A Novel Waveform Generation Methodology for Power Estimation

{Shang-Wei Tu, Peng-Chuan Huang, Ya-Shih Huang}[#1], {Jack Yen, Sean Lin}[#2]

MediaTek Inc.[#1], Synopsys Inc.[#2]

Hsinchu, Taiwan

www.mediatek.com, www.synopsys.com

ABSTRACT

Obtaining an accurate maximum power number of a design is very crucial, since this number can be used for the power mesh implementation, the IR-drop signoff, the power specification review, and the PMIC (Power Management IC) specification review. However, it is very hard to create a single pattern to trigger the worst-case scenario of a SoC. Divide and Conquer is a common solution for such problem. That is, we have to create many patterns to trigger all major blocks into worst-case scenarios, but migrating all the patterns from the RTL simulation to the post-layout simulation is very complicated and time-consuming. In this paper, we propose a novel methodology for converting multiple RTL FSDB into a post-layout simulation FSDB based on the Siloti replay engine. With this methodology, we can achieve both the efficiency of generating the post-layout simulation waveform and the accuracy similar to merging multiple post-layout simulation waveform.

Table of Contents

Table of Figures

1. Introduction

Obtaining accurate power numbers of a design is as important as reporting the correct timing of a design, especially for mobile devices. From our experiences, we do have some silicon failures due to power issues when we move to advanced technology nodes. Besides, the most famous silicon failure due to overheating is the Qualcomm Snapdragon 810 [1]. Hence, it is very crucial to get accurate power numbers before the chip taped out, since many hardware decisions are made depending on these power number (e.g., the power mesh implementation, the power switch insertion, the requirement of on-die or on-board capacitance, and the maximum current supported by PMIC). In addition, a tapeout checklist usually contains parts of power related items based on the correct power numbers; e.g. the IR-drop signoff, the power specification review, the power analysis based on possible use cases, and the PMIC (Power Management IC) specification review.

Obtaining the accurate power at the early design stage has been a well-known dilemma. From **Figure 1**, when the design stage approaching tapeout, many physical design details are implemented. Hence, we can get a very accurate power number with a given post-layout simulation waveform. However, it will take extremely long runtime and many efforts to get the correct power number, but designers have nothing to do at the very last stage. On the other hand, we can easily get a very rough power number at the RTL design stage with a given RTL simulation waveform. Although the runtime is short and the efforts are medium, it is risky to make any hardware decision or do IR sign off based on this power number. Hence, getting accurate enough power number at the early design stage is still a very difficult question till now.

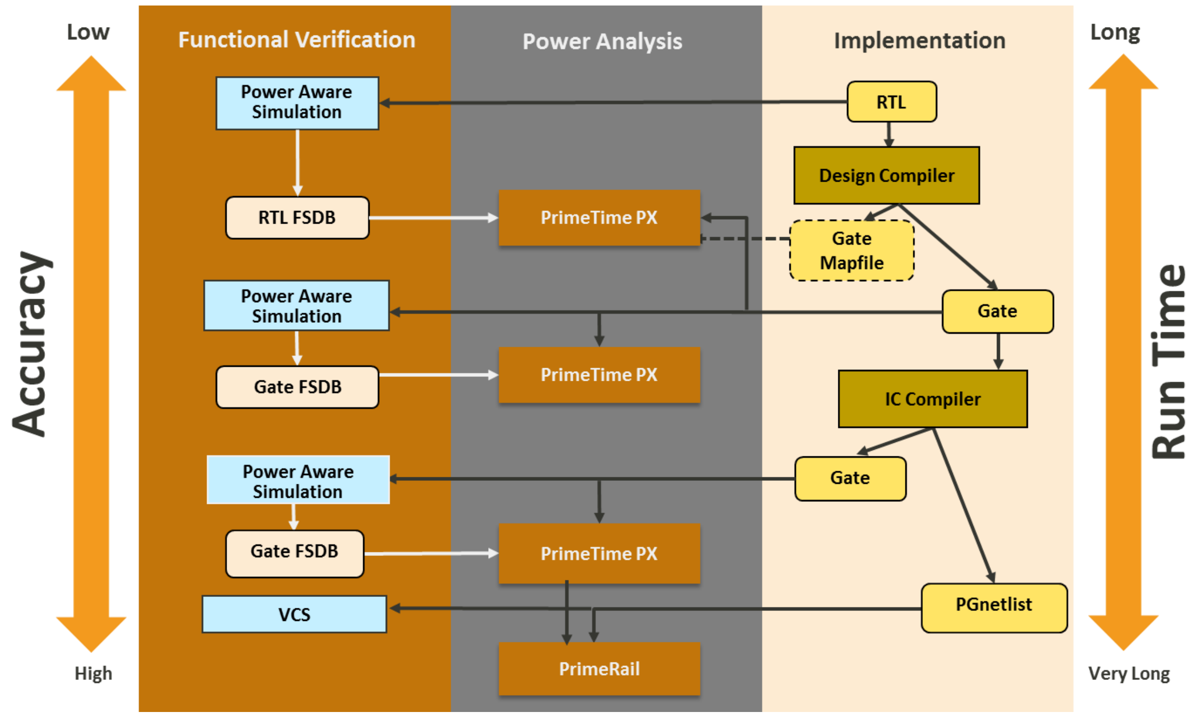


Figure 1. Design stage comparing with the accuracy and run time of the power.

Recently, some alternative solutions are proposed to mitigate the power dilemma. One direction is to reduce the runtime of the post-layout simulation, such as VCS FGP (Fine-Grained Parallelism) [2] and RocketSim [3]. Another direction is to get an accurate enough power number with medium efforts and fast runtime by replaying RTL waveform to netlist with SDF annotated [4][5]. Although these solutions do not completely solve the dilemma, they do help close the gap by reducing the run time and/or minimizing the environment setting up efforts. We have adopted the Siloti solution, but we have soon found some issues when executing the real projects. The problem is that Siloti is efficient for replaying RTL waveform to the interested design scope and generating the post-layout simulation like waveform, but to get the total power of a subsystem, we have to execute many runs with the power estimation tool, such as PrimeTime PX, to get the power number of each design scope under the subsystem and then manually sum all the power numbers. The same iterations will happen when we want to use ICC2 to report the subsystem power and do power optimization or use IR signoff tool, such as RedHawk. These iterations are tedious and time-wasting.

