Improving Productivity
Using Formal Analysis by Designers

Eric Faehn
eric.faehn@st.com
Design Engineer & Customer Support

Remy Chevallier
remy.chevallier@st.com
Methodology & Verification Engineer

Abstract

Formal analysis is a very powerful verification technique, but it has usually been regarded as a method to be used only by expert verification engineers. Recently, standard assertion formats, improved technology, and pragmatic methodologies have all contributed to make formal analysis very useful for designers as well. At multiple teams within STMicroelectronics, formal analysis with Incisive Formal Verifier (IFV) is becoming a mainstream approach for design and verification engineers alike.

In this paper, we discuss several important aspects of deploying formal analysis throughout recent projects. We outline the verification flow prior to the introduction of formal, and describe the motivation to investigate this technology. We considered our initial usage quite successful: IFV found a previously unknown bug in a complex DRAM refresh controller and verified two additional blocks in only three days, versus three weeks for a similar block on a previous project.

The largest section of this paper covers our most recent experience using IFV to verify a customer design. We describe how and why we selected particular blocks in this complex controller for formal analysis and why we decided against others. We discuss some of the assertions written by the designers, outline the challenges faced in properly constraining the design for formal, and give some examples of the types of design bugs we found.

We also cover the role of assertions in system verification as well as some specific issues regarding gated clocks that we had to address. Finally, we summarize the overall verification results for this project and the lessons we learned for application on future projects. Throughout this paper, we focus on the designer's role. Strong designer involvement has been a key for growth in the usage of formal analysis and we are pleased to be able to report our success in this area.

1. Introduction

The development of embedded DRAM (eDRAM) systems is performed within ST by a dedicated team divided into two parts. A first part is dedicated to the development of the hard macros and the second part provides the controllers which are driven by the customer application and by the customer specification. In fact, a dedicated controller is developed for each project.

Moreover, the complexity of the controller is increasing: new complex design techniques like multi-clock are mandatory to achieve customer requests and hard macro constraints.

Even if many sub-blocks of the design can be reused in projects, the verification step increases dramatically. Today, the verification time is a huge part for a project, and the designers spend more and more time in the verification effort to be confident about their design.

The eDRAM controller team has therefore to minimize the verification time in order to increase its productivity and to enhance the verification process.

A verification method which improves the coverage of the design and which can be used easily by designers needed to be evaluated [1]. We decided to do a trial project using the property checking tool Incisive Formal Verifier (IFV) provided by Cadence [2] because this tool uses the same interface as a simulator used in the team, and because this tool is designer oriented.

This paper is organized as follows. Section 2 describes the verification flow used in this team today. Section 3 presents the evaluation and the integration of the new methodology in the flow of the team. Section 4 illustrates the tool usage on a real customer project. At last, we conclude in section 5.

2. Today's flow

The current verification methodology is mainly based on random simulation on RTL level using an integrated checker engine. This testbench is developed in parallel with a system behavioral model. This method allows ensuring the model compliance toward the specification. Moreover the exactness of the test vectors and the accuracy of the integrated checker engine can be proved.

When the RTL is completely implemented this random testbench allows a first level of simulation that enables a general check of the controller behavior. After correcting the most obvious errors, a code coverage tool is used in parallel with the testbench in order to identify the uncovered code. Thanks to these metrics (FSM state coverage, FSM arc coverage, Toggle coverage, Expression coverage and Block coverage), more directed testbenches are implemented based on the random one in order to achieve the 100% code coverage target without any errors.

The verification process consists in performing only a top level verification based on a top level specification and this implies some essential drawbacks.

Depending on the customer protocol the test vectors can be very complex and a significant amount of time is required to develop the initial random testbench. Moreover the design is considered final when the verification begins. As a result bugs are found very late in the design process. They can also be very difficult to identify and to correct.

In addition the verification is always performed by the controller designer. This decision allows completing the verification process more quickly because the designer has the more precise knowledge of the customer protocol. However this method increases the occurrence probability of a missed bug even if a validation plan is developed by the designer and reviewed by another engineer before the verification process.

Due to the increasing design complexity, the amount of time require to perform the verification is becoming dramatic. The verification currently represents approximately 70% of the project and this number will only grow. As a result a new methodology had to be developed to enhance the productivity. Moreover this method should also allow the designer to improve the quality of the controller.

The results and conclusions obtained during the IFV trial project are presented in the next section.

3. Integration of formal analysis in the flow

The integration of a new tool inside the verification process is never obvious: it has to improve the quality of the design and the verification runtime. Moreover, the tool has to follow the constraints of the team organization: in our case, the design implementation and its verification are performed by the same team, so the tool must be usable by everyone and not only dedicated to the verification experts.

