Transition-activity Aware Design of Reduction-stages for Parallel Multipliers

Saeeid Tahmasbi Oskuii
Norwegian University of Science and Technology
7491 Trondheim, Norway
oskuii@iet.ntnu.no

Per Gunnar Kjeldsberg
Norwegian University of Science and Technology
7491 Trondheim, Norway
pgk@iet.ntnu.no

Oscar Gustafsson
Linköping University
58183 Linköping, Sweden
oscarg@isy.liu.se

ABSTRACT

We propose an interconnect reorganization algorithm for reduction stages in parallel multipliers. It aims at minimizing power consumption for given static probabilities at the primary inputs. In typical signal processing applications the transition probability varies between the most and least significant bits. The same is the case for individual signals within the multiplier. Our interconnect reorganization exploits this to reduce the overall switching activity, thus reducing the multiplier's power consumption. We have developed a CAD tool that reorganizes the connections within the multiplier architecture in an optimized way. Since the applied heuristic requires power estimation, we have also developed a very fast estimator fine tuned for parallel multipliers. The CAD tool automatically generates gate-level VHDL code for the optimized multipliers. This code and code for unoptimized multipliers have been compared using state of the art power estimation tools. The reduction in power consumption ranges from 7% up to 23% and can be achieved without any noticeable overhead in performance and area.

Categories and Subject Descriptors

B.2.1 [**ARITHMETIC AND LOGIC STRUCTURES**]: Design Styles;
B.6.1 [**LOGIC DESIGN**]: Design Styles—*Combinational logic*

General Terms

Algorithms, Design

Keywords

parallel multiplier, partial product reduction, power consumption, transition activity

1. INTRODUCTION

Multipliers, being among the most frequently used parts of digital systems, have continuously been subjects to in-depth research. They occupy a relatively large area of the total chip and are often among the limiting parts for speed. Furthermore, the multiplier is one of the main contributors to the total power consumption of a digital system and much work has been aimed at minimizing power consumption of multipliers [1][2]. Our focus in this paper is on parallel multipliers. Three separate computation steps can be assumed for parallel multipliers:

1. Form all partial products in parallel
2. Reduce these partial products through series of reduction stages
3. Sum the final two outputs generated in step 2 using a carry propagate adder and compute the final output.

Each of these steps have been studied by researchers for years. Several algorithms are proposed to generate the partial products optimally (step 1) e.g. [3], [4] and [5]. Minimization of the number of partial products, minimization of transition activities of generated partial products and overcoming the problem of sign-bit extension in two's complement multiplications are the main objective of such algorithms. [6], [7], [8] and [9] propose methods for second step, i.e reduction stages. The method proposed by Wallace [9] and refined by Dadda [7] are today normally the basis for the second computation step, which is the main focus of this paper. Both techniques employ full adders and half adders in order to compute the partial additions. Habibi and Wintz in [10] compare the speed, complexity and cost of a number of schemes for fast multiplication including Wallace, Dadda and Braun's schemes [11]. They conclude that the approaches of Wallace and Dadda provide the fastest multipliers while Dadda's method is proven to require the minimum computational resources. In addition, the computational complexity bound of $O(\log_2 W)$, which has been proven to be the smallest time needed to perform multiplications, is attained by these multipliers. Bickerstaff et al introduce a small change in the Dadda scheme [6]. Without introducing any delay overhead, a few extra half-adders are added in the second computation step leading to smaller output vectors from this step. The final carry propagate adder will thus be smaller and faster. In our investigations we use the technique provided in [6] since it has minimum number of reduction stages and needs a smaller propagation adder to produce the final output. In this scheme the second computation step runs as follows. For each stage, the number of full-adders used in each column is $\lfloor b_i/3 \rfloor$, 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

GLSVLSI'07, March 11–13, 2007, Stresa-Lago Maggiore, Italy.

