Sequential Equivalency Checking with Cadence JasperGold

NYCU-EE IC LAB FALL-2023

Lecturer:Hsien-Chi Peng

Outline

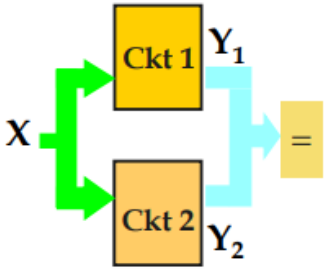
- ✓ Section 1. Introduction to SEC
- ✓ Section2. JasperGold – SEC Setup
- ✓ Section3. JasperGold – Proof & Debug
- ✓ Section4. JasperGold – Clock Gating Verification

Outline

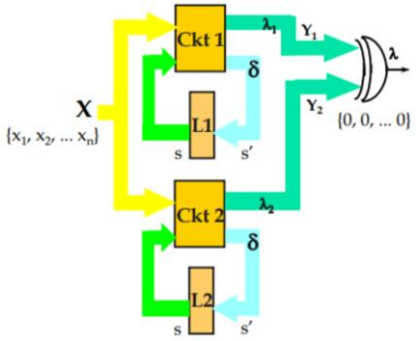
- ✓ **Section 1. Introduction to SEC**
- ✓ Section2. JasperGold – SEC Setup
- ✓ Section3. JasperGold – Proof & Debug
- ✓ Section4. JasperGold – Clock Gating Verification

Equivalence Checking - EC

- ✓ **Problem**
  - **Given one RTL design (specification) and another optimized RTL/GL design (implementation)**
  - **Do they implement the same set of behavior?**
  - **Functional Verification of the change requires a lot of effort!**
    - Full regression required even minor changes
- ✓ **Solution**
  - **EC guarantees equivalence**



CEC



SEC

- ✓ **Equivalence checking**
  - **Combinational equivalence checking (CEC)**
    - Output depend only on present inputs
  - **Sequential equivalence checking (SEC)**
    - Output depend on present inputs as well as past sequence of inputs

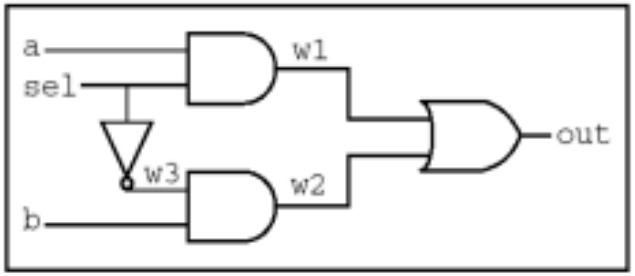
Example: Equivalence Checking

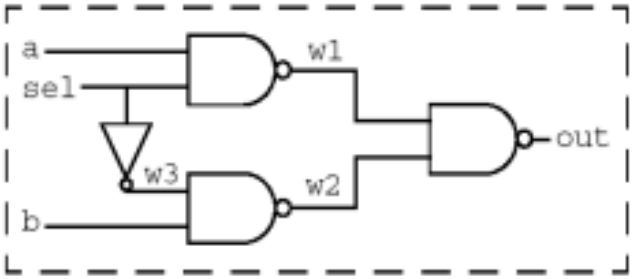
```
out = sel ? a : b ;
```

```
always @ (sell or a or b)
    if (sel) out = a;
    else out = b;
```

```
module des1 (out,a,sel,b)
    output out;
    input a,sel,b;
    wire w1,w2,w3,
      and u1(w1,a,sel)
      and u2(w2,w3,b);
      not u3(w3,sel);
      or u4(out,w1,w2)
endmodule
```

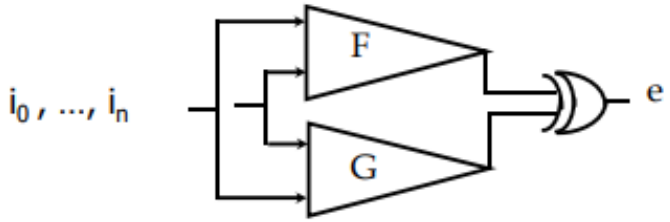
```
module des1 (out,a,sel,b)
    output out;
    input a,sel,b;
    wire w1,w2,w3,
      nand u1(w1,a,sel)
      nand u2(w2,w3,b);
      not u3(w3,sel);
      nand u4(out,w1,w2)
endmodule
```





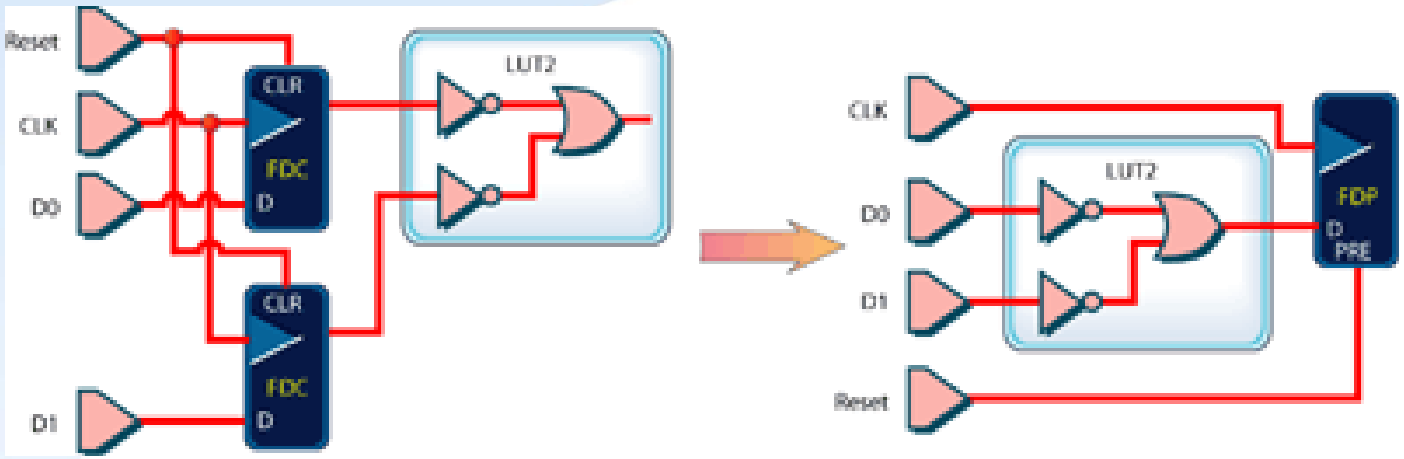
CEC/LEC vs. SEC

- ✓ **Combinational Equivalence Checking (CEC, LEC)**
  - Same functional behavior at all external ports and internal state element?
  - Requires state-matching design
  - Performs EC only on the remaining logical cones