In this paper, we propose a novel waveform generation methodology for the power estimation, IR signoff, power optimization, and potential power related applications. We collaborate with Synopsys to enhance Siloti for concurrently replaying multiple RTL waveforms to different design scopes and generating a single merged post-layout simulation like waveform for a subsystem or even the whole chip. With this solution, we can easily get the total power of a subsystem by executing only single run with PrimeTime PX and the generated waveform. To solve the signal conflicts when merging multiple waveforms to different design scopes, we also come out a solution and collaborate with Synopsys to implement into Siloti. In addition, we also define the specification for how to convert the FSDB with conflict signals to SAIF (Switching Activity Interchange Format) and Synopsys help implement this feature into Siloti. By converting the merged FSDB to the IEEE standard format, third party tools can also utilize the merged waveform to do certain power related applications. PrimeTime PX is also enhanced for supporting the waveform with conflict signals.

The rest of this paper is organized as follows. Section 2 gives an overview to the proposed waveform generation methodology. Section 3 describes the inputs which user should prepare for this flow. Then, in section 4, we go through the details of the specification for how Siloti handling the overlap issue between different replaying scopes and the signal conflict issues on the scope interfaces. Following, Section 5 describes the detail steps of Siloti before replaying the RTL waveforms. Next, in Section 6, we demonstrate the outputs and reports of the proposed flow. Finally, Section 7 concludes this paper and gives some future works.

2. Overview of the Proposed FSDB Merging and Replaying Methodology

The overall flow of the proposed solution is illustrated in **Figure 2**. For generating a merged post-layout simulation like waveform for a sub-system or a whole chip, our flow requires many input files including the RTL file list, the gate file list, the SVF files, the FSDB configuration file, the RTL FSDB files, the "wi_config" file, and the SDF file. Then the first step **Compile KDB** of the flow will read in the RTL file list and the gate file list to do the compilation and generate the Verdi KDB (Knowledge Data Base). Following, the second step **Gen. G2R mapping** will use the Verdi KDB and the SVF files generated from DesignCompiler to generate the gate to RTL mapping file. The third step **Gen. port mapping** is to generate the port mapping file according to the given replaying scope from the FSDB configuration file. Next, the fourth step **Extract** will read in all RTL FSDB files, the FSDB configuration file, and the "wi_config" file to generate the Siloti testbench and the corresponding gate design for replaying. Still the next step **VCS Compile** is execute the VCS compilation to compile the Siloti testbench and design. The final step **Replay** is execute the VCS simulation with SDF annotated and output the merged post-layout like FSDB at the simulation end.

We will explain the details of all the inputs and the steps in the following sections with some examples. However, we will not go through the concepts of Siloti in this papers. If readers are interested in the basic concepts of Siloti, [6] is a good reference.

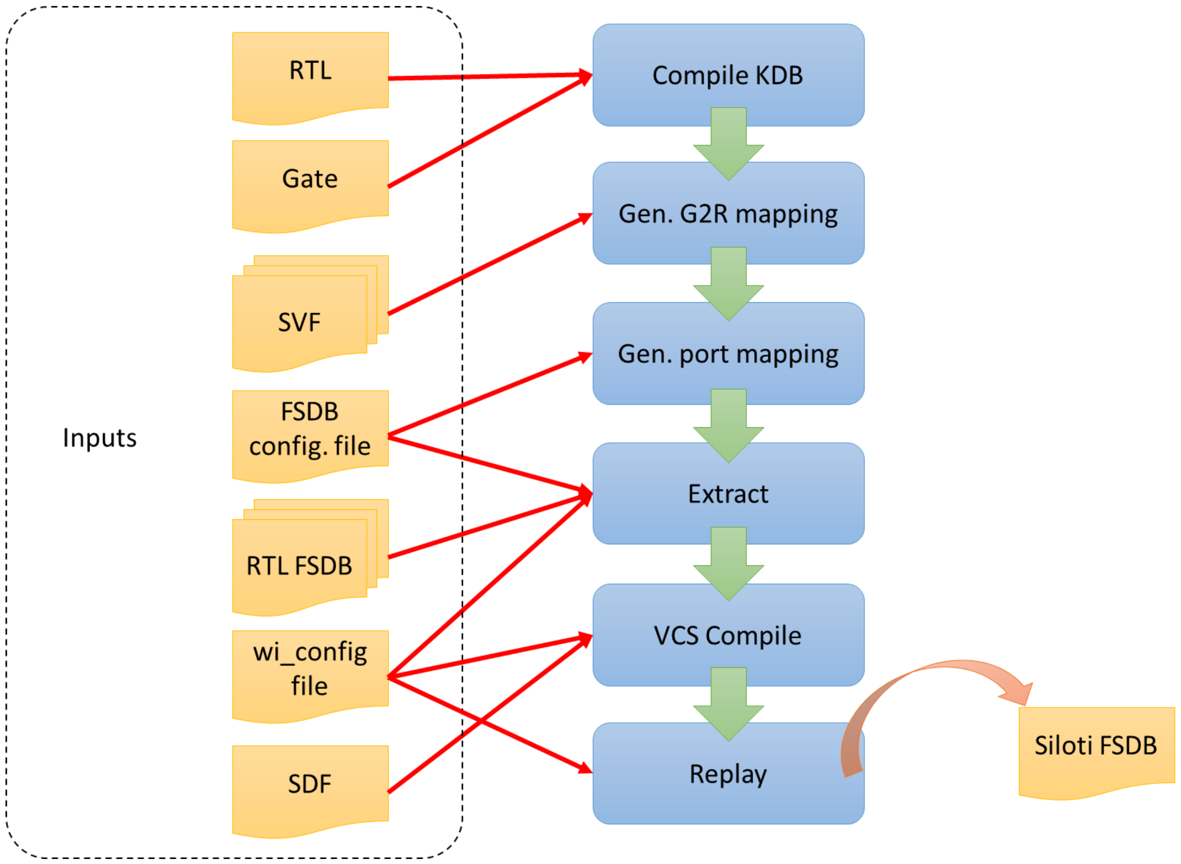


Figure 2. Proposed flow for RTL waveform merging and replaying.

