2025 DESIGN AND VERIFICATION DVCON CONFERENCE AND EXHIBITION CHINA

accellera SYSTEMS INITIATIVE

UNISOC

UNISOC

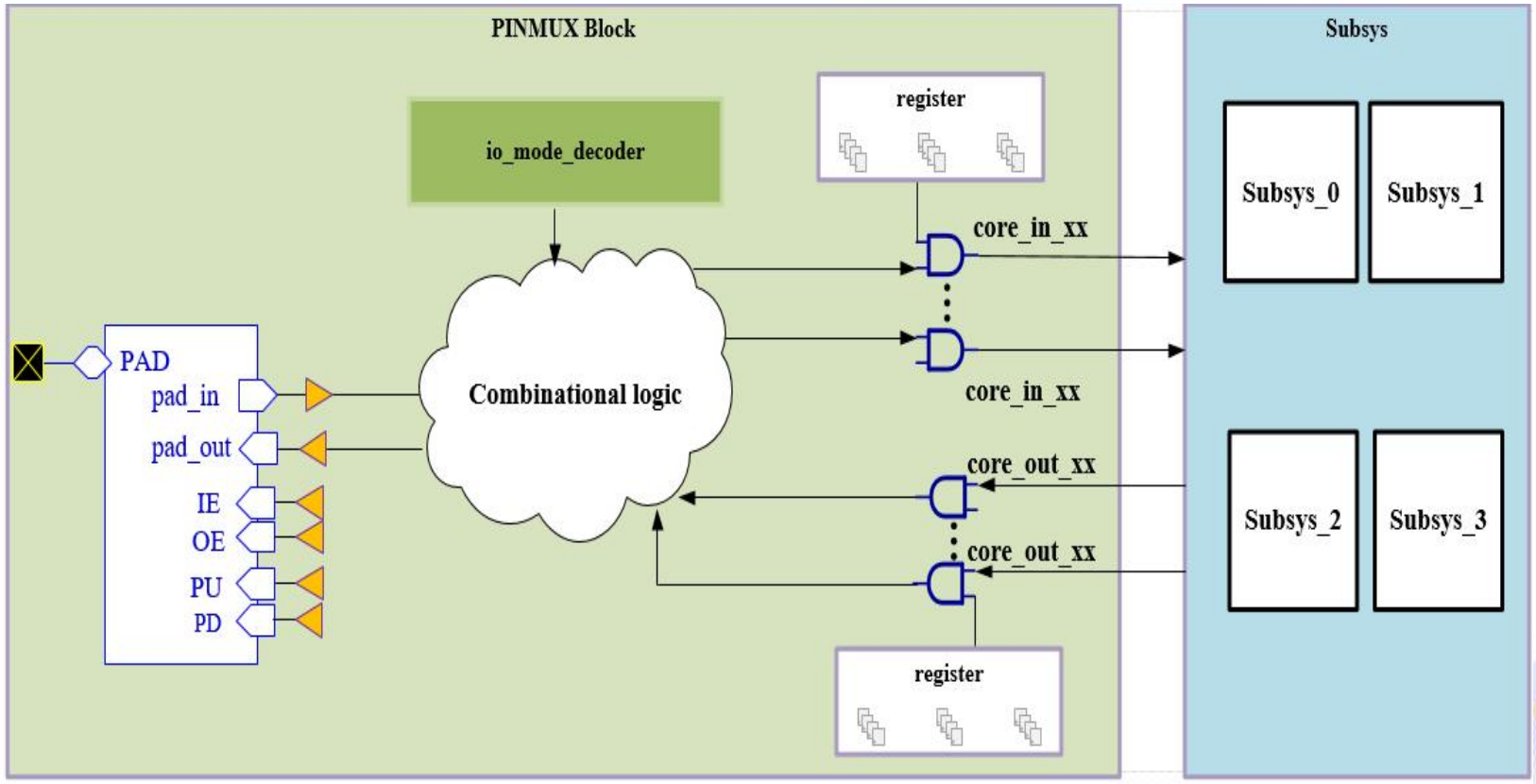
Formal Verification Sign-off Methodology for Pin Multiplexing Based on Jasper CONN

Xubao.Wang (xubao.wang@unisoc.com)
Chengzhe.Li (chengzhe.li@unisoc.com)
Xufeng.Zhang (xufeng.zhang@unisoc.com)

16 Apr 2025 上海 SHANGHAI

Catalogue

Pinmux Design Architecture



Design Feature:

① PAD cell: instance pad model

② Mode decoder: various mode confirmed

③ Combinational logic: logic selection with basic logic gates

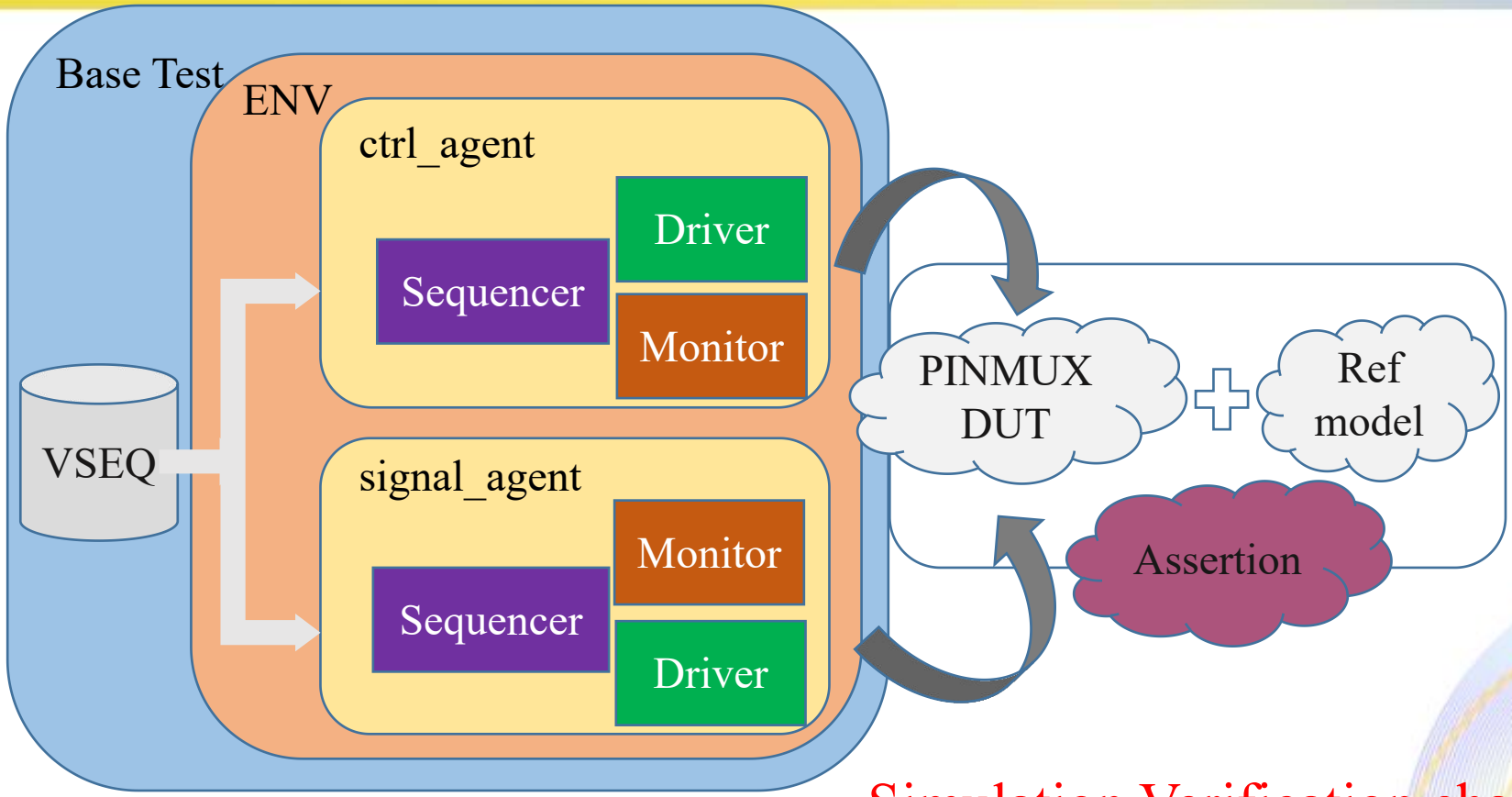
④ Registers: control of pull-up/pull-down, sleep mode, and ie/oe enable

Verification plan:

a) Block Level: Check the connection between PAD and core inside PINMUX

b) Soc Level: the connections between PINMUX and subsystems within the SoC

Simulation Verification Solution

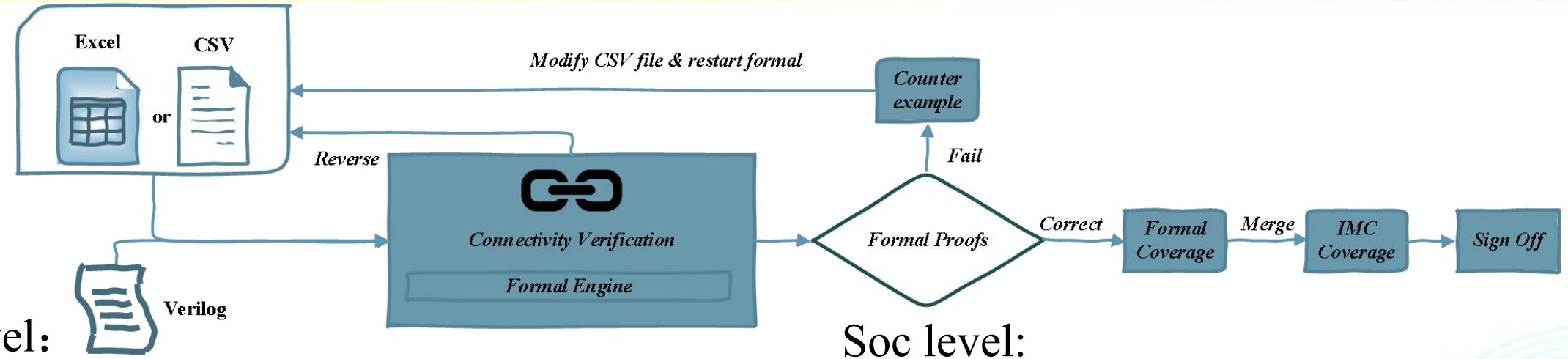


I. ctrl_agent: configure the PINMUX module to a specific operational mode
II. signal_agent: generate data transactions;
III. Assertion: checks the reference signal and DUT output;(automatically generated by python)

Simulation Verification challenges:

1. A single simulation cannot exhaustively cover all incentives，need a large number of random test cases;
2. Coverage collected is time-consuming and inefficient with random transactions (now simAI can help)

Formal Verification Solution



Block level：

- Excel sheet or a comma-separated-value (CSV) file; The generation of csv files is fully automated by python script;
- Register-Transfer Level (RTL)

Soc level:

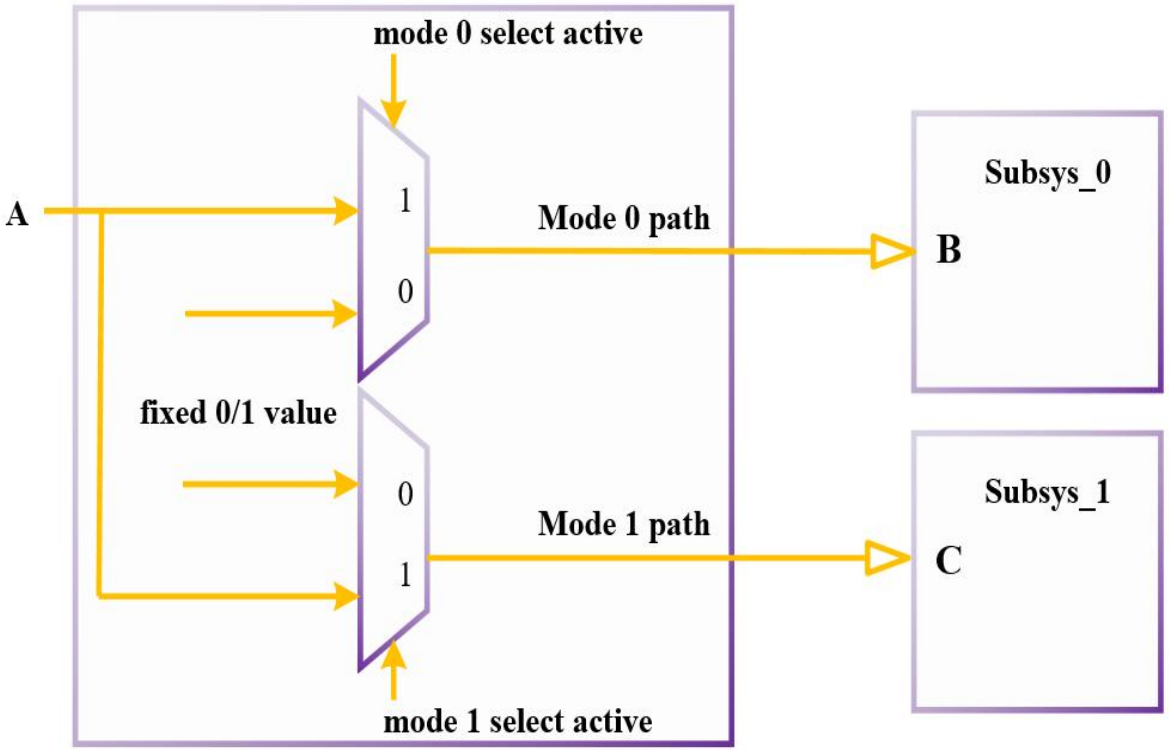
- wire reverse extraction from analyzed and compiled RTL, directly output an Excel file or csv for review

```
check_conn –reverse –src {pinmux} –dest {subsys_0} –complexity (straightforward | simple | conditional | complex) –save_as reverse.csv
```

1 NAME, SRC BLOCK, SRC SIGNAL, DEST BLOCK, DEST SIGNAL
2 CONNECTION, a_0,chip_top,GPIO_xx, IO_MUX_CORE, core_in_xx
3 ,COND_EXPR, ptest==1&&PAD_xx==0

(ptest==1&&PAD_XX==0) |-> (chip_top.GPIO_xx == IO_MUX_CORE.core_in_xx)

