Power Optimized Partial Product Reduction Interconnect Ordering in Parallel Multipliers

Saeeid Tahmasbi Oskuii
Dept. of Electronics and Telecom.
Norwegian Univ. of Science and Tech.
Trondheim NO-7491, Norway
Email: saeeid@iet.ntnu.no

Per Gunnar Kjeldsberg
Dept. of Electronics and Telecom.
Norwegian Univ. of Science and Tech.
Trondheim NO-7491, Norway
Email: pgk@iet.ntnu.no

Oscar Gustafsson
Dept. of Electrical Engineering
Linköping University
SE-581 83 Linköping, Sweden
Email: oscarg@isy.liu.se

***Abstract*— When designing the reduction tree of a parallel multiplier, we can exploit a large intrinsic freedom for the interconnection order of partial products. The transition activities vary significantly for different internal partial products. In this work we propose a method for generation of power-efficient parallel multipliers in such a way that its partial products are connected to minimize activity. The reduction tree is designed progressively. A Simulated Annealing optimizer uses power cost numbers from a specially implemented probabilistic gate-level power estimator and selects a power-efficient solution for each stage of the reduction tree. VHDL simulation using ModelSim shows a significant reduction in the overall number of transitions. This reduction ranges from 15% up to 32% compared to randomly generated reduction trees and is achieved without any noticeable area or performance overhead.**

I. Introduction

Multipliers are essential blocks in digital signal processing systems. In many situations, multipliers lie directly in the critical path and the processing speed is ultimately limited by the multiplication speed. Meanwhile, the increasing need for portable multimedia systems necessitates low-power operators in order to maintain reliability and provide longer hours of operation. Multipliers are among the main contributors to the total power, in particular because performance requirements often necessitates use of high-speed parallel multipliers. Power efficient design of such parallel multipliers are hence very important. A parallel multiplier performs the multiplication in three computation steps:

1) Form all partial products in parallel
2) Reduce these partial products through series of reduction stages
3) Sum the final two output vectors from step 2 using a carry propagate adder to compute the final output vector.

The speed and power consumption of the parallel multipliers have always been critical issues and, therefore, the subject of many research projects; these results are surveyed in [1]–[4]. Speedup is mainly achieved by reducing the number of partial products generated in the first step or using faster schemes to accumulate partial products. Baugh-Wooley [5] and Modified Booth encoding [6], [7] are among the most dominant and efficient approaches for the first step.

After the generation of partial products the objective is to rapidly reduce these partial products to the final two vectors. Several well-known schemes have been developed in the past. A carry-save tree constructed from full-adders and half-adders as an efficient and fast reduction scheme was introduced by Wallace [8]. In [9] Dadda, as a generalization of the Wallace scheme, employed the notion of digital counters and proposed a minimization method for the number of counters in the reduction tree. Later on Bickerstaff et al. in [10] refined the Dadda scheme further by inserting a number of half-adders in the structure of the reduction tree. This modified Dadda scheme leads to smaller output vectors from reduction step without any speed overhead in the reduction step. The smaller input vector sizes of the final carry propagate adder result in faster overall computation time for the multiplier. In addition to the Wallace and Dadda trees which utilize full-adders and half-adders, [11], [12] suggest using higher orders of compressors in the reduction tree. In this work we perform our experiments using the modified Dadda scheme described in [10]. However, our proposed technique is, with small adjustments, also applicable to trees based on Wallace and earlier Dadda schemes and also to trees based on 4:2 or higher order compressors.

The modified Dadda scheme constructs the reduction step of the multiplier as follows: For each stage, the number of full-adders used in each column is $\lfloor \frac{b_i}{3} \rfloor$, where b_i is the number of bits in column i. This provides the maximum reduction in the number of bits entering the next stage. Half-adders are used only (a) when required to reduce the number of bits in a column to the number of bits specified by the Dadda series [9], or (b) to reduce the rightmost column containing exactly two bits.

Figure 1 illustrates the reduction tree of a 12×12 bits unsigned multiplier. Each dot corresponds to a single-bit partial product. Vertical boxes containing three bits and two bits illustrate full-adders and half-adders respectively. The SUM and CARRY outputs for each adder at one stage are used as inputs at the next stage (or sent directly to the final propagation adder). Each column represents partial products of a certain order of magnitude. A SUM output at one stage will place a dot in the same column at the next stage. A CARRY output at one stage will place a dot in the column to its left i.e. one order of magnitude higher.