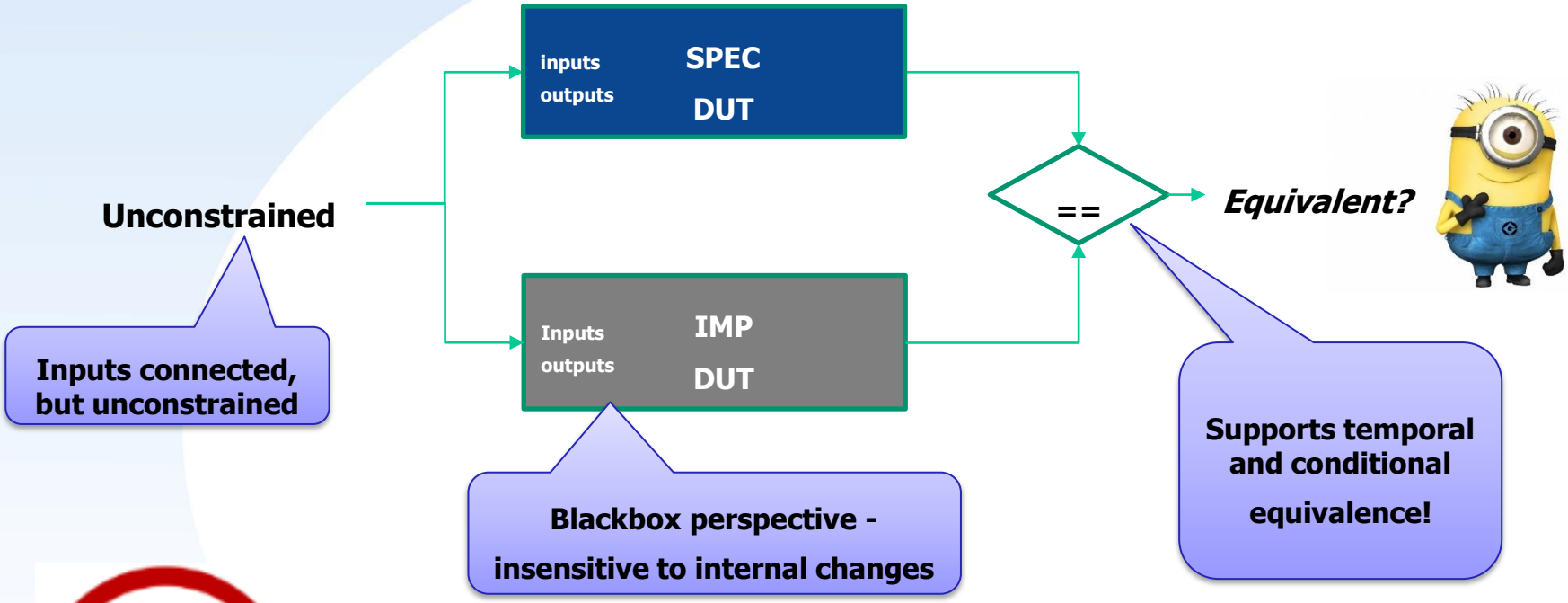
- ✓ **Sequential Equivalence Checking (SEC)**
  - Same functional behavior at all external ports?
  - Applies to state-matching and non-state-matching designs
  - Ignores internal sequential differences

Limitations of traditional LEC



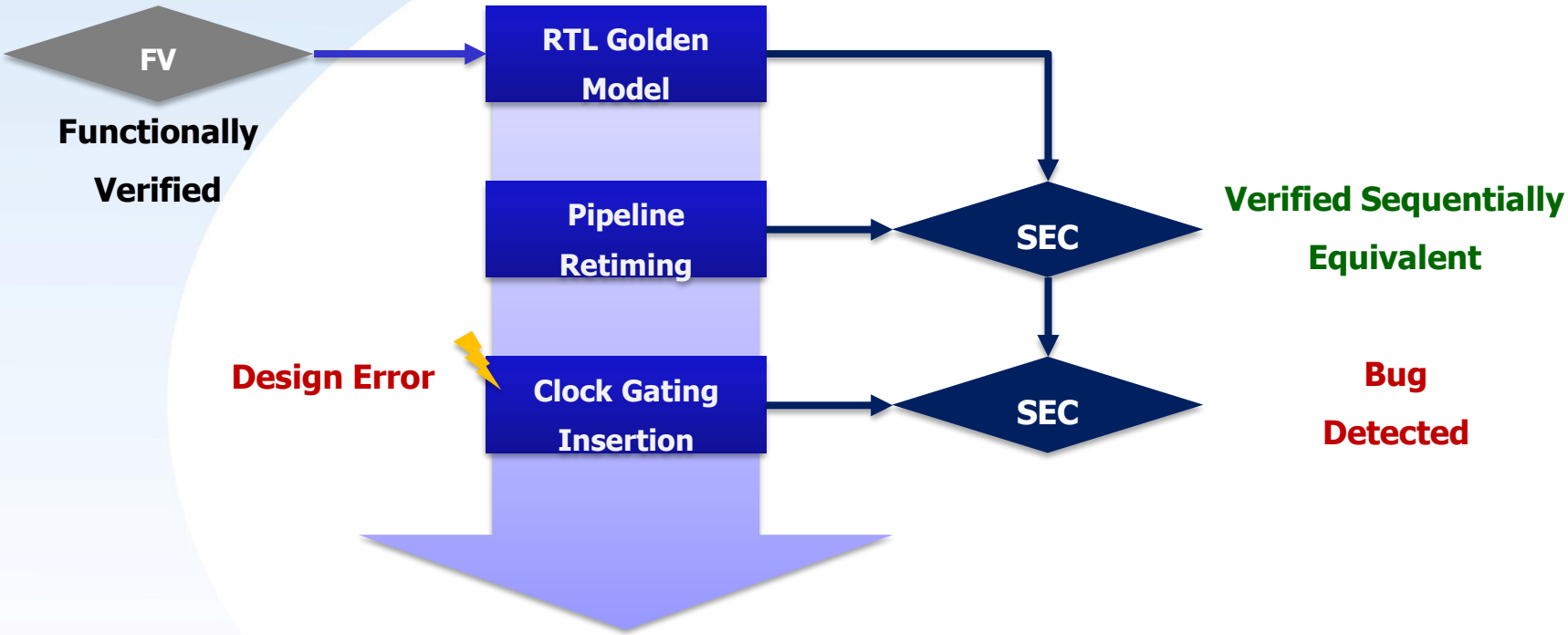
- ✓ **Example is sequentially equivalent, but…**
  - Sates cannot be matched
  - Internal logic cones are different
- ✓ **CEC → NOT Equivalent !**
- ✓ **SEC → Equivalent !**

SEC Technique – Miter Construction



Replacing weeks of simulation regression runs!

