1-1-1990

Faster Circuits and Shorter Formulas for Multiple Addition, Multiplication and Symmetric Boolean Functions

Michael Paterson
Warwick university

Uri Zwick
Warwick University

Nicholas Pippenger
Harvey Mudd College

Faster circuits and shorter formulae for multiple addition, multiplication and symmetric Boolean functions

Michael S. Paterson [*] *Nicholas Pippenger* [†] *Uri Zwick* [‡]

Extended Abstract

Abstract

A general theory is developed for constructing the shallowest possible circuits and the shortest possible formulae for the carry save addition of n numbers using any given basic addition unit. (A carry save addition produces two numbers whose sum is equal to the sum of the n input numbers).

More precisely, it is shown that if BA is a basic addition unit with occurrence matrix N then the shortest multiple carry save addition formulae that could be obtained by composing BA units are of size $n^{1/p+o(1)}$ where p is the unique real number for which $\|N\|_p = 1$. (Here $\|N\|_p$ is the usual L_p norm of the matrix N). An analogous result connects the delay matrix M of the basic addition unit BA and the minimal q such that multiple carry save addition circuits of depth $(q + o(1)) \log n$ could be constructed by combining BA units.

Based on these optimal constructions of multiple carry save adders we construct the shallowest known multiplication circuits. The depth of the obtained $n \times n$ bit multiplication circuit, which uses only dyadic gates, is $3.71 \log n$. As for carry save addition, the result of the multiplication is given as a sum of two numbers. This construction improves previous results of Ofman, Wallace, Khrapchenko and others.

Multiple carry save adders can also be used to construct formulae for symmetric Boolean functions. Here we are able to construct Boolean formulae of size $O(n^{3.16})$ for many symmetric Boolean functions, including the majority function and Boolean formulae of size $O(n^{3.30})$ for all symmetric Boolean functions. This improves previous results of Khrapchenko, Pippenger, Paterson and Peterson.

[*]Department of Computer Science, University of Warwick, Coventry, CV4 7AL, England. This author was partially supported by a Senior Fellowship from the SERC and by the ESPRIT II BRA Programme of the EC under contract # 3075 (ALCOM).

[†]Department of Computer Science, University of British Columbia, Vancouver, British Columbia, Canada V6T 1W5. This author was partially supported by an NSERC operating grant and an ASI fellowship award.

[‡]Department of Computer Science, University of Warwick, Coventry, CV4 7AL, England. This author was partially supported by a Joseph and Jeanne Nissim postdoctoral grant from Tel-Aviv University.

1. Introduction

The question 'How fast can we multiply ?' is one of the most fundamental questions in computational arithmetic. One approach is to try to minimise the number of bit operations involved, or equivalently the circuit size. Examples of such work are the algorithms of Ofman-Karatsuba [6] and Schönhage-Strassen [19] (see also [2],[12]).

While this answers (at least partially) the question in the context of a sequential computation model, it fails to address the question in the often more appropriate parallel model. To that end one should investigate the depth, rather than the size of multiplication circuits.

This question was already answered, qualitatively, by Avizienis [1], Dadda [4] , Ofman [14], Wallace [21] and others in the early 1960's by showing that using redundant number representations, n numbers of arbitrary size can be added in depth $O(\log n)$. They all used a component called a $3 \rightarrow 2$ *Carry Save Adder (CSA)* which reduces the sum of 3 numbers to the sum of only 2 numbers in a (small) constant depth. It is easy to see that using $\log_{3/2} n + O(1)$ levels of such *CSA*'s it is possible to reduce the sum of n numbers to the sum of only 2. This operation will henceforth be called ***multiple carry save addition***. An $n \times n$ bit multiplication can be solved by performing multiple carry save addition of n numbers of length less than $2n$. The resulting two numbers are in this case of length at most $2n$ and they can be added (if required) using a Carry Look Ahead adder with additional depth $(1+o(1))\log n$ (see [3],[7])[1].

More detailed considerations can be used to show that the Schönhage-Strassen algorithm can also be implemented in depth $O(\log n)$ (cf. [20]). Mehlhorn and Preparata [13] presented an optimal area-time VLSI implementation of this algorithm.

Since multiplication is so fundamental, it is interesting to investigate the constants implied by the $O(\log n)$ notations above. Admittedly, constant factors are usually ignored in theoretical investigations. We think however, that in this case the investigation is justified by the importance of the problem and the rich theory which will be revealed in the process.

The Schönhage-Strassen multiplication algorithm uses the Discrete Fourier Transform (DFT) to reduce dramatically the size of the circuits. Since the computation of a DFT essentially involves multiple additions, it seems that it cannot be used to reduce the circuit depth. The best approach known today for constructing shallow multiplication circuits is to consider multiplication as a special case of multiple addition. We will pursue this direction and investigate the exact depth needed to perform multiple addition or, in fact, multiple carry save addition.

