Efficient Verification Strategies to Accelerate Complex SoC Design Validation using ZeBu Emulator

Thavamani Pachiyappan (*thavamani.p@samsung.com*)

Samhith Kumar Pottem (*samhith.p@samsung.com*)

Sarang Kalbande (*sarang.mk@samsung.com*)

Garima Srivastava (*s.garima@samsung.com*)

Rakesh Singh (*rakeshprasad.singh@synopsys.com*)

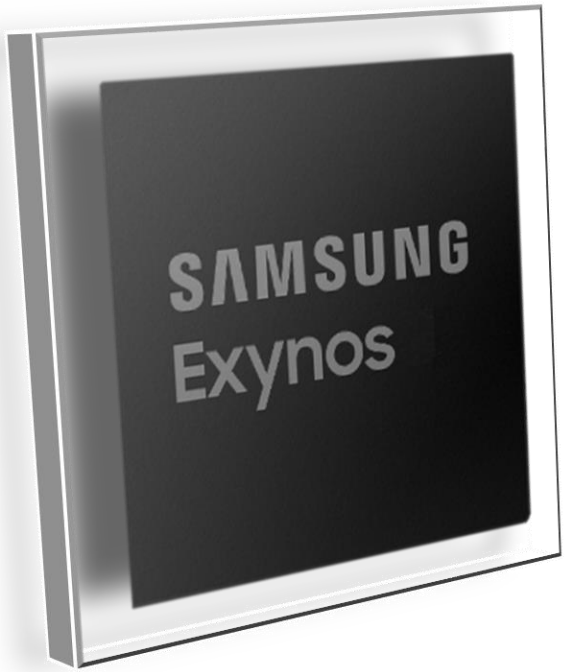
Samsung Semiconductor India Research

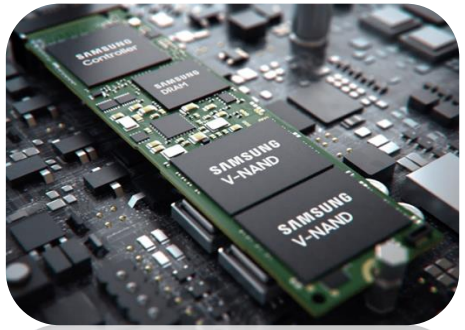
Agenda

- Introduction
- Motivation
- Emulation Compilation Flow
- Compilation Challenges
- Optimization Strategies – Compile Time
- Run Time Challenges
- Optimization Strategies – Run Time
- Conclusion
- Future Scope

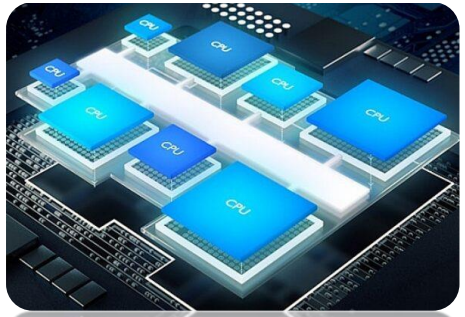
Introduction

- Growth in Semiconductor Industry.
- Design complexity of SoCs.

