Power Optimization of Weighted Bit-Product Summation Tree for Elementary Function Generator

Saeeid Tahmasbi Oskuii*, Kenny Johansson†, Oscar Gustafsson†, and Per Gunnar Kjeldsberg*
*Department of Electronics and Telecommunications, Norwegian University of Science and Technology (NTNU),
NO-7491 Trondheim, Norway, Email: {saeeid.oskuii, per.gunnar.kjeldsberg}@iet.ntnu.no
†Department of Electrical Engineering, Linköping University, SE-581 83 Linköping, Sweden,
Email: {kennyj, oscarg}@isy.liu.se

Abstract— **In this paper we propose a method for lowering the power consumption in our previously proposed method for approximating elementary functions. By rearranging the interconnect ordering in the summation tree we show that it is possible to lower the power consumption in the range of 5.4% to 25.6% compared to a random ordering. The reduction tree is progressively designed and the interconnect ordering is decided based on the transition activities of the partial products. The reduction in power consumption comes with no overhead in performance or area compared to the random ordering.**

I. Introduction

Approximation of elementary functions finds applications in several areas [1]. For example, in direct digital frequency synthesis (DDFS), logarithmic number systems, transforms such as DFT (FFT) and DCT, computer graphics, and neural networks. Also it finds applications as seed value generation for Newton-Raphson-based division and square-root computations.

The authors have previously proposed a method for approximating elementary functions as a weighted sum of bit-products. The resulting architecture is composed of a number of and-gates and a summation tree, leading to an architecture that can be easily pipelined to an arbitrary degree. Starting with computation of sine and cosine an approach based on trigonometric identities was proposed [2]. In [3] an optimization procedure was outlined and the effect of finite word-length was studied. Then, [4] presented a modified architecture that turns off parts of the summation tree to reduce the power consumption. The proposed method was generalized to arbitrary elementary functions in [5]. In [6] the application of the proposed method to logarithmic number systems was studied. Here, some modifications to produce better results for certain functions were proposed. The resulting architecture has similarities with those proposed in [7]–[10]. However, our proposed method uses a completely different technique to derive the approximation. Our proposed approximation method is not limited by the behavior of the approximated function but can be used for all possible functions. However, the complexity depends on the function, for example, the complexity would be high for an irregular function composed by random values. Furthermore, the method in [11] has some similarities, given that the ROMs in [11] are only using one input-bit each.

Also, in another line of work the authors have looked at reducing the power consumption in parallel multipliers by utilizing knowledge about the switching activity of the input data [12], [13]. Based on this knowledge the interconnect of the summation tree is reordered to minimize the total switching activity in the multiplier.

In this work we show that similar ideas can be used in the design of the summation tree for the approximation of elementary functions. As the switching activity differs based on the number of bits in the bit-products it is shown that this strategy can result in even larger savings compared to a general multiplier.

In the next section we review the earlier proposed method for function approximation using a weighted sum of bit-products. For more details we refer the reader to [5]. Then in Section III the proposed optimization methodology is presented. Results of the optimization are presented in Section IV. Finally, our conclusions are presented in Section V.

II. Background

A. Function Approximation using a Weighted Sum of Bit-Products

A function, $f(X)$, where X is an integer composed of N bits, x_i, can be represented as

$$f(X) = \sum_{j=0}^{2^N-1} c_j p_j \qquad (1)$$

where c_j is the weight and p_j is a bit-product [5].

Each bit-product, p_j, is composed of the bits x_i that are one in the binary representation of j, hence

$$p_j = \begin{cases} 1 & j = 0 \\ \prod_{i \in S_j} x_i & j > 0 \end{cases} \quad \text{where} \sum_{i \in S_j} 2^i = j \qquad (2)$$

Note that the bit-products can be computed using simple logic AND-operations, for example, $p_{25} = x_4 x_3 x_0$ corresponds to a 3-input AND gate.

