

# Automated Formal Verification of Spinner Based SOC IO Pad Frame Logic: Challenges and Solution

Amit Roy, Supriya Bhattacharjee, Bijitendra Mittra  
Interra Systems Inc, Bangalore, India

Subir K. Roy  
Texas Instruments Inc, Bangalore, India

**DAC User Track 2010**

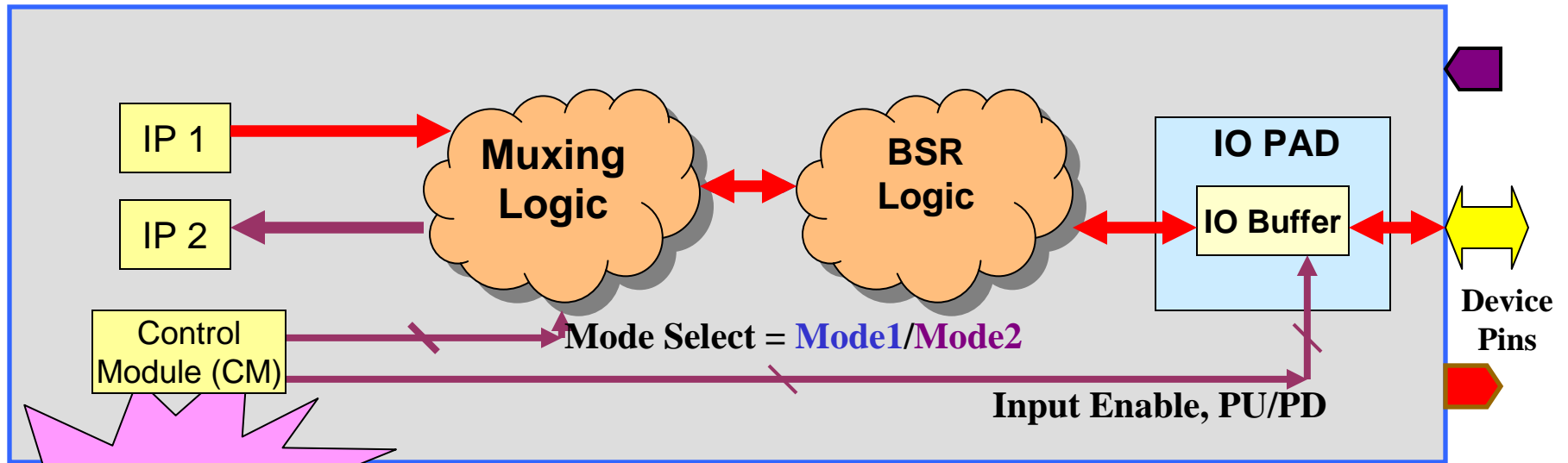
# Outline

- **Introduction**
  - Typical IO Subsystem Logic Structure
  - Architecture of Spinner Generated IO RTL
  - IO Logic Verification Environment
- **Proposed Solution**
  - Initial Solution: Spinner FV Flow & it's limitations
  - Improved Solution: Unified Spinner FV Flow
  - Unified Spinner FV Flow Diagram
- **Results**
- **Conclusion & Summary**

# Introduction

- Current SoCs are very complex in nature
  - Incorporating increased functionalities and feature sets, which results in:
    - Complex IO Structure consisting of:
      - Intricate Multiplexing, Custom IO Cells, Power Management Logic etc.
    - Complex IO fabric generated through third party tools like Spinner
    - Manual Integration of SOC Core Logic into Spinner generated top level wrapper
- Need of the hour:
  - Leverage power of Formal Verification to carry out exhaustive validation of such Complex IO fabrics by covering all possible scenarios
  - Start the verification process at an early phase of RTL development to result in a better Time To Market (TTM)
  - Develop a robust and useful push-button verification flow that will be re-usable across SOC's