We will present U_2-circuits (i.e., circuits over the basis U_2 of all unate dyadic Boolean functions) of depth $5.07\log n+O(1)$ and B_2-circuits (i.e., circuits over the basis B_2 of all dyadic Boolean functions) of depth $3.71\log n + O(1)$ for the multiple carry save addition of n numbers of arbitrary size. This improves a previous result of Khrapchenko [11] and the naive estimates of Ofman and Wallace.

Our constructions can again be described in terms of Carry Save Adders (CSA's). Each CSA is built using an array of Full Adders (FA's). Though we cannot claim that our circuits are U_2- or B_2-optimal, we will prove that they are the optimal circuits that could be constructed using 3 bit FA's (in the B_2 case) or using the best known 7 bit FA's (in the U_2 case).

Our theory is more general however. To any basic addition unit BA (3-bit and 7-bit FA's are just special cases) we attach a *delay matrix* which describes the relative delays of the outputs of the BA unit with respect to its inputs. If the addition unit is implemented using U_2- or B_2-circuits then the entries of the matrix are the depths of the input variables with respect to the outputs. Alternatively, the delay matrix may describe characteristics of a basic adder regarded as a 'black box'. We then show how to extract from the delay matrix the minimal constant q such that depth $(q+o(1))\log n$ circuits for the carry save addition of n numbers could be constructed using BA units. Our proofs of the upper bounds are constructive. We explicitly construct in each case circuits with optimal behavior.

An analogous theory is developed for formula size. It is shown that the shortest formulae for the carry save addition of n numbers that can be obtained using a basic adder BA with *occurrence matrix* N are of size $n^{1/p+o(1)}$ where p is the unique real number satisfying $\|N\|_p = 1$ where $\|N\|_p$ is the usual L_p norm of the matrix N. The occurrence matrix gives for each input variable of BA the number of times it appears in the formula of each output. Again we present explicit constructions achieving these bounds.

Using this general theory we construct U_2-formulae of size $O(n^{4.60})$ and B_2-formulae of size $O(n^{3.16})$ for each output bit in the carry save addition of n numbers, in an $n\times n$ bit multiplication and for many symmetric Boolean functions including the majority of n bits. We also construct U_2-formulae of size $O(n^{4.85})$ and B_2-formulae of size $O(n^{3.30})$ for all symmetric Boolean functions of n variables. These constructions improve previous results of Khrapchenko [10], Pippenger [17], Paterson [15] and Peterson [16]. Notice that the improved constants are this time in the exponent.

A summary of the 'numerical' results obtained for multiple carry save addition together with the previously known results is given in table 1.1. The improvements we get are quite marginal in some of the cases. Our results, however, could only be improved by either designing improved addition units or by designing circuits or formulae which are not based on a fixed underlying addition unit or, which use the existing units in a non-arithmetical way. Moreover, whenever a new addition unit is designed, our theory immediately describes the

[1] Unless otherwise stated, all the logarithms are taken to base 2.

	U_2-depth		
$5.12 \log n$	Khrapchenko	(1978)	FA_7
$5.07 \log n$	*	(1990)	FA_7 (opt.)
	B_2-depth		
$3.71 \log n$	*	(1990)	FA_7 (opt.)
	U_2-formula size		
n^5	Pippenger	(1974)	FA_3
$n^{4.62}$	Khrapchenko	(1972)	FA_7
$n^{4.60}$	*	(1990)	FA_7 (opt.)
	B_2-formula size		
$n^{3.54}$	Pippenger	(1974)	FA_3
$n^{3.47}$	Paterson	(1978)	FA_3
$n^{3.32}$	Peterson	(1978)	FA_3
$n^{3.21}$	*	(1990)	FA_3 (opt.)
$n^{3.16}$	*	(1990)	FA_7 (opt.)

Table 1.1. Results for multiple carry save addition.

way in which optimal use could be made of it.

In the next section we present an exact formulation of our main results. In section 3 we show how our general theory produces some of the 'numerical' results mentioned. Due to lack of space we will not be able to present here complete proofs of the results. For these the reader is referred to the full version of the paper [18]. Section 4 introduces the basic ideas used in the lower bounds proofs. Section 5 demonstrates the general construction methods used. Additional constructions are described in section 6. These constructions are used in section 7 to compute general symmetric Boolean functions. An interesting open problem is raised in the last section.

2. Statement of Results

A *basic adder* (or a *bit adder*) BA is a black box with k input bits $x_1, \ldots, x_k$ and $\ell < k$ output bits for which the condition $\sum y_i 2^{b_i} = \sum x_j 2^{a_j}$ is always satisfied. The numbers $a_1, \ldots, a_k$ and $b_1, \ldots, b_\ell$ are termed the *significances* of the different inputs and outputs.

