Efficient Addition on Field Programmable Gate Arrays

Andreas Jakoby [1] and Christian Schindelhauer [2]

[1] Institut für Theoretische Informatik, Med. Universität zu Lübeck
jakoby@tcs.mu-luebeck.de
[2] Heinz Nixdorf Institute and Depart. of Mathematics and Computer Science, Univ. Paderborn,
schindel@uni-paderborn.de

Abstract. We investigate average efficient adders for grid-based environments related to current Field Programmable Gate Arrays (FPGAs) and VLSI-circuits. Motivated by current trends in FPGA hardware design we introduce a new computational model, called the λ-wired grid model. The parameter λ describes the degree of connectivity of the underlying hardware. This model covers among others two-dimensional cellular automata for $\lambda = 0$ and VLSI-circuits for $\lambda = 1$. To formalize input and output constraints of such circuits we use the notion of input and output schemas. It turns out that the worst case time and area complexity are highly dependent on the specific choice of I/O schemas. We prove that a set of regular schemas supports efficient algorithms for addition where time and area bounds match lower bounds of a broad class of I/O schemas.
We introduce new schemas for average efficient addition on FPGAs and show that addition can be done in expected time $O(\log\log n)$ for the standard VLSI model and in expected time $O(\sqrt{\log n})$ in the pure grid model. Furthermore, we investigate the rectangular area needed to perform addition with small error probability, called area with high probability. Finally, these results are generalized to the class of prefix functions.

1 Introduction

To investigate the average time behavior of parallel systems we introduced the notion of average-complexity for circuits [16]. It turns out that for many basic Boolean functions like addition, threshold, or comparison there are special circuits, that provide a substantial speedup in the average for a wide class of probability distributions. This analysis was extended to a broader class of functions, the prefix functions, in [17, 12, 13]. It is not known how Boolean circuits with optimal average case behavior can be used in practice. A straight-forward approach is to use these averagely efficient circuits in asynchronous VLSI-design. However, new problems arise when it comes to verification, efficient combination, and placement of such circuits. Note that the analysis of expected time and worst time of a given circuit is a computationally infeasible task [14]. Determining these parameters is a precondition to predict hazards or to calculate local clocks. Hence, it seems to be necessary to design new circuits reflecting the restrictions given by the VLSI-environment.

A promising computational device for the implementation of average efficient circuits are Field Programmable Gate Arrays (FPGAs). FPGAs evolved from programmable array logics which can be configured to compute primitive Boolean functions, e.g. by defining clauses of a conjunctive normal form. By the time more and more computational power was given to the basic logical units, called *logic blocks*, while the degree of chip integration increased their number. Programmable Gate Arrays introduced in [6] (see also [9–11] and [7, 20] for surveys) can be used as an alternative to traditional hardware design: To implement the functionality of special processors it is cheaper and faster to configure an FPGA-chip, e.g. for mobile telephones or hand-held devices, than to make an Application Specific Integrated Circuit (ASIC). So, FPGAs extraordinarily reduce integrated circuit manufacturing time and prototype costs. The FPGAs' universal layout allows the emulation of every specific hardware after appropriate configuration. The underlying concept is a *sea of gates*: a uniform array of logic blocks with a regular interconnection network.

FPGAs of the latest generation, e.g. [1, 2, 23, 24], provide up to some 60k free programmable logic blocks, some 64k bit RAM, and a clock frequency of 100 MHz, which on first sight seems not much compared to the current generation of processors and memory chips. But FPGAs are inherently parallel and fully programmable (some are even reconfigurable during computation). Furthermore, the integration of FPGAs has not yet reached the same level as of ASICs.

FPGAs are used as co-processors for high speed designs by implementing a variety of compute-intensive arithmetic functions including FFT, discrete-cosine transforms (DCT), and interpolators [22]. Furthermore, they are useful for cryptographical and cryptoanalytical applications: In [3] it is suggested to use FPGAs for a brute force attack against the DES crypto-system. More computational power is only achievable by a substantial longer development period and with higher costs.

There are two basic approaches to design the interconnection of gates within an FPGAs: the routing and the mesh design. In the *routing design* logic blocks are arranged along the boundary of the chip and the interior is completely occupied by a routing network. In the *mesh design* logic blocks are positioned in a grid and each block is connected to its direct neighbors. There is a restricted number of long wires facilitating long range interconnections of logic blocks. The mesh design allows a greater number of logic blocks. On the other hand its interconnection structure is more restricted than in the routing design.

In the mesh design the interconnection between logic blocks vary widely between manufacturers and between manufacturers' chip series. Some offer connections to diagonal adjacent neighbors, arbitrary zigzag-lines, buses for cells on the same row or column, connections to the second next neighbor, connections to the forth next neighbor, and so forth. Mesh based FPGAs provide a subset of these connection structures while not every combination of these may be used in a certain configuration. Also there are differences in the computational power of the logic blocks, in the number of clocks, and in the distribution and amount of memory. For a general approach it is necessary to define a computational model which covers a wide range of different FPGA designs.

In this paper we introduce a computational model for FPGAs, the λ*-wired grid model* motivated by common FPGA design principles [7]. The parameter λ addresses the amount of available long range interconnections. For $\lambda = 0$ it corresponds to the