# Typical I/O Pad Frame Logic Structure



Configuring CM to  
MODE1 (IP1)/  
MODE2 (IP2)

```
always (((fv_conf_gpmc_a2[2:0] == 3'b001)) -> (fv_gp0_io_in[10] == gpmc_a2));
```

```
always ((fv_conf_uart2_ctsn[2:0] != 3'b000) -> (fv_uart2_ctsn_in == 0));
```

```
always (((fv_conf_gp0_io5[2:0] != 3'b000) && (fv_conf_gp0_io5[2:0] != 3'b001))  
-> (gp0_io5 == 1'bZ));
```

```
always ((fv_conf_mcb_fxr[2:0] == 3'b000)  
-> (fv_mcb_fxr_in == mcb_fxr));
```

```
always (((fv_conf_mcb_fxr[2:0] != 3'b000) &&  
(fv_conf_mcb_clkr[2:0] == 3'b000))  
-> (fv_mcb_clkr_in == mcb_clkr));
```

```
assert always (gp0_io5 == '0');
```

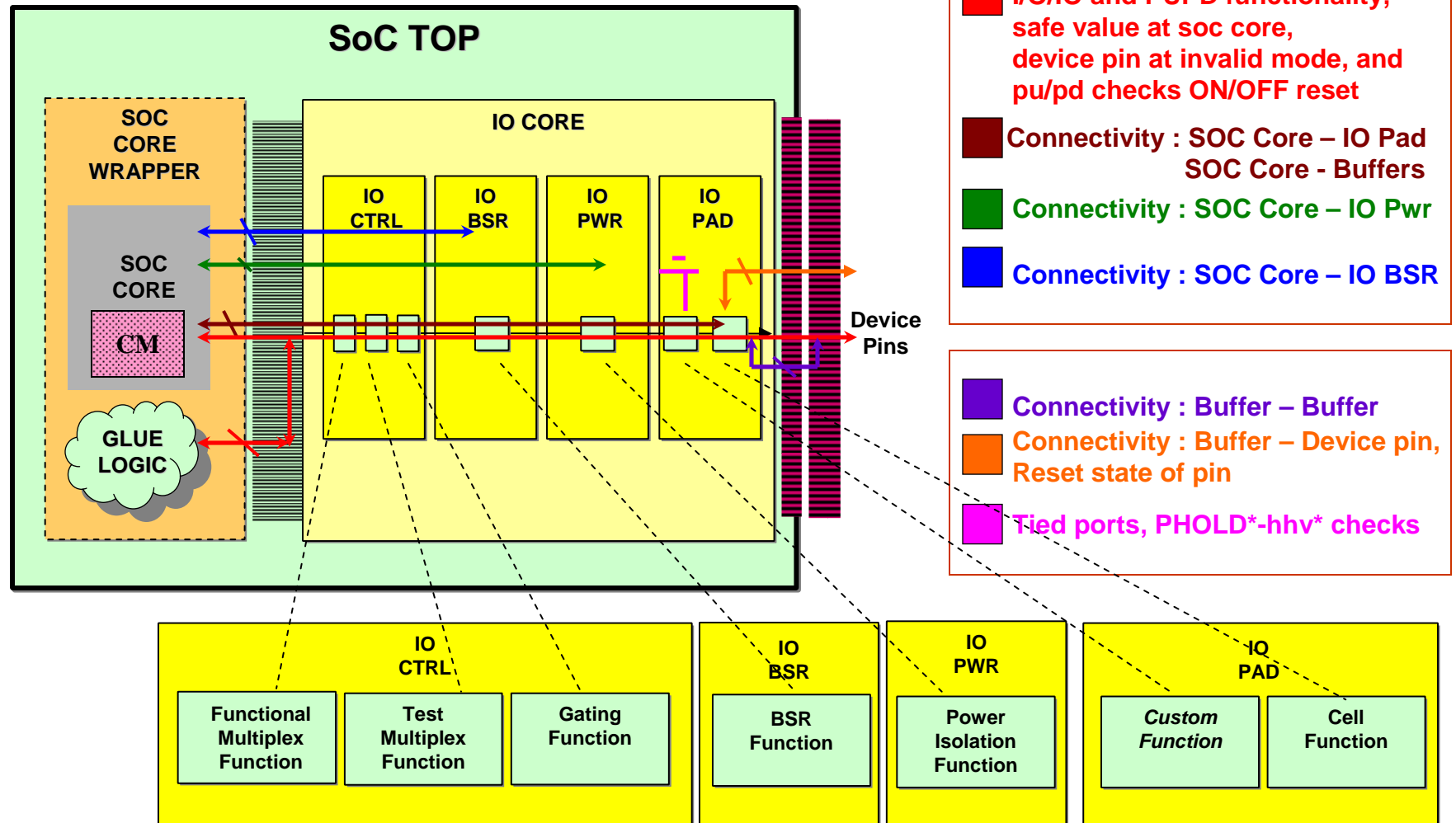
```
assert (gp0_io5 == '0');
```

```
always (fv_ioBuf_port_i_tsi0_clk_pi == '0');
```

```
always (fv_ioBuf_port_i_tsi0_dclk_pi_2 ==  
not(topReset));
```

