

# How Fast Can You Run SLEC For Verifying SIEMENS **Design Optimizations and Bug Fixes**

Jin Hou Francisco Chen Siemens EDA

Wenli Liang, Lina Guo Chunlin Wang Nokia Shanghai Bell Co., Ltd

Agenda	SLEC Flow For Bug Fixes And ECO					
<ul> <li>How to verify design optimization using SLEC</li> <li>How to verify bug fixes and ECO using SLEC</li> <li>How to verify the functional correctness of bug fixes using SLEC</li> <li>When and where to apply SLEC</li> </ul>	<u>Inputs:</u> DUT0 = original RTL DUT1 = DUT0 with ECO or bug fix Desired output:	Tie inputs together, und	constrained DUT1			

Match/Mismatch = is the behavior of DUT1

#### **SLEC Flow For Design Optimization**



### **Verify Synchronizer Insertion**

Verify synchronizer insertion hasn't injected any bug for module "dp\_phy\_icb" Before synchronizer insertion: After synchronizer insertion:

= (~icb\_ena\_traffic & auto\_kill\_us ) | kill\_us; wire us\_kill = (~icb\_ena\_traffic & auto\_kill\_ds ) | kill\_ds; wire ds\_kill = (~icb\_ena\_traffic & auto\_kill\_ext) | kill\_ext; wire ext\_kill wire traffic\_dead = us\_kill | ds\_kill | ext\_kill;

//------

wire cap\_mode = debug\_con[4] ; assign debug\_contrl[3:0] = debug\_con[3:0]; //-----

// Combinatorial Assignments

wire us kill wire = (~icb\_ena\_traffic & auto\_kill\_us ) | kill\_us; = (~icb\_ena\_traffic & auto\_kill\_ds ) | kill\_ds; wire ds\_kill wire ext\_kill = (~icb\_ena\_traffic & auto\_kill\_ext) | kill\_ext; wire traffic\_dead = us\_kill | ds\_kill | ext\_kill;

= debug\_con[4] ; wire cap mode assign debug\_contrl[3:0] = debug\_con[3:0]

// Combinatorial Assignments

//------

different only in the way I expect? (i.e. did I fix only what had to be fixed and nothing else got broken?)

#### Benefit:

cap\_mo cap st

correction e dbg\_gem\_ ds tc d ds tc gem e

is to gem s idle xor

lu\_color\_rem

lu linear gem lu\_que

lu queue ren

lu\_aes\_e lu assign Can eliminate weeks of sim. regressions



NOKIA

### Verify Bug Fixes For Module "tc\_gem\_frame"

- Bug: For a certain type of frames, the design could generate "sof", but didn't generate "eof". Only output "ds\_gem\_eof" was affected.
- Results: Only seconds to verify bug fix didn't damage good functions.

	cap_done		Mappings	S							▼ 7 X
	db_ram_wraddr		Start D	un (8	👌 Man Signals	H Filter: Type here <ctrl+e< th=""><th>ക</th><th>0</th><th></th><th></th><th></th></ctrl+e<>	ക	0			
	db ram wrdata		Start N			Type here <outres< td=""><td>- C\$)+</td><td>Ŭ</td><td></td><td></td><td></td></outres<>	- C\$)+	Ŭ			
	db_ram_wren	$\otimes$			Namo	Padius	Time		Spoc Signal	Impl Signal	
	ds gem aes ena	$\otimes$			Indine	Raulus	11110		Spec Signal	impi Signai	
<u> </u>	ds.gem_color_reman	$\otimes$	•	F	SLEC_output_11	21	1s		spec.ds_gem_eof	impl.ds_gem_eof	
rt	ds gem data	$\otimes$	•	P	SLEC output 1		1s	5	spec.cap done	impl.cap done	
lk	ds_gem_data_valid	8					10			implicate rom wroddr	
na	ds gem ena	$\otimes$		ų	SLEC_output_2		15	:	spec.db_ram_wraddr	impi.db_ram_wraddr	
en .	ds_gem_eof	$\otimes$	C	P	SLEC_output_3		1s	5	spec.db_ram_wrdata	impl.db_ram_wrdata	
ta .	ds gem linear gemid			P	SLEC output 4		1s		spec.db ram wren	impl.db_ram_wren	
	ds gem port id	$\otimes$					1		opeo.ub_run_men	implicib_loan_men	
na .	ds gem pti	$\otimes$	<b> </b>   <b> </b>	<u>p</u>	SLEC_output_5		1s	5	spec.ds_gem_aes_ena	impl.ds_gem_aes_ena	
na	ds_gem_queue	$\otimes$		P	SLEC_output_6		1s	5	spec.ds_gem_color	impl.ds_gem_color	
d .	ds gem queue remap	$\otimes$		<b>D</b>	SLEC output 7		15		spec ds dem color reman	implids dem color reman	
or .	ds gem sof	$\otimes$					13		spee.us_gem_color_lemap	implias_gem_color_temap	
ip .	ds_gem_soh	$\otimes$	•	<b>P</b>	SLEC_output_8		1s	5	spec.ds_gem_data	impl.ds_gem_data	
ie .	gem assigned sto	$\otimes$	- C	P	SLEC output 9		1s	5	spec.ds gem data valid	impl.ds gem data valid	
р	gem err stb	$\otimes$		10-11	SLEC output 10		Ωs		spec ds dem ena	implids dem ena	
	gem frame stb	$\otimes$		2			05		spec.us_gem_ena	Impl.us_gem_ena	
	gem_idle_stb	$\otimes$	•	<b>P</b>	SLEC_output_12		1s	5	spec.ds_gem_linear_gemid	impl.ds_gem_linear_gemic	1
	gem oos	$\otimes$			SLEC output 13		19		spec ds gem pld len	implids dem pld len	-
	lat_port_id	$\otimes$	- ·				13		spee.us_gem_pid_ien	implus_gem_pid_ien	-
	lu rd ena	_		$\mathbf{X}$							
_gem_f	rame		Iranscrip	M	appings		$\checkmark$				



