A specification for a reconfigurable optoelectronic VLSI processor suitable for digital signal processing

D. Fey, B. Kasche, C. Burkert

Friedrich-Schiller-University Jena, Department for Computer Architectue and -communication
Ernst Abbe Platz 1-4, 07745 Jena, Germany

O. Tschäche

Friedrich-Alexander-University Erlangen/Nürnberg, Department for Computer Structures
Martensstr. 3, 91058 Erlangen, Germany

(April 1, 1997)
revised version: September 25, 1997

Keywords: smart pixel, optoelectronic VLSI, digital signal processor, CORDIC

Abstract

A concept for a parallel digital signal processor based on optical interconnections and optoelectronic very large scale integrated circuits is presented. It is shown that the right combination of optical communication, architecture and algorithms allows a throughput that outperforms pure electronic solutions.

The usefulness of low-level algorithms from the add-and-shift class is emphasized. These algorithms lead to fine-grain, massively parallel on-chip processor architectures with high demands for optical off-chip interconnections. A comparative performance analysis shows the superiority of a bit serial architecture. This architecture is mapped onto an optoelectronic 3-D circuit and the necessary optical interconnection scheme is specified.

1. Introduction

At present one of major problems in current VLSI systems is limited bandwidth because of too few and too slow external pins[1]. Optoelectronic very large scale integrated (VLSI) circuits[2–4] using high-dense optical interconnection[5–7] schemes offer the potential to eliminate these bottlenecks. The success of optoelectronic VLSI systems depends largely upon applications with a need for high input/output (I/O) bandwidth. Digital signal processing (DSP) is one of such bandwidth intensive computational tasks. Typical signal processing tasks, as for example medical image processing and detection of radar signals, is characterized by high data accesses as well as high computing performance. An efficient solution for a DSP requires the right combination of processor architecture, low-level algorithms and powerful off-chip communication.

In this paper we present an architecture for a reconfigurable parallel digital signal processor solving the communication bottleneck by using optical interconnections. The reconfigurability in our solution is realized by setting the content of our processor state machine to a specific value. Depending on this value our processor carries out in hardware one of eight different elementary functions, which are often used in signal processing tasks. To these functions belong exponential function, logarithm, sine, cosine, arc tangent, square root, multiplication and division. Signal processors often use convergence procedures as CORDIC or so-called bit algorithms for a hardwired solution of such functions. As long as such algorithms are implemented in a single processor this makes no serious difficulties with current microelectronics technology. In contrast to this a parallel implementation of such reconfigurable DSP elements is difficult to achieve, due to area consuming units. The reason for that is that area consuming units as tables and barrel shifters are needed. Tables and barrel shifters are fundamental for an efficient implementation of such low-level algorithms. The table stores constant values needed in the algorithm. The barrel shifter is necessary to carry out bit shifts about variable bit lengths in constant time. Moreover, long on-chip interconnections are necessary to distribute data that have to be broadcasted. In the paper we will show that some of the mentioned problems are avoidable by means of parallel optical off-chip interconnections. Thus using optoelectronic technology an implementation of a massively parallel DSP system becomes possible.

As solution for the optoelectronic VLSI (OE-VLSI) circuits we laid our focus on a H(ybrid)-SEED technology. This technology seems to be one of the most mature OE-VLSI techniques so far, at least if we judge this in the sense of public availability via the CO-OP consortium. However, our architecture proposal can be mapped also without difficulties onto OE-VLSI circuits based on vertical-cavity surface-emitting laser (VCSEL) arrays[8]. For the H-SEED technology we determine the computing performance to expect versus the clock cycle and compare this result with existing pure-electronic DSP systems. Since we aspire a fine-grain processor architecture with as much as possible DSP elements on chip, we consider fault-tolerant mechanisms for essential. A first step towards this direction is the introduction of a fault model in the VHDL description of our DSP system to investigate optical cross talk between neighbored optical detectors.

The paper is organized as follows. Section 2 reviews the functionality of the low-level algorithms we favor for our parallel optoelectronic DSP elements. Moreover, we present in Section 3 different alternatives for architectures implementing thoselow-level algorithms. The structure of the processing elements (PEs) inclusive their interconnection requirements are specified in more details in Section 4. In Section 5 the different architecture models are evaluated by a theoretical analysis of their computing performance to expect. In Section 6 that architecture model, we evaluated as the best one, is mapped onto an OE-VLSI circuit. Furthermore the corresponding optical interconnection scheme is specified. The mapping process was supported by a high-level synthesis of a VHDL description of our processor system using the design tool Synopsys[9]. The usefulness of a high-level description like VHDL is demonstrated in Section 7. There we present a fault model introduced in the VHDL description to simulate the effects of optical cross talk between neighbored detectors. Finally we finish with a summary and an outlook.