3. Inputs of RTL FSDB Merging and Replaying

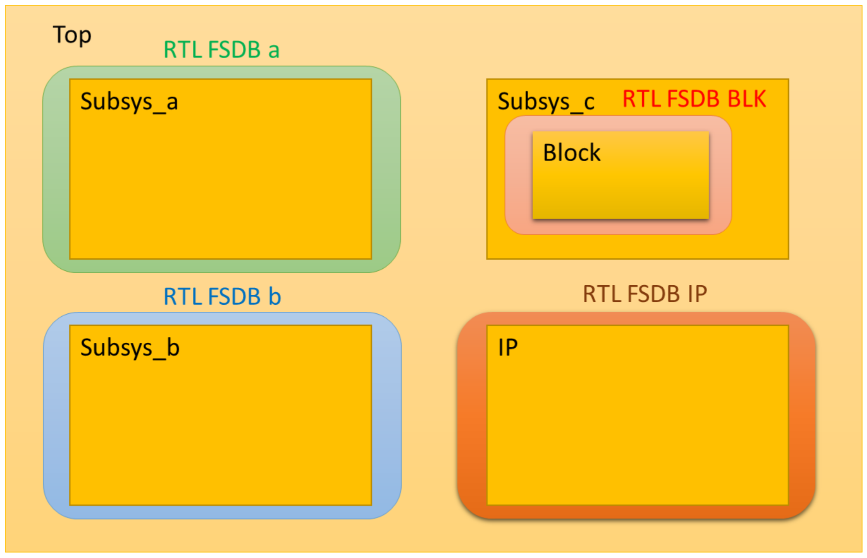


Figure 3. An example design and the corresponding RTL FSDBs for merging and replaying.

In this session, we will go through all the inputs in **Figure 2** with an example design and the corresponding RTL FSDBs illustrated in **Figure 3**, and their formats and the reasons of why they are required will also be introduced. In addition, we will also briefly discuss the methods for checking the quality of each input. This is very important for minimizing the costly debugging iterations during the project execution.

For the **RTL** file list, its input format is the same with the file list used for VCS, so we suggest using VCS to compile the RTL file list and confirming all compiling error cleaned before inputting to Siloti. The RTL design information will be used for generating the mapping information between the RTL design and the gate netlist.

For the **Gate** file list, its input format is the same with the file list used for VCS, so we suggest using VCS to compile the RTL file list and confirming all compiling error cleaned before inputting to Siloti. The gate design information will be used for generating the mapping information between the RTL design and the gate netlist. In addition, the gate design will also be used to generate the Siloti instrumented design for replaying the RTL waveforms.

For the **SVF** files, those files are automatically generated by DesignCompiler when synthesizing the RTL design, and those files are used for guiding the mapping the registers and primary inputs between the RTL and netlist. Hence, we have to collect all SVF files for each target modules to be replayed (i.e., *Subsys_a*, *Subsys_b*, *Subsys_c*, and *IP* in **Figure 3**). To check the mapping quality, we can check the mapping rate reported by Siloti in "crdb_summary.log".

For the **FSDB configuration file**, the format is illustrated in **Figure 4**. Each line in the configuration file contains four parts. The first part is the RTL FSDB file to be merged and replayed on the netlist. The second part is the time window in the RTL FSDB to be replayed. One thing should be noted is that Siloti currently has one constraint for the time window in the configuration file. That is the duration specified for each RTL FSDBs should be the same. Hence, in **Figure 4**, all durations are exactly 600ns. The third part is the instance scope in the RTL FSDB to be used for replaying (i.e., the source waveform of registers in RTL FSDB to be fetched by Siloti for replaying). The last part is the instance scope in the netlist to be replayed (i.e., the destination registers in the netlist which will be forced by Siloti to generate the merged gate FSDB). For the qualification of this configuration file, Siloti lacks of an effective method to check the correctness of the configuration file. Users have to check it manually.

```
rtl_a.fsdb –begin_time 300ns –end_time 900ns –from_scope tb.Top.Subsys_a –to_scope Top.Subsys_a
rtl_b.fsdb –begin_time 1200000ps –end_time 1800000ps –from_scope tb.Top.Subsys_b –to_scope
Top.Subsys_b
rtl_blk.fsdb –begin_time 5600ns –end_time 6200ns –from_scope tb.Top.Subsys_c.Block –to_scope
Top.Subsys_c.Block
rtl_ip.fsdb –begin_time 1770800ns –end_time 1771400ns –from_scope tb.Top.IP –to_scope Top.IP
```

Figure 4. Example for the FSDB configuration file "fsdb_list" according to the design and waveform given in Figure 3.

For the **RTL FSDB** files, designers have to use "$fsdbDumpvars" system task to dump the RTL waveforms and then collect all required RTL FSDBs for replaying. Although these inputs are very easy to prepare, Siloti also lacks of an effective method to check the mapping rate between the RTL FSDBs and the netlist. Users have to check it manually.

For the **"wi_config"** file, the format is illustrated in **Figure 5**. The "fsdb_list" specified in the *set*

FSDB_list command is illustrated in **Figure 4**. The *set Scope* command is used to specify the topmost scope for replaying. Then, the “G2R.list.gz” and “port.map.gz” are the outputs generated after executing ***Gen. G2R mapping*** and ***Gen. port mapping***. The “G2R.list.gz” contains all registers mapping information between the RTL design and netlist, and “port.map.gz” contains all primary inputs mapping information for each replaying scope specified in “fsdb_list”. The RTL and corresponding netlist mapping points are what Siloti uses to get the waveforms from RTL FSDBs and force the waveforms to, respectively. Next, the “wi_compile.rc” specified in *set simulation_compile_script* is used to give the VCS debug options and post-layout simulation related options. The example command and options of “wi_compile.rc” is demonstrated in **Figure 6**. Still next, the “wi_run.rc” specified in *set simulation_run_script* is used to give the VCS simulation command. Since the VCS simulation command *simv* is straightforward, we do not go through the detail in this paper. The time window is specified with *set Begin_time*, *set End_time*, and *set Time_Unit* commands. According to the example in **Figure 4**, the duration should be set to 600ns. For the “delay_wrap” set in *set Delay* command, readers can refer to [6] for the details of how to extract the delay of register Q pin. Following, the SDF file “Top.sdf.gz”, SDF annotation hierarchical path, and the timing corner are specified with *set SDF* command. Finally, if there is any hard macro required to be excluded from replaying the waveform, those hard macros can be listed in “vc_exclude_mod.list” and specified with *set ExcludeModule* command.

```
set FSDB_list                      = fsdb_list
set Scope                          = Top
set Map                            = G2R.list.gz
set Map                            = port.map.gz
set simulation_compile_script      = wi_compile.rc
set simulation_run_script          = wi_run.rc
set Begin_time                     = 0
set End_time                       = 600000
set Time_Unit                      = ps
set Delay                          = delay_wrap
set SDF                            = “Top.sdf.gz” “Top” “” “” “MAXIMUM”
set ExcludeModule                  = vc_exclude_mod.list
```