#### Verifying The Functional Correctness Of The Bug Fixess

- PropCheck and simulation are the traditional ways to verify functional correctness
- SLEC is to verify equivalence between signal pairs
- Leverage it to verify functional correctness by providing SVA assumption as golden
  - Cut the incorrect output "ds\_gem\_eof" in the version before bug fix.
  - Write SVA assumption to constrain free "ds\_gem\_eof" (golden reference)
  - SLEC to compare the golden reference with the one in the bug fix version.

## Makefile

- Cut bad "spec\_ds\_gem\_eof" and constrain it as golden reference
- Compare {spec.ds\_gem\_eof impl.ds\_gem\_eof}

qverify -c -od log fix -do "\ slec configure -spec -d tc gem frame -work work spec; \ slec configure -impl -d tc gem frame -work work impl; \ netlist cutpoint spec.ds gem eof; \ Cut spec.ds\_gem\_eof vlog -sv properties.sv -mfcu -cuname sva bind; slec compile -cuname sva bind; \ Control spec.ds\_gem\_eof using SVA slec unmap -name SLEC output \*; \ assume property slec map {spec.ds\_gem\_eof impl.ds\_gem\_eof} -target; \

	*	Name	Radius	Time	Spec Signal	Impl Signal
0	F	SLEC_output_19	6	0s	spec.icb_dataout	impl.icb_dataout
0	R	SLEC_output_20	6	0s	spec.icb_interrupt	impl.icb_interrupt
0	E	SLEC_output_25	3	0s	spec.us_kill	impl.us_kill
0	-	CLEC autout 1		00	anaa aan mada	implean mode

• Waveforms of SLEC\_output\_25

+	cycle	3	-1	0	1	2	3	4			
ŧ	Primary Clocks			Primary Clocks							
	Property Signals			Property Signals							
	bec.us_kill	1									
	impl.us_kill ₪	0									
	ontrol Point Signals			Control Point Signals							
•	🕺 pec.icb_address	120	XXXX	(		120					
•	🕺 mpl.icb_address	120	XXXX			120					
•	斜 spec.icb_datain	200000	XXXXXXXX		0	χ		200000			
•	斜 impl.icb_datain	200000	XXXXXXXX		0	χ		200000			
	斜 spec.icb_rdnwr	0									
	🙌 impl.icb_rdnwr	0									
	🕺 spec.icb_strobe	1									
	斜 impl.icb_strobe	1			~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~						
	Contributors			Contributors							
0	Nonconcurrent			Nonconcurrent							
			St	art		re la					

Defined the delay between spec.us\_kill and impl.us\_kill, then the tool proved it. slec map {spec.us kill impl.us kill} -target -impl latency 1

## When And Where To Apply SLEC

- Run SLEC when designs are relatively stable
  - Use Lint or AutoCheck tools at early stage of RTL development
  - Use SLEC at late stage of RTL development
    - Design functions are stable

- Functional verification has been applied since one design version is the reference
- Run SLEC at module level where modifications are made
  - Most efficient and run time is fast.

Conclusions:	Module	Targets	Proven	Fired	Run time (second)
• Results:	us_bwc_crc_aid	37	20	17	17
	ds_phy_icb	40	37	3	2
	tc_gem_frame	27	26	1	1

- Questa SLEC is easy to setup, run and debug non-equivalence
  - Its automatic mapping and assume-guarantee features make setup and run fast.
- Save lots of simulation time to rerun tests after design optimization and bug fixes.
  - Simulation time hours vs Questa SLEC seconds in our test cases.