Automated Specification Driven Verification by Generation of SystemVerilog Assertions

Ferdinando Pace
SENSIRION AG

***Abstract*-In this paper, a tool is described that automatically generates both testbench and SystemVerilog assertions checking the signal timing specified in Excel format. Such timing specifications can be dependent on programmable on-chip parameters (e.g. A/D converter settling time, chopping frequency, filter oversampling ratio) and the generated SystemVerilog assertions cover exhaustively all parameter combinations. Multi-clocked timing specifications are also supported. Moreover, the method used to specify the timing provides a compact description for long timing sequences, e.g. periodic signals. The tool is used to verify signal timings specified to drive an A/D converter inside a sensor chip developed by SENSIRION. In general, this tool can be used to verify signal timing specifications of analog/mixed-signal or digital IPs.**

I. INTRODUCTION

Analog/mixed-signal IPs (e.g. A/D converters) often require complex control signal timing sequences. Such timing sequences can also be dependent on programmable on-chip parameters (e.g. settling time, chopping frequency and oversampling ratio) and more than one clock source (e.g. switched-capacitor filter where several clock edge combinations are used).

Figure 1 illustrates a practical example of a timing sequence for an A/D converter using 4 phases (IDLE, RESET, SETTLING, CONVERSION) in which 3 signals (Reset, Settling, Chop) are driven according to timing specification expressed in clock cycles. A phase in which duration is not a single number indicates a programmable parameter: Settling can last 2 or 4 clock cycles, period of Chop is 4 or 8 clock cycles, Chop can toggle for 32 or 64 clock cycles.

Clearly, the verification effort increases significantly depending on the number of parameter combinations allowed.

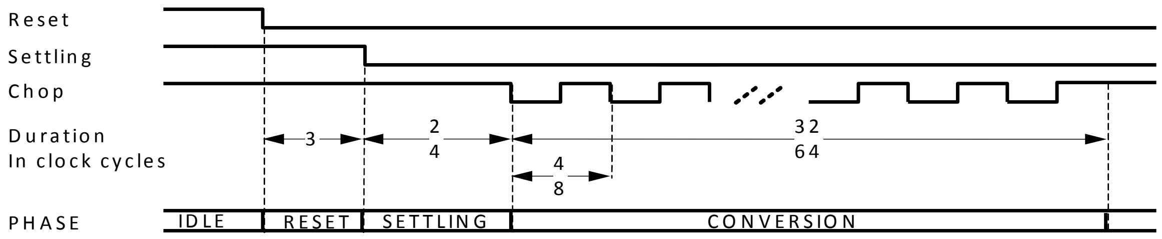


Figure 1. Example of timing sequence
required by an A/D converter

Typically, the verification challenge involves the following steps:

- The Specification Writer understands and writes the IP timing sequence in the appropriate format.
- The Design Engineer implements the RTL code that generates the IP timing sequence.
- The Verification Engineer verifies that the signal timing generated by the RTL code fulfills the specification for all possible parameter combinations.

This process is very critical because it relies on flawless communication and depends on error prone subjective generation and interpretation of the specification that could lead to unverified timing sequences with the risk of undetected design errors.

Several verification methods are available:

- Visual comparison of the timing diagram generated by the RTL design with the timing diagram produced by the Specification Writer or signal waveforms produced by the IP level testbench: this visual method is error prone by nature.
- Semi-automated comparison with specific EDA tools (e.g. SimCompare tool by Cadence) of the simulation database generated by the RTL design with the simulation database generated by the IP level testbench: the comparison set-up is tedious. Moreover, log files generated by EDA tools have to be analyzed carefully otherwise there is the risk of misinterpretation of warning messages that hide key comparison information.
- Assertions development (e.g. Verilog, SystemVerilog, PSL) to verify the timing diagram: the handcrafted development of assertions is time consuming especially if the number of assertions is dependent on programmable on-chip parameters.

Obviously, overlooked specification errors and wrong analysis of simulation database comparisons can cause costly silicon respins.

