

# Formal Verification of Connections at SoC-level

Penny Yang, Prasun Das, Yuya Kao, Mingchu Kuo



# Introduction

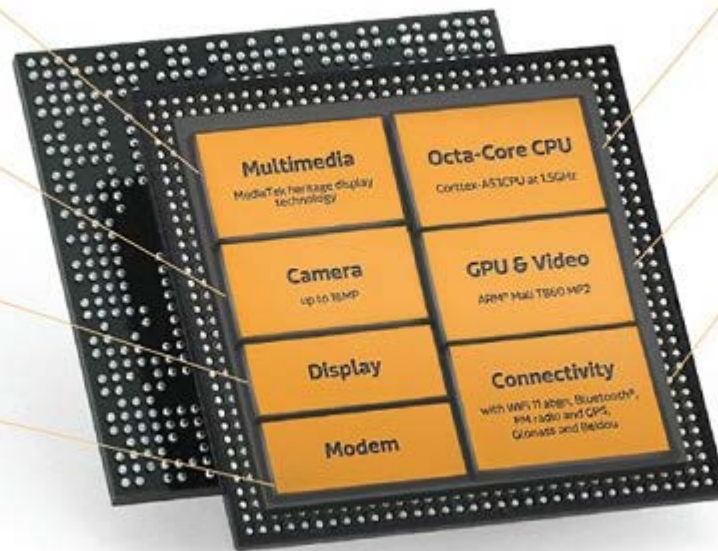
- Complete verification of connections at SoC level is a fundamental requirement to ensure correct operation

MiraVision™ display technology delivers superior user experience without tradeoffs

High-resolution 16MP camera support, with advanced camera features (PDAF, PIP/VIV, IHDR support...)

Support for displays up to 1920x1080 resolution w/6 blending layers (MT6750T)

World-Mode modem supporting LTE Cat.6 with 20+20 CA, CDMA SRLTE



Octa-Core A53 CPU at up to 1.5GHz, offering balance between performance and power-efficiency

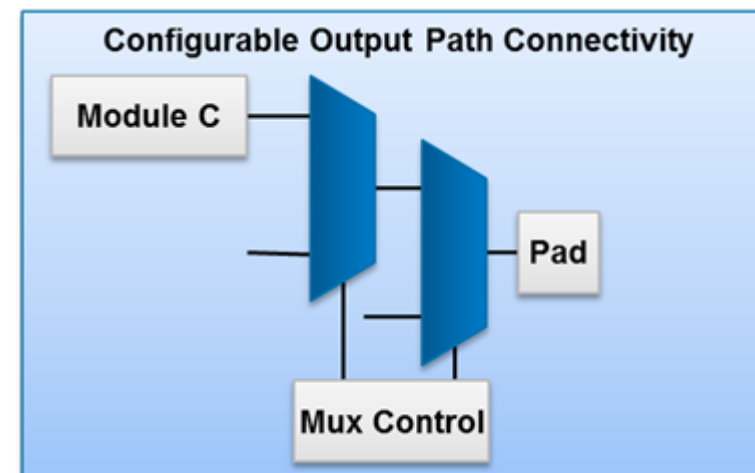
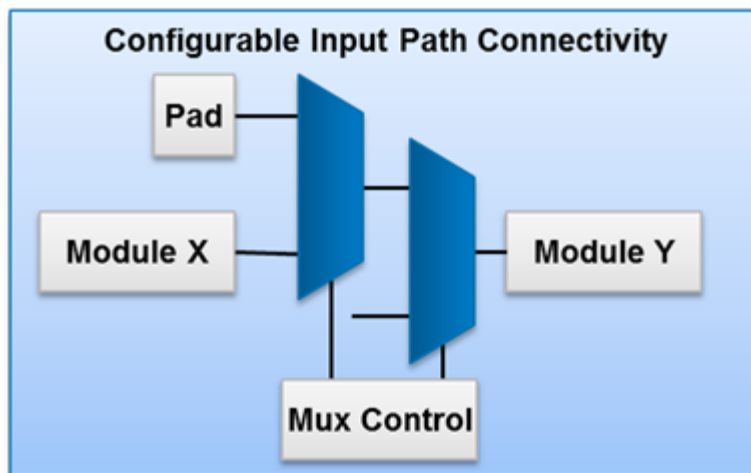
Video encoding and decoding up to 1080p at 30fps (MT6750T)

Integrated connectivity

\*Unit sizes shown are for illustrative purposes only.

