Automatic Investigation of Power Inefficiency

Kuo-Kai Hsieh[1], Wen Chen[2], Monica Farkash[2], Jayanta Bhadra[2], Li-C. Wang[1]

[1]University of California, Santa Barbara (kuokai@umail.ucsb.edu, licwang@ece.ucsb.edu)
[2]NXP Semiconductors N.V. ({wen.chen, monica.farkash, jayanta.bhadra}@nxp.com)

***Abstract*- This work presents an automatic method to identify and suggest to designers missed power optimization opportunities by finding useless activities in the design during simulation, where useless activities are defined as toggling events that do not contribute to a given task. The proposed method leverages functional regression test suites and coverage collection mechanisms, which are both common in modern verification methodologies hence the method can be seamlessly integrated into existing environments. We present the evaluation of this method on a commercial low-power SoC design with details on the achieved results which include two power-inefficiency issues.**

I. Introduction

The power density of integrated circuits has been increased as the technology node scales down. Therefore, heat dissipation has become a problem, which has raised the requirements to reduce the power consumed by integrated circuits. With the emergence of Internet of Things, there have been increasing requirements on the extended battery life of devices such as sensors nodes in wireless networks, wearables and implants. Ultimately, these requirements are translated to stringent low power requirements for integrated circuits and power efficiency has been a main focus for today's complex System-On-Chip (SoC) designers.

Power consumption of integrated circuits consists of two main parts: dynamic power and static power. Dynamic power is the sum of two factors: switching power and short-circuit power. Switching power can be expressed by the following formula:

$$P_{switching} = \alpha{\cdot}f{\cdot}C_{eff}{\cdot}V_{dd}^{2} \qquad (1)$$

where α is the switching activity, f is the clock frequency, C_{eff} is the effective capacitance of the circuit and V_{dd} is the supply voltage. Short-circuit power can be expressed by the following formula:

$$P_{short\text{-}circuit} = I_{sc}{\cdot}V_{dd}{\cdot}f \qquad (2)$$

where I_{sc} is the short-circuit current during switching, V_{dd} is the supply voltage, and f is the clock frequency. Static power, dubbed as leakage power, is a function of the supply voltage, the threshold voltage and the transistor size.

Various techniques have been proposed to reduce power consumption of integrated circuits, ranging from architecture level to technology level, including power gating, clock gating, dynamic voltage frequency scaling (DVFS), logic restructuring and resizing, pin swapping, operand isolation, substrate biasing, and so on.

Power gating and clock gating have been the most effective and widely used approaches for power reduction. Power gating relies on shutting off the blocks or transistors that are not used, which can reduce leakage power effectively by 10-50X. Clock gating shuts off blocks or registers that are not required to be active. It can reduce about 20% of dynamic power. Power gating and clock gating can both be performed at domain/block level, or at the leaf (transistor/register) level. Power gating, when performed at domain/block level, is supported by the design flow using power intent formats such as UPF and CPF. While it is an effective power reduction technique, power gating increases the design complexities since it requires specific power down sequences, isolation cells, and state retention cells. On the other hand, clock gating has lower overhead.

Both power gating and clock gating require the identification of situations where such optimizations are possible. Block level power gating and clock gating are usually driven by the power management software, which dictates when to shutoff the power or clock of certain block based on the application running on the chip. Logic/transistor/register level power or clock gating is usually achieved by instrumenting some logic in the design phase that shuts off certain logic blocks based on the condition of other logic blocks. Such power optimizations are generally based on manual analysis and are strongly dependent on the knowledge of the design at hand and the experience of each individual designer. There have been several research works to automate the identification of power optimization opportunities by structural/formal analysis [1] [2] [3]. However, these analyses are usually applied at a local scale and cannot find power opportunities that cross block boundaries. The identification of areas where power optimizations are possible at RTL level is a critical but a difficult and imprecise process.