The weights, c_j, which in the following also are referred to as coefficients, are computed by vector multiplications according to

$$c_j = q_j F \qquad (3)$$

where F is a column vector containing all function values. The row vector q_j is the j:th row of Q_N, which is obtained

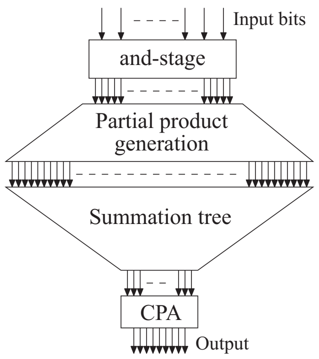

Fig. 1. Architecture used for approximating elementary functions.

by

$$Q_i = \begin{cases} 1 & i = 0 \\ \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix} \otimes Q_{i-1} & 1 \leq i \leq N \end{cases} \qquad (4)$$

where the $\otimes$ operation denotes the Kronecker product.

Using (3) the coefficients are directly obtained from the function values, for example, $c_0 = f(0)$, $c_1 = -f(0) + f(1)$, and so on. For functions with certain properties, basically all continous functions, many of the coefficients, c_j, will be small, i.e., truncated to zero for a limited coefficient wordlength. Hence, we can approximate a function $f(X)$ with $\hat{f}(X)$ as

$$\hat{f}(X) = \sum_{j \in G} c_j p_j \qquad (5)$$

where G is the set of indices for the M coefficients with largest absolute value, i.e. $|G| = M$ and $\min_{j \in G}\{|c_j|\} \geq \max_{j \notin G}\{|c_j|\}$.

B. Architecture

To implement the expression in (5) an effective architecture, suitable for high-speed implementations, was proposed in [2]. The architecture is illustrated in Fig. 1, where the and-stage is used to compute the required bit-products, p_j.

The partial product generation stage only shifts and possibly inverts the outputs of the and-stage, according to the corresponding coefficients, c_j. The bit-products are included in the columns corresponding to nonzero digits in the coefficient representation. Hence, by using a minimum signed digit (MSD) representation, the number of partial products to be added is decreased.

In the third stage, a summation tree accumulates the partial products. The reduction of partial products can be performed in logarithmic time using redundant multioperand adders with redundant outputs in carry-save form. Efficient schemes for a parallel summation tree was introduced by Wallace [14] and Dadda [15], [16]. These schemes are based on full-adders and half-adders and the reduction is achieved using several layers of full-adders/half-adders. In this work we have used a modified Wallace/Dadda reduction scheme proposed by Bickerstaff et al. in [17] for two reasons: First, it is optimal in terms of number of required full-adders and half-adders. Second, it produces shorter output vectors thus requiring smaller carry propagate adder (CPA). This scheme constructs the reduction tree as follows: For each stage, the maximum number of full-adders are used, i.e. $\lfloor \frac{b_k}{3} \rfloor$, where b_k is the number of bits in column k. Dadda in [15] specifies a sequence of intermediate partial product matrix heights that provides the minimum number of reduction stages. Half-adders are used only (a) when required to reduce the number of bits in a column to the number of bits specified in this sequence, or (b) to reduce the rightmost column containing exactly two bits.

Finally, the last stage is a CPA used to form a non-redundant representation of the output.

Note that the architecture can easily be pipelined to an arbitrary degree to obtain the desired throughput. In [4] it was shown that the power consumption can be decreased if the architecture is divided into multiple summation trees. For example, all bit-products that include the input bit x_i can be separated into a summation tree that is turned off when x_i is zero. In the context of this paper we do not consider this architecture, but the proposed method can be applied to each summation tree in the modified architecture, so they complement each other.

