2022 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

MUNICH, GERMANY
DECEMBER 6 - 7, 2022

Overcoming System Verilog Assertions limitations through temporal decoupling and automation

Mattia De Pascalis, Xia Wu,
Matteo Vottero, Jacob Sander

SyoSil Systems on Silicon

accellera SYSTEMS INITIATIVE

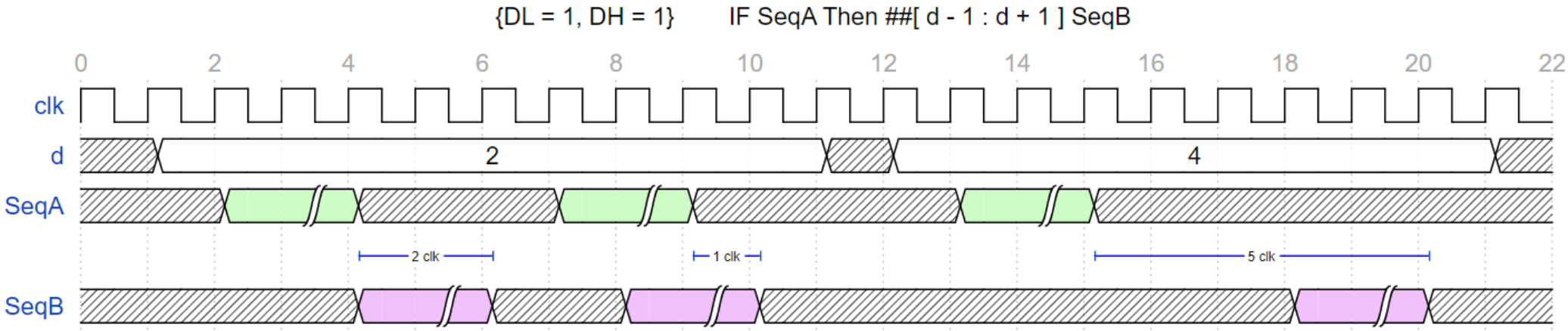
Background

- SVA is a de facto standard for Formal Verification
  - Completeness of logical relations
  - Timing relations between several events or sequences
  - Glue logic used to abstract class of problems

- Identify and automate a class of assertions
  - Common abstraction/glue logic
  - Decouple Assertion and glue logic
  - Automatic direct and inverted assertion generation

Class of problem(s) considered in the paper

- Temporal relationship between two “generic” sequences SeqA, SeqB
  - d is dynamic (reconfigurable)
  - Internal uncertainties DL, DH] (DL, DH constants and d – DL > 0)
  - IF SeqA THEN ##[d – DL : d + DH] SeqB



Problems discussed in the paper

- SVA language does not support dynamic delays
  - Need to abstract the problem
  - Need to develop an architecture that implement the solution
- Decouple Sequences, helper logic and assertions
  - Sequences shall be developed as “standalone” library
  - Shall be possible to try different type of helper logic models
  - It should be easy to debug
- Automatic generation of inverse assertion

Abstract and overcome SVA limitation

SeqA.triggered `|-> ##[d - DL : d + DH] SeqB.triggered` *[1]*

- **d** = dynamic delay, DL and DH constants
- Not supported in SVA as it is

- Assumptions:
  - Introduce SeqAD = SeqA delayed of **K** clock cycles
  - Substitute SeqA with SeqAD in [1]
  - Express new timing relationship between SeqAD and SeqB

Abstract and overcome SVA limitation

SeqAD.triggered `|-> ##[d - K - DL : d - K + DH] SeqB.triggered` *[2]*

- Make use of auxiliary logic such that K = **d** - DL

SeqAD.triggered `|-> ##[d-(d - DL)- DL : d-(d - DL)+DH] SeqB.triggered` *[3]*

SeqAD.triggered `|-> ##[0 : DL + DH] SeqB.triggered` *[4]*

- [4] does not contains dynamic delay anymore
- SeqAD can be modelled with auxiliary code
- [4] Can be expressed in SVA