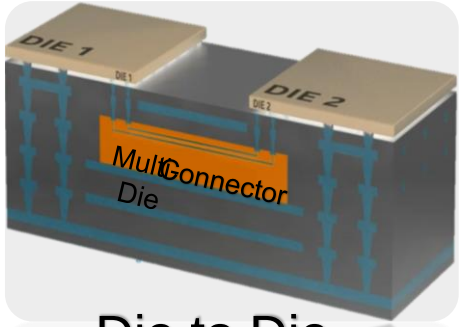




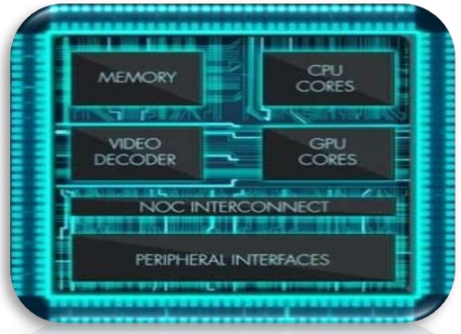
Memories



MultiCore
CPUs, GPUs,
NPUs, DSPs



Die to Die

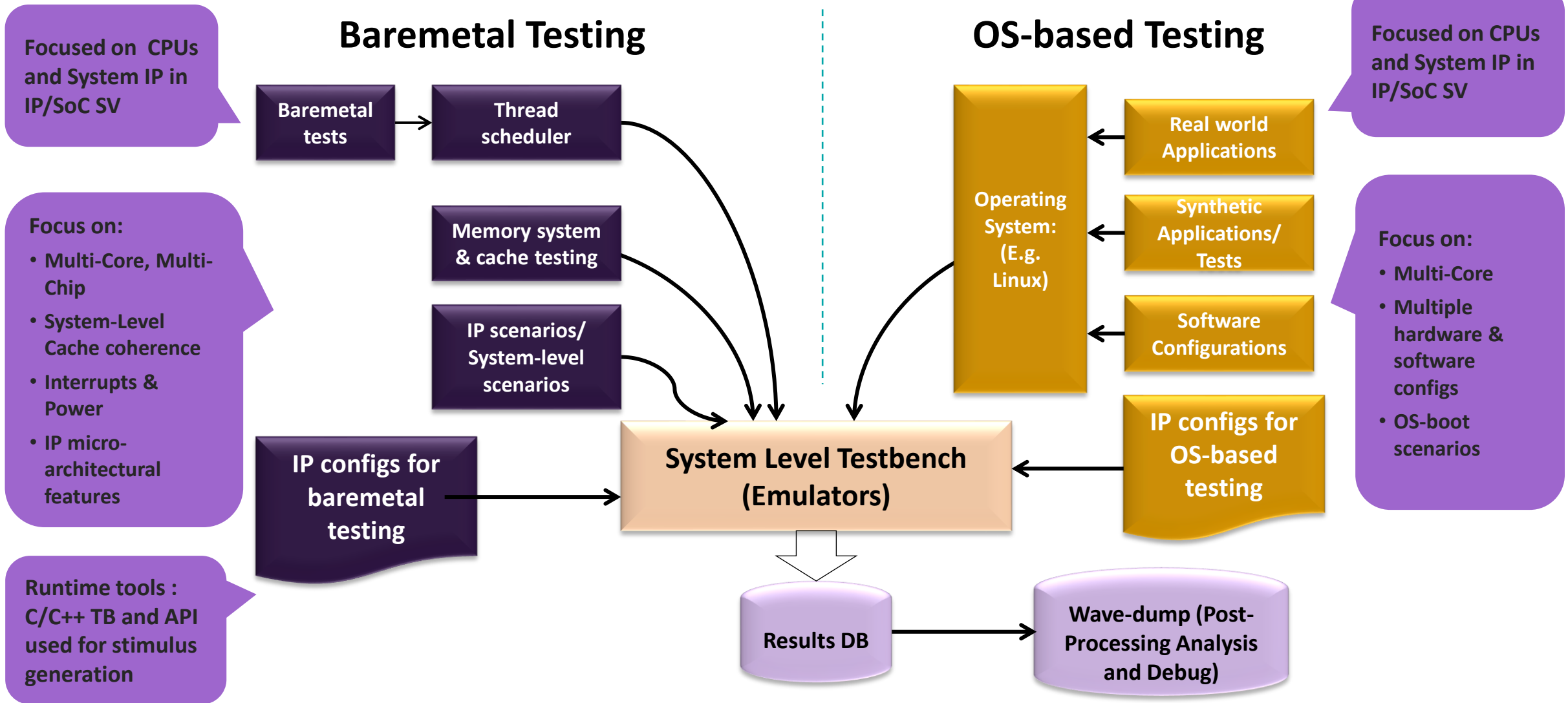


Complex
Subsystems