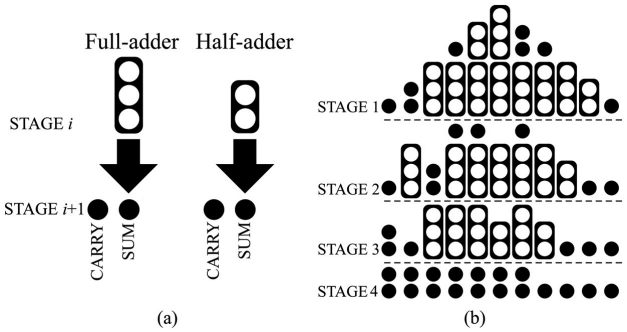

Figure 1: (a) Simplified representation of full-adders and half-adders. (b) Modified Dadda reduction scheme for 6×6 unsigned multiplier

number of bits in a column to the number of bits specified by the Dadda series, or (b) to reduce the rightmost column containing exactly two bits.

Fig. 1.b illustrates a 6×6 unsigned multiplier. Each dot corresponds to a single-bit partial product. Vertical lines connecting 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 (one order of magnitude higher).

The rest of this paper is organized as follows. Section 2 describes power consumption in parallel multipliers. In this section we also discuss effects of switching activity on power consumption. Sections 3 and 4 describe our heuristic and CAD tool for optimized design of the reduction stages, including previous work in this field. Finally, Section 5 demonstrates the usefulness of the techniques through a number of experiments.

2. POWER CONSUMPTION IN PARALLEL MULTIPLIERS

Power consumption in digital circuits can be divided into static and dynamic power. Our optimization techniques focus on interconnects of a fixed set of half- and full-adders. The influence on the static power consumption, mainly leakage power, is negligible. The average dynamic power consumption of digital circuits is given by [12]:

$$\text{Total Power} \simeq P_{dyn} = \frac{1}{2} {V_{DD}}^2 f_c \sum_{i=1}^{N} C_i \alpha(x_i) \qquad (1)$$

where V_{DD} is the supply voltage and f_c is the clock frequency. C_i is the load capacitance at node x_i and N is the total number of nodes in the multiplier structure. $\alpha(x_i)$ is the switching activity at node x_i and is strongly affected by the temporal and statistical characteristics of signals in the circuit. If the signals on the inputs of a gate arrive at different points in time, this may result in glitches at the output node of the gate. These glitches directly increase the switching activity in the multiplier. The probability that two signals have the same logic value will in the following be referred to as the correlation probability. For the signals K and L this is denoted as $p_{K,L}$. Assuming at most

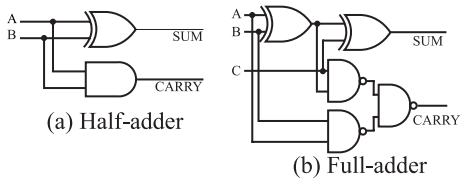

Figure 2: Gate level representation of half-adder and full-adder

Table 1: Static probabilities and functional logic definition of SUM and $CARRY$ for full-adders and half-adders

	Full-adder	Half-adder
SUM	$A \oplus B \oplus C$	$A \oplus B$
$CARRY$	$A \cdot B + B \cdot C + C \cdot A$	$A \cdot B$
p_{SUM}	$p_A + p_B + p_C + 4p_A p_B p_C$ $-2(p_A p_B + p_B p_C + p_C p_A)$	$p_A + p_B$ $-2p_A p_B$
p_{CARRY}	$p_A p_B + p_B p_C + p_C p_A - 2p_A p_B p_C$	$p_A p_B$

one logic change per clock cycle, the switching activity at a node x can be defined as the probability that the logic value at the node changes between two consecutive clock cycles:

$$\alpha(x) = 1 - p_{x(n),x(n-1)} \qquad (2)$$

For a given logic element switching activity at its output(s) can be computed using the probability of its inputs. Under the assumption of strong temporal independence, switching activity $\alpha(x)$ can be represented as:

$$\alpha(x) = 2p_{x=1}p_{x=0} = 2p_{x=1}(1 - p_{x=1}) \qquad (3)$$

where $p_{x=0}$ and $p_{x=1}$ denotes the probability of zero and one at node x respectively. For simplicity in this paper we will use p_x instead of $p_{x=1}$. $p_x = 0.5$ maximizes $\alpha(x)$ and any skew from 0.5 will reduce $\alpha(x)$.

