Design, Analysis and Implementation of an Adder by Ladner and Fisher

Manfred Minimair

10 June 1994

e-mail: `manfred.minimair@risc.uni-linz.ac.at`

Abstract

An addition circuit by Ladner and Fisher is analysed. It has size bounded by $(8+6\cdot2^{-k})n$ and depth bounded by $2\lceil\log_2 n\rceil+2k+2$, where $0 \leq k \leq \lceil\log_2 n\rceil$ and n is the bit length of the input numbers. Moreover the implementation of an instance of this adder capable of adding two numbers of 8-bit length was timed. The average computing time of its logical design and its layout is 26.8 ns and 94.2 ns, respectively. The layout of this circuit was done on an FPGA by *Atmel*.

Contents

1 Introduction

Addition is one of the most frequently performed basic operations on recent digital computers. Therefore it is crucial to design efficient addition circuits. Despite the simplicity of the operation of adding two binary numbers there exist several adders based on very different design ideas. Thus if one needs to implement an addition circuit one must decide which circuit is most appropriate for its planned application. This report gives a survey on the most important results concerning addition circuits and particularly reviews the design of an adder by Ladner and Fisher. Moreover the results of testing an implementation of this adder are summarized. In this way this paper is supposed to serve as a basis for the comparison of the Ladner-Fisher adder with the implementations of adders from different sources. (A comparison and evaluation of the implementations of various different adder designs can be found, for instance, in [2].)

Below there follows a short description of the contents of this work.

First in section 2 several results about the complexity of general boolean functions are presented (cf. [12], [6]). The author shows how these results can be applied in order to obtain bounds on the complexity of the addition function. In this section there is also a theorem about the optimal size of the addition function stated (cf. [9]). Moreover there can be found a table containing the depths and sizes of several known adders (cf. [1], [8], [10], [11], [4], [5], [12]).

Section 3 contains the design and analysis of the adder by Ladner and Fisher according to [5]. In subsection 3.1 the general design of the adder is presented. The parallel algorithm solving the problem of adding two n-bit binary numbers is developed step by step in the frame of automata theory. In the following subsection 3.2 the problem of prefix computation is defined and a circuit solving this problem is given. The complexity of this circuit is also stated. The circuit solving the prefix problem is the main part of the adder by Ladner and Fisher resulting from the parallel algorithm developed in subsection 3.1. In subsection 3.3 the complexity of this adder is determined by using the results of subsections 3.1 and 3.2.

Then in section 4 the author carries out some theoretical considerations about the running time behaviour ot the adder by Ladner and Fisher. Moreover the theoretical results are confirmed by the timing results of the author's implementation of the adder. This specific implementation is capable of adding two numbers of 8-bit length. Its circuit was drawn with the use of the CAD-program *Workview* by *Viewlogic*. Moreover the layout was done on an FPGA by *Atmel* automatically. Both the logical design and the layout were simulated and timed on a PC equipped with a CPU by *Intel*. The average computing time of the logical design and the layout is 26.8 ns and 94.2 ns, respectively.

Finally, the appendix contains the drawings of the circuit.

2 Complexity of addition circuits

2.1 Problem statement

First we fix some notations used throughout this paper.

1. Let $\mathbf{B} := \{0, 1\}$. $\mathbf{B}$ serves as a binary alphabet by which all information processed by the circuits is encoded. $\mathbf{B}_{n,m} := \{f | f : \mathbf{B}^n \to \mathbf{B}^m\}$ denotes the set of the so-called boolean functions from $\mathbf{B}^n$ to $\mathbf{B}^m$. We abbreviate $\mathbf{B}_{n,1}$ by $\mathbf{B}_n$.

2. Let S be a circuit, then $C(S)$ and $D(S)$ denote its size and depth, respectively.

3. Let $\Omega \subseteq \mathbf{B}_2$ be a basis for a circuit, then the size or complexity of f w.r.t. Ω is denoted by

$$C_\Omega(f) := \min\{C(S) | S \text{ computes } f \text{ w.r.t. } \Omega\}.$$

   Further

$$D_\Omega(f) := \min\{D(S) | S \text{ computes } f \text{ w.r.t. } \Omega\}$$

   is called the depth of f w.r.t. Ω and

$$L_\Omega(f) := \min\{C(S) | S \text{ computes } f \text{ w.r.t. } \Omega, S \text{ has fan-out } 1\}$$

   is the formula size of f w.r.t. Ω.