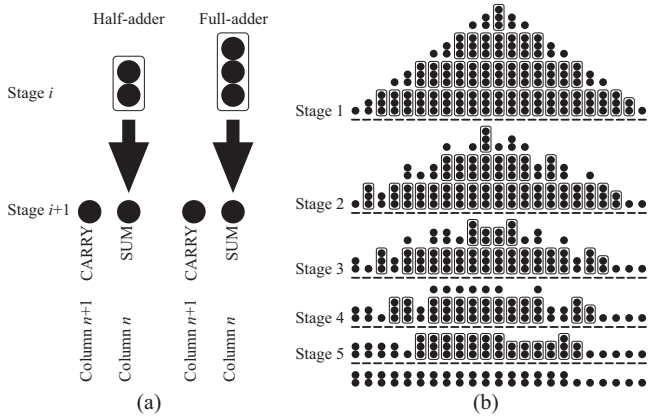


Fig. 1. (a) Simplified representation of full-adders and half-adders. (b) Modified Dadda reduction scheme for 12×12 bits unsigned multiplier

As shown in Figure 1, reduction schemes, for example the modified Dadda scheme, do not specify the interconnection order of the partial products; i.e. there is large freedom in choosing the partial products to be connected to the full-adders and half-adders. As long as the partial products belong to a unique column, they can be interchanged and used as inputs to any of the half-adders and full-adders without any change in functionality. Related previous work includes [13]–[17]. We will come back to this in the following sections. Our objective in this work is to choose a permutation of inputs that results in a minimized dynamic power consumption for the multiplier. The main contribution of this paper is a technique for progressive design of the reduction tree. We have implemented and employ a very fast probabilistic gate-level power estimator that also model glitch generation accurately. This will be discussed further in Section II.

This paper is organized as follows. Section II describes the power consumption model for the parallel multipliers and our power estimation technique. In Section III we elaborate reduction tree generation techniques and previous work. Our progressive design method is described in Section III-B. The usefulness of the optimization technique is demonstrated in Section IV through a number of experiments. The run-time and complexity issues are addressed in Section V. Finally, we give our conclusions in Section VI.

II. Power Consumption in Parallel multipliers

The computation process of a multiplier manipulates the input data bits to generate the partial products and afterwards compresses the partial products to generate the final outputs. This process requires a large number of transitions which will dissipate dynamic power. The average dynamic power, which is arguably the dominant component of the total power consumption of a CMOS circuit, is given by [18]:

$$P_{av}^{dynamic} = \frac{1}{2} V_{DD}^2 \sum_{i=1}^{N} C_i D_i \tag{1}$$

where V_{DD} is the supply voltage, N is the total number of nodes, C_i is the total load capacitance at node i and D_i is the transition density at node i defined as [19]:

$$D_i = \lim_{T \to \infty} \frac{n_i(T)}{T} \tag{2}$$

where $n_i(T)$ is the number of transitions in a time interval of length T. Assuming synchronous primary inputs for the multiplier operands, the input transitions appear at the same time and the resulting transitions propagate from inputs to the final output. With these assumptions the transition density D_i in a pure combinational circuit can be rewritten as:

$$D_i = f_{clk} \overline{n_i} \tag{3}$$

where $\overline{n_i}$ is the average number of transitions at node i in one clock cycle and f_{clk} is the clock frequency. Deterministic delay assignments for the logic gates lead to a finite set of possible transition times for each node. Given the transition probabilities at primary inputs of a combinational circuit, a set S_i of transition times with their probabilities (p_j) at each node i can be constructed. D_i can then be specified as:

$$D_i = f_{clk} \sum_{j \in S_i} p_j \tag{4}$$

In addition to the brute force approach of computing D_i by simulation, a number of probabilistic methods have been proposed in order to estimate the transition density D_i for a given combinational logic circuit [20]–[23]. We utilize the Simple Waveform Set method described in [23] as the underlying power estimation kernel in our optimization algorithm.

