Using Small Benchmark to Train Models of Switching Frequency for RTL Design

Xingyu Li
Department of Electrical Engineering and Computer Science
University of California, Berkeley
xingyuli9961@berkeley.edu

***Abstract*—This paper presents a tentative methodology that fast estimates the switching frequency behaviour of signals in the RTL designs in the software simulation time. First it runs a small assembly test on the hardware and uses VCD dump and instruction trace log to extract the signal activity. Then it uses the data extracted to train the model and estimates the switching frequency of different signal for any given sequence of functions. Previously, there are proposed methodology that needs long-time cycle-by-cycle simulation or provides power estimation on a larger scale. On contrary, this project intends to find a signal switching frequency pattern after training and give a convenient estimation for any program without simulating the program on the hardware target. This method is demonstrated with simple designs such as Riscv-mini, and can be also used for more complex designs.**

I. Introduction

Power efficiency has become an important constraint for both systems remaining low power consumption and chips reaching high performance without breaking thermal limitations. As Moore's law is likely to slow down and become less guaranteed in the future, designing circuits that meet the power requirement will impose great challenges. Then, fast and accurate power estimation method in different steps of the design process will be a necessity for designers to modify the circuits timely and achieve their goals.

Previously, there are several micro-architecture-level power analysis tools employed by computer architects, such as [1], [2], [3]. These methods conveniently allow researchers to do analysis and optimization without developing RTL design. However, these methods needs to use existing hardware simulator such as [4] and [5] and, as a result, limits the design to be compatible to the simulators and can hardly support novel, non-traditional designs. Also, they also require long simulation time to collect all the architectural activities.

An alternate option is using commercial CAD tools to get accurate power evaluation for the RTL implementation. In order to cope with long simulation time, several methodology [6], [7] and [8] train macro power models between power and selected signals, and use FPGA emulation as a faster way to collect samples of signal activities.

Inspired by both types of methodology mentioned above, this projects demonstrates a trial to model signal switching frequency, a major parameter for the dynamic power, which can be later used for further power estimation. First, signal activities are extracted from the VCD dump and a design-specific switching frequency model is constructed and trained with respect to the instruction codes. After the model is built, switching frequency behaviour can be estimated for any give sequence of instructions.

II. Related Work

In the architectural level, using cycle-accurate hardware simulator like [4] and [5] to collect data and employing methodology such as [1], [2], [3] to build model for power estimation is widely used. The Aladdin [3] is a pre-RTL, power performance simulator for accelerator-centric systems. It takes in algorithms that describe accelerators and uses dynamic data dependence graph to model the signal activities. With the signal flow is correctly analyzed, it can provide a cycle-level power estimation. However, Aladdin and methods alike remain in the architectural level and can not reflect the power difference between different implementation of the RTL, especially for those specially customized circuits.

In order to take the actual RTL design into account, there are some proposed probabilistic models to estimate the switching activities in the circuit level. [9] proposed to use Bayesian Networks to build a high accuracy model for the combinational circuits.

Another widely used method is to use hardware monitoring counters [10], [11]. For an existing physical system with these counters, it can provide a fast power estimation. However, for new designs, these methods have such a high overhead that made them infeasible.

There are also interesting methods [6], [7] and [8] that automatically builds macro power model on signal activities and gives runtime estimation with the aid of the FPGA emulation. Simmani [8] exploits the observation that signals can be clusters by their switching pattern and the signals in each pattern have similar effects on the dynamic power dissipation, and it cluster signals from the VCD dumps from RTL simulation. Then it trains the selected signals against cycle-accurate power traces from commercial CAD tool to build regression models. It then automatically adds counters in the FPGA design and the software can continuously pull information from the counter and provide a relatively instant power estimation. Simmani focuses more on the general power consumption of a complex design running applications, but it might not be help if more detailed, cycle-level information is needed.

Almost none of the methods provides a fast cycle-level power estimation in the RTL level. This project proposes to build a macro model for the signal switching activity for the RTL design. If this model can give an instant and relatively accurate switching frequency pattern for any given sequence of instructions, then the designers can speed up their progress. And, this is the motivation for the project.

III. Methodology

There are four steps in the methodology. First, we need to run several programs on the target design and store the VCD dumps and the instructions trace files. After data collection, decoding step transfers instructions into simplified and standardized form, and read the VCD dumps for later analysis. Then, the modeling and training step construct a macro model of the switching frequency using the VCD dumps of sufficient workload. Finally, we can test the model with another programs and compare the estimated switching frequency with their actual output from the VCD dumps of the RTL simulation, and make updates if needed.

A. Data Collection

First we need to compile the RTL design in the Synopsys VCS to further run applications on the design and derive the signal activities. Then, I need to run simulations of micro-benchmarks and some random instruction streams are for initial model training. The collection of benchmark should be sufficiently large and full of variation so that 1) they contain a comprehensive variation of different instruction-transition types, 2) and there are enough data for later model training and verification. The final version of the special short program should satisfy this requirements alone.