4. As complexity and depth of Boolean functions can increase only by a constant factor if we switch from one complete basis to another (cf. [12]) we restrict ourselves to the complete basis $\Omega = \mathbf{B}_2$. Hence from now on C_Ω, D_Ω and L_Ω are abbreviated by C, D and L, respectively.

5. $A_n \in \mathbf{B}_{2n,n+1}$ is the addition function, having two n-bit numbers x and y as inputs and computing the $(n+1)$-bit representation of $x+y$. An n-bit number z is written in the form $z = (z_n, \ldots, z_1)$ and its value is $|z| = 2^{n-1}z_n + \ldots + 2z_2 + z_1$.

In the following section we summarize the most important results concerning circuits S computing A_n with the purpose of minimizing $C(S)$ and $D(S)$.

2.2 Bounds on the complexity of the addition circuit

First bounds of the complexity of arbitrary $f \in \mathbf{B}_n$ are stated. From these results bounds of the complexity of A_n are derived. Then an overview of the complexities of existing circuits for A_n is given.

The following theorems state lower and upper bounds for the depth of a Boolean function.

Theorem 2.1 [12] *Let $n \in \mathbf{N}$ and $f \in \mathbf{B}_n$, then*

$$\lceil \log_2(L(f)+1) \rceil \leq D(f)$$

Theorem 2.2 [12] *Let $n \in \mathbf{N}$ and $f \in \mathbf{B}_n$, then*

$$D(f) \leq n + \lceil \log_2 n \rceil - 1$$

One can also find a lower bound of the sizes of special Boolean functions:

Definition 2.3 [12] *$f \in \mathbf{B}_n$ is called non degenerated if f depends essentially on all its variables, i.e. if the subfunctions of f for $x_i = 0$ and $x_i = 1$ are different.*

Lamagna and Savage prove the following theorem in [6] (as Wegener writes in [12]).

Theorem 2.4 [6] *Let $f \in \mathbf{B}_n$ be non degenerated, then*

$$n - 1 \leq C(f)$$

adder	depth	size
Brent [1]	$\log_2 n + O(\sqrt{\log_2 n})$	$O(n \log_2 n)$
Carry Look Ahead [8]	$O(\log_2 n)$	$O(n)$
Conditional Sum [10],[11]	$2\log_2 n + 1$	$3n\log_2 n + 10n - 6 \qquad n = 2^k$
Krapchenko [4]	$\lceil \log_2 n \rceil + 7\sqrt{2\lceil \log_2 n \rceil} + 16$	$3n + 6 \cdot 2^{\lceil \log_2 n \rceil}$
Ladner-Fisher [5]	$2\lceil \log_2 n \rceil + 2k + 2$	$(8 + 6 \cdot 2^{-k})n \qquad 0 \leq k \leq \lceil \log_2 n \rceil$
School Method [12]	$2n - 1$	$5n - 3$

Figure 1: Depths and sizes of several adders

As $L(f) \geq C(f)$ one can derive for non degenerated Boolean functions f from the previous theorems 2.2, 2.4:

$$\lceil \log_2 n \rceil \leq D(f)$$

Obviously A_n can be written in the following form.

$$A_n = (\hat{A}_{n+1}, \hat{A}_n, \ldots, \hat{A}_1),$$

where $\hat{A}_i \in \mathbf{B}_{2n}$ computes the i-th digit of the result of A_n. Clearly $\hat{A}_{n+1}$ is non degenerated. Hence it follows

$$\lceil \log_2 n \rceil \leq D(A_n) \leq n + \lceil \log_2 n \rceil - 1$$

and

$$n - 1 \leq C(A_n).$$

This lower bound for $C(A_n)$ seems rather rough as only $C(\hat{A}_{n+1})$ was taken into account in order to derive it. But Red'kin shows that:

Theorem 2.5 (Red'kin) [9]

$$C(A_n) = 5n - 3$$

Figure 1 contains bounds of the sizes and depths of several existing adders.