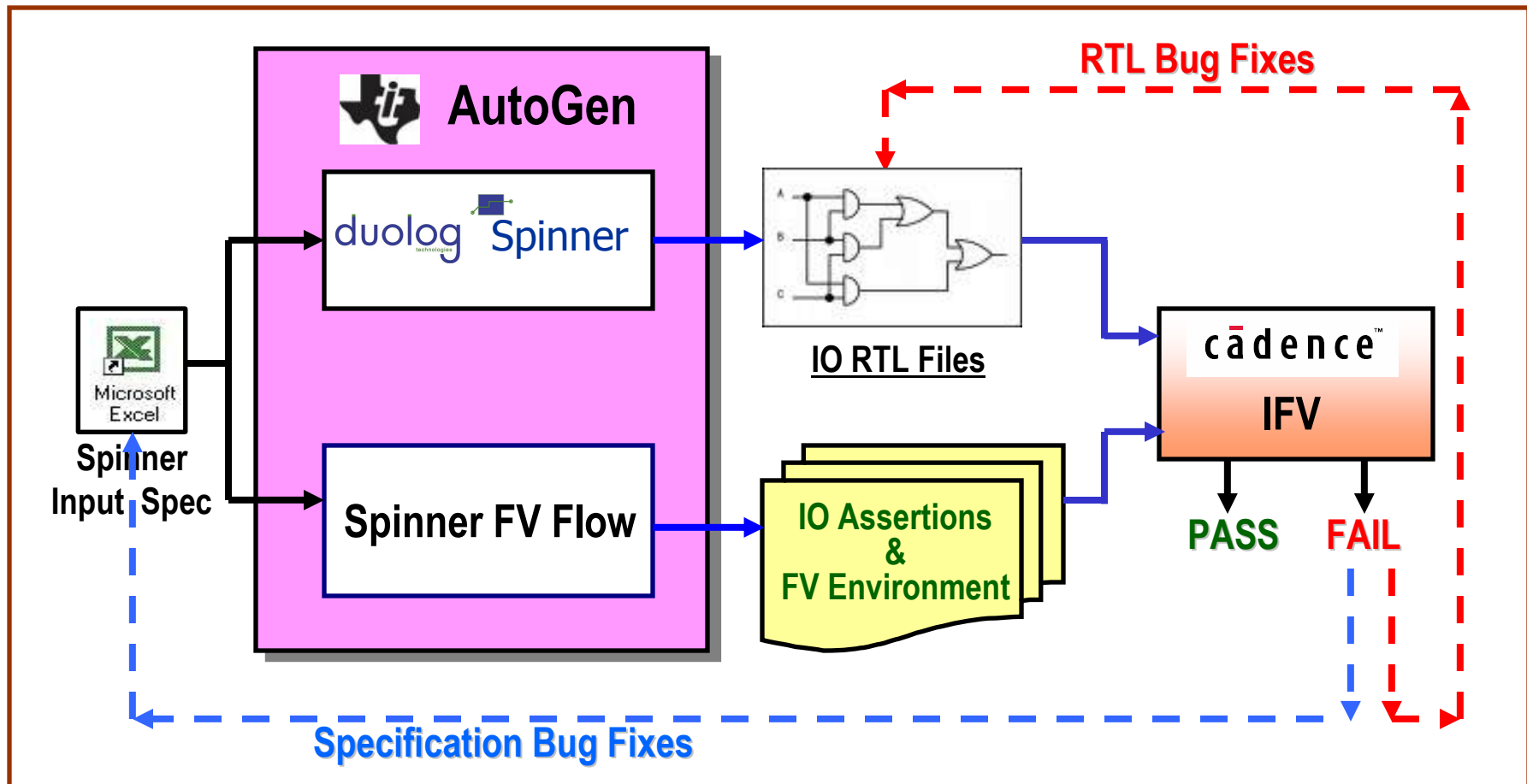
```
always (fv_ioBuf_port_i_tsi0_dclk_pholdl_3 == '0');
```

