJX_SPI_PORT_CHECK_3cd	100.00%	100.00%

Code Coverage

- **S**tatement—Did we cover every statement?
- **B**ranch—Did we cover every IF branch and CASE entry?
- **F**inite State Machine—Did we cover all states and transitions?
- **E**xpression—Did we fully test our single-bit expressions?
- **C**ondition—Did we test all the conditions in our IF statements?

Code Coverage

Statement Coverage

Statement Coverage

```
process (a)
begin
  z <= '0'; if (a = 0) then z <= '1';
end process;
```

Line Coverage

Code Coverage

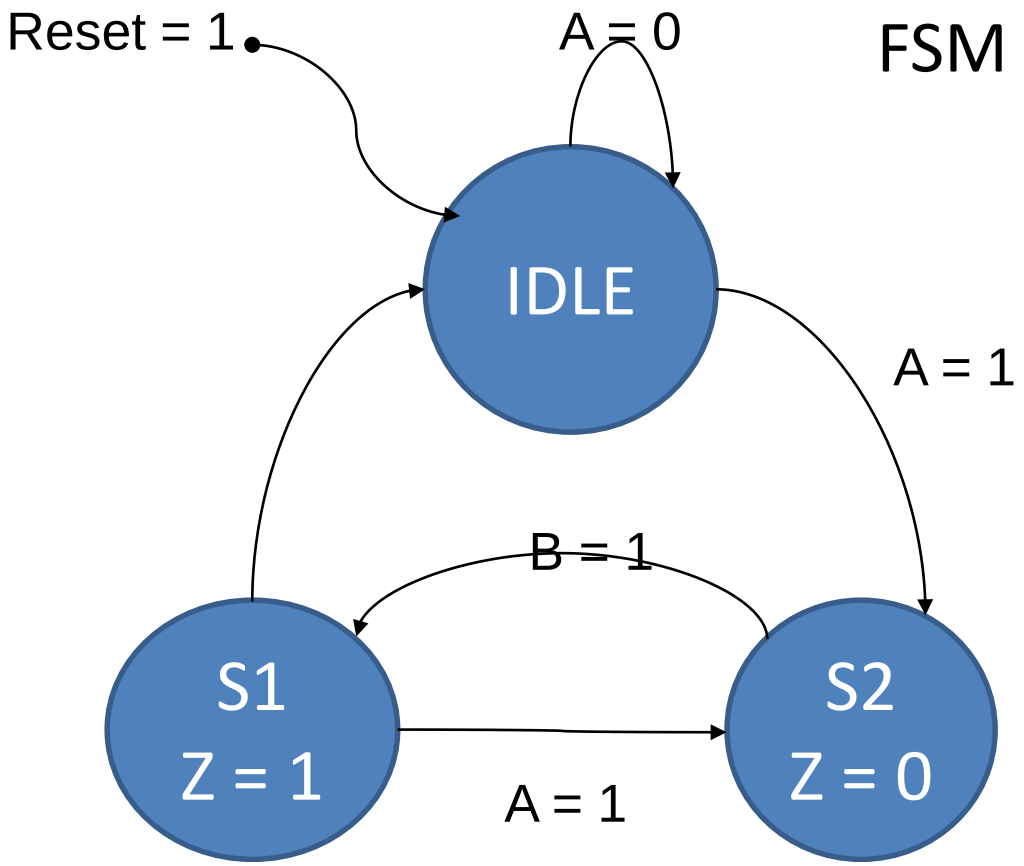
Branch Coverage

```
if (a = '0') then
  z <= '1';
else
  z <= '0';
end if;
```

Branch Coverage

```
case a is
  when '0' => z <= '1';
  when '1' => z <= '0';
  when others => z <= '0';
end case;
```