Figure 5. Example wi_config file according to the design and waveform given in Figure 3.

```
vcs –full64 –sverilog –f wi_run.f –debug_acc+pp+f+w+cbk+fwn –
debug_region=lib+cell +notimingcheck <post-sim. related options>
```

Figure 6. Example wi_compile.rc for Siloti replaying.

4. Specification of RTL FSDB Merging and Replaying

For a SoC design, there are many nets used to connect between combinational logics, registers, and macros. Hence, if we replay a single RTL FSDB to a certain design scope at a time, there is not thing special to be dealt with. That is Siloti can handle the replaying waveform normally. However, when we try to replaying multiple RTL FSDBs to multiple design scopes, there could be design scopes overlapping or signal conflicts between the interfaces of the scopes. Therefore, we have to define the specification for handling such cases for Siloti to implement. In the following, we will go through each case and the corresponding specification for Siloti to handle it.

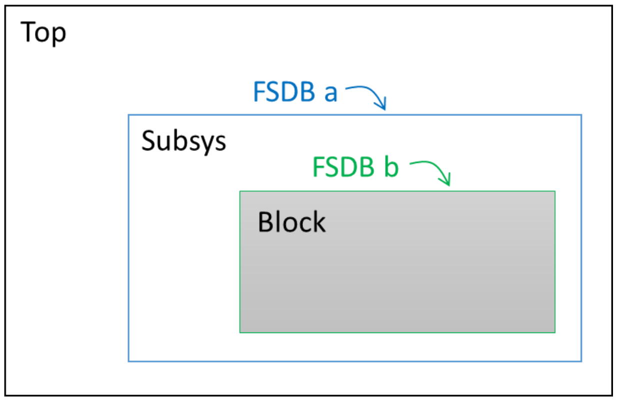


Figure 7. Example for replaying scope overlapping.

The first case to be discussed is that the replaying scopes of multiple RTL FSDBs have some overlaps. For the example illustrated in **Figure 7**, we want to replay "FSDB a" on the *Top.Subsys* scope and "FSDB b" on the *Top.Subsys.Block* scope, so "FSDB a" and "FSDB b" have overlapping on the scope *Top.Subsys.Block*. How do we expect Siloti to handle the overlapping issue? The specification for this issue is "**Last one win**". That is the latter one FSDB will overwrite the overlapping scope of the former one FSDB, if there is any scope overlapping between different FSDBs in the FSDB configuration file as demonstrated in **Figure 4**.Hence, for the example in **Figure 7**, if we want to correctly replay "FSDB a" on the *Top.Subsys* scope and "FSDB b" on *Top.Subsys.Block*, we have to prepare the FSDB configuration file as illustrated in **Figure 8** for Siloti to merge and replay.

```
rtl_a.fsdb –begin_time 300ns –end_time 900ns –from_scope tb.Top.Subsys –to_scope Top.Subsys
rtl_b.fsdb –begin_time 5600ns –end_time 6200ns –from_scope tb.Top.Subsys.Block –to_scope
Top.Subsys.Block
```

Figure 8. Example FSDB configuration file for the replaying case in Figure 7.

The second case to be discussed is that there is signal conflict at the interfaces of the replaying scopes. For the example illustrated in **Figure 9 (a)**, we want to replay "FSDB a" to *Top.Subsys_a* and "FSDB_b" to *Top.Subsys_b*. However, we find that there is a conflict between connected wires *net_t*, *net_a*, and *net_b*, since *net_a* and *net_b* are replayed with different RTL FSDBs. Hence, the question is: "How to determine the waveform on *net_t*?" Based on the concept of the worst-case scenario, we define the specification as "**Replaying the busiest waveform from the connected wires in downstream scopes to the top level wire, if top level wire is not covered with any FSDB**". Hence, for the example in **Figure 9 (a)**, the waveform of *net_t* will be replayed with the waveform of *net_b* which is busier than the waveform of *net_a*. The Siloti generated waveform is demonstrated in **Figure 9 (b)**. One thing should be noted that the wire *net_buf* will not be replayed with any waveform, since the specification is replaying busiest waveform from the downstream connected wires. That is *net_buf* does not directly connected with any of *net_a*, *net_b*, or *net_t*.

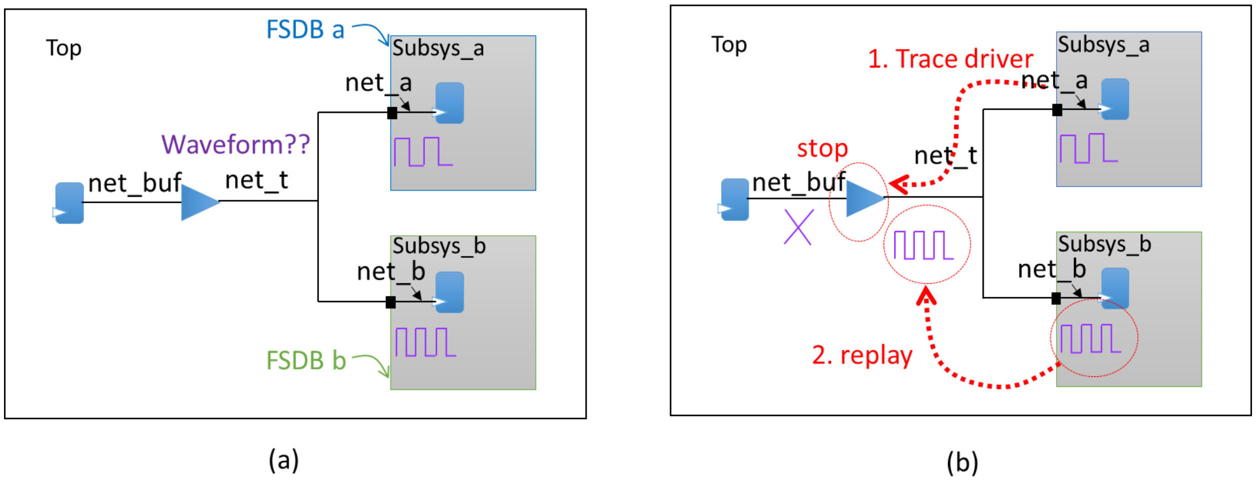


Figure 9. (a) Case of signal conflict at the interfaces of the replaying scopes. (b). Specification for Siloti to handle the signal conflict.