Half-adders and full-adders are basic elements which are frequently used in parallel multipliers, especially in the partial products reduction stage (Fig. 2). Functional representation of CARRY and SUM is shown in Table 1 where $\oplus$ represents a boolean XOR, + represents a boolean OR and $\cdot$ represents a boolean AND function. The probability of one at the output of these blocks is a function of the probability of one at the inputs [13][14]. Static probabilities given in Table 1 can be determined from the functional definition of carry and sum given in Table 1 assuming independent inputs.

It can be easily seen from Table 1 that static probabilities of partial products generated in the multiplier will diverge and differ through reduction steps, even when they are equal at the primary inputs. In a standard multiplier for example, assuming 0.5 one-probabilities at the primary inputs of the multiplier, the resulting one-probability for initial partial products will be 0.25. This is because the partial products are results of AND operations between one bit from each operand. Assigning 0.25 as one-probability at the inputs of a full-adder will result in $p_{SUM} = 0.4375$ and $p_{CARRY} = 0.15625$. Thus from the very beginning in the partial products reduction step, different bits will have different transition probabilities. Partial products generated using a Booth encoding technique also has different one-probabilities [15]. Furthermore, in typical signal processing applications the transition activity for the most significant bits is lower than for the least significant bits since the changes in input values tend to be much smaller than the

whole dynamic range. If this variation in one-probabilities for the individual input operand bits is taken into account, then this will increase the differences between the transition activities of the partial product bits. Our goal in this paper is to utilize the differences between the one-probabilities of the partial product bits. We exploit these differences in the design process to minimize the overall transition activity in the multiplier and thereby reducing its power consumption.

3. HEURISTIC FOR INTERCONNECT REORGANIZATION

Several reduction stages are required to reduce the partial products generated in a parallel multiplier. As shown in Fig. 1, at each stage a number of bits with the same order of magnitude must be grouped and connected to adder cells. The selection of these bits and their grouping influences the switching activity of the next stages. This is what we will exploit to reduce the overall switching activity of the multiplier. One brute force approach to find the optimal solution is to perform a exhaustive search of all implementation alternatives and estimate their power consumption. As demonstrated in Table 2, this is obviously impossible because of the huge number of possibilities. Therefore utilizing simpler methods and heuristics are inevitable. As discussed below, we have done a number of important observations regarding the switching activity. They will be used to focus the search at the interesting parts of the solution space. The discussion below also includes the little previous work we have found in this field.

We consider only one column in one stage at a time. As the generated carry bits from adders propagate from LSB towards MSB, optimization of columns is performed from LSB to MSB and from first stage to last stage. Thus it can be ensured that the optimization of columns and stages that has already been performed will still be valid when later optimizations are being performed.

We here assume that optimization of columns in early stages of reduction can be performed without taking the organization of interconnection in later stages into account. This localization of search might in general result in suboptimal structures because of the fact that the structure of later stages may influence the optimal solution for earlier stages. However, this small degradation from the optimal solution reduces the search space dramatically. Localization of search space has been employed in numerous optimization methods, especially in optimization algorithms for hardware design [16]. The observations below are provided under this assumption of independence between columns.

Glitches and spurious transitions spread in the reduction stage after a few layers of combinational logic. To avoid them is not feasible in most cases. Therefore it seems beneficial to assign partial products having high switching activity a short path and thus perform addition of bits with lower switching activities in the earlier stages of the reduction tree and keep the ones with higher switching activity for the later stages. The partial products first added will affect larger parts of the reduction tree.

Our experiments, as well as other work that has employed the above mentioned assumption [17], show that by sorting the transition activities and assigning bits to full-adders and half-adders a noticeable power saving can be achieved. [17] focuses on Booth-encoded carry-save multipliers while we generalize this for any parallel multiplier and any reduction stage. As discussed earlier Booth-encoded multipliers [4] adjust the first step of the multiplication process, i.e. partial product generation. Our technique is focused on the second reduction step.

Table 2: Number of search alternatives in one column

Number of bits in one column	3	6	9	12	18	24
Number of search alternatives in exhaustive search	1	20	1680	3.7e5	1.4e11	3.7e17
Number of search alternatives with our assumptions	1	3	12	55	1428	43263

The path of the Carry-in input of a full-adder (C in Fig. 2) is shorter than the other two inputs. A transition on this input will therefore result in less activity. The input with the highest transition activity among three inputs of the full-adder should therefore be connected to the Carry-in input. For a full-adder with inputs (A, B, C), $max(\alpha_A, \alpha_B) < \alpha_C$.