An acyclic interconnection of BA 'gates' is called a BA-circuit. A BA-circuit that receives the binary representation of n numbers and produces at most ℓ numbers whose sum is equal to the sum of the n input numbers is called an $n \to \ell$ *Carry Save Adder*, or in short $CSA(n \to \ell)$. The length of the input numbers will not affect the depth of the circuits and the size of the formulae for $CSA(n \to \ell)$. We will therefore ignore it in the sequel. In fact, we may assume (until section 6) that the input numbers extend to infinity on both sides.

A BA-circuit is said to be *arithmetical* if it is possible to assign significances to all the 'wires' and the 'gates' in the circuit such that the k wires entering a BA gate with significance s have significances $s + a_1, \ldots, s + a_k$ and the ℓ wires leaving this gate have significances $s+b_1, \ldots, s+b_\ell$.

In this work we are interested in constructing BA-circuits for multiple carry save addition with minimal delay or with minimal formula size.

We assume that a *delay matrix* M is attached to the bit adder BA. The entry m_{ij} in this matrix specifies the relative delay of the i-th output bit of BA with respect to its j-th input bit.

The *delay* of each wire in a BA-circuit is defined in the following way. The inputs to the circuit are defined to have delay 0. If the k inputs to a bit adder BA have delays $x_1, \ldots, x_k$ then the i-th output bit of this BA will be assigned delay $y_i = \max_{1 \le j \le k}\{m_{ij} + x_j\}$. We will express this by $y = M \diamond x$ where the $\diamond$ denotes the $\{\max, +\}$ inner product.

If a bit adder BA is implemented using an Ω-circuit (here $\Omega = B_2$ or U_2) and the entry m_{ij} of the delay matrix M is set to be the length of the longest path from the input x_j to the output y_i in this implementation, then the abstract delay of a wire defined above is exactly the depth of this wire in the Ω-circuit obtained from the BA-circuit by replacing every copy of BA by its Ω implementation.

Similarly we can attach to every bit adder BA an *occurrence matrix* N which will enable us to define the *formula size* of every wire in a BA-

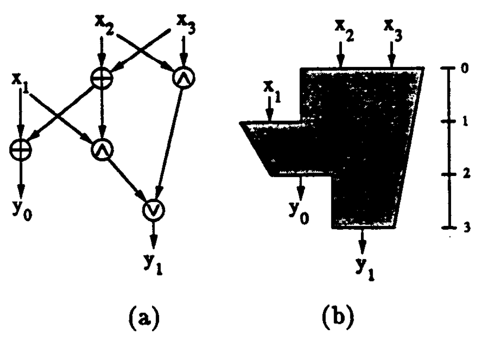

Figure 2.1. An optimal implementation of an FA_3 by a B_2-circuit.

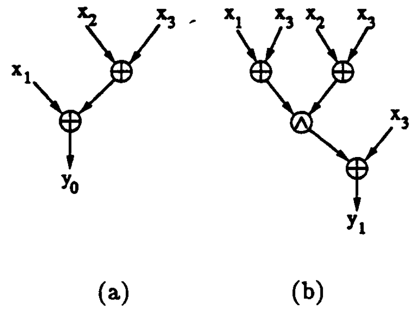

Figure 2.2. An optimal implementation of an FA_3 by B_2-formulae.

circuit. The inputs to a circuit are defined to have formula size 1. If the k inputs to a bit adder BA have sizes $x_1, \ldots, x_k$ then the i-th output bit of BA will be assigned formula size $y_i = \sum_{j=1}^{k} n_{ij} x_j$. We will abbreviate this by writing $y = Nx$ where this time the usual $\{+, \times\}$ inner product is used.

If we are given a set of ℓ Ω-formulae in the variables $x_1, \ldots, x_k$, one for each output bit y_i of the bit adder BA, then by composing these formulae we can obtain for every wire in a BA-circuit an equivalent Ω-formula. If the occurrence matrix N is defined by letting n_{ij} be the number of occurrences of the variable x_j in the basic formula for y_i then, the abstract formula sizes defined above coincide with the usual definition of formula size as the number of variables appearing in the formula.

A B_2-implementation of a *3-bit Full Adder* (FA_3) is given in Fig. 2.1(a). It is easy to check that the delay matrix of this FA_3 is $\begin{pmatrix} 1 & 2 & 2 \\ 2 & 3 & 3 \end{pmatrix}$. Notice that x_1 may be supplied to this unit one unit of time after x_2 and x_3 are supplied and that y_0 is obtained one unit of time before y_1. Thus, the FA_3 can be described schematically by the 'gadget' appearing in Fig. 2.1(b). This implementation will be used in our $3.71 \log n$ depth $CSA(n \to 2)$ circuits.

