2014 DESIGN AND VERIFICATION DVCON CONFERENCE AND EXHIBITION EUROPE

Data path verification on cross-domain with formal scoreboard

Liu Jun, Intel Mobile Communications, Munich, Germany (jun.b.liut@intel.com)

***Abstract*—In today's complex System on Chip (SOC) designs, multiple asynchronous clock domains are increasingly getting deployed. More and more data are transacted from one unit to another, crossed different clock domains. Hence, formal verification (FV) methodologies are required for exhaustive verification. The traditional formal property verification (FPV) is not ideally suited for data-path intensive designs. Additionally, FPV is highly dependent on the quality of the coded properties. Thus, instead of going by the usual flow of assertion based FPV, we explored the use of pre-tested, automated and data-path optimized FV Proof Accelerators (PAs) provided by the EDA FV tool. Use of these PAs expedite the verification of Clock Domain Crossing (CDC) based designs by empowering the user to do quick setup of the formal environment, by providing increased proof convergence and by eliminating the risk of human errors by automating property generation. The formal approach described here is generic enough to be easily applied to a wide range of CDC-based designs.**

Keywords—functional verification, formal verificaion, data path verification; clock domain cross;

I. INTRODUCTION

The solution is used to verify end-to-end data integrity across the data path. In case of applying traditional FPV for such kind of checking, usually the properties themselves require a large memory to verify the design intent. The Scoreboard is designed to overcome these types of problems and make formal verification possible in situations that would otherwise be hard to approach with formal techniques.

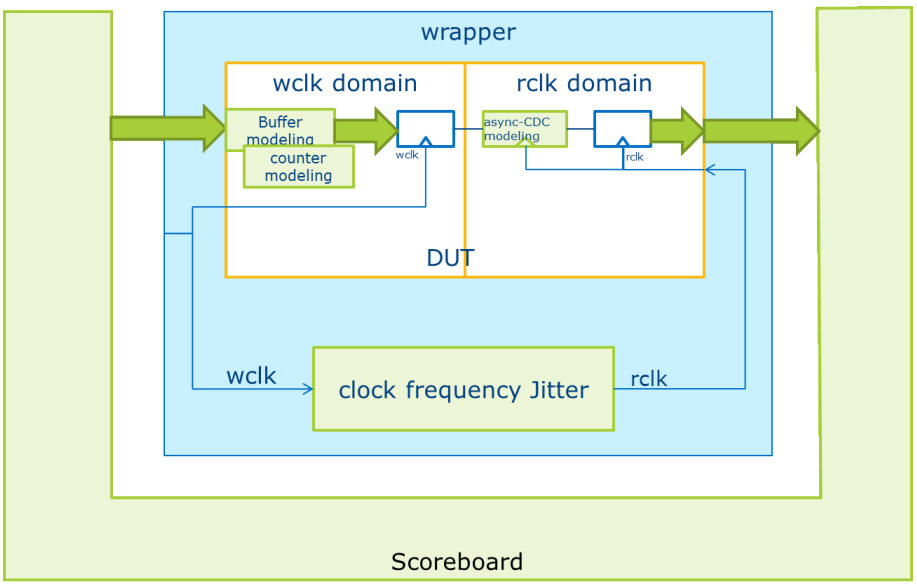

Figure 1. Formal score board solution for verification data path

The figure 1 shows the formal scoreboard solution for verification of data path. The DUT (Design Under Test) is bind with scoreboard via a Verilog wrapper. The scoreboard will verify that data will be transmitted properly. In the wrapper there is a clock frequency jitter, which could generate two different clock signals driving the two clock domains of DUT. In the DUT the original sync-register is replaced with an async-CDC modelling block. The async-CDC modelling block can emulate the situation of signal crossing the two clock domains. In

order to speed up the proof, the accelerator modules are applied in the DUT to replace the buffer and counter units.

In this solution contents many Verilog components beside DUT. There are three types of the modules for modelling, verifying, and accelerating. The modelling modules emulate the behaviors of function of design and environment, similar as assumption. The verification modules are in charge of checking of the correctness of data transmission. The accelerating modules will replace the original part of DUT, abstract the function of the original one in order to decrease proof time. In each type there are several modules available to adapt different design/verification requirements.