A. Simple Waveform Set

The Simple Waveform Set (SWS) method performs power estimation by propagating all possible waveforms through the combinational circuit as simple waveforms. Simple waveforms are waveforms with at most one transition. Therefore the non-simple waveforms generated in the circuit as glitches are decomposed into simple waveforms. This decomposition lowers the complexity of computations drastically without introducing large errors. The estimation commences by assigning SWSs to input nodes. Four waveforms are introduced for each input node:

$$\text{SWS}_i = \{W_{11}, W_{10}, W_{01}, W_{00}\} \tag{5}$$

where W_{11}, W_{00}, W_{10} and W_{01} are holding one, holding zero, one-zero transition and zero-one transition respectively. These transitions are assumed to occur at time 0. Under the independence assumption for input nodes before and after time 0, occurrence probabilities of such waveforms can be assigned from the static probabilities of each input node. Let p be one-probability at the corresponding input node. Then $p(W_{11}) = p^2$, $p(W_{00}) = (1-p)^2$ and $p(W_{10}) = p(W_{01}) = p(1-p)$. Once the SWSs are assigned to the input nodes, waveform propagation can begin. The SWS at the output of a logic gate can be computed from its input SWSs. As discussed earlier SWS requires a deterministic delay model for the gates. Although any arbitrary delay model is applicable, we utilize a fanout delay model for simplicity. The delay of the gates are assumed to be proportional to their

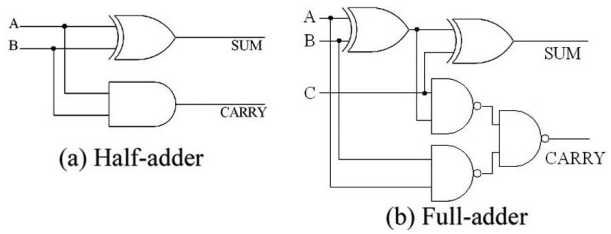


Fig. 2. Typical gate-level implementation of (a) half-adder and (b) full-adder

fanout number. In general the problem of interdependencies between waveform of different nodes arises in the presence of reconvergent fanouts in combinational logic. The SWS algorithm incorporates correlation coefficients and the notion of macrocorrelation to overcome this problem [24]. SWS also accounts for glitch filtering due to the inertial delay of logic gates by assigning mask vectors to each waveform [23]. A logic gate will remove a glitch if it is shorter than the inertial delay of the gate.

A useful observation can be inferred from the structure of the multiplier. Every partial product is connected to exactly one full-adder or half-adder if it is not connected to the final carry propagate adder. As shown in Figure 2, the inputs of full-adders and half-adders have fanout to one XOR gate and one NAND/AND gate. The structure of the reduction tree hence does not accommodate high-fanout nodes, and the maximum fanout is two. This is an important consideration that means that glitch filtering happens rarely in the structure of a reduction tree. Glitches, if created, will propagate to the output vectors. This, in fact, reduces the computational complexity of power estimation when using the SWS algorithm because the estimator does not need to track the glitch filtering in the successor nodes.

III. Constructing the Reduction Tree

Manipulations of the reduction tree can target speed or power consumption. Although the functionality is kept unchanged through the alterations of the reduction tree, the performance and power consumption will be different.

A. Previous Techniques for Reduction Tree Design

As Figure 2(b) illustrates, the delay from an input to an output in a full-adder is not the same for all inputs and outputs; e.g., input C is a faster input than A and B, and output CARRY is a fast output compared to SUM. Oklobdzija et al. in [15] proposed a method for reduction tree design that minimizes the critical path delay.