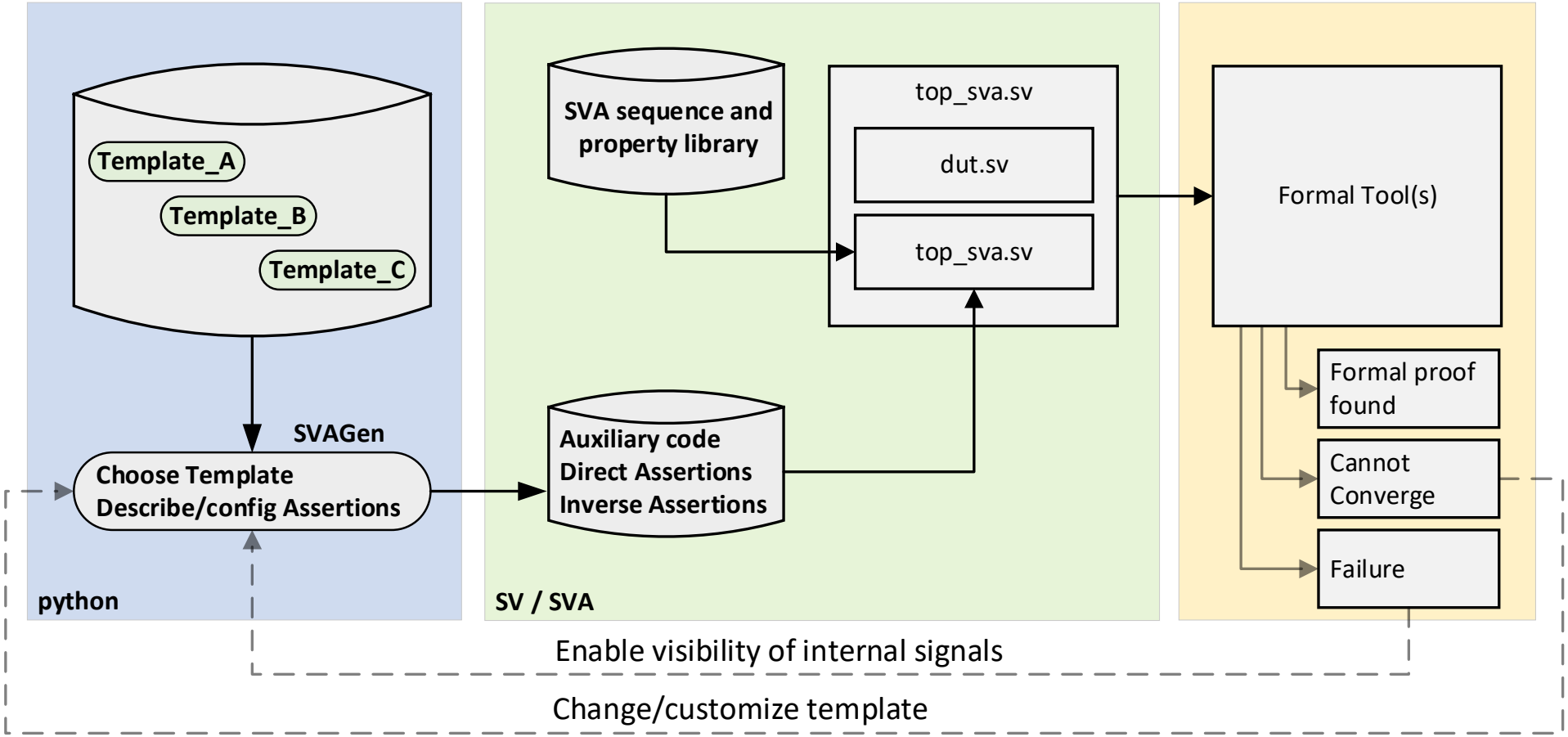
Problems discussed in the paper

- SVA language does not support dynamic delays
  - Need to abstract the problem
  - Need to develop an architecture that implement the solution
- Decouple Sequences, helper logic and assertions
  - Sequences shall be developed as “standalone” library
  - Shall be possible to try different type of helper logic models
  - It should be easy to debug
- Automatic generation of inverse assertion

SVAGen architecture

- Python framework
  - Templates to describe a specific auxiliary logic architecture
  - Describes timing relationship between SVA sequences
  - Configure each assertion with any of the available templates
  - Configure each assertion with higher/lower logic (help debug)
  - Enable/Disable automatic inverse assertion generation
- Available as opensource tool