In order to prevent this, we need a rigorous flow from the objective specification to the verification.
This flow is ideally implemented using an automated tool which satisfies the following requirements:

1. Unique, objective, unambiguous description of specification
2. User friendly, ideally an automated flow once the specification is available
3. Cheap, open source tools
4. Easy to integrate into existing project simulation environment
5. Flexible, it accepts programmable on-chip parameters, multiple clock sources
6. Exhaustive, all possible combinations for programmable on-chip parameters are covered
7. Self-checking, automatically generated assertion statements can be tested stand-alone before being integrated into the simulation environment

This paper proposes a tool which widely satisfies above requirements. It consists of the following sections:
Section II gives a description of the tool. Section III shows a few signal timing scenarios together with the corresponding SystemVerilog assertions and the testbench waveforms produced by the tool. Section IV analyzes the results achieved using the tool. Section V is the final summary.

II. TOOL DESCRIPTION

This section describes the ATG ("Assertion and Testbench Generator") tool. A flow diagram of ATG is given in Figure 2.

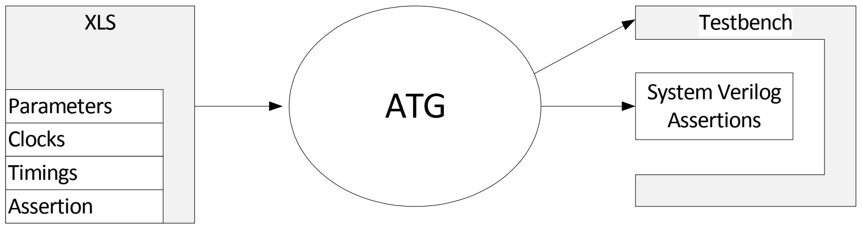


Figure 2. Flow diagram

In summary, the tool accepts specifications expressed with one or more files written in the Microsoft Excel format (XLS) to generate as outputs a Verilog testbench and SystemVerilog assertions.

II a) Inputs to ATG

As input, ATG requires an XLS file composed of 4 sheets:

1) The Parameters sheet describes all signals which affect the specification of a signal inside the IP. Table 1 is a Parameters sheet example.

REGISTER HIERARCHY	`DACQ.DAcqCfg2RegxDP	`DACQ.DAcqCfg1RegxDP	`DACQ.DAcqAdvCfgRegxDP	`DACQ.DAcqAdvCfgRegxDP
SIGNAL NAME	SE	OS	CH	CM
SIGNAL WIDTH		[2:0]	[1:0]	
Values	0	0	0	0
	1	1	1	
		2	2	
		3		
		4		
		5		
		6		
		7		

Table 1. Parameters sheet example

2) The Clocks sheet describes clocks which affect the signal specifications. Table 2 is a Clocks sheet example and Figure 3 shows the corresponding waveforms.

	PClkxCI	DclkxCI
phase delay	0ns	1ns
high pulse	5ns	7ns
low pulse	5ns	3ns

Table 2. Clocks sheet example

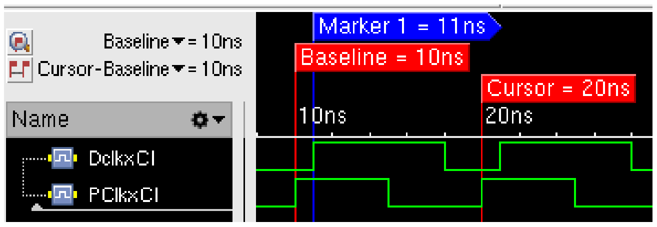


Figure 3. Clocks sheet waveforms generate by Table 2

3) The Timings sheet describes the timing sequence specification for each signal. Below follows a description of the sheet structure. An example is given in Table 3.