model of two dimensional cellular automata. For $\lambda = 1$ it describes the theoretical model of VLSI-circuits as discussed in [21]. If λ is large enough it models FPGAs in the routing design. Like in VLSI we investigate time and area as complexity measures in the λ-wired grid model.
The computational complexity of basic functions like sorting, addition, multiplication, and many more are well studied in the VLSI-circuit model [21]. One achievement was to find ultimate lower bounds for time and area based on the planarity of chips and the speed of light. But, matching upper bound results do not necessarily lead to practical VLSI-algorithms. To achieve practical solutions input and output compatibility has to be realized. Our approach to I/O compatibility is the notion of *input* and *output schemas* to determine the meaning of an input or output bit at a pin position at any time.
We focus on the time and area complexity of the addition of long binary numbers as well as on arbitrary confluent prefix functions as introduced in [17]. For a specific input and output schema Brent and Kung [4] show that addition can be implemented in time $O(\log n)$ and area $O(n)$. This refers to the area occupied by the gates and the wires. It implements the Ladner-Fisher carry-propagation network [18] for partially pipelined input, i.e. an input is partitioned into a sequence of parallel blocks of equal length. In this paper we show that this bound is best possible for a wide class of schemas, called compact schemas. These bounds are also optimal, if the input and output schema are not compact, but closely related and if λ, the degree of connectivity, is arbitrarily high.
We introduce new schemas for efficient addition on FPGAs and show that addition can be done in expected time $O(\log\log n)$ for the standard (1-wired grid) model and in expected time $O(\sqrt{\log n})$ in the pure grid model. Furthermore, we investigate the size of a rectangular area that correctly performs addition with high probability, called *area w.h.p.* Following the ideas in [19] area w.h.p. can help reducing the over-all area, e.g. many adders using this smaller area share one large area consuming worst-case adder, which is only necessary for a small fraction of the inputs.
This paper is structured as follows. In Section 2 we introduce the notion of I/O schemas and a computational model for FPGAs, the λ-wired grid model. In Section 3 we present efficient schemas and prove tight bounds for the computational complexity of addition of two binary numbers. In Section 4 we focus on algorithms for the average case.

2 The λ-Wired Grid Model

An algorithm on a FPGA occupies a rectangular area $A = m_1 \cdot m_2$ consisting of $m_1 \times m_2$ programmable gates $(g_{i,j})_{i\in[m_1],j\in[m_2]}$, called *logic blocks* and an *interconnection network* connecting each logic block to four neighbor gates. Inputs and outputs may only occur at the $2m_1 + 2m_2$ *frontiers* $(g_{i,0})_{i\in[m_1]}$, $(g_{i,m_2+1})_{i\in[m_1]}$, $(g_{0,j})_{j\in[m_2]}$, and $(g_{m_1+1,j})_{j\in[m_2]}$ ($[n]$ denotes the set $\{1,..,n\}$). For the interconnection structure we distinguish between the *grid*, where interconnections are allowed only between adjacent gates, and the *λ-wired connection network*, where cells have a constant number of wires connecting them with distant neighbors according to the configuration of an FPGA connection network. The input and output occurs only on a subset of the frontiers, called *pins*. We will address the input pins by $p_1, \ldots, p_r$. The computation of a circuit is done in *rounds* (e.g. synchronized by a local clock). In each round a gate receives the output of its neighbors and computes a pre-configured function depending

on these outputs and its state.

To obtain a high performance these algorithms should be integrated into a data flow. Therefore, it is very important that input and output time behavior is compatible. Pins may be used in a serial or parallel way to interchange input and output data. In general serial I/O helps to reduce the area needed for computation, while parallel I/O reduces the time needed for the whole computations. It is well known that a combination of these communication patterns can provide both qualities [4]. Our formalism for I/O compatibility is called *input* or *output schema*. We assume a unit time model for the *cycle time*, i.e. at each time step a new bit may arrive at each pin.

Definition 1 *An* **input or output schema (I/O schema)** $P \subseteq [n] \times [r] \times \mathbb{N}$ *of a vector* $x_1, \ldots, x_n$ *for pins* $p_1, \ldots, p_r$ *is a set of triples* (i, j, t)*, that describes that input bit* x_i *is available at pin* p_j *at time point* t*. For a schema* P *and pin* p_j *we define the start time* T_s *and the final time* T_f *by* $T_s(P, j) := \min_{(i,j,t)\in P} t$, $T_s(P) := \min_{(i,j,t)\in P} t$, $T_f(P, j) := \max_{(i,j,t)\in P} t$, *and* $T_f(P) := \max_{(i,j,t)\in P} t$.

The grid model, defined below, is the basic computational model for FPGAs in the mesh design (see Fig. 1). It is similar to two-dimensional cellular automata.

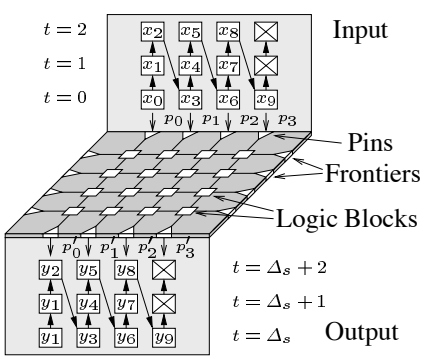

Fig. 1. The Grid-model

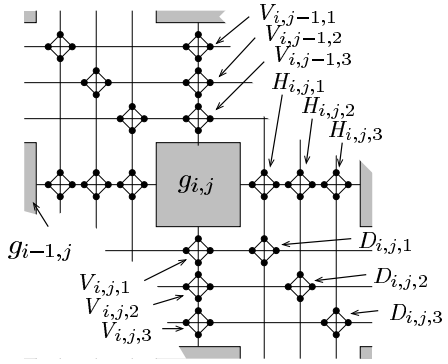

Fig. 2. The λ-wired grid model

Definition 2 *An algorithm in the* **grid model** *consists of an* $m_1 \times m_2$ *gate array* $(g_{i,j})_{i\in[m_1],j\in[m_2]}$ *and frontiers* $(g_{i,0})_{i\in[m_1]}$, $(g_{i,m_2+1})_{i\in[m_1]}$, $(g_{0,j})_{j\in[m_2]}$, *and* $(g_{m_1+1,j})_{j\in[m_2]}$ *which are grid like connected by* E

$$\{\{g_{i-1,j}, g_{i,j}\} \mid j \in [m_2],\ i \in [m_1+1]\} \cup \{\{g_{i,j-1}, g_{i,j}\} \mid j \in [m_2+1],\ i \in [m_1]\}.$$