# SoC Challenges – Padmux

- Larger gate count, but limited I/O pins
- Leads to shared I/O pins, muxing to control access
- Requires significant high level interconnect wiring
- Introduces significant possibility of errors
- Hard to do ECO because combinational logic is often optimized after synthesis

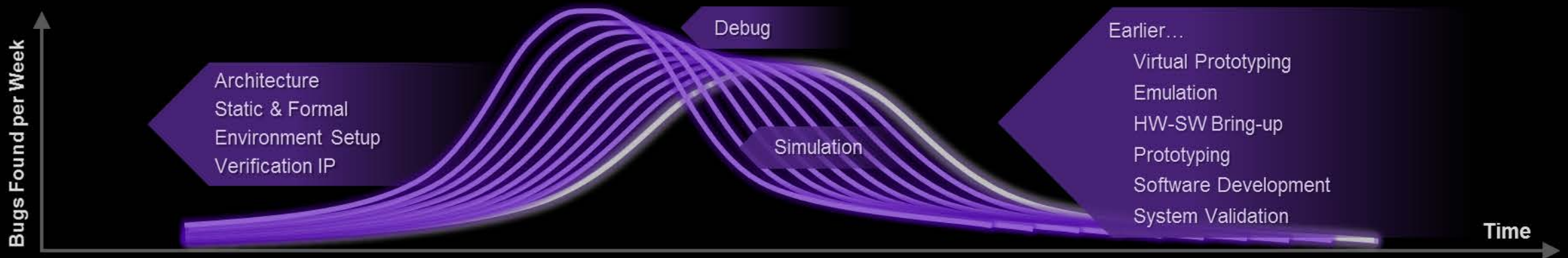


# SoC Challenges – Global reset

- SoC designs have multiple sources of global reset, such as power-on reset, hardware reset, software reset and watchdog timer reset.
- These are top-level signals that should be connected to all asynchronous resets in the design, i.e., all asynchronous resets in the design should be asserted when the global reset is asserted.
- Taking watchdog reset verification as an example, a directed test created to verify if the watchdog reset operation works correctly must trigger the watchdog reset condition in the middle of the simulation to check if the watchdog reset has propagated to all the intended flip-flops in the entire SoC.

# Background

- Before using formal verification, chip level simulation was used to verify the connections at SoC-level.
  - Fewer simulation patterns in chip level environment
  - Corner case bugs sometimes appeared in uncovered codes
  - Integration of chip level test-bench often comes late in a project cycle
- To be able to shift left, formal verification is adopted due to its exhaustiveness and easy setup.



# Motivation

- Formal property verification has been proven to be a reliable method for different kinds of designs' verification signoff. However,
  - Run time could be several days and the iterations was very slow.
  - Manual abstraction is required to achieve full proof
  - Require design knowledge to partition the SoC into several different sub-systems, black-box the modules that were not used, constrain the designs with constants and assumptions.
  - Several months to work on these settings from the beginning till the end of the project
- Trial and error was time consuming and manual abstraction was also prone to false alarms.

# Using formal Application on connectivity checking

- Formal Applications are customized for easy setup, use, and debug.
- It is perfect for beginners because there is no need to have formal background or knowledge to write SVA.
- Problems which are ideal for formal connectivity checking
  - SoC I/O Connectivity
  - Block pin muxing/demuxing
  - Connectivity & constant checking of macros
  - Scan mode connectivity & constant checking
  - Reset and global signal connectivity
  - Registers to Debug Bus



# Difference between CC and FPV

## Connectivity Checks

- Value check and directional connection
- With auto-blackboxing mechanism
- Can debug through schematic + waveform + text

## Formal Property Verification

- Temporal and combinatorial signal relationships
- No notion of direction in SVA
  - “Out2 == Po2” same as “Po2==Out2”
- Manual partition and abstraction
- Can debug through waveform