In a full-adder, transition in input C affects only one XOR gate while transition in A and B affects two XOR gates. Obviously, under the single column assumption, the power consumption of such an addition will be minimized. This property has been the main focus of other works such as [18]. However, the main objective in [18] is speed optimization rather than power optimization.

The transition activities of the outputs of full-adders are in general close to the largest input transition activity. Thus the transition activities of the three inputs should be as similar as possible in order to minimize the overall transition activity of the multiplier. For any two full-adders x and y with inputs (x_A, x_B, x_C) and (y_A, y_B, y_C) respectively, the following property should hold. If $\alpha_{x_C} < \alpha_{y_C}$ then $max(\alpha_{x_B}, \alpha_{x_A}) < min(\alpha_{y_B}, \alpha_{y_A})$.

As we demonstrate in this paper, our observations regarding transition activity are very important for the overall power consumption. We therefore employ these to prune away a large number of uninteresting solutions. Table 2 shows the number of search alternatives for one column as a function of number of bits in that column. Although the number of search alternatives is still exponentially growing, the growth rate is much slower, and a search can be tractable even for 24 bits in one column.

For each alternative interconnection pattern, power consumption of the multiplier needs to be estimated. Because the speed of the power estimation is a critical performance factor for our heuristic, we have developed a specialized estimation program in MATLAB for our parallel multipliers. See the next Section. This also facilitates efficient interaction between our optimization algorithm and the power estimation program. The overall optimization heuristic is summarized in Fig. 3. We iterate over all stages, starting with the one closest to the primary outputs. For each stage we iterate over all columns, starting with the LSB. We use our observations to find the permutations that can hold the best solution for the current column. Power estimates are used to select the best one.

Fig. 4 gives an example where 7 bits with the same order of magnitude are to be added. According to the rules of designing the reduction tree, 2 full-adders must be used and one bit will be kept for the next stage together with the sum and carry bits generated by the full-adders. Using our observations in this section, we keep the bit with the highest

```
optimize_multiplier(static probabilities of input bits)
for current_stage = 1 to no_of_stages{
  for current_col = 1 to no_of_columns{
    possible_permutations = list of all useful permu-
              tations of inputs for half- and full-adders
    for all m ∈ possible_permutations{
      update transition activities;
      estimate power consumption of the multiplier;
      if lowest estimate found so far
          keep m for next stages and columns;
} } };
```

Figure 3: Algorithm `optimize_multiplier()`

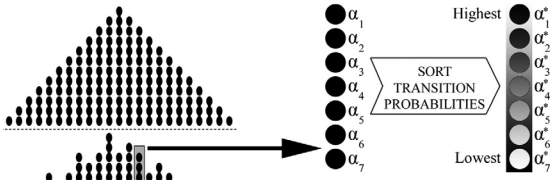

Table 3 - Possible permutations of 7-bits

#	To next stage	Full-adder 1 A	B	C	Full-adder 2 A	B	C
1	α_1^*	α_4^*	α_3^*	α_2^*	α_7^*	α_6^*	α_5^*
2	α_1^*	α_5^*	α_3^*	α_2^*	α_7^*	α_6^*	α_4^*
3	α_1^*	α_5^*	α_4^*	α_2^*	α_7^*	α_6^*	α_3^*

Figure 4: Example of the reduction tree design and assignments to full-adders

transition activity for the next stages and assign the other bits to the full adders. The three interesting permutations of these seven bits that will be examined in our optimization algorithm are given in Table 3. For comparison, a full search of these permutations would require examination of 7! = 5040 permutations. The ordered α_i^*s in Table 3 are found through a sorting of the transition activities α_is of the 7 bits, as illustrated in the right part of Fig. 4.

As discussed above, the heuristic can not guarantee finding the optimal solution. We have, however, performed a full search of all possible implementations for a 6×6 bit multiplier. Our heuristic found the optimal solution. Due to the number of permutations, it has not been possible to perform a full search of larger multipliers.