After Siloti implementing the proposed specifications, Siloti can handle the overlapping scope issues and signal conflict issues as expected. However, we could get a strange waveform as demonstrated in **Figure 10**. We can observe that *Top.W3*, *Top.S1.W1*, and *Top.S2.W2* are connected together, but they have different waveforms which are replayed from “FSDB a”, “FSDB b”, and “FSDB c”. If we directly use this waveform to feed into PrimeTime PX to report power, PrimeTime PX will complain the signal conflict issue on *Top.W3*, *Top.S1.W1*, and *Top.S2.W2*. This is because they are considered as the same net in PrimeTime PX, and they are supposed to have exactly the same waveform. Hence, we also have to come out the solution for PrimeTime PX to report the power of the “strange” waveform.

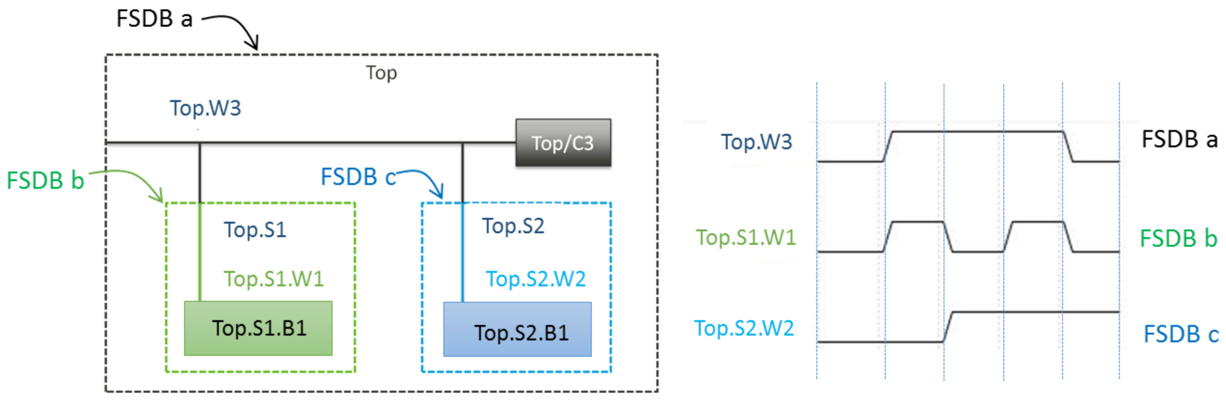


Figure 10. Example of Siloti generated merged FSDB with conflict signals.

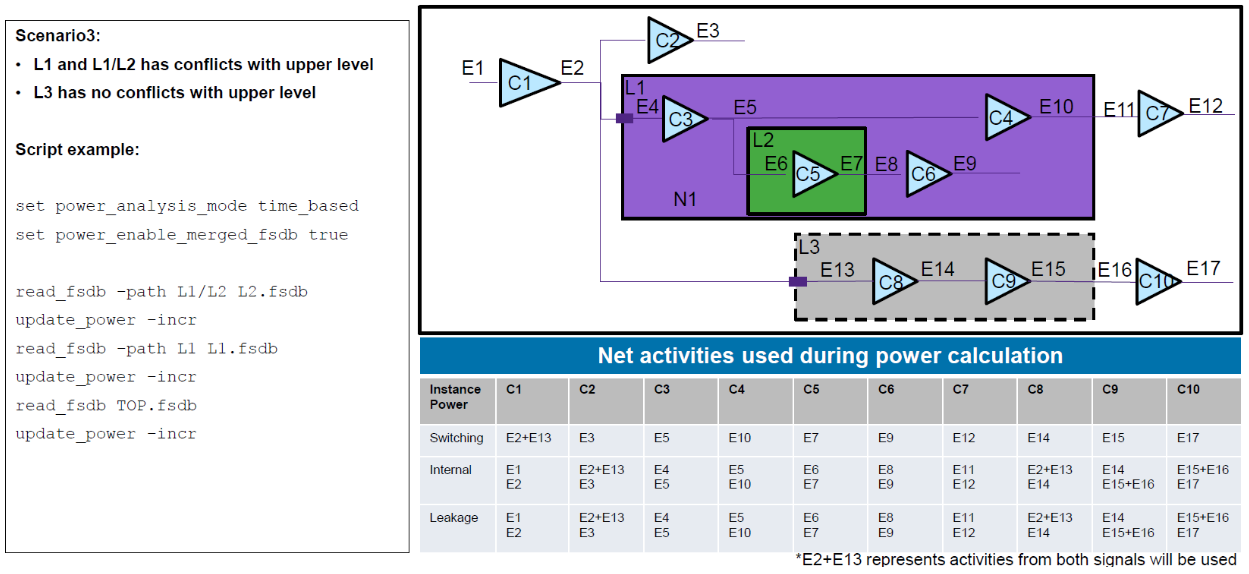
One proposed solution for reporting the power of the conflict waveform is converting the FSDB to the SAIF with the conflict signals averaged. In other words, the key is averaging the toggle and the probability of the conflict signals in the Siloti generated FSDB and then converting to an output file in the standard SAIF format. Hence, if we apply our solution on the example demonstrated in **Figure 10**, the SAIF TC (toggle count) of *Top.W3*, *Top.S1.W1*, and *Top.S2.W2* will be all set to (2+4+1)/3 = 2.3, T0 (*total time the design object has the value 0*) will be (0.4+0.6+0.4)/3 = 0.47, and T1 (the total time the design object has the value 1) will be (0.6+0.4+0.6)/3 = 0.53. Therefore, there is no signal conflict in the converted output SAIF file by applying our solution. Then we can use this SAIF file to feed in PrimeTime PX, ICC/ICC2, or DesignCompiler to report the power of the design. In addition, since SAIF is the IEEE standard, third party tool can also read in the SAIF file smoothly. We have collaborated with Synopsys and implemented this solution as a VC App. The usage of this App is very simple. The example command is as below:

% fsdb2saif merged.fsdb –o merged.saif

% saif_eco –f wi_run.f –saif merged.saif –same_net_map wi_same_net_drv.map –o average.saif

where "wi_run.f" and "wi_same_net_drv.map" are generated in **Extract** step in **Figure 2**.