2. Algorithm study

In this section we give a short overview of the used class of algorithms and we will take a closer look to the basic ideas of the mathematical background. Our research aimed at a development of algorithms well suited for smart pixel architectures. That means, we were looking for algorithms which match very well the boundary conditions given by the optoelectronic hardware. To exploit the potential of optical interconnections most efficiently we are favoring fine grained architectures. These architectures are characterized by a high need for off-chip communication. Hence, the PEs should be as small as possible to integrate as much as possible PEs onto one chip. The benefits from optical interconnections for fine grained smart pixel PEs were also stressed in literature in the past[10–13].

A. Add-and-shift algorithms

Our aim was to find a uniform optoelectronic hardware to calculate standard functions like sine, cosine, arc tangent, square root, logarithm, exponential, multiply and division function. This can be done by a Taylor series development using floating point multiplication units (FPMUs). Since FPMUs need comparatively much area on chip we didn't pursued this possibility.

Instead of using FPMUs we investigated different architectures based on add-and-shift operations. Those algorithms are using only simple operations such as *adding*, *shifting* and *multiplexing*. CORDIC algorithms, being well known in signal processing, belong to the class of add-and-shift algorithms. They have been developed and investigated for a long time especially for trigonometric functions[14]. CORDIC is an iterative procedure as well as so called bit algorithms, which also belong to the class of add-and-shift algorithms. They are called bit algorithms because within each iteration a new bit, starting from the most significant bit position, becomes valid. Bit algorithms are more appropriate for the calculation of square root, multiplication, division, exponential and logarithmic function.

By comparing bit algorithms and CORDIC we have observed that the calculation scheme of CORDIC shows a more uniform structure. But for the non trigonometric functions it is sometimes necessary to use the CORDIC scheme twice, i.e. we need $2n$ iteration steps. On the other hand bit algorithms deliver the results in n steps. But they show lower efficiency to calculate trigonometric functions.

Unfortunately in literature a uniform scheme to integrate all the eight functions, mentioned above, was not found. It was our task to modify them in such a way that they could be mapped onto a fine grained optoelectronic hardware structure. We combined bit and CORDIC algorithms and got as result the scheme shown in Fig. 1.

To our knowledge, such a uniform algorithm scheme, which allows to carry out eight different standard functions simultaneously, can not be found in literature.

We developed these algortihms especially for an optoelectronic smart pixel architecture. A uniform algorithm structure will lead to a regular hardware setup of a PE, and additionally, to a more space-invariant optical off-chip interconnection scheme. The higher the space invariance is, the more dense the interconnections should be. We pursued this approach to exploit in our DSP architecture one strength of optics over electronics, the high space bandwidth. This will lead to a paralllelism that is not possible to achieve in electronics. In electronics there is no stringent requirement for uniform structures. Mostly barrel shifters and fast adders are used. This is the way to prefer if the intention is just to integrate one or two processors. But our intention is to implement a massively parallel DSP. This DSP must be realized as OE-VLSI circuit and since this technology is young, the optical interconnection scheme should be as simple as possible.

The calculations, shown in Fig. 1, are carried out on a triple (x, y, z). The idea of our algorithms is that the y component goes to zero and the other components converge to the searched function values simultaneously. The procedure starts with `(a)` a shift operation on two components of the triple. Next we have to perform `(b)` three add operations on x, y and z. Finally `(c)` we check the value of y to measure the progress of convergence. Within the CORDIC algorithms we have a two side convergence. That means, if y is below zero we have to increase it, i.e. we have to add in the next iteration. Otherwise, if y is above zero we have to carry out a subtraction. Within the bit algorithms there is a one side convergence. If y is below zero the calculations carried out in `(a)` and `(b)` are obsolete. Hence, we have to continue with the old values in the next iteration.

The concrete iteration scheme for the eight functions, which we have developed, is shown in Tab. 1. The first five algorithms belong to the class of bit algorithms. The last three are CORDIC procedures. The multiplications with 2^{-i} are replaced by shifts to the right. The values for $\arctan(2^{-i})$ and $\log(1+2^{-i})$ are precalculated constant values, as well as the initial value $\kappa = \prod_{i=0}^{n-1}(\sqrt{1+2^{-2i}})^{-1}$ required in the CORDIC procedures.