Pinmux Verification Completeness

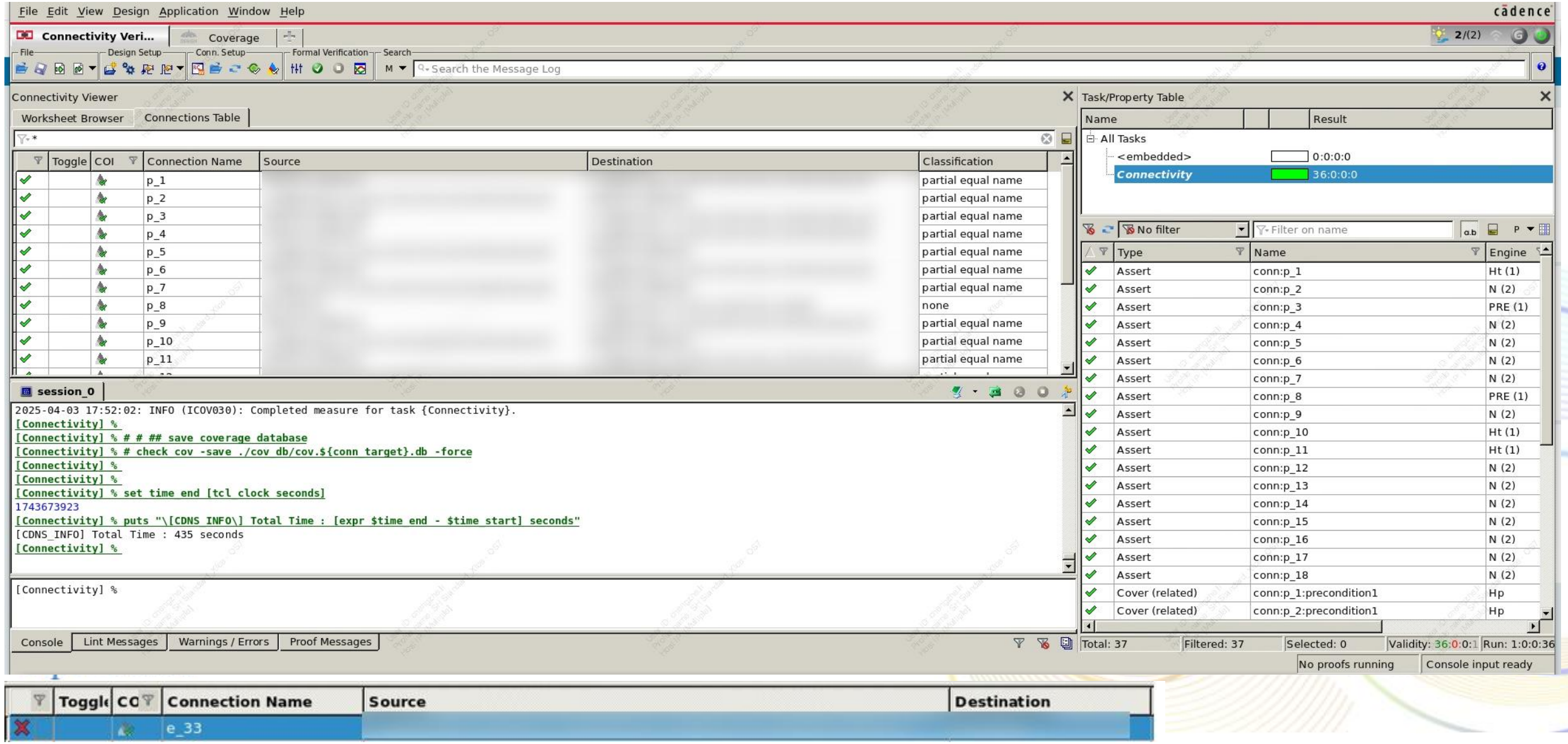


a single PAD has input functions across multiple modes, the CSV for one input function must reflect checks for fixed values of signals under other modes

