2014 DESIGN AND VERIFICATION DVCON CONFERENCE AND EXHIBITION EUROPE

A Pragmatic Approach to Metastability-Aware Simulation

Joseph Bulone, Kalray, Montbonnot Saint Martin, France (*joseph.bulone@kalray.eu*)

Roger Sabbagh, Mentor Graphics Corporation, Ottawa, Canada (*roger_sabbagh@mentor.com*)

***Abstract*— This paper describes the experience of using metastability-aware simulation to verify the clock-domain crossing (CDC) logic in portions of a multi-purpose processor array IC. We describe how we found critical CDC bugs, which were neither addressable by regular simulation nor by CDC formal tools. We explain how we extended the results of the formal approach to enable meta-stability aware simulation, with a relatively low impact on the global simulation time, and with randomization and automation within a continuous integration flow.**

Keywords*— *Clock-domain crossing, Metastability, Formal verification, RTL simulation

I. INTRODUCTION

A. Metastability

Synchronous storage elements, such as flip-flops, may enter a metastable state when signal timing requirements are violated. This can happen when the input does not remain stable for a window of time before and after the sampling clock edge, defined as the metastability window, as shown in Figure 1. In the metastable state, the register output will be hovering between valid logic levels for some period of time, defined as the settling time, until it transitions to a stable high or low level. The level that it settles at, high or low, is non-deterministic and has no correlation to the value of the input, so it's essentially random.

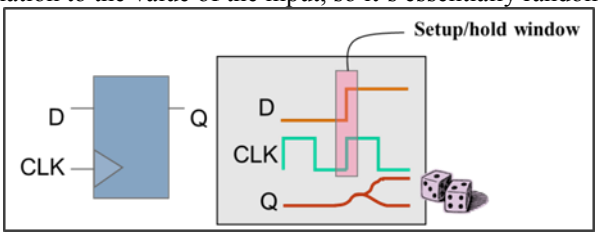

Figure 1. Metastability window

A. RTL simulation and metastability

Setup and hold violations do not have any effect on the signals in an RTL simulation. It's an ideal world, where, depending on the alignment of the clock edges, all signal transitions stemming from one clock domain will either be entirely sampled by the next active edge of the clock in the receiving clock domain, or entirely missed and picked up on the following edge. As a result, all simultaneous CDC signal transitions will cross the clock domain boundary in unison, without regard to the metastability window. In reality, this is not always the case. In silicon, simultaneous CDC signal transitions that occur within the metastability window may arrive skewed in time with respect to each other due to metastability effects.

B. Single-cycle delay effect

When a setup-time violation occurs in silicon, the register may become metastable and settle to the *opposite* logic value as that of the input signal, only to take the correct value on the next receiving clock edge, as shown in Figure 2. The effect is a single-cycle delay in the propagation of the signal transition when compared to RTL simulation.

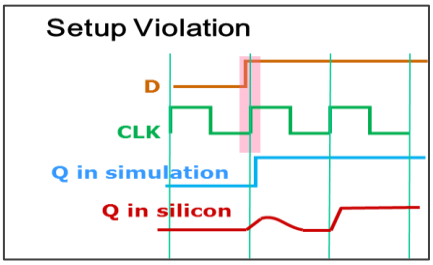

Figure 2. Single-cycle delay effect

C. *Bleed-through effect*

When a hold-time violation occurs in silicon, the register may again become metastable, but this time, settle to the *same* logic value as that of the input signal, as shown in Figure 3. When that happens, the signal will propagate one cycle earlier than observed in RTL simulation, thus it has a bleed-through effect.

It's important to note that *both* the setup and hold effects must be modeled in order to detect all metastability related bugs that could otherwise go undetected in RTL simulation.

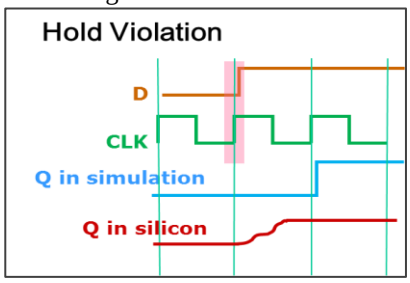

Figure 3. Bleed-through Effect

D. *CDC reconvergence*

CDC reconvergence occurs when signals are derived from the outputs of at least two different synchronizers. As established above, metastability effects may cause transitions at multiple synchronizer outputs to be shifted in time with respect to each other. A circuit that is designed to accommodate these effects can safely recombine the CDC signals so that the resulting output always maintains its coherency. An example of this type of "good" reconvergence is a binary counter that is first gray-coded before each bit of the counter is individually synchronized[1]. This ensures that the counter value is not corrupted due to metastability effects.