In the configuration phase each logic block $g_{i,j}$ *can be configured to implement any function* $h_{i,j} : \Sigma^5 \to \Sigma^5$ *for a finite alphabet* Σ*. The computation phase starts at time* 0*. At every time step* t $(v_{i,j,t}, v^{\mathtt{n}}_{i,j,t}, v^{\mathtt{e}}_{i,j,t}, v^{\mathtt{s}}_{i,j,t}, v^{\mathtt{w}}_{i,j,t}) \in \Sigma^5$ *describes the state of a gate* $g_{i,j}$*. The internal state is given by* $v_{i,j,t}$*. Each of the other four values is transmitted to the corresponding neighbor. Unless gates are pins, for* $t = 0$ *all values are initialized with a special symbol* β*. This symbol* β *also describes the values of all non-pin frontiers. In each round* $t \geq 0$ *the following computation takes place*

$$\begin{aligned}(v_{i,j,t+1}, v^{\mathtt{n}}_{i,j,t+1}, v^{\mathtt{e}}_{i,j,t+1}, v^{\mathtt{s}}_{i,j,t+1}, v^{\mathtt{w}}_{i,j,t+1}) \\ := \ h_{i,j}(v_{i,j,t}, v^{\mathtt{s}}_{i,j-1,t}, v^{\mathtt{w}}_{i+1,j,t}, v^{\mathtt{n}}_{i,j+1,t}, v^{\mathtt{e}}_{i-1,j,t}).\end{aligned}$$

The values of the pins are determined by the input string and an input schema. Unused positions of the I/O schema are indicated by the special symbol β*.*

FPGAs in the mesh design have an additional number of long wires for fast interconnections. The potential of high functionality of logic blocks results in larger area and slower time compared to a hardware layout of the same functionality. In FPGA design this is compensated by the number of wires which can be interconnected by *switches*. Such a switch connects the ends of four wires $\mathtt{n},\mathtt{e},\mathtt{s},\mathtt{w}$ named by the cardinal directions. It features the following three *configurations*, which describe all three possible two-by-two connections, i.e. $\{\{\mathtt{n},\mathtt{s}\},\{\mathtt{e},\mathtt{w}\}\}$, $\{\{\mathtt{n},\mathtt{e}\},\{\mathtt{s},\mathtt{w}\}\}$, and $\{\{\mathtt{n},\mathtt{w}\},\{\mathtt{s},\mathtt{e}\}\}$. To simplify our model we assume bidirectional communication along the wires.

Definition 3 *For a switch or logic block p let p^d be its* **port** *to the cardinal direction $d \in \{\mathtt{n},\mathtt{e},\mathtt{s},\mathtt{w}\}$. For a port p^d let $v(p^d,t) \in \Sigma$ be the input value received by p from direction d at time t and $w(p^d,t) \in \Sigma$ the corresponding output value. If two ports p^d and $q^{d'}$ are connected we have $v(p^d,t) = w(q^{d'},t)$ and $w(p^d,t) = v(q^{d'},t)$.*

There is no common design principle in practice for the use of switches. For an overview see [7]. It turns out that many mesh based FPGA designs support the λ-wired connection model which is defined as follows:

Definition 4 *The λ-wired connection network connects logic blocks $g_{i,j}$ for $i \in [m_1], j \in [m_2], d \in \{\mathtt{n},\mathtt{e},\mathtt{s},\mathtt{w}\}$ with the switches $H_{i,j,k}$, $V_{i,j,k}$ and $D_{i,j,k}$ for $i \in [m_1 - 1]$, $j \in [m_2 - 1]$, $k \in [\lambda]$. The definition of the connections E_λ follows analogously to Fig. 2. For a configuration C of all switches, the resulting connection network $E_\lambda(C)$ describes a matching of all ports of logic blocks $g_{i,j}$.*

In practice there is a variety of connection networks for FPGAs which essentially provide similar features like the λ-connection network. The parameter λ varies for different FPGA designs, e.g. on the Xilinx XC4000E one can directly implement a 10-connection network and for the Xilinx XC4000X one can achieve $\lambda = 13$ (The switches used in the Xilinx architecture correspond to our model).
Note that the parameter λ also describes the relationship between size of logic blocks and area needed for wires, since for many hardware designs the area allocated for logic blocks is approximately as large as the area needed for wiring. We combine this connection network with the grid model:

Definition 5 *An algorithm in the* **λ-wired grid model** *consists of an $m_1 \times m_2$ gate array and a corresponding λ-connection network. In the configuration a function $h_{i,j} : \Sigma^5 \to \Sigma$ is assigned to the logic block $g_{i,j}$ and a configuration C of the network is chosen. The computation proceeds analogously to the computation in the grid-model except that in each round $t \geq 0$ the following computation takes place*

$$(q_{i,j,t+1}, w(g^{\mathtt{n}}_{i,j}, t+1), w(g^{\mathtt{e}}_{i,j}, t+1), w(g^{\mathtt{s}}_{i,j}, t+1), w(g^{\mathtt{w}}_{i,j}, t+1)) := h_{i,j}(q_{i,j,t}, v(g^{\mathtt{n}}_{i,j}, t), v(g^{\mathtt{e}}_{i,j}, t), v(g^{\mathtt{s}}_{i,j}, t), v(g^{\mathtt{w}}_{i,j}, t))) .$$

For $\lambda = 0$ the λ-wired grid model is equivalent to the grid model. An algorithm in the λ-wired grid model computes a function f w.r.t. I/O schemas P_I and P_O if for every input x given in accordance with P_I it outputs $f(x)$ using schema P_O.
Unlike for machine models like Turing machines or directed acyclic circuits it is not clear which notion of time for computation is most important for FPGAs. We use the following notions: The **over-all time** is the number of all time steps needed to compute the function, i.e. the time difference between the first input appears and the last output is delivered. The **latency** Δ_s is the time difference between the first given input

and the first valid output. The **follow-up time** Δ_f is the time difference between the last available input bit and the last output. Of course the standard measure is the overall time. The follow-up time is the decisive measure for pipelines consisting of many sub-circuits. The latency gives a measure for the best possible speedup, e.g. if another algorithm A waits for the outputs of a computation and A's computation depends only on one input element. A's computation may start even before the preceding computation (with small latency) is finished. To be able to take advantage of such effects we assume that there are acknowledgement mechanisms as presented in [8, 16].
Since we consider the (potentially long) wires and the mechanism controlling the I/O of an algorithm as a valuable resource we prefer schemas that use every pin in every step of a time interval. We call such a schema 1-compact, or simply *compact*. If only a fraction of $\frac{1}{c}$ of all input (or output) possibilities is used during the schema's term, we call the schema c-compact.

