2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Novel Approach to ASIC Prototyping

Mohamed Saheel (mohamed.saheel.nandikotkur.hussainsaheb@intel.com)
Balasubramanian, Suresh (suresh.balasubramanian@intel.com)
Rousseau, Vijayakrishnan (vijayakrishnan.rousseau@intel.com)
Intel Corporation
1900 Prairie City Road, Folsom, CA - 95630

***Abstract* – Advancement in EDA tools and methodologies has enabled a much easier way to integrate and validate multi IP interaction today. Emulation provides respectable acceleration at IP level. But at subsystem/SoC level validation, due to larger design and increased complexity, design speed becomes a bottleneck for validation. ASIC prototyping can be used as a platform to achieve early prototypes that run practically at real-time speeds which allows for early validation of hardware IP and software-hardware interaction. Another hurdle is the usual turnaround time (TAT) for model bring up. With a good understanding of FPGA platform, tradeoffs can be made to reuse test bench collateral from emulation that match design and validation requirements. It is also imperative to make sure that FPGA platforms are available for all users at an organizational level. This paper discuss how this novel approach to prototyping enhanced the validation of subsystem/SoC designs followed by tradeoffs made to have a short TAT for model builds and also how this platform was made available for all users across multiple orgs and geographies.**

I. INTRODUCTION

With ever growing SoC design size and complexity, software simulation is becoming a bottleneck due to both simulation speeds and modeling accuracy. In addition to this, we see a lot of push in the industry on left shift to enable software and hardware co-validation. Due to these reasons FPGA prototyping seems to be an ideal platform which empowers designers and validators to exploit benefits like faster model speed while guaranteeing accurate hardware behavior. Early bring up of software is another area which requires higher design speeds. Prototyping platforms are a good choice to validate software early and avoid encountering bugs in post-silicon. Some of the common challenges seen with FPGA prototyping are as follows:

1. Model bring-up and iterative builds
2. Debug visibility & Emulation model reuse
3. Performance tuning
4. Organization wide usage

II. CHALLENGES

A. Model Bring-up and Iterative Builds:

With legacy prototyping solutions, it is usual to see long model bring up times that can span anywhere between a month to several months. As the targeted designs outlined for prototyping are usually big in size spanning multiple FPGAs, it was typical to see a time spanning up to 3 months. Usual issues seen were with regard to partitioning the design, model dead on arrival, minimal debug visibility, etc.

Partitioning: This novel approach made use of vendor tools that handled partitioning strategies automatically. This is usually a constraint file provided by user to run multiple strategies instead of depending on one fine-tuned strategy that is specific to the design. This not only provided stability and fast TAT for the current design bring up, but also reduced time taken for future builds. An optimal approach to pin multiplexing is of utmost importance when prototyping multi board designs because this can have a direct effect on model performance. With this novel approach, the partitioning tool was intelligent enough to find a good tradeoff between partitioning the design and multiplexing of pins again based on the constraints provided by the user.

Model Dead on Arrival: It was often seen after months of effort spent on model bring-up that the model was nonfunctional. This issue was exacerbated by limited debug tools available with multi FPGA designs. As this approach reused a functional RTL emulation model, any model bring up issues were easily debugged by reproducing the scenario in emulation. Furthermore, any issues seen further down the line could always be reproduced 100% on mainstream emulation models.

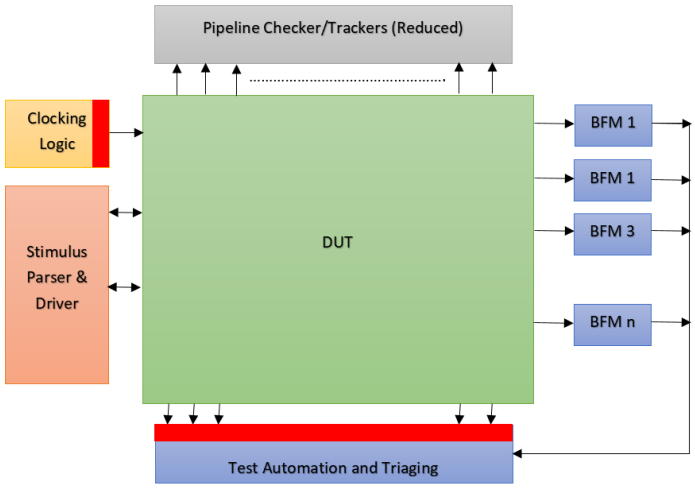
B. Debug visibility & Emulation model reuse:

One of the primary requirement with this approach was to ease the arduous task of debugging FPGA failures. With legacy prototyping approach as the design was custom built for FPGA, it was impossible to reproduce failures

on any other platforms like emulation. But in this approach, an already functioning mainstream emulation model was reused and issues seen on FPGA were 100% reproducible in emulation. The main advantage of emulation is complete design visibility for a long simulation time. This approach greatly helped in debugging issues with an average TAT of 1 day which otherwise would be a multi-week effort.

