System to catch Implementation gotchas in the RTL Restructuring process

Anmol Rattan - ST Microelectronics
Satinder Singh Malhi - ST Microelectronics
Balwinder Singh Soni - ST Microelectronics
Anuj Kumar - Synopsys Inc
Navneet Chaurasia - Synopsys Inc
Sami Akhtar - Synopsys Inc

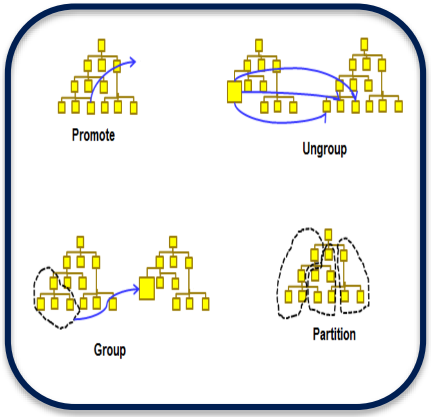
life.augmented

Motivation

- **RTL Re-structuring** - Design demanding Physical Design changes, Reuse of legacy IPs, Derivative Design turnaround time to market.
  - Physical design changes require the RTL to be restructured for LBIST logic insertion or to meet the Area, Timing and Power requirements
  - Derivative designs often require architectural changes to accommodate changes in bus width configuration, test logic insertion, power domain creation, memory configuration, sub-system hardening/partitioning, etc.

  Typically the restructuring of the RTL is either done manually or using home-grown automation scripts, or through a RTL Restructuring/Assembly EDA tool.

Hierarchy Manipulation & RTL Restructuring



- **Design issues induced :** Irrespective of the approach used, RTL restructuring inadvertently leads to inducing of design issues - often resulting in sub-optimal QoR during design implementation.
  Adds long engineering cycles to iteratively fix implementation bottle-necks and to achieve desired implementation results.

- **Long TAT to catch these Design issues :** Huge loss of project time and wasted engineering cycles
  - When the issues are caught during Formal Functional checking or Synthesis, as it involves iterations of debug/analysis and exchange across backend and frontend teams
  - Significant manual effort is involved to debug/fix the issues

life.augmented

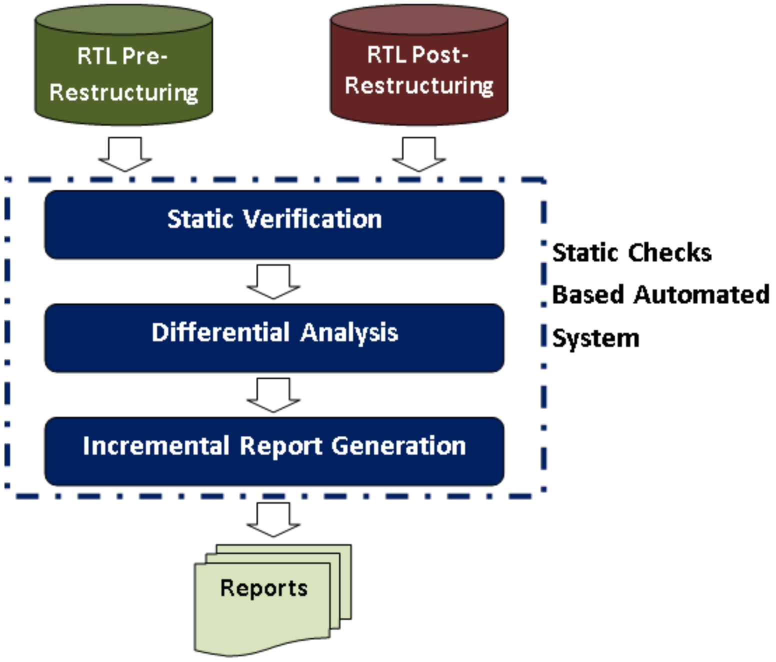
System to catch Implementation issues at RTL

- An automated system is proposed which uses a set of Static Rule Checks to catch all undesired design changes leading to connectivity issues upfront at the RTL stage
- Basically targets 2 versions of the design (RTL) – in this case pre/post-Restructuring

Flow Description

- *Pre-Restructuring RTL design is run through a set of predefined Static checks (which identify the potential issues that would impact backend implementation)*
- *The generated set of result/info is fed in along with the Post-Restructuring design and run through same set of checks*
- *The issues present in the pre-RTL Restructuring are masked*
- *Only the induced issues due to RTL Restructuring are presented in an easy to comprehend manner*

Primarily helping identify the incremental issues added due to the restructuring (change) process



- Typical Static check runs would typically flag huge number of violations (in 10's of thousands) for a full-SOC, thus making comprehensive checking/analysis undesirable for designers/integrators already pressed against release schedules
- Therefore, a pre-selected set of relevant checks, combined with the differential approach and simplistic presentation of the results – significantly brings down the data to analyze/debug – making it an appealing/practically useful solution

life.augmented

System to catch Implementation issues at RTL

- The checks are pre-selected from available Static Checks of Lint/CDC/Connectivity solutions in standard EDA industry tools
- The selection is based on industry experience, anticipation of potential issues, and desired validation of assumptions
- Requires no special design setup – and leverages the design setup already used for running static, synthesis, or simulation tools