This work brings forward an automatic method to identify and suggest to designers missed power optimization opportunities during RTL simulation. The novel method leverages existing functional regression test suites and thus is easy to implement and can be integrated seamlessly into any design methodology. In addition, the diversity of tasks of functional regression test suites is a perfect fit to the proposed method. The method starts with the basic assumption that any activity detected during the simulation of the design should be an activity required for a task to complete correctly. Reversing the above statement implies that any activity not required for a task must not be detected during its execution otherwise, it shows a missed power optimization opportunity. We will elaborate this idea in the next section and detail the proposed method and implementation in the following sections. Finally, we will present the experimental results from applying the method on the verification environment of a commercial design before we conclude the paper.

II. BASIC IDEA

Let us assume that we have a perfectly power-efficient SoC design. In such a design, there would be no useless activity. Given a task to be fulfilled, a useless activity is considered an activity in the design that will not affect the correctness of the execution of the task. This means that, if we *turn off* the part of the design containing the useless activity, the execution of the task will not be affected. In practice, the design would not be perfectly optimized for power therefore there could be useless activity in the execution of certain tasks. Any activity that we can identify as useless could point to a potential power optimization opportunity.

Fig. 1 illustrates the basic idea. Our process identifies the activity within the design during the simulation run of a task, and records it to use it in the future as a baseline. Subsequently we run the same task on the same design, each run with slightly different testbench instrumentation. The instrumentation changes the original baseline by excluding (by artificially disabling) a different subcomponent each run. If the disabled subcomponent would have contained useful activities, the execution of the task is expected to fail. However, if the simulation on the instrumented design does not report failure, it points to a potential problem. The problem could be missing checkers or incorrect checkers in the testbench, which allows a task to be considered as passed even though it failed, or it could show useless activity in the subcomponent that was turned off by our instrumentation. Both are critical in an HW development methodology. The first requires enhancement of the testbench because missing checkers can result in functional bugs going unidentified. The second identifies areas with potential power optimizations.

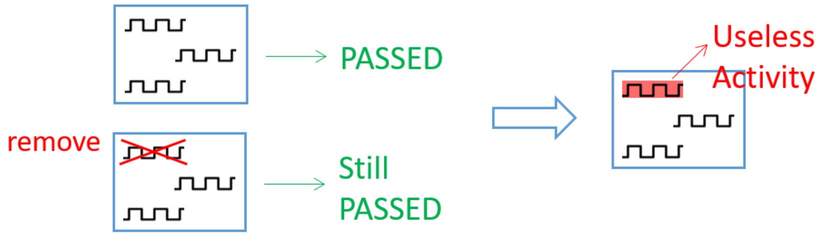

Figure 1. Illustration of the basic idea

The identified areas are opened as issues to designers, who then evaluate the merits of each case, and decide accordingly. Our approach is the only one automatically identifying the location in a design where a power optimization can be suggested, and on top it has the advantage that it provides the exact task, and conditions, for which that optimization could be done, to allow a quick assessment of the validity of the optimization and its overall power savings potential. If the optimization should be indeed implemented in the design is a decision left to designers.

III. METHODOLOGY

Fig 2. illustrates the overall flow of our methodology. The flow is comprised of four major steps as detailed in the following four subsections.

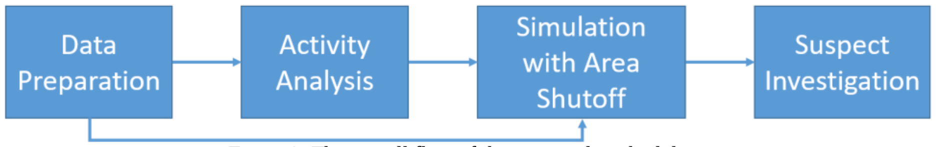

Figure 2. The overall flow of the proposed methodology

A. *Data Preparation*

There are two conditions that need to be fulfilled before we apply this solution: we need access to functional regression tests and we need to partition the design. While regression tests suites are already developed and readily available as essential components of the existing verification process, the partitioning of the design into smaller, well defined areas, is an effort required by this solution.