Code Coverage



FSM Coverage

Transition Coverage

State Coverage

Code Coverage

Condition Coverage

```
if (a or b) then
  Z <= C + 1;
end if;
```

- Condition 1 -> a = 1
- Condition 2 -> b = 1

Code Coverage

Expression Coverage

```
Z <= A or B;
```

A	B	Z
0	0	0
1	0	1

A	B	Z
0	0	0
0	1	1

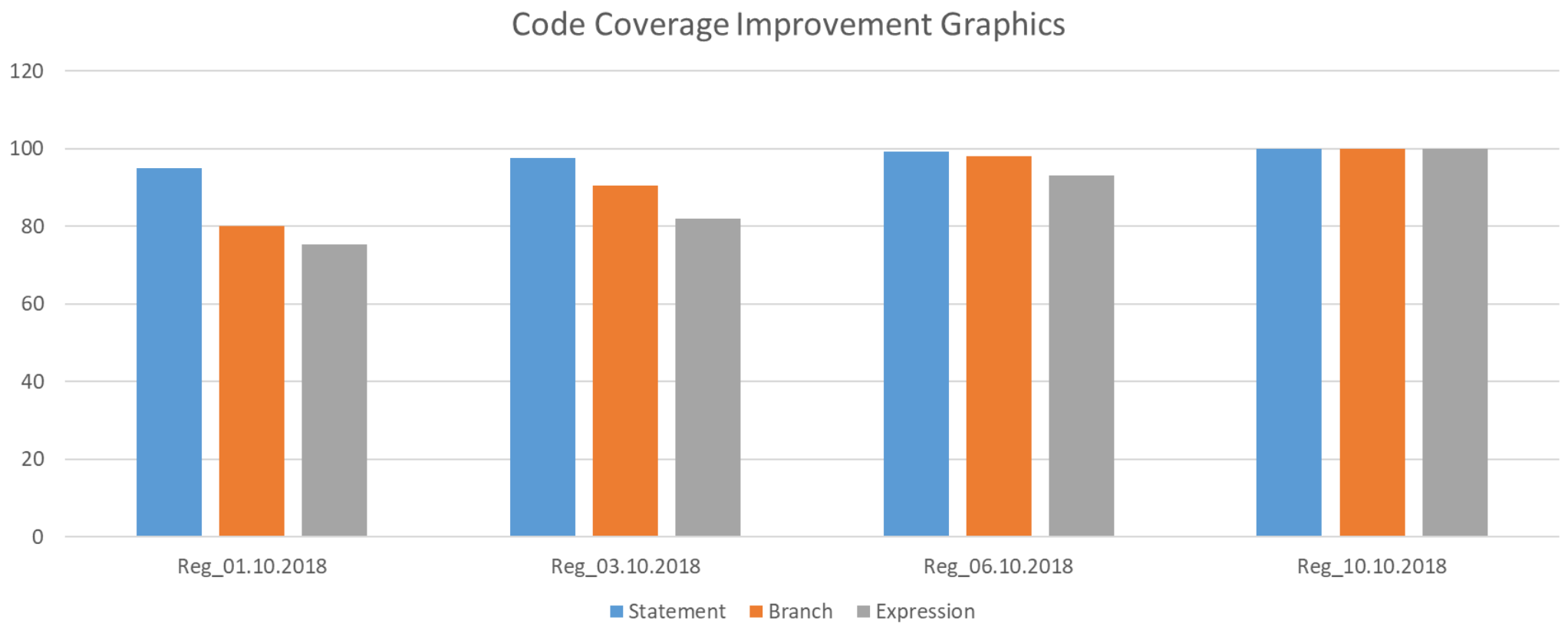
Code Coverage

- Why corresponding statement is not covered?
- Why corresponding IF branch or CASE entry is not covered?
- Why corresponding state or transition is not covered?
- Why corresponding single-bit expression is not covered?
- Why corresponding condition in related IF is not covered?