However, any circuit whose operation is not immune to these effects will appear to operate correctly in RTL simulation, but will fail in silicon. An example of this type of "bad" reconvergence is shown in Figure 4. In this circuit the DATA and COMMAND signals are in phase with each other in the transmitting clock domain, but after synchronization with separate FIFOs, they will be skewed in the receiving clock domain.

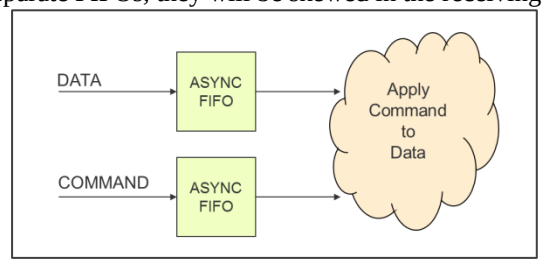

Figure 4. Bad CDC reconvergence circuit

II. RECONVERGENCE VERIFICATION

A. *Classical methods*

Verifying CDC reconvergence logic is one of the most challenging aspects of CDC verification. There are two primary classical methods for this: 1) use of modified synchronization cell simulation models and 2) CDC formal analysis.

With the first approach, synchronizer circuit models are modified for use in simulation in an attempt to model metastability effects. Typically, a 2-DFF synchronizer cell model is modified to randomly insert an extra cycle of delay. This has a number of limitations[2], including:

a) The bleed-through effect is not modeled. This method only adds the single-cycle delay effect, so will miss potential bugs related to advancing signal transitions due to the bleed-through effect.

b) Metastability effects are only modeled on CDC paths which use the instantiated synchronizer cell models. This will miss adding metastability effects on paths where designers use other synchronization methods, including RTL inferred synchronizers. Also, it is nearly impossible to follow this approach when verifying designs containing 3rd party IP blocks.

Static CDC formal analysis of a design can detect CDC reconvergence points; however, it has its limitations, including:

a) It is generally incomplete. Reconvergence is typically analyzed to a specific sequential depth starting from the clock domain boundaries. There could be additional reconvergence points beyond this depth that are not discovered by this method.
b) It cannot distinguish between good and bad reconvergence points in complex circuits. Simple reconvergence structures, such as gray-coded counters, can be proven to be good by the formal tools. However, more complex circuits, such as the reconverging FIFOs in Figure 4, cannot. In today's designs, which have many clock domains and thousands of CDC signals, manually checking the reported reconvergent CDC paths is inefficient and impractical.

B. Metastability-aware simulation

Due to the limits of traditional methods mentioned above, we decided to adopt an automated approach to reconvergence checking with metastability-aware simulation.

With this method, behavioral metastability effects models are generated for all CDC signals[3]. CDC formal analysis is very capable of exhaustively detecting and reporting all CDC signals, regardless of the synchronizer type or lack thereof. Hence, it is well suited for the task of automatically generating and instantiating these models, which are bound to each CDC signal in the design as shown in Figure 5.

The metastability effects models monitor each associated CDC signal in order to detect when metastability could possible occur; that is when:

- the clocks are aligned to within the defined metastability window, and
- the Tx signal is changing, and
- the Rx register is loading a new value due to a change in the Tx signal

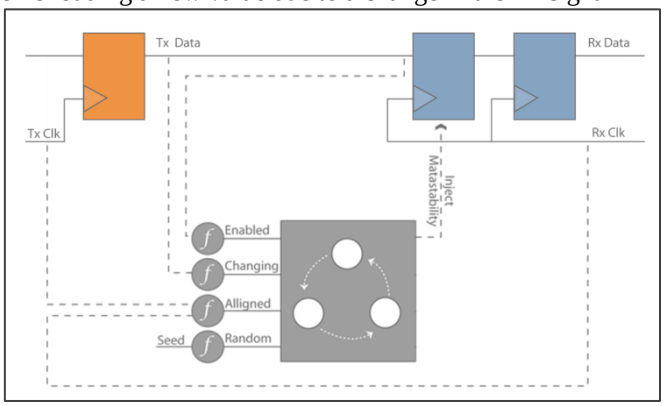

Figure 5. Metastability effects injection

If all of these preconditions are met, then the value of the Rx register is *randomly* perturbed on a per-bit basis in order to model both the single-cycle delay and bleed-through effects. The random behavior can be controlled by specification of a seed, defined as the metastability seed, in order to vary the operation of the model across multiple simulation runs.