The square root function seems to be outside of a uniform scheme due to an additional add operation. The addend "$+2^{-2(i+1)}$", which is to be seen in the y variable, can be realized by a *bit set* operation. This is possible due to the fact, that the $(2i+2)^{th}$ bit is always zero, when the add operation is performed.

If the condition given in the last column of Tab. 1 is fulfilled then for the first five functions the calculation keeps valid, otherwise not. In case of the last three functions the calculation is always done, but the above algebraic signs are used for the next iteration if the result of the question is true. But it is still a uniform structure as it shows Fig. 1. Hence, the inquiry can be always done at the end of each iteration on the y variable of the just finished iteration.

B. The *logarithm* function as an exapmle

As an exapmle for all eight functions we want to show the derivation of the algorithm calculating the function values of the logarithm $f(\varphi) = \log(\varphi)$. For that we use the additivity of the logarithm values of each factor, if the logarithm of a product is to carry out.

$$\log(\prod_{i=1}^{n} y_i) = \sum_{i=1}^{n} \log(y_i) \tag{1}$$

Using (1) it is practical to express φ as a product of common values. So we start with the basic approach

$$\varphi = \prod_{k=1}^{n} \alpha_k^{-\delta_k} \quad , \tag{2}$$

whereby $\alpha_k = (1+2^{-k})$ and $\delta_k \in \{0,1\}$. Rewriting (2) we get

$$1 = \varphi \prod_{k=1}^{n} \alpha_k^{\delta_k} \quad . \tag{3}$$

Consequently we have (2) fulfilled if and only if (3) is fulfilled as well. Hence, we can define y_n with

$$y_n := \varphi \prod_{k=1}^{n} \alpha_k^{\delta_k}. \tag{4}$$

To approximate φ the series $(y_k)_{k=1}^{n}$ shall converge to 1. Rewriting (3) we deduce that $y_{n+1} = y_n \cdot \alpha_{n+1}$ and finally

$$y_{n+1} = y_n + y_n \cdot 2^{-(n+1)} \quad . \tag{5}$$

By approximating φ with the product $\prod_{i=1}^{n} \alpha_k^{-\delta_k}$ there will be a residue $R(\varphi, n)$ such that

$$\varphi = \prod_{k=1}^{n} \alpha_k^{-\delta_k} + R(\varphi, n) \quad . \tag{6}$$

Applying the logarithm to both sides we get

$$\log(\varphi) = \log(\prod_{k=1}^{n} \alpha_k^{-\delta_k} + R(\varphi, n)) \quad . \tag{7}$$

If y_{n+1} goes to 1 it is ensured that the residue $R(\varphi, n)$ goes to zero, such that

$$\begin{aligned} \log(\varphi) &= \log(\prod_{k=1}^{n} \alpha_k^{-\delta_k}) \\ &= -\sum_{k=1}^{n} \log(\alpha_k^{\delta_k}) \quad . \end{aligned} \tag{8}$$

So it is self-evident to define x_n as this sum, such that we get

$$x_{n+1} := x_n - \log \alpha_n^{\delta_n} \quad . \tag{9}$$

Since the series $(y_k)_{k=1}^n$ is a monoton increasing series, that means if y_{n+1} is greater than 1, the last approximation step would be irreversible. Hence, the last iteration step has to be canceled. This is expressed in a formula language by

$$\delta_k := \begin{cases} 0 & \text{if } y_{k+1} > 1 \\ 1 & \text{else} \end{cases} \tag{10}$$

We still have to determine the initial values for x_k and y_k. Since we add in the x-variable the precalculated values of a table value we have to initialize x_0 with zero. For the y-variable it holds the following, due to the validity of (3) we get $y_1 = \varphi(1 + 2^{-1})$ and deduced by (5) $y_1 = y_0 + y_0 \cdot 2^{-1}$, such that we get

$$x_0 := 0 \qquad\qquad y_0 := \varphi \tag{11}$$

Since this algorithm inquires about y greater than 1 it is not the used version. We selected another algorithm, using the condition y greater than 0, but its derivation is more complicated and longer. It can be read in literature[15,16]. In a more or less complicated way all the other 7 algorithms for standard functions can be derivated. All of them force to a uniform hardware structure[12].

3. Architecture model for the DSP System

As can be seen in our algorithms of Tab. 1 the DSP system requires only simple operations as *shifting*, followed by *adding* and finally *checking a condition*. For that we use an array of PEs (see Fig. 2) which will be more specified in the next section. In each row of this array exactly one iteration of our algorithms is carried out. So n rows are required, each one upon the other. The communication structure is a `NEWS` structure, i.e. to the 4 points of the compass. Within one row there are also long lines (not shown in Fig. 2) to carry out a data transfer over a distance of two or more processing element positions.