III. Proposed Method

The summation tree consists of weighted bit-products. The commutativity property of addition makes partial products with equal weight interchangeable without changing the functionality. However, the different solutions, obtained by interchanging equal weight partial products, differ in temporal and physical properties. Indeed, the freedom of choosing among the numerous permutations of equal weight partial products can be used to reduce the computation delay, dynamic power, static power and other measures of the circuit. These possibilities are considered in several works [12], [13], [18], [19]. In this work, we apply the progressive reduction-tree design method in [13] to the summation-tree of the elementary function generators. The algorithm for the progressive reduction tree design is summarized in Fig. 2. The algorithm generate_summation_tree() is executed after the generation of the primary partial products; i.e. `TheCircuit` is initialized to contain the AND-terms and the primary partial products generation step as described in Section II. The function compute_SWSs(`TheCircuit`) estimates the power consumption using the simple waveform set (SWS) method which is a probabilistic gate-level power estimator [20]. The power estimator computes a simple waveform set which is a set of possible transition times and their occurrence probabilities for each node of the circuit. The computation of the SWS for a node is based on the SWSs at the predecessor nodes; i.e. the computation starts from primary inputs and traverses towards outputs. Note that each time a circuit portion is appended to `TheCircuit`, this function is executed, computing the power estimation for the new portion using the estimated transition sets of the predecessor nodes. After the partial products are generated, the summation tree is progressively designed. For each column of each stage in the summation tree, a Simulated Annealing optimization [21] is performed. A randomly chosen permutations is the start configuration. This configuration is

```
generate_summation_tree()
Inputs: static probabilities of input bits and
        primary partial products generation pattern
initialize circuit TheCircuit to primary partial products;
compute_SWSs(TheCircuit);
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) OR (uniform random r ∈ [0,1) < $e^{\frac{minP-P}{\alpha}}$)
        minP=P;
      else
        current_perm=old_perm;
    }
    TheCircuit.append(FAs, HAs,current_perm);
    compute_SWSs(TheCircuit);
  }
}
```

Fig. 2. The proposed optimization algorithm

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 the innermost loop, estimate_power(), computes the transition densities exclusively for the current full-adder and half-adder stage, i.e. the stage that is being designed, using the SWSs computed earlier. This power estimation is limited to few full-adders/half-adders and is hence fast; making it feasible to perform a large number of iterations.

The algorithm generate_summation_tree() is implemented in C++. Inputs to the optimization algorithm are the static probabilities of the primary inputs and the partial product generation pattern. The power estimation in this algorithm uses a fanout delay model for its computations; i.e. the delay of a logic gate, regardless of its function, is equal to its number of fanouts and is an integer number. If needed, a more complex delay model can also be implemented. After completion of the optimization algorithm, the equivalent VHDL code for the circuit is generated. This algorithm constructs an optimum summation tree such that it consumes less energy while operating. A similar algorithm can be developed by altering generate_summation_tree() to maximize the energy consumption and generate a worst-case summation tree. We will compare the power consumption of the optimum and worst-case summation trees in the following section. Designing the summation tree randomly or regardless of the transition activities of partial products results in a power consumption value that lies somewhere between optimum and worst-case values. In fact our experiments show that the power consumption of random summation trees is biased towards the worst-case values.

Table I- Function definitions

Circuit	$f(X)$	Range of $f(X)$	# inputs	# coefs.	Coef. wordlength	Accuracy [bits]
SINE	$\sin(\pi X/4)$	$0 \le f(X) \le 1/\sqrt{2}$	9	89	20	16.00
COSINE	$\cos(\pi X/4)$	$1/\sqrt{2} \le f(X) \le 1$	9	91	20	16.00
LOG2-100	$\log_2 \lvert X \rvert$	$-8 \le f(X) < 0$	8	99	11	9.07
LOG2-149	$\log_2 \lvert X \rvert$	$-8 \le f(X) < 0$	8	149	14	12.00
Φ^--227	$\log_2 \lvert 1-2^X \rvert$	$-8.54 < f(X) < 0$	11	227	12	9.15
Φ^--239	$\log_2 \lvert 1-2^X \rvert$	$-8.54 < f(X) < 0$	11	229	11	9.00
Φ^--400	$\log_2 \lvert 1-2^X \rvert$	$-8.54 < f(X) < 0$	11	394	15	12.15
Φ^+-70	$\log_2 \lvert 1+2^X \rvert$	$0 < f(X) \le 1$	11	70	13	9.01
Φ^+-82	$\log_2 \lvert 1+2^X \rvert$	$0 < f(X) \le 1$	11	79	11	9.26
Φ^+-173	$\log_2 \lvert 1+2^X \rvert$	$0 < f(X) \le 1$	11	169	14	12.12

IV. Results

In this section the proposed method is applied to a number of elementary functions (Table I). The detailed definitions of these functions, except SINE and COSINE, are given in [6]. The number in the different circuit names correspond to the number of coefficients (after optimization some coefficients are truncated to zero so the number of bit-products used might be a few less). After the generation of the partial products, the algorithm generate_summation_tree() is executed for each example. The primary inputs of all circuits are assumed to be uncorrelated and random with uniform distribution. The one-probabilities of the primary input bits are set to be 0.5. The VHDL code resulting from the optimization algorithm is compiled for a 0.35μm standard cell CMOS library. The power consumption is then estimated using Synopsys NanoSim, which is an accurate circuit simulator featuring common HSPICE models for greater accuracy.