The whole SoC design can be divided into different areas based on user’s choice. Usually, the partitioning is based on the design module hierarchy and functionality. Modules designed for the same functionality are usually grouped in an area. The user can specify the targeted design area where he/she wants to uncover the power inefficiencies. The targeted design area for investigation can be based on the designers’ experience or on hotspots identified from the heat map of the power dissipation of the chip.

Fig. 3. Shows how regression tests provide toggle coverage and partitioned areas provide important signals used later on in the solution.

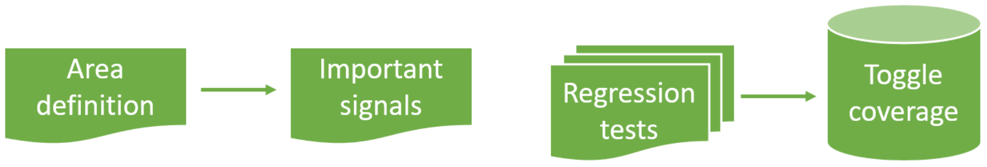

Figure 3. The required source and the result of data preparation.

The first source of data is the coverage gathered from running the regression test suite. Collecting toggle coverage from functional regression tests is straightforward since coverage collection is supported by the majority of modern simulators. As will be discussed later, only the number of toggling for each output of each area is required.

The second important information is represented by something we call important signals. From each area defined during partitioning we extract important signals. The important signals of each area are the signals that enable activity analysis and the signals that enable area shutoff. As will be described later, the output signals of each area fulfill both requirements. In our implementation, we consider each instance of a module as a different partition. Given the hierarchical path of an instance of a module we can extract the important signals automatically via Cadence Simvision with the developed TCL script:

```
# print the signals of a given instance to stdout
proc retrieveSignals {path} {
    dbfind set -scope $path
    dbfind search
    set signals [dbfind find -result %n]
    puts $signals
}

# read database
database open <Simvision_database_file>

# set output signals only
dbfind open -database <Simvision_database> -type signal \
    -searchinputs False -searchoutputs True -searchinouts False \
    -searchinternals False -searchfibers False -searcherrors False \
    -searchassertions False -signalformat path -scopeformat path

retrieveSignals <instance_hierarchy_path>
```

B. *Design Activity Analysis*

The method requires running each test under different instrumentation, each such instrumentation resulting in the disabling of a different area. Theoretically we would need to run the cross product of all tests and all areas. Due to practical reasons we need to manage the amount of pair (test, area) we decide to run for our analysis.

As illustrated in Fig. 4, we first choose the areas we want to analyze by ranking them according to the amount of activity they exhibit, then, for each such area, we choose the tests to be used for that particular area, based on the coverage each test brings to that area.

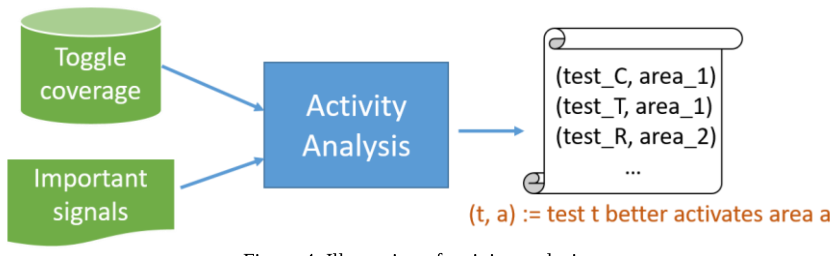

Figure 4. Illustration of activity analysis

Deciding which metric to use to measure activity in a design area requires practical considerations. A simplistic metric based on watching the activity of all signals would not work since some signal toggling is inevitable regardless whether a design area is active or not. For example, a peripheral can be disabled but its bus input signals would still be toggling, which could be considered as activity of a design area under a simplistic metric although it should not. We studied which signals are good indicators of activity of a design area. The result of our research shows that flip-flops and primary outputs are acceptable indicators of activity of the areas. Users can also provide information regarding important signals in the design area and designate them as the indicators.