- The 1st column describes the boolean condition under which the SystemVerilog assertion property terminates (DISABLE condition).
- The 2nd column describes the boolean condition under which the SystemVerilog assertion property triggers (TRIGGER condition)
- The 3rd column describes the label used by ATG to build SystemVerilog assertion for a group of rows. This label can be omitted if a group of rows is not required.
- The 4th column describes the repetition factor for the group of rows specified in the 3rd column. This value can be omitted if a group of rows is not required. All consecutive rows which have the same values for both 3rd and 4th columns are grouped together to build the SystemVerilog assertion. The following values are valid for the repetition factor (please refer to [1] for more details about valid SystemVerilog assertion constructs):
  - Sequence repetition expression allowed by SystemVerilog language:
    - [* N] where N is a positive integer
    - [* N : M] where N, M are positive integers
    - [* N : $] where N is a positive integer
    - [= N] where N is a positive integer
    - [-> N] where N is a positive integer
  - Arithmetical function of parameters defined in the Parameters sheet.

- The 5th column describes the label used by ATG to build SystemVerilog assertion for the corresponding row. This label can be also omitted.
- The 6th column describes the repetition factor for the corresponding row. The same values as described for the 4th column are valid.
- The columns from 7th to the one before last describe the signal values for the current row.
- The last column describes the event at which:
    - The signal values for the current row are generated by the testbench (launching event)
    - The signal values for the previous row are checked by the SystemVerilog assertion property (capturing event).

  The concept is that the signal values generated by the testbench on the launching event of the current row are checked by the assertion property on the capturing event of the next row. This convention is introduced to allow the specification of multi-clocked timing diagrams where a signal launched on a clock event is checked on the following clock event. The clocks used by launching and capturing events may also be different.

DISABLE	TRIGGER	Group Name	Group Value	Row Name	Row Value	Chop	Settling	Reset	EVENT
Reset==1	$fell(Reset)								@(posedge Clk)
					[* 3]	1'b1	1'b1	1'b0	@(posedge Clk)
					5*(SE+1)	1'b1	1'b0	1'b0	@(posedge Clk)
		conversion	2**(2+CH+OS)	chop_low	2**(2-CH)	1'b0	1'b0	1'b0	@(posedge Clk)
		conversion	2**(2+CH+OS)	chop_high	2**(2-CH)	1'b1	1'b0	1'b0	@(posedge Clk)
									@(posedge Clk)

Table 3. Timings sheet example referred to Parameters sheet described in Table 1

4) The Assertion sheet is the prefix name assigned to the SystemVerilog assertion property.

II b) The ATG tool implementation

The ATG tool is composed of two scripts (see Figure 4) written in the Python language:

1. Assertion.py generates a SystemVerilog coverage module that contains all SystemVerilog assertions generated. The TRIGGER condition extracted from XLS file is used to build the assertions. This module is recommended to be plugged into the IP or DUT level simulation environment.
2. Testbench.py generates a Verilog testbench that calls Assertions.py to generate the SystemVerilog assertion properties and, additionally, produces the stimuli to trigger and verify those properties. An artificial TRIGGER condition (Assertion_Trigger signal, see Figure 6) with non-overlapped SystemVerilog implication operator ("|=>") is imposed by the script to build the assertions such that it is easier to verify the assertions without the burden to generate the TRIGGER condition specified in XLS file. The testbench is very useful if the user would like to debug the generated assertions with the help of signal waveforms. Moreover, the signal waveforms generated by the testbench can be used as snapshot for reference timing specifications.

The tool accepts more than one XLS file as a specification if the XLS files differ in the timing sheet only. This can be very useful to collect several timing specifications in a single module coverage (or a single testbench) file.