The emulation model used for prototyping also included hand selected transactors deemed important by design/validation engineer. This novel prototyping approach supported minimal number of transactors and with minor modifications they were readily usable. These transactors would provide vital first level debug information to narrow down and potentially debug issues without waveform captures.

Fig. 1: Depicts all components in the FPGA environment:



Components highlighted in red are the ones that required minor modification when porting to FPGA platform. Another change would be to support limited number of BFMs(Bus Functional Model) in FPGA. Most of the changes were done to the testbench environment due to fundamental differences between emulation and FPGA platform. These can be broadly broken down as follows:

1. *Clocking Logic:* Generation of clocks can be done in multiple ways based off the capabilities offered by an emulation platform. But with the FPGA platform used here, the clocks needed to be generated using platform specific pragmas. As few of the designs need accurate clock modeling, the implementation of clock delay calculation logic was kept constant across emulation and FPGA platforms and the delays generated were used for accurate clock generation.
2. *BFM and Checkers:* Emulators can inherently have a large number of BFMs or checkers spying on DUT interface and driving stimulus or recoding events. The usual approach in emulation to either add or remove them is done based off the performance hit incured vs. validation capability affected by it. But with FPGA, there is an additional hardware related constraint. Usually FPGAs can support minimal BFMs or Checkers. The approach used here was to first optimize any performance related bottle necks with the implementation of the BFM followed by making an informed decision on which BFMs and checkers to retain during execution on FPGA. This decision was made in conjunction with the Validation and Design leads to make sure that enough information was readily availabe from the FPGA run to start debug during failures.
3. *Test Automation and Triaging:* One of the most important aspects to influence a widespread usage is to offer an easy and intuitive way to execute tests and view results. Most of the users making a move towards FPGA for validation are already familiar with using emulation. Hence, the run, automation and triage environment was kept the same between emulation and FPGA. This made it easy for users to understand how to load models on FPGA, queue up regressions and view results.

C. Performance tuning:

The key purpose of using prototyping is to achieve a fast model to stress test the design with long duration testcases. Performance is directly hampered by design partitioning & routing, clock performance, pin multiplexing,

etc. In this approach, as we run multiple partitioning and routing strategies automatically, the end-user has control to choose one of the passing partitioning and routing strategies, and control the master clock frequency of the FPGA.

D. Organization wide usage:

These new FPGA platforms were made available for users across the whole organization and did not require users to work on the board in a lab as it was traditionally done. This provided users access to FPGA boards across various business units and geographies and truly achieved the goal of providing a platform for all users to run extensive real-life test-cases.

III. SOLUTIONS AND METHODOLOGY

A. Design Setup:

Our primary goal while setting up a design for prototyping was to make sure the environment is platform agnostic and can be portable between FPGA and Emulation. This mindset from the get-go made sure that we have an environment that would reap the benefits of execution speed on FPGA and good debug visibility on emulation. We tackled this challenge from the following viewpoints:

1. *Testbench Setup:*
    a. An important stage of the model bring up for prototyping is designing the testbench to generate and drive stimuli that will validate all aspects of the design. In legacy FPGA setup, it was required that we use a custom stimuli driver logic. This caused several delays with model bring up along with design compliance issues. But with this approach outlined here, emphasis was laid on reuse of collateral with respect to all components which includes testbench, design, testing etc. This assured that we started off with a stable environment and any further tweaks required for a functional model would be minimal.

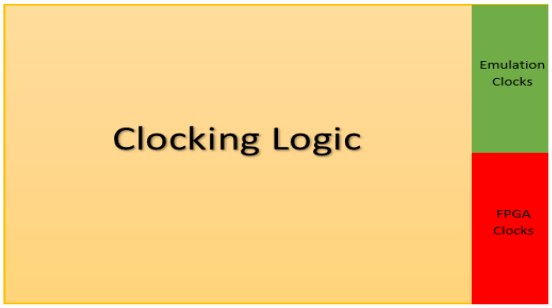
    Other optimizations were reduction of top level ports and splitting of DPI-C calls across multiple always blocks. This reduction and splitting of ports and DPI-C calls provided performance gains while increasing the number of testbench components that can be ported over to the FPGA platform.