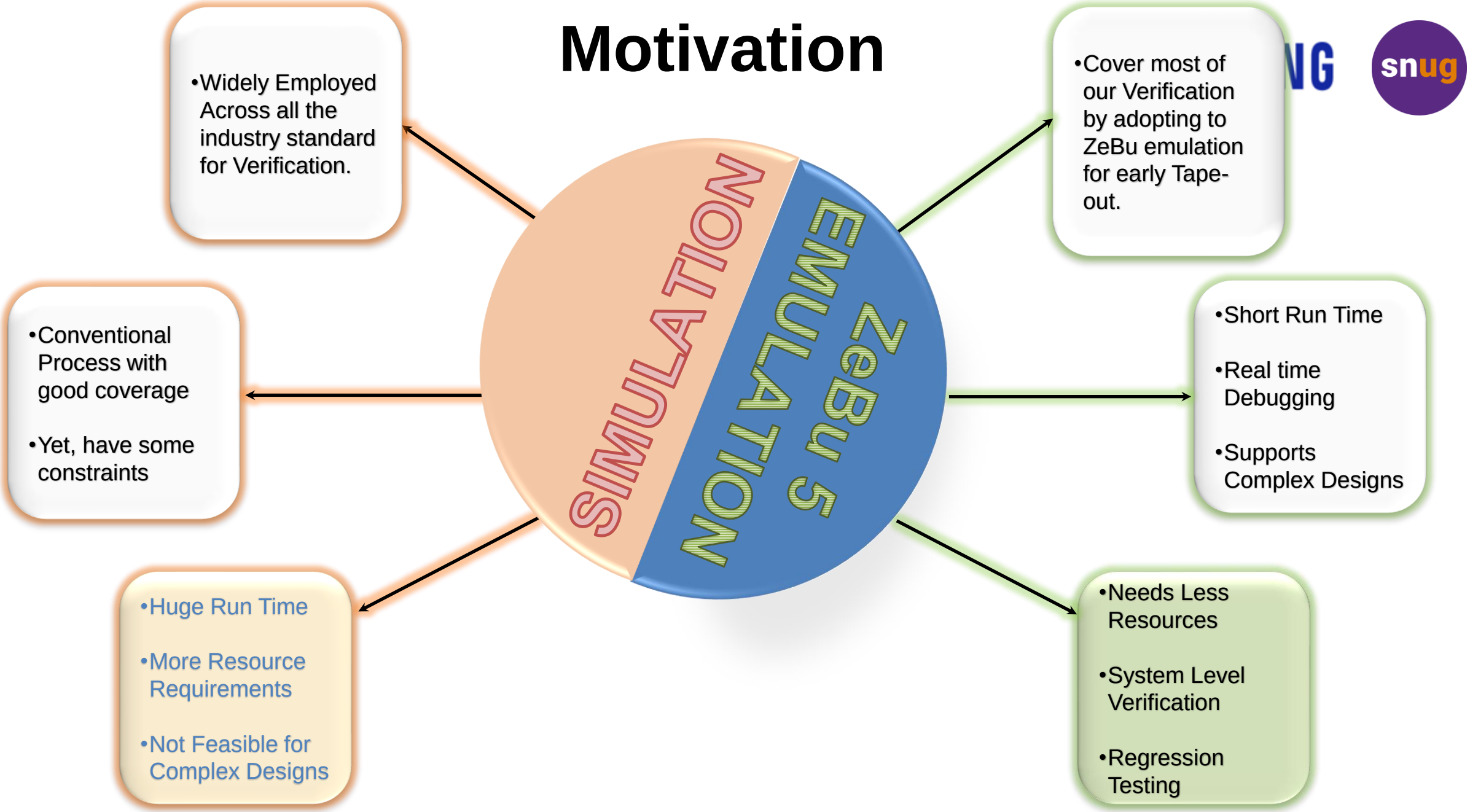
Introduction (Contd..)

- Growth in Semiconductor Industry.
- Design complexity of SoCs.
- Challenge to Validate chip on time.
- Ensuring maximum verification coverage with No bug escape.

