Fast and Accurate Activity Evaluation in Multipliers

Arnaud Tisserand

Fast and Accurate Activity Evaluation in Multipliers

Arnaud Tisserand
LIRMM, CNRS–Univ. Montpellier 2
161 rue Ada. F-34392 Montpellier, FRANCE
arnaud.tisserand@lirmm.fr

***Abstract*—This article reports the first results on fast and accurate power evaluation in arithmetic operators. The proposed method uses two steps: 1) accurate useful activity evaluation, 2) fast glitching activity estimation. The first step is based on circuit emulation using FPGA. Activity counters are inserted into the low-level description of the evaluated operator. The modified description is synthesized and downloaded into the FPGA. The operator activity behavior is emulated on the FPGA using large test vectors. The useful activity values accumulated in the FPGA are transferred to the computer. The second step uses the formal model we proposed in [1] for glitching activity estimation. The complete method is demonstrated on basic multipliers.**

I. Introduction

Modeling and reducing the power or energy consumption of arithmetic operators is an active research topic. This is still the case for basic operators such as adders and multipliers. In practice, there is a complex trade-off between:

- The *number system(s)* used to represent the data (radix, width, number coding, digit coding...);
- The *algorithm(s)* used to compute the mathematical operations (evaluation methods, speed/area trade-offs, basic blocks, fused operations...);
- The *characteristics of data* (domain, accuracy, activity, space/time correlations...);
- And some *circuit constraints* (specific cells in the standard library, logic style...).

Designers have to choose one operator among several solutions and many parameters [2]: various algorithms, representations of numbers, architectures, pipelining.... See [3] for a short survey on low-power arithmetic operators. While accurate power models exist for adders [4], [5], it is difficult to choose a multiplier, and its parameters, with respect to its practical power consumption [6], [7], [8]. Recent works such as [9] show that power may be significantly reduced using different multiplication algorithms.

One of our long term projects is to study the arithmetic impact on the power consumption of dedicated applications such as digital signal processing (DSP), multimedia, cryptography or specific embedded high-performance computing (HPC). For that purpose, one needs to compare the power consumption of different internal representations of numbers, different algorithms for the internal computations and different architectures and parameters. Then fast and accurate extraction of features such as speed, circuit area and power or energy consumption of arithmetic operators is required.

Standard solutions for these comparisons include circuit measurement, theoretical analysis, circuit simulation or estimation. Circuit measurements lead to very accurate power results but at very high cost and delay due to circuit fabrication. Theoretical analysis only provides bounds and distribution estimations for very basic operators and simple data characteristics (e.g. [4] for adders). So those methods are not often used in practice. Circuit simulation using electrical simulators is widely used. It can lead to accurate results but it requires very long simulation times. Some power estimation models and/or tools are available (e.g. [10]). They are fast and very accurate at the system or application levels but their accuracy is very limited at the operator level.

Below we present a method based on circuit emulation on FPGA for useful activity evaluation. Glitching activity is estimated using a formal model we proposed in [1]. Some background information and a high-level description of the proposed method are presented in Section II. The accurate useful activity evaluation using circuit emulation on FPGA is presented in Section III. The formal model for the glitching activity estimation we proposed in [1] is recalled in Section IV. The complete method is demonstrated, validated and compared to standard solutions in Section V.

II. Background and High-Level Description of the Proposed Method

The power consumption of digital circuits has two kinds of contribution: the *static* power consumption due to leakage and the *dynamic* power consumption due to activity (i.e. signal transitions). See [11] for a good reference.

The static power of arithmetic operators can be accurately approximated by a function of the circuit area required for the implementation (and several technological parameters). The area of an operator is easily extracted from the layout. Then the proposed method does not deal with static power.

The activity in digital circuits (and arithmetic operators) is caused by two kinds of signal transitions. The *useful activity* due to the input transitions that produce the internal transitions required to perform the computation (i.e. complete and stable transitions). The *glitching activity* (also called *redundant activity*) is caused by different delays from several inputs to the same output or circuit defects. Specific circuit styles or careful placement and routing can significantly reduce the glitching contribution. But important energy savings can also be achieved on the glitching activity by using specific number systems or optimized arithmetic algorithms [3].

