A General Method for Energy-Error Tradeoffs in Approximate Adders

Zvi M. Kedem*
Kirthi Krishna Muntimadugu†

NYU CS Technical Report TR-2013-952
March 15, 2013

Abstract

Approximate adders are adders with conventional architectures run in an overclocked mode. With this mode, erroneous sums may be produced at the savings of energy required to execute the computation. The results presented in this report lead to a procedure for allocating the available energy budgets among the adders modules so as to minimize the expected error. For simplicity, only the uniform distribution of the inputs is considered.

1 Introduction

The simplest and the most straightforward binary adder is the ripple-carry adder [1]. In [2], energy-error tradeoffs in approximate ripple carry adders were studied. Approximate adders have standard architectures but are overclocked and therefore for some inputs they produce incorrect sums. An approach was presented to optimize the allocation of the available energy budgets to the adder's components so that the average error is minimized.

The techniques in that paper relied on the specific architecture of ripple carry adders and were not applicable to general adders. We generalize those techniques so that they are applicable to other adders and use the Kogge-Stone adder [1] as an example. We will use the notation of [2] but modify it where it makes the presentation of our results clearer. Although this report is largely self contained, familiarity with that paper will be helpful, especially for the motivation and the technological constraints assumed.

2 Carry chains

We will be concerned with the addition of two integers in the range $[0, \dots, 2^n - 1]$, for some fixed n. Using standard powers of 2 representation, such numbers are in one-to-one correspondence with vectors of length n over $\{0, 1\}$. For an integer in the range $[0, \dots, 2^n - 1]$, say a, the corresponding vector will be denoted by $\mathbf{a}$ and also written as $(a_{n-1} \dots a_0)$. Such numbers and vectors will be referred to also as n-bit numbers and n-bit vectors. We will also consider some numbers in the range $[0, 2^{n+1} - 1]$ and corresponding vectors of length $n + 1$; they will naturally be referred to as $(n + 1)$-bit numbers and vectors. Generally a number discussed will be an n-bit number. Otherwise, its length will be stated explicitly or will be clear from the context.

Given two numbers a and b, we define two $(n + 1)$-bit numbers c and s by

$$c_i = \begin{cases} 0 & \text{if } i = 0 \text{ or } i > 0 \text{ and } a_{i-1} + b_{i-1} + c_{i-1} \le 1 \\ 1 & \text{otherwise,} \end{cases}$$

$$s_i = \begin{cases} c_n & \text{if } i = n \\ 0 & \text{if } i < n \text{ and } a_i + b_i + c_i \in \{0, 2\} \\ 1 & \text{otherwise.} \end{cases}$$

*New York University, zvi.kedem@nyu.edu
†kirthi.krishna@gmail.com

Thus, considering a standard right to left addition of two n-bit numbers s_i is the sum and c_{i+1} is the carry produced in position i, for $i = 0, \ldots, n-1$. By convention, $c_0 = 0$. Of course, $s = a + b$.

We adapt a standard definition of a carry chain [1] as follows

Definition 1. Given n-bit numbers there a and b, there is a *carry chain starting* at i and *ending* at j for $1 \leq i \leq j \leq n$ if and only if

1. $a_{i-1} = b_{i-1} = 1$
2. $a_k \neq b_k$ for $i \leq k \leq j-1$
3. $a_j = b_j$ or $j = n$

We will say that such a carry chain is *induced* by $i-1$ and also refer to it as an *i-j carry chain* or as carry chain from i to j. The set of all the carry chains will be denoted by $C(a,b)$.

The following notation will be helpful:

$$C_{ij}(a,b) = \begin{cases} \top & \text{if there is a carry chain starting at } i \text{ and ending at } j \\ \bot & \text{otherwise.} \end{cases}$$

When a and b are understood from the context, we may omit them.

There are two useful properties:

1. If $C_{ij}(a,b) = \top$ and $s = a + b$, then $s_k = 0$, for $k = i, \ldots, j-1$ and $s_j = 1$.
2. If $C_{ij}(a,b) = C_{pq}(a,b) = \top$ and $(i,j) \neq (p,q)$, then the intervals $[i,j]$ and $[p,q]$ are disjoint, or informally, the two carry chains do not overlap.