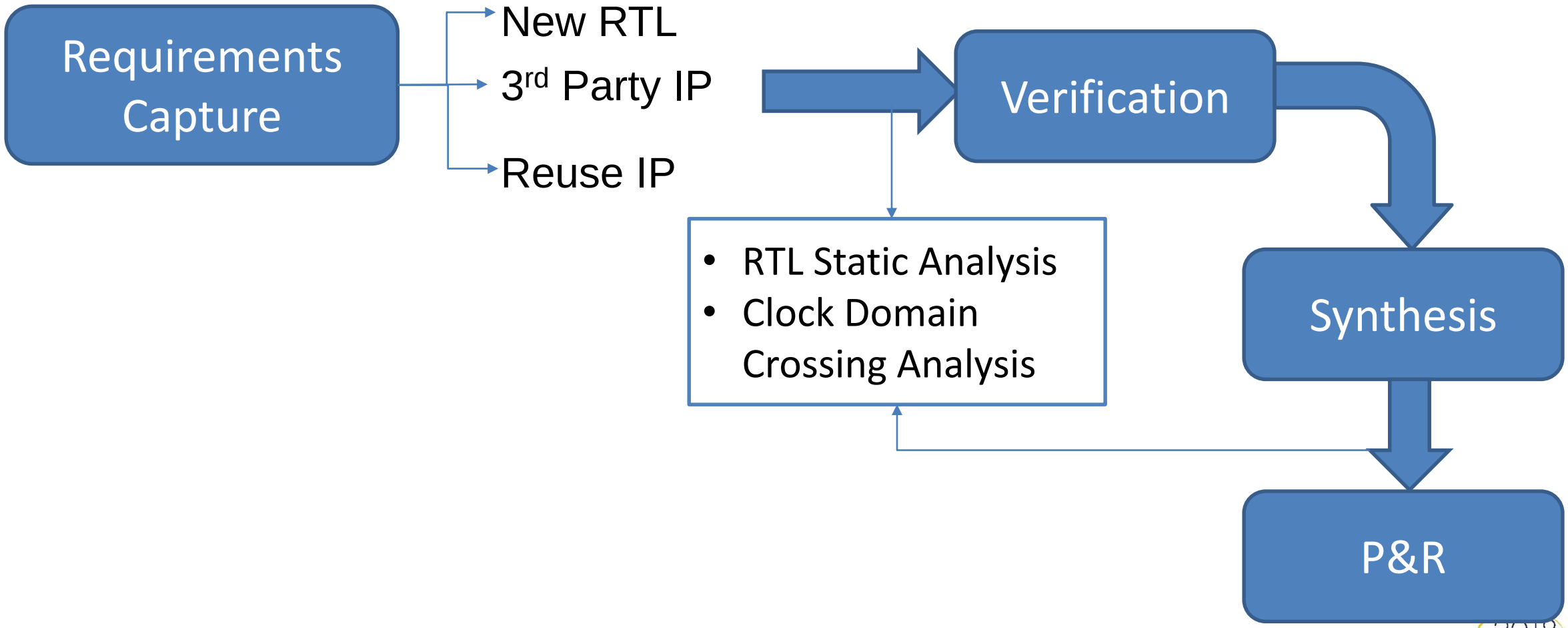
Code Coverage

- Reason of the code coverage holes:
  - Unused Codes Lines
  - Missing Test Scenario