The method proposed below evaluates the total activity in arithmetic operators. It can be used to evaluate their dynamic

power. It is decomposed into two steps:

1) Useful activity evaluation (detailed in Section III);
2) Glitching activity estimation (detailed in Section IV).

The useful activity is accurately evaluated using emulation on FPGA and large and relevant test vectors. The results of the first step are used as parameters for the next step. In the second step, the glitching activity is estimated using the formal model we proposed in [1].

The evaluation flow is presented on Figure 1. The low-level description of the evaluated operator is provided by the input netlist. Usually it comes from in-house arithmetic operator generators. This netlist is used for both the useful activity evaluation (right part of Figure 1 detailed in Section III) and the glitching activity estimation (left and bottom part of Figure 1 detailed in Section IV).

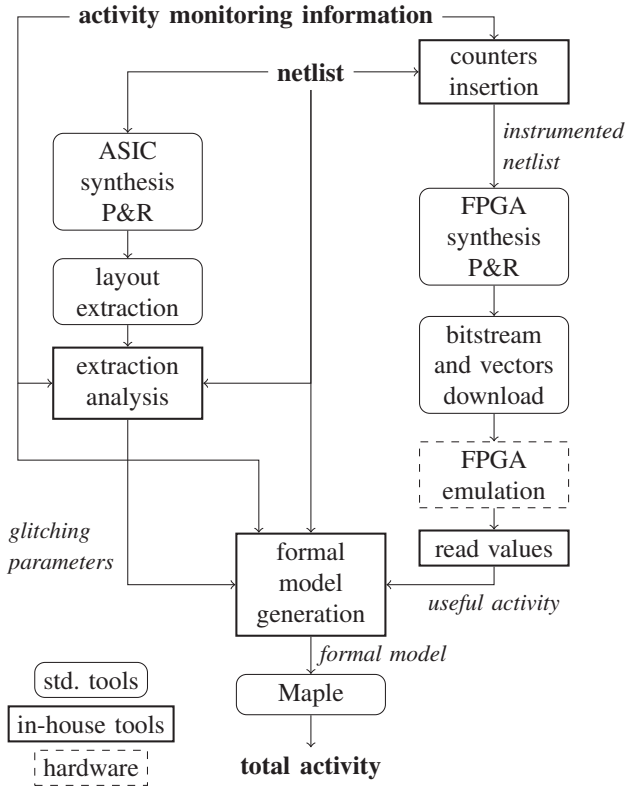

Fig. 1. Evaluation flow of the proposed method

III. Useful Activity Evaluation using Circuit Emulation on FPGA

In order to accurately evaluate the useful activity in arithmetic operators, we propose to count the number of useful transitions using FPGA emulation. At each location (i.e. net at a gate output) where activity monitoring is required, an activity counter is inserted. The activity counter is a simple block composed of a D flip-flop, a XOR gate and a k-bit counter as depicted on Figure 2. The D flip-flop and the XOR gate produce a 1 activity signal each time there is a useful transition on the monitored net. The counter counts the total number of such transitions. The counter size parameter k depends on the test vector length. Some additional control logic is required for the counter clock management, to reset the counters and to read the counters values at the end of the emulation. We use a simple serial scan chain method to read the value stored into the counter. This control block can be shared among several activity counters.

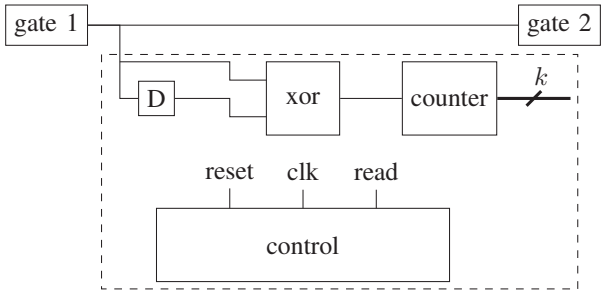

Fig. 2. Activity counter (dashed block) inserted at the output of gate 1

Activity monitoring locations are provided by the user[1]. A C++ program modifies the original netlist into an "instrumented" netlist by inserting activity counters on each monitored net (see Figure 1). The instrumented netlist is synthesized, placed and routed using standard FPGA tools. The clock frequency of the instrumented operator is lower than original one (i.e. original and activity signals must be stable). Then only useful or theoretical transitions are counted. The generated bitstream and test vectors are downloaded into the FPGA. During the emulation sub-step on the FPGA, test vectors are applied to the evaluated operator and activity values are accumulated. The final sub-step reads the activity values in FPGA from the host.