# Connectivity Checking App

- Tcl commands, CSV formats and Excel files are supported as the input format.
- User can debug the logics in the path using schematics/text/waveform.

```
add_cc -src pad -dest ind -enable \
{ibe && ~obe} -name inpath
```

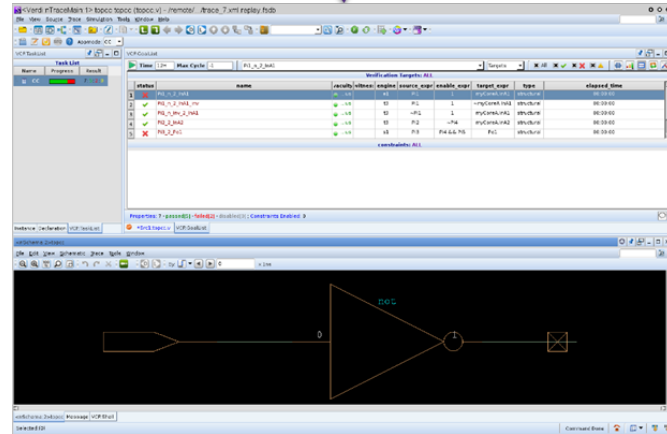
ENABLE	SOURCE	DESTINATION	NAME
"chip_top.test_ctl[5]	chip_top.test_ctl[4]	{ds}	to_ds
chip_top.test_ctl[5]	chip_top.test_ctl[3]	{sr}	to_sr
chip_top.test_ctl[5]	chip_top.test_ctl[3]	{se}	to_se
chip_top.test_ctl[5]	chip_top.test_ctl[1]	{pe}	to_pe
chip_top.test_ctl[5]	chip_top.test_ctl[0]	{pd}	to_pd
{chip_top.test_ctl[5] && "chip_top.port_ctl[5]}	chip_top.moda	{ibe,ibe,obe,dout}	from_moda

Most common input format

Constraints

VC Formal SoC Connectivity Checking Application

DUT



# Input Formats

- Source
- Destination
- Enable
- Name
- Start line
- Comment pattern

```
# Generate reset and register list
report_ff_reset -list
# Reset path
set cpu_rst_en u_reset.wdr
source reset_checker.tcl
```

```
#reset_checker.tcl
add_cc -name c1 -src 0 -dest u_cpu.resetn_i -enable $cpu_rst_en
add_cc -name c2 -src 1 -dest u_cpu.g1.reset_i -enable $cpu_rst_en
...
```

	A	B	C	D	E	F
1	From	To	Enable	Name		
2	Pi2	top.myCoreA.In2	top.in_mux_sel == 1'b0	in2A		
3	Pi3	top.myCoreB.In1	top.in_mux_sel == 1'b1	in1B		
4	top.myCoreA.Out2	Po2	top.out_mux_sel == 1'b0	out2A		
5	top.myCoreB.Out1	Po2	top.out_mux_sel == 1'b1	out1B		
6						

```
#padmux_checker.csv
u_cksys.tck, u_io.PAD_MCK.O, {gpio_mode==1}, a1
u_io.PAD_MDAT.I, u_cksys.dsp_out, {gpio_mode==6}, a2
...
```

# Debug Structurally Disconnected Path

- Structurally disconnected check

```
[Error] CC_UID017: Connection 'InA2_2_Pi1' is structurally disconnected.  
No path from source to target. Please fix the connection
```

- Debug specific path

```
91 PIN-BIDIR west_mux.pad2 (HS = multi_path_switch)  
92 PORT-BIDIR west_mux.pad2 (HS = inout_mux)  
93 OPERATOR west_mux.out2 CONNECT (HS = inout_mux)  
94 PORT-OUT west_mux.out2 (HS = inout_mux)  
95 PIN-OUT west_mux.out2 (HS = multi_path_switch)  
96 PIN-IN east_mux.B3 (HS = multi_path_switch)  
97 PORT-IN east_mux.B3 (HS = inout_mux)  
98 OPERATOR east_mux.muxB MUX (HS = inout_mux)  
99 OPERATOR east_mux.pad2 BUF_IF (HS = inout_mux)  
100 PORT-BIDIR east_mux.pad2 (HS = inout_mux)  
101 PIN-BIDIR east_mux.pad2 (HS = multi_path_switch)  
102 3 internal objects  
103 11
```