Example RTL Refinement Flow with SEC



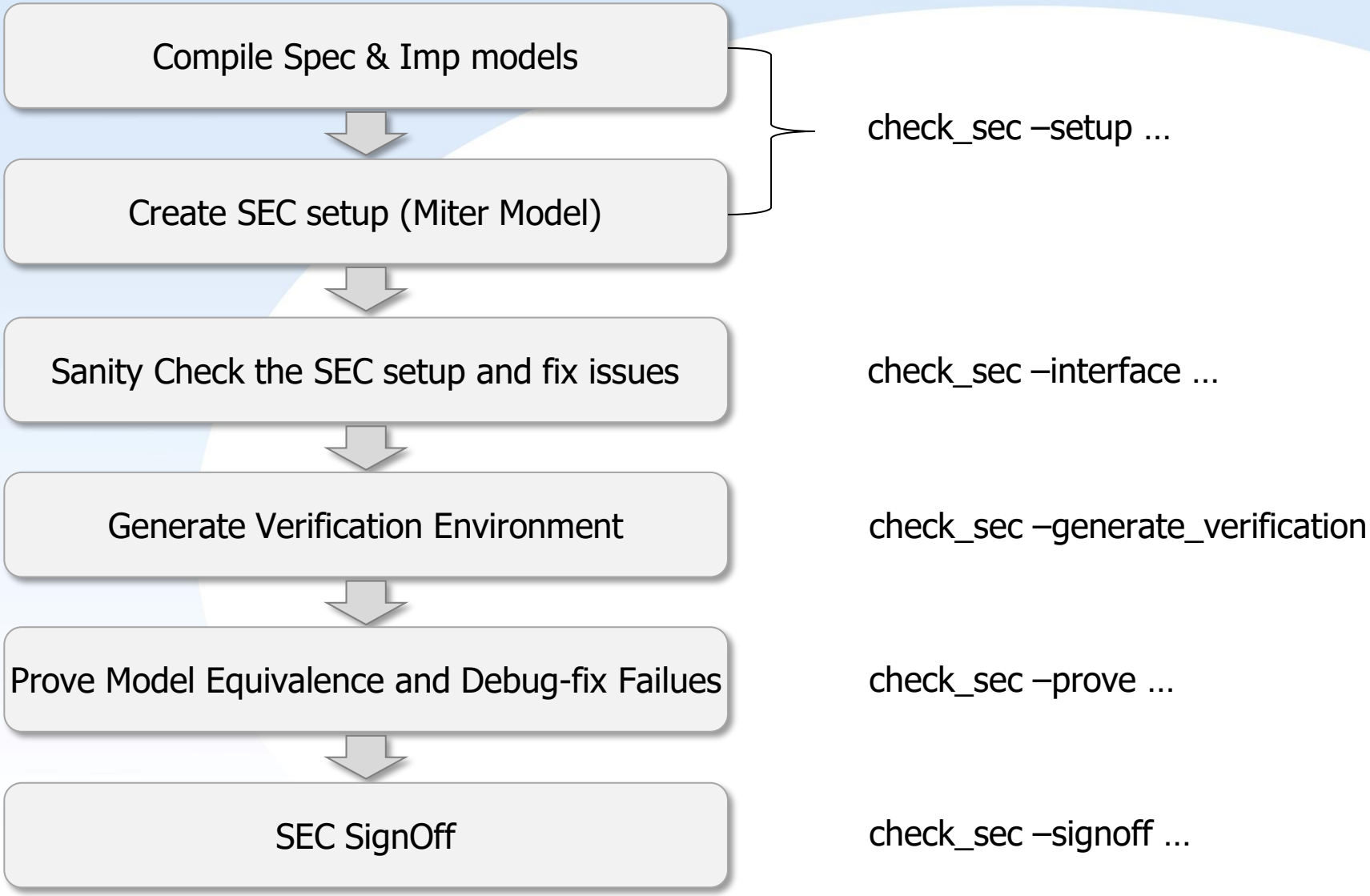
SEC Adoption Hints

- ✓ **Recommend incremental flow, and position SEC as the main validation tool**
  - Start from golden refine for power, area, speed. Each step verified with SEC
- ✓ **Understand the similarity, know what is the difference**
  - "Similarity" is important – not design size: Flop count isn't meaningful in discovery
  - Know the transformation, this helps splitting into sub transformations for SPEC
- ✓ **Clock gating is sweet spot**
  - If mapping constraints may make it involved
  - Almost all customer started with this
- ✓ **Re-partitioning and logic optimization is sweet spot**
  - High similarity. Seek in "implementation" teams for this transformation step

Outline

- ✓ Section 1. Introduction to SEC
- ✓ **Section2. JasperGold – SEC Setup**
- ✓ Section3. JasperGold – Proof & Debug
- ✓ Section4. JasperGold – Clock Gating Verification

SEC flow at a glance



SEC Mapping GUI

Signal type filters

Mapping pairs

Design browsers for both SPEC and IMP

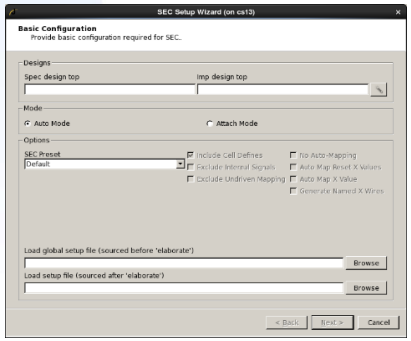
Property table

SEC App – GUI Flow

✓ SEC Setup

***1. Setup* Wizard**

- **Compile source files**
- **Elaborate SPEC and IMP**
- **Load UPF files**
- **Specify auto mapping options**



2. Interface Check

- **Performs completeness analysis of your current mapping (Optional)**

3. Generate Verification

- **Create assertions and assumptions for mapping pairs (Required)**

4. Clock Tree Analysis

- **Run clock tree analysis (Optional)**

SEC App – GUI Flow

✓ SEC Proof and Analysis

5. Proof

- **Run formal engines**

6. Verification Completeness

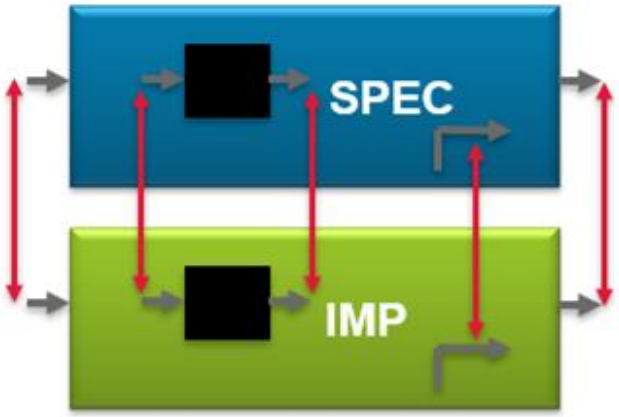
- **Specify filtering and waiving**
- **Calculate Sign-Off report**

7. Clear

- **Clear all or specific data (optional)**

What happens during Setup

- ✓ **Build both designs**
- ✓ **Connect them into a miter model**
- ✓ **Apply auto mapping**
  - Primary/DUT inputs (Connection)
  - Primary/DUT outputs (Check)
  - Black-box outputs (Connection)
  - Black-box inputs (Check)
  - Undriven or dangling signals (Connection)
  - Stopats from both designs (Both)
  - Uninitialized state elements initial value (Initial value connection)
  - Blackbox: MPRAM ports, dividers / multipliers