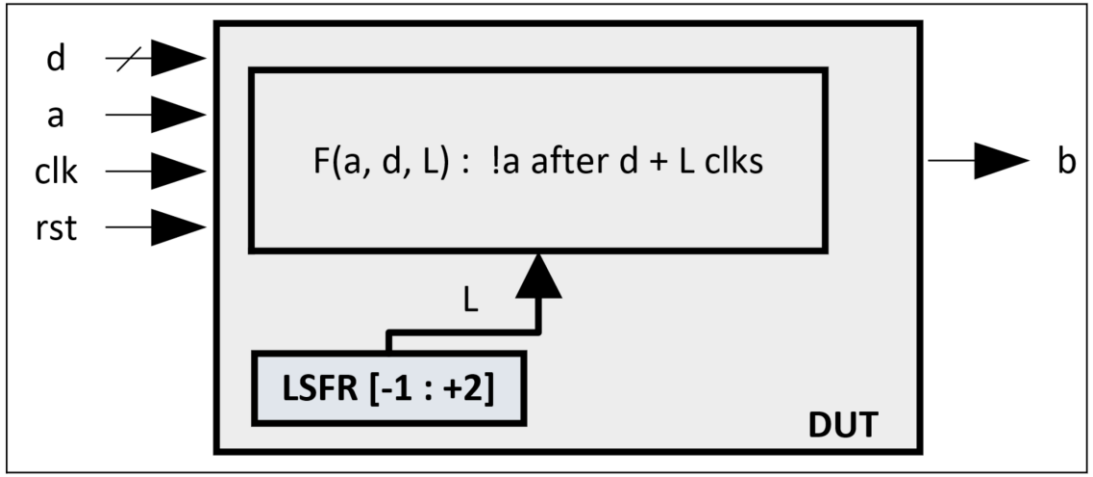
SVAGen architecture



Validation and test

- DUT definition
  - d : Reconfigurable delay
  - L : rnd in range [-1 : +2] (LSFR)
  - d – DL >= 0 → d >= 1

- SVA Sequence definition
  - Develop generic sequences
  - No timing relationship in sequences




```
sequence s_a;
  $rose(a);
endsequence: s_a;

sequence s_b;
  $fell(b);
endsequence: s_b;
```