In practice, it is often useful to monitor the activity of several nets. If just the total number of transitions is required, we use a shared version as depicted on Figure 3. The number of ones in the activity signals for all individual nets is converted into a binary value using a full adder (FA) tree (in case of Figure 3, at most 3 ones are converted into a 2-bit value using one FA cell). This global activity value is accumulated into a counter. The size of the counter depends now on the test vector length and the number of monitored nets.

The test vectors can be very large since the operating frequency is the instrumented operator clock in the FPGA. This frequency is significantly higher than the "frequency" of a simulated operator (even with a good VHDL simulator). In practice, there is difference of several orders of magnitude. The test vectors correspond to real-world values downloaded from the host computer. The fact these vectors are composed of relevant values leads to a very accurate activity evaluation and not only bounds or rough estimations. The accuracy of the useful activity evaluation only depends on the "quality" of the test vectors. The proposed method has been validated using

[1]This may be hard and long in practice. We plan to work on a model and/or a tool to simplify this sub-step.

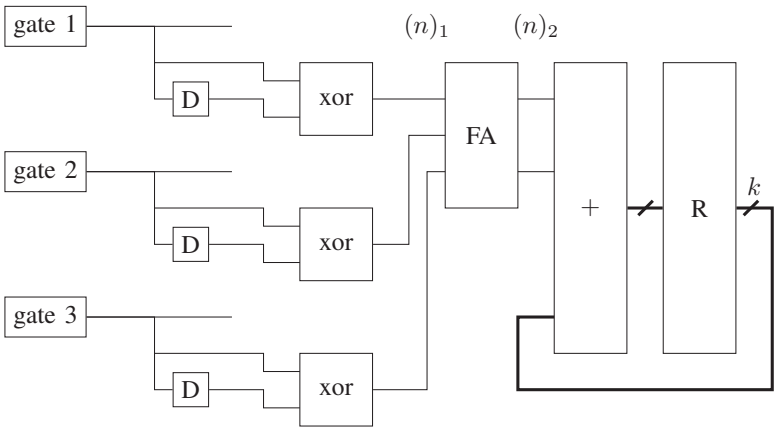

Fig. 3. Shared activity counter for multiple monitored nets (FA cell converts the radix-1 representation $(n)_1$ of the activity value into a binary value $(n)_2$ for accumulation)

comparisons to VHDL simulations with activity monitoring (counting the transitions in the signal history).

Table I reports FPGA implementation results of original (without counters) and instrumented operators (with counters). Two instrumented versions have been implemented: one with individual counters for all nets and one with a shared activity counter for all nets. The evaluated operators are $(n \times n)$-bit multipliers based on Booth-2 recoding, Wallace reduction tree and ripple carry adder for carries assimilation. The target FPGA is the Xilinx Virtex II-Pro family. Implementation has been done using ISE 8.1 tools with standard efforts and no pipeline. All results are normalized with respect to the non-instrumented version (without counters).

version	n	k	area	delay
no counter	32	–	1.0	1.0
indiv. cnt.	32	10	14.2	0.5
indiv. cnt.	32	20	25.6	0.3
shared cnt.	32	20	2.5	0.8
shared cnt.	32	40	2.5	0.8
no counter	64	–	1.0	1.0
shared cnt.	64	20	2.4	0.7
shared cnt.	64	40	2.4	0.7

TABLE I
FPGA IMPLEMENTATION RESULTS FOR THE USEFUL ACTIVITY EVALUATION OF $(n \times n)$-BIT STANDARD MULTIPLIERS

These implementation results show that in the case of individual activity counters the circuit area is very huge. Inserting an activity counter at each monitored net leads to circuits mostly composed of activity counters. But as the arithmetic operators are very small compared to the huge number of programmable gates available in the current largest FPGAs, it is still possible to monitor all nets of an arithmetic operator. In the case of a shared counter for multiple monitored nets, the area bloat is about a factor 2.5 while the circuit can still operate at a frequency close to 70–80% of the initial clock rate (e.g. 70% of about 30 MHz for the multipliers and FPGA considered in these experiments).