SEC Setup Example Tcl Command

```
check_sec –setup  \
        -spec top_spec  \
        -spec_analyze { -sva top_spec.v }  \
        -spec_analyze { -verilog lib1.v }  \
        -spec_analyze { -vhdl lib1.vhd }  \
        -spec_elaborate_opts { -bbox_m module_a }  \
        -imp top_imp  \
        -imp analyze { -sva top_imp.v }  \
        -imp analyze { -verilog lib.v }  \
        -imp analyze { -vhdl lib1.vhd }  \
        -imp elaborate_opts { -bbox_m module_b }
```

Golden Design (spec)

Revised Design (imp)

SEC Interface Check

- ✓ **SEC allow for automatic mapping of interface signals**

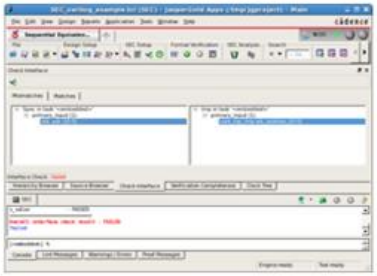
  check_sec –setup … (includes implicit auto mapping)

  - Input pairs will be driven to be equal → Connection
  - Output pairs will be asserted to be equal → Check

- ✓ **Problem**
  - Missing input mapping may cause false failures on outputs
- ✓ **Solution**
  - Interface check to confirm mapping is complete

Interface check helps to avoid false failures:

→ **check_sec – interface …** →



SEC Interface Check – Mapping Unmapped Signals in GUI

Switch to „Check Interface" tab and display unmapped interface signals

Signal expressions are pre-filled.

Select SPEC signals to map

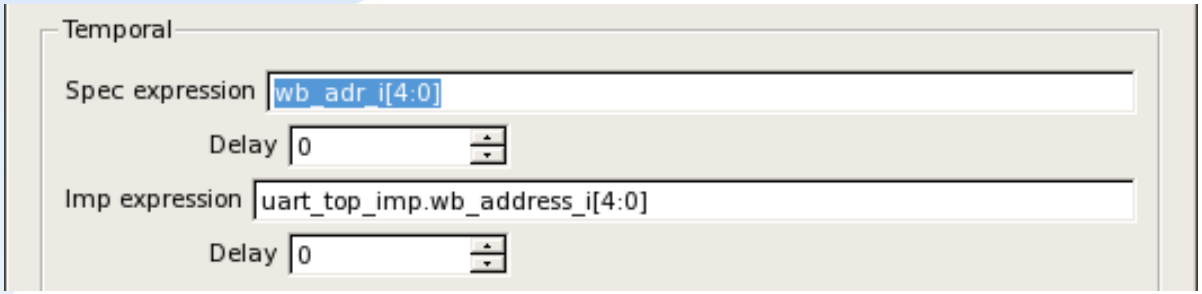
Select IMP signals to map

Right-Mouse-Button Menu „Quick Map" - Open Advanced Mapping Dialog

Create new mapping pair

Manual Mapping with Tcl Commands

- ✓ **Simple mapping**




```
check_sec -map -spec {{wb_adr_i[4:0]}} -imp {{uart_top_imp.wb_address_i[4:0]}}
        -global ...
```

- ✓ **Advanced mapping using regular expressions**

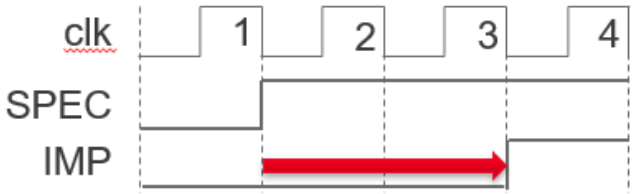
Substitution

```
check_sec -map -spec {wb_adr(.*)} -imp {top_imp.wb_address$1$} –regexp –verbose
```

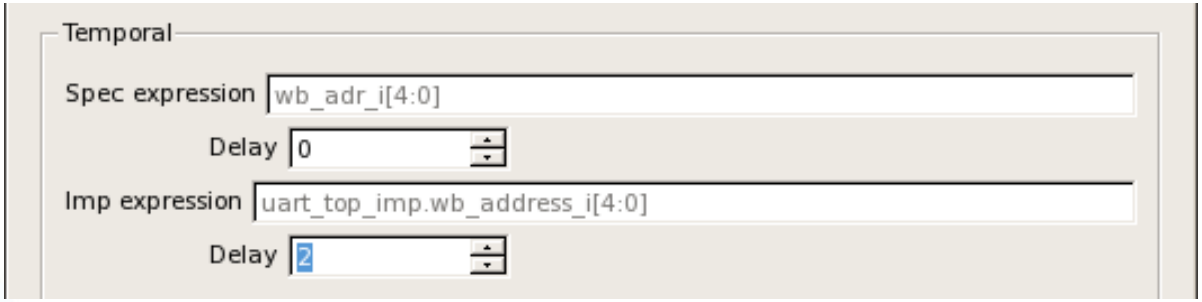
See http://www.tcl.tk/man/tcl8.5/TclCmd/re_syntax.htm for details

Manual Mapping with Delay

- ✓ **SPEC and IMP mapping points are equivalent, but with 2 cycles delay**
  - Used for validating Pipeline Retiming for example



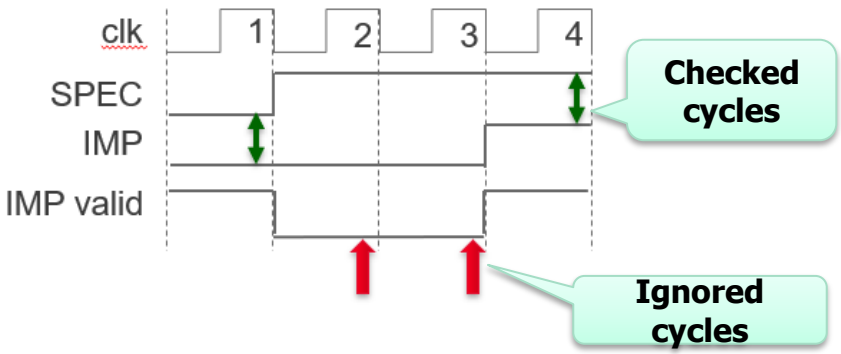
- ✓ **Specify mapping with delay**



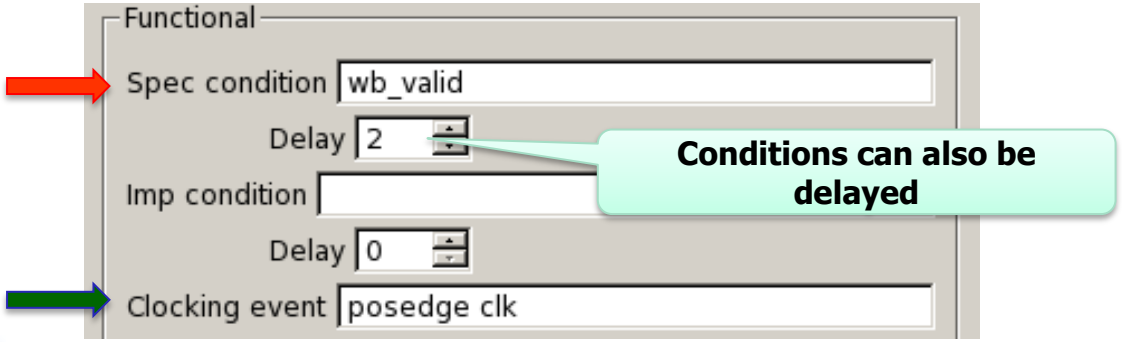
check_sec -map -spec … -imp … -imp_delay 2