C. Metastability coverage metrics

As with any verification methodology, metastability-aware simulation must provide metrics in order to detect coverage holes. The metastability-effects models also include functional coverage points to report events observed at each CDC signal, such as:

- clocks are in alignment
- metastability is possible (all conditions met)
- single-cycle delay effect injected
- bleed-through effect injected

If the reported coverage is too low, action can be taken to improve it and increase the level of confidence in reconvergence verification. For example, for any given clock domain boundary, if the clocks did not come in to alignment very often, then the simulation environment could be modified to change the phases of the clocks or add jitter and wander. Also, the metastability window could be widened. If the clocks were in alignment, but metastability was not possible very often on a particular crossing, then the level of activity on the CDC signal may be too low, so the test stimulus could be modified to increase the number of CDC signal transitions.

III. METASTABILITY-AWARE SIMULATION OF THE MPPA-256

A. *The CDC verified design*

A CDC verification flow has been applied for the development of the Kalray MPPA® (Multi-Purpose Processor Array) 256 processor[4]. It integrates 256 processing engine (PE) cores and 32 resource management (RM) cores on a single 28nm CMOS chip. These VLIW cores are distributed across 16 compute clusters and 4 I/O subsystems, each with a locally shared memory, as shown in figure 6.

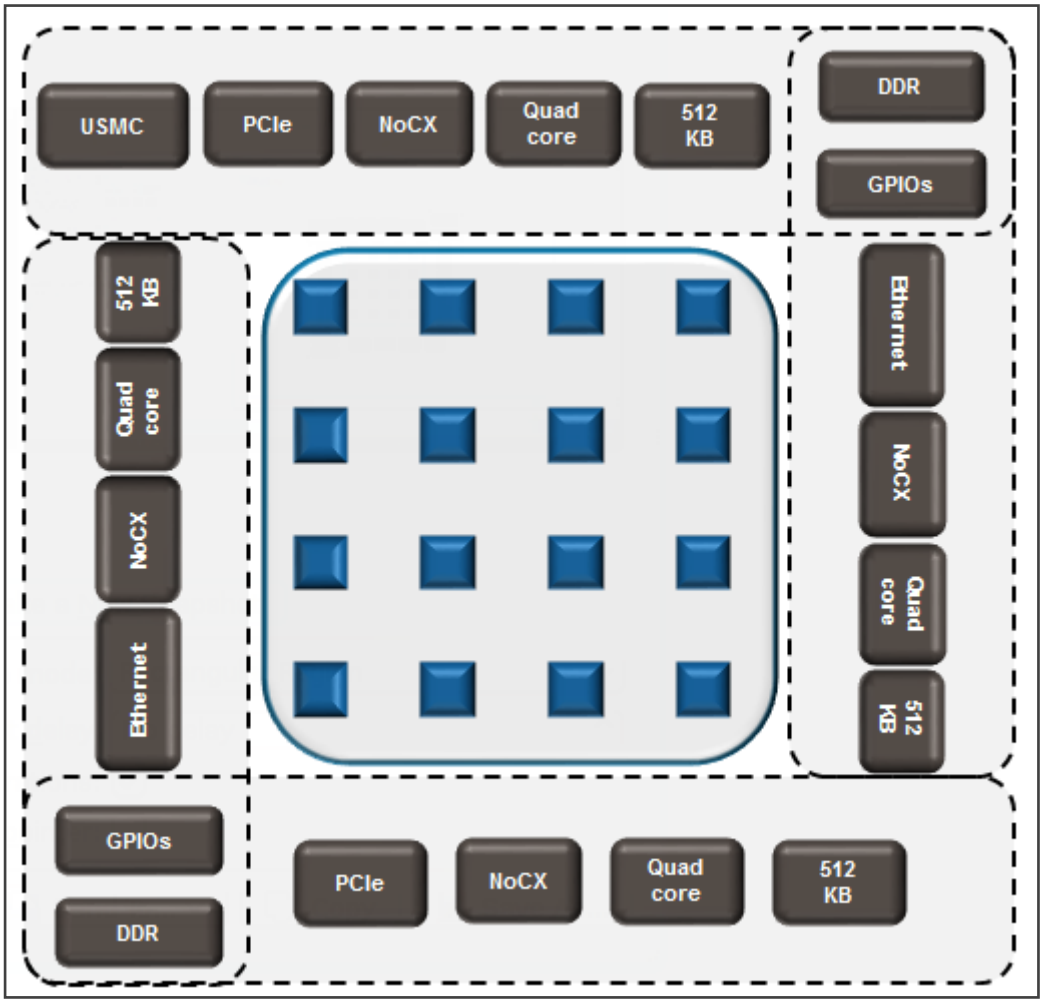

Figure 6. MPPA® block diagram

From a clock domain perspective, the design contains many asynchronous clock domains including the main CPU compute cluster clock and the various asynchronous clocks related to the external interfaces of the I/O subsystems. These include the DDR, PCIe, Ethernet (with NoC eXtension) controller clocks, and also the Univeral Static Memory Controller (including a NOR/NAND Flash controller), as well as the JTAG clock.

B. *The application of a classical static CDC verification and its limitations*

In a first step, all the asynchronous blocks have been checked in stand-alone mode using a CDC static analysis tool[5]. This step provides a lot of information about the identified CDC schemes or missing schemes. Our goal was to be able to conclude whether all the reported CDC violations were relevant or not. For some types of violations, it's enough to check some key design properties through an alternate means in order to guarantee the CDC correctness. We have been able to conclude to the CDC correctness of our internal IPs (as in the case of our asynchronous FIFO) by manual verification and also using a property checking tool[6]. Unfortunately, on 3rd party IP blocks, their complexity prevents us from achieving our analysis: several tens of thousands of violations remained. We have explored several different approaches to resolve these violations:

- A systematic waiving was too risky of a solution. Real violations could be missed.

- Refining the results by selecting specific modes.

  This has been attempted, but it is time consuming and provides partial information: it can find some CDC bugs, but it cannot ensure that there are no additional CDC bugs as it is impractical to check all modes (to the point where all violations are understood and validated) and also to check the transitions between modes.