vcf>

Message VC Formal Console

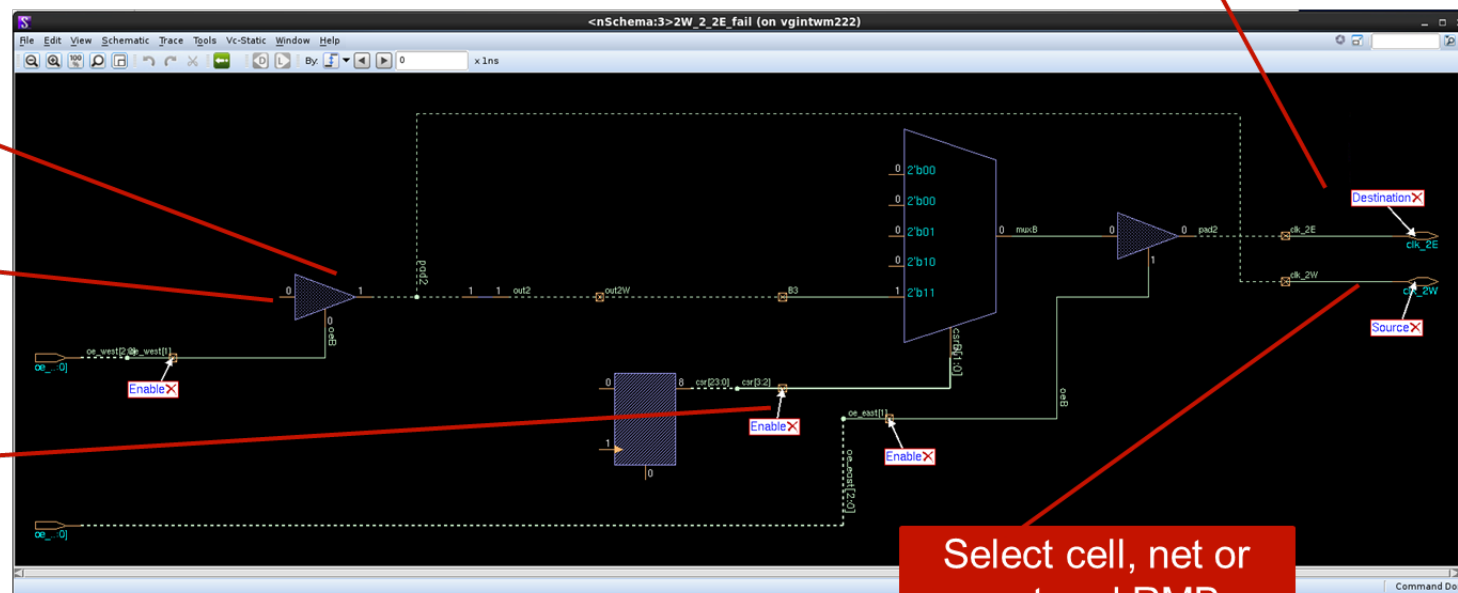
# Debugging in Schematic View

Signal values annotated on schematic

Double click on input to show driving logic

Select cell, right mouse click to show RTL source

Shows source, destination & enables



Select cell, net or port and RMB menu: copy signal path

# Debugging in a single GUI platform

The screenshot displays the VeriTrace GUI interface. On the left, a 'Task List' window shows a task 'CC' with a progress bar and a result of '7:2:0'. Below it, a 'VCP.Shell' window shows a 'List Results' section with a 'Connectivity List' containing 7 connections, some connected and some unconnected. A context menu is open over the 'Verification Targets' table, with several items circled in red. Callout boxes with arrows point to these items and other parts of the interface.