When the optimization target is decreasing power consumption, we need to take into account the one-probabilities of the circuit nodes as well as the large number of spurious transitions in the reduction tree. A reduction tree, that is not pipelined, is a pure combinational circuit with several layers of logic. Therefore the spurious transitions amount to a large part of the power consumption and cannot be neglected. Yu et al. in [13] introduced a number of design guidelines for low-power multipliers. Among them are assigning a shorter logic depth to a partial product with high transition activity. Another consideration from [13] is sorting transition probabilities, quantifiers calculated from one-probabilities of partial products, and assigning the inputs of counters in the order that is achieved from sorting. This sorting of transition probabilities can lower the energy dissipation but the improvements are not significant because the spurious transitions are completely ignored. As Figure 2(b) shows, the path from input C to output SUM contains only one logic gate while the path from input A or B contains two logic gates. Therefore, the pure sorting approach does not necessarily result in a good solution. By incorporating a probabilistic power estimator in the optimization algorithm, the method in [17] increased power saving further. The proposed algorithm in [17] starts with a complete multiplier where the connections are initialized to a random permutation. Then, for each stage and each column in the reduction tree, a number of useful permutations are specified and tested one at a time. For each permutation, power consumption of the complete multiplier is estimated. If it results in an overall power reduction the permutation is kept, otherwise it is discarded. The number of possible permutations increases factorially for larger multiplier sizes; Therefore the number of useful permutations is kept small using a number of assumptions. As the power estimator is being run repeatedly in the core of several loops, it must be very fast. The power estimator in [17] is a fast estimator that is built on a 0-algorithm for signal probability computation [25], ignoring interdependencies in the circuit. The power consumption is estimated using interpolation of data acquired from Synopsys Power Compiler for basic logic cells.

```
optimized_multiplier(static probabilities of input bits)
initialize circuit TheMultiplier;
PPG=generate_initial_partial_products(suitable_algorithm);
TheMultiplier.append(PPG);
compute_SWS(TheMultiplier);
for current_stage= first_stage to last_stages{
  for current_col= LSB to MSB{
    [FAs,HAs]=compute_required_adders;
    current_perm=random_perm;
    minP=∞;
    for current_iteration= 1 to no_of_iterations{
      old_perm=current_perm;
      swap two random positions in current_perm;
      P=estimate_power(FAs,HAs, current_perm);
      if (P < minP)
        minP=P;
      else
        if uniform random r ∈ [0,1) < e^((minP−P)/α)
          minP=P;
        else
          current_perm=old_perm;
    }
    TheMultiplier.append(FAs, HAs,current_perm);
    compute_SWS(TheMultiplier);
  }
}
```

Fig. 3. The proposed optimization algorithm

B. Progressive Reduction Tree Design

Our proposed method for reduction tree design is summarized in Figure 3. The algorithm starts with initialization of the circuit to primary partial products generated by a suitable algorithm discussed in Section I. The function compute_SWS(`TheMultiplier`) computes the Simple Waveform Sets for the nodes of the multiplier circuit for which SWSs are not computed earlier. Note that each time a circuit portion is appended to `TheMultiplier`, this function is executed, computing SWSs for the new portion using the SWSs computed earlier. After the partial products are generated, the multiplier structure is progressively designed. For each column of each stage in the reduction tree, a Simulated Annealing optimization [26] is performed. A randomly chosen permutations is the start configuration. This configuration is altered in each iteration. A new power estimation is computed for the perturbed configuration. If the move decreases the value of the estimated power, the move is accepted and the new permutation is retained. If Simulated Annealing gets stuck in a local optima then the Boltzmann factor is calculated, and a random number, uniformly distributed in the interval [0,1), is chosen. If the random number is less than the calculated Boltzmann factor, then the new permutation is retained, otherwise, the move is discarded and the configuration before this move is used for the next step. The power estimator in this loop computes the transition densities (Eq. 4) exclusively for the current full-adder and half-adder stage, i.e. the stage that is being designed, using the SWSs computed earlier.

There is a substantial difference in our proposed algorithm and the algorithm proposed in [17]. In this work we use a more accurate and more complex power estimator (SWS) but only engage the estimator for one layer of full-adders or half-adders at any given time. However [17] uses a simpler power estimator and runs the power estimations for the complete multiplier. Compared to the method in [17], the power estimation is faster even though it is more accurate and more complex. The reason is that the estimation is performed on only a small part of the multiplier rather than the complete multiplier. This allows running the optimization algorithm for larger number of permutations. Running the power estimator for the complete multiplier as in [17], even for the parts that are not optimized yet, means that the effects are considered for all successive nodes. But because parts of the multiplier are not optimized yet the actual effects will be different when those parts are optimized as well. On the other hand, running the power estimation exclusively for the current full-adders/half-adder stage means that we do not consider the effects on the partial products that are not the direct outputs of the immediate successive full-adders/half-adders.