4. POWER ESTIMATION

After assigning a set of different permutations to the bits of a particular column, the power consumption of each configuration is estimated. Statistical power estimation methods are favorable due to straightforward utilization of input static probabilities [12][19]. The average transition activities of individual circuit nodes is estimated and accordingly these transition activities are propagated through the whole circuit. Glitches are thus taken into account in a probabilistic way. General power estimation tools, such as Power-Compiler from Synopsys, must be able to take all types of circuits into account and are consequently relatively slow. To speed up our estimation we have developed an estimation algorithm in MATLAB which is fine tuned for our purpose, and therefore very fast. Reconvergent fanouts that appear in the structure of the multiplier create dependencies between partial products. However, exact consideration of these dependencies is computationally expensive. Therefore in order to achieve higher speed of power estimation we choose to neglect the dependencies between partial products; i.e we assume that knowledge of logic value for a certain partial product does not change the static probabilities of the others. As shown in Table 4 comparing with accurate results provided by Synopsys Power Compiler, the estimation error due to neglecting these dependencies is small, as the interdependencies spread widely in the circuit because of the structure of the multiplier, and their effects are in a random manner. The transition activities are calculated for every node using the static probabilities and the transition activities of the preceding nodes. In order to estimate transition activities of the given circuit, we perform a linear approximation using data acquired from the Synopsys tools Design-Compiler and Power-Compiler for elementary gates that are used in adder circuits, e.g. XOR, NAND and AND gates. The target technology used in our simulations is a typical $0.35\mu m$ CMOS library. Estimated transition activities are propagated through all nodes in the design from the input towards the outputs. This information is used together with node load information, which is also extracted from our target library, to calculate the dynamic power of the multiplier. As shown in Table 4, the estimation error is less than 1% compared with power estimates provided by Power-Compiler with transition activity propagation (the table will be discussed in detail in the next Section). Our estimation algorithm runs on the MATLAB platform at a speed about 80 times faster than Power-Compiler. Using a Pentium *IV* 1.4GHz machine one power estimation run for a 12×12 multiplier is performed in 0.09 second with our tool while it takes 8 seconds using Power-Compiler. For a 24×24 multiplier the times are 0.29 and 31 seconds respectively. The run-time differences between the tools are important since the estimates are performed a large number of times during an execution of the optimization algorithm.

Table 4 - Estimated power for different multipliers [mW]

		$M_{OPT}(\Omega_{12}^{12}, \Omega_{12}^{12})$	$M_{WC}(\Omega_{12}^{12}, \Omega_{12}^{12})$	Averaged Random Multiplier	$M_{SORT}(\Omega_{12}^{12}, \Omega_{12}^{12})$
Ω_{12}^{12} set as input static probabilities of both operands	Power-Compiler with accurate transition activities from ModelSim	98.04	105.14	102.23	100.54
	Power-Compiler with transition activity propagation	102.40	108.18	105.97	104.31
	Our power estimation tool in MATLAB	102.26	108.28	106.06	104.52
Ω_{6}^{12} set as input static probabilities of both operands	Power-Compiler with accurate transition activities from ModelSim	28.26	33.94	31.15	29.56
	Power-Compiler with transition activity propagation	29.19	34.78	32.70	30.81
	Our power estimation tool in MATLAB	29.23	34.90	32.67	30.89

5. EXPERIMENTS

The optimization algorithm as well as a gate-level VHDL code generator for the optimized structure is implemented in MATLAB. Inputs to the optimization program are the static probabilities of the input bits. To prove the quality of our methodology, we have performed more accurate power estimates on the generated VHDL code. Transition activity of every single node in the designed multiplier is calculated in ModelSim using large number of input vectors with the desired input static probabilities. These transition activities are fed into Synopsys Power-Compiler and are utilized for power estimation. A typical $0.35\mu m - 3.3V$ CMOS

Table 5: Estimated power for optimized multipliers [mW]