- Checking all the required properties associated with the detected CDC schemes.

  These properties seem, at first glance, to be enough to check the full design CDC correctness, but, in fact, that's not the case.

  - Firstly, there can be complex hand-shake schemes which ensure a global correctness while including a local CDC classical scheme which does not satisfy some of its prerequisites. For example, a path which is ignored and thus invalidated in a specific mode. In such a case, data variations due to metastability effects at the synchronizer inputs will have no ill effects.

  - Secondly, as previously mentioned, reconvergence issues will only be flagged according to a depth parameter. As the depth limit is increased, the processing time increases, as does the number of reported reconvergent paths. For reconvergent paths, the requirements are that the functional logic takes into account all the potential delays and/or data modifications for implementing the desired behavior. These requirements are complex to define and to verify. Due to lack of 3rd party IP implementation knowledge, we could not practically use this approach.

  - Analysis can show that CDC properties for correctness can be met by constraining the timing on some specific signals, which in turn can translate into constraining the software timing. If that is an acceptable solution, it means that it now becomes a new type of verification requirement at the system level, to be managed using static and/or dynamic methods.

Hence, we decided to improve our CDC flow, adding two steps which achieved the goals of enabling complete and accurate modeling in simulation and systematic use within our automated, non-regression verification flow.

C. Complete and accurate metastability-aware simulation

In a second step, metastability-aware simulation was used to circumvent some limitations of the classical formal/static approach mentioned above. One could try to emulate some metastability behavior via ad-hoc instrumentation of the design which would vary the clock delays onto the flops. This is not a straight-forward task at all, as you have first to differentiate each bit within a bus and also to take care of the relationship between the transmitting and receiving clocks. In order to avoid such burdensome development, which would surely lead to an incomplete metastability model in the end, we used the automated CDC-FX feature of our static solution. This feature generates both a metastability simulation model, depending on the CDC points found in the design, and also checkers for the functional properties associated with several kinds of CDC schemes.

One essential global parameter which must be defined is the metastability window. We chose to define a window width as 99% (49% for setup time + 50% for hold time) of the receiving clock period to obtain realistic modeling, for two reasons:

1. A window larger than 100% of the receiving clock period becomes overly pessimistic and can lead to false CDC simulation errors. For example, it can generate single cycle pulses which would not occur in reality.

2. A window smaller than 100% of the receiving clock period, combined with fixed or regular clock relationships between the clocks (especially for mesochronous clocks), may lead to missed opportunities to model the effects of metastability, and thus could be overly optimistic.