The algorithm `optimized_multiplier` shown in Figure 3 constructs a multiplier that is optimized for low power consumption. When reporting experimental results in the next section, we denote multipliers generated using `optimized_multiplier` by index OPT (e.g., $M_{OPT}^{16\times16}$). An algorithm called `worst_case_multiplier` generates a multiplier that is connected in a worst-case fashion to consume maximum energy. `worst_case_multiplier` is achieved by altering the estimate_power function to multiply -1 to the estimated power value. The multipliers generated using `worst_case_multiplier` are indexed by WC (e.g. $M_{WC}^{16\times16}$).

Table I- Average number of transitions for different multipliers (the input static probabilities Ω_i^{16} are applied to both operands)

i	$M_{OPT}(\Omega_i^{16},\Omega_i^{16})$	$M_{WC}(\Omega_i^{16},\Omega_i^{16})$	M_{Random}
16	91.48	111.98	106.05
15	79.79	98.96	93.56
14	65.27	87.85	81.60
13	55.03	76.55	69.33
12	45.09	65.21	58.09
11	36.71	52.91	47.56
10	29.16	42.83	37.46
9	21.70	33.34	28.72

IV. Experimental Results

Our proposed method shown in Figure 3, the SWS power estimator, and a gate-level VHDL code generator, are all implemented in C++. Inputs to the optimization program are the static probabilities of the primary inputs. After completion of the optimization algorithm, the equivalent VHDL code for the multiplier is generated. To evaluate the quality of our optimization algorithm, this VHDL code is run through a ModelSim simulation. Input stimuli with the same static probabilities are used, and the average number of transitions for all nodes is collected. We report the sum of average node transitions for all nodes in the reduction tree. As node load capacitance for the reduction tree does not have large variations, average number of transitions are strongly related to the power consumption.

In the first part of our experiments, we consider a multiplier in a system where the word-length is dynamically varying. This can for instance appear in systems where the quality of service requirements change with time. We denote the input probabilities of an n-bit primary input operand by Ω_i^n, where i bits from the MSB side have a one-probability of 0.5 and $n-i$ bits from the LSB side have a one-probability of zero. For example, if the static probabilities of input word $A_{0..5}$ is represented by Ω_4^6, it means that input bits A_0 (LSB) and A_1 have 0 one-probabilities and input bits A_2, A_3, A_4 and A_5 have 0.5 one-probabilities. An $n \times m$ bits multiplier resulting from algorithm `optimized_multiplier` with input static probabilities Ω_i^n and Ω_j^m, is represented by $M_{OPT}(\Omega_i^n,\Omega_j^m)$. Similarly an $n \times m$ bits multiplier resulting from algorithm `worst_case_multiplier` with input static probabilities Ω_i^n and Ω_j^m, is represented by $M_{WC}(\Omega_i^n,\Omega_j^m)$. We have constructed optimized (and worst-case) 16×16 bits multipliers that are optimized for input probability vectors Ω_i^{16} ($i = 9..16$). In Table I the average number of transitions is given for $M_{OPT}(\Omega_i^{16},\Omega_i^{16})$ and $M_{WC}(\Omega_i^{16},\Omega_i^{16})$ when static input probability vector Ω_i^{16} is applied to both operands. The rightmost column in Table I is dedicated to M_{Random}. The

Table II- Average number of transitions for different multipliers

Input static probabilities applied to both operands

$M_{OPT}(\Omega_i^{16}, \Omega_i^{16})$ i	Ω_{16}^{16}	Ω_{15}^{16}	Ω_{14}^{16}	Ω_{13}^{16}	Ω_{12}^{16}	Ω_{11}^{16}	Ω_{10}^{16}	Ω_{9}^{16}
16	91.5	80.0	67.4	57.7	48.6	38.8	31.0	23.7
15	92.9	79.8	68.0	58.2	47.8	39.1	30.5	23.5
14	95.5	82.1	65.3	55.8	47.7	38.1	30.3	23.0
13	97.4	84.5	71.4	55.0	46.6	38.4	30.2	23.0
12	96.6	84.9	71.8	59.3	45.1	37.1	29.6	23.5
11	98.6	87.1	74.8	63.1	49.9	36.7	29.5	22.6
10	99.7	87.9	75.2	64.3	51.6	40.4	29.2	22.6
9	99.2	88.0	77.0	65.7	54.8	42.8	31.5	21.7

numbers in this column are average numbers of transitions for 10 randomly generated multipliers. From Table I, it can be seen that the average numbers of transitions for optimized multipliers are from 22% to 53% smaller than the worst-case multipliers. Compared to random multipliers, optimized multipliers have from 15% to 32% smaller average numbers of transitions. The saving in power consumption is larger when more bits have zero one-probabilities. The reduction in power consumption with our current method is much larger than the one achieved in [17]. It reports a 7% to 23% improvement compared to worst-case multipliers (which is already shown to be better than the pure sorting approach described in [13]).