Modelling modules: async-CDC, async-CDC-wire, async-CDC-reg, clock-frequency-jitter ,etc.

Verifying modules: scoreboard, scoreboard-priority, etc.

Accelerator modules: fifo, fifo-priority, memory, multiplier, etc.

II. DATA PATH VERIFICATION WITH FORMAL SCOREBOARD

A. *Verifying Modules : Formal Scoreboard*

The formal scoreboard is embedded with all the required assertions for verification of end-to-end data integrity across a data path. In this type of verification problem, the properties themselves require a large memory in order to verify the design intent. For instance, to verify that packets are never corrupted across a data path, the requirement needs a FIFO to keep track of the incoming data to compare that data with the observed emitted data at the other end. The solution is designed to overcome these types of problems and make formal verification possible in situations that would otherwise be hard to approach with formal techniques.

The following error conditions are identified with this Verification Component:

1) Dropped Data Packets
When two tagged data packets enter the DUV but only one of the data packets exits, then data was dropped.

2) Duplicated Data Packets
When only one tagged data packet enters the DUV but two such data packets exit, then data was duplicated.

3) Out-of-Order Data Packets
When tagged data packets enter the DUV in a given order but exit in a different order, then data sequencing was not respected.

4) Corrupted Data Packets
When a tagged data packet enters the DUV but this data packet exits with a different value, then data integrity was violated.

The correctness of the transactions across the DUT means the packet faithful transmission, maintaining the order, data duplication and corruption. Validating the design would require plugging in the scoreboard to the DUV and connecting it to the corresponding interface signals. All the required checks are embedded in the binary and wouldn't need any specific assertions to be coded by the validator. There are sanity checks enabled as cover points to assist the validation closure.

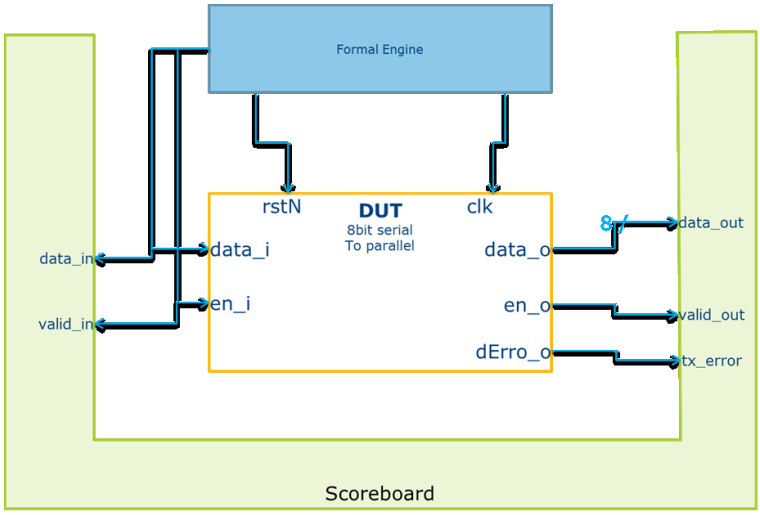

Figure 1. Bind example of Scoreboard

The Figure 2 shows the example of binding the simple formal score board with a 8-bit serial to parallel receiver. The instantiation for VHDL DUT could be like:

```
dp : Formal_scoreboard
generic map ( PKT_LATENCY => 65, LOG_PKT_LATENCY => 7
)
port map (
clk1 => clk, clk2 => clk,
rst1N => ( rstN and not tx_error ),
rst2N => ( rstN and not tx_error ),
incoming_sop => ( 0 => valid_in, others => '0' ),
incoming_eop => ( 0 => valid_in, others => '0' ),
incoming_vld => ( 0 => valid_in, others => '0' ),
incoming_data0 => ( data_in ),
outgoing_sop => ( others => valid_out ),
outgoing_eop => ( others => valid_out ),
outgoing_vld => ( others => valid_out ),
outgoing_data0 => ( data_out(0) ),
outgoing_data1 => ( data_out(1) ),
outgoing_data2 => ( data_out(2) ),
outgoing_data3 => ( data_out(3) ),
outgoing_data4 => ( data_out(4) ),
outgoing_data5 => ( data_out(5) ),
```