One can see that the school method of addition leads to an adder with optimal size but with far too big depth.

For $n = 2^k$ the size of Krapchenko's adder is only by the factor 1.8 larger then $C(A_n)$.

Both the Carry Look Ahead Adder and the Ladner-Fisher adder have depth $O(\log_2 n)$ and size $O(n)$. In this paper the design and analysis of the Ladner-Fisher adder will be described because of its good complexity and because it is designed from an abstract viewpoint which can easily be generalized to problems similar to addition (cf. [5]).

3 Design and analysis of the adder by Ladner and Fisher

3.1 Design of the adder

In order to design an addition circuit we reformulate the problem of adding two n-bit numbers in the language of automata theory. In the following definition an automaton, representing a full adder, is defined.

Definition 3.1 (full adder) *Let*

$$F := (I_F, S_F, Z_F, \delta_F, \gamma_F, s_0)$$

be a finite Mealy-machine, where I_F is the input alphabet, S_F the set of states, Z_F the output alphabet, $\delta_F : S_F \times I_F \to S_F$ the transition function, $\gamma_F : S_F \times I_F \to Z_F$ the output function and s_0 the initial state. We define

$$I_F := \mathbf{B}^2, S_F := Z_F := \mathbf{B},$$

$$\delta_F(s,(x,y)) := (x \wedge y) \vee ((x \oplus y) \wedge s),$$

$$\gamma_F(s,(x,y)) := x \oplus y \oplus s,$$

where $\oplus$ denotes the XOR-function.

F obviously realizes a full adder, i.e. an $i \in I_F$ represents the two digits to be added and s_0 is the carry from previous computations. The result of the addition will be returned by γ_F, the new state of F, computed by δ_F, is the carry of the addition.

Let $\hat{\delta_F}$ be the usual extension of δ_F to input words $\in I_F^*$, the free monoid over I_F. Then we can compute $A_n((x_n,\ldots,x_1),(y_n,\ldots,y_1))$ by using F in the following way. Let

$$s_0 := 0,$$

$$s_j := \hat{\delta_F}(s_0,(x_1,y_1)\ldots(x_j,y_j)),\ 1 \le j \le n$$

then

$$A_n((x_n,\ldots,x_1),(y_n,\ldots,y_1)) = (s_n, \gamma_F(s_{n-1},(x_n,y_n)),\ldots,\gamma_F(s_0,(x_1,y_1)))$$

This is an instance of the following problem of automata theory.

Problem 3.2 *Let $M := (I,S,Z,\delta,\gamma,s_0)$ be a finite Mealy-machine. Given an input word $w := i_1\ldots i_n$ we want to compute the output generated by M while processing w and the final state of M after having processed w. Thus we want to compute the following $(n+1)$-tuple:*

$$(s_n, \gamma(s_{n-1},i_n),\ldots,\gamma(s_0,i_1)),$$

where

$$s_j := \hat{\delta}(s_0, i_j\ldots i_1),\ 1 \le j \le n$$

In the rest of this section a parallel algorithm is developed that solves problem 3.2.

If the s_j in 3.2 are known in advance the components

$$\gamma(s_{j-1},i_j),\ 1 \le j \le n$$

of the result can be easily computed in parallel. Hence we must find a method to calculate the s_j efficiently.

By definition (cf. [7]) each $i \in I$ generates an element δ_i of the monoid of M $\mathcal{M}_M := \{\hat{\delta}(.,w) : S \to S \mid w \in I^*\} \subseteq S^S$, where

$$\delta_i(s) := \hat{\delta}(s,i) = \delta(s,i) \in \mathcal{M}_M.$$

Using this notation, we have for $1 \le j \le n$

$$\begin{aligned} s_j &= \delta(s_{j-1},i_j) \\ &= \delta(\delta(\ldots\delta(s_0,i_1),i_{j-1}),i_j) \\ &= (\delta_{i_j} \circ \ldots \circ \delta_{i_1})(s_0) \end{aligned}$$

Thus each single s_j can be computed in two steps:

1. calculate the functions

$$g_j := \delta_{i_j} \circ \ldots \circ \delta_{i_1} \in \mathcal{M}_M,\ 1 \leq j \leq n$$

2. evaluate g_j at s_0, i.e. compute

$$s_j = g_j(s_0),\ 1 \leq j \leq n$$

Calculating all function products $g_j \in \mathcal{M}_M$ in step 1 is an instance of the so-called prefix problem which is defined in the next section. There we also present a circuit that solves this problem efficiently.