A very important difference is made by the level of detail we use in our analysis. For example, we can consider the toggling of a signal as a boolean variable, i.e. whether the signal is toggled or not, or as an integer, i.e. how many times the signal switches while running the test. Our research shows that measuring the toggling in integer can better capture the activities in an area while running a test, which confirms our intuition. For example, suppose test A uses DMA to move 1-byte data and test B uses DMA to move 4k-byte data. Their toggling of DMA signals in boolean may be the same but their toggling of DMA signals in integer differ greatly. As a result, the metric is defined as the sum of the toggling of the indicator signals in integer.

The automatic analysis proposed by this article was performed area by area and is based on the assumption that among the regression tests there would be at least one that exercises each area. For each area, we compute the metric for each test based on the toggle coverage results. The computed value measures the activity resulted from each test on the targeted area. Let the highest activity be A. We define a threshold k so that only the tests with the metric value greater than k*A will be considered significant enough to be rerun under instrumentation. Thus, for each targeted design area, we can derive a list of tests that we can use for our method. We can continue to reduce the number of tests, depending on our resources.

C. Rerun simulation with Active Area Shutoff

The next step is to rerun the active tests with the targeted design area *turned off*. We achieve the effect of turning off a design area by freezing its output signals. Commercial RTL simulators support the features of signal freezing using *force* statements via the TCL interface. This approach avoids re-compiling the design and saves computation time and disk storage.

Deciding when to turn off an area is another important issue. Identifying the exact time when the given area is active under a given test is not a trivial task. To obtain this information, dedicated monitors could be implemented to report the activity of each area. To skip this implementation, a simple approach is to turn off the area immediately after power-on reset, and then never turn it back on. Note that this simple approach may miss some power optimization opportunities because of its coarse control. Below is a sample TCL script for IUS to freeze output signals of a given area after power-on reset.

```tcl
proc freeze_sig {sig} {
 force $sig #$sig
}

proc freeze_area {
 freeze_sig <sig_1>
 freeze_sig <sig_2>
 ...
 freeze_sig <sig_N>
}

run 100 ns
# Set a break point for freezing the given area
stop -name break_rst -delbreak 1 -condition {#<end_of_reset_indicator> == 1'b1} \
    -continue -execute {freeze_area}
run
```

Where sig_1, sig_2, …, sig_N are obtained from the data preparation step.

If a design area plays a role in the execution of a task, turning it off would cause the task to fail. Presumably, the tests would fail with the design area disabled. For those simulation runs for which the test still passes in spite of the disabled area, further investigation is required.

D. *Suspect Behavior Investigation*

Fig. 5 shows the flow of the suspect behavior investigation. For those simulation runs that still pass, there would be two possible reasons: (1) the design area does not affect the execution of a task and therefore it can be turned off during the execution of that task. (2) There are some weaknesses in the testbench so that the failure of the task is not detected, resulting in a false simulation pass. The first case indicates that a power optimization opportunity is identified so that power/clock gating of certain logic can be applied. The latter case indicates that the testbench quality needs to be improved by adding the necessary checkers.