We decided to start by the assessment of the IFV tool on designs developed in the team, and, if this was successful, to define in a second step how the tool will be included inside the verification flow

3.1 Preliminary study: is this tool usable in our case?

We decided to split the evaluation into two phases. The first one focused on the tool capacity. The chosen test case was an already verified block which was representative of the complexity of the blocks designed in the team.

The aim of the second phase was to compare the performances of our verification methodology used today, and the verification methodology including IFV. In this phase, a designed block in a previous technology was rebuilt from scratch. The verification and the performances could be compared easily.

Phase one: How the block complexity is handled by IFV

The block used for this phase was a refresh controller with a complex eDRAM protocol interface.

The verification of the block with IFV was performed within four days. At the end, the most complex part was to model the inputs with constraints that follow the customer protocol. The specification of the properties themselves was performed quickly: 19 properties were written in addition to the component used to define the protocol.

A bug, which could lead to a missed refresh operation in a corner, was not detected by the classical flow but was highlighted using these properties. In terms of performance, all the properties were verified in less than a minute on a Linux 32-bit workstation with 2 GB of memory.

Phase two: evaluate the productivity gain with IFV

In this phase the design and the properties were developed in parallel. The block is an initialization module and is separated in 2 sub-blocks designed and verified independently. The duration of the initialization sequence is defined by parameters.

For each sub-block, the same methodology was used. A first check was performed on the design by IFV in order to check the reachability of the states and the quality of the HDL code. These checks were performed automatically by the tool. In a second run, the assertions written by the designers for this sub-block were checked.

The first block was designed and verified in one day (25 properties written: 19 assertions, 5 cover and 1 constraint, all run in 41 seconds) and the second one in 2 days (27 properties: 19 assertions, 3 cover and 5 constraints, all run in 34 seconds).

Finally the verification was done at the top level, all the properties were reused and some constraints were changed in assertions. These changes concerned the constraints written on signals that are not primary inputs of the block but were connected between the sub-blocks. The runtime at top level was also very short: 85 seconds to run all the properties. To conclude, the complete design implementation and its verification took 3 days. This performance has to be compared to 3 weeks spent for the design and the verification of the equivalent block developed with the previous methodology.

3.2 Proposal for improving verification flow

At the end of the evaluation we concluded that IFV improves greatly the productivity and the verification coverage. We decided to include this tool inside the design and the verification flow of the team. Moreover, the new verification approach being slightly different, our design methods had to be improved in order to maximize the benefits generated by IFV.

Instead of verifying the design only at the top level, the designers would use IFV during the design of each critical block. The automatic checks and designer assertions would ensure the correctness of the block.
In this case, bugs and architecture limitations would be detected much more sooner and this would allow the designer to correct them earlier in the project.

This methodology seemed very powerful and fruitful but it required a major change: the top level specification developed today becomes insufficient. New specifications, describing the targeted behavior of each block, must be written.

4. Customer product verified by formal analysis

4.1 Verification strategy

The product is a low-power cmos090 embedded DRAM IP which is provided as the association of one 4Mbit embedded DRAM macrocell, one dedicated 32K128 RTL based controller and a single RTL based programmable built-in self test (BIST) engine used for the test of the eDRAM macrocell (Fig 1). The controller has a custom tightly-coupled memory (TCM) interface derived from the ARM® TCM protocol. The IP is running at 250MHz.

The formal methodology with IFV was introduced in the controller verification in order to evaluate its use in a customer project and to enhance confidence regarding the IP validation.

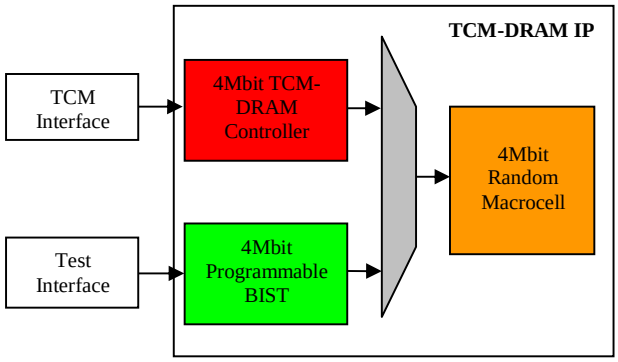

Figure 1 : TCM-DRAM IP

The first step was to identify the controller blocks where the return on investment is the highest. The selected candidates are the blocks containing the control logic because they are bug sensitive and the formal properties are usually highly effective in this case. As the macrocell behavioral model is not synthesizable the data path, which is already difficult to check in formal, was not verified by IFV but by using the classical random testbenches along with their scoreboards. Following these criteria, 4 blocks out of 5 were selected for their whole control logic to be verified in formal. In order to fully verify the control logic and to allow not only the designer but also a verification engineer to write some formal properties, a detailed functional block level specification describing the behavior of each output was developed.