Step 2 can be carried out for all s_j in parallel after having computed all g_j by using the circuit of the next section.

Summarizing we have the following algorithm.

Algorithm 3.3 [5] *Problem 3.2 can be solved by executing the following steps:*

1. *Compute* $\delta_{i_1}, \ldots, \delta_{i_n}$ *in parallel.*
2. *Compute* $g_1 = \delta_{i_1}, \ldots, g_n = \delta_{i_n} \circ \ldots \circ \delta_{i_1}$ *by a parallel prefix algorithm.*
3. *Compute* $s_1 = g_1(s_0), \ldots, s_n = g_n(s_0)$ *in parallel.*
4. *Compute* $\gamma(s_0, i_1), \ldots, \gamma(s_{n-1}, i_n)$ *in parallel.*

3.2 Parallel prefix computation

Problem 3.4 (prefix problem) [5] *Let* $(H, \circ)$ *be a semigroup. The* prefix problem *is to compute, for given* $x_1, \ldots, x_n \in H$*, each of the products*

$$x_1 \circ x_2 \circ \ldots \circ x_j,\ 1 \leq j \leq n$$

In this section a family $\mathcal{P}_k(n)$ of circuits solving the prefix problem for n inputs is defined. Each inner node of the circuits corresponds to the semigroup operation $\circ$. Therefore each inner node has two inputs a_1, a_2 and its output is $a_1 \circ a_2$. The size and the depth of the circuit corresponding to the additional parameter $k \in \mathbf{N}_0$ changes according to this parameter, i.e. when k decreases the depth also decreases but the size increases.

The construction of the family $\mathcal{P}_k(n)$ is shown in figure 2 and 3[5]. When $n = 1$, $\mathcal{P}_k(n)$ is only a single input node and contains no products.

The following theorem can be proved by induction.

Theorem 3.5 [5] *The circuits recursively defined in figure 2 and 3 solve the prefix problem. For all* $n \in \mathbf{N}$ *the following bounds for the size and depth of* $\mathcal{P}_k(n)$ *exist:*

1. $D(\mathcal{P}_k(n)) \leq k + \lceil \log_2 n \rceil$ *for all* $k \geq 0$,
2. $C(\mathcal{P}_k(n)) \leq 2(1 + \frac{1}{2^k})n - 4$ *for all* $0 \leq k \leq \lceil \log_2 n \rceil$

If n *is a power of* 2*, we have:*

$$C(\mathcal{P}_k(n)) = 2(1 + \frac{1}{2^k})n - F(5 + \log_2 n - k) + 1 - k \ \ \textit{for all}\ \ 0 \leq k \leq \lceil \log_2 n \rceil$$