Manual Mapping with Conditions

- ✓ **SPEC and IMP mapping points are equivalent only under a condition wb_valid**
  - Used for protocols with handshake



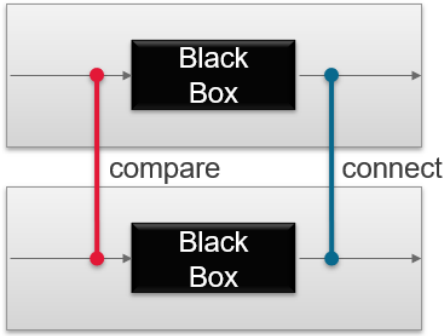
- ✓ **Specify mapping with condition and clock**




```
check_sec -map -spec ... -imp ... -spec_condition wb_valid \
                                  -clock {posedge clk}
```

Mapping Black-Boxes

- ✓ **Black-Box hides functionality**
  - Internal logic of block is removed
  - Done via the elaborate stage (-bbox_m or –bbox_i)



- ✓ **Meanings for SEC**
  - Black-Box outputs are considered inputs, and need to be connected
  - Black-Box inputs are considered outputs, and need to be compared (assertions)
  - Black-Box ports are automatically mapped by SEC
- ✓ **Notes**
  - Black-Box modules must be guaranteed to be equivalent
  - If Black-Box input assertions fail, the Black-Box output connections are no longer valid
  - Output connections are theoretically valid up to the cycle in which the input assertions fail

Outline

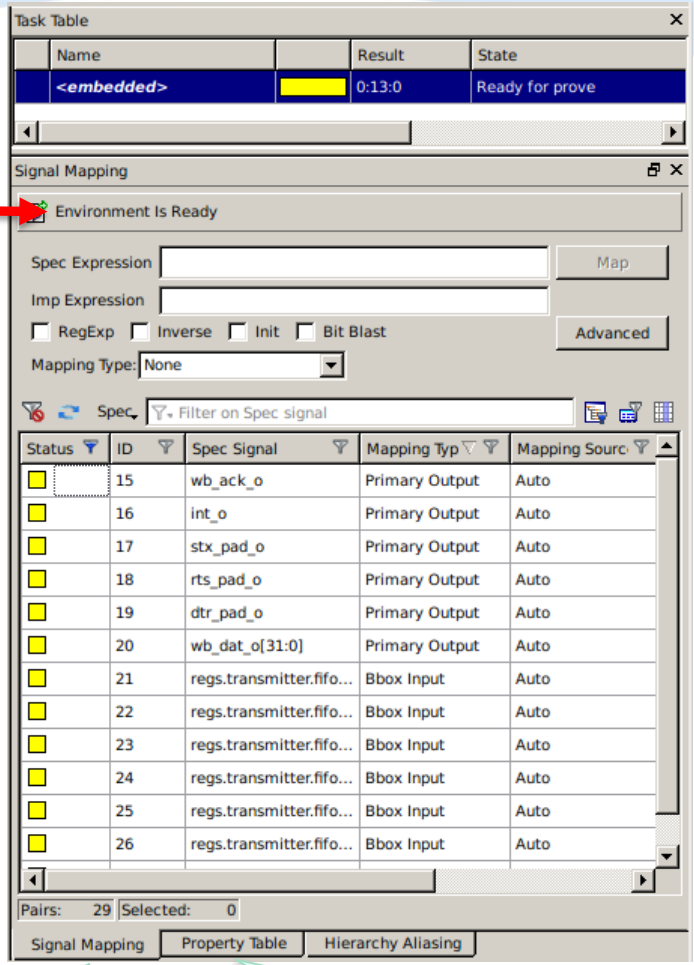
- ✓ Section 1. Introduction to SEC
- ✓ Section2. JasperGold – SEC Setup
- ✓ **Section3. JasperGold – Proof & Debug**
- ✓ Section4. JasperGold – Clock Gating Verification

Generate Verification Environment

check_sec -generate

- ✓ **Generates a task with SEC properties**
- ✓ **Example messages**

```
INFO (ISEC031): Exposing connection mappings.
INFO (ISEC021): Computing SEC environment configuration.
INFO (ISEC032): Computing clock data.
INFO (ISEC033): Computing reset data.
INFO (IRS003): Setting non-resettable flop to value 0
during reset analysis.
INFO (ISEC034): Exposing the remaining mapping entries.
INFO (ISEC042): Running interface check.
```



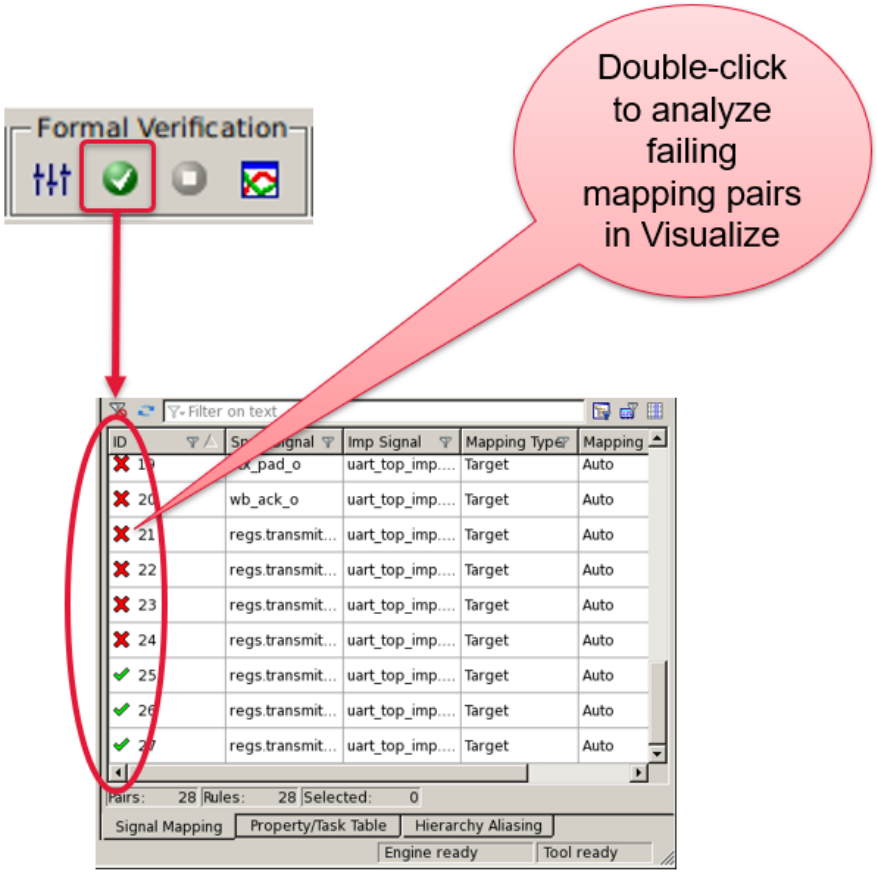
All the mapped signal pair and their status