3 Graph model

The n-bit adders we consider are constructed out of n 1-bit adders (either full or half, depending on the design) and additional auxiliary circuitry. An n-bit adder will be modeled as a DAG $G = (V, E)$. The vertices in V are themselves, in general, circuits and are chosen so that it is possible to supply a specific voltage to the gates of a vertex, but it is not possible to supply different voltages to the gates in the vertex's circuit. We assume for now some specific voltage assignments, and will optimize them later.

We will refer to the vertices as *modules* , n of which are 1-bit adders. We assume that

1. The inputs a and b are provided at time 0
2. The sum s is read at time T
3. For each i, if $c_i = 0$, then the correct value of s_i is produced at time $\leq T$

We now consider some fixed a and b. For each i, the correct value of s_i is produced and does not change any more at time $d_i(a,b)$. All $d_i(a,b)$'s form vector $\mathbf{d}(a,b)$.

Consider some k, $1 \leq k \leq n$, and the case where for some i and j, $i \leq k \leq j$, and $C_{ij} = \top$. Then the information that $c_k = 1$ must be obtained from the inputs in positions $i-1, \ldots, k-1$. The time to account for the propagation of this information and the computation of the correct value of depends on the topology of G and the delays in the vertices in V. We denote this time interval by d_{ik}. All the d_{ik}s are collected into an array $\mathbf{D}$, in which only entries in positions (i,j) for $i \leq j$ are not empty; d_{ii} denotes the time to compute s_i if $c_i = 0$.

4 Carry chains and the errors

We fix $\mathbf{D}$ and T, so we do not list them for a while, though many of the variables depend on them. Given a and b, s and s^{A} are defined. Though they depend on a and on b, we do not indicate that explicitly if no confusion will result. Let $\delta = s - s^{\mathrm{A}}$. Define

$$\delta_k = \begin{cases} -1 & \text{if } d_{ik} > T \text{ and for some } i \text{ and } j, C_{ij}(a,b) = 1 \text{ and } i \leq k < j \\ 1 & \text{if } d_{ik} > T \text{ and for some } i \text{ and } j, C_{ij}(a,b) = 1 \text{ and } i \leq k = j \\ 0 & \text{otherwise.} \end{cases}$$

Then the error in the computation is

$$\mathrm{Er}(T, \mathbf{D}, a, b) = |\delta| = \left| \sum_{k=0}^{n} \delta_k 2^k \right|, \tag{1}$$

and the average error is

$$\mathrm{Er}_{\mathrm{avg}}(T, \mathbf{D}) = \operatorname*{avg}_{a,b} \mathrm{Er}(T, \mathbf{D}, a, b) = \frac{1}{2^{2n}} \sum_{a,b} \mathrm{Er}(T, \mathbf{D}, a, b). \tag{2}$$

So we need to compute

$$\sum_{a,b} \left| \sum_{k=0}^{n} \delta_k 2^k \right|, \tag{3}$$

which is the a sum of 2^{2n} terms, and therefore this task is impractical even for $n = 32$.

The set of all the carry chains has $n(n+1)/2$ elements. We will use C as a variable varying over this set. If C contributes an error for a pair inputs, we call it an *error-contributing* chain for those inputs and denote the error contributed by it by $\mathrm{EC}(C)$. For other chains, $\mathrm{EC}(C) = 0$. We can rewrite (3) as

$$\sum_{a,b} \left| \sum_{C \in C(a,b)} \mathrm{EC}(C) \right|. \tag{4}$$

For the special case where the adder considered is an RCA, the condition

$$\mathrm{EC}(C) \geq 0, \qquad \text{for every } C \tag{5}$$

is true. Therefore, for RCA, [2] used the following

$$\sum_{a,b} \left| \sum_{C \in C(a,b)} \mathrm{EC}(C) \right| = \sum_{a,b} \sum_{C \in C(a,b)} \mathrm{EC}(C) = \sum_{C} \mathrm{EC}(C) \cdot \mathrm{card}\left\{(a,b) \mid C \in C(a,b)\right\},$$

where card$\{\ldots\}$ stands for the cardinality of $\{\ldots\}$. Expression card$\left\{(a,b) \mid C \in C(a,b)\right\}$ can be easily computed for each chain C from Definition 1.

However, for adders other than RCA, (5) *does not necessarily hold*. So a different procedure for computing (4) must be established, which we do next.