where $F(m)$ *denotes the* m*-th Fibonacci number.*

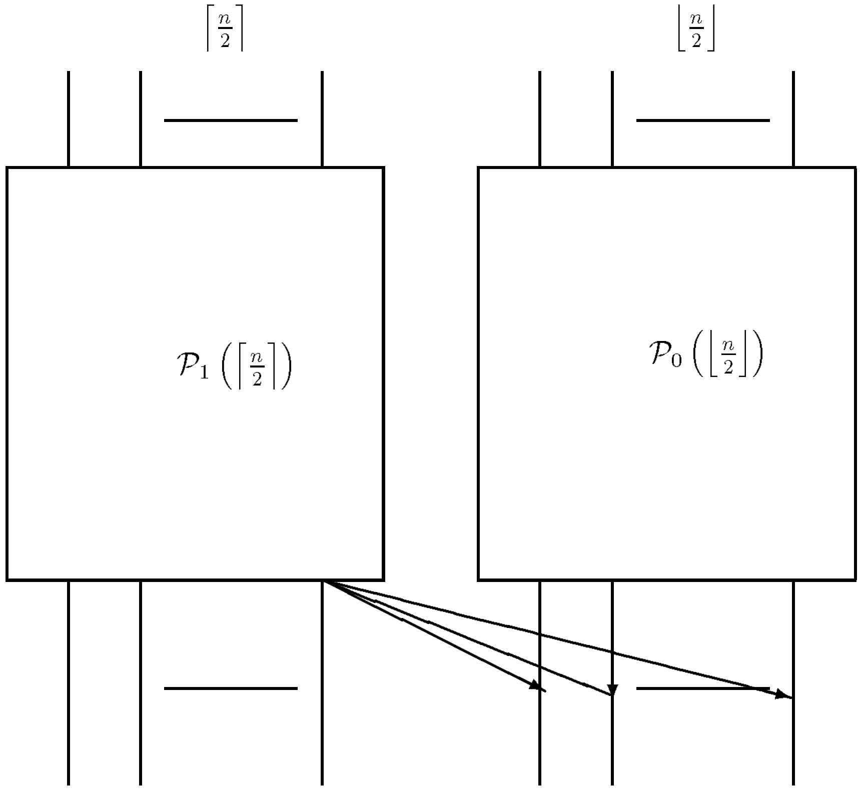

Figure 2: The construction of $\mathcal{P}_0(n)$.

One can also derive a bound for the fan-out of the circuits $\mathcal{P}_k(n)$. Let $fo(k,n)$ denote the maximal fan-out of all nodes in $\mathcal{P}_k(n)$.

If $n \leq 2^{k+1}$ and $1 \leq k$ we know that $\mathcal{P}_k(n) = \mathcal{P}_{k-1}(n)$ because at the bottom of the recursion in figure 3 a circuit $\mathcal{P}_l(1)$ for some $l \geq 1$ is executed. In the other case we have

Theorem 3.6 [5] *Let $k \geq 0$ and $n > 2^{k+1}$, then*

$$fo(k,n) \leq \lfloor \frac{n + 2^k - 1}{2^{k+1}} \rfloor + k$$

3.3 Complexity of Ladner's and Fisher's adder

Let

1. c_1 (d_1) be the size (depth) of a circuit computing δ_i for $i \in I$,
2. c_2 (d_2) be the size (depth) of computing the functional composition in $\mathcal{M}_M$,
3. c_3 (d_3) be the size (depth) of computing functional evaluation,
4. c_4 (d_4) be the size (depth) of computing $\gamma(s,i)$ for $s \in S$ and $i \in I$,
5. $c(n)$ ($d(n)$) be the size (depth) of a product circuit solving the prefix problem.

Assuming that the initial state s_0 can be encoded and decoded at no cost this yields the following bounds for a circuit S realizing algorithm 3.3:

$$\begin{aligned} C(S) &\leq c_2 c(n) + (c_1 + c_3 + c_4)n, \\ D(S) &\leq d_2 d(n) + d_1 + d_3 + d_4. \end{aligned}$$