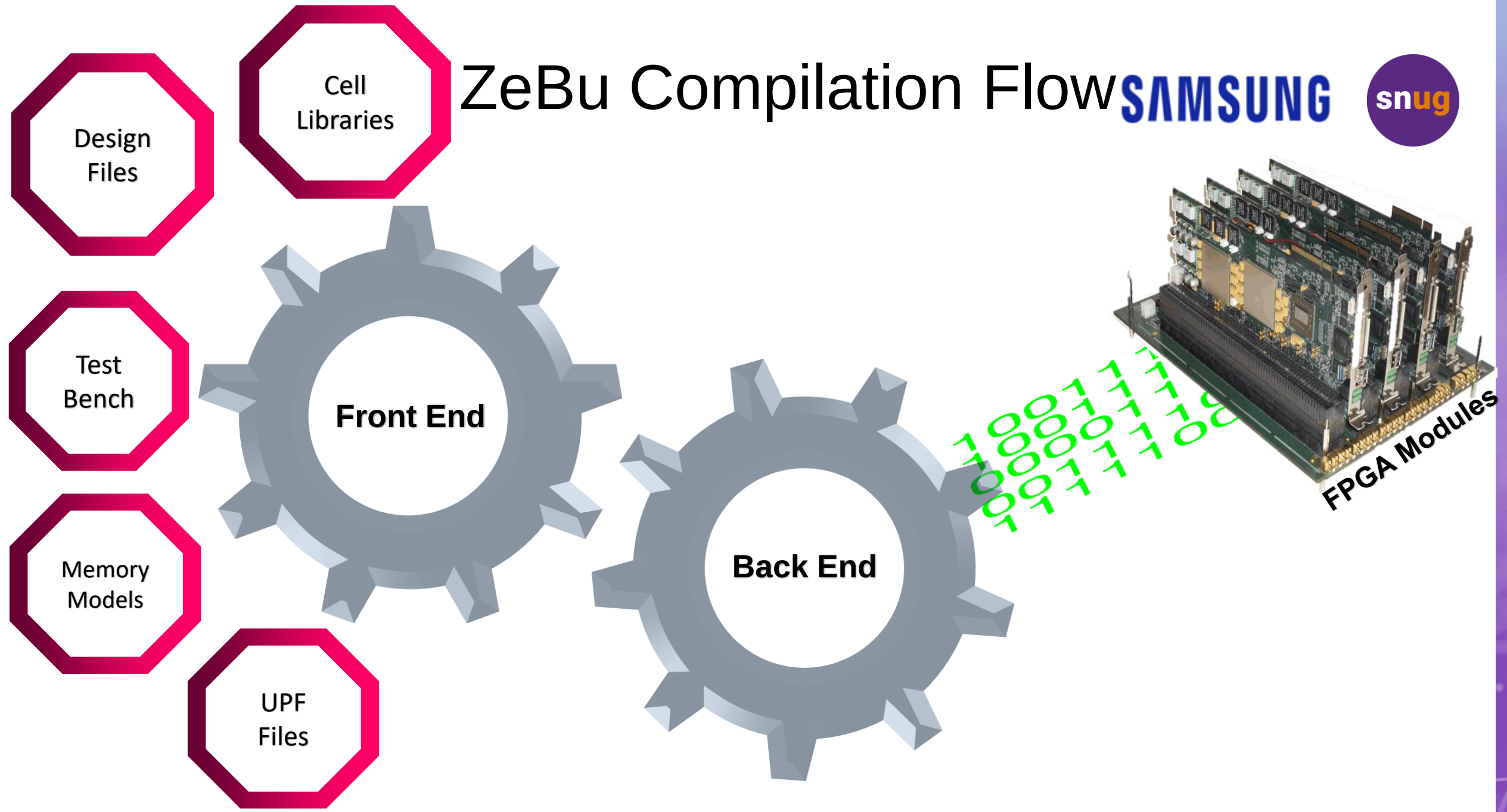
Introduction - System-Level Verification Overview



Motivation


•Widely Employed Across all the industry standard for Verification.
•Conventional Process with good coverage
•Yet, have some constraints
•Huge Run Time
•More Resource Requirements
•Not Feasible for Complex Designs
SIMULATION
ZeBu EMULATION
•Cover most of our Verification by adopting to ZeBu emulation for early Tape-out.
•Short Run Time
•Real time Debugging
•Supports Complex Designs
•Needs Less Resources
•System Level Verification
•Regression Testing