Definition 6 *An schema P with n elements using r pins is called c-**compact**, if $r(T_f(P) - T_s(P) + 1) \leq c\,n$. We call 1-compact schemas* **compact**. *A schema P is called a* **one-shot** *schema, if every input bit x_i occurs only once in P.*

Such a rigid notion of input and output behavior is not suitable for average-efficient circuits where some outputs may occur earlier. We allow earlier outputs if it does not disturb the chronological order of outputs at a pin.

Definition 7 *A* **relaxed schema** $RP(P) \subseteq 2^{[n]\times[m]\times\mathbb{N}}$ *is defined by a schema P and a set of functions $f \in \mathcal{F}$ with $f : [m] \times \mathbb{N} \to \mathbb{N}$ strictly monotone increasing w.r.t. time, the second parameter.*

$$P' \in RP(P) \quad :\Longleftrightarrow \quad \exists f \in \mathcal{F} : P' = \{(i, j, f(j,t)) \mid (i,j,t) \in P\}\,.$$

A δ-relaxed schema $RP_\delta(P) \subseteq 2^{[n]\times[m]\times\mathbb{N}}$ is a relaxed schema where the set of functions $\mathcal{F}$ is restricted by $f(j,t) \leq t + \delta$, for all j, t.

There are situation where it is reasonable to allow an output schema to change the topological and chronological order at the output pins. We call these kind of schemas *general relaxed schemas* as a specific union of schemas (we restrict ourselves to reasonable schemas, where the position and time of the transmitted string can be easily determined).

3 Efficient Schemas for the Addition

3.1 Definitions

First of all, we assume that for given binary input numbers $a, b \in \{0,1\}^n$ bits a_i and b_i arrive at a logic block at the same time. In the next step this logic block computes $x_i \in \{\texttt{pro}, \texttt{gen}, \texttt{del}\}$, using the mapping $(0,0) \mapsto \texttt{del}, (0,1) \mapsto \texttt{pro}, (1,0) \mapsto \texttt{pro}, (1,1) \mapsto \texttt{gen}$, called the $\{\texttt{pro}, \texttt{gen}, \texttt{del}\}$*-representation* of a and b. Then the standard parallel prefix operation for addition has to be computed on the vector x_i to determine the result of the addition. For this we define the operator $\otimes : \{\texttt{pro}, \texttt{gen}, \texttt{del}\}^* \to \{\texttt{pro}, \texttt{gen}, \texttt{del}\}$ by $\otimes(\varepsilon) = \texttt{pro}$ for the empty string ε and for any string $w \in \{\texttt{pro}, \texttt{gen}, \texttt{del}\}^*$ define $\otimes(w\ \texttt{pro}) := \otimes(w)$, $\otimes(w\ \texttt{gen}) = \texttt{gen}$, and $\otimes(w\ \texttt{del}) := \texttt{del}$. The standard parallel prefix operator on an input $x \in \{\texttt{pro}, \texttt{gen}, \texttt{del}\}^n$ is defined by $\mathrm{PP}_\otimes(x) := y_1 \ldots y_n$ where $y_i = \otimes(x_1 \ldots x_i)$. The

sequence $(y_i)_{i\in[n]}$ directly describes the sum $a+b$.
Given the precalculated vector $(x_i)_{i\in[n]}$ we investigate the time and area complexity of the schemas bit-parallel word-serial (or **WS** for short), bit-serial word-parallel (or **WP** for short), bit-serial snakelike word-parallel (or **SWP** for short), snakelike bit-parallel word-serial (or **SWS** for short), and offset bit-parallel word-serial (or **OWS** for short) (see Fig 3). For converting a given schema into another schema for example WS into WP see e.g. [5].

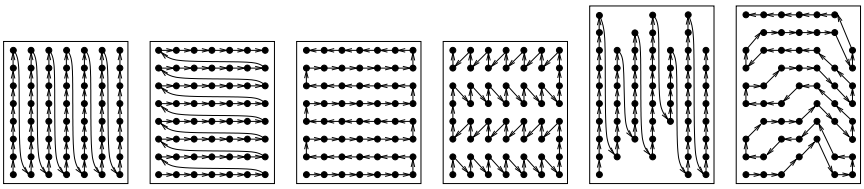

Fig. 3. Graphical description of the schemas: from left to right a) bit-parallel word-serial, b) bit-serial word-parallel, c) bit-serial snakelike word-parallel, d) snakelike bit-parallel word-serial, e) offset bit-parallel word-serial, and f) relaxed bit-serial snakelike word-parallel. In these diagrams the time is growing from the bottom to the top.