The useful activity evaluation time has been reported for a few experimentations. It has been compared to the corresponding VHDL simulation time (using Modelsim 6.1 fast simulator). Speedup factors up to 160 has been reported for large operators (e.g. 32-bit operands) and long test vectors (e.g. with 10^6 values). For small operators (e.g. 8-bit multipliers) and small test vectors (e.g. with 10^2 values), the proposed method may lead up to a 10 times slowdown factor.

The analysis of the evaluation and simulation duration is illustrated on Figure 4. The proposed evaluation method uses a sequence of different sub-steps (and tools): counters insertion, synthesis and place and route, bitstream and vectors download, emulation on the FPGA and finally the activity counters reading. The standard simulation method is simpler: simulator start and simulation. The small vertical lines in the emulation and simulation areas show the duration of each individual test (i.e. the computation of one value on the evaluated operator). It is clear that the proposed method has a large overhead due to FPGA implementation (synthesis and P&R) and download. But it also has the fastest evaluation rate. As soon as the operators and/or the vectors are large enough, the proposed method leads to very small evaluation times compared to standard simulation.

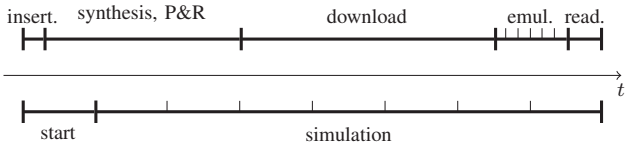

Fig. 4. Time decomposition of the proposed evaluation and standard simulation methods

The current bottleneck is the FPGA implementation (synthesis and P&R) and download sub-steps. It is difficult to reduce the implementation time. We plan to try partial reconfiguration to speedup this sub-step. The download of the bitstream and test vectors is long due to slow rate programming and

communication cables used between the host and the FPGA board. In the future we plan to work using a card with fast programming (for bitstream download) and communications (for test vectors download and counters reading) such as PCI or Hyper-Transport links.

IV. Glitching Activity Estimation using Formal Model

We use the formal model we proposed in [1] for the glitching activity estimation. This method uses formal expressions (in practice multivariate polynomials) to model all required characteristics of the monitored gates (delay, activity). There is one polynomial for each gate (or small set of gates) and for each kind of contribution (delay, activity). The model parameters are input activities, gate delays and some gate characteristics. The parameters related to actual values are determined by a characterization step based on small electrical simulations. Those parameters are part of a small database that just depends on the technology and gate library. So the characterization is only done once there is a new gate library.

As an example, the delay behavior of a FA gate (logic functions with (x, y, z) inputs and (c, s) outputs such that $2c + s = x + y + z$) is modeled by the polynomials:

- sum output: $s_0 + s_1\delta(t_x, t_y, t_z)$
- carry output: $r_0 + r_1\delta(t_x, t_y, t_z)$

where $\delta = \max(t_x, t_y, t_z) - \min(t_x, t_y, t_z)$ and the coefficients are determined during the characterization step. For more details refer to [1]. Some of the variables of the model are the activity values obtained from the useful activity evaluation method presented in Section III. The accuracy of the useful activity evaluation step is a key point of the accuracy of the complete method.

The formal model allows us to use formal probability distribution for the monitored characteristics and/or the variables. The corresponding polynomials are used as part of the input variables of the next gate. The complexity of the model increases with the operator depth. We use Maple to evaluate the actual values of the complete model. Using such a computer algebra system, the evaluation of thousands of high-degree polynomials is very fast (a very few minutes for the largest evaluated operators).