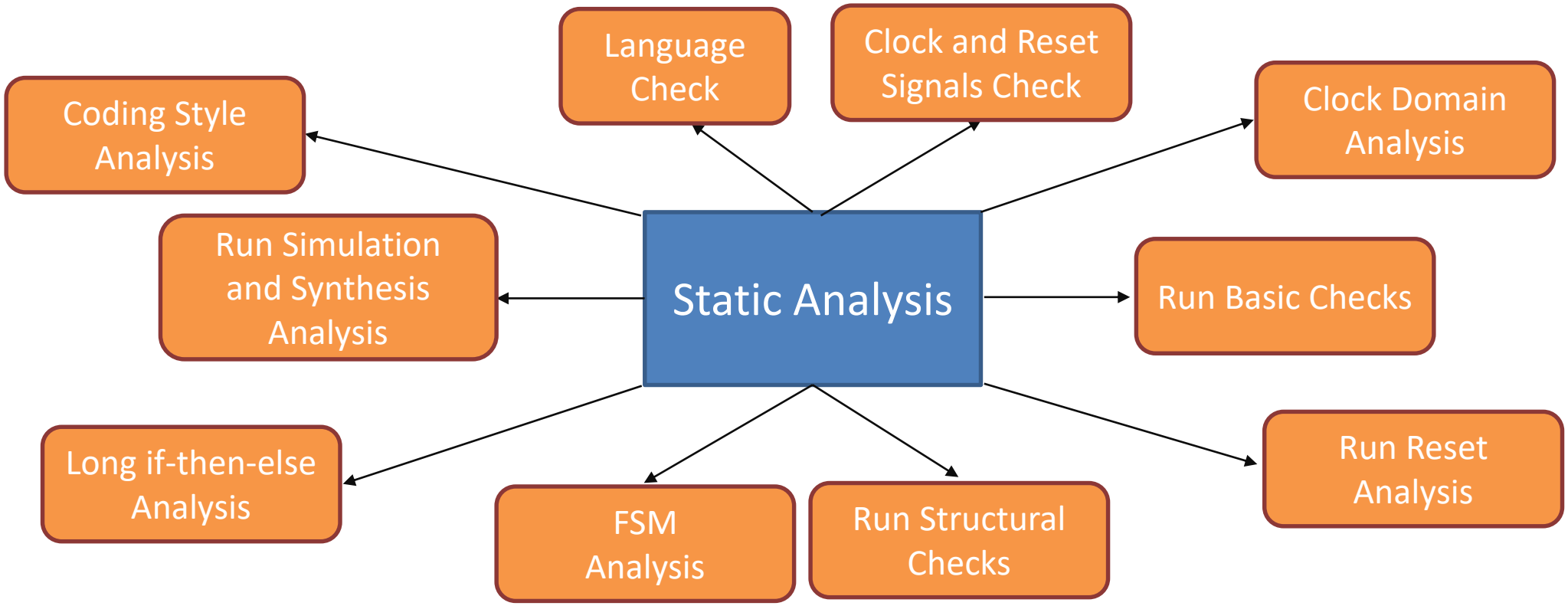
Code Coverage



RTL Analysis



RTL Analysis



RTL Analysis – Static Checks

- [x] No X-source problems
- [x] Clocks
- [ ] Clock Domain Crossings
- [ ] Advanced Clock Environment
- [ ] Coding style
- [x] Coding conventions
- [x] Assignment Checks
- [x] Simulation/Synthesis
- [x] No implied_latches
- [x] Case statements
- [x] No size conflicts
- [ ] Naming
- [x] Signal Identification
- [ ] Comments
- [x] FSM Checks
- [ ] Miscellaneous checks
- [ ] SDC Verification
- [x] Cycle Based Simulation
- [x] Input IP
- [x] New RTL
- [x] Golden RTL
- [x] DFT
- [x] Principles of Verifiable RTL
- [x] Reuse Methodology Manual
- [x] Semiconductor Reuse Standard
- [x] DO-254
- [x] STARC
- [x] UltraFast Design Methodology for Vivado
- [x] Quartus II Best Practices
- [x] Microsemi RTG4 Best Practices

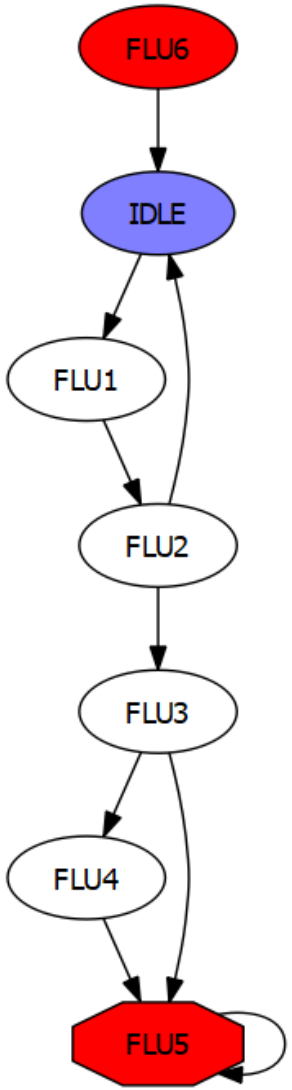
Package: DO-254

Description: This package includes checks that aid in compliance with the DO-254 military standard.