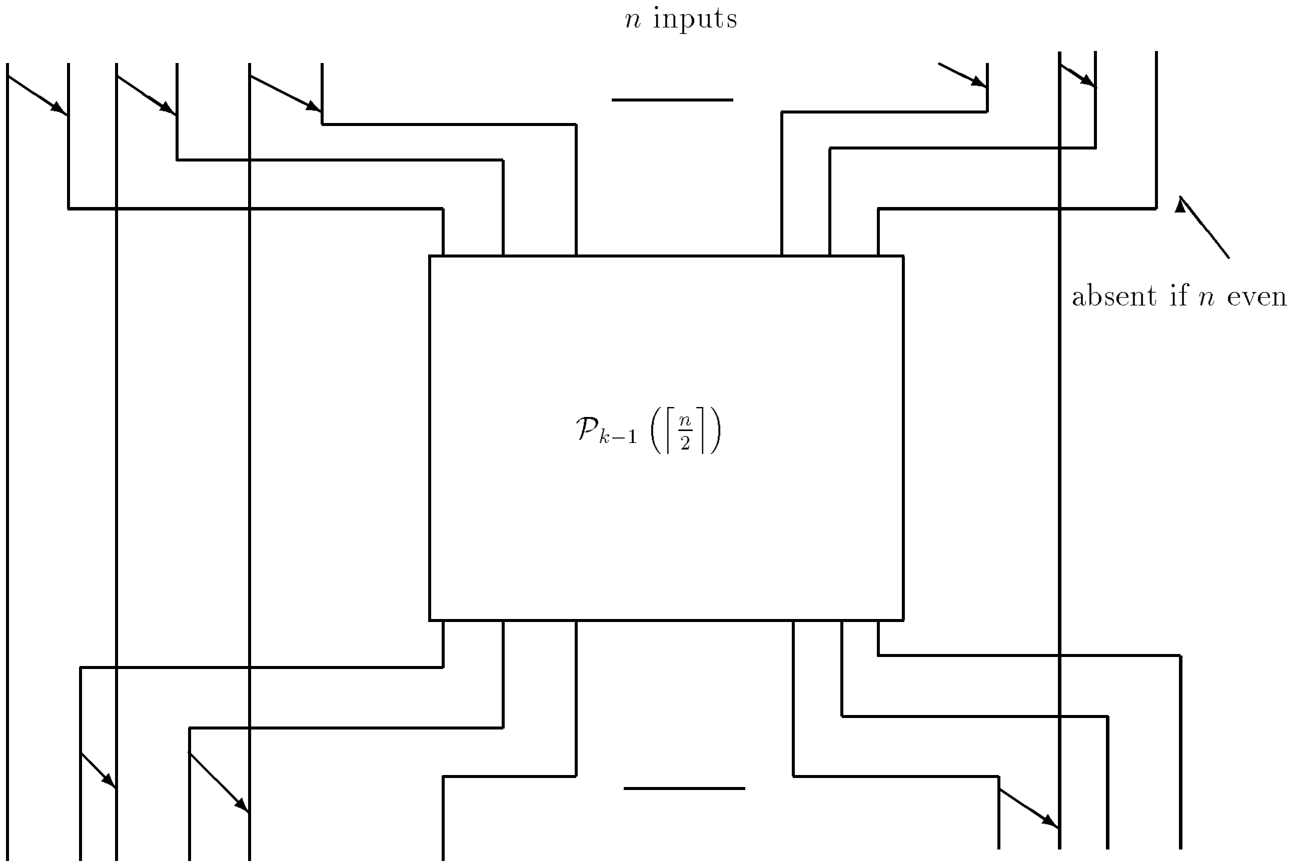

Figure 3: The construction of $\mathcal{P}_k(n)$, $k \geq 1$.

input	function	
xy	gq	
00	00	$g = x \wedge y$
10	01	$p = x \oplus y$
01	01	
11	10	

Figure 4: Computation of the functions from the inputs.

		function $g_2 p_2$			
	$\circ$	00	01	10	
function	00	00	00	10	$g = g_2 \vee (g_1 \wedge p_2)$
$g_1 p_1$	01	00	01	10	$p = p_1 \wedge p_2$
	10	00	10	10	

Figure 5: Composition table.

In order to apply these results to the problem of computing A_n we encode the functions $\delta_{F\,i}$ of the fullader 3.1 for $i \in I_F$ in the following way.

First note that $\delta_{F\,(1,0)} = \delta_{F\,(0,1)}$ and that $\delta_{F\,(0,0)}$, $\delta_{F\,(1,0)}$, $\delta_{F\,(1,1)}$ are pairwise different. Moreover $\delta_{F\,(0,0)}$ represents the identity function on S_F and $\{\delta_{F\,(0,0)}, \delta_{F\,(1,0)}, \delta_{F\,(1,1)}\}$ is already closed under function composition $\circ$. Hence $\mathcal{M}_F = \{\delta_{F\,(0,0)}, \delta_{F\,(1,0)}, \delta_{F\,(1,1)}\}$.