B. Decoding

For an arbitrary functional circuit block, we can concatenate the meaningful part of all its inputs and form a sequence of standard "instructions". For instructions using standard ISA, the decoding stage might be optional. However, as instructions are generally formatted in a way that is friendly for hardware usage but not for modeling for this purpose. We can decode into a more concise format with respect to their ISA type, such as simplifying an instruction by using its opcode. Also, the signals and their cycle-by-cycle switching activity should be extracted from VCD dumps for model training.

C. Modeling and Training

A simple polling algorithm to record the switching behaviours for different types of instruction-transitions and building a model upon that seems quite effective for the data set used in this project. Recurrent neural networks is another choice as the instruction trace is a temporal sequence.

D. Prediction, Validation, and Benchmark Modification

After the model is built, we can use the model to predict the estimated switching times during the executions of the instructions. With the frequency of the clock provided, it can calculate the switching frequencies. Finally, we compare the estimated results with the actually VCD dumps from the RTL simulation, and make some modifications to the special benchmark we write and other modeling parameters if necessary.

IV. Experiment Stage and Results

A. Experiment Setup

The hardware target design chosen in this project is riscv-mini. It is a simple RISC-V 3-stage pipeline processor with simple instruction and data caches. The reason to choose riscv-mini is that I can run iterations of different RV32I programs simulation in a short time and it has 407 different signals, which is not too large to start this trial.

With respect to the codes ran on the riscv-mini, there are standard assembly tests and several larger benchmarks, such as median, multiply, towers and so on. A custom assembly test written by myself is also set up and needs to be changed later for better modelling the switching behaviours.

B. Initial Observations

After collecting all the simulation results and post-processing the data, I first tried to investigate if there's some pattern in the switching behaviour. Though there's no obvious pattern for a certain type of opcode, it seems that a certain transition from two opcodes have similar switching behaviour of certain signals over time. As the decoding technique on the instruction doesn't consider the data-dependent behaviours, I move my focus onto the control signals that are related to the instruction type/function instead of the data.

C. Training the models

For this step, I took nine small assembly tests and four larger benchmarks, including median, qsort, multiply, and towers, as my data set. For each iteration, I choose one of the larger benchmarks as the verification data, and the rest as the training data. First, I just tried to use a simple polling to compute the average switching activity of certain type of opcode transitions. However, after iterations of training, only less than one-third of the signals can get a 90% correction, which is lower than the portion of the date-irrelevant signals.

Other more complicated models are also implemented. Instead of simplifying the instructions only be their opcode, I further characterized the instructions by their actual type and polling for all transitions of the more detailed types. Besides, I tried to use a combinations of both general and more detailed version to build the model. However, this obviously requires the program be more representative and takes longer to train the model. The model turned out to be less effective as the benchmark I use failed to satisfy the requirement just mentioned. And, this became the motivation for me to hard-code a better assembly test that is suitable for this model algorithm.

D. Modifying the Small Assembly Benchmark

Aiming to write a small assembly benchmark that is both concise and representative, I added all kinds of instruction transitions into the assembly file and tried to either optimize it as a feedback of simulation results in each iteration. Several types of modification includes compressing the code, incorporating more detailed types of instructions into the code, and adding more variations into the data. This has an obvious improvement in the accuracy of the signal switching frequency prediction. For the latest version, the code only takes about two hundred cycles, but about half of the signals has a 95% accuracy of the 4 larger benchmarks. For the rest of the signals, they are either dependent on the actual data being processed, or related to the state of the cache system. Then, by a small comparison of the accuracy, it can also help fast separate the data-dependent signals from the others. Hence, this small assembly benchmark has a reasonably good prediction on the switching frequency on the riscv-mini design.

Benchmark	Number of signals has a correction rate $>95\%$
Towers	197
Multiply	217
Median	188
Qsort	199

E. Trial of RNN

Recurrent Neural networks [13], RNN, is a neural network that nodes of different layers form a directed graph along a temporal sequence. The model is trained by iterations of forward passing and error back-propagating. It can use its internal states to store and process a sequence of inputs. As RNN can exploit the temporal relation of sequential inputs, my initial thoughts for this project is to use RNN to build a cycle-accurate model of switching activity. I implemented and modified the RNN code suggest by [14], and tried to build models unpon that.

However, the progress of build the RNN model was not as expected. I first tried to treat each instruction as a large integer and dump all the inputs into the algorithm. As this took too long but provided a not-reliable result, I attempted to decode the instructions into opcode types and even more detailed types. I also tried to tune different inner state dimensions, sequence length, and other parameters with different training and verification data. Within the limitation of the data I have and the tuning techniques I uses, the behaviour of the RNN model is too far away (at least 50% off) from the actual behaviour. It seems that a larger and more comprehensive data set is required for training the RNN model and more time are required for tuning the parameters. The lack of efficiency of this method impose a larger overhead on a new RTL design and it may even take longer than simply running a VCS/verilator simulation. As a result, I stopped further using the RNN model and shifted my focus on trying to find other more applicable models, which led to methods mentioned above.