Checks:

- GRST (Gated reset)
- HCCC (Do not hard-code constants)
- REGO (Register all module outputs)
- UNREACHABLE_STATE (Report Unreachable states)
- MCD (No case default)
- MDA (Missing case default assignment)
- CSL (Complete sensitivity lists)
- MEB (Missing else block)
- ETB (Empty then block)
- LEC (Little endian checks)
- CCLP (Infinite and 0-count loops)
- SLCC (Separate lines for commands)
- INT_TRI (Internal tristate objects)
- COMMENT_END_STMNTS (Comment end statements)
- MCA (Missing case item assignment)
- MIA (Missing if assignment)
- LATCH_CREATED (Report on every latch created or inferred)
- COMMENT_NET_DEC (Comment net declarations)
- STATE_VAR_NAME (Run name analysis on FSM state variable names)
- COMMENT_PORT_DEC (Comment port declarations)
- RST (All flip-flops resettable)
- RESET_POLARITY (Check reset polarity consistency)
- FOREIGN_LANGUAGE_KWD (Report usage of Foreign Language

RTL Analysis – FSM Checks



View - or1200_except.v

```
575                         FLU5: begin
576                         // To Fix Uncomment Section below
577 /*
578                         if (!if_stall && !id_freeze) begin
579                                   state <= #1 IDLE;
580                                   except_type <= #1 `OR1200_EXCEPT_NONE;
581                                   extend_flush_last <= #1 1'b0;
582                                 end
583                                 else state <= #1 FLU6;
584
585 */
586                                 state <= #1 FLU5;  // To fix comment it
587
588                         end
589
590                         FLU6: begin
591                                 if (!if_stall && !id_freeze) begin
592                                   state <= #1 IDLE;
593                                   except_type <= #1 `OR1200_EXCEPT_NONE;
594                                   extend_flush_last <= #1 1'b0;
595                                 end
596 //                              else state <= #1 FLU6;   // To fix uncomment it
597                               end
598                 endcase
599         end
600 end
601
602 endmodule
```

RTL Analysis – Long Path Analysis

Paths

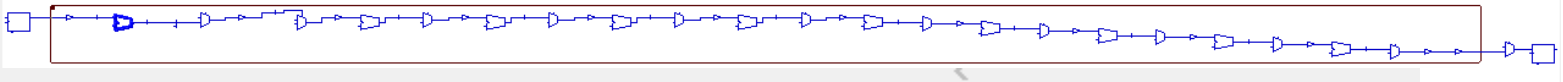
Path Type	Length	From	To
DFF To DFF >=			

DFF to DFF: cordic.angle_s -> cordic.sinout_s: 22
DFF to DFF: cordic.angle_s -> cordic.cosout_s: 22
DFF to DFF: cordic.sinout_s -> cordic.tanout_s: 5
DFF to DFF: cordic.cosout_s -> cordic.tanout_s: 5
DFF to DFF: cordic.state -> cordic.sinout_s: 4
DFF to DFF: cordic.state -> cordic.cosout_s: 4
DFF to DFF: cordic.state -> cordic.angle_s: 4

Path:

cordic.angle_s
cordic.U1.buf_5
cordic.U1.z0_s_11_3
cordic.U1.add_13
cordic.U1.add_13_8_0
cordic.U1.concat_0_11_0
cordic.U1.mux_14
cordic.U1.buf_15