An FA_3 can also be implemented using the two formulae shown in Fig. 2.2. The corresponding occurrence matrix of this implementation is $\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 3 \end{pmatrix}$. Paterson [15] and Peterson [16] both used this implementation to obtain their $O(n^{3.47})$ and $O(n^{3.32})$ results. By using this FA_3 implementation in an optimal way we can reduce the size of the obtained formulae to $O(n^{3.21})$. We can improve this bound to $O(n^{3.16})$ using a novel design of a *7-bit Full Adder* (FA_7) with occurrence matrix $\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 3 & 4 \\ 2 & 2 & 2 & 4 & 4 & 4 & 9 \end{pmatrix}$.

Our best U_2-results are obtained using an implementation of an FA_7 due to Khrapchenko [10],[11].

We denote by $\delta(BA) = \delta(M)$ the minimal constant δ such that arithmetical BA-circuits of depth $(\delta + o(1)) \log n$ for $CSA(n \to \ell)$ could be constructed. It is unlikely that better results could be obtained using non-arithmetical BA-circuits but we were not able to prove it.

Similarly, we denote by $\epsilon(BA)$ the minimal exponent ϵ such that arithmetical BA-circuits with attached formula size $n^{\epsilon + o(1)}$ for $CSA(n \to \ell)$ could be constructed.

After these preparations we can state our general results :

Theorem 2.1 (Optimal depth and size)

(i) $\delta(M) = \max\left\{ \delta : \begin{array}{c} \forall x \in R^k \text{ and } y = M \diamond x \\ \sum_{j=1}^{k} 2^{x_j/\delta} \le \sum_{i=1}^{\ell} 2^{y_i/\delta} \end{array} \right\}$

(ii) $\epsilon(N) = \max\left\{ \epsilon : \begin{array}{c} \forall x \in (R^+)^k \\ \|x\|_{1/\epsilon} \le \|Nx\|_{1/\epsilon} \end{array} \right\}$.

In the formula size case we can state the result more concisely, $\epsilon(N) = 1/p$ where p is the unique real number satisfying $\|N\|_p = 1$.

Using this formulation we can find $\epsilon(N)$ for every matrix N using numerical methods. The components of the (unique) vector $x \in (R^+)^k$ satisfying

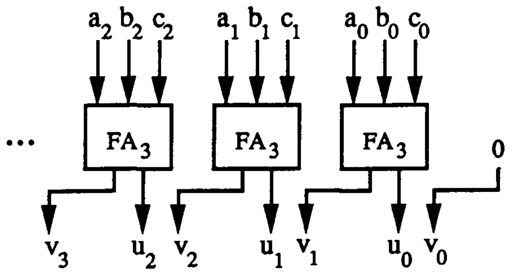

Figure 2.3. Constructing a *CSA*($3 \to 2$) using *FA*$_3$'3.

$x_1 = 1$ and $\|x\|_p = \|Nx\|_p$ will be important design parameters in the constructions achieving the optimal exponent. They will represent the ratios between the sizes of the different subformulae feeding the same adder.

The delay case is somewhat more complicated. If the matrix M is of the form $M = \beta - \alpha^T$ where $\alpha \in R^k, \beta \in R^\ell$ (that is $m_{ij} = \beta_i - \alpha_j$), in which case we say that it is *modular*, then it is easy to verify that $\delta(M) = 1/\log\lambda(M)$ where $\lambda = \lambda(M)$ is the unique positive root of the equation

$$\sum_{j=1}^{k} \lambda^{\alpha_j} = \sum_{i=1}^{\ell} \lambda^{\beta_i} \quad .$$

If M is an arbitrary matrix, denote by $P(M)$ the set of all modular matrices dominating it, that is $M' \in P(M)$ iff M' is modular and $m'_{ij} \geq m_{ij}$ for all i, j. This set will be called the *modular polyhedron over* M. It can be checked that the set of vertices of $P(M)$, denoted by $P^*(M)$ is finite and that $\delta(M) = 1/\log\lambda(M)$ where

$$\lambda(M) = \max_{M' \in P^*(M)} \lambda(M') \quad .$$

Therefore $\lambda(M)$ can be computed by solving a finite number of (polynomial) equations.

An array of bit adders BA could be used in order to construct a $CSA(k \to \ell)$ which we will denote by CSA_{BA}. Fig. 2.3 shows, for example, how a $CSA(3 \to 2)$ could be constructed using an array of FA_3's.

A *CSA-network* is an acyclic network composed of a fixed type of *CSA* units. The CSA's are now considered to be the basic 'gates' and each 'wire' now corresponds to a number and not just to a bit. This enables us to consider the constructions from a higher level viewpoint. Each *CSA* may in turn be implemented by an array of bit adders. Clearly, any CSA_{BA} network could be expanded into an arithmetical *BA*-circuit (which in turn might be expanded into Ω-circuits or formulae).