Consider the case when $C(a,b)$ has at least one carry chain contributing to error for inputs a and b. Take the leftmost (largest position numbers) chain contributing an error, and denote it by $C_{\mathrm{L}}(a,b)$. Its error dominates the sum of the other errors contributed by all the other chains: in $C(a,b)$

$$\left|\mathrm{EC}\big(C_{\mathrm{L}}(a,b)\big)\right| > \left| \sum_{C \in C(a,b) \setminus C_{\mathrm{L}}(a,b)} \mathrm{EC}(C) \right|,$$

and therefore,

$$\operatorname{sgn}\left(\mathrm{EC}\big(C_{\mathrm{L}}(a,b)\big)\right) = \operatorname{sgn}\left(\sum_{C \in C(a,b)} \mathrm{EC}(C) \right).$$

Define

$$\widehat{\operatorname{sgn}}(a,b) = \begin{cases} \operatorname{sgn}\left(\mathrm{EC}\big(C_{\mathrm{L}}(a,b)\big)\right) & \text{if } C_{\mathrm{L}}(a,b) \text{ exists} \\ 0 & \text{otherwise.} \end{cases}$$

Then, we may rewrite (4) as

$$\sum_{a,b} \left(\widehat{\operatorname{sgn}}(a,b) \sum_{C \in C(a,b)} \mathrm{EC}(C) \right)$$

Positions	**Inputs**
$0, \dots, i-2$	00, 01, 10, 11
$i-1$	11
$i, \dots, j-1$	01, 10
j	00, 11
$j+1, \dots, p-2$	00, 01, 10, 11
$p-1$	11
$p, \dots, q-1$	01, 10
q	00
$q+1, \dots, n-1$	00, 01, 10

Figure 1: Input conditions for an *i-j* carry chain and for the leftmost *p-q* carry chain. The left column lists the positions and the right column the pairs of inputs allowed in the positions. Note that the sequence $q+1, \dots, n-1$ could be empty.

(we are multiplying the inner sum by -1 if it is negative, and by $+1$ if it is positive). Therefore (1) can be written as

$$\begin{aligned}\sum_{a,b} \left| \sum_{C \in \mathcal{C}(a,b)} \mathrm{EC}(C) \right| &= \sum_{a,b} \sum_{C \in \mathcal{C}(a,b)} \widehat{\mathrm{sgn}}(a,b) \cdot \mathrm{EC}(C) \qquad (6)\\ &= \sum_{C} \mathrm{EC}(C) \sum_{\sigma \in \{-1,+1\}} \sigma \cdot \mathrm{card}\left\{(a,b) \mid C \in \mathcal{C}(a,b) \wedge \widehat{\mathrm{sgn}}(a,b) = \sigma\right\}.\end{aligned}$$

Note that the number of terms to be summed is approximately n^2. So to compute (4) efficiently, it is enough to compute all the terms efficiently.

For every C it is easy to compute its $\mathrm{EC}(C)$ as this does not depend on the inputs, other than some very specific inputs. Indeed, if we have a carry chain from i to j, we just need to examine what happens for the two inputs of length $j-i+1$, namely $(1, \dots, 1, 1)$ and $(0, \dots, 0, 1)$. This will allow us to compute the error. What remains to be done is to account for $\widehat{\mathrm{sgn}}(a,b)$ for all inputs (a,b) in which the carry chain under consideration occurs.

So consider then a carry chain C from i to j. If there is no error inducing carry chain to its left for some (a,b) in which it occurs, then $\widehat{\mathrm{sgn}}(a,b) = \mathrm{sign}(\mathrm{EC}(C))$. This case is simpler to handle, so we will consider here the case where there are such chains, and let the leftmost carry chain be from p to q. For this situation, $n \geq q \geq p > j \geq i \geq 1$ and the conditions in Fig. 1 hold. The number of input configurations producing such carry chains is

1. If $q = n$ then $2^{n-p} 2^{j-i} 4^{(p-1)-(j-i+1)}$.
2. If $q < n$ then $3^{n-q+1} 2^{q-p} 2^{j-i} 4^{(p-1)-(j-i+1)}$.

Keeping i and j fixed, we consider all the possible values for p and q. For some inputs, the *i-j* carry chain is the leftmost error-inducing chain, and for some inputs there are p and q for which there is an error-inducing *p-q* carry chain and it is the leftmost such carry chain. By examining all the cases, we can easily compute

$$\sum_{\sigma \in \{-1,+1\}} \sigma \cdot \mathrm{card}\left\{ (a,b) \mid i\text{-}j \text{ carry chain} \in \mathcal{C}(a,b) \wedge \widehat{\mathrm{sgn}}(a,b) = \sigma \right\},$$

and from here, iterating over all applicable i and j, we comput the needed $\mathrm{Er}_{\mathrm{avg}}(T, \mathbf{D})$, using (2).