Search: All

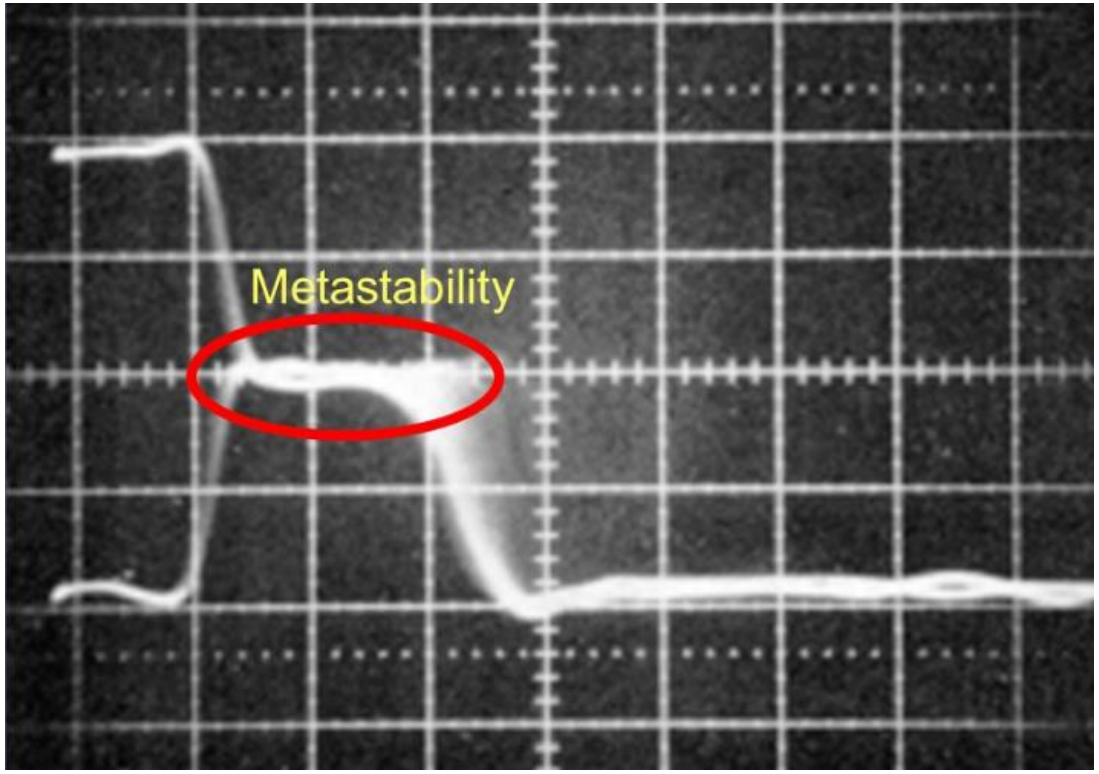


Enable Cross Probing to RTL?

View - cordicp_s.v

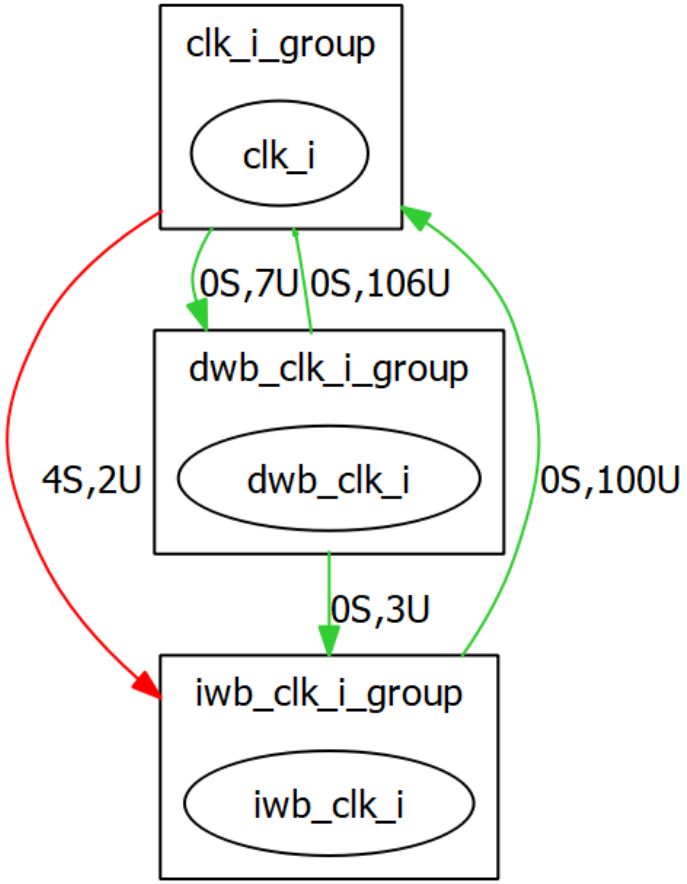
```
58  wire[11:0] z7_s;
59  wire[11:0] z8_s;
60  wire[11:0] z9_s;
61  wire[11:0] z10_s;
62
63  assign y0_s = {12{1'b0}} ;
64  assign z0_s = angle ;
65  assign x1_s = xinit_c ;
66  assign y1_s = ((z0_s[11]) == 1'b0) ? x1_s : (~x1_s) ;
67  assign z1_s = ((z0_s[11]) == 1'b0) ? z0_s - atan0_s : z0_s + atan0_s ;
68  assign x2_s = ((z1_s[11]) == 1'b0) ? x1_s - ({y1_s[11], y1_s[11], y1_s[WIDTH - 2:1]}) : x
69  assign y2_s = ((z1_s[11]) == 1'b0) ? y1_s + ({x1_s[11], x1_s[11], x1_s[WIDTH - 2:1]}) : y
70  assign z2_s = ((z1_s[11]) == 1'b0) ? z1_s - atan1_s : z1_s + atan1_s ;
71  assign x3_s = ((z2_s[11]) == 1'b0) ? x2_s - ({y2_s[11], y2_s[11], y2_s[11], y2_s[WIDTH -
72  assign y3_s = ((z2_s[11]) == 1'b0) ? y2_s + ({x2_s[11], x2_s[11], x2_s[11], x2_s[WIDTH -
73  assign z3_s = ((z2_s[11]) == 1'b0) ? z2_s - atan2_s : z2_s + atan2_s ;
74  assign x4_s = ((z3_s[11]) == 1'b0) ? x3_s - ({y3_s[11], y3_s[11], y3_s[11], y3_s[11], y3_
75  assign y4_s = ((z3_s[11]) == 1'b0) ? y3_s + ({x3_s[11], x3_s[11], x3_s[11], x3_s[11], x3_
76  assign z4_s = ((z3_s[11]) == 1'b0) ? z3_s - atan3_s : z3_s + atan3_s ;
```