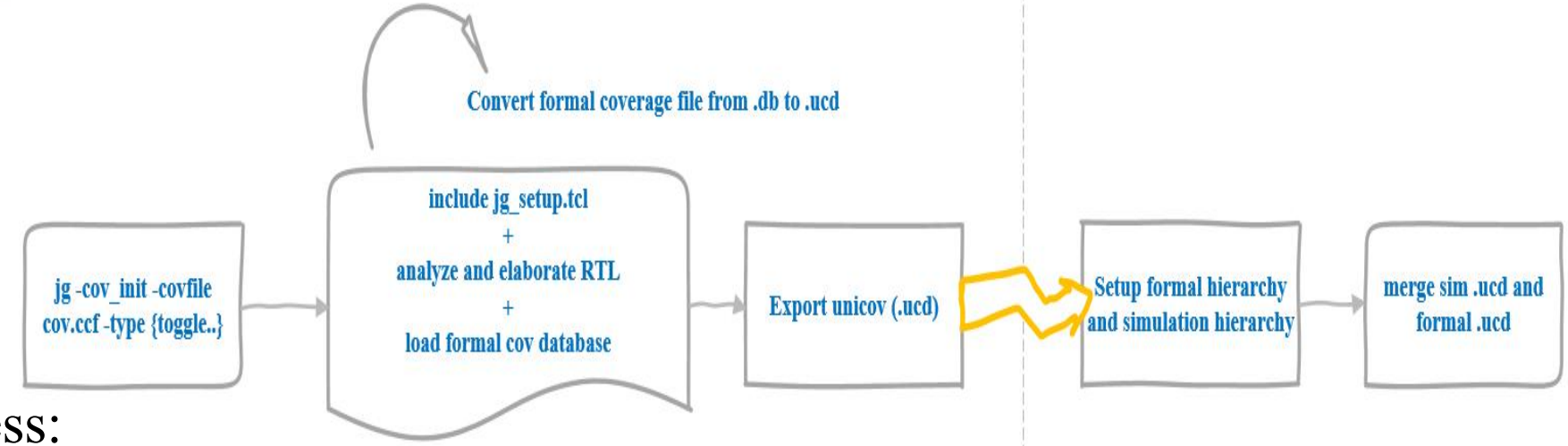
1 NAME, SRC BLOCK, SRC SIGNAL, DEST BLOCK, DEST SIGNAL
2 CONNECTION, b_0,chip_top, A, Subsys_0, B
3 ,COND_EXPR, mode 0 = 1 && mode 1=0

1 NAME, SRC BLOCK, SRC SIGNAL, DEST BLOCK, DEST SIGNAL
2 CONNECTION, b_1,chip_top,A, Subsys_1, 0
3 ,COND_EXPR, mode 0 = 1 && mode 1=0