# Architecture of Spinner Generated I/O RTL



Complete SoC RTL is not loaded for the formal analysis – only Spinner RTL and relevant SoC core logic is loaded

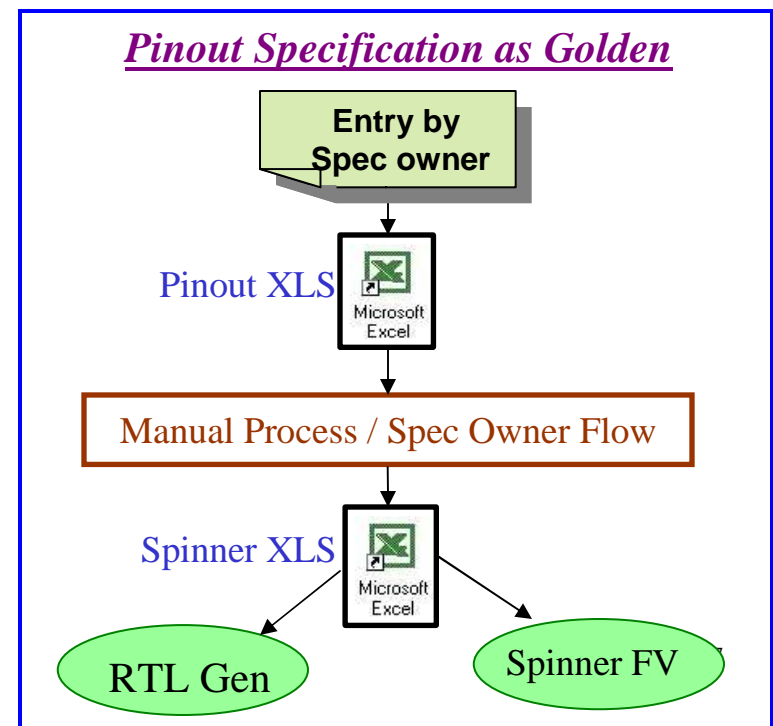
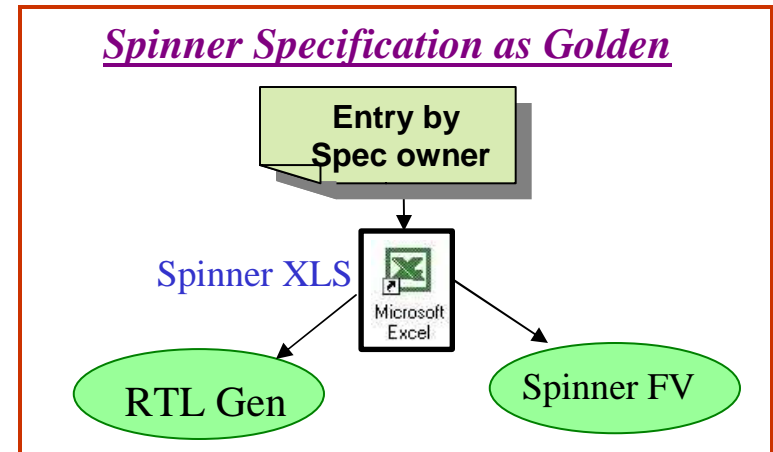
# I/O Logic Verification Environment