Various multiplier structures	Input static probabilities applied to both operands						
	Ω_{12}^{12}	Ω_{11}^{12}	Ω_{10}^{12}	Ω_{9}^{12}	Ω_{8}^{12}	Ω_{7}^{12}	Ω_{6}^{12}
$M_{OPT}(\Omega_{12}^{12},\Omega_{12}^{12})$	98.04	87.70	74.07	61.24	49.90	37.87	28.26
$M_{OPT}(\Omega_{11}^{12},\Omega_{11}^{12})$	98.32	87.68	73.90	60.69	49.60	38.04	28.26
$M_{OPT}(\Omega_{10}^{12},\Omega_{10}^{12})$	98.29	87.97	73.65	59.67	48.68	37.60	28.26
$M_{OPT}(\Omega_{9}^{12},\Omega_{9}^{12})$	98.96	88.83	74.40	59.14	48.33	37.07	28.14
$M_{OPT}(\Omega_{8}^{12},\Omega_{8}^{12})$	98.65	88.55	74.55	59.91	48.28	36.74	27.65
$M_{OPT}(\Omega_{7}^{12},\Omega_{7}^{12})$	98.63	88.59	74.94	60.85	49.35	36.57	27.10
$M_{OPT}(\Omega_{6}^{12},\Omega_{6}^{12})$	98.66	88.69	75.14	61.73	50.03	37.34	26.90

library is used for VHDL simulation in ModelSim as well as synthesis and power estimation in Design-Compiler and Power-Compiler. ModelSim simulations consider the realistic delays provided from the library and therefore, propagation of glitches and spurious transitions are more accurately taken into account here compared to the simulations with transition activity propagation. We use two multiplier versions in our comparisons. The optimized structure generated by algorithm `optimize_multiplier`, and the worst case structure generated by another algorithm similar to `optimize_multiplier` but aiming at getting the maximum possible switching activity when rearranging the interconnections in the reduction stages. A random multiplier version achieved if transition activities are not taken into account could fall anywhere in between. As shown in Table 4, experiments indicate that it will typically lie near the middle. This is also intuitively correct, since it is not likely that all of the many random interconnects will be optimal or that all will be worst case. $M_{SORT}(\Omega_i^n, \Omega_j^m)$ in rightmost column of Table 4 is the multiplier obtained by only sorting the partial products by their transition activity and assigning the bits with low transition activities earlier in reduction stages (the approach from [17]). No other interconnect optimization is performed on $M_{SORT}(\Omega_i^n, \Omega_j^m)$.

In the first part of our experiments we use a simplified model for input probabilities. We assume that static probabilities of primary input bits to the multiplier can only have two values 0.5 and 0. We represent input probabilities with Ω_j^i, where i is the number of input bits and j is the number of non-zero static probabilities. We also assume that input bits with zero static probabilities will be located in the LSB part of the input word. This situation can happen in systems where the resolution of signals vary dynamically. If the static probability of input word $A_{0..5}$ is represented by Ω_4^6, it means input bits A_0 and A_1 have 0 one-probabilities and input bits A_2, A_3, A_4 and A_5 have 0.5 one-probabilities. A power-optimized $n \times m$ multiplier with input static probabilities Ω_i^n and Ω_j^m resulted from the algorithm `optimize_multiplier` is represented by $M_{OPT}(\Omega_i^n, \Omega_j^m)$. $M_{WC}(\Omega_i^n, \Omega_j^m)$ is the multiplier generated by worst-case algorithm and input static probabilities Ω_i^n and Ω_j^m. In Table 4 the power consumption numbers are given for optimum multipliers and worst-case multipliers. Three values for power consumption are provided, one from our estimation tool in MATLAB, one from Power-Compiler with transition activity propagation and one from Power-Compiler with accurate transition activities provided by ModelSim VHDL simulations. The fidelity of the estimates are very good, as the difference between the optimal and worst-case solutions are almost the same for all techniques. This shows that our MATLAB estimation tool is well suited for selecting the best solution among a number of alternatives. Glitches are taken into account in all three power estimation methods. According to the results from power estimation with accurate transition activities, 22% of the power consumption in reduction stages of a 12×12 multiplier is due to glitches. Glitch-free portion of power consumption is obtained by computing the transition activities from ModelSim simulations with zero delay. Power estimation with transition activity propagation results in higher power numbers compared to power estimation with accurate transition activities due to a conservative handling of glitches. Using transition activity propagation technique, 25% of the power consumption in reduction stages of a 12×12 multiplier is due to glitches. Glitch-free portion of power consumption in this case is obtained by assigning switching activities using equation 3 (instead of using higher switching activities obtained statistical approximations methods used in Synopsys Power-Compiler). Table 4 and 5 provide the power estimations with accurate transition activities for 12×12 multipliers optimized for different number of active bits. With 6 active bits for both operands, the multiplier optimized for this, $M_{OPT}(\Omega_6^{12}, \Omega_6^{12})$, consumes 5.0% less power than $M_{OPT}(\Omega_{12}^{12}, \Omega_{12}^{12})$. $M_{OPT}(\Omega_6^{12}, \Omega_6^{12})$ consumes about 23% less power compared to $M_{WC}(\Omega_6^{12}, \Omega_6^{12})$ when Ω_6^{12} is applied to inputs. M_{WC} is an extreme case that is very unlikely to happen. In order to have fair comparison of our algorithm we have generated 10 random multipliers where the interconnects are chosen randomly and we provide averaged power numbers for these multipliers as well (column 3 in Table 4). $M_{OPT}(\Omega_6^{12}, \Omega_6^{12})$ consumes about 10.2% less power compared to the averaged power consumption of random multipliers with Ω_6^{12} as inputs. From this we can conclude that it is beneficial to optimize the multiplier to the number of bits that are active most of the time i.e. if the signal resolution varies dynamically, the multiplier should be optimized to the resolution that is used most of the time, not the best resolution.