ZeBu Compilation Flow



Compilation Challenges

- Placement and Routing Issue
- High Compile Time
- Huge Routing Congestion

Compilation Optimization Strategies – P&R Stage

Compilation Optimization Strategies – Compile Time

- Specific zFAST attributes and UTF command used
  - For any module with greater than 10000 LUT6, tried remapping so that the total LUT6 after remapping are less than 30% of the total LUT count.

```
synthesis -advanced_command {Compile:MinLut6ToRemap = 10000}
synthesis -advanced_command {Compile:TargetLUT6Ratio = 0.3}
```

- Reduced LUT6 with Cost setting using UTF command
  - Without optimization, one of FPGA has gone into PARFF and then it was routing congested with Level 6
  - Looking into CSV file, there was highly connected logic in NoC module
  - Tried using synthesis optimization to spread this logic using lot more LUT with less inputs i.e. reducing the number of LUT5 and LUT6 compared to other types of LUT

```
optimization -lut_cost { 1 1 1 5 6} –module {*noc*} -regex
```

Compilation went through without PARFF and no congestion – **Saved overall compile time as well**

Compilation Optimization Strategies – Routing Congestion

1 **zTopBuild**

- Partitioning auto
- Manual :
  - defcore <core-name> -path_list {<instance path>}
  - use_fpga -zcore <core-name> -module <module>

2 **zCoreBuild**

- Partitioning auto
- Manual
  - defmapping <instance path> <fpga-name>

3 **Partition Options**

- defgroup {instances} : Group set of module instances.
  - **Each group will be placed in separate module in zTopBuild partitioning and**
  - **separate FPGA in zCoreBuild partitioning**

BEFORE OPTIMIZATION

ZCORE	REG	LUT	RAMLUT	LUT6	MUXCY	BRAM	URAM	DSP	LUT (w/o weighting)|	IO
Part_0	5,806K	16M	0	3,694K	8,102	2,136	150	3,437	16M	14938
Part_1	5,223K	15M	0	3,432K	8,477	3,660	455	44	15M	13579
Part_2	6,625K	9,649K	0	2,698K	8,150	2,769	244	0	9,649K	25198
Part_3	6,317K	13M	0	3,115K	8,480	2,505	140	323	13M	16285
Part_4	4,989K	15M	0	3,409K	8,477	4,205	303	36	15M	6627

AFTER OPTIMIZATION

	REG	LUT	RAMLUT	LUT6	MUXCY	BRAM	URAM	DSP	LUT (w/o weighting)|	IO
Part_0	7,958K	17M	0	3,989K	8,576	3,753	0	2,579	17M	17121
Part_1	6,586K	9,611K	0	2,721K	8,150	3,851	0	0	9,611K	26841
Part_2	6,264K	19M	0	4,443K	8,865	6,638	0	44	19M	7085
Part_3	7,129K	21M	0	5,101K	8,782	6,000	0	1,217	21M	9037

Emulation Performance and Partition Efficiency

```
ztopbuild –advanced_command {zcorebuild_command * {cluster_constraint
{add_group} {name=DDR0} –keep_group –fnmatch –path_list {top.dut.BLK_MIF0
top.lpddr5_0 top.lpddr5_1 top.zebu_srm_Xdram0*}}}
```

Run Time Challenges

- Performance Drop
  - Inter-connectivity between the blocks
  - Clocking
  - Data Paths

  As a result, the emulator driver clock frequency is very low.