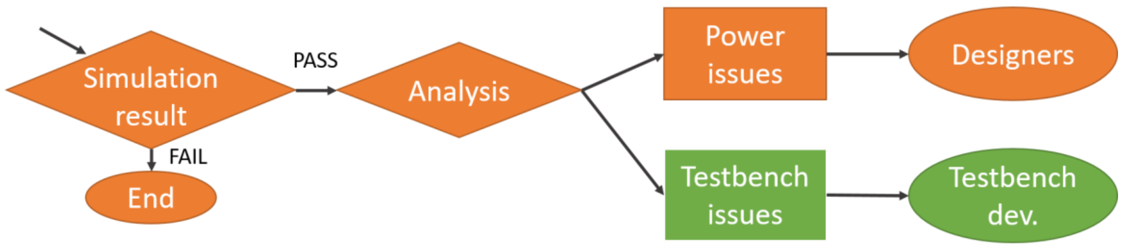

Figure 5. Illustration of suspect investigation.

IV. Experimental evaluation

A. *Experiment Setting*

We implemented the described methodology and applied it on a latest dual-core commercial microcontroller SoC design targeting ultra-low-power applications. We used without additional changes the in-house verification environment, which was a C-test based environment. The C stimuli were compiled into machine code and then executed on cores in RTL simulation. The correctness was ensured both by self-checks in C stimuli and the checkers in the testbench.

We leveraged the already existing functional regression test suite consisting of over 1400 tests. We divided the design into 100 different areas. Toggle coverage results were collected and analyzed, resulting in the active tests for each design area based on a threshold. Due to limited simulation budget, we only re-simulated at most 200 high-coverage tests for each area. Note that after the data preparation step, the proposed method can run by its own and continues reporting the suspects of power inefficiency. Also, note that we applied this methodology near the end of the verification process of the SoC, where the design had been highly optimized and the testbench had been comprehensive.

B. *Experiment Results and Analysis*

We found out nine suspects of power inefficiency. By manually analyzing those suspects, we identified two definitely missed power optimization opportunities in the design and three testbench issues. Our analysis also found one possible missed power optimization opportunity that we decided would not bring real power savings. The other three were false positives.

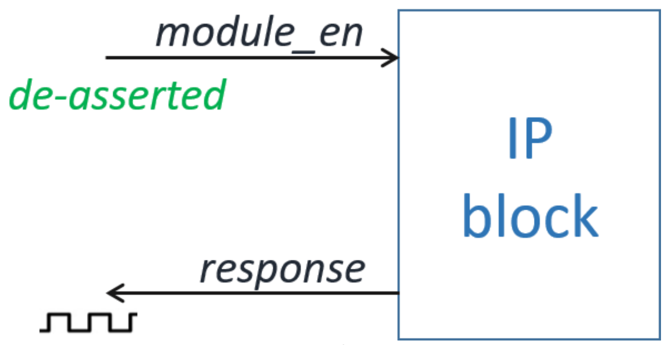

Figure 6. Illustration of design issue 1

Fig. 6 shows one missed power optimization opportunity automatically identified by our method. Our solution revealed unnecessary signal switching at the primary outputs of an IP. As shown in Fig 5, *module_en* is a primary input of the IP block, which enables/disables the IP. *xfr_err* is a primary output of the IP. *xfr_err* fans out to some other blocks and affects the computation at other blocks. However, by examining the logic carefully, we discovered that *xfr_err* only affects the computation of the other blocks if *module_en* is asserted. In other words, *xfr_err* should be controlled by *module_en* so that it will not switch when the IP block is disabled. However, in the design, *xfr_err* was not controlled by *module_en* so it toggled frequently when it did not contribute to the computation. *xfr_err* fans out to multiple places and thus the load capacitance could be large. Therefore, the unnecessary toggling of *xfr_err* is a waste of power.