Table II compares the average number of transitions for $M_{OPT}(\Omega_i^{16}, \Omega_i^{16})$ when different static input probability vectors are applied. The multiplier that is optimized for Ω_i^{16} is the one that consumes the least power when Ω_i^{16} is applied. However, when a different input static probability vector is applied, the power consumption of this multiplier is not minimum any longer. From Table II we can conclude that in a system with dynamically varying data word-length, it is beneficial to optimize the multiplier to the word-length that is most frequently used. The static probabilities of the multiplier input operands depend strongly on the nature of those signals. So far in our examples we have assumed that the input signals have a uniform distribution and that the word-length is varying. Our second experiment involves unequal static probability assumptions for the primary input bits. For many natural signals the switching activities of the MSB-bits are lower than that of LSB-bits. In these cases, the multiplier structure can be optimized for the pattern of static probabilities of the input bits. For this purpose we choose input signals with log-normal distribution. The log-normal distribution is associated with any random variable whose logarithm is normally distributed, which is very common for natural signals. It can model any variable that can be thought of as the multiplicative product of many small independent identically-distributed factors. The probability density function is given by:

$$f_X(x) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{(\ln x - \mu)^2}{2\sigma^2}}, \qquad x > 0 \qquad (6)$$

where σ and μ are the mean and standard deviation of the variable's logarithm respectively. In our experiments σ is chosen to be one. μ assumes three different values: $\ln 2^8$, $\ln 2^9$ and $\ln 2^{10}$. Figure 5 illustrates the one-probabilities of input bits assuming these values for μ and σ. The probability distribution of such random variables are shown in Figure 4. Table III specifies the average number of transitions for multipliers that are optimized for different input static probability vectors. The multiplier generated using the `optimized_multiplier` algorithm in general consumes less power than a random multiplier, even for cases where the input static probability vector that is applied is not what the multiplier is optimized for. From this we can conclude that a part of the optimization is independent from the input static probabilities. However, further power savings can be achieved if the optimization is performed for the input static probabilities of multiplier's inputs. For example if the inputs are variables with log-normal distribution ($\mu = \ln 2^8$), the multiplier that is optimized for these static probabilities consumes 5% less power than $M_{OPT}(\Omega_{16}^{16}, \Omega_{16}^{16})$.

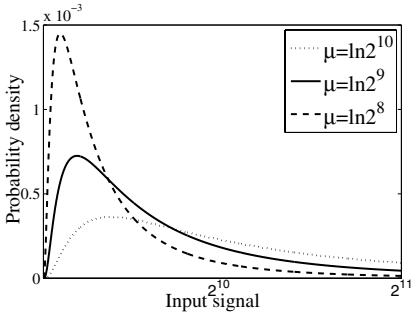


Fig. 4. Log-normal signal distribution

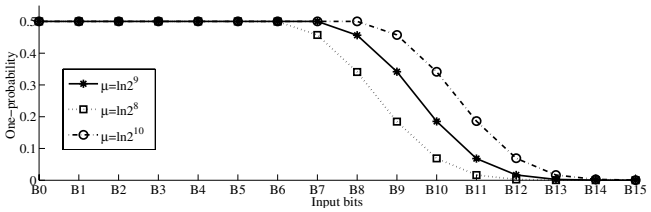


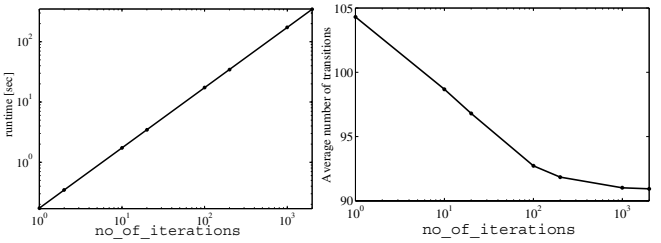
Fig. 5. One-probability of input bits assuming log-normal input signal distribution (for $\mu = \ln 2^8$, $\mu = \ln 2^9$ and $\mu = \ln 2^{10}$)