Now let $w := i_1 \ldots i_n$ be an input word, where $i_j \in \mathbf{B}^2$. Then we partition $\mathbf{B}^2$ into three classes according to the way a carry at some position j of w is treated. If

1. $i_j = (0,0)$ then we know that a carry from previous bits need *not* be propagatet and a new carry need *not* be generated at position j,
2. $i_j = (1,0)$ or $i_j = (0,1)$ then possibly a carry must be *propagated* but a new carry need *not* be generated,
3. $i_j = (1,1)$ then a carry from previous bits need *not* be propagated but a new carry must be *generated*.

Letting $p = 1$ denote that the carry possibly must be propagated and $g = 1$ denote that a carry must be generated we have a two bit encoding (g, p) of the functions $\delta_{F\,(x,y)}$ shown in figure 4[5].

Figure 5[5] shows how two functions $g_1 p_1$ and $g_2 p_2$ are composed.

If (g, p) represents the function $\delta_{F\,i}$ then the evaluation at a certain state s has the following form (cf. definition 3.1):

$$\delta_{F\,i}(s) = g \vee (p \wedge s).$$

The output $\gamma_{F\,i}(s_0, i)$ can also be calculated using (g,p):

$$\gamma_{F,\,i}(s, i) = s_0 \oplus p.$$

From this we can calculate the constants from above

$$\begin{aligned} c_1 &= 2, & d_1 &= 1, \\ c_2 &= 3, & d_2 &= 2, \\ c_3 &= 2, & d_3 &= 2, \\ c_4 &= 1, & d_4 &= 1, \end{aligned}$$

Hence we have the following bounds for the size and depth of an addition circuit A that is derived from algorithm 3.3

	k							
	0		1		2		3	
bits	$D(A)$	$C(A)$	$D(A)$	$C(A)$	$D(A)$	$C(A)$	$D(A)$	$C(A)$
4	6	20	8	20	10	20		
8	8	52	10	49	12	49	14	49
16	10	125	12	113	14	110	16	110
32	12	286	14	250	16	238	18	235
64	14	632	16	539	18	503	20	491
128	16	1363	18	1141	20	1048	22	1012

Figure 6: Depths and sizes of small adders.

$$\begin{aligned} C(A) &\leq 3c(n) + 5n, \\ D(A) &\leq 2d(n) + 4. \end{aligned}$$

Observing that the evaluation at s_0 reduces to $\delta_{F^i}(s_0) = g$ if the initial state $s_0 = 0$ one concludes there is no need to compute p at the last level before step 3 in algorithm 3.3. Thus this results in total savings of $3n$ in size and of 2 in depth. Hence we have

$$\begin{aligned} C(A) &\leq 3c(n) + 2n, \\ D(A) &\leq 2d(n) + 2. \end{aligned}$$

Combining these results with the results from the previous section we get for $0 \leq k \leq \log_2 n$

$$\begin{aligned} C(A) &\leq (8 + \frac{6}{2^k})n, \\ D(A) &\leq 2\log_2 n + 2k + 2. \end{aligned}$$

For small n figure 6[5] contains the sizes and depths of the corresponding addition circuits A.

4 Implementation of the addition circuit

In this section first some theoretical consideration about the running times of the previously described adders are carried out. Then the results of testing the circuit for A_8 with parameter $k = 0$ are given. The appendix contains the drawings of the corresponding circuit. It was drawn with the use of the CAD-program *Workview* by *Viewlogic*. Moreover the layout was done on an FPGA by *Atmel* automatically. Both the logical design and the layout were simulated and timed on a PC equipped with a CPU by *Intel*.

By regarding the composition tabel in figure 5 one observes that

$$(g, p) \circ (0, 0) = (0, 0)$$

and

$$(g, p) \circ (1, 0) = (1, 0),$$

for all $(g, p) \in \{(0, 0), (1, 0), (0, 1)\}$.

	input	logical time (ns)	layout time (ns)
	10101010		
+	01010101	29.8	130.9
	00110011		
+	11001100	29.4	130.9
	00001111		
+	11110000	29.4	130.9
	00110101		
+	11001010	29.4	130.5

Figure 7: All digits are different.

	input	logical time (ns)	layout time (ns)
	10011001		
+	10010110	23.7	97.1
	10011001		
+	01101001	26.3	95.7
	10011001		
+	10101010	26.3	90.8
	10011001		
+	01010101	28.8	105.4

Figure 8: 4 digits are different/equal.