    A vital aspect of the testbench environment is to perform correct tracking and checking during and after each test run. All critical checkers/trackers are kept alive in this approach. In addition to this, capabilities are added to make sure that results compiled after a run are not very cryptic to the end user.

    b. The compile infrastructure was broken down into intuitive stages which ensured that the user had the flexibility to run an iterative compile without having to run the whole flow for each change. The environment also had a powerful post processing stage to collate all errors and warnings of interest in one place.

2. *Clocking Strategy:* As outlined previously a few of the designs integrated at the SoC level were extremely sensitive to clock changes and also required dynamic frequency changes during test execution. To fulfil this requirement clock delay/frequency calculation logic was kept the same across emulation and FPGA models. This approach made sure that we decoupled how stimulus is driven to clock generation pragmas while keeping the implementation for delay calculation platform agnostic.

    Fig. 2: Depicts reuse of clocking logic:



    Above diagram gives a visual representation of the difference in clocking logic between emulation and FPGA model:

a. *Clocking Logic:* This block contains the important logic of delay/frequency calculation of the clocks. The expectation with this implemenation is to have a platform agnostic implementation that is compatible with the both emulation and FPGA without need for special constraints.
b. *Emulation and FPGA clocks:* These two blocks represent the small difference in clock generation logic. This is required as the pragmas used for clocks differ between emulation and FPGA.

3. *FPGA constraints setup:* As the primary objective was to prototype a SoC subsystem which would span across multiple FPGAs, FPGA partitioning strategies were set up with compile stability and automation in mind, and not just outright performance. This setup provided us a faster TAT for model bring up and guaranteed functionality. All constraints were set-up to have broad tolerance to accommodate ever-changing IPs while preserving respectable acceleration over emulation. This was primarily achieved by having multi-pass partitioning of the design with different strategies, while keeping all other factors unchanged on the FPGA platform.

B. Continuous Integration (CI) and automation:

After achieving stability with standalone automated compile, the next logical step was to integrate the FPGA compile in CI flow. The following block diagram depicts a simplified CI system. The CI flow can be broken down into the following logical stages:

1. *Sandbox Environment:* This is the first stage of the CI flow. In this stage, each check-in below represents a standalone commit that a user is attempting to integrate into the golden repository. The Sandbox Environment is responsible for running a predetermined set of checks that validate the user's changes. At the end of this stage, the checked-in code is promoted to the next stage. If failures are seen, the logs with errors and a detailed report file are stored in a trash disk for a user to investigate.

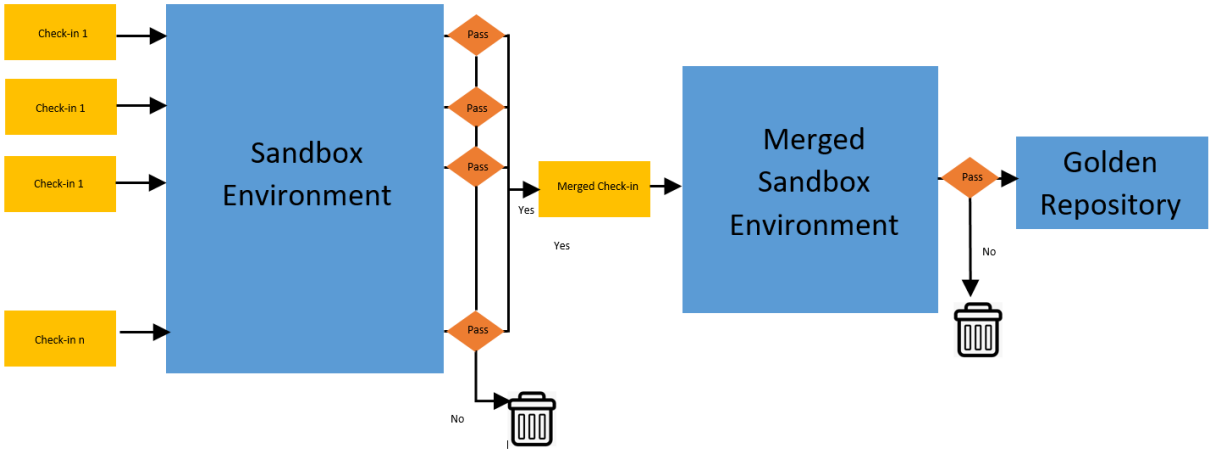
   The main objective of this stage is to validate that the changes being checked in are functional and do not break any other aspect of the project.