Assertions and Assumes

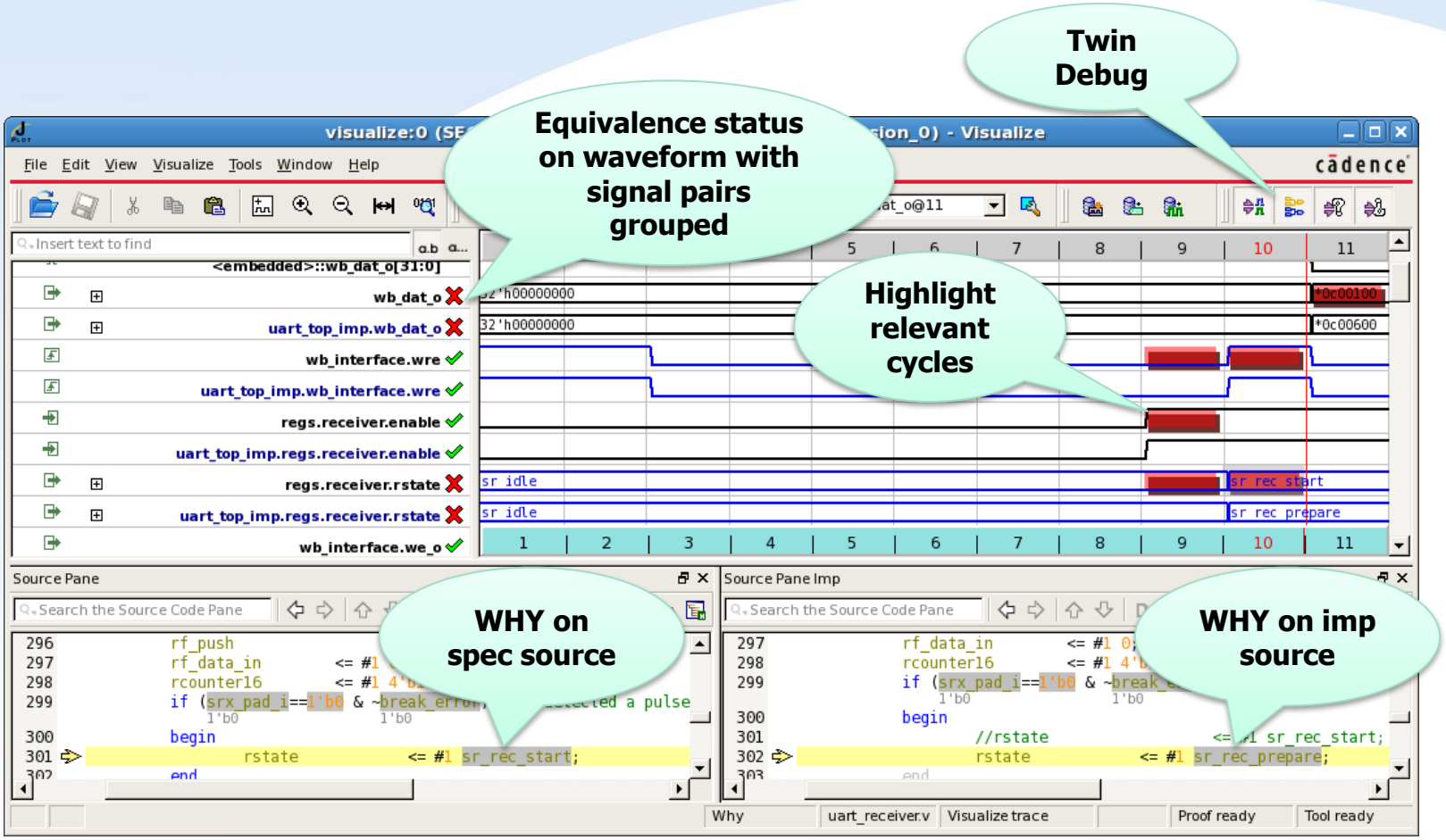
Prove

check_spec – prove

- ✓ **Runs formal analysis on the generate task**
- ✓ **Mapping pairs results**
  - Proven: pair is equal
  - CEX: pair is not equal
  - Undetermined: prove did not finish
  - Unprocessed: prove was not started
- ✓ **All green → Done**
- ✓ **Red → Debug**