Thus if either a pair $i_j = (0,0)$ or a pair $i_j = (1,1)$ appear at the j-th position of an input word $w = i_1 \dots i_n$ of the automaton F one needs not know $i_1 \dots i_{j-1}$ in order to compute g_j in step 2 in algorithm 3.3. For g_j can simply be calculated by

$$\begin{aligned} g_j &= \delta_{F\, i_j} \circ g_{j-1} \\ &= \delta_{F\, i_j}. \end{aligned}$$

In this case there are fewer dependencies between the various g_j's and a higher amount of parallelism is achieved in step 2. Therefore the adders derived from algorithm 3.3 and based on the product circuit by Ladner and Fisher will add two numbers $x = (x_n, \dots, x_1)$, $y = (y_n, \dots, y_1)$ the faster the greater the number of indices j with $x_j = y_j$ is.

Figures 7, 8, 9 contain the running times in nanoseconds of the logical simulation of the circuit and of the simulation after having done the layout. There are numbers

	input	logical time (ns)	layout time (ns)
	00000000		
+	00000000	23.3	82.4
	11111111		
+	11111111	21.7	77.2
	10101010		
+	10101010	22.9	77.2
	11001100		
+	11001100	21.8	77.2

Figure 9: All digits are equal.

n=100	$\bar{t}$ (ns)	s_t (ns)
logical	26.8	2.2
layout	94.2	11.4

Figure 10: Mean value, standard deviation.

of three groups of characteristic inputs added. In the first group (figure 7) the corresponding digits of the two input numbers are all different. In the second group (figure 8) 4 digits are different and 4 digits are equal. In the third group (figure 9) both inputs are equal.

As one would expect from the previous considerations adding inputs out of the first group is most time consuming, whereas inputs of the third group are processed in the shortest amount of time.

Finally, figure 10 contains the empirical mean value $\bar{t}$ and the empirical standard deviation s_t of the running times t_j needed to add 100 randomly chosen input pairs, where $\bar{t} = \frac{1}{n}\sum_{j=1}^{n} t_j$, $s_t = (\frac{1}{n-1}\sum_{j=1}^{n}(t_j - \bar{t})^2)^{\frac{1}{2}}$ [3] and $n = 100$. Regarding figure 10 one observes that after having done the layout the simulation of the circuit is $3-4$ times slower than the logical simulation. Moreover the standard deviation increases by a factor of about 5.

References

[1] Brent, *On the addition of binary numbers*, IEEE Trans.on Comp. 19, pp.758-759

[2] C.-J. Chang, S. Vassiliadis, J. G. Delgado-Frias, *An investigation of binary CLA and ripple CMOS adder designs*, Microprocessing and Microprogramming 40 (1994), pp. 1-21

[3] K.A. Brownlee, *Statistical Theory and Methodology*, 2nd edition, John Wiley & Sons, Inc., New York 1965

[4] Krapchenko, *Asymptotic estimation of addition time of parallel adder*, Syst.Th.Res. 19, pp. 105-122

[5] E. Ladner, M. Fisher, *Parallel Prefix Computation*, Journal of the Association for Computing Machinery, Vol.27, No. 4, Oct. 1980, pp.831-838

[6] Lamagna, Savage, *On the logical complexity of symmetric switching functions in monotone and complete bases*, Technical Report Brown Univ.

[7] R. Lidl, G. Pilz, *Applied abstract Algebra*, Springer, New York 1984

[8] Ofman, *On the algorithmic complexity of discrete functions*, Sov.Phys.Dokl. 7, pp. 589-591

[9] Red'kin, *Minimal realization of a binary adder*, Probl.Kibern. 38, pp. 181-216, 272

[10] Sklansky, *An evaluation of several two-sum and binary adders*, IRE Trans.Elect.Comp. 9, pp. 213-226

[11] Sklansky, *Conditional-sum addition logic*, IRE Trans.Elect.Comp. 9, pp. 226-231

[12] I. Wegener, *The Complexity of Boolean Functions*, Wiley-Teubner, Stuttgart 1987