The other proposed solution for reporting the power of the conflict waveform is enhancing PrimeTime PX to directly support the waveform. However, to focus on the topic of this paper, we will not go through the details of the specification for PrimeTime PX. If readers are interested in the specification, the graph demonstrated in **Figure 11** is a good reference.



Scenario3:

- **L1 and L1/L2 has conflicts with upper level**
- **L3 has no conflicts with upper level**

Script example:

```
set power_analysis_mode time_based
set power_enable_merged_fsdb true

read_fsdb -path L1/L2 L2.fsdb
update_power -incr
read_fsdb -path L1 L1.fsdb
update_power -incr
read_fsdb TOP.fsdb
update_power -incr
```

Net activities used during power calculation

Instance Power	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Switching	E2+E13	E3	E5	E10	E7	E9	E12	E14	E15	E17
Internal	E1 E2	E2+E13 E3	E4 E5	E5 E10	E6 E7	E8 E9	E11 E12	E2+E13 E14	E14 E15+E16	E15+E16 E17
Leakage	E1 E2	E2+E13 E3	E4 E5	E5 E10	E6 E7	E8 E9	E11 E12	E2+E13 E14	E14 E15+E16	E15+E16 E17

*E2+E13 represents activities from both signals will be used

Figure 11. Specification for PrimeTime PX to support the signal conflict waveform.

5. Steps for RTL FSDB Merging and Replaying

In this section, we will go through the details of each step in **Figure 2** and also describe the corresponding output.

The first step **Compile KDB** is using *vericom* command to compile both the RTL file list and the gate file list, and the Verdi KDB will be generated after compilation. Below are the example commands:

% vericom –sv –f rtl_file_list.f –lib kdb_rtl

% vericom –sv –f gate_file_list.f –lib kdb_gate

After the compilation done, you will see two directories generated to save the corresponding KDB for the RTL design and the netlist (i.e., */kdb_rtl.lib++* and *./kdb_gate.lib++*). The KDB will be used for generating the gate-to-RTL mapping information, generating an instrumented gate design for replaying, and debugging.

The second step **Gen. G2R mapping** is using *crdb* command to load the RTL/netlist KDB, indicate the mapping scope, read in SVF files, and set the excluding macros. Below are the example commands:

```
% crdb –RTL "-lib kdb_rtl –top Top" –GATE "-lib kdb_gate –top Top" \
-impSVF " Subsys_a.svf Subsys_b.svf Subsys_c.svf IP.svf " \
-target_scope "Top" –allreg –expGateUncor –top_inport \
-excludeModule vc_exclude_mod.list –expMap G2R.list.gz –crdb G2R.crdb
```

After the *crdb* command done, the gate-to-RTL register mapping file "G2R.list.gz" and mapping database "./G2R.crdb" will be generated. The "./G2R.crdb" will be further used to generate gate-to-RTL primary input mapping information in the following step.

The third step **Gen. port mapping** contains two sub-steps. In first sub-step, we use the Synopsys provided VC App to extract all netlist primary inputs of all replaying scope listed in the FSDB configuration file (i.e., "fsdb_list" in **Figure 4**). Below is an example command:

```
% getModInput_batch.pl –lib kdb_gate –top Top –modules " Top.Subsys_a Top.Subsys_b
Top.Subsys_c.Block Top.IP " –o port.txt
```

Then in second sub-step, we use *crdb* command to load the mapping database, read in "port.txt", and generate the "port.map.gz". Below is an example command:

```
% crdb –crdb G2R.crdb –txt port.txt –expMap port.map.gz
```

The generated "port.map.gz" contains the gate-to-RTL primary input mapping information for all replaying scope. The "G2R.list.gz" and the "port.map.gz" are what Siloti uses to fetch the corresponding waveforms from the original RTL FSDBs and forces to the corresponding signals in the netlist. If these two files are not generated, users should debug the corresponding log file instead of proceeding following steps. Users can also check "crdb_summary.log" to confirm the quality of the mapping result. In general, we suggest the mapping rate should be greater than 99% for all the replaying scope to ensure the quality of the Siloti generated FSDB.

The fourth step **Extract** is using *wisim* command to read in "wi_config" file (including the RTL FSDB configuration file), load the netlist KDB, generate single merged RTL FSDB, and generate the testbench and the instrumented gate design for replaying. Below is an example command:

```
% wisim –lib kdb_gate –top Top –target whatIf_Top –config wi_config –replay_extract
```

After executing this command, the files for Siloti replaying are generated. Those files are "wi_run.f" (deisgn file list for replaying), "wi_test.sv" (testbench), "wi_design.v" (instrumented design), "wi_manip.fsdb" (merged RTL FSDB from the RTL FSDBs specified in the RTL FSDB configuration file), and "wi_same_net_drv.map" (for converting the Siloti generated FSDB to the averaged SAIF file).

The fifth step **VCS Compile** is using *wisim* command to load the netlist KDB, specify the top design scope, read in "wi_config" file (including "wi_compile.rc" file), and invoke the VCS compilation. Below is an example command:

```
% wisim –lib kdb_gate –top Top –target whatIf_Top –config wi_config –replay_compile
```

After executing this command, the VCS database "./csrc" and "./simv.daidir" will be generated.