The delay matrix and the occurrence matrix associated with a bit adder *BA* could also be used to describe the characteristics of the carry save adder CSA_{BA}. Using this convention we can define the delay and formula size of every wire in a CSA_{BA}-network. A wire in a CSA_{BA}-network carries a number and corresponds to a set of wires, each carrying a bit, in the equivalent *BA*-circuit. The delay (or formula size) of a number in a CSA_{BA}-network is an upper bound on the delays (or formula sizes) of the bits in the *BA*-circuit which are part of its representation.

Working with individual *BA*'s may seem to give more power than working with CSA_{BA}'s but we can show that the optimal performances of arithmetical *BA*-circuits could in fact be obtained using CSA_{BA}-networks. For the proof the reader is referred again to [18].

3. Examples

The size of the optimal formulae for multiple carry save addition that can be obtained using the FA_3 described in Fig. 2.2 is $n^{\epsilon+o(1)}$ where according to Theorem 2.1

$$\epsilon = \min\left\{\frac{1}{p} : \begin{array}{l}\forall x_1, x_2, x_3 \geq 0, x_1^p + x_2^p + x_3^p \leq \\ (x_1 + x_2 + x_3)^p + (x_1 + x_2 + 3x_3)^p\end{array}\right\}$$

A numerical solution gives $\epsilon \simeq 3.2058$ and equality is achieved when $x_1 = x_2 = 1$, $x_3 = 0.3926$. This yields formulae of size $O(n^{3.21})$. As was mentioned before we can get better results using FA_7's.

The delay matrix of the FA_3 described in Fig. 2.1 is the modular matrix $M = (2\ 3)^T - (1\ 0\ 0)$. Therefore $\lambda(FA_3)$ is the unique positive root of the cubic equation $\lambda^3 + \lambda^2 - \lambda - 2 = 0$. Numerically we can find that $\lambda \simeq 1.2056$ and that $\log_\lambda n \simeq 3.71 \log n$.

The delay matrix of the Khrapchenko's FA_7 [11] is the non-modular matrix $\begin{pmatrix} 4 & 6 & 6 & 6 & 6 & 6 & 6 \\ 5 & 6 & 6 & 7 & 7 & 7 & 7 \\ 5 & 6 & 6 & 6 & 6 & 6 & 6 \end{pmatrix}$.

The optimal vertex in the modular polyhedron of this matrix turns out to be $\begin{pmatrix} 5 & 6 & 6 & 6 & 6 & 6 & 6 \\ 6 & 7 & 7 & 7 & 7 & 7 & 7 \\ 5 & 6 & 6 & 6 & 6 & 6 & 6 \end{pmatrix}$ and therefore $\lambda(FA_7)$ is the unique positive root of the equation $\lambda^7 + 2\lambda^6 - \lambda - 6 = 0$. Numerically we can find that $\lambda \simeq 1.1465$ and that $\log_\lambda n \simeq 5.07 \log n$.

4. Lower Bounds

Theorem 4.1 *Let BA be a basic addition unit. If Net is a CSA_{BA}-network with n inputs and ℓ outputs then depth(Net) $\geq \log_{\lambda(BA)}(n/\ell)$.*

Proof : Consider a CSA_{BA}-network with n inputs and ℓ outputs. If the the inputs to a CSA_{BA} in the network have delays $x_1, \ldots, x_k$ then the outputs of this unit have delays $y_1, \ldots, y_\ell$ where $y = M \diamond x$. The definition of $\lambda = \lambda(BA)$ ensures that $\sum \lambda^{x_j} \leq \sum \lambda^{y_i}$. Using induction we get a similar relation for the whole network. The n inputs have delay 0. If the ℓ outputs all have delay at most d then we get that $n \leq \ell\lambda^d$ or equivalently $d \geq \log_\lambda(n/\ell)$. □

The argument used here is similar to a proof that a binary tree can have at most 2^ℓ leaves at depth at most ℓ using Kraft's inequality ($\sum 2^{-\ell_i} \leq 1$ where the ℓ_i's are the depths of the leaves in a binary tree).

As was mentioned at the end of section 2, a simple argument could be used to show that the above result holds also for general arithmetical BA-circuits.

The proof of the formula size lower bound is similar.

5. Basic depth constructions

We demonstrate the general technique by constructing the shallowest possible FA_3-circuits for multiple carry save addition. As we have stated earlier their depth is about $\simeq 3.71 \log n$. We will use the implementation of Fig. 2.1. It could be shown that no B_2-implementation of an FA_3 will produce shallower results. We have not been able to improve this result using more complicated adders.