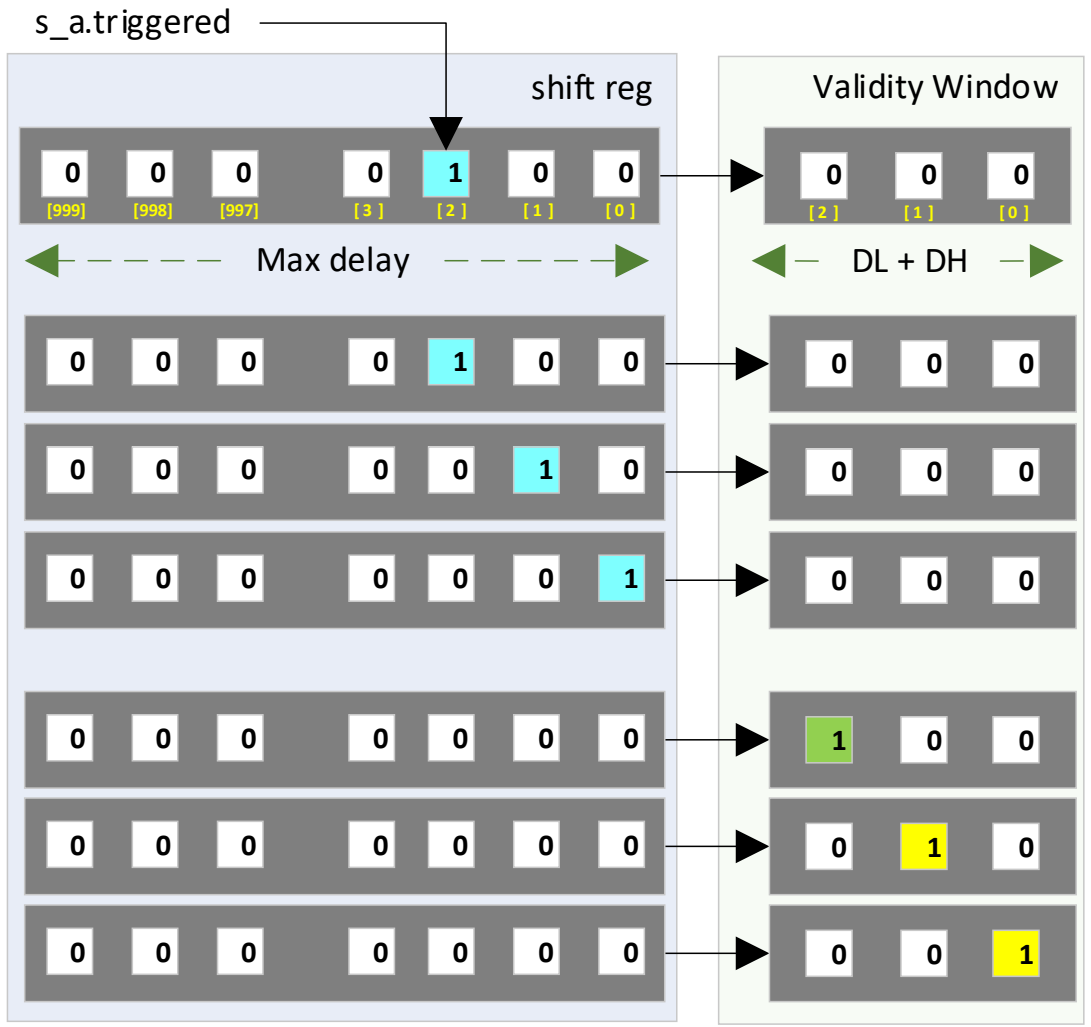
Example

- SVAGen configuration
  - Select template for dynamic delay
  - Configure assertion
  - Run SVAGen tool
  - Integrate output file into Formal TB

```
assertionModule =
 AssertionContainer(                          <-- SVAGen class
  Parser(                                     <-- SVAGen class
   "./sva_gen/templates/fifo/template.yaml",  <-- Template YAML
   "./sva_gen/templates/fifo/template.svh" ,  <-- Template SVH
   "out_macro.svh",                           <-- Output macro name
   "out_checkers.svh"))                       <-- Output checkers name

assertionModule.add_assertion(
 AssertionRangeDelay                          <-- Assertion type
 ( "rose_a_then_fell_b",                      <-- assertion name
  "s_a",                                      <--Sequence for precondition
  "s_b",                                      <-- Sequence for consequence
  "d",                                        <-- delay signal
  "1",                                        <-- ABS min uncertainty
  "2",                                        <-- ABS max uncertainty
  1000))                                      <-- max delay allowed
```

Example

- Shift Reg
  - Register s_a trigger in position [d – DL]
  - Advance every clk cycle 1 position
- Validity Window
  - First “event” activate direct assertion
  - Inv assertion looks at full Validity Window
- Helper logic generated by SVAGen
  - RangeDelay Template in this example



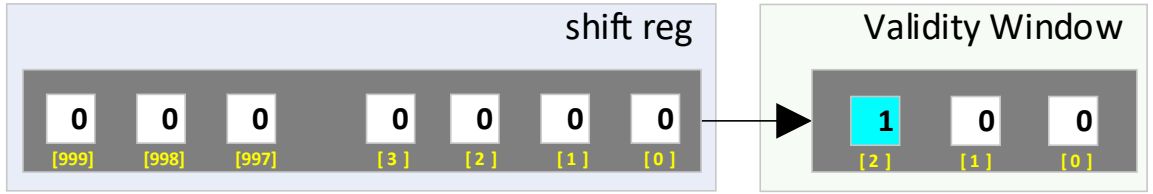
Example

- Auxiliary logic
  - Handle corner cases
  - Handle fifo/shift reg update
  - Handle Validity window update
  - Handle runtime addr calculation
  - Update all data structures

```
...
always @(posedge clk)
begin
 if(top.rst)
 begin
  rose_a_then_fell_b_fifo                 <= '0;
  rose_a_then_fell_b_window               <= '0;
  rose_a_then_fell_b_s1_triggered_d0 <= '0;
  rose_a_then_fell_b_s1_triggered_d1 <= '0;
  rose_a_then_fell_b_s1_triggered_d2 <= '0;
 end else
 begin
  rose_a_then_fell_b_window[`rose_a_then_fell_b_WINDOW_L - 1 : 0]
    <= rose_a_then_fell_b_window[`rose_a_then_fell_b_WINDOW_L : 1];
  if(32'(rose_a_then_fell_b_delay - `rose_a_then_fell_b_DL) > 2)
     rose_a_then_fell_b_window[`rose_a_then_fell_b_WINDOW_L] <= rose_a_then_fell_b_fifo[0];
  if(32'(rose_a_then_fell_b_delay - `rose_a_then_fell_b_DL) == 0)
     rose_a_then_fell_b_window[`rose_a_then_fell_b_WINDOW_L] <= s_a.triggered;
  if(32'(rose_a_then_fell_b_delay - `rose_a_then_fell_b_DL) == 1)
     rose_a_then_fell_b_window[`rose_a_then_fell_b_WINDOW_L] <= rose_a_then_fell_b_s1_triggered;
  if(32'(rose_a_then_fell_b_delay - `rose_a_then_fell_b_DL) == 2) rose_a_then_fell_b_window[`rose_a_then_fell_b_WINDOW_L]
     <= rose_a_then_fell_b_s1_triggered_d0;

  rose_a_then_fell_b_fifo <= rose_a_then_fell_b_fifo >> 1;
  rose_a_then_fell_b_fifo  [32'(rose_a_then_fell_b_delay - 1 - `rose_a_then_fell_b_DL)] <= s_a.triggered;
  rose_a_then_fell_b_s1_triggered_d0 <= rose_a_then_fell_b_s1_triggered;
  rose_a_then_fell_b_s1_triggered_d1 <= rose_a_then_fell_b_s1_triggered_d0;
  rose_a_then_fell_b_s1_triggered_d2 <= rose_a_then_fell_b_s1_triggered_d1;
 end
end
...
```