Below are given examples about how to execute the scripts:

- Assertion.py –i DSP –x Specification1.xls –o CoverageDSP.sv
- Testbench.py –i DSP –x Specification1.xls –x Specification2.xls –o Testbench.sv

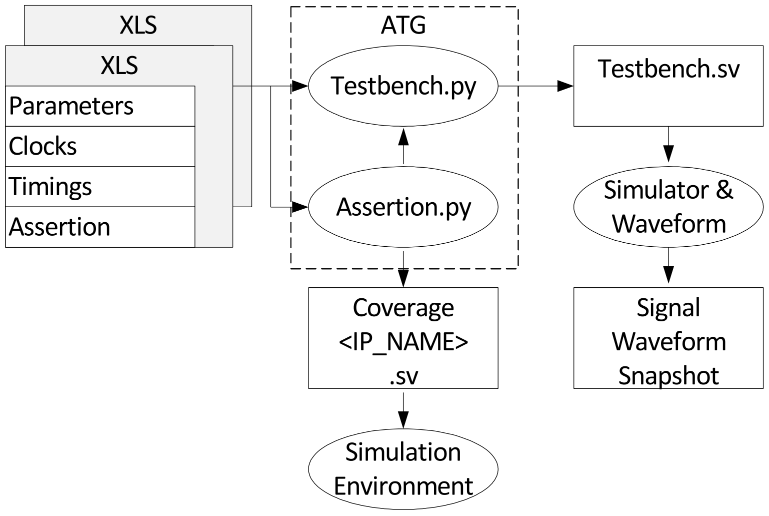


Figure 4. Detailed tool flow diagram

II c) Outputs from ATG

Figure 5 is an example of a SystemVerilog coverage module that contains all the SystemVerilog assertions generated with ATG. Figure 6 is an example of signal waveforms produced by the testbench generated by ATG.

```
module coverageDSP;
 import DSP_RTL_pkg::*;
//Assertion below extracted from Specification file ./ADCChopping.xls
property A_ADC_CHOP_DSP_Del_0__Osr_0__Chop_0__;
disable iff (`DSP.A[0]==1)
(@(posedge `DSP.Clk) ( $fell(`DSP.A[0]) && `DSP.Del==0 && `DSP.Osr==0 && `DSP.Chop==0 ) |->
 @(posedge `DSP.Clk)  ( `DSP.A[2]==1'b1 && `DSP.A[1]==1'b1 && `DSP.A[0]==1'b0 ) [* 2] ##1
 @(posedge `DSP.Clk)  ( `DSP.A[2]==1'b1 && `DSP.A[1]==1'b0 && `DSP.A[0]==1'b0 ) [* 3] ##1
(@(posedge `DSP.Clk)  ( `DSP.A[2]==1'b0 && `DSP.A[1]==1'b0 && `DSP.A[0]==1'b0 ) [* 4] ##1
 @(posedge `DSP.Clk)  ( `DSP.A[2]==1'b1 && `DSP.A[1]==1'b0 && `DSP.A[0]==1'b0 ) [* 4])[*5]
);
endproperty
assert property A_ADC_CHOP_DSP_Del_0__Osr_0__Chop_0__
endmodule : coverageDSP
```

Figure 5. SystemVerilog Coverage Module Example

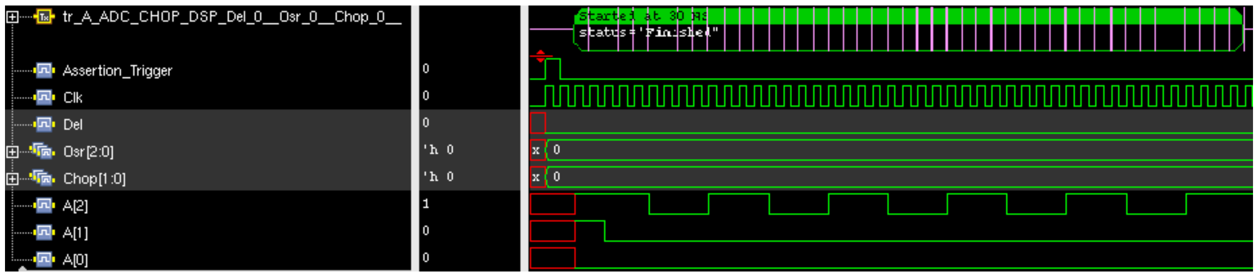


Figure 6. Signal waveforms example related to Figure 5

III. Application Examples

This section describes two example use cases suitable for the application of ATG.