D. A systematic use within our automated, non-regression verification flow

Upon completion of steps 1 and 2, we now have the ability to detect and model almost all the potential metastability events. The third and final step is to achieve the best possible coverage of metastability effects in

simulation. This is achieved by running all the non-regression tests with FX modeling and by modifying the metastability seed. Figure 7 summarizes how non-regressions were launched automatically within a continuous integration flow. The flow is based on three tools git, hudson/jenkins[5] and ruby. They respectively serve for version control, continuous integration flow and scripting. Two main jobs have been used. The first job manages the integration requests, i.e. when a user decides that his branch has to be merged within a main branch. This initiates the run of a first suite of tests in order to allow a main branch update. This main branch is shared with the other users, i.e. they will be able to resynchronize their local branches to it. The process and the test selection are made such that good confidence is provided to the resulting merge and that the short regression does not last more than one hour. A second job manages the launching of the long, non-regression test suite when a main branch has been updated. Inside the long, non-regression test-suite, metastability runs are derived from the functional tests running on platforms involving asynchronous clocks. On these platforms, starting from user developed associated configuration files, an initial CDC compilation and analysis step generates additional files including SVA assertions. These files are integrated into the classical compilation and elaboration steps for simulation. When one test is executed on a platform, the metastability seed is derived from the git hash reference of the full database associated to the merge result. This trick ensures both simulation reproducibility for a given database revision and the ability to run simulations with different metastability seeds even when a given platform did not changed from one revision to another. The flow is completely automated and thus it allows the users to concentrate on debugging the failure cases.

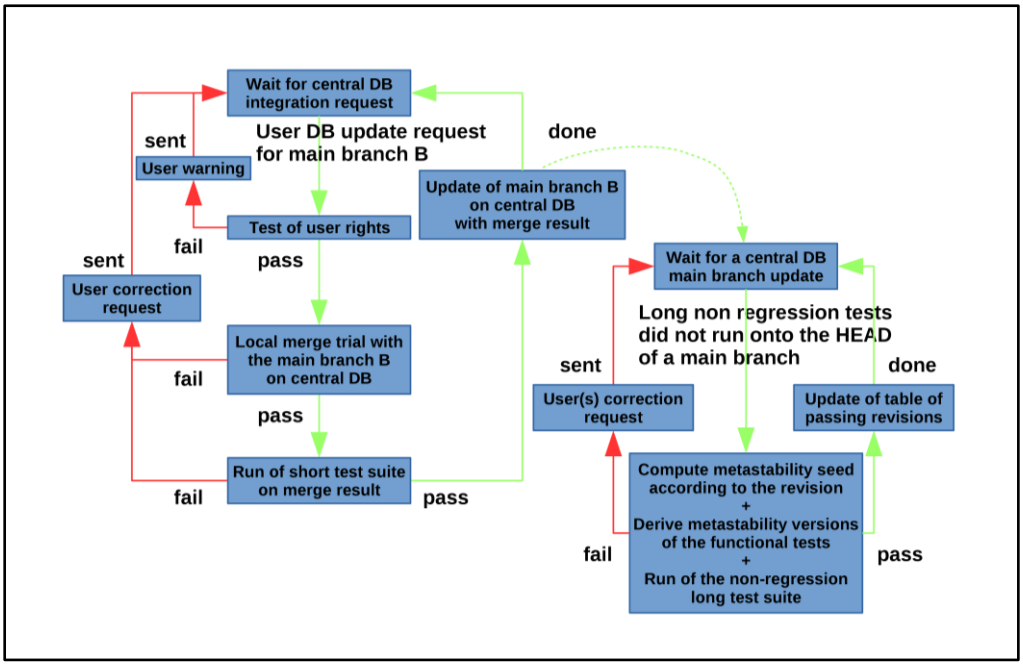

Figure 7. Automated non-regression flow integrating metastability checking

IV. RESULTS

E. *Impact on simulation time*

In term of impact on the simulation performance, we can separate the platform setup step from the test runs onto a given platform. We observed that:

- A platform setup duration is about doubled e.g. from 3 minutes to 6 minutes. This was insignificant as it was done once for the platform, after which, many tests could be run in parallel.
- A low impact onto the runtime which is in the same range or below the impact of running the simulations in parallel on one machine e.g. during the same hudson job, on the same machine, the metastability-aware simulation took 36 minutes whereas its pure RTL version took 29 minutes.

F. *Impact on design quality and verification efficiency*

At least two important cases showed the added value of the metastability-aware simulations:

- We found a bug in the DDR controller. We have been able to provide the waveforms to the IP provider for identification and also we have checked the returned patch. This has explained the hard to debug issue observed in our FPGA prototype platform and also helped to provide the temporary software work-around. As shown in Figure 4, two asynchronous FIFOs were at the origin of reconvergent paths after several cycles in the receiving clock domain.