The results of the power estimation are given in Table II. The power is estimated for clock frequency at 20MHz. In addition to the accurate power consumption values obtained from NanoSim, the average transition activities of the circuits are reported in Table III. The average transition activities in Table III are obtained by simulating the resulting VHDL codes in ModelSim using a fanout delay model. Note that the optimization algorithm uses the fanout delay model.

In order to evaluate the quality of the optimized circuits, we compare the results with the worst-case circuits as well as the random circuits (the fourth columns in Tables III and II). For each circuit we have generated 10 versions where the interconnection between the full-adders and half-adders are chosen randomly without considering the transition activities. We have then estimated the average transition activities for these random circuit versions as well as the mean value of the average transition activities. In the tables we present results of the circuit version that has an average transition activity closest to the mean value. The rightmost columns in Tables III and II are the reduction in power consumption compared to the random circuits. Comparing the values in Tables III and II, we can conclude that the random circuits are closer to worst-case circuits in terms of average transition activities and power consumption.

In some cases the random circuits are estimated to consume more power than the worst-case circuits because of the

Table II- Power estimation results from NanoSim [mW]

Circuit	Optimized	Worst-Case	Random	Improvement
SINE	4.280	4.646	4.618	7.3%
COSINE	3.623	4.200	4.191	13.6%
LOG2-100	2.996	3.283	3.604	16.9%
LOG2-149	4.956	5.817	5.904	16.1%
Φ^--227	5.436	7.147	7.297	25.6%
Φ^--239	4.448	5.304	5.569	20.1%
Φ^--400	9.944	13.91	13.24	25.0%
Φ^+-70	2.112	2.608	2.562	17.6%
Φ^+-82	2.074	2.130	2.192	5.4%
Φ^+-173	4.620	5.604	5.717	19.2%

Table III- Average transition activity per clock cycle

Circuit	Optimized	Worst-Case	Random	Improvement
SINE	1334.2	2023.0	1807.5	26.2%
COSINE	1525.5	2204.7	1976.9	22.8%
LOG2-100	1029.4	1660.8	1477.9	30.4%
LOG2-149	1889.1	3145.4	2777.8	32.0%
Φ^--227	1871.1	2961.8	2691.6	30.5%
Φ^--239	1487.0	2393.5	2290.1	35.1%
Φ^--400	3484.5	5739.4	5278.5	34.0%
Φ^+-70	596.2	889.0	820.2	27.3%
Φ^+-82	548.1	775.3	700.1	21.7%
Φ^+-173	1400.4	2036.9	1869.9	25.1%

differences in the power estimator that is embedded in the optimization algorithm and NanoSim. The power estimation that is used in the optimization algorithm to generate the optimized and worst-case circuits is in gate-level and uses a fanout delay model; while NanoSim is an accurate transistor-level analog power estimator with a real delay computation mechanism based on the actual node capacitances. For example, NanoSim is capable of accounting for glitches that are not full-swing something the power estimator in the optimization program is incapable of. From Table II the power consumption for the optimized circuits is 3-29% lower than the worst-case. The gain is even larger when we compare the average transition activities in Table III (29-40%). This shows the capacity of the optimization method. Larger improvements in power consumption can be achieved if a more accurate delay model is used in the embedded power estimator.

The differences in improvements between the circuits can be explained by looking at the bit-products. For Φ^+-82 most bit-products are short, which limits the possibilities to obtain switching activity reductions as most bit-products have similar properties. For Φ^--400 there are 224 bit-products with more than 4 input bits and 24 with more than 8 input bits, hence, a lot of different interconnect choices can be made. Moreover, having a larger number of bit-products increases the possibilities for improvements. Even more important than studying long bit-products is probably the short ones. The circuits SINE, COSINE, and Φ^+-82 have almost the same number of bit-products. However, looking at the single bit products, i.e., the bit-products which are dependent on a single input bit and have highest switching activity, SINE has all 10 input bits as bit products, COSINE has 8, and Φ^+-82 has only 5; hence smallest savings are obtained for Φ^+-82. The improvement in power consumption compared to the random circuits is in the range of 5.4% to 25.6%.