V. Runtime and Complexity

The power estimate function is located in the core of three loops in the algorithm described in Figure 3. The computational complexity of the power estimator is proportional to the product of the number of nodes and the average logic depth [23]. The number of nodes is approximately equal to the number of bits in `current_col` of `current_stage`. The upper-bound for the average logic depth is $2 \times$ `current_stage`. Therefore for an $n \times n$ bits multiplier, the computation complexity can be approximated to $O(\texttt{no\_of\_iterations} \times n \times (\log_{1.5}^{n})^2)$, as the number of stages required to compress the partial products is approximately $\log_{1.5}^{n}$. `no_of_iterations` is the number of iteration in the Simulated Annealing optimization. In our experiments, constructing a 16×16 bits multiplier using our method requires approximately 3 minutes on a PC with Intel

Table III- Average number of transitions for different multipliers

Multiplier structure	Input static probabilities applied to both operands: Uniform Ω_{16}^{16}	Log Normal $\mu = \ln 2^{10}$	Log Normal $\mu = \ln 2^{9}$	Log Normal $\mu = \ln 2^{8}$
$M_{OPT}(\Omega_{16}^{16}, \Omega_{16}^{16})$	91.48	43.86	34.64	27.84
$M_{OPT}(^{\text{Log Normal}}_{\mu = \ln 2^{10}})$	97.34	43.39	35.23	27.93
$M_{OPT}(^{\text{Log Normal}}_{\mu = \ln 2^{9}})$	99.90	44.21	34.30	27.10
$M_{OPT}(^{\text{Log Normal}}_{\mu = \ln 2^{8}})$	101.93	45.86	35.40	26.41
$M_{WC}(\Omega_{16}^{16}, \Omega_{16}^{16})$	111.98	54.09	43.27	33.66
$M_{WC}(^{\text{Log Normal}}_{\mu = \ln 2^{10}})$	109.61	55.42	43.12	34.00
$M_{WC}(^{\text{Log Normal}}_{\mu = \ln 2^{9}})$	107.88	53.13	44.46	33.62
$M_{WC}(^{\text{Log Normal}}_{\mu = \ln 2^{8}})$	107.71	52.79	42.95	34.17
M_{Random}	106.05	50.85	40.47	31.28



(a) (b)

Fig. 6. (a) Runtime versus number of iteration in Simulated Annealing (b) Average number of transitions versus number of iterations in Simulated Annealing

Pentium 1.4GHz processor. `no_of_iterations` is chosen to be 1000. Figure 6 shows the influence of `no_of_iterations` on the achieved results. When the number of iterations is small, the optimization algorithm generates a close-to-random multiplier. More iterations, which translate into longer runtime, result in more power efficient multipliers compared to a random multiplier. However the improvements saturate after a certain number of iterations and the algorithm is not able to produce better results.

VI. Conclusions

A method for generation of power optimized reduction tree interconnection order in parallel multipliers is presented. Energy saving is achieved without any noticeable area or speed overhead compared to a random reduction tree. Our method is based on Simulated Annealing optimization which employs probabilistic gate-level power estimation to compute cost functions. Automatically generated VHDL codes for the resulting multipliers are simulated using ModelSim and the average number of transitions are reported. The average number of transitions is reduced significantly (at least 15%) compared to random multipliers. This is significantly better than results from earlier similar work, both by us and others. The improvements are even larger if some input bits have low transition activity.

References

[1] E. de Angel and E. J. Swartzlander, "Low power parallel multipliers," *VLSI Signal Processing*, vol. IX, pp. 199–208, Oct. 1996.

[2] B. Parhami, *Computer Arithmetic - Algorithms and Hardware Design*. New York, US: Oxford University Press, 2000.

[3] I. Koren, *Computer Arithmetic Algorithms*, 2nd ed. A. K. Peters, Natick, Massachusetts, 2002.

[4] M. D. Ercegovac and T. Lang, *Digital Arithmetic*. Morgan Kaufmann Publishers, 2004.