Formal Prove Result

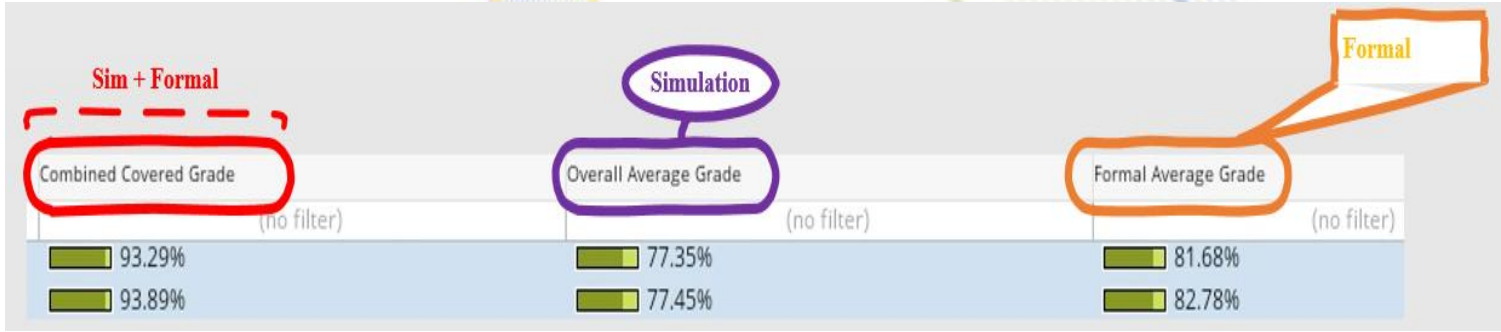


Coverage Merge Flow

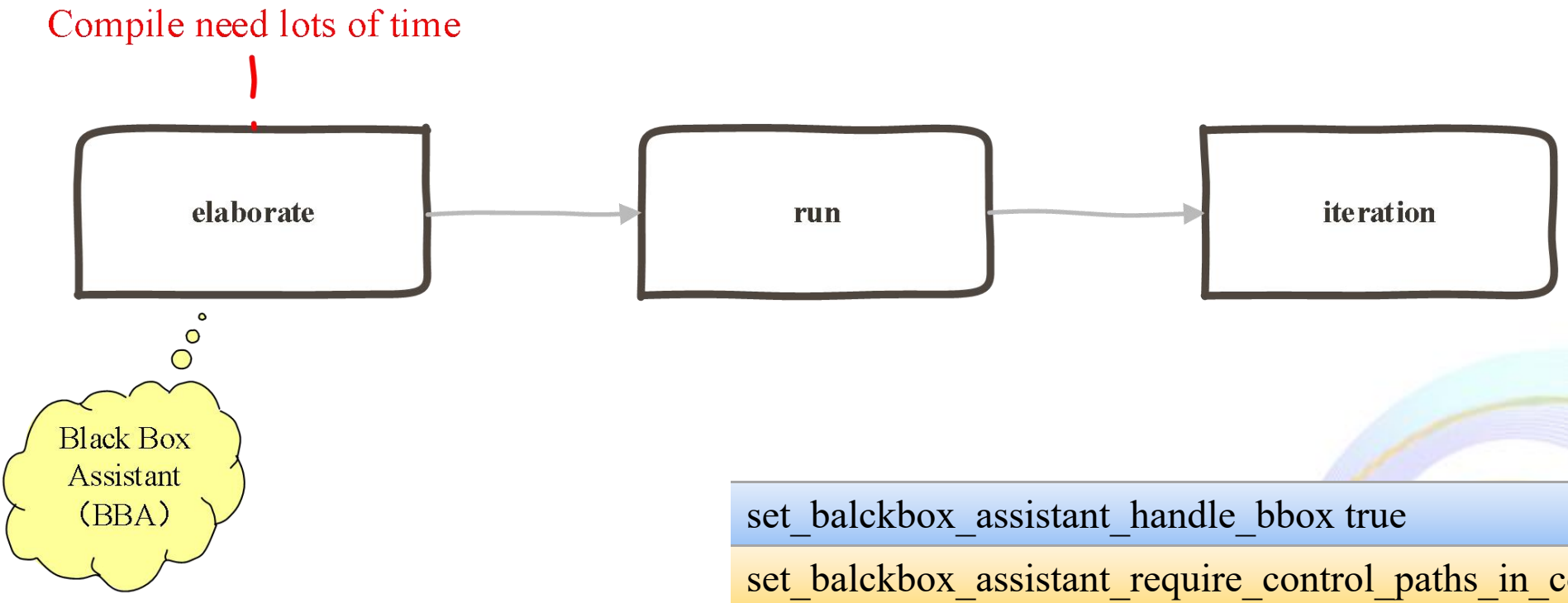


Merge process:

1. First, convert formal's .db to .ucd; (database –export_unicov)
2. Specify source (formal_top) and target (sim_top)
3. merge /sim_cov/cov /formal/cov/



Difficulties and Breakthrough Achieved



1. BBA

automatically add modules that are not related to the source signal and DEST signal path to the black box list

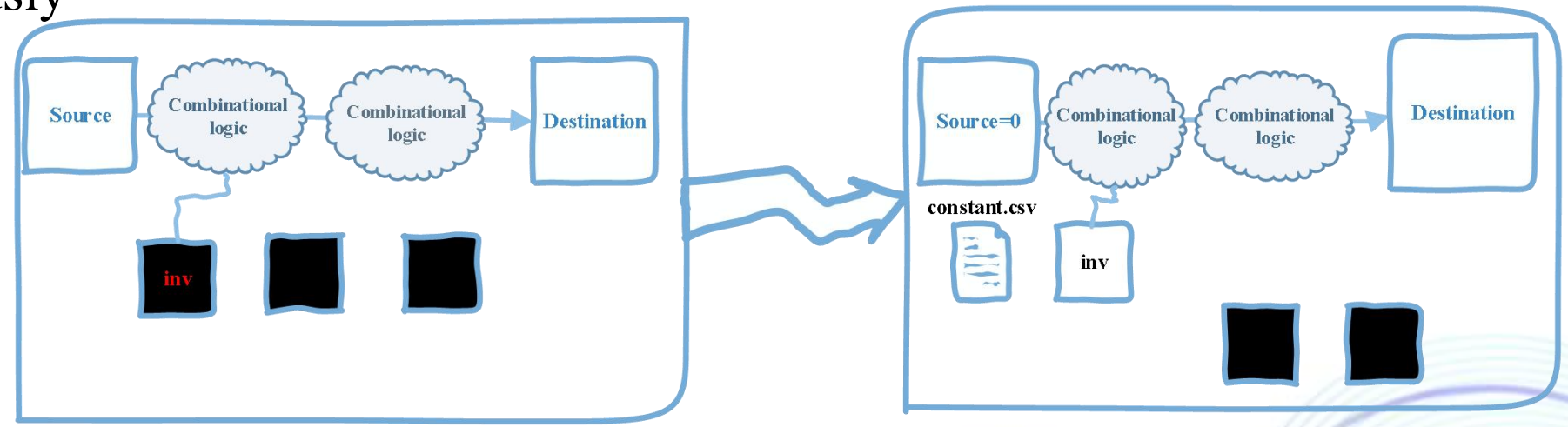
set_balckbox_assistant_handle_bbox true
set_balckbox_assistant_require_control_paths_in_connections true
set_balckbox_assistant_require_clock_paths_in_connections true
blackbox_assistant –config –connectivity_map jg_${conn_target}.csv –export [get_proj_dir]/.bbalist

Difficulties and Breakthrough Achieved

BBA will cause the previously passed assertion to fail

```
#Jasper bba proc
include jg_proc.tcl
#Jasper setup
include jg_setup.tcl
#cov init
include jg_cov_init.tcl
#analyze + elaborate design
include jg_read_design.tcl
#load conn CSV file
check_conn –load ${conn_target}.csv
```

enable_bba ${conn_target}.csv
elaborate …



The solution involves setting the input source signal as a constant during the verification phase

1 NAME, SRC BLOCK, SRC SIGNAL, DEST BLOCK, DEST SIGNAL
2 CONNECTION, a_0,chip_top,GPIO_xx, IO_MUX_CORE, core_in_xx
3 ,COND_EXPR, ptest==1&&PAD_xx==0