The static probabilities of the input operands in a multiplier depend strongly on the nature of input signals. For many natural signals the static probability of the MSB-part is smaller than that of the LSB-part. In these cases, it can be very beneficial to optimize the multiplier structure for the given pattern of input static probabilities. We applied our proposed algorithm to three 12×12 multipliers that have inputs with log-normal distributions. The log-normal distribution is associated to any random variable whose logarithm is normally distributed. A log-normal distribution results if the variable is the product of a large number of independent, identically-distributed variables. Its probability density function is given by:

$$f_X(x) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{(\ln x - \mu)^2}{2\sigma^2}} \qquad x > 0 \qquad (4)$$

Three values $\ln 2^5$, $\ln 2^6$ and $\ln 2^7$ for μ are chosen in our experiments. σ is chosen to be one for all experiments. Fig. 5 illustrates the one-probabilities of input bits assuming these values for μ and σ. Both operands of the mul-

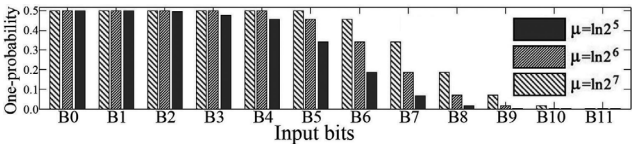

Figure 5: One-probabilities of input bits to the multiplier. Input signals for both operands have log-normal distribution with $\mu = \ln 2^5$,$\mu = \ln 2^6$ and $\mu = \ln 2^7$

Table 6: Power for $M_{\mu=\ln 2^5}$, $M_{\mu=\ln 2^6}$ and $M_{\mu=\ln 2^7}$ [mW]

Static probabilities of input bits (both operands)	Optimized 12×12 Multipliers				
	$M^{OPT}_{(\Omega^{12}_{12},\Omega^{12}_{12})}$	$M^{OPT}_{\mu=\ln 2^7}$	$M^{OPT}_{\mu=\ln 2^6}$	$M^{OPT}_{\mu=\ln 2^5}$	$M^{WC}_{\mu=\ln 2^5}$
Ω^{12}_{12}	98.04	98.54	98.62	98.79	104.52
Log-Normal $\mu = \ln 2^7$	56.02	54.23	54.70	54.82	61.81
Log-Normal $\mu = \ln 2^6$	44.09	42.95	42.60	42.98	48.97
Log-Normal $\mu = \ln 2^5$	33.82	32.84	32.71	32.27	37.86

tiplier have equal probability functions. Three multipliers, $M_{\mu=\ln 2^5}$, $M_{\mu=\ln 2^6}$ and $M_{\mu=\ln 2^7}$, are optimized for input static probabilities associated with log-normal distributions with $\mu = \ln 2^5$, $\mu = \ln 2^6$ and $\mu = \ln 2^7$ respectively. Table 6 summarizes estimated power for these multipliers with different input static probabilities. When log-normally distributed inputs with $\mu = \ln 2^5$ are applied to the multipliers, the $M_{OPT}(\Omega^{12}_{12}, \Omega^{12}_{12})$ uses 4.8% more power than $M_{\mu=\ln 2^5}$. The worst case multiplier requires 17.3% more power than $M_{\mu=\ln 2^5}$ in this case.