III a) Example 1

A single clock (CK1) is used. This clock has a period of 10ns and a duty cycle of 50% (see Table 4). In Table 5 are specified three parameters (SE, CV and CH). A repetition factor (2**CV) for a group of 2 rows is used in Table 6 to describe a periodic signal (A[2]) that has a duty cycle dependent on a parameter (CH).

	CK1
phase delay	0ns
high pulse	5ns
low pulse	5ns

Table 4. Clocks sheet

REGISTER HIERARCHY	`DSP	`DSP	`DSP
SIGNAL NAME	SE	CV	CH
SIGNAL WIDTH		[2:0]	[2:0]
Values	0	1	3
	1	2	5

Table 5. Parameters sheet

DISABLE	**TRIGGER**	**Group Name**	**Group Value**	**Row Name**	**Row Value**	**A[2]**	**A[1]**	**A[0]**	**EVENT**
`DSP.A[0]==1	$fell(`DSP.A[0])					1'b1	1'b1	1'b1	@(posedge `DSP.CK1)
				reset	[* 2]	1'b1	1'b1	1'b0	@(posedge `DSP.CK1)
				settling	SE+3	1'b1	1'b0	1'b0	@(posedge `DSP.CK1)
		Conversion	2**CV	chop_low	10-CH	1'b0	1'b0	1'b0	@(posedge `DSP.CK1)
		Conversion	2**CV	chop_high	CH	1'b1	1'b0	1'b0	@(posedge `DSP.CK1)
									@(posedge `DSP.CK1)

Table 6. Timings sheet

ATG generates the following SystemVerilog assertion properties from the Parameters and Timings sheets above:

1. A_CHOP_DSP_SE_0__CV_1__CH_3__
2. A_CHOP_DSP_SE_0__CV_1__CH_5__
3. A_CHOP_DSP_SE_0__CV_2__CH_3__
4. A_CHOP_DSP_SE_0__CV_2__CH_5__
5. A_CHOP_DSP_SE_1__CV_1__CH_3__
6. A_CHOP_DSP_SE_1__CV_1__CH_5__
7. A_CHOP_DSP_SE_1__CV_2__CH_3__
8. A_CHOP_DSP_SE_1__CV_2__CH_5__

Figure 7 shows the SystemVerilog code generated by ATG for one (A_CHOP_DSP_SE_0__CV_2__CH_3__) of the SystemVerilog assertion properties above:

```
property A_CHOP_DSP_SE_0__CV_2__CH_3__;
disable iff (`DSP.A[0]==1)
(@(posedge `DSP.CK1)  ( $fell(`DSP.A[0]) && `DSP.SE==0 && `DSP.CV==2 && `DSP.CH==3 ) |->
 @(posedge `DSP.CK1)   ( A[2]==1'b1 && A[1]==1'b1 && A[0]==1'b0 ) [* 2] ##1
 @(posedge `DSP.CK1)   ( A[2]==1'b1 && A[1]==1'b0 && A[0]==1'b0 ) [* 3] ##1
(@(posedge `DSP.CK1)   ( A[2]==1'b0 && A[1]==1'b0 && A[0]==1'b0 ) [* 7] ##1
 @(posedge `DSP.CK1)   ( A[2]==1'b1 && A[1]==1'b0 && A[0]==1'b0 ) [* 3])[*4]
);
endproperty
```

Figure 7. SystemVerilog assertion code

Figure 8 shows the testbench waveforms generated by ATG that verifies the SystemVerilog assertion code in Figure 7.