${conn_target}.csv

1 NAME, SRC BLOCK, SRC SIGNAL, DEST BLOCK, DEST SIGNAL
2 CONNECTION, a_0,chip_top,0, IO_MUX_CORE, core_in_xx
3 ,COND_EXPR, ptest==1&&PAD_xx==0

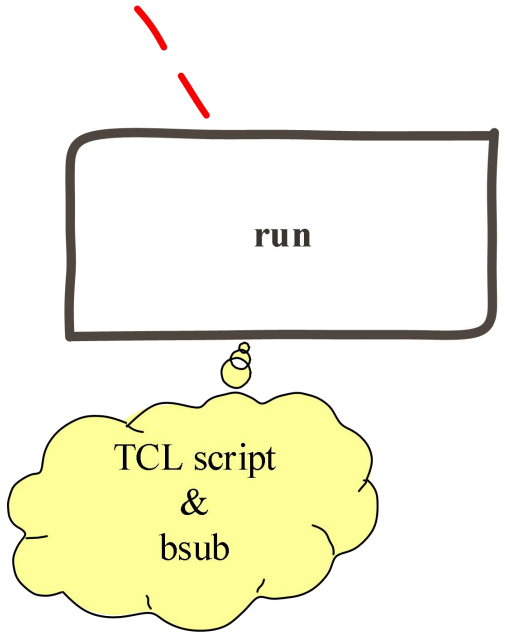
${conn_target}.constant.csv

Difficulties and Breakthrough Achieved

2. Parallel verification

a template tcl and leverage the bsub command for execution;

Manually run sequentially in Linux terminals

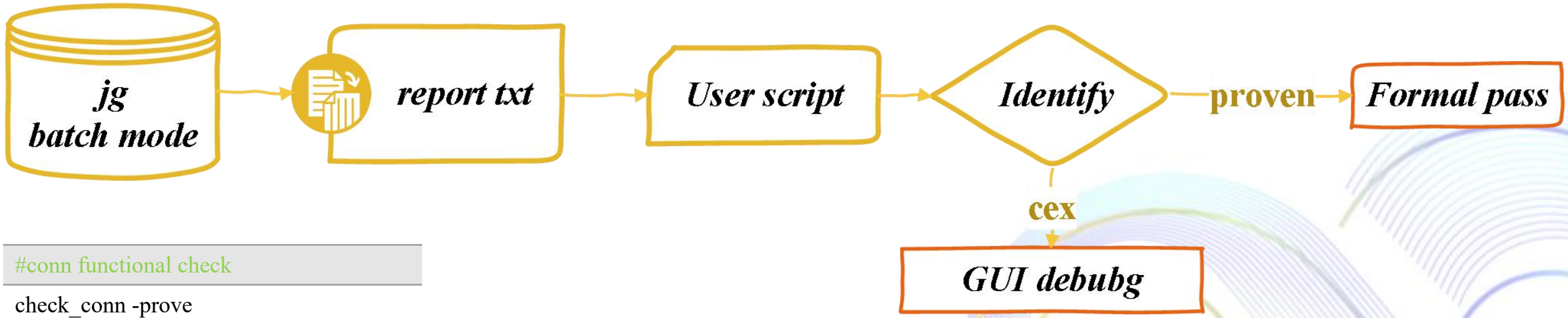



```
clear –all
set time_start [tcl_clock seconds]
set conn target CONN TARGET
#Jasper setup
include jg_setup.tcl
#cov init
include jg_cov_init.tcl
#analyze + elaborate design
include jg_read_design.tcl
#load conn CSV file
check_conn -load ./csv/${conn_target}.csv
#clock and reset
clock -none
reset -none
#include conn constraints
include ./cons/cons.${conn_target}.tcl
#conn structural check
check_conn -validate
#conn functional check
check_conn -prove
#switch tab to Connectivity
check_cov -configure_gui -task Connectivity
#coverage measurement
check_cov -measure -task Connectivity
#save coverage database
check_cov -save ./cov_db/cov.${conn_target}.db -force
set time end [tcl clock seconds]
```