5 Example: Kogge-Stone adder

The Kogge-Stone adder (KSA) works in three seperate stages. In the first stage, the KSA computes information that represents whether a carry is generated or propagated at a certain bit position based on the inputs to the adder. In the second stage, it uses the information computed in the first stage to determine whether a set of consecutive positions generates a carry or propagates a carry. This can be computed fully in parallel. A complete description of these functions can be found in [3]. At the end of the second stage the carry inputs to all bit positions of the sum are known. In the third stage, the final sum bits are computed.

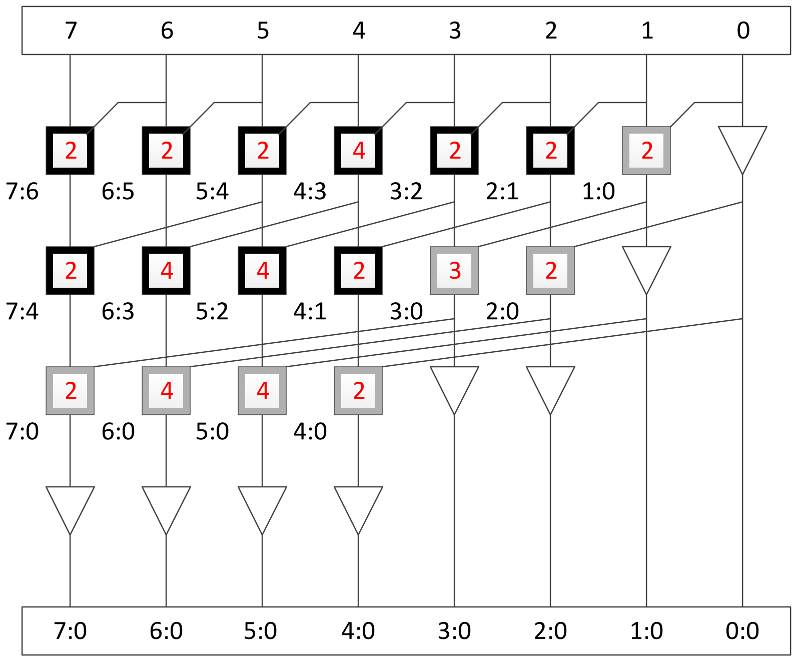

Figure 2: An 8-bit Kogge-Stone adder. The delays of the modules are written inside the vertices representing them.

The second stage is the most complex of all three. As we can see from Fig. 2, which we adapt from [4], the KSA decreases time taken for computation by computing the carry bits for each position almost in parallel. This creates the possibility that the carry bit reaches position $k + 1$ before it reaches position k given a certain allocation of propagation delays to the various blocks. We will note here that irrespective of the allocation of propagation delays in the ripple-carry adder that condition can never happen in its simple architecture.

The numbers in the modules in Fig. 2 are the delays introduced in these modules. In Fig. 3, we consider the computation of $s = a + b$, where $\mathbf{a} = (001010110)$ and $\mathbf{b} = (000111010)$. There are two carry chains, C_1 from position 2 to position 4 induced by position 1 and C_2 from position 5 to position 7 induced by position 4. The correct sum is $\mathbf{s} = (010010000)$, and is obtained at time instance of 10. Let us consider the case when $T = 7$, that is, the adder is overclocked and an approximation s^{A} is read instead of the true s.

Based on the topology of G and on the delays in Fig. 2, we compute $\mathbf{D}$, and from this the values of $\mathbf{s}^{\mathrm{A}}$ at the various instances of time. Note that sometimes the error is positive and sometimes negative. At time T, both of the carry chains induce errors. We compute the absolute error produced at time T:

1. C_2 the leftmost error-inducing carry chain and it contributes an error of -96, and therefore,

$$\widehat{\mathrm{sgn}}(a, b) = \mathrm{sgn}\left(\mathrm{EC}\big(C_{\mathrm{L}}(a, b)\big)\right) = -1.$$

2. C_1 is an error-inducing carry chain, but not the leftmost and it contributes an error of $+16$.

Therefore, at time $T = 7$, $\left|s - s^{\mathrm{A}}\right| = (-1) \cdot (-96) + (-1) \cdot (+16) = +80$.