**Task and Status** (points to Task List)

**RMB click to get menu** (points to the context menu)

**Show Schematic or Waveform** (points to 'View Trace...' and 'New Schematic Path')

**Tcl command window** (points to VCP.Shell)

**Show all or selected results** (points to the 'Targets' dropdown)

**Verification Targets: ALL**

status	name	vacuity	witness	engine	source_expr	enable_expr	target_expr	type	elapsed_time
✗	Pi1_n_2_InA1	...us		e1	Pi1	1	myCoreA.InA1	structural	00:00:00
✓	Pi1_n_2_InA1_inv	...us		t3	Pi1	1	~myCoreA.InA1	structural	00:00:00
✓	Pi1_n_inv_2_InA1	...us		t3	~Pi1	1	myCoreA.InA1	structural	00:00:00
✓	Pi2_2_InA2	...us						ural	00:00:00
✗	Pi3_2_Po1	...us						ural	00:00:00
✓	Pi6_2_Po2	...us						ural	00:00:00
✓	bitselect	...us						ural	00:00:00

**Context Menu Items:**

- View Trace...
- Navigator
- Property Complexity Report
- Property Progress Report
- New Schematic Path
- Check Selected Properties
- Clear Selected Properties
- Enable/Disable...
- Modify selected Property(s)
- Add selected Property(s) to a temp (USR) Group...
- Create new Task for selected Property(s)...
- Show Property Source
- Copy Name

**VCP.Shell Output:**

```
List Results
Connectivity List:
-----
> Connection
# Connection: 7
[ 0] connected      (non_vacuous) - Pi2_2_InA2
[ 1] unconnected    (non_vacuous) - Pi3_2_Po1
[ 2] connected      (non_vacuous) - Pi6_2_Po2
[ 3] unconnected    (non_vacuous) - Pi1_n_2_InA1
[ 4] connected      (non_vacuous) - Pi1_n_2_InA1_inv
[ 5] connected      (non_vacuous) - Pi1_n_inv_2_InA1
[ 6] connected      (non_vacuous) - bitselect
```

# Results

Case	# of connections for verification	Verification time w/ traditional FPV based CC flow	Verification time w/ Optimized CC flow	Improvements
<b>Reset 1</b>	173981 (103971 proven, 70010 failed)	35 hrs	40 mins	52X
<b>Reset 2</b>	25000 (23241 proven, 1759 failed)	4 hrs	12 mins	20X
<b>Reset 3</b>	1 (1 proven)	8 hrs	3 mins	160X
<b>Padmux 1</b>	4732 (4731 proven, 1 failed)	4 hrs	3 hrs	1.3X
<b>Padmux 2</b>	447 (441 proven, 6 failed)	5.6 hr	3.4 hr	1.7X

The design is an SoC with 64,056,916 register bits

# Results

- The application of CC is optimized for connectivity checking problems.
- The run time for formal verification is improved significantly with the innovative automatic abstraction flow optimized in CC compared to FPV.
- Thus, we can easily integrate the commands into MediaTek's regular flow using either tcl or csv format



# Conclusion

- This presentation presents highly automated methodologies using Formal techniques to verify the correctness of global reset schemes and padmux connection, without the large amount of effort required by manual abstraction in SoC.
- The methodologies described above have been deployed on 10 projects at MediaTek.
- As the results show, we have found that the strength of CC App is more efficient than pure FPV based connectivity verification methodologies.
- The connectivity verification flow can be completed in hours, without any inconclusive properties, on an SoC size design.

# Future Work

- Measure Connectivity Checking Coverage
- Spec completeness – are all paths checked?
- Combine with simulation coverage data for connectivity checks
- Uses the same toggle coverage goals as VCS
- Creates a coverage database that can be merged with simulation connectivity coverage data

# Connectivity Coverage in GUI

The screenshot displays the Verdi GUI for connectivity coverage analysis. The main window shows the source code for the `multi_path_switch` module. The coverage summary indicates a score of 25.26% for the module.

**CovDetail Table:**

Variable	Type	Coverage	Display	depth
clk	port	0.00%	0.00%	
clk_1E	port	100.00%	100.00%	
clk_1N	port	100.00%	100.00%	
clk_1S	port	0.00%	0.00%	
clk_1W	port	100.00%	100.00%	
clk_2E	port	100.00%	100.00%	
clk_2N	port	100.00%	100.00%	
clk_2S	port	100.00%	100.00%	

**Summary Table:**

Variable	0->1	1->0	depth
clk	X	X	