The last step **Replay** is using *wisim* command to read in "wi_config" file (including "wi_run.rc"), load the netlist KDB, specify the top design scope, and invoke the VCS simulation. Below is an example command:

```
% wisim –lib kdb_gate –top Top –target whatIf_Top –config wi_config –replay_sim
```

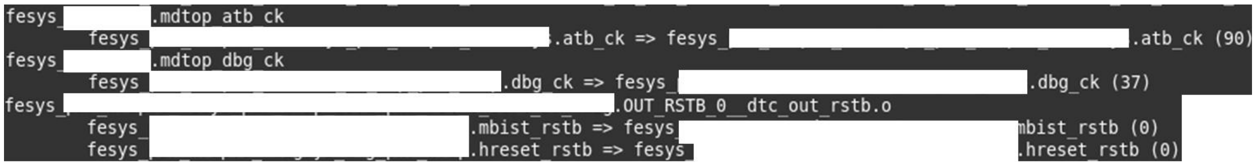
After executing this command, the merged post-layout simulation like FSDB will be generated.

6. Siloti Merging and Replaying Results

Our first experiment is applying the proposed methodology to generate merged post-layout simulation like FSDB for a modem subsystem which is integrated in our certain smart phone project. We collected all the input files as illustrated in **Section 3.** from corresponding designers and executed the commands as demonstrated in **Section 5.** After Siloti replaying done, we can directly open the output FSDB to check. Another method for checking the replaying result is checking text-based report “wi_same_net_drv.txt”. This report contains all the same nets in the interfaces of the replaying scopes, and the toggles of the interface nets are also recorded. The format of “wi_same_net_drv.txt” is demonstrated in **Figure 12**, and an example “wi_same_net_drv.txt” is demonstrated in **Figure 13**. In **Figure 14**, we demonstrated the merged netlist FSDB and the source RTL FSDBs for replaying. We can observe that the signals in the merged FSDB are exactly replayed from different RTL FSDBs with different time windows.

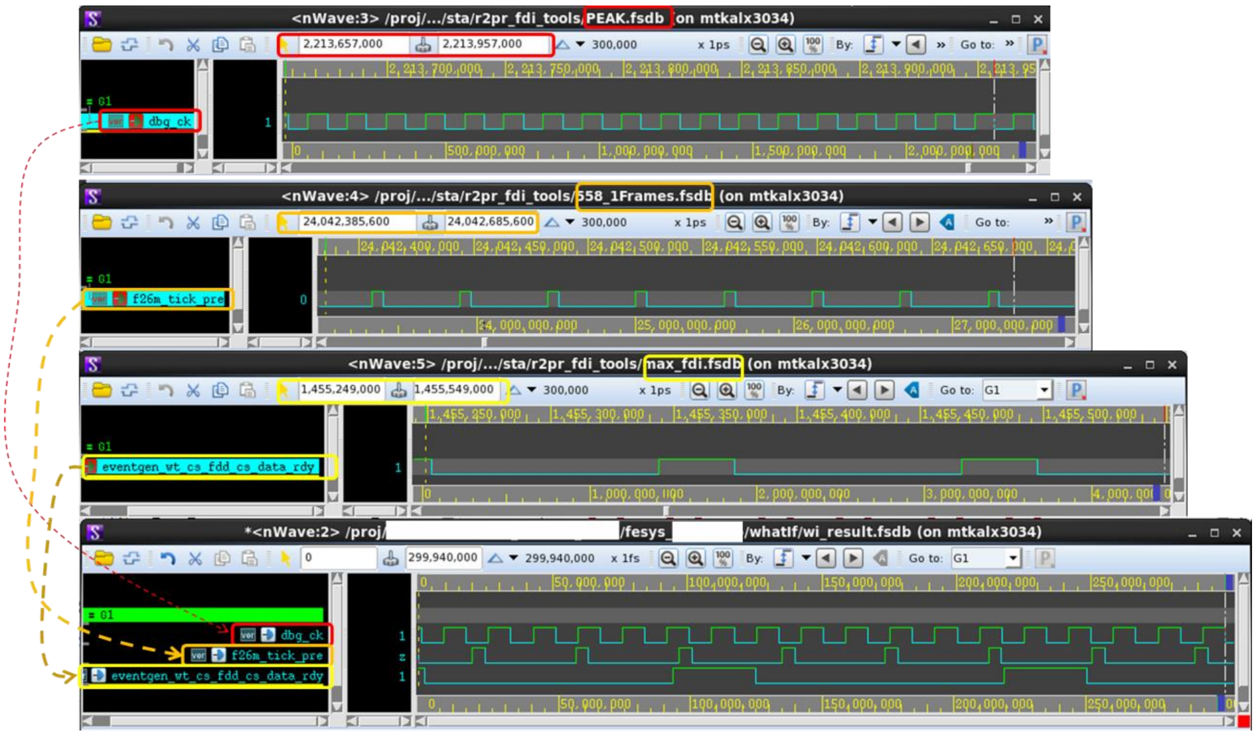
```
<Driver_gate_sig>
        <Sink1_gate_sig> => <RTL signal> (toggle_count)
        <Sink2_gate_sig> => <RTL signal> (toggle_count)
        ....
```

Figure 12. Format of conflict signal report.



```
fesys_[redacted].mdtop_atb_ck
      fesys_[redacted].atb_ck => fesys_[redacted].atb_ck (90)
fesys_[redacted].mdtop_dbg_ck
      fesys_[redacted].dbg_ck => fesys_[redacted].dbg_ck (37)
fesys_[redacted].OUT_RSTB_0__dtc_out_rstb.o
      fesys_[redacted].mbist_rstb => fesys_[redacted]mbist_rstb (0)
      fesys_[redacted].hreset_rstb => fesys_[redacted].hreset_rstb (0)
```

Figure 13. Example report of “wi_same_net_drv.txt”.



<nWave:3> /proj/.../sta/r2pr_fdi_tools/PEAK.fsdb (on mtkalx3034)
dbg_ck
<nWave:4> /proj/.../sta/r2pr_fdi_tools/558_1Frames.fsdb (on mtkalx3034)
f26m_tick_pre
<nWave:5> /proj/.../sta/r2pr_fdi_tools/max_fdi.fsdb (on mtkalx3034)
eventgen_wt_cs_fdd_cs_data_rdy
*<nWave:2> /proj/[redacted]/fesys_[redacted]/whatIf/wi_result.fsdb (on mtkalx3034)
dbg_ck
f26m_tick_pre
eventgen_wt_cs_fdd_cs_data_rdy

Figure 14. Source RTL FSDBs and Siloti generated merged FSDB.