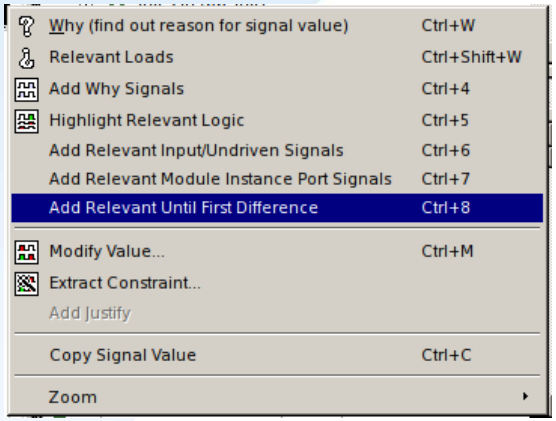


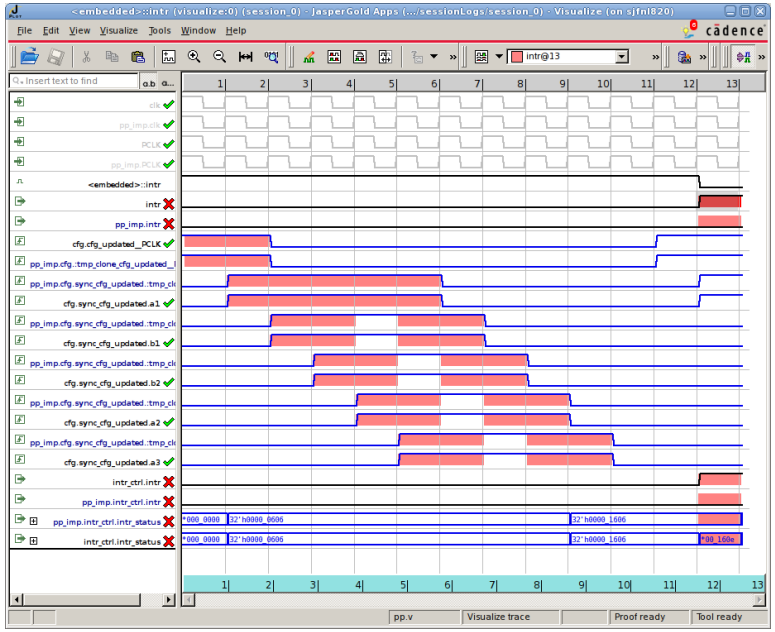
Debugging CEX in SEC App



Highlight Relevant Logic Until First Difference

Unique JasperGold Debugging feature for SEC Waveforms



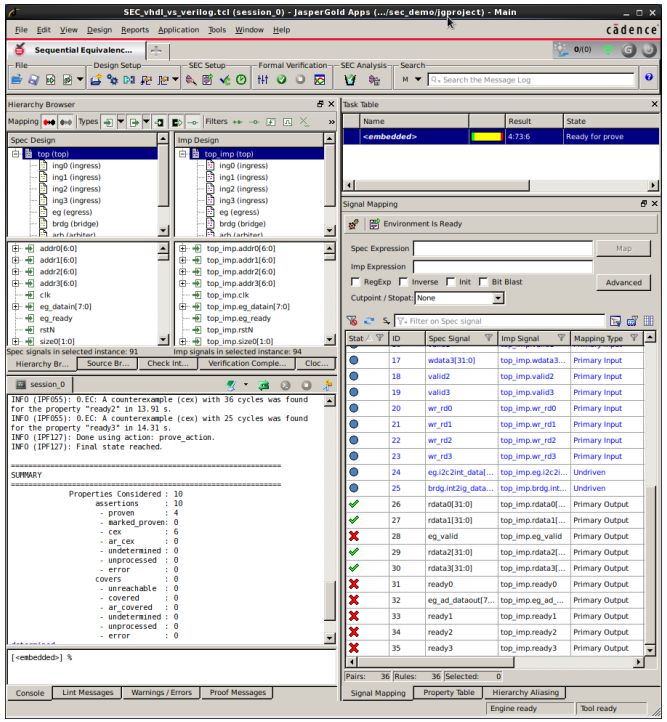


Also useful for traditional backtracing:

- Apply „Highlighting until first difference“
- Remove added signals again
- Perform Why-Plot Iterations

SEC Summary

- ✓ **Sophisticated Proof and Bug Hunting strategies provide state of the art performance**
- ✓ **Eliminates the need for weeks of simulation regressions after minor design changes**
- ✓ **Perform sequential equivalence checking on small and large sub-system blocks**
- ✓ **Optimized workflow GUI simplifies and speeds verification environment setup and comparative analysis and debug features using "Twin's Debug"**

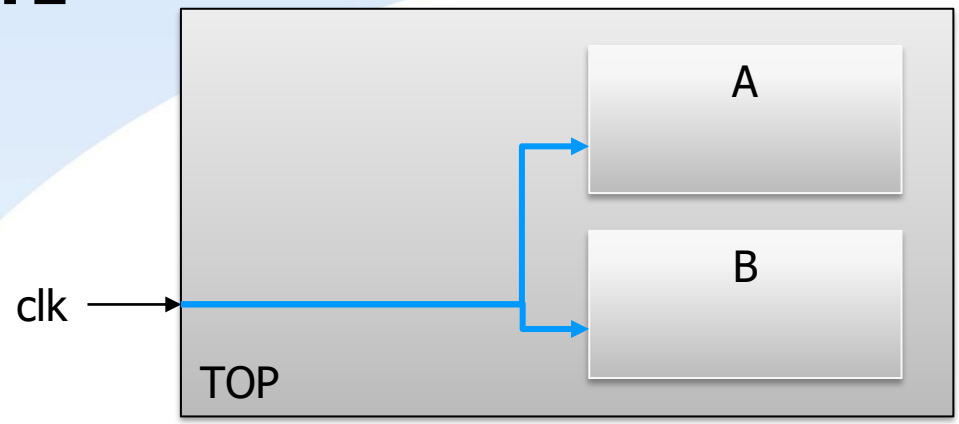


Outline

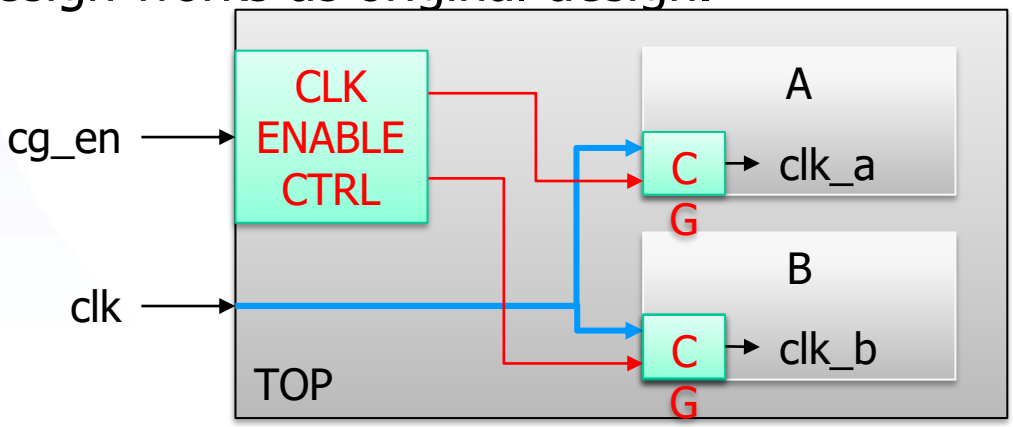
- ✓ Section 1. Introduction to SEC
- ✓ Section2. JasperGold – SEC Setup
- ✓ Section3. JasperGold – Proof & Debug
- ✓ **Section4. JasperGold – Clock Gating Verification**

Module Level Clock Gating

- ✓ **Original RTL**



- ✓ **Optimized RTL**
  - cg_en can enable or disable clock gating functionality. If cg_en is 0, design works as original design.