Our construction uses $CSA{=}CSA_{FA_3}$ units. We want to connect the CSA's in such a way

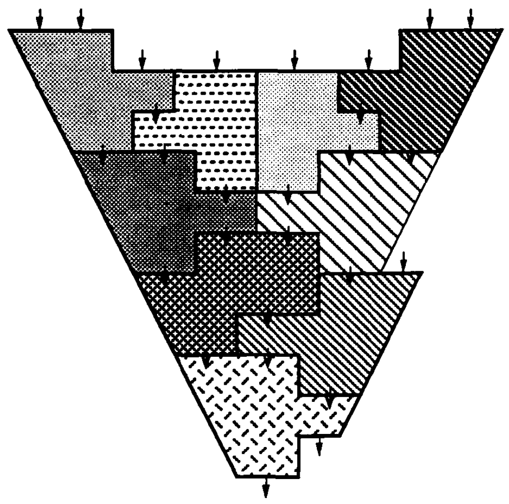

Figure 5.1. The final stages of a 'good' *CSA*-network

that on most wires no delays will be introduced. Consulting the delay matrix of the *CSA* (or Fig. 2.1(b)) we realize that to achieve that we should feed each *CSA* by two numbers that are ready at some time d and by a third number which is ready at time $d+1$. The two output numbers will then be ready at times $d+2$ and $d+3$. Such a *CSA* will be said to lie at *level d*.

Fig. 5.1 gives an impression of how the final stages of a 'good' construction might look like (the first stages are usually non-planar however).

We specify a *CSA*-network by specifying the number A_d of level d *CSA*'s used in it. The number of numbers in the network produced at time d is $Gen_d = A_{d-2} + A_{d-3}$. The number of numbers in the network consumed at time d is $Con_d = 2A_d + A_{d-1}$.

We choose $A_d = \lceil n\lambda^{-d} + 1 \rceil$ if $d \geq 0$ and $n\lambda^{-d} \geq 1$, and $A_d = 0$ otherwise, where λ is the unique positive root of the equation $\lambda^3 + \lambda^2 - \lambda - 2 = 0$ ($\lambda \simeq 1.2056$).

If $Bal_d = Con_d - Gen_d > 0$ then the network can accept Bal_d new inputs at time d. If $Bal_d < 0$ then the network yields $|Bal_d|$ outputs at time d.

It can be checked that $Bal_0 \geq n$, $Bal_d \geq 0$

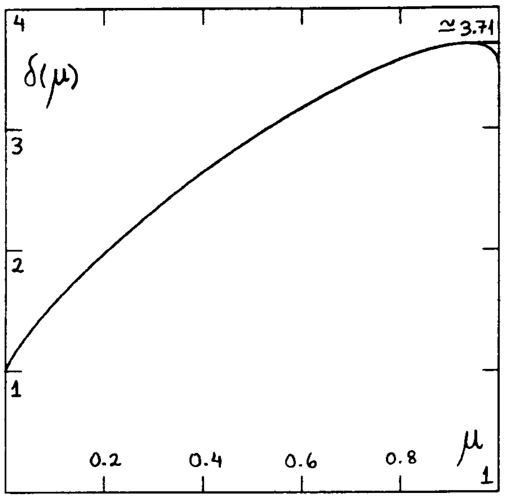

Figure 6.1. The function $\delta(\mu)$.

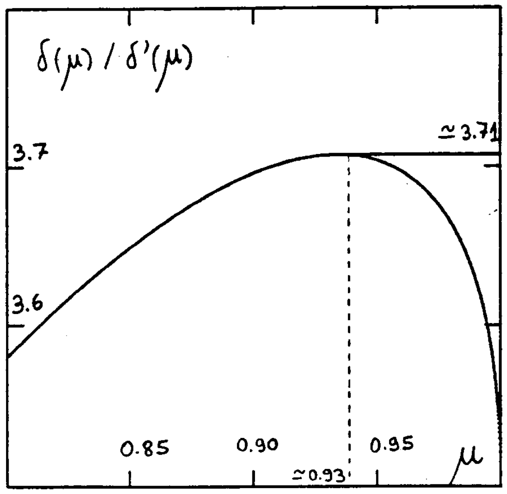

Figure 6.2. An enlarged graph of $\delta(\mu)$ and $\delta'(\mu)$.

for $1 \leq d \leq \ell$ and $Bal_{\ell+1}, Bal_{\ell+2}, Bal_{\ell+3} \geq -6$ where $\ell = \lfloor \log_\lambda n \rfloor$. We can therefore input at least n numbers at time 0 and we get at most 18 outputs at or before time $\ell + 3$. The sum of these 18 numbers can be reduced to only 2 using a small number of additional *CSA*'s. The depth of the whole network will therefore be $\ell + O(1) = \log_\lambda n + O(1) \simeq 3.71 \log n$.

6. Additional depth constructions

The lower bounds claimed, and partially proved in section 4, apply only to the maximal delay (or formula size) in a *BA*-circuit for carry save addition. Could we get the less significant output bits faster ?

When a *CSA*-network is designed, as was done in the previous section, the length of the input numbers is not taken into account. A uniform bound is therefore obtained for all the output bits. Though we cannot reduce the maximal delay (or formula size) by using individual *BA* units, we can use them to get the less significant bits much faster.