[5] C. Baugh and B.A.Wooley, "A twos complement parallel array multiplication algorithm," *IEEE Trans. Computers*, vol. C-22, pp. 1045–1047, Dec. 1973.

[6] O. MacSorley, "High-speed arithmetic in binary computers," *Proc. of the IRE*, vol. 49, no. 1, pp. 61–91, Jan. 1961.

[7] A. Booth, "A signed binary multiplication technique," *Quarterly Journal of Mechanics and Applied Mathematics*, vol. 4, no. 2, pp. 236–240, 1951.

[8] C. Wallace, "A suggestion for a fast multiplier," *IEEE Trans. Electronic Computer*, vol. 13, pp. 14–17, Feb. 1964.

[9] L. Dadda, "On parallel digital multipliers," *Alta Frequenza*, vol. 45, pp. 574–580, 1976.

[10] K. Bickerstaff, M. Schulte, and E. Swartzlander, "Reduced area multipliers," in *Proc. of ASAP*, Oct. 1993, pp. 478–489.

[11] A. Weinberger, "4:2 carry-save adder module," *IBM technical disclosure bulletin*, vol. 23, Jan 1981.

[12] P. Song and G. de Micheli, "Circuit and architecture trade-offs for high-speed multiplication," *IEEE Journal of Solid-State Circuits*, vol. 26, no. 9, pp. 1184–1198, Sept. 1991.

[13] Z. Yu, L. Wasserman, and A. Willson, "A painless way to reduce power dissipation by over 18% in Booth-encoded carry-save array multipliers for DSP," in *Proc. SiPS*, Oct. 2000, pp. 571–580.

[14] P. Larsson and C. Nicol, "Transition reduction in carry-save adder trees," in *Proc. intr. symp. on Low power electronics and design*, 1996, pp. 85–88.

[15] V. Oklobdzija, D. Villeger, and S. Liu, "A method for speed optimized partial product reduction and generation of fast parallel multipliers using an algorithmic approach," *IEEE Trans. on Computers*, vol. 45, no. 3, pp. 294–306, 1996.

[16] P. Mokrian, G. Howard, G. Jullien, and M. Ahmadi, "On the use of 4:2 compressors for partial product reduction," in *IEEE Canadian Conf. on Electrical and Computer Engineering*, vol. 1, May 2003, pp. 121–124.

[17] S. Tahmasbi Oskuii, P. Kjeldsberg, and O. Gustafsson, "Transition-activity aware design of reduction-stages for parallel multipliers," in *Proc. of Great Lakes Symposium on VLSI*, March 2007, pp. 120–125.

[18] F. Najm, "Power estimation techniques for integrated circuits," in *Proc. Intl. Conf. on Computer Aided Design*, 1995, pp. 492–499.

[19] F. N. Najm, "Transition density: a new measure of activity in digital circuits," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 12, no. 2, pp. 310–323, 1993.

[20] F. Najm, R. Burch, P. Yang, and I. Hajj, "Probabilistic simulation for reliability analysis of CMOS VLSI circuits," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 10, pp. 1372–1381, November 1990.

[21] C. Ding, C. Tsui, and M. Pedram, "Gate-level power estimation using tagged probabilistic simulation," *IEEE Trans. on Comput.-Aided Des. Integr. Circuits Syst.*, vol. 17, pp. 1099–1107, November 1998.

[22] F. Hu and V. Agrawal, "Dual-transition glitch filtering in probabilistic waveform power estimation," in *Proc. 15th ACM Great Lakes symposium on VLSI*, April 2005, pp. 357–360.

[23] S. Tahmasbi Oskuii, P. Kjeldsberg, and E. Aas, "Probabilistic gate-level power estimation using a novel waveform set method," in *Proc. of Great Lakes Symposium on VLSI*, March 2007, pp. 37–42.

[24] S. Ercolani, M. Favalli, M. Damiani, P. Olivo, and B. Riccó, "Estimate of signal probability in combinational logic networks," in *Proc. 1st European Test Conf.*, 1989, pp. 132–138.

[25] K. Parker and E. McCluskey, "Probabilitic treatment of general combinational networks," *IEEE Trans. Computers*, vol. 24, no. 6, pp. 668–670, 1975.

[26] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, 4598, pp. 671–680, May 1983.