Definition 8 *Let $x_1,\ldots,x_n$ be the input or output string. For r pins and $t=\lceil\frac{n}{r}\rceil$ we define the following schemas.*

$$r\times t\text{-WS} := \left\{\left(i,\left\lceil\tfrac{i}{t}\right\rceil, 1+(i-1) \bmod t\right) \mid i\in[n]\right\} \tag{1}$$

$$r\times t\text{-WP} := \left\{\left(i, 1+(i-1) \bmod r, \left\lceil\tfrac{i}{r}\right\rceil\right) \mid i\in[n]\right\} \tag{2}$$

$$\begin{aligned} r\times t\text{-SWP} := {} & \left\{\left(i, 1+(i-1) \bmod r, \left\lceil\tfrac{i}{r}\right\rceil\right) \mid i\in[n] \text{ and } \left\lceil\tfrac{i}{r}\right\rceil \text{ is odd}\right\} \cup \\ & \left\{\left(i, r-(i-1) \bmod r, \left\lceil\tfrac{i}{r}\right\rceil\right) \mid i\in[n] \text{ and } \left\lceil\tfrac{i}{r}\right\rceil \text{ is even}\right\} \end{aligned} \tag{3}$$

$$\begin{aligned} (r,t,d)\text{-SWS} := {} & \left\{\left(i, 1+\left\lceil\tfrac{i}{d}-1\right\rceil \bmod r, (i-1) \bmod d + d\left\lceil\tfrac{i}{dr}\right\rceil\right) \mid i\in[n]\right. \\ & \text{and } \left\lceil\tfrac{i}{dr}\right\rceil \text{ is odd}\} \quad \cup \quad \{\,(i, 1+\left\lceil\tfrac{i}{d}-1\right\rceil \bmod r, \\ & \left.(i-1) \bmod d + \left\lceil\tfrac{i}{dr}\right\rceil\cdot d) \;\mid\; i\in[n] \text{ and } \left\lceil\tfrac{i}{dr}\right\rceil \text{ is even}\right\} \end{aligned} \tag{4}$$

r-OWS is a general relaxed schema with r pins with starting times $T_s(P,j)$ and lengths $t_j := T_f(P,j)-T_s(P,j)+1$, where $n=\sum_j t_j$. The ith non-empty element at pin p_j corresponds to the element x_k for $k=\sum_{\nu<j} t_\nu+i$. (5)

3.2 Upper Bounds

Theorem 1 *In the grid model the addition of two n bit numbers with input schema P_I and output schema P_O can be computed in*

a) $A\in O(n), T\in O(r+t), \Delta_s,\Delta_f\in O(r+t)$ *for $r\times t$-WS P_I,P_O,*
b) $A\in O(r^2), T\in O(r+t), \Delta_s,\Delta_f\in O(r)$ *for $r\times t$-SWP P_I,P_O,*
c) $A\in O(dr+r^2), T\in O(d+r+t), \Delta_s,\Delta_f\in O(d+r)$ *for (r,t,d)-SWS P_I,P_O,*
d) $A\in O(r\log t)$, $T\in O(r+t)$, $\Delta_s=1$, $\Delta_f\in O(r+t)$ *for $r\times t$-WS P_I and r-OWS P_O.*

Sketch of the algorithms: For all algorithms pins are located only in $(g_{0,j})_{[m_2]}$.
a) The algorithm for the WS I/O schema consists of three phases: In the first phase the algorithm stores the input of each pin p_i by using a FIFO-data structure Q_i. Furthermore, it applies $\otimes$ to all inputs received at p_i yielding $z_i=\otimes_{j=1}^{t} x_{(i-1)k+j}$. Then, it computes $\mathrm{PP}_\otimes(z_1\ldots z_r) =: k_1\ldots k_r$. Finally, it generates the output for each pin p_i by applying the prefix operator $\mathrm{PP}_\otimes$ with initial value k_{i-1} to the elements in Q_i.

d) We use a similar strategy for the OWS output schema. In first phase the algorithm counts the number c_i of leading `pro`s at each pin p_i. While this counting takes place no output is generated at p_i. If the algorithm receives a non-propagate symbol at p_i, it generates the output on-line at p_i determined by the rest of the input on p_i. As in the WS output schema, it computes the partial results z_i. In the second phase it computes $\mathrm{PP}_\otimes(z_1 \dots z_r) =: k_1 \dots k_r$. Finally, it generates c_{i+1} copies of k_i on pin p_i.
b) For the SWP I/O schema we pipeline the input through an $r \times 2r$-field. For the first row we define the internal values $z_i^0 = x_i$, which appear according to the SWP schema. In the t-th row we calculate $z_i^t = z_{i-1}^{t-1} \otimes z_i^{t-1}$. Note that diagonal connections can be simulated by an appropriate encoding, e.g. increasing the alphabet Σ. In the last row of the field no values `pro` remain unless the input contains a `pro`-chain of length at least $2r$. If $z_i^{2r} = \texttt{pro}$ then the last non-`pro` value that was computed at this logic block $g_{i,2r}$ determines the output value y_i. Using standard techniques one can fold this field such that inputs and outputs are placed on the same side of the array.
c) The algorithm for the SWS I/O schema is a straight-forward combination of the algorithms for the SWP and the WS schema. ∎

For the λ-wired grid model we omit the WS and the SWS schema since they do not provide more efficient algorithms than the WP and the OWS schemas neither in the worst case nor in the average case.

Theorem 2 *For $\lambda > 0$ in the λ-wired model addition of two n bit numbers can be performed with input schema P_I and output schema P_O as follows:*
1. *if P_I and P_O are $r \times t$-WP schemas then $A \in O(\min(n + \frac{n \log n}{\lambda},\ r \log r))$, $T \in O(t + \log r)$, and $\Delta_s, \Delta_f \in O(\log r)$.*
2. *if P_I is a $r \times t$-WS and P_O is a r-OWS schema then $A \in O(r \log t + \frac{r \log r}{\lambda})$, $T \in O(t + \log r)$, $\Delta_s \in O(1)$, and $\Delta_f \in O(\log t)$.*

The algorithms for the λ-wired model corresponds to the grid-based algorithms except that we use a tree structure as described in [4] to compute the carries of parallelly received input elements (for the WP schema) or to compute $\mathrm{PP}_\otimes(z_1 \dots z_r)$ (for the OWS schema). The algorithm presented in [4] pipelines through a tree structure to improve the area. For the OWS schema we use the carry computation tree only once to compute $\mathrm{PP}_\otimes(z_1 \dots z_r)$. Hence, we can reduce the area needed to implement the tree structure by a factor of λ. This does not help for the WP schema since we cannot compress information processed in the pipeline. The following corollary improves the AT^2 bound of $O(n \log^2 n)$ shown in [4] for high connectivity parameter λ ($\mathrm{llog}\, n := \log \log n$).