- AutoGen is a common platform within TI, built on Atrenta's 1Team Genesis

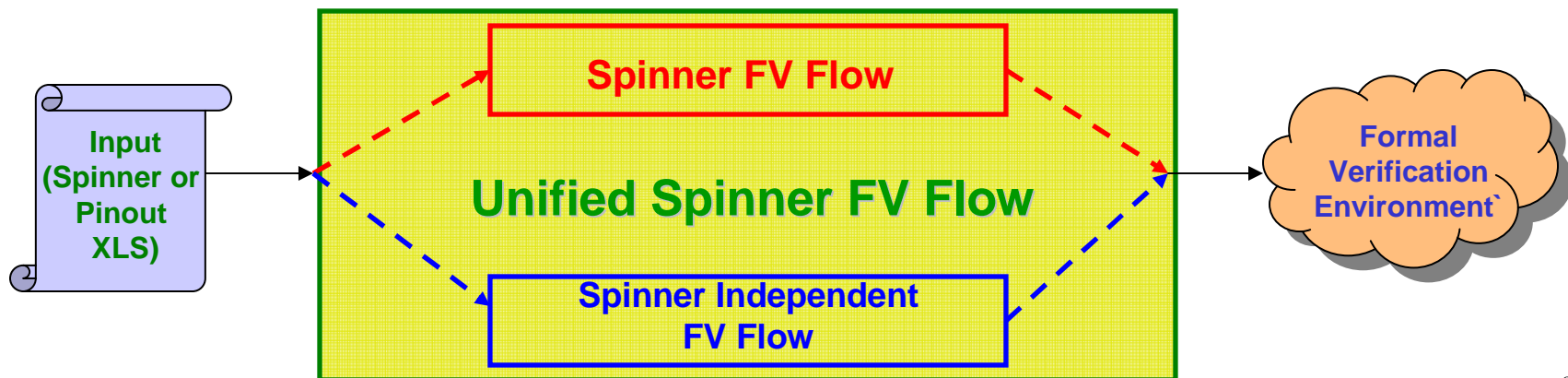
# Initial Solution: Spinner FV Flow

- Developed push-button **FV flow for Spinner generated IO Pad frame RTL**
  - Takes as input the Spinner specification XLS
  - Generates PSL/SVA assertions along with the verification environment
    - Assertions for all the layers of the I/O fabric structure and critical functionalities of IO pads
    - Assertions to check the functionality of custom IO cells such as MIPI DPHY, SMART2, I2C etc.
- Limitations of using Spinner XLS as input
  - ☹ The IO subsystem RTL and the assertions get generated from the same Spinner XLS.
  - ☹ No information on SOC core logic is available in Spinner XLS. So, cannot verify connectivity with SOC core logic
  - ☹ Some of the in-house design groups use Pinout XLS (TI internal IO Spec format) as the golden specification
  - ☹ To use the existing flow for Pinout XLS, Spinner XLS needs to be created manually from it.