6 Energy-error tradeoffs

This section briefly outlines an adaptation of the approach already presented in Sections IV–V of [2] for ripple-carry adders, and an interested reader should consult that paper and its references, to fill out details omitted here.

The energy consumption E of a circuit consisting of the dynamic and the static energy consumption can be

			Position									
			7	6	5	4	3	2	1	0		
a	$\longrightarrow$	0	0	1	0	1	0	1	1	0		
b	$\longrightarrow$	0	0	0	1	1	1	0	1	1		
Time											s^{A}	$s - s^{\mathrm{A}}$
0		0	0	0	0	0	0	0	0	0	0	+145
1		0	0	1	1	0	1	1	0	1	109	+36
2		0	0	1	1	0	1	1	0	1	109	+36
3		0	0	1	1	0	1	1	0	1	109	+36
4		0	0	1	1	0	1	0	0	1	105	+40
5		0	0	1	1	0	0	0	0	1	97	+48
6		0	0	1	1	0	0	0	0	1	97	+48
7		0	1	1	1	0	0	0	0	1	225	−80
8		0	1	1	1	1	0	0	0	1	241	−96
9		0	1	1	1	1	0	0	0	1	241	−96
10		0	1	0	0	1	0	0	0	1	145	0

Figure 3: An example of addition in an overclocked Kogge-Stone adder. Carry chains are highlighted at the instances of time in which they induce errors.

approximated as

$$\mathrm{E} \approx \sum_{\ell=1}^{N} \left(\frac{\gamma_\ell w_\ell}{\epsilon_\ell^2(\nu_\ell)} + \mathrm{P}_\ell(\nu_\ell) T \right),$$

where N is the number of gates, ℓ varies over gate number, γ_ℓ is a constant, ν_ℓ is the voltage supplied to gate ℓ, $\epsilon_\ell(\nu_\ell)$ is the worst-case propagation delay of gate ℓ when supplied with voltage ν_ℓ, w_ℓ is the average switching activity of gate ℓ during time $[0, T]$, and $\mathrm{P}_\ell(\nu_\ell)$ is the static power consumption of gate ℓ during time $[0, T]$.

The elements of **D** can be expressed using ϵ_ℓ, for $\ell = 1, \ldots, N$, and it is also convenient to formulate the resulting optimization problem as a geometric program [5]. To that end, the errors in a chain are approximated by a special form, so that ultimately the optimization problem to be solved can be expressed as

$$\begin{array}{rl} \text{minimize} & \mathrm{Er}_{\mathrm{avg}}(T, \boldsymbol{\epsilon}) = \displaystyle\sum_{C} k_C \prod_{\ell=1}^{N} \epsilon_\ell^{a_{C,\ell}} \\ \text{subject to} & \text{min-delay}_\ell \ \leq \epsilon_k \leq \ \text{max-delay}_\ell, \quad \ell = 1, \ldots, N \\ \text{and} & \displaystyle\sum_{\ell=1}^{N} \left(\frac{\gamma_\ell w_\ell}{\epsilon_\ell^2} + \mathrm{P}_\ell(\nu_\ell) T \right) \leq \ \text{Energy Budget}, \end{array}$$

with appropriate constants k_C's and $a_{C,\ell}$'s. Here k_C's are the coefficients of $\mathrm{EC}(C)$'s in the second line of (6) for each respective carry chain.

Please see [2] for the procedure to solve this optimization problem and for a simulation approach to evaluate the solution using HSPICE.

Acknowledgement

We thank Michael Overton for his suggestion to use geometric programming to solve our original optimization problem.

References

[1] I. Koren, *Computer Arithmetic Algorithms*, 2nd ed. AK Peters, Ltd., 2002.

[2] Z. M. Kedem, V. J. Mooney, K. K. Muntimadugu, and K. V. Palem, "An approach to energy-error tradeoffs in approximate ripple carry adders," in *Proceedings of the International Symposium on Low Power Electronics and Design*, 2011, pp. 211–216.

[3] N. H. E. West and D. M. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, 4th ed. Addison-Wesley, 2010.

[4] D. Harris, "Introduction to CMOS VLSI design. Lecture 11: Adders," http://www.cmosvlsi.com/lect11.pdf, accessed: 2013-03-07.

[5] S. Boyd, S. J. Kim, L. Vandenberghe, and A. Hassibi, "A tutorial on geometric programming," *Optimization and Engineering*, vol. 8, no. 1, pp. 67–127, 2007.