Corollary 1 *In the $\log n / \mathrm{llog}\, n$-wired-model the addition of two n bit numbers with input schema $n / \log n \times \log n$-WS and output schema $n / \log n$-OWS needs area $A = O(n\, \mathrm{llog}\, n / \log n)$ and over-all time $T = O(\log n)$, thus $AT^2 = O(n \log n\, \mathrm{llog}\, n)$.*

3.3 Lower Bounds

Here, we investigate the question whether improved I/O schemas exist which enable more efficient adders than those presented in the previous section. The only restrictions are that either the input and output behaviors are similar or that both schemas are compact. The two following theorems show that indeed the presented schemas and algorithms are optimal w.r.t. time and area. Since both Theorems use similar arguments we present a proof sketch after Theorem 4.

Theorem 3 *In the grid model any algorithm adding two n bit numbers using r pins and schema duration t has an over-all time of at least $\frac{n}{r} + \sqrt{r}$ for any input schema and $t + r$ for any one-shot compact input schema. Furthermore, it has*

1) at least area $\Omega(\min(r^2,\ n)\ -\ \delta r)$ for any one-shot compact δ-relaxed input schema and relaxed output schema of the same type;

2) at least area $\Omega(\min(r^2,\ n))$ for any one-shot compact input schema and any one-shot compact output schema;

3) at least area $\Omega(n)$ for $\alpha \geq 1$ and $2\alpha t \leq \sqrt{n}$ for any one-shot compact input schema and one-shot α-compact output schema.

Theorem 4 *For the λ-wired grid model any algorithm adding two n bit numbers using r pins and schema duration t the over-all time is at least $\frac{n}{r} + \log r$ for any one-shot compact input schema. Furthermore, it has*

1) at least area $\Omega(\min(r\ \log r,\ n)\ -\ \delta r)$ for any one-shot compact δ-relaxed input schema and relaxed output schema of the same type;

2) at least area $\Omega(\min(r\ \log r,\ n))$ for any one-shot compact input schema and any one-shot compact output schema;

3) area $\Omega(n)$ for $\alpha \geq 1$ and $2\alpha t \leq \log n$ for any one-shot compact input schema and one-shot α-compact output schema.

Proof Sketch: 1.: Let P_I be a one-shot compact input schema and $P_O \in RP(P_I)$. Let

$$A := \{\, i \mid \exists j :\ (i,j,0) \in P_I \text{ and } |\{k < i \mid \exists j :\ (k,j,0) \in P\}| > r/2 - 1 \,\}$$
$$B := \{\, i \mid \exists j, \ell :\ (i,j,\ell) \in P_I \quad \text{and} \quad \ell \geq t/2 \,\}$$
$$B_j := \{\, i \mid \exists \ell :\ (i,j,\ell) \in P_I \,\} \cap B\,.$$

Note that $|A| \geq \frac{r}{2}$. For $i \in A$ define $C_i := \{\, j \notin B \mid \forall k \in \{j+1,..,i\} :\ k \notin B \,\}$. We choose $x_i := \texttt{pro}$ for all $i \in \overline{B}$ and we select for all $i \in B$ the value of $x_i \in \{\texttt{del}, \texttt{gen}\}$ such that the inputs in B has a high Kolmogorov complexity.

Note that each output y_i with $i \in A$ either depends on an input bit in B or on a `pro`-chain C_i containing at least $\frac{r}{2}$ input elements received by the algorithm at the beginning on different pins. Hence, each of these outputs is delayed by at least $\min\{\log(r/4), t/2\}$ steps, resp. by $\min\{r/4, t/2\}$ steps in the grid model. This implies that every output element at this pin is delayed by at least this number of steps. Therefore, also the output elements corresponding to B_i will be delayed and have to be stored. Because of the high Kolmogorov complexity of the input sequence addressed by B the algorithm needs a memory of size at least $r/2 \cdot \min\{\log(r/4), t/2\}$ for the λ-wired model, resp. $\min\{r/4, n/4\}$ for the grid model.

A δ-relaxation of the input schema w.r.t. a compact schema P reduces this bound by δr, because elements addressed by B can be delayed by δ steps in the input schema.

2. and *3.*: Now let P_I, P_O be one-shot compact schemas and $i_1 < i_2 < \cdots < i_r$ be the sequence of indices of the input which arrive at time step $t-1$. Define

$$A := \{\, i_1, \ldots, i_r \,\},\quad B_1 := \{\, i \mid i_1 \leq i \leq i_{\lceil r/2 \rceil} \,\},\quad B_2 := \{\, i \mid i_{\lceil r/2 \rceil + 1} \leq i \leq i_r \}$$

$$C_k := \{\, i \mid \exists j, \ell : (i,j,\ell) \in P_I \wedge \ell < k \,\},\ D_k := \begin{cases} B_1 \text{ if } |B_1 \cap C_k| \leq |B_2 \cap C_k| \\ B_2 \text{ if } |B_1 \cap C_k| > |B_2 \cap C_k| \end{cases}.$$

For all $i \in D_k$ we choose $x_i = \texttt{pro}$ and for all $i \notin D_k$ we choose $x_i \in \{\texttt{del}, \texttt{gen}\}$ such that this sequence has a high Kolmogorov complexity. Note that the last position y_i in D_k can only be determined after $t + \log(r/2)$ steps. Assume that P_O is α-compact with $1 \leq \alpha \leq \frac{t + \log(r/2) - k}{t}$ for an appropriately chosen k. Then, any algorithm cannot

generate the first output element before time step $t + \log(r/2) - \alpha \cdot t > k$. Hence, the algorithm has to withhold the output corresponding to C_k. Note that C_k addresses the input elements received within the first k steps. Note that at least half of these are not in D_k and possess high Kolmogorov complexity. This implies $A \in \Omega(\min(n, r \cdot k))$. By choosing $k = \log(r/2) + (1-\alpha)t$ the claim follows. ∎

4 Efficient on the Average

In this section we investigate average bounds for over-all time and follow-up time. Furthermore, we measure the *area necessary with high probability (area w.h.p)* defined as follows. Consider a partition of the over-all area of an algorithm into two rectangles R_0 and R_1 with area A_0 and A_1 where all pins are adjacent to R_0. The algorithm needs area A_0 w.h.p. if for a random input of length n the communication between R_0 and R_1 consists only of special symbols β with probability $1 - n^{-c}$ for some constant c. Synchronizing output elements is an area-consuming task. For this reason, we use relaxed output schemas allowing results to be given out as soon as they are available. However, all of the here presented algorithms except the OWS schema produce compact (non-relaxed) output schemas w.h.p.