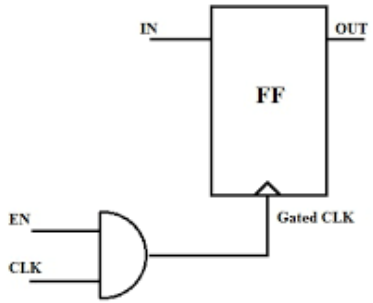
- **Enable condition for entire module, group of DFF**
- **Can be configured globally**

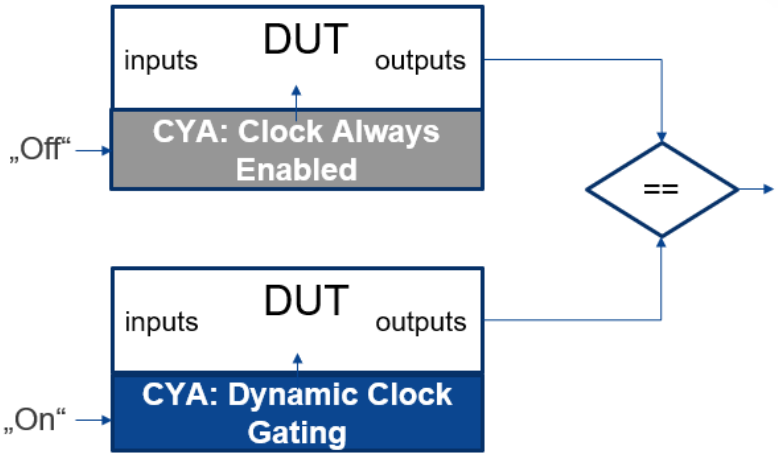
Clock Gating (CG) Verification

- ✓ **Phase 1: Original RTL vs. Optimized RTL (CG off)**
  - **Verify that the CG insertion logic in itself is safe**
    - "If CG is disable, is the optimized design equivalent to the original design"?
  - **Increased confidence**
    - Chip can be configured to use RTL equivalent to the original version of the RTL in case optimized RTL with CG logic enabled has undesired behavior
- ✓ **Phase 2: Original RTL vs. Optimized RTL (CG on)**
  - Verify that RTL with CG logic enabled is equivalent in functionality to the RTL with CG logic off
  - Combined with Phase 1 Pass result, this is the same as checking Original RTL vs Optimized RTL (CG On)

Clock Gating (CG) Verification Setup

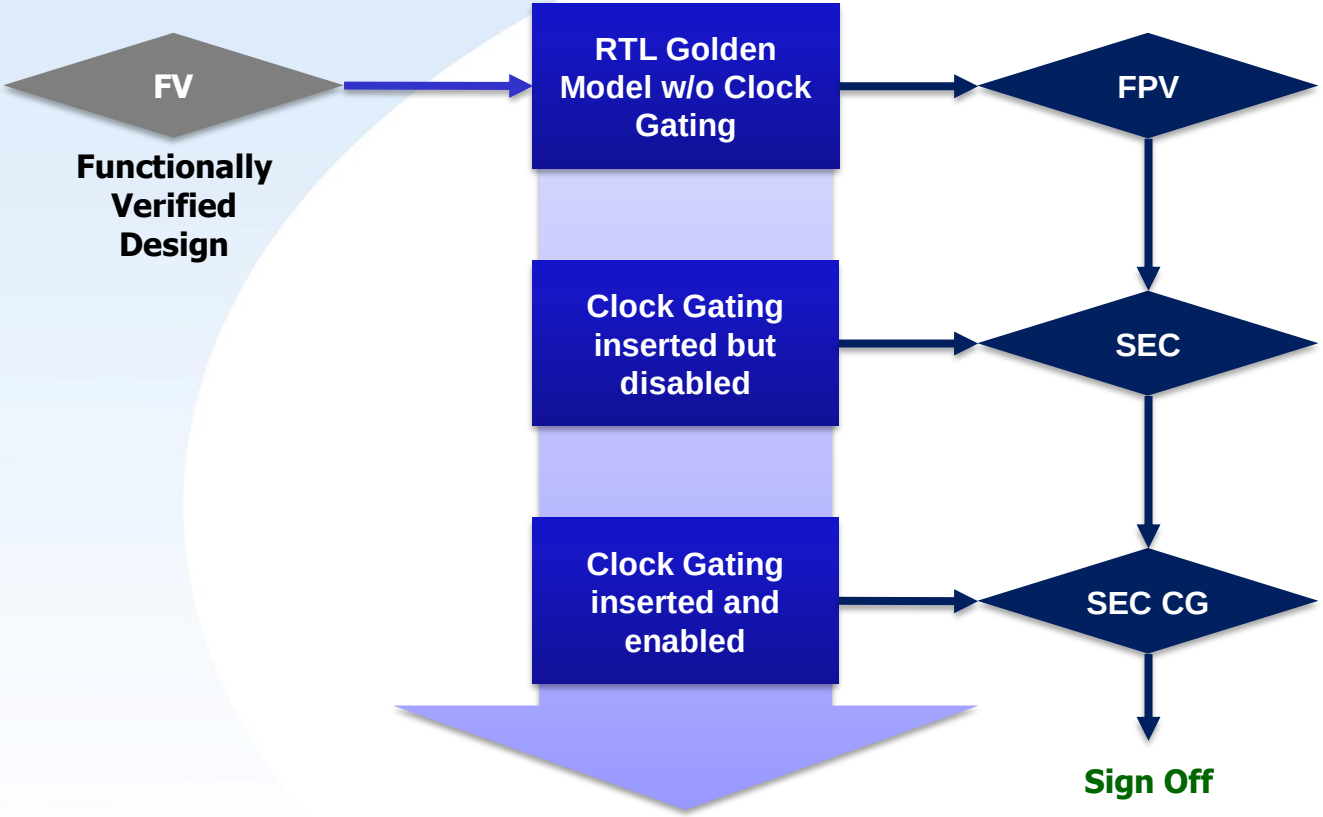
- ✓ **IMP and SPEC are identical**
- ✓ **Design has clock gating logic**
  - Purpose is to save dynamic power consumption
  - Clock is off when not needed, enabled when needed
  - No functional difference on outputs
  - Configuration logic (CYA) introduced to disable CG, in case bug is found in Silicon
- ✓ **Goal: CG does not impact functionality**
- ✓ **Value: Extremely high**
  - Clock Gating bugs are critical
  - Usually means „lights out" for whole chip area
  - Disabling CG means to miss advertised power specification
- ✓ **Effort: Low**
  - Easy to setup, difficult to debug
  - Constraints/conditional mappings may be needed






```
Disable clock gating in SPEC
% assume {cg_en == OFF} -env
```

Clock Gating Verification Sign-Off Flow



Goal:
Clock and Reset Setup Design Exploration

Goal:
Feature Addition - Compare backward compatibility

Goal:
Clock Gating Verification

Clock Gating Verification Setup

- ✓ **Clock Gating Enable signal must be manipulated in SEC CG setup**

- ✓ **If clock gating enable signal is internal signal: Add stopats**

```
stopat {clk_cfg_en imp_top.clk_cfg_en} -env
```

- disconnects RTL driver
- creates free undriven signals

- ✓ **Waive clock gating enable signal pair**

```
check_sec –waive –waive_signals
        {clk_cfg_en imp_top.clk_cfg_en}
```

- unmaps existing mappings
- prevents initial value mapping
- discards interface check failure

- ✓ **Control clock gating in SPEC**

```
assume {clk_cfg_en == 0} -env
# assume {imp_top.clk_cfg_en == 1} -env
```

- disable clock gating in SPEC
- free clock gating in IMP

Clock Gating Verification Proof

- ✓ **Proof strategy "Design Style – Clock Gating" is optimized for this problem**

  set_sec_autoprove_strategy design_style

  set_sec_autoprove_design_style_type clock_gating