Our second experiment is applying the proposed methodology to generate merged post-layout simulation like FSDB for a smart phone SoC design which has 40 million instances. We replayed 8 RTL FSDBs to 8 different subsystems. However, the total run time was over 28 hours. After collaborating with Synopsys to optimize the performance of Siloti, the run time is reduce to about 12 hours. The toggle rate map of the generated merged netlist FSDB is demonstrated in **Figure 15**. Hence, with the proposed solution, designers do not have to spend couples of days to create a very complex pattern to trigger a whole chip into a busy state, and they neither have to spend couples of weeks to build the post-layout simulation environment to generate the netlist FSDB. With the proposed methodology, the whole process becomes a one day job when all input files are ready. This is a brand new method for designers to generate waveforms for the power estimation or the IR signoff.

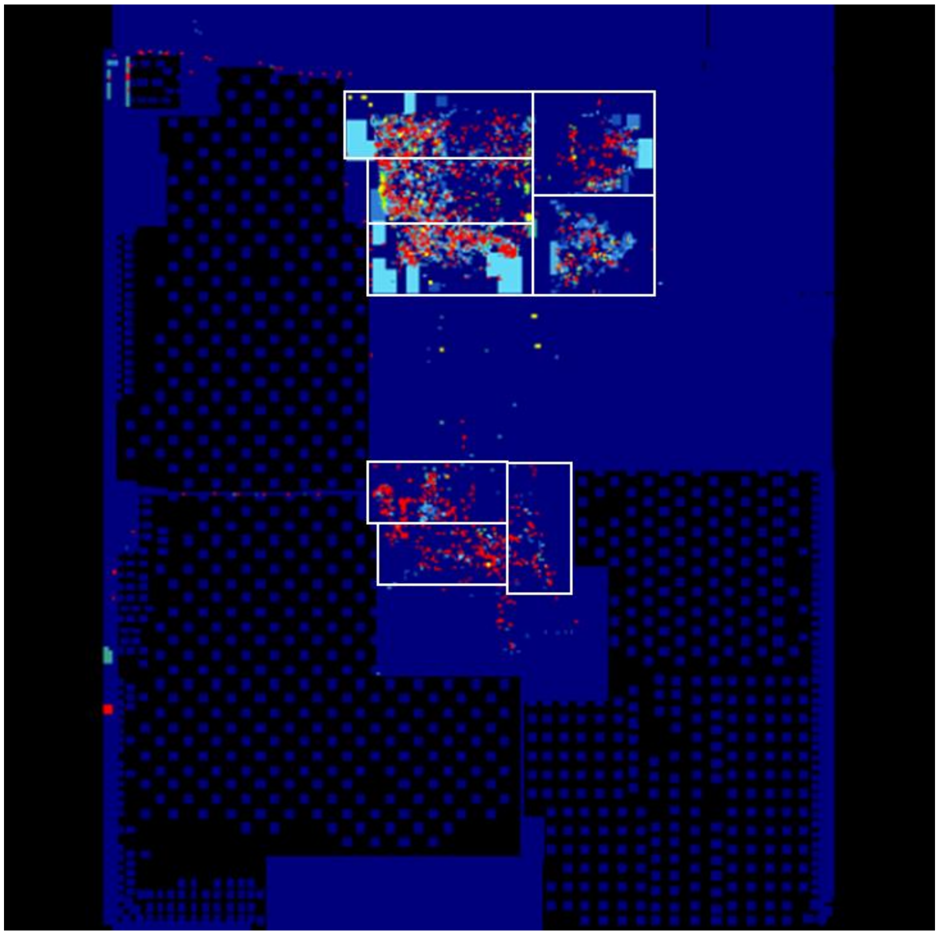


Figure 15. Toggle rate map of Siloti generated merged FSDB for a smart phone SoC design.

7. Conclusions and Future Works

In this paper, we propose a novel waveform generation methodology for the power estimation, IR signoff, power optimization, and potential power related applications. We collaborate with Synopsys to enhance Siloti for concurrently replaying multiple RTL waveforms to different design scopes and generating a single merged post-layout simulation like waveform for a subsystem or even the whole chip. With the proposed methodology, the waveform generation becomes easier than the traditional process. That is designers neither have to spend couples of days to prepare a complex pattern nor have to spend couples of weeks to build the post-layout simulation environment to generate the netlist waveform. Now, it becomes a one day job with all inputs are ready. We also proposed a solution for solving the signal conflict issues in the merged waveform by averaging the wires which have conflicts and then converting to a SAIF file. By converting the merged FSDB to the IEEE standard format, all power related tools can utilize the merged waveform to do certain power applications. We also collaborate with Synopsys to enhance PrimeTime PX for supporting the waveform with conflict signals. Results are also demonstrated in this paper for proving the effectiveness of the proposed methodology.

The future work contains:

1. Enhance Siloti to generate separate gate-to-RTL mapping rate report for each target replaying gate scope.
2. Enhance Siloti to generate separate RTL FSDB mapping rate report for each target replaying gate scope.
3. Develop new solutions for efficient debugging between the RTL FSDBs and the merged gate FSDB.

We believe that with these enhancements ready, users can execute this new methodology efficiently and smoothly.

8. References

[1] *https://www.extremetech.com/extreme/209096-in-depth-with-qualcomm-on-the-snapdragon-810*, 2015.

[2] VCS FGP, *https://www.synopsys.com/verification/simulation/vcs.html*, 2017.

[3] RocketSim, *https://www.cadence.com/content/dam/cadence-www/global/en_US/documents/tools/system-design-verification/parallel-simulation-wp.pdf*, 2017.

[4] Pooja Saseendran, Syed Thameem, Kaushik Saiprasad, "*The last microwatt – Challenges in Pre Silicon Power Estimation for Low Power SoCs*", SNUG India 2013.

[5] Ophir Turbovich, "*Speeding Power Estimation from Weeks to Hours*", SNUG Israel 2013.

[6] Yung-Jen Chen, Hsing-Han Tseng, Jia-Shiun Yang, and Lance Liu, "*An Efficient Methodology to Achieve Accurate IR Analysis in Early Stage*", SNUG Taiwan 2016.