- Run Time Debug Analysis

Run Time Optimization Strategies

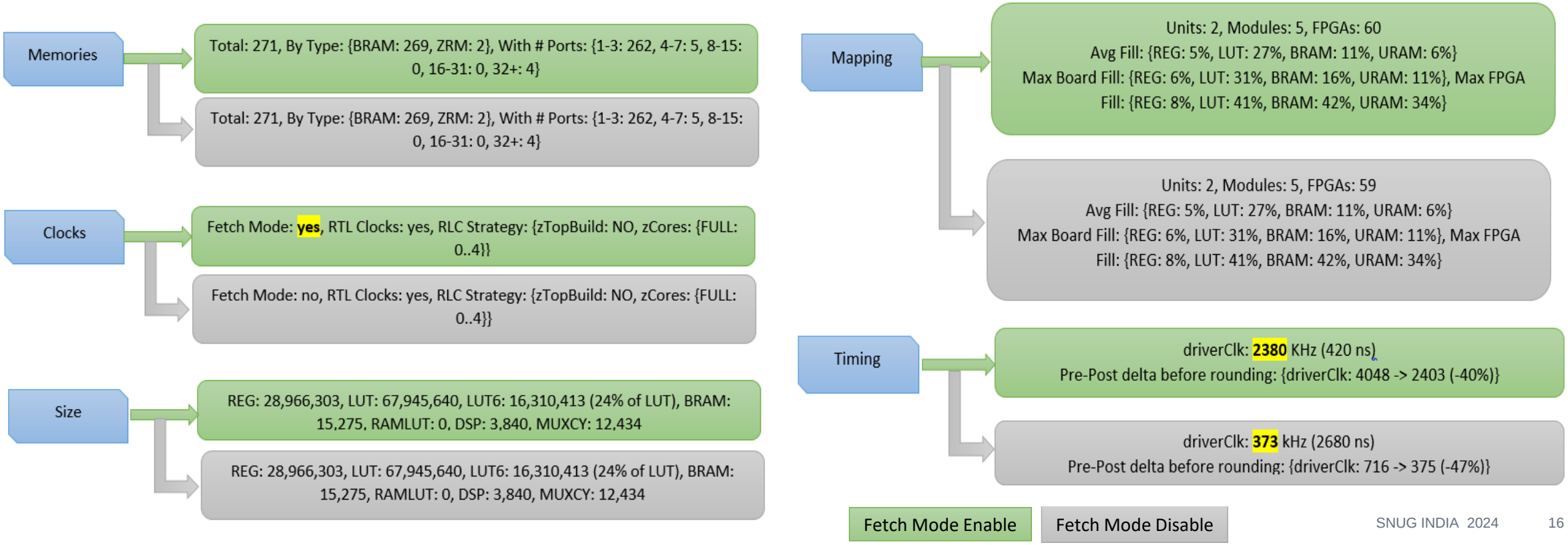
- Make a base compile with general optimization switches
- Analyze the Inter Module/Inter FPGA connectivity
- Clock Cone Analysis
  - Overhead of doing clock localization
  - Elements in clock cone/Reset cone
  - Unique clock groups as identified by the compiler
- Analyze zTIME report to check critical clock/data paths
- Enable optimizations incrementally (cluster constraints, winding path, data localization)

Iterate over above steps

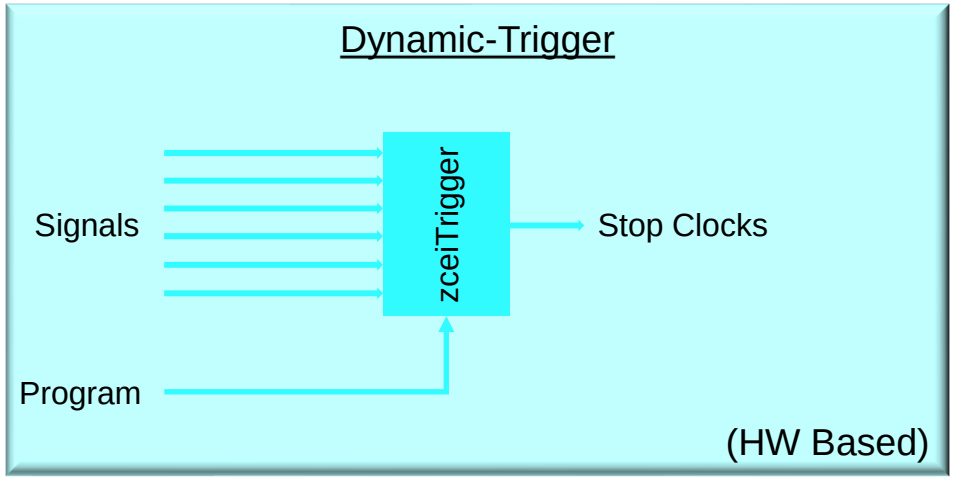
Run Time Optimization Strategies – Performance Improvement