Figure 5 illustrates the model results (after the formal evaluation) for a very simple example. Wallace and Dadda trees are considered for 5-bit partial products reduction array. The model uses degree-0 polynomial and all coefficients equal to 1. The numerical values of Figure 5 correspond to the glitching activity (number of transitions) in the reduction trees with arbitrary units.

The model also includes a basic support for other constraints such as unequal raise/fall times and routing capacitances. The formal model method has been validated using comparisons to Spice electrical simulations. In practice, about 10–20% accuracy is obtained compared to the Spice simulations [1]. The estimation accuracy depends a lot on the accuracy of the analysis performed during the characterization process. Our main problem is the characterization step. Currently it is mostly done by hand. We plan to automate this process in the future.

V. Validation and Comparison

Activity and power evaluations of some multipliers have been done. The implemented multipliers are $(n \times n)$-bit multipliers based on Booth-2 recoding, Wallace or Dadda reduction trees and final 3 blocks addition for the carries assimilation. Various operand sizes n have been implemented: 8, 16, 24 and 32 bits. ASIC implementation has been done for a 0.35μm AMS CMOS technology target using Synopsys and Cadence tools. The obtained results have been compared to Spice extensive simulations. Some of them are reported on Figure 6. The obtained results show about 20% accuracy compared to Spice simulations while there is a speedup factor about 10 up to 30 for the estimation time of large operators.

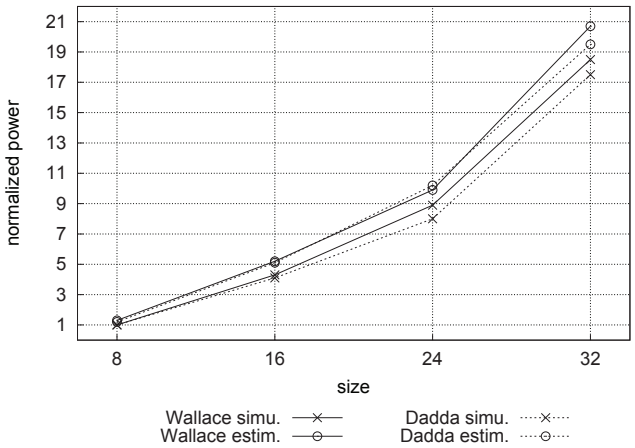

Fig. 6. Power estimation results compared to Spice simulations

The proposed method can be easily used to evaluate the internal power distribution into the various stages or parts of arithmetic operators. One can use a shared activity counter for each monitored stage or part. For instance Table II reports the power distribution of a 32-bit multiplier based Booth recoding, Dadda reduction tree, final 3 blocks addition for the 0.35μm AMS CMOS technology. That power distribution is close to the one obtained using long electrical simulations.

step	delay	power
partial products	15%	16%
reduction	54%	67%
assimilation	31%	17%

TABLE II
Power decomposition for basic multipliers

VI. Conclusion

In this paper the first results for fast and accurate power evaluation of arithmetic operators are reported. The proposed