As discussed in Section 2, significant parts of the power reduction in the optimal multiplier appears regardless of input static probabilities. This can be seen in the differences between the two rightmost columns of Table 6. The rest depends on input static probabilities. According to Synopsys Design-Compiler's area estimation, the circuit and routing area of the generated multipliers are the same. This is because the difference between the generated multipliers is only in the interconnects of the adders. The required standard cells are exactly the same for all of the architectures. The speed of the energy-optimized multiplier is somewhat lower than the average version. Using Design-Compiler we have, however, not observed performance penalties of more than 1% for any of our generated optimized circuits compared to average performance of the randomly generated multipliers.

6. CONCLUSIONS AND FUTURE WORK

We have presented an interconnect reorganization algorithm for reduction stages in parallel multipliers. The resulting multipliers are optimized for power without adding any overhead in area. Automatically generated VHDL code for the multipliers have been synthesized using Synopsys Design-Compiler in a 0.35μm CMOS process. Power consumption is estimated by Power-Compiler by applying the transition activity information for all nodes provided from ModelSim simulations. Power estimations show from 7% to 23% improvement depending on the static probabilities of the input bits. The improvement is largest when input signals have varying activity probabilities for the different bits of the input word.

Further speed up of the power estimation step is achievable by only recalculate the power consumption for the parts of the multiplier that have changed. Furthermore, it should be possible to apply a similar approach to reduction stages with 4-2 compressors.

7. REFERENCES

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

[2] L. Wanhammar. *DSP Integrated Circuits*, chapter 11. Academic Press, New York, USA, 1999.

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

[4] A. Booth. A signed binary multiplication technique. *Journal of Mechanics and Applied Mathematics*, 4(2):236–240, 1951.

[5] O. McSorley. High speed arithmetic in binary computers. Proc. of IRE, 49(1):67–91, Jan. 1961.

[6] K. Bickerstaff, M. Schulte, and E. Swartzlander. Reduced area multipliers. In *Proc. of ASAP*, pages 478–489, Oct. 1993.

[7] L. Dadda. On parallel digital multipliers. *Alta Frequenza*, 45:574–580, 1976.

[8] P. Denyer and D. Myers. Carry-save arrays for VLSI signal processing. In *VLSI '81, (J.P. Gray, ed.), Academic Press, New York*, pages 151–160, 1981.

[9] C. Wallace. A suggestion for a fast multipliers. *IEEE Trans. Electronic Computer*, 13:14–17, Feb. 1964.

[10] A. Habibi and P.A.Wintz. Fast multipliers. *IEEE Trans. Computers*, 19:153–157, February 1970.

[11] E. Braun. *Digital Computer Design*. Academic Press, New York, USA, 1963.

[12] F. Najm. A survey of power estimation techniques in VLSI circuits. IEEE Trans. VLSI Systems, 2(4):446–455, Dec. 1994.

[13] K. Parker and E. J. McCluskey. Probabilistic treatment of general combinational networks. *IEEE Trans. on Computers*, C-24:668–670, June 1975.

[14] M. Cirit. Estimating dynamic power consumption of CMOS circuits. In *Proc. ICCAD*, pages 534–537, Nov. 1987.

[15] K. Khoo, Z. Yu, and A. Willson. Improved-booth encoding for low-power multipliers. In *Proc. of ISCAS*, volume 1, pages 62–65, May 1999.

[16] D. Gajski, F. Vahid, S. Narayan, and J. Gong. *Specification and Design of Embedded Systems*, chapter 6. Prentice Hall, New Jersey, USA, 1994.

[17] 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*, pages 571–580, Oct. 2000.

[18] 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*, 45(3):294–306, 1996.

[19] M. Xakellis and F. Najm. Statistical estimation of the switching activity in digital circuits. In *Proc. of DAC*, pages 728–733, 1994.