After loading of the score board and DUT in tool, the following assertions/covers will be generated:

Table I. Formal score board assertions

Assertions	Description
Packet_sanity	Verify the data path cannot dead lock
Packet_no_extra	Data packets is transferred without duplication
Packet_integrity	Data packets is transferred without corruption (dropped, duplicated, out of order)

Table II. Formal score board sanity checks

Sanity Check	Description
Packet_sanity	At least one packets are pushed and popped
Reset active	Reset signal is active
Reset toggle	Reset signal toggles at least once

B. Modelling modules

The architecture for verification includes modelling counters and buffers along with the asynchronous clock domain crossing modelling. The clock frequency jittering mechanism does allow frequencies to be changed during verification which helps in covering a wide range of clock ratios. The async-CDC module can exhibit non-deterministic behavior when the active edge of the input domain clock lines up with the active edge of the output domain clock.

1) Async-CDC:

This module is instantiated once for each flop (or bus of flops) that are in the fanin of flops immediately after the CDC. The figure 3 illustrate the non-deterministic behavior of the sync-flop. The ansyc-CDC module will replace the original flop to model this behavior

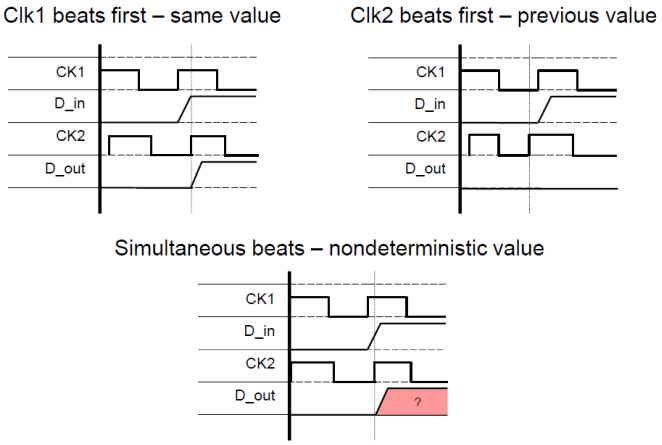

Figure 3. the non-deterministic behavior

The instantiation example of Async-CDC module

```
R1_model: formal_model_async_cdc
generic map ( SIG_WIDTH => 8 )
port map (
clk => clk1,
rstN => not rst,
sig_in => S1,
sig_en => (others => '1'),
sig_out => new_R1 );
```

2) Frequency jitter

The frequency jitter is used to model the effects of frequency jitter when proving assertions in a design under verification (DUV) that includes multiple clock domains. Using this module makes it possible for the ratio between fast and slow clocks to vary within a defined range instead of always being locked at the same value.