Example

- Assertions generated from SVAGen
  - Only static delay are used
  - Full proof with formal tool



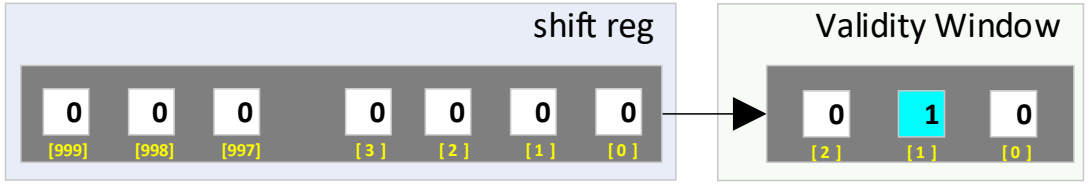

```
property p_rose_a_then_fell_b;
  validity_window[ 2 ] |-> ##[ 0 : `DL + `DH ] rose_a_then_fell_b_triggered;
endproperty: p_rose_a_then_fell_b;

a_rose_a_then_fell_b       : assert property(`CLK_SEQ  p_rose_a_then_fell_b);
```

Problems discussed in the paper

- SVA language does not support dynamic delays
  - Need to abstract the problem
  - Need to develop an architecture that implement the solution
- Decouple Sequences, helper logic and assertions
  - Sequences shall be developed as “standalone” library
  - Shall be possible to try different type of helper logic models
  - It should be easy to debug
- Automatic generation of inverted assertion

Example

- Assertions generated from SVAGen
  - Inv assertion exploits Validity Window
  - Validity Window stores info about past events
  - Full proof with formal tool




```
property p_rose_a_then_fell_b_inv;
  rose_a_then_fell_b_triggered |-> validity_window != '0;
endproperty: p_rose_a_then_fell_b_inv;

a_rose_a_then_fell_b_inv : assert property(`CLK_SEQ  p_rose_a_then_fell_b_inv);
```

Validation and results

- Time to find proof depends on 2 main factors:
  - Interval of delay
  - Range of delay
  - Always used uncertainties of +/- 3

Interval		10		Interval		20		Interval		50		Interval		100	
Range		Time (sec)		Range		Time (sec)		Range		Time (sec)		Range		Time (sec)	
Min	Max	A_D	A_INV	Min	Max	A_D	A_INV	Min	Max	A_D	A_INV	Min	Max	A_D	A_INV
0	10	1	0,2	0	20	1,3	1,1	0	50	1,7	2,3	0	100	4,4	3,8
11	20	1	1	21	40	1,4	1,2	51	100	2,6	4,2	101	200	12,4	13,6
21	30	1	1,6	41	60	1,4	1,5	101	150	7,6	6,8	201	300	42,6	36,2
	...				...				...				...		
971	980	27,5	12,6	941	960	13,9	24,5	851	900	101,5	203,5	701	800	334,4	230,4
981	990	28,2	11,3	961	980	12,6	15,7	901	950	56,9	47,6	801	900	136,6	283,4
991	1000	11,6	13,8	981	1000	16,3	16,7	951	1000	32,4	27,7	900	1000	126,3	48,2
Total		1352	843			390,8	505,8			675,67	973,67			981	1313,8

Conclusion

- Automatic handling of dynamic delay
- Automatic handling of inverse assertion
- Easy framework to switch auxiliary logic
- Reusable methodology for formal and simulation
- Tool is open sourced and available for download at this link

Future developement

- Investigation for more efficient auxiliary logic
- Refactoring of SVAGen for better reusability

Questions