For every $1.2056 \simeq \lambda(FA_3) \leq \lambda < 2$ and $\nu = \frac{\lambda^3}{2+\lambda-\lambda^2}$ we can build an FA_3-circuit for $CSA(n \to 2)$ in which the B_2-depth of the two output bits with significance $i = \mu \log n$ is $(1 + \mu \log \nu) \log_\lambda n + O(1)$. Each such construction corresponds to a tangent, with a positive slope, to the function

$$\delta(\mu) = \min\Big\{\frac{1 + \mu \log \nu}{\log \lambda} : \lambda(FA) \leq \lambda < 2\Big\}.$$

A graph of the function $\delta(\mu)$ is given in Fig. 6.1. For $\lambda = \lambda(FA_3)$ we get that $\nu = 1$ and the construction coincides with the construction obtained from the CSA_{FA_3}-network described in section 5. It is interesting to note that the function $\delta(\mu)$ assumes its maximal value $\delta(FA_3)$ for every $\mu \geq \mu^*$ where $\mu^* \simeq 0.93$.

The FA_3 construction with parameters λ and $\nu = \frac{\lambda^3}{2+\lambda-\lambda^2}$ is obtained by using $A_{s,d} = \lceil n\nu^s \lambda^{-d} + 1 \rceil$ FA_3's with significance s which lie at depth d if $n\nu^s\lambda^{-d} \geq 1$ and $s, d \geq 0$, and $A_{s,d} = 0$ FA_3's otherwise. It can be checked that the circuit can receive at least $n\nu^s$ input bits with significance s for every $s \geq 0$ and therefore a $CSA(n \to 2)$ construction is obtained whenever $\nu \geq 1$.

Bit counting is a special case of multiple addition. A bit counting circuit receives n input bits and outputs their sum. If we choose $1 < \lambda < \lambda(FA_3)$ in the above construction then $\nu < 1$ and we do not get a carry save adder but we do get a bit counter (or more precisely, a carry save counter). Such a construction corresponds to a tangent (with a negative slope) to the function $\delta'(\mu)$ obtained by allowing λ in the definition of $\delta(\mu)$ to lie in the range $1 < \lambda < 2$. An enlarged graph of the functions $\delta(\mu)$ and $\delta'(\mu)$ is given in Fig. 6.2. The functions $\delta(\mu)$ and $\delta'(\mu)$ agree when $\mu \leq \mu^*$. For $\mu > \mu^*$ the function $\delta(\mu)$ is constant while $\delta'(\mu)$ is decreasing. Note that the constructions with $\nu < 1$ have the peculiar property of producing the more significant output bits before the less significant ones.

We were not able to construct a single FA_3-circuit for $CSA(n \rightarrow 2)$ which outputs the two output bits with significance $i = \mu \log n$ at time $(\delta(\mu) + o(1)) \log n$ or a bit counting circuit which outputs the two bits with significance $i = \mu \log n$ at time $(\delta'(\mu) + o(1)) \log n$. It is interesting to know whether such constructions are possible.

We can get a B_2-circuit for $CSA(n \rightarrow 2)$whose behaviour is described by the function $\delta(\mu)$ by using a separate FA_3-circuit for each significance $s < \mu^* \log n$ and a common FA_3-circuit for all the significances $s \geq \mu^* \log n$. Note that the result of a carry save addition is not unique (any number can be written as the sum of two other numbers in more than one way) and therefore taking the two output bits of each significance from a different construction will usually produce an incorrect result. We can overcome this difficulty by adding the initial segments of length $\mu^* \log n$ of the two output numbers of each construction. This will increase the depth of the outputs by only $O(\log \log n)$. The results are now unique and we can safely take each bit from a different construction. We relied here on the fact that in each construction with $\nu > 1$ the less significant output bits are produced before the more significant ones. We therefore cannot use the same method to construct a bit counting circuit whose behaviour is described by the function $\delta'(\mu)$ for $0 \leq \mu \leq 1$.

The above discussion suggests the following question. Is the most significant bit (ms-bit) in the output of a bit count the hardest to compute ? Notice that for $n = 2^m - 1$ this ms-bit in simply the majority of the n input bits.

Similar constructions could be used to optimise the formula size of the individual output bits and similar questions could be raised.

7. Symmetric functions

A symmetric Boolean function of n bits could be computed by first counting the number of 1's in the input and then computing a function of the $\lceil \log(n+1) \rceil$ bit result. If the function of these $\log n$ bits can be computed in depth $o(\log n)$ then this additional stage will not effect the asymptotic behaviour of the obtained circuits of formulae.

This is the case for a large family of interesting symmetric Boolean functions such as threshold functions (including majority), Mod_k counting functions and others.

Any Boolean function of $m = \lceil \log(n+1) \rceil$ bits could be computed at depth m. In fact, any such function could be computed in depth $(1 + o(1))m$ even if the i-th input bit is supplied only at depth i. Similarly, for every function of m variables there exists a formula computing it in which the i-th variable appears at most 2^i times. We use these observations to construct circuits and formulae for general symmetric Boolean functions.