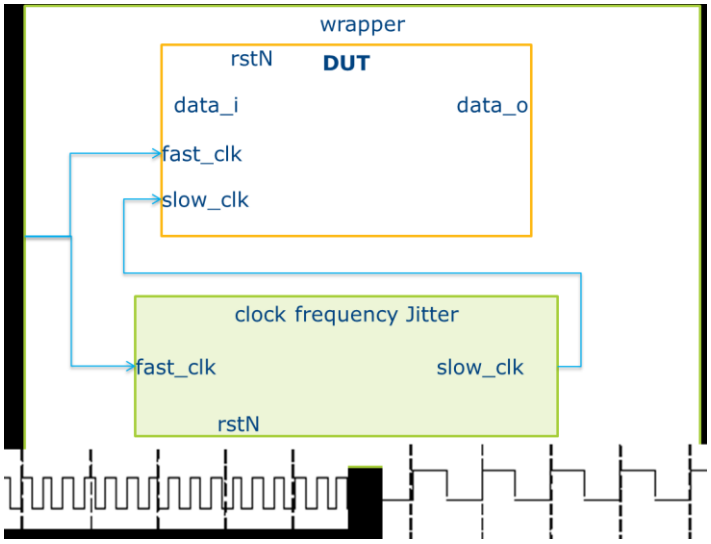

Figure 4. the example of bind clock frequency jitter with DUT

Figure 4 shows the connection of frequency jitter with DUT. By modification of the parameters of frequency jitter, the clock frequency and ratio of the clocks will be changed. The frequency relationship has a three-value range. An additional parameter specifies the number of cycles the system takes to recover after frequencies deviate from their nominal relationship.

Table III. Formal score board sanity checks

Parameter	Description
RATIO_LOW	Lowest ratio value between fast clock and slow clock
RATIO_LOW	Median ratio value between fast clock and slow clock
RATIO_LOW	Highest ratio value between fast clock and slow clock
RECOVERY_CYCLE	Number of cycles the DUV takes to recover from a frequency glitch

As Table III shows, we don't need to redefine the clock and elaborate; we can get the clock signals with three different frequencies on the given ratio of the fastest clock. It help us to emulate the async-clocks environment.

C. Accelerator modules FIFO

By application of formal property verification there are always buffers, memory or mathematic multiplication/divide units in our design. These structures are rarely of interest in the overall verification of the design since they are well understood and seldom contain bugs. However, these structures can become major performance bottlenecks for formal verification tools due to the huge state space they can introduce. In this paper we will only show the example how we use the abstract FIFO to replace the original one accelerate the proof speed.

Our EDA vendor enable safe abstraction of FIFOs and memories while preserving sufficient information across these structures to ensure that the Formal Scoreboard provides accurate results, both in terms of valid proofs and meaningful counterexamples. These modules make it possible to fully verify end-to-end data integrity properties that would otherwise be impossible to handle with formal technology.

The usage of abstract FIFO is very simple. User need to replace the driver of the output signal of the DUV's FIFO with abstract FIFO output, and connect the signals with the DUT. Below is the example of instantiation of abstract FIFO of VHDL design

```
fifo_model: formal_model_fifo
generic map ( PTR_WIDTH => 5, DATA_WIDTH => 1, DATA_REG_WIDTH => 4 )
port map (
clk1 => clk,
clk2 => clk,
rst1N => not rst,
rst2N => not rst,
datain => din(32 downto 32),
datain_reg => din(64 downto 61),
push => wr,
pop => rd,
empty => empty,
full => full,
pkt_vld => '1',
polarity => polarity,
dataout => dout,
dataout_reg => dout_reg
);
```

III. PROJECT FEEDBACK

The scoreboard is deployed in the IMC project team. Compare traditional simulation method, it saves about 80% time by validation of an async-FIFO block (exact data will be given in final submission) compare the dynamic method. Compare the FPV method about 50% time is saved. With tool we get the code coverage of 94%.

IV. SUMMARY

The formal verification is applied more and more in recent year. We can find the static formal verification in every corner of hardware verification. With the development of the tools and the methods, the hardware validators have more chance to use formal method solve the problem, find the bug in their design. This scoreboard method fills another application area with many advantages:

Very adaptable: the element of the whole method could be applied separately. The score board method is not only to verify the FIFO data path. It can also be applied in any type of data path verification. The abstract modeling method can be applied separately; it can be deployed in FPV or other type of formal verification applications, to increase the proof speed. The clock frequency jitter can be used to emulate the multi clock environment.

Easy to implement: this method is very easy to use. Not much formal knowhow is required. The beginner of the formal verification can also handle with it. It only needs the basic System Verilog knowhow.

Very efficient: compare the traditional simulation method, the score board shows its powerful efficiency in project task. So far the flow is setup, the running time/proof time is relative short. For module level it normally takes only minutes to verify the data path.

V. ACKNOWLEDGEMENT

The authors express their appreciation to DVcon Techical Committee for review and comment the abstract of this paper. We also thank Su Linglong from the Intel base band team to provide the useful information from the productive project.