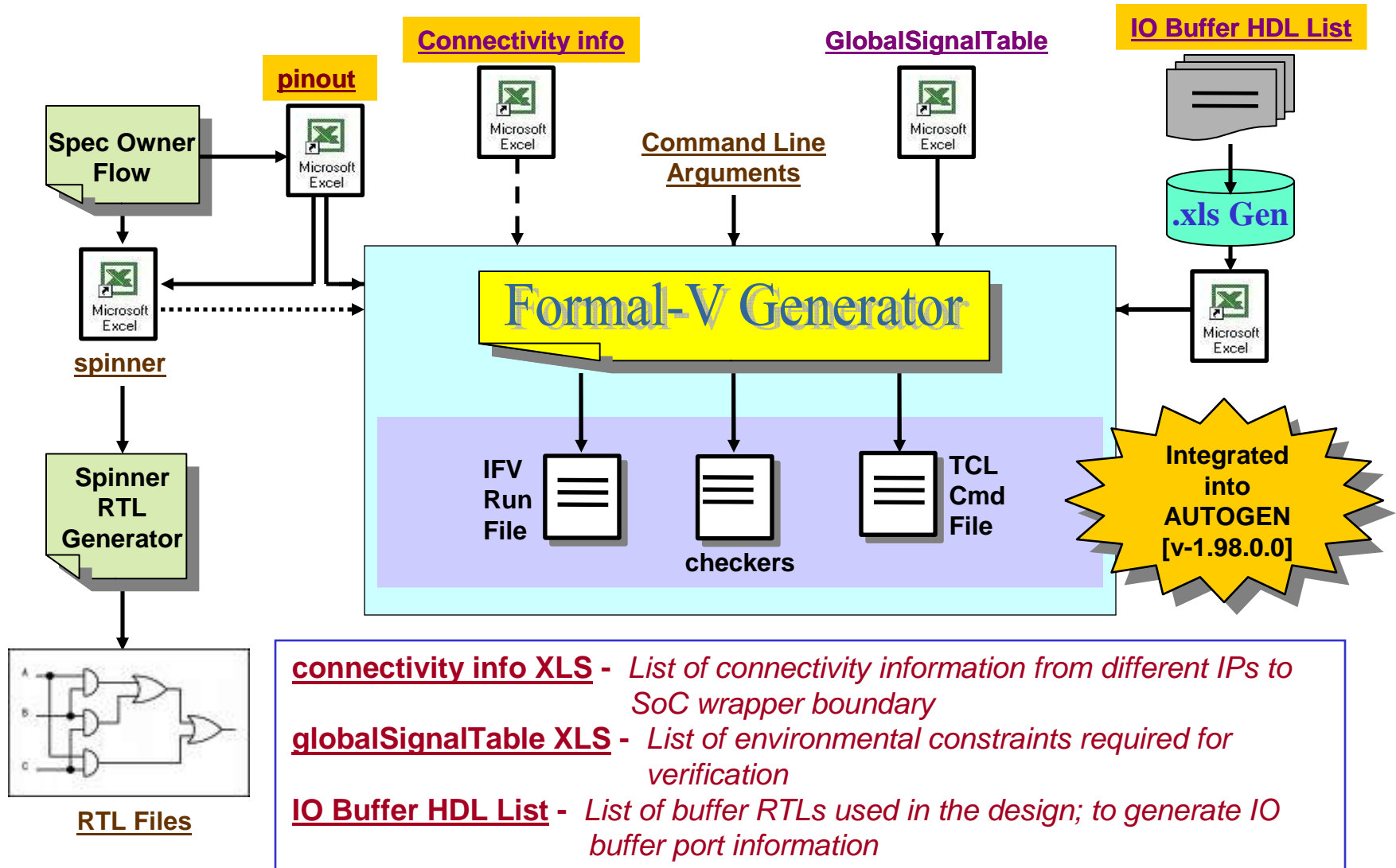
# Improved solution: Unified Spinner FV Flow

- To overcome the limitations of the Spinner FV flow, we have come up with a new **Unified Spinner FV Flow**,
  - Preserves all the features of the Spinner FV flow
  - Addition of new features like, Reset Checks, Z-value Checks, PU/PD based Checks, Connectivity checks etc. are supported
  - Added a sub-flow independent of Spinner specification XLS
  - Unified Spinner FV Flow is a superset subsuming the Spinner FV and Spinner independent FV sub-flows
  - A flow that caters to all design teams within TI WW
  - Integrated into common framework for better availability to design teams