The first verification step consisted of writing some white box properties, which were based on the RTL code, while implementing the system in order to realize a design implementation bring up. Moreover the automatic checks provided by IFV were also launched in order to identify the biggest implementation bugs and to ensure reachability of all states. Then black box properties, which consider block inputs and outputs only, were written in order to control the general block behavior toward the detailed specification. In this project protocol properties were not considered because no constraints were specified.

4.2 Block verification

While implementing the design all blocks are always made parameterizable in order to be reused easily in another similar eDRAM system. A package containing the system constants (counter maximum value, address bus width, etc…) is always defined. It was decided to use the same technique for the properties in order to check the design quickly with IFV when the block is used in another project with a different configuration. Some

parameters are defined based on the package constants and used while writing the properties in order to describe the general block behavior without considering the specific block usage in this project. In addition such a methodology also allows decreasing easily the design complexity if some explored results are obtained during a formal verification run.

Moreover it is also crucial to minimize the number of written properties to increase productivity. As a result the properties are implemented not only to be reused in another project but also in another block of the design. To be precise the properties verifying the output of a given block are reused as assumptions for the next connected block (Fig 2)

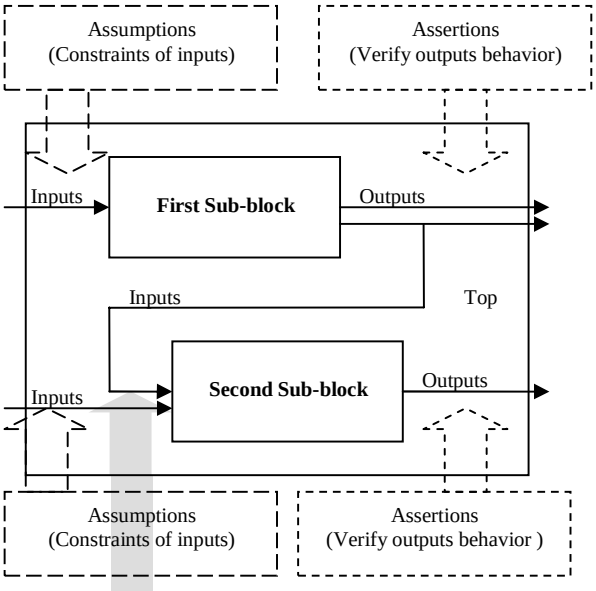

Figure 2 : Inner Design Reusable Properties

In the clock gated enable signal generation block 16 assume properties, 33 assertion properties and 8 cover statements were written along with some auxiliary code. They allowed identifying one bug that would have been difficult to catch during dynamic simulations because it did not affect the memory content but led to a misalignment toward the specification in a corner case situation. To be precise the system performance is divided by three when a particular set of commands is issued by the customer.

Then the initialization sub-blocks were verified and in this case inner design reusable properties were used. Moreover these sub-blocks are essentially sensitive to the reset. It was mandatory in this case not to consider the reset as a constant while using IFV and that led to a property modification: “abort reset” had to be added in each temporal property. While developing the properties another issue was encountered and led to a verification strategy modification. In fact the controller is a low power system and some gated clock structures are implemented to switch off the different blocks when they are unused. As a result some block behavior could not be tested directly in formal because part of the control logic is managed by the clock.

Two distinct solutions can be considered to solve this issue: writing auxiliary code or take the whole system into account during the formal proof. As writing additional code is too time-consuming the second solution was selected and is described in the following section.

4.3 System verification

Performing the formal verification at system level allowed writing the remaining properties quickly because they had to follow exactly the detailed specification. It was only a matter to translate the properties into PSL [3]. Finally 11 assume properties, 82 assertion properties and 15 cover statements were implemented. This property implementation phase was performed quickly but running them with IFV has been more difficult.

Importing the whole design into IFV is actually increasing the complexity: as the cone of influence of each property becomes bigger, the run time is highly increased because the formal proof is following an exponential complexity that is function of the logic cone. In order to get the first results quickly the design parameters have been decreased.

The first results were quite alarming because lots of properties were failing. After further analysis we noticed that some properties of the clock gated enable signal generation block were now failing. Another issue was introduced by the whole design consideration and this one was also related to the use of gated clock structures but also to the non-constrained reset value. In fact due to the several gated clocks derived from the system input clock it is mandatory to write directly the sampling event in each property, @rising_edge (CLKi). In other words the definition of a default clock is not feasible any more. As a result each assertion had for instance the following structure:

Assert_Property_Name: assert always (
(Property_Description) abort (Reset = ‘0’))
@(rising_edge CLKi);

An active reset operation is switching off the clocks and the above property could not allow IFV checking the reset value because no sampling event occurs when reset is low. The property is never aborted when reset is active and IFV is always finding a counter example. Finally the sampling event was modified as follows to take the reset into account:

@((rising_edge CLKi) or (rising_edge Reset)).