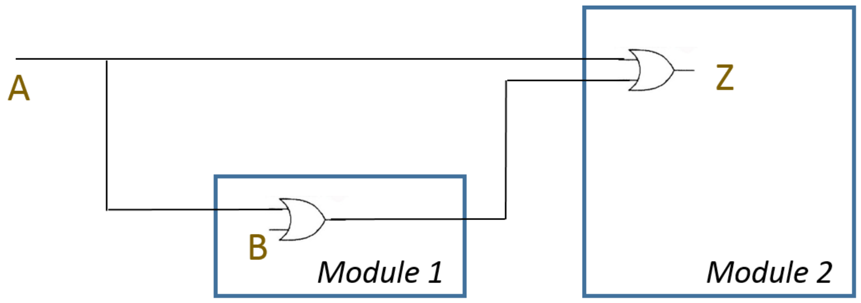

Figure 7. Illustration of design issue 2

Fig. 7 shows another missed power optimization opportunity in the design this time in relation to redundant logic. Some tests passed while Module 1 is shut off. As shown in Fig. 7, the root cause is that signal A is ORed twice, i.e. Z = A | B | A. This was not identified by the IP designer because the logic involved spreads across multiple modules. The redundant logic leads to more power, area and worse timing.

As mentioned before, there are situations when suspected behaviors are not real optimization opportunities. One such possible missed power optimization opportunity involves the logic used to send interrupt signals to both cores. In the test, only one core was configured to accept the interrupt while the other core would block the interrupt. Therefore, the activity of sending interrupt to the blocked core is deemed useless. However, it is not very easy to optimize this inefficiency since the blocking signal is deep inside the core and thus is possibly physically distant from the logic sending the interrupt. Pulling a long wire could also increase power. Therefore, it is better not to gate the logic sending the interrupt due to the design trade-off.

The method pointed three times to situations in which the test should not have been declared passed. Three of the identified issues were related to missing checkers. One test verified some functionalities under *stop mode*, but it lacked checkers to verify whether *stop mode* is entered or not. Another test lacked checkers for AHB protect functionality. The other testbench issue was related to redundant configuration. In the test, a block was set to its default mode multiple times, though even those actions of setting the default mode failed, the block was still in default mode. There was no checking mechanism to check whether the action of setting the block to default mode actually succeeded or not.

Of the three false positives, two were due to improper division of the area boundaries so that some signals were improperly counted as good indicator signals. The other one is a special case, where the output signals were clock signals. These clock signals were toggling before the normal test stopped the clock. Since it is a clock signal, the toggling in integer count is very high, but they should not be counted as real activity. The false positives were resulted from the imperfect design area division and activity metrics.

V. Discussion and conclusion

In this paper, we proposed a methodology that automatically uncovers missed power optimization opportunities in the design or possible testbench weakness. The proposed methodology has the following advantages:

1. It leverages readily available functional regression test suites and analyzes the toggle coverage that is commonly collected in a practical verification flow. Thus, this solution can be seamlessly integrated into the design flow with little overhead.
2. The existing static analysis methods usually worked at each module locally and could miss power optimization opportunities that spread across module boundaries. This methodology can identify such missed opportunities across multiple modules. The experiment results also show that it can discover the opportunities on highly optimized designs.
3. Besides missed power optimization opportunities, the methodology can also be used to certify a testbench and point out the places where the testbench needs improvement. It is a testbench certification approach in the context of low-power verification.
4. It is the only automatic method of this type, which can be used to point out to designers the areas where power optimizations might have been missed.

Considering the very low price to add this solution versus the unique benefits it brings, it can be easily integrated into existing methodologies and used to help improve the quality of our designs.

Acknowledgment

We would like to thank Matthew Becker for his help with the verification environment setup and investigation of uncovered issues.

References

[1] N. Raghavan, V. Akella, and S. Bakshi, “Automatic Insertion of Gated Clocks at Register Transfer Level,” Proceedings. Twelfth International Conference on VLSI Design, Jan. 1999. pp. 48-54.
[2] F. Theeuwen, E. Seelen, “Power Reduction Through Clock Gating By Symbolic Manipulation,” in Symp. Logic and Architecture Design, Dec. 1996, pp. 184-191.
[3] C. Eisner, M. Farkash, “Method for multi-cycle clock gating,” U.S. Patent 8245178, Publication date: Aug 2012.