V. Conclusion and future work

In summary, we can exploit the feature that the each type of instruction-type transition shares similar switching behaviour on the control signals in the long term, and write a concise and comprehensive assembly test to fast poll data and build a switching frequency model of the control and data-irrelevant signals. Based on this model, we can get a relatively reasonable estimation for any software/instructions sequence without running a hardware simulation. If we can know more information about the operation voltage and capacitance related to each signal, then we can get a roughly estimation of the switching power. Also, this may help circuit designers as a side guidance for determining the actual configuration of each sub-blocks of circuits to reach better power efficiency. For application specific hardware design, knowing the switching frequencies of different signals could be a head-boost for power optimization.

There are also some limitations of this approach. First, this only works for the data-independent signals. In the riscv-mini, this works quite well for only half of the signals, while the other half data-dependent signals might still need other models to estimate. The logical method suggested in Aladdin [3] would be more suitable to use data dependencies for analyzing switching activities. Also, this approach failed to achieve the initial goal of building a cycle-accurate model. This methods only works for prediction of switching frequency in the long run, and a better model (maybe an improved version of the RNN) should be explored to solve this.

Currently, this project is a case study on the riscv-mini with RV32I; in order to generalize and improve this method some future work are needed:

1) Including more instructions: This small benchmark only contains the integer instructions while there are more extension instructions of the riscv ISA. Multiply, floating points, and vectors extension may be added to create new benchmarks, though the conciseness would be a challenge if there would be too many types of instruction transitions.

2) Testing it with more complicated designs: As the benchmarks are more complicated, more complicated hardware designs, such as rocket chip, boom, are need to test these benchmarks. If this method would work well for these complex design, it might become useful to provide some side guidance in the design flow.

3) Dealing with data-dependent signals: The biggest regret of this project is failing to use recurrent neural networks to build a cycle accurate model of the switching activities. Though it is questionable for the efficiency of the RNN training as compared to the brute force simulation, this would be an interesting method and I will come back to as I learn more knowledge about the algorithms and techniques of the neural networks.

References

[1] D. Brooks, V. Tiwari, and M. Martonosi. 2000. “Wattch: a framework for architectural-level power analysis and optimizations,” SIGARCH Comput. Archit. News 28, 2 (May 2000), 83–94.

[2] S. Li, J. H. Ahn, R. Strong, J. Brockman, D. Tullsen, and N. Jouppi,"McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in MICRO, 2009.
[3] Y. S. Shao, B. Reagen, G.-Y. Wei, and D. Brooks, "Aladdin: A pre-RTL, power-performance accelerator simulator enabling large design space exploration of customized architectures," in ISCA, 2014.
[4] T. M. Austin Ph.D. and SimpleScalar, LLC, SimpleScalar tool set.
[5] N. Binkert, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, D. A. Wood, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, and T. Krishna, "The gem5 simulator," ACM SIGARCH Computer Architecture News, vol. 39, Aug 2011.
[6] D. Kim, V. Iyer, "Automatic Activity-based Power Modeling for Arbitrary RTL"
[7] D. Kim, A. Izraelevitz, C. Celio, H. Kim, B. Zimmer, Y. Lee, J. Bachrach, K. Asanovic, "Strober: Fast and Accurate Sample-Based Energy Simulation for Arbitrary RTL," 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), Seoul, 2016, pp. 128-139.
[8] D. Kim, J. Zhao, J. Bachrach, and K. Asanović. 2019. "Simmani: Runtime Power Modeling for Arbitrary RTL with Automatic Signal Selection, " in Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '52). Association for Computing Machinery, New York, NY, USA, 1050–1062.
[9] S. Bhanja, B. Ranganathan, "Switching Activity Estimation of VLSI Circuits Using Bayesian Networks", IEEE Transactions On Very Large Scale Integration(VLSI) Systems, Vol. 11, No. 4, August 2003
[10] T. Li and L. K. John, "Run-time modeling and estimation of operating system power consumption," in SIGMETRICS, 2003.
[11] T. F. T. Wenisch, R. R. E. R. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsafi, and J. C. J. Hoe, "SimFlex: Statistical Sampling of Computer System Simulation," IEEE Micro, vol. 26, pp. 18–31, Jul 2006.
[12] Donggyu Kim, Riscv-mini, https://github.com/ucb-bar/riscv-mini
[13] I. Goodfellow, Y. Bengio, A. Courville, "Deep Learning", MIT Press, 2016
[14] F. Shaikh, "Build a Recurrent Neural Network from Scratch in Python – An Essential Read for Data Scientists", https://www.analyticsvidhya.com/blog/2019/01/fundamentals-deep-learning-recurrent-neural-networks-scratch-python/