After debugging these properties, the remaining issues are 3 explored results that are still to be resolved. To achieve passing results, some IFV configuration parameters had to be set to their optimal value. For example the effort was set to high (30 minutes maximum to check each property), the engine was set to axe (adapted to assertions requiring huge evaluation cycles) and the halo was set off (efficient when pass is the expected result because the whole cone of influence is directly considered). IFV was then run again and all properties (11 assume properties, 82 assertion properties and 15 cover statements) passed within 448 seconds on a Linux 32-bit workstation with 2 GB of memory.

The final step was to set the design parameters to their real value and run IFV another time to check the design in its correct configuration. During this step the engine was set to axe, the halo was set to off and the time was set to high. This sanity check was performed on the 69K gates design and 104 out of 108 properties passed within 4 hours and 30 minutes. 4 properties did not pass due to a tool limitation. In fact the maximum counter width defined in IFV is 16 bits and these properties require 18 or 19 bits to perform the complete property formal evaluation. As a result these properties had to be removed during the final formal verification and had to be double checked during dynamic simulations

The controller design logic had now been checked using formal methods and the target wass to launch the classical dynamic simulations to verify not only the data path but also the properties effectiveness and accuracy.

4.4 Reuse of properties during dynamic simulations

The plan was to check all properties during the dynamic simulations in order to verify the formal constraints that have been applied on the design while running IFV. The option **–assert** was added in the script and the tool was launched. The preliminary results were quite disturbing because almost all the properties were failing.

A first analysis underlined that the properties that have to check the initial value of a given signal can not be used during dynamic simulations because the initial value is always ‘X’. To solve the issue the formal property **assert (signal_name = initial_value)**, must be modified as follows: **assert always (active_reset → (signal_name = initial_value))**.

Moreover a further study showed that the gated clock structures were also producing issues in dynamic simulation. As the dynamic simulator was performing a timing based analysis and not a cycle based one like in formal, the gated clock structures were creating delta cycle time issues. As a result the falling edge of the clock was used to modify the input stimuli. In addition the dynamic simulator was always considering the input clock as the evaluation event which is why the rising edge of the reset had to be removed from the properties evaluation event in order to avoid other delta cycle issues.

Finally, like during the final formal run, 4 properties required 18 or 19 bits of counter to check them but the maximum width of the counters defined in NCsim is 16 bits. As a result these properties had also to be removed during dynamic simulations.

After performing these modifications the results were in line with IFV. In other words all the assertions passed. Reusing the formal properties in dynamic was therefore not straightforward. As debugging these properties is not an easy task especially when another engineer is in charge of the dynamic simulations, the recommendation is to use a generic parameter and to write directly the properties for both situations. So the properties will be easy to use during the whole product life.

5. General conclusion

The usage of IFV underlines one more time that the property checking methodology generates a big gain in the functional verification area. In our case, mainly thanks to a tool that can be easily integrated in our environment and its ease of use by the designers, the verification effort decreases dramatically and its use improves greatly our productivity.

In addition, putting this tool in our flow has other beneficial impacts on our methodology.

First the block design and the formal verification are done simultaneously which speeds up design and verification phases.

Then in order to use property checking methodology a more accurate specification has to be written. That allows the designer to describe the behavior of the sub-blocks more precisely. This step is not only mandatory to use a property checking tool, but also a way to optimize the global architecture.

Moreover, all the work done at block level is reusable at a higher level either during formal or dynamic verification. In addition these properties could also be provided to the customer in order to act as an embedded specification for functional verification.

However the designer has to consider directly the formal and dynamic requirements while developing the properties in order not to waste time. Moreover some tool improvements, like the evaluation counter width, are mandatory to run the verification with the real parameters. Indeed without this sanity check the designer can never be sure that the verification results are correct in the real design configuration.

Finally, to maximize the gain of this new methodology, an expert should specify the verification strategy and the planning.

6. References

[1] Sami Maisniemi, Jari Kalinainen, "Assertion-Based Verification with PSL Integrated with an Existing RTL Verification Environment", PSL/SUGAR CONSORTIUM MEETING DATE 2004

[2] White paper, "Getting the Most Out of Formal Analysis": http://www.cadence.com/whitepapers/formal_analysis_wp.pdf

[3] Accellera, PSL Language Reference Manual. version 1.1, June 9th 2004: http://www.eda.org/vfv/docs/PSL-v1.1.pdf

STMicroelectronics
850 rue Jean Monnet
38926 Crolles cedex France
www.st.com