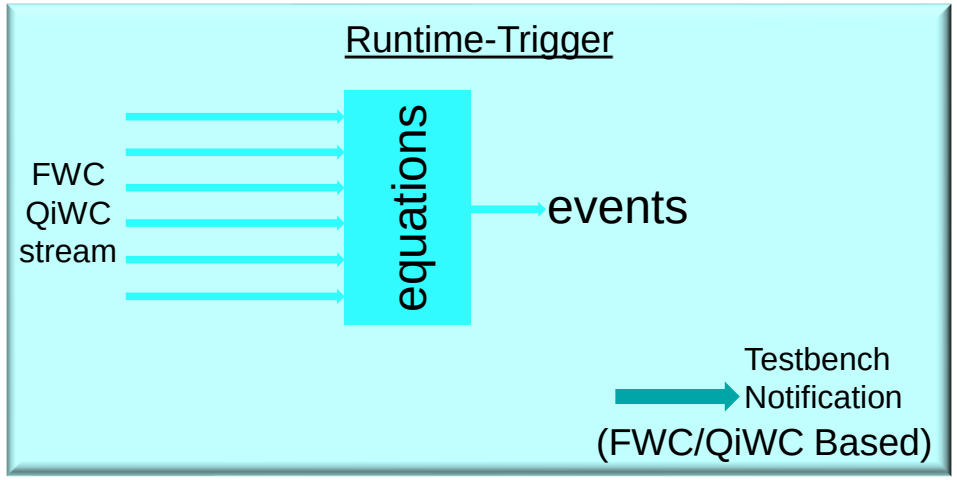
- Need to improve and Optimizing driver clock frequency in the Zebu emulator
- Analysis of Fetch Mode enabled and Disabled feature has also been explored to analyse driver clock frequency for specific design.

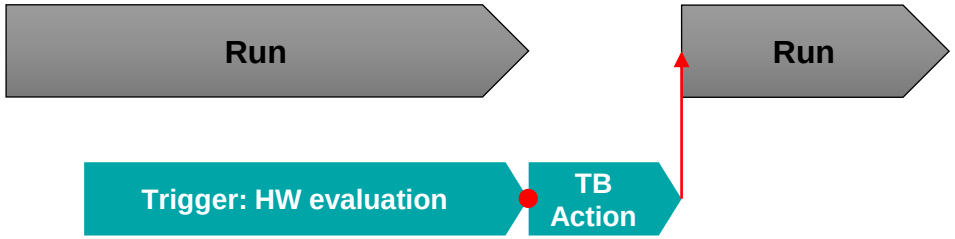
Improved ZS5 Driver Clock Freq with Fetch Mode Enable/Disable

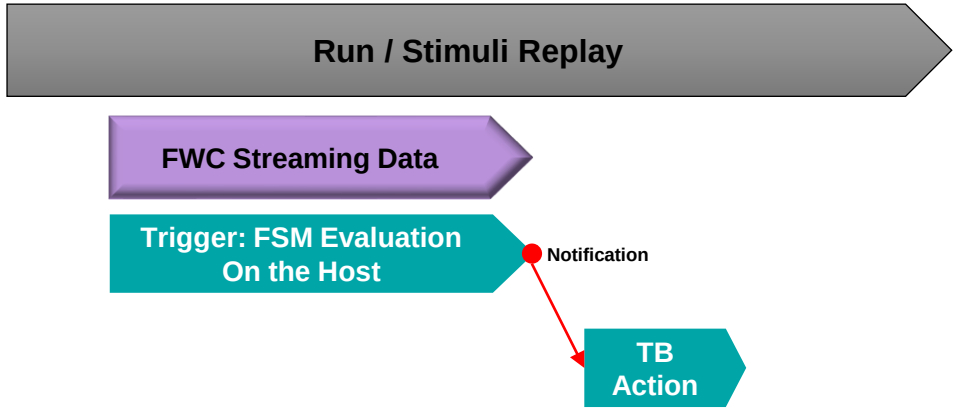


Run Time Optimization Strategies -Trigger Technologies for Debug









CONCLUSION

- Reduction of Compile Time
- Significant Performance Improvement
- On the fly Debug Mechanism

FUTURE SCOPE

- Gate Level Emulation + Power Aware Verification
- DFT Scenarios Validation
- IST Validation
- Complex XTORs Integration and Validation
- Power Estimation


THANK YOU
Our Technology, Your Innovation™