Lemma 1 *For any $c > 0$ the probability that the $\{\texttt{pro}, \texttt{gen}, \texttt{del}\}$-representation W of two uniformly chosen binary numbers $A, B \in \{0,1\}^n$ contains a contiguous propagate sequence of length $2(c+1) \cdot \log_3 n$ is bounded by n^{-c}.*

The algorithms used for averagely efficient addition are similar to the algorithms as presented in section 3 except that they generate outputs as soon as they are available. Intuitively, Lemma 1 implies that for the algorithm for the SWP schema presented above it is sufficient w.h.p. to use a $r \times O(\log n)$-field. For the OWS schema the counters c_i can be bounded by $O(\log n)$ w.h.p. For the relaxed WS and OWS output schema the algorithm has to resolve $\texttt{pro}$-chains $z_1 \ldots z_n$, which are shorter than $O(\frac{\log n}{t})$ w.h.p.

Theorem 5 *In the grid model the addition of two n bit random bits with input schema P_I and output schema P_O can be computed within time $T = t + \Delta_f$ and:*

P_I	P_O	$E[\Delta_f]$	A *w.h.p.*
$r \times t$-*WS*	*rel.* $r \times t$-*WS*	$O(t + \frac{\log n}{t})$	$O(n)$
$r \times t$-*WS*	r-*OWS*	$O(\min(\log n, t + \frac{\log n}{t}))$	$O(r \min(\llog n, \log t))$
$r \times t$-*SWP*	*rel.* $r \times t$-*SWP*	$O(\min(\log n, r))$	$O(r \min(\log n, r, t))$
(r,t,d)-*SWS*	*rel* (r,t,d)-*SWS*	$O(d + \min(r, \frac{\log n}{d}))$	$O(r \min(d + r, \log n, t))$

It turns out that the algorithm addressed by the following Lemma is a basic building block for the design of average efficient adders for λ-wired grids.

Lemma 2 *There exists an $O(n \llog n)$ area and $O(\llog n)$ time bounded algorithm in the 1-wired grid model that computes the addition of two n bit random numbers w.h.p. if the input and output is given in parallel ($n \times 1$-WP).*

Proof Sketch: We partition the input into blocks $U_i := x_{i\ell+1}, .., x_{(i+1)\ell}$ of length $\ell := c \log n$ for an appropriately chosen constant c and compute $\mathrm{PP}_{\otimes}(U_i U_{i+1})$ for each i using the algorithm of [4]. Let $y_{(i+1)\ell+1}, \ldots, y_{(i+2)\cdot\ell}$ be the suffix of the result of this computation. Lemma 1 implies that $y_1, .., y_n$ is the correct result whp. ∎

Using the worst-case algorithms one can improve time and area by replacing all subroutines computing $\mathrm{PP}_{\otimes}$ by the algorithm of Lemma 2. To guaranty correctness if a propagate-chain of length $c\log n+1$ occurs, the computation is delayed in such a case, and a worst-case algorithm takes over for the rest of the calculation.

Theorem 6 *In the λ-wired model adding two n bit random numbers with input schema P_I and output schema P_O can be computed in over-all time $T = t + \Delta_f$, where*
$E[\Delta_f] \in O(\min(\log n, t+\operatorname{llog} n))$, $A \in O(r(\min(\log t, \operatorname{llog} n) + \frac{\operatorname{llog} n - \log t}{\lambda}))$ *whp for $r \times t$-WS P_I and r-OWS P_O resp.*
$E[\Delta_f] \in O(\min(\log r, \operatorname{llog} n))$ *and* $A \in O(r \min(\log r, \operatorname{llog} n, t + \frac{t \operatorname{llog} n}{\lambda}))$ *whp for $r \times t$-WP P_I and rel. $r \times t$-WP P_O.*

For a generalization, we observe that only the algorithm constructed for the OWS output schema uses a special property of the addition. All other bounds, presented here, can be generalized to a broader class of prefix functions. The worst case upper bounds presented in Section 3 holds for any prefix function. The average bounds presented in Section 5 hold for a sub-class of these functions, the so-called *confluent prefix functions*, introduced in [17]. Furthermore, the algorithms provide the same average complexity measures if the input probability distribution is generalized to *binomial approximable distributions* as dicussed in [13]. The lower bounds presented in Section 4 also apply for *diffluent prefix functions* introduced in [13]. Because of space limitation we omit the definitions of diffluent and confluent prefix functions.
As already noted, the algorithms for the offset schemas cannot be applied directly to general prefix functions. The following Theorem shows that there exists an alternative average efficient algorithm using this I/O schema for confluent prefix functions.

Theorem 7 *In the grid model the computation of confluent prefix function for a random input of length n with input $r \times t$-WS input schema and r-offset output schema can be computed within area w.h.p. $O(r \min(\sqrt{\log n}, \sqrt{t}))$ and expected follow-up time $O(\min(\log n, t + \frac{\log n}{t}))$. In the λ-wired grid model the corresponding area bound w.h.p. is $O(r(\min(\sqrt{\log n}, \sqrt{t}) + \frac{\operatorname{llog} n - \log t}{\lambda}))$ while the expected follow-up time is $O(\min(\log n, t + \operatorname{llog} n))$.*

The key idea of the proof is to replace the counters of the corresponding adders by an area efficient data structure for storing intermediate values.