2. *Merged Sandbox Environment:* In this stage a set of checks are run on a merged repository. The merged repository is generated by merging changes with all the check-ins that have successfully passed the first stage (Sandbox Environment). The primary objective of this stage is to test if changes being checked-in function as expected when merged with changes from other users that are currently in the CI environment.

   If any failures are seen at this stage, the CI algorithm has the intelligence to narrow down the exact change that caused the failure and remove that faulty check-in alone from the CI pipeline while retaining and promoting all non-faulty check-ins.

3. *Golden Repository:* Once the checked-in changes have completed the Merged Sandbox stage successfully, the changes are merged into the golden repository. A release of the golden model is done from this Golden Repository to users at a predetermined cadence with a final sanity check done with respect to model stability.

Fig. 3: Basic blocks of CI system.



As FPGA compile can be done using the automated compile flow, the above CI environment was extensively used to keep the model alive since the inception of a new project. Integration of a full FPGA and emulation compile was feasible for small IPs at every stage, but larger IPs were recommended to have an emulation compile of FPGA

model alone. Runtime checks were also made mandatory in CI to catch issues at an early stage and reduce debug effort later.

4. Model TAT and Stability :

As SoCs continue to get more complex and grow in size, reuse of existing prototype setup is of utmost importance. This reuse will drastically bring down model bring up and FPGA environment development time. The automated build process and integration into CI was a major step toward qualifying small IPs to be FPGA compliant. This inherently increased the frequency at which models were built, narrowed the gap between compiles and also helped in fixing failures as they were seen in CI. This automation at the IP level helped the SoCs tremendously as IPs were pre-qualified to be FPGA compliant and also provided a snapshot to revert. As the designs were identical across emulation and FPGA platforms, emulation compile and runtime checks were added to CI. This in turn gave a higher confidence that designs are functional once integrated at a SoC level.

5. Debug ability:

Having a good methodology to debug is vital to reduce delays caused by issues seen on FPGA platform. This can happen due to multiple factors like:

1. Problems with FPGA hardware: These problems were easily identified and fixed as scheduled housekeeping and error checking tools were run to identify and report issues to the vendor. When these issues were noticed, the platform was removed from the farm and immediately worked on to bring it back online.

2. Problems with design: To debug these issues, the problematic test case was run on the emulation snapshot of design. 100% of design related issues seen in FPGA was reproducible in emulation.

3. Problems due to compile: Compile related issues were initially debugged by the internal FPGA model build team and narrowed down to different buckets. The post processing stage that was part of the automated compile flow was extensively used here to bucket errors.

6. Organization wide Usage:

With traditional FPGA prototyping solutions, the engineer running a testcase on the platform was required to be present physically in the lab where the FPGA hardware was present. This would be a feasible approach if the whole organization is located at a single site, but with today's large organizations, spread across various geographies this would not be acceptable unless the organization is willing to take on additional overhead. But the FPGA platform used in this approach had the capability of being integrating on a server farm. This allowed users to remotely connect to the FPGA server, download testcases from any remote host and run testcases. In addition to the ease of using the platform remotely, additional bookkeeping hooks were put into the farm to make sure that the resources were used judiciously and no FPGA hardware time was wasted.

IV. RESULTS AND CONCLUSION

1. Model bring up time was drastically reduced when compared to traditional prototyping flows. An iterative model was churned out in a span of 2 days. This in comparison to previous flow, is a 96% reduction in model bring-up time.
2. Test cases can be run on hardware and tracking/checking can be done in real-time. There is no limitation of post processing.
3. All collateral ranging from RTL, BFM, stimuli parser, etc. were owned in-house. This greatly reduced model debug time.
4. Due to the capability of 100% reproduction of failures/error on emulation, debug was less tedious and all information required was easily available.
5. A 10x improvement is seen on FPGA with respect to model speed for the same design when migrating from emulation.
6. There is no need to maintain a custom copy of validation collateral that is specific for prototyping.
7. Multiple Software and RTL bugs were exposed on SoC runs which would otherwise be exposed in post-silicon.

IV. REFERENCES

[1]*https://www.edn.com/design/programmable-logic/4015096/Getting-the-most-out-of-ASIC-prototyping-with-FPGAs?utm_source=eetimes&utm_medium=relatedcontent*

[2] *https://www.eetimes.com/author.asp?section_id=36&doc_id=1324000*