In order to get a shallow B_2-circuit, for example, for a general symmetric Boolean function we choose the carry save adder construction with slope 1. This corresponds to choosing $\lambda = \nu = \frac{1+\sqrt{17}}{4} \simeq 1.2808$ and the depth of the obtained construction is about $3.81 \log n$.

Using similar methods we get the $O(n^{3.30})$ B_2-formulae and $O(n^{4.85})$ U_2-formulae for general symmetric Boolean functions.

8. Concluding remarks.

An interesting (and still open) question is whether the optimal B_2- or U_2-circuits and formulae for multiple carry save addition could be obtained using fixed basic addition units. At least in the U_2 case we have reason to believe that the answer is 'no'. Unless very special conditions are met we can construct from every basic addition unit another unit which is slightly better.

References

[1] Avizienis A., *Signed-digit number representation for fast parallel arithmetic. IEEE Trans. Elec. Comp. Vol. EC10 (1961), pp. 389-400.*

[2] Aho A.V., Hopcroft J.E., Ullman J.D., *The design and analysis of computer algorithms. Addison-Wesley, 1974.*

[3] Brent R., *On the addition of binary numbers. IEEE Trans. on Comp., Vol. C-19 (1970), pp. 758-759.*

[4] Dadda L., *Some schemes for parallel multipliers. Alta Frequenza, Vol. 34 (1965), pp. 343-356.*

[5] Dunne P.E., *The complexity of Boolean networks. Academic Press, 1988.*

[6] Karatsuba A., Ofman Y., *Multiplication of multidigit numbers on automata. Soviet Physics Dokl., Vol. 7 (1963), pp. 595-596.*

[7] Khrapchenko V.M., *Asymptotic estimation of addition time of a parallel adder. Problemy Kibernet., Vol. 19 (1967), pp. 107-122* (in Russian). English translation in *Syst. Theory Res., Vol. 19 (1970), pp. 105-122.*

[8] Khrapchenko V.M., *Complexity of the realization of a linear function in the class of π-circuits. Mat. Zametki, Vol. 9 (1971), pp. 35-40* (in Russian). English translation in *Math. Notes Acad. Sciences USSR, Vol. 9 (1971), pp. 21-23.*

[9] Khrapchenko V.M., *Methods of determining lower bounds for the complexity of π-schemes. Mat. Zametki, Vol. 10 (1972), pp. 83-92* (in Russian). *Math. Notes Acad. Sciences USSR, Vol. 10 (1972), pp. 474-479* (English translation).

[10] Khrapchenko V.M., *The complexity of the realization of symmetrical functions by formulae. Mat. Zametki Vol. 11 (1972) pp. 109-120* (in Russian). English translation in *Mathematical Notes of the Academy of Sciences of the USSR Vol. 11 (1972) pp. 70-76.*

[11] Khrapchenko V.M., *Some Bounds for the Time of Multiplication. Problemy Kibernet., Vol. 33 (1978), pp. 221-227* (in Russian).

[12] Knuth D.E., *The art of computer programming, Vol. 1 (second edition). Addison-Wesley.*

[13] Mehlhorn K., Preparata F.P., *Area-Time Optimal VLSI Integer Multiplier with minimum computation time. Information and Control, Vol. 58 (1983), pp. 137-156.*

[14] Ofman Y., *On the algorithmic complexity of discrete functions. Doklady Akademii Nauk SSSR, 145 pp. 48-51* (in Russian). English translation in *Sov. Phys. Doklady, Vol. 7 (1963) pp. 589-591.*

[15] Paterson M.S., *New bounds on formula size. Proc. of 3rd GI conference on Theoretical Computer Science 1977, Lecture Notes in Computer Science 48, Springer-Verlag 1977, pp. 17-26.*

[16] Peterson G.L., *An Upper Bound on the size of formulae for symmetric Boolean functions. Technical Report No. 78-03-01, University of Washington.*

[17] Pippenger N., *Short formulae for symmetric functions. IBM report RC-5143, Yorktown Heights, NY (November 20, 1974).*

[18] Paterson M.S., Pippenger N., Zwick U., *faster circuits and shorter formulae for multiple addition, multiplication and symmetric Boolean functions. Selected papers from LMS Symposium on* Boolean function complexity, *Durham 1990. Cambridge University Press, 1991 (to appear).*

[19] Schönhage A., Strassen V., *Schnelle Multiplikation grosser Zahlen. Computing, Vol. 7 (1971), pp. 281-292.*

[20] Wegener I., *The complexity of Boolean functions. Wiley-Teubner series in Computer Science, 1987.*

[21] Wallace C.S., *A suggestion for a fast multiplier. IEEE Trans. Electronic Comp. EC-13 (1964) pp. 14-17.*