5 Conclusions

The results presented above can be used to optimize the average behavior of an FPGA if an appropriate schema is chosen. Table 1 summarizes the results of Theorem 5. In Table 1 we optimize the time implied by the output schema. If schemas have equal asymptotic expected time behavior we present the one with smaller area. The expected follow-up time of the relaxed SWS schema is minimized by choosing $d = \min(\sqrt{\log n},\ r,\ t)$. From Theorem 6 it follows for the λ-wired grid model there is no asymptotical difference in the expected over-all time $E[T] = t + E[\Delta_f]$ for the relaxed WP and OWS output schema. Only for $t \in o(\log n)$ and connectivity parameter $\lambda \in \omega(1)$ the OWS schema provides a more area efficient algorithm than the relaxed WP. For other parameters it seems that the relaxed WP schema should be preferred. Note that the corresponding algorithm produces a compact (non-relaxed) WP output schema w.h.p.

$r \in O(\sqrt{\log n})$	$r \in \omega(\sqrt{\log n}) \cap o(\frac{n}{\sqrt{\log n}})$	$r \in \Omega(\frac{n}{\sqrt{\log n}}) \cap o(n)$	$r \in \Omega(n)$
rel. SWP rel. SWS	rel. SWS	OWS	rel. WS, rel. SWP rel. SWS, OWS
$E[\Delta_f] = O(\frac{n}{t})$ $A \in O(r^2)$	$E[\Delta_f] \in O(\sqrt{\log n})$ A whp$\in O(r \min(\log n, r, t))$	$E[\Delta_f] \in O(\frac{\log n}{t})$ A whp$\in O(r \log t)$	$E[\Delta_f] \in O(\log n)$ $A \in O(n)$

Table1: Output schemas which allow expected time efficient algorithms for grids.

References

1. Actel Corporation, *ProASIC* $^{\text{TM}}$ *500K Family*, Product Spec., October 2000.
2. Atmel Corp.,*AT 40K FPGAs with FreeRAM* $^{\text{TM}}$, Rev. 0896B-01/99, Prod. Spec., Jan. 1999.
3. M. Blaze, W. Diffie, R. Rivest, B. Schneier, T. Shimomura, E. Thompson, M. Wiener, *Minimal key lengths for symmetric ciphers to provide adequate commercial security: A report by an ad hoc group of cryptographers and computer scientists*, 1996, www.bsa.org.
4. R. Brent, H.T. Kung, *A Regular Layout for Parallel Adders*, IEEE Transaction on Computers, C-31, 1982, 260-264.
5. W. P. Burleson, L. L. Scharf, *Input/Output Design for VLSI Array Architectures*, Proceedings VLSI'91, 8b2.1-10, 1991.
6. W. Carter, K. Duong, R. Freeman, H. Hsieh, J. Ja, J. Mahoney, L. Ngo, S. Sze, *A User Programmable Gate Array*, Proc. CICC'86, 1986, 233-235.
7. K. Compton, S. Hauck, *Configurable Computing: A Survey of Systems and Software*, Northwestern University, Dept. of ECE Technical Report, 1999.
8. I. David, R. Ginosar, M. Yoelli, *An Efficient Implementation of Boolean Functions and Finite State Machines as Self-Timed Circuits*, ACM SIGARCH, 1989, 91-104.
9. K. El-Ayat, *A CMOS Electronically Configurable Gate Array*, Proc. ISSCC, 1988, 76-77.
10. A. El Gamal, *An Architecture for Electronically Configurable Gate Arrays*, Proc. CICC'88, 1988, 15.4.1- 15.4.4.
11. H. Hsieh, K. Duong, J. Ja, R. Kanazawa, L. Ngo, L. Tinkey, W. Carter, and R. Freeman, *A Second Generation User Programmable Gate Array*, Proc. CICC'87, 1987, 515-521.
12. A. Jakoby, *Die Komplexität von Präfixfunktionen bezüglich ihres mittleren Zeitverhaltens*, PhD dissertation, University of Lübeck, 1998.
13. A. Jakoby, *The Average Time Complexity to Compute Prefix Functions in Processor Networks*, Proc. 16th STACS, 1999, 78-89.
14. A. Jakoby, C. Schindelhauer, *On the Complexity of Worst Case and Expected Time in a Circuit*, Proc. 13th STACS, 1996, 295-306.
15. A. Jakoby, R. Reischuk *Average Case Complexity of Unbounded Fanin Circuits*, Proc. 15th Conference on Computational Complexity (CCC), 1999, 170-185.
16. A. Jakoby, R. Reischuk, C. Schindelhauer, *Circuit Complexity: from the Worst Case to the Average Case,* Proc. 26th SToC, 1994, 58-67.
17. A. Jakoby, R. Reischuk, C. Schindelhauer, S. Weis *The Average Case Complexity of the Parallel Prefix Problem,* Proc. 21st ICALP, 1994, 593-604.
18. R. Ladner and M. Fischer, *Parallel prefix computation*, J. ACM, 27 (4), 1980,831–838.
19. J. Reif, *Probabilistic Parallel Prefix Computation,* Comp. Math. Applic. 26, 1993, 101-110.
20. R. Tessier, W. Burleson, *Reconfigurable Computing for Digital Signal Processing: A Survey*, to appear in Y. Hen Hu (ed) *Programmable Signal Processors*, Marcel Dekker Inc., 2001
21. J. Ullman, *Computational Aspects of VLSI*, Computer Science Press, 1984.
22. J. Vuillemin, P. Bertin, D. Roncin, M. Shand, H. Touati, P. Boucard, *Programmable Active Memories: Reconfigurable Systems Come of Age*, IEEE Trans. VLSI Systems 4 (1), 1996.
23. Xilinx Corp., *XC4000E and XC4000X Series Field Programmable Gate Arrays*, Prod. Spec., Version 1.6, 1999.
24. Xilinx Corp., *Virtex-II Platform FPGA Data Sheet (DS031)*, Prod. Spec., Ver. 1.5, 2001.