V. Conclusion

In this work we have proposed a power reduction scheme for circuits approximating elementary functions based on weighted bit-products. By progressively constructing the summation tree focusing on the switching activity we obtain savings in the range of 5.4% to 25.6% compared to a random interconnect ordering.

References

[1] J.-M. Muller, *Elementary Functions: Algorithms and Implementation*, 2nd ed. Birkhäuser Boston, 2005.

[2] L. Wanhammar, K. Johansson, and O. Gustafsson, "Efficient sine and cosine computation using a weighted sum of bit-products," in *Proc. European Conf. Circuit Theory Design*, vol. 1, Cork, Ireland, 2005, pp. 139–142.

[3] O. Gustafsson, K. Johansson, and L. Wanhammar, "Optimization and quantization effects for sine and cosine computation using a sum of bit-products," in *Proc. Asilomar Conf. Signals, Syst., Comp.*, Pacific Grove, CA, 2005, pp. 1347–1351.

[4] K. Johansson, O. Gustafsson, and L. Wanhammar, "Low power architectures for sine and cosine computation using a sum of bit-products," in *Proc. IEEE NorChip Conf.*, Oulu, Finland, 2005, pp. 161–164.

[5] K. Johansson, O. Gustafsson, and L. Wanhammar, "Approximation of elementary functions using a weighted sum of bit-products," in *Proc. IEEE Int. Symp. Circuits Syst.*, Kos Island, Greece, 2006, pp. 795–798.

[6] K. Johansson, O. Gustafsson, and L. Wanhammar, "Conversion and addition in logarithmic number systems using a sum of bit-products," in *Proc. IEEE Norchip Conf.*, Linköping, Sweden, 2006, pp. 39–42.

[7] R. Stefanelli, "A suggestion for a high-speed parallel binary divider," *IEEE Trans. Computers*, vol. C-21, no. 1, pp. 42–55, 1972.

[8] E. M. Schwarz and M. J. Flynn, "Hardware starting approximation method and its application to the square root operation," *IEEE Trans. Computers*, vol. 45, no. 12, pp. 1356–1369, 1996.

[9] D. M. Mandelbaum and S. G. Mandelbaum, "A fast, efficient parallel-acting method of generating functions defined by power series, including logarithm, exponential, and sine, cosine," *IEEE Trans. Parallel Distrib. Syst.*, vol. 7, no. 1, pp. 33–45, 1996.

[10] D. D. Caro, E. Nappli, and A. Strollo, "Direct digital frequency synthesizers with polynomial hyperfolding technique," *IEEE Trans. Circuit & Syst. II*, vol. 51, no. 7, pp. 337–344, 2004.

[11] H. Hassler and N. Takagi, "Function evaluation by table look-up and addition," in *Proc. Symp. Comp. Arith.*, 1995, pp. 10–16.

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

[13] S. Tahmasbi Oskuii, P. G. Kjeldsberg, and O. Gustafsson, "Power optimized partial product reduction interconnect ordering in parallel multipliers," in *Proc. 25th IEEE Norchip Conf.*, Denmark, Nov. 2007.

[14] C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Transactions on Electronic Computers*, pp. 14–17, February 1964.

[15] L. Dadda, "Some schemes for parallel multipliers," *Alta Frequenza*, vol. 34, pp. 349–356, March 1965.

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

[17] K. C. Bickerstaff, M. J. Schulte, and E. E. Swartzlander, Jr., "Reduced area multipliers," in *Proc. Intr. Conf. on Application-Specific Array Processors*, 1993, pp. 478–489.

[18] 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. IEEE Workshop Signal Processing Syst.*, October 2000, pp. 571–580.

[19] 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. Computers*, vol. 45, no. 3, pp. 294–306, 1996.

[20] S. Tahmasbi Oskuii, P. G. Kjeldsberg, and E. J. Aas, "Probabilistic gate-level power estimation using a novel waveform set method," in *Proc. 17th Great Lakes Symp. on VLSI*, March 2007, pp. 37–42.

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