Difficulties and Breakthrough Achieved

3. accelerate the convergence of results

identify the formal verification tool has fully proven the design in batch mode

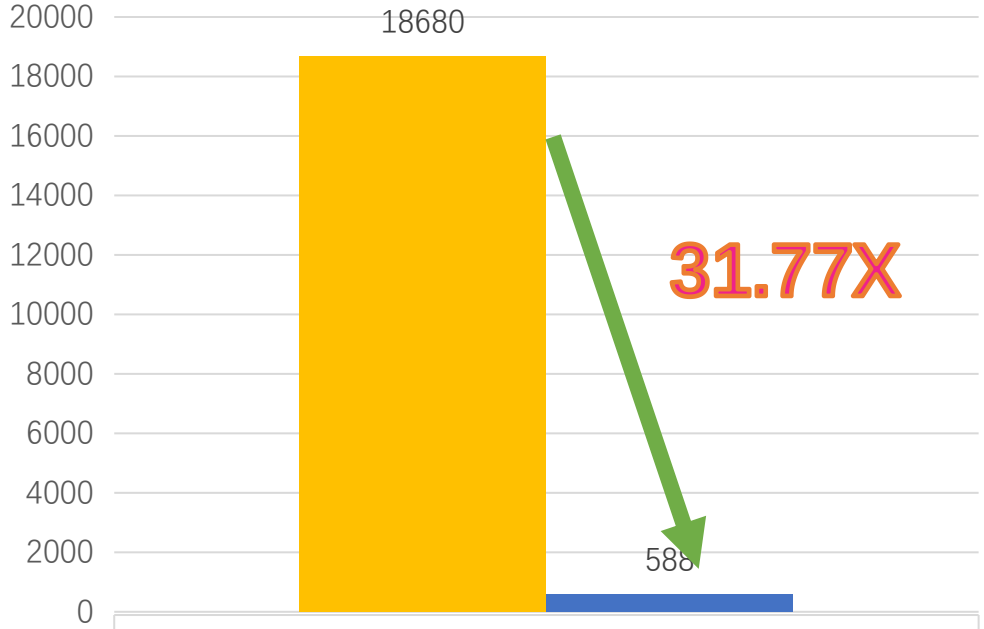



```
#conn functional check
check_conn -prove
#report verification result
check_conn –report –file xx.txt -force
```

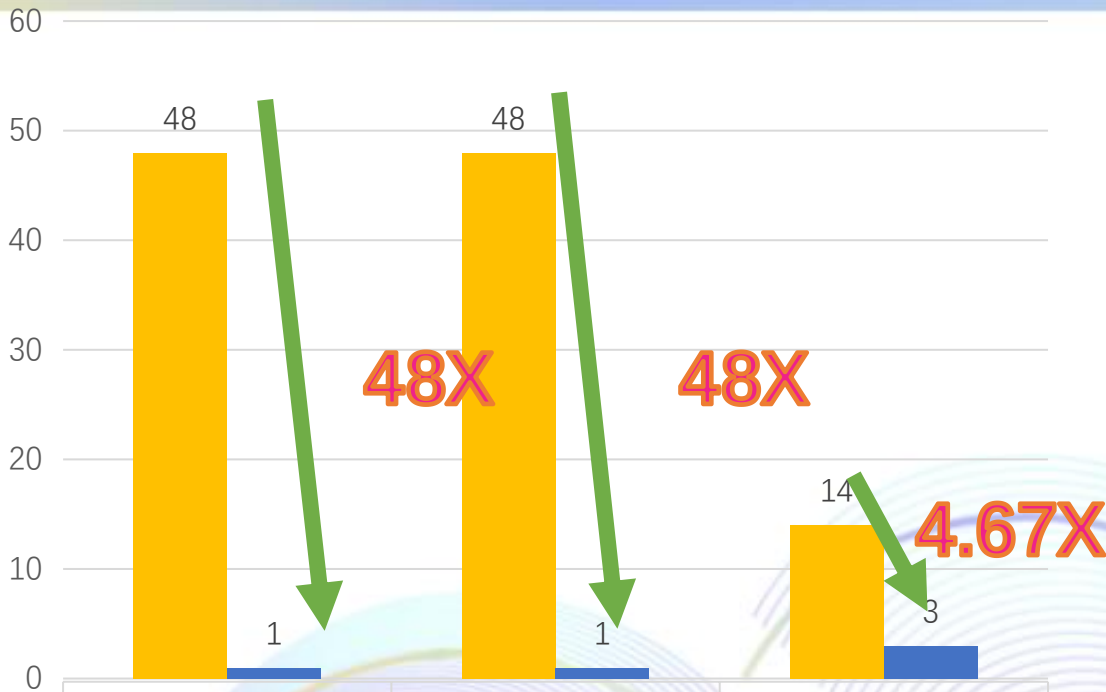
```
1 ================================================================================================================================================
2                                                          CONNECTIVITY REPORT
3 ================================================================================================================================================
4 Connection Name  | Source Block | Source          | Destination Block | Destination      | Status      | Toggle        | COI                                | Classification
5 ------------------------------------------------------------------------------------------------------------------------------------------------
6  o_0                                                                                        proven        not-generated   Source is inside destination's COI   none
7  o_10                                                                                       cex           not-generated   Source is inside destination's COI   none
```

Formal & Simulation Comparison

Efficiency improvement in formal verification



	Lines of code (line)
UVM verification	18680
Formal verification	588



	Time for environment setup (hour)	One-time verification period (hour)	Verification completion time (day)
UVM verification	48	48	14
Formal verification	1	1	3

- Formal streamlines the 10,000+ lines of code to just 588 lines;
- Time required for environment setup and a single formal verification within an hour;
- Formal verification complete the whole verification in merely three days.

Summary

Formal Verification

incentives are more comprehensive

easier iteration

coverage merge supported

better project inheritance

Faster

Stronger