# Unified Spinner FV Flow Diagram



# Statistics for Unified Spinner FV Runs on different SOC's

SOC Verified using Unified Spinner FV Flow	Total no. of IO Pad used in the design	No. of RTL releases verified by the flow	Total number of Assertions Generated per RTL release	Total number of Bugs or Issues found	Unified FV Sub- Flow used to generate assertions and formal environment	Average time taken to complete single IFV run on one RTL release (min)	Boundary Toggle Coverage of the top level device pin and SOC core func. ports
<u>SOC 1</u>	212	7	9137	34	Spinner Independent	95	100% & >90%
<u>SOC 2</u>	331	4	7290	8	Spinner Dependent	63	100 % & NA
<u>SOC 3</u>	387	2	7732	4	Spinner Independent	77	NA & NA

**Note: Formal verification is applied to each and every IO Pad in each SOC in every verification run**

# Conclusion & Summary

- 😊 Unified FV Flow supports both the flows using Spinner XLS as well as Pinout XLS providing an alternative verification path resulting in higher confidence
- 😊 Push-button automated flow resulting in minimum manual intervention needed to generate the verification environment , which can be regressed on every new RTL release
- 😊 Exhaustive verification using FV at early stage of design development; Bugs caught early in the design cycle; Lesser time taken compared to conventional methods; Better quality end product; Less TTM
- 😊 All SOC teams benefited due to availability of FV flow through AutoGen (common platform available within TI)
- 😊 Boundary Toggle Coverage report enables users to improve the verification quality
- 😞 Manual creation of “connectivity” specification from SOC Core/IPs to IO core : This process is time consuming and error prone
- 😞 Relevant portions of the design are loaded; Abstraction of RTL needed
- 😐 Future scope : BTC and CM FV Flows to be integrated into Unified Spinner FV Flow

# Extensions to Unified Spinner FV Flow

- **Boundary Toggle Coverage (BTC) Flow**
  - Independent post-processing script
  - Provides the toggle coverage information of the top level SOC ports from the IFV runs
  - BTC parses the IFV log files for passed assertions and the top level RTL for the list of ports and generates a toggle coverage report for all the top level SOC ports
  - If an assertion passes, it is guaranteed by any model checking based formal tool that both 0-to-1 and 1-to-0 transitions have been verified for the given pair of signals
    - The pair of signals in a passed assertion is reported as covered whereas signals used in a failed assertion are reported as not covered
  - The generated coverage is captured in the form of an XLS
    - Gives better visibility on the completeness of verification using the FV Flow
- **Control Module (CM) FV Flow**
  - Flow to verify the following behavior of any Control Module within the SOC
    - Internal register read and write checks
    - Connectivity checks from Control Module to the boundary of the SOC core
    - Default Value Checks (just after reset deactivation) of the ports, such as, Input Enable, Mode Select, Pull Controls, and Control Signals of the Custom IOs, at the boundary of the SoC Core Module.

# Set of Assertions Generated by Unified Spinner FV Flow for an SOC

Category of Assertions (Functionalities verified using Unified Spinner FV Flow)	No. of Assertions (approx) generated per RTL release
IO Muxing Checks	499
Device Pin Pull-UP/Pull-Down Checks	57
Default Value on Reset Checks	317
PHOLD, HHV Tie-value Checks	2544
I2C Buffer Functionality Checks	240
SOC Core to Custom IO Connectivity Checks	900
Custom IO to Top device pin Connectivity Checks	29
SOC Core to BSR boundary connectivity Checks	972
Protected/Inhibit/Prohibit Value Checks	137
SOC Core to IO Pad buffer connectivity Checks	3002
Special SMART2, MIPI cell connectivity Checks	440