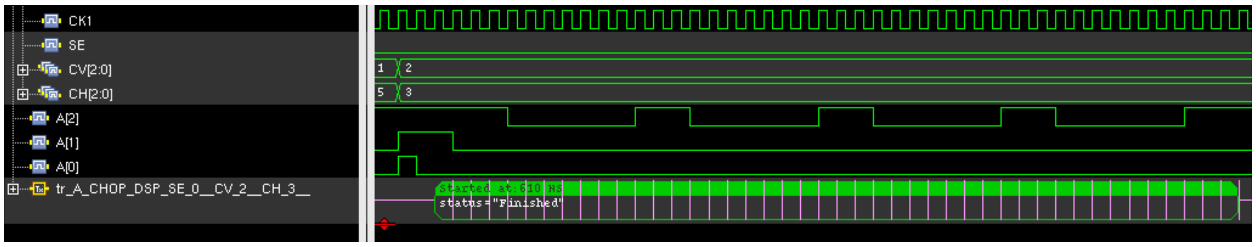


Figure 8. Testbench waveforms

III b) Example 2

Two clocks (CK1 and CK2) are used (see Table 7). The two clocks have the same period but a different duty cycle. A repetition factor (CH) for a group of 6 rows is used in Table 9 to describe the signals generated by the clocks.

	CK1	CK2
phase delay	0ns	1ns
high pulse	5ns	7ns
low pulse	5ns	3ns

Table 7. Clocks sheet

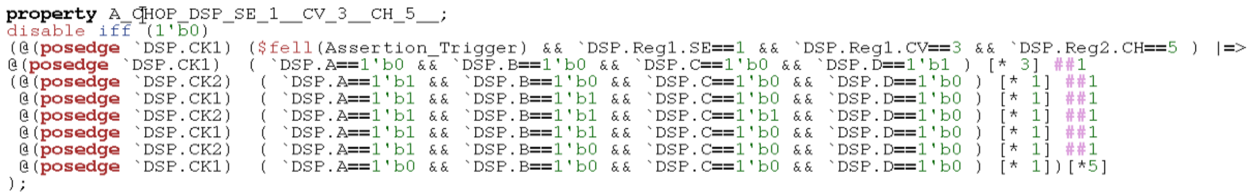
REGISTER HIERARCHY	`DSP.Reg1	`DSP.Reg1	`DSP.Reg2
SIGNAL NAME	SE	CV	CH
SIGNAL WIDTH		[2:0]	[2:0]
Values	1	3	5

Table 8. Parameters sheet

DISABLE	TRIGGER	Group Name	Group Value	Row Name	Row Value	`DSP.A	`DSP.B	`DSP.C	`DSP.D	EVENT
1'b0	$rose(`DSP.D)					1'b0	1'b0	1'b0	1'b0	@(posedge `DSP.CK1)
				ALLZERO	CV	1'b0	1'b0	1'b0	1'b1	@(posedge `DSP.CK1)
		Conversion	CH	RISE_A	SE	1'b1	1'b0	1'b0	1'b0	@(posedge `DSP.CK1)
		Conversion	CH	RISE_B	SE	1'b1	1'b1	1'b0	1'b0	@(posedge `DSP.CK2)
		Conversion	CH	RISE_C	SE	1'b1	1'b1	1'b1	1'b0	@(posedge `DSP.CK1)
		Conversion	CH	FALL_C	SE	1'b1	1'b1	1'b0	1'b0	@(posedge `DSP.CK2)
		Conversion	CH	FALL_B	SE	1'b1	1'b0	1'b0	1'b0	@(posedge `DSP.CK1)
		Conversion	CH	FALL_C	SE	1'b0	1'b0	1'b0	1'b0	@(posedge `DSP.CK2)
										@(posedge `DSP.CK1)