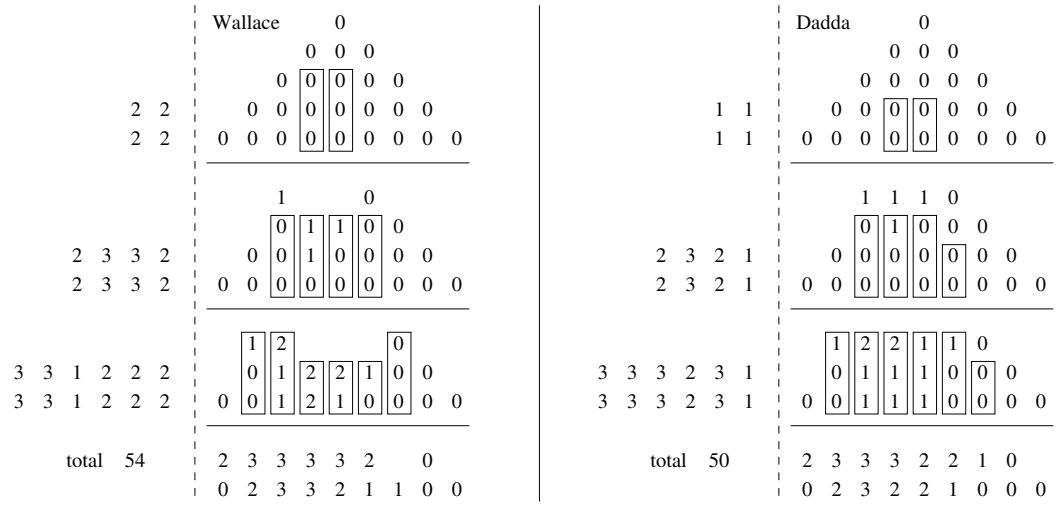

Fig. 5. Glitching activity estimation for 5-bit partial products trees using degree-0 polynomials and all coefficients equal to 1

method uses useful activity evaluation based on circuit emulation on FPGA and glitching activity estimation using a formal model from [1]. The proposed method has been implemented using in-house tools and calls to standard CAD tools. It has been validated on simple arithmetic operators (adders and basic multipliers).

The proposed activity evaluation method leads to accurate power estimations (less than 20% difference compared to Spice simulations) and very fast estimation times (about 100 speedup factor).

In the future, we plan to:

- Improve the useful activity evaluation using faster communications between the host and the FPGA;
- Improve the formal model for glitching activity using more accurate models and characterization;
- Validate using other operators such as divisors, elementary functions evaluators (e.g. [12]) and small blocks for cryptography applications.

We also plan to use the fast power evaluation in order to help the design of low-power arithmetic operators in our synthesis tools and for arithmetic operators dedicated to cryptographic applications with low power signature.

ACKNOWLEDGMENT

This work has been partially supported by the grant ANR-AF-2006-ROMA "*Reconfigurable Operators for Multimedia Applications*".

REFERENCES

[1] A. Tisserand, "Estimation rapide de l'activité parasite pour l'optimisation des arbres de réduction de multiplieurs," in *6ième journées d'études Faible Tension Faible Consommation (FTFC)*, Paris, France, May 2007, pp. 127–130.

[2] M. D. Ercegovac and T. Lang, *Digital Arithmetic*. Morgan Kaufmann, 2003.

[3] A. Tisserand, "Low-power arithmetic operators," in *Low Power Electronics Design*, C. Piguet, Ed. CRC Press, Nov. 2004, ch. 9.

[4] A. Guyot and S. Abou-Samra, "Modeling power consumption in arithmetic operators," *Microelectronic Engineering*, vol. 39, pp. 245–253, 1997.

[5] C. Nagendra, M. J. Irwin, and R. M. Owens, "Area–time–power tradeoffs in parallel adders," *IEEE Transactions on Circuits and Systems–II: Analog and Digital Signal Processing*, vol. 43, no. 10, pp. 689–702, Oct. 1996.

[6] M. J. Flynn and S. F. Oberman, *Advanced Computer Arithmetic Design*. Wiley-Interscience, 2001.

[7] I. S. Abu-Khater, A. Bellaouar, and M. I. Elmasry, "Circuit techniques for CMOS low-power high-performance multipliers," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 10, pp. 1535–1546, Oct. 1996.

[8] P. F. Stelling, C. U. Martel, V. G. Oklobdzija, and R. Ravi, "Optimal circuits for parallel multipliers," *IEEE Transactions on Computers*, vol. 47, no. 3, pp. 273–285, Mar. 1998.

[9] Z. Huang and M. D. Ercegovac, "High-performance low-power left-to-right array multiplier design," *IEEE Transactions on Computers*, vol. 54, no. 3, pp. 272–283, Mar. 2005.

[10] E. Senn, J. Laurent, N. Julien, and E. Martin, "SoftExplorer: Estimating and optimizing the power and energy consumption of a C program for DSP applications," *EURASIP Journal on Applied Signal Processing*, pp. 2641–2654, Jan. 2005.

[11] N. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, 3rd ed. Addison Wesley, 2005.

[12] A. Tisserand, "Automatic generation of low-power circuits for the evaluation of polynomials," in *Proc. 40th Asilomar Conference on Signals, Systems and Computers*. Pacific Grove, California, U.S.A.: IEEE, Oct. 2006, pp. 2053–2057.