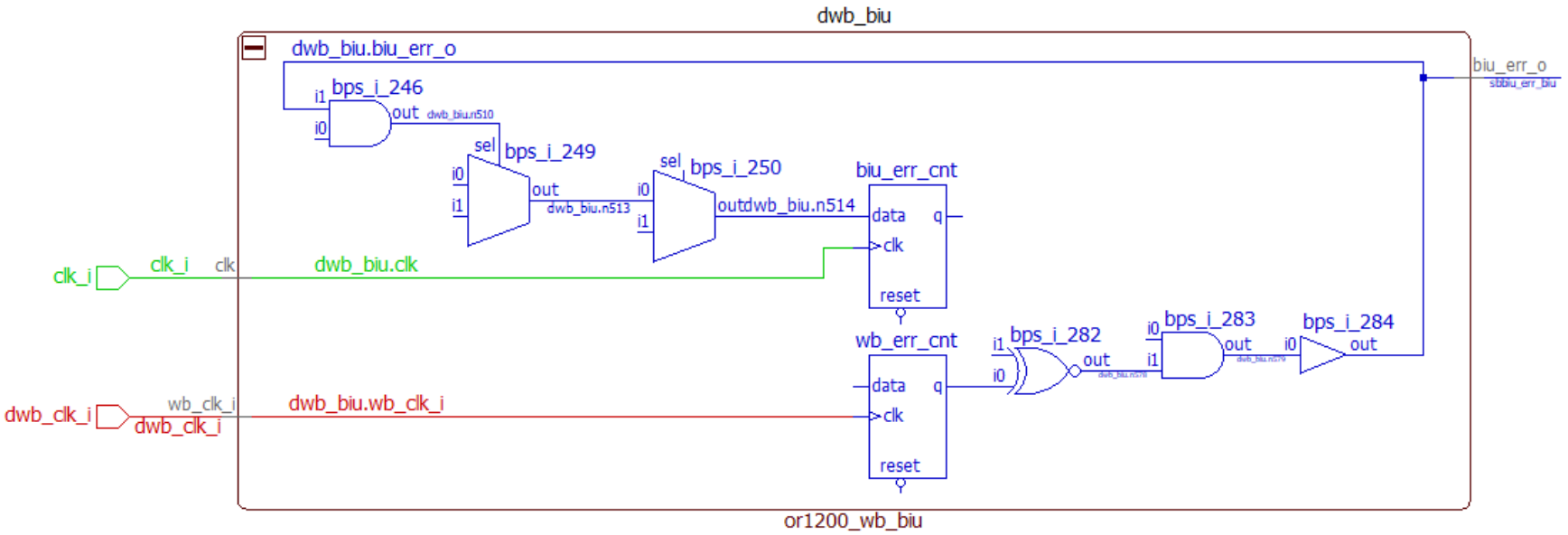
RTL Analysis



2016 Industry research reports Clocking/CDC Errors are the #2 cause of respins

RTL Analysis - CDC





Questions