Table 9. Timings sheet

ATG generates a single SystemVerilog assertion property (A_CHOP_DSP_SE_1__CV_3__CH_5__) from the Parameters and Timings sheets. Figure 9 shows the SystemVerilog assertion code generated by ATG. Figure 10 shows the testbench waveforms generated by ATG to verify the SystemVerilog assertion code.



```
property A_CHOP_DSP_SE_1__CV_3__CH_5__;
disable iff (1'b0)
(@(posedge `DSP.CK1) ($fell(Assertion_Trigger) && `DSP.Reg1.SE==1 && `DSP.Reg1.CV==3 && `DSP.Reg2.CH==5 ) |=>
@(posedge `DSP.CK1)  ( `DSP.A==1'b0 && `DSP.B==1'b0 && `DSP.C==1'b0 && `DSP.D==1'b1 ) [* 3] ##1
(@(posedge `DSP.CK2)  ( `DSP.A==1'b1 && `DSP.B==1'b0 && `DSP.C==1'b0 && `DSP.D==1'b0 ) [* 1] ##1
 @(posedge `DSP.CK1)  ( `DSP.A==1'b1 && `DSP.B==1'b1 && `DSP.C==1'b0 && `DSP.D==1'b0 ) [* 1] ##1
 @(posedge `DSP.CK2)  ( `DSP.A==1'b1 && `DSP.B==1'b1 && `DSP.C==1'b1 && `DSP.D==1'b0 ) [* 1] ##1
 @(posedge `DSP.CK1)  ( `DSP.A==1'b1 && `DSP.B==1'b1 && `DSP.C==1'b0 && `DSP.D==1'b0 ) [* 1] ##1
 @(posedge `DSP.CK2)  ( `DSP.A==1'b1 && `DSP.B==1'b0 && `DSP.C==1'b0 && `DSP.D==1'b0 ) [* 1] ##1
 @(posedge `DSP.CK1)  ( `DSP.A==1'b0 && `DSP.B==1'b0 && `DSP.C==1'b0 && `DSP.D==1'b0 ) [* 1])[*5]
);
```

Figure 9. SystemVerilog assertion code

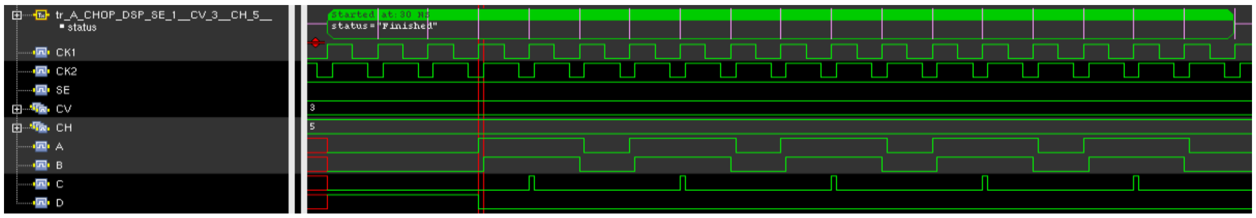


Figure 10. Testbench waveforms

IV. CONCLUSIONS

In this paper, I developed and used ATG to verify the signal timings specified to drive an A/D converter inside a sensor chip. The programmable parameters affecting this A/D converter signals were the settling time, the chopping frequency, the oversampling ratio, the conversion mode (unlimited or single shot) and a single clock source. The number of parameter combinations is 128. ATG generated the SystemVerilog properties covering all combinations that were verified with both the testbench generated by ATG and top level simulations.

ATG was used to produce both example test cases described in the section III with the intention to show how to use ATG in different signal timing scenarios.

The next step is the deployment of ATG to the digital and analog designers in order to collect their feedback for the usability of the tool and further improvements required.

I believe that if a single clock source is required for the specification, ATG is mature enough to specify and verify analog/mixed-signal circuits. Moreover, ATG can also be used to build assertion properties and SystemVerilog coverage modules required for IP functional verification.

V. SUMMARY

In this paper, the tool ATG was presented which provides automated specification driven verification by generation of SystemVerilog assertions from parametric timing specification.

It has been shown that the use of ATG to verify complex signal timing sequences can significantly reduce the verification effort covering exhaustively the specification given.

ACKNOWLEDGMENT

I would like to thank Daniel Mueller and Samuel Fuhrer for their very careful and valuable review of the paper.

REFERENCES

[1] B. Cohen, S. Venkataramanan, A. Kumari and L. Piper, “SystemVerilog Assertion Handbook, 3rd Edition” pp. 23,144