- We are able to verify the correctness of the flash controller with simulations involving software dependent CDC constraints. Figure 8 shows the kind of schematic involved in the CDC verification. In this example, a scheme based tool will deduce that when the signal select_new_data equals '1', the transmitted data must be proven to be stable. However, this cannot be proven when considering the hardware alone, since it also depends upon the software behavior. One can write S/W code that will not respect these constraints, in which case, it will result in hardware failures. One way to cope with this verification is to determine sufficient restrictions which can be placed on the software, and which can be proven as sufficient and met by the code. Another way is the dynamic approach of metastability-aware simulation while running the S/W code. It is to be noticed that firstly this configuration is a special case of reconvergence which occurs in the multiplexor between the control bit and the register data bits, and secondly, that the data and bit register stimuli could have come from whatever sources i.e. the constraints are in reality stimuli constraints applied to a scheme, whatever the source of the stimuli may be.

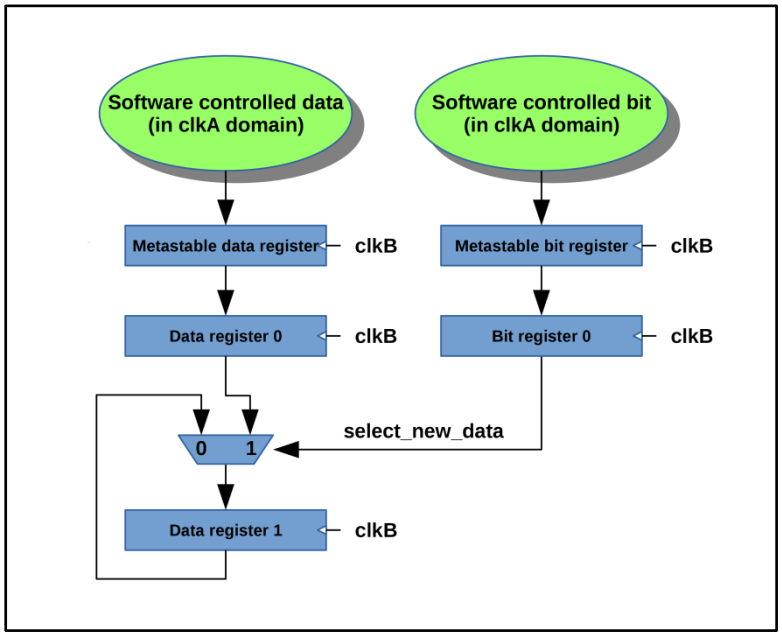

Figure 8. Example of a software dependent CDC scheme

V. CONCLUSION

In this paper, we have demonstrated a pragmatic approach to cover CDC issues related to reconvergence phenomena or stimuli dependent CDC constraints (e.g. when the constraints are propagated to software). The approach is based both on metastability model insertion after static analysis and on a continuous integration flow. It is especially effective to cope with 3rd party IP blocks, without having to know their internal details.

From a user point of view, future works could concentrate on several points, including:

- Making better link between stimuli derived constraints obtained from scheme analysis and software or external stimuli constraints and also setting-up a flow to propagate them to the environment / the software.

- Improving the metastability model in order to ensure that no unrealistic fault injection is performed and avoid debugging an unrealistic behavior.
- Having a global formal verification approach in which the formal properties are described and proven, not only for a design modelized as cycle based, but also for a design modelized taking into account the metastability effects.

ACKNOWLEDGMENT

The authors are grateful to Stephane Labert from Mentor Graphics for his efficient and responsive tool support.

REFERENCES

[1] Clifford Cummings, "Clock Domain Crossing Design & Verification Techniques Using System Verilog", SNUG 2008

[2] Roger Sabbagh and Hiroaki Iwashita, "Does Your Simulation Model Metastability Effects Completely and Accurately?", DVCon 2008

[3] Tai Ly, Chris Ka-Kei Kwok, L. Curtis Widdoes Jr., "A Methodology for Verifying Sequential Reconvergence of Clock-Domain Crossing Signals", DVCon 2005

[4] Benoît Dupont de Dinechin, et al. "A clustered Manycore Processor Architecture for Embedded and Accelerated Applications", HPEC 2013

[5] Questa CDC: http://www.mentor.com/products/fv/questa-cdc

[6] Questa Formal: http://www.mentor.com/products/fv/questa-formal/

[7] Jenkins: http://jenkins-ci.org