Main checks in the Flow

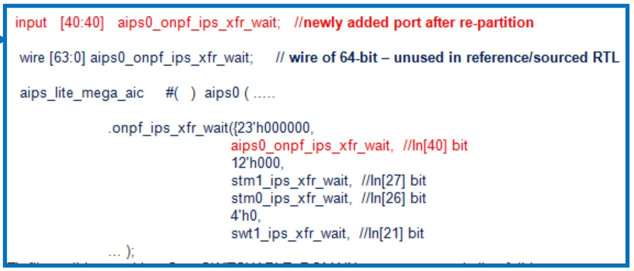
- *Assignment to input ports*
- *Instances having unconnected/floating ports, hanging nets, loaded but undriven inputs/outputs, unloaded but driven, inputs tied to constants, ..*
- *Multiply driven nets, read but not set inouts*
- *Width mismatches of ports/connecting nets*
- *Configuration mismatch with specification*
- *Value propagation checks to validate assumptions (values reaching/expected) at specific nodes/ports*
- *Unconstrained clocks/resets to validate clock/reset specifications*

Results

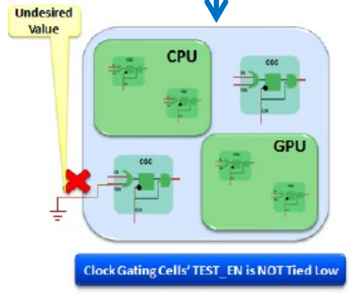
- The flow has been successfully tested on several versions of real multi-million gate Automotive SOC designs (done for 2-SoC designs)

Statistics of the designs	*Design#1*	*Design#2*
Gate Count	*~15 million NAND2 eq*	*~17 million NAND2 eq*
Std Cell Instance Count	*4.45 million instances*	*4.8 million instances*
Flop Count	*~659k*	*~690k*
Device Frequency	*180MHz*	*200MHz*
Typical Runtimes for the designs	***Design#1***	***Design#2***
Design Compilation Rt	*50~60 mins*	*~60 mins*
Synthesis Rt	*55~58 hrs (~3400 mins)*	*~55 hrs (~3300 mins)*
Formal Functional Rt	*~45 hrs (~2700 mins)*	*~47 hrs (~2800 mins)*
Proposed Static Checks based system Rt	***~10 mins***	***~11 mins***

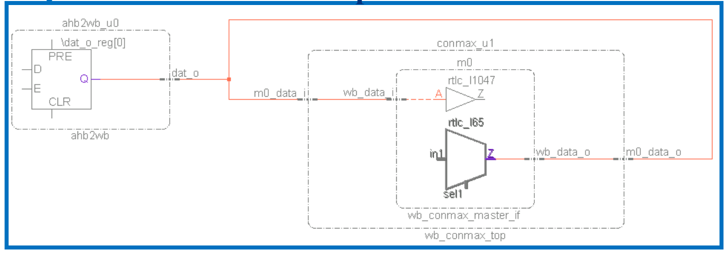
- Some typical issues which we were able to catch upfront using this flow
  - *Mismatch of widths (wrong signal definitions/connections)*
  - *Input ports were being written to (wrong assign statements induced),*
  - *Multiple drivers were found*
  - *Parsed Parameter values overridden by out of range random/default values*
  - *Buffer insertion in direct port to port connections*
  - *Connectivity issues in the test logic*
  - *Tied Low clock-gating logic preventing the propagation of the clock(s)*



```
input   [40:40]   aips0_onpf_ips_xfr_wait;   //newly added port after re-partition

wire [63:0] aips0_onpf_ips_xfr_wait;     // wire of 64-bit – unused in reference/sourced RTL

aips_lite_mega_aic     #(   )  aips0 ( .....

                .onpf_ips_xfr_wait({23'h000000,
                                    aips0_onpf_ips_xfr_wait,  //In[40] bit
                                    12'h000,
                                    stm1_ips_xfr_wait,  //In[27] bit
                                    stm0_ips_xfr_wait,  //In[26] bit
                                    4'h0,
                                    swt1_ips_xfr_wait,  //In[21] bit
                ... );
```



Multiple Drivers



Results/Conclusion

- Comparative TATs
  - *Issue caught at Synthesis* *: ~1 week+ (Syn Rt + exchange across backend/frontend design teams)*
  - *Issue caught at Formal Fn check* *: 2~3 days+ (Rt + debug/analysis)*
  - ***Proposed Static Checks based system*** ***: 10~20 mins (uses same setup)***

- Enables designers/system integrators to catch potential backend implementation & RTL Re-structuring issues early in the design cycle

- Saves a huge amount of time and iterations between backend and frontend teams
  - Enables timely delivery schedules
  - Fits into the sign-off flow from frontend team to backend team

- Automated flow ensures easy integration with the users current flow
  - requires no special setup or learning curve
  - Insignificant overhead in runtime – as this flow runs in ~10mins

- Increases confidence in the Restructured RTL and helps the frontend team focus on the job at hand – i.e. SOC Integration

The proposed Static Checks based system provides a huge value by catching significant issues upfront with an insignificant overhead in terms of runtime and requires no special setup

life.augmented