For our DSP architecture we want to investigate a bit parallel and a bit serial approach.

A. Bit parallel approach

For the bit parallel processing we focused on two alternatives for the addition process: a ripple carry adder (RCA) and a conditional sum adder (CSA).

The most simple way to add two numbers would be a RCA. The carry bit is always given to the next bit position to the left (see Fig. 3). So the PEs shown in Fig. 2 correspond to 1-bit full adders. The functionality is shown in (12).

$$\begin{aligned} s &:= a_i \text{ xor } b_i \text{ xor } c_{i-1} \\ c &:= a_i b_i \text{ or } a_i c_{i-1} \text{ or } b_i c_{i-1} \end{aligned} \tag{12}$$

All communication takes place only between next neighbors. So a communication over a longer distance is not required. But the whole add operation takes n steps to finish. An additional step is necessary to carry out the shift operation. To process the whole algorithm it takes

$$\text{steps} = n \cdot (n + 1) \tag{13}$$

This disadvantage of n steps per addition is avoided by using a conditional sum adder, but for the price of a more complicated interconnection scheme. The CSA exploits the hardware most efficiently in relation to the time necessary for one add operation[17,18]. Although an adder based on carry look ahead techniques can carry out an addition in constant number of time steps a CSA has a better gate per bit ratio. The same holds for look-ahead adders with logarithmic time costs[17,19]. Hence, we don't consider look-ahead techniques since for massively parallel structures using a CSA will lead to better throughput rates. The general idea behind a CSA is to carry out for each bit 1-bit additions at the beginning, considering both no carry occurs (14) and a carry (15) occurs from the nearest right position.

$$\begin{aligned} S0 &= \overline{a}b + a\overline{b} \\ C0 &= ab \end{aligned} \tag{14}$$

$$\begin{aligned} S1 &= ab + \overline{a}\overline{b} \\ C1 &= a + b \end{aligned} \tag{15}$$

Afterwards the valid bits are selected by multiplex operations.

$$\begin{aligned} S0_{out} &= \overline{C0_{in}}S0 + C0_{in}S1 \\ C0_{out} &= \overline{C0_{in}}C0 + C0_{in}C1 \\ S1_{out} &= \overline{C1_{in}}S0 + C1_{in}S1 \\ C1_{out} &= \overline{C1_{in}}C0 + C1_{in}C1 \end{aligned} \tag{16}$$

The multiplexing starts in blocks of one bit width. The width of a block increases consequently from 1 bit width to 2,4,8,16 and so on (see Fig. 4). Finally one operation is finished in logarithmic time, exactly speaking after $(1 + \operatorname{ld} n)$ steps.

The interconnections of the modules with such a functionality yield the sum of a and b at $S0$ and the difference of a and b at $S1$, assuming the 2-complement representation of the numbers. The most significant bit (MSB) is on the left and the least significant bit (LSB) is on the right. For each bit position there is one PE. In this case each PE of Fig. 2 realizes the half adder functionality shown in (14) and (15), and the multiplexer operation shown in (16).

B. Bit serial approach

This approach of the bit and CORDIC algorithms integrates a complete iteration in one PE combining the operations *adding, shifting* and *checking a condition*. In contrast to the bit parallel approach the PEs expect their data in a serial fashion.

The functionality of one row of Fig. 2 is realized by one PE. Such a PE consists of a full adder followed by a shift register to store the sum bits of the add operation. A vector of n vertically arranged PEs calculates exactly one of the eight functions of Tab. 1. Hence, several of these vectors arranged side by side, can execute different functions simultaneously. The input bits are fed into the full adder step by step starting with the LSB. The sum bit is stored in the first flip flop of the shift register and the carry bit goes to the carry flip flop (see Fig. 5). In the next step each bit in the shift register is moved to the left. All PEs of a vector are pipelined. After finishing the add operation for the MSB it is possible to decide the multiplex condition (see Tab. 1 right most column). To decide whether the value of the corresponding variable is greater or below zero we just have to check the sign bit. The following shift can be realized by a so called memory grabber functionality. Due to calculating only one iteration within each PE the grab position is clearly determined and the shift operation can be realized with fixed interconnections.

4. Structural description of one PE

A. Bit parallel method

The specification for one PE for a bit parallel approach using the conditional sum adder is described in this section.

We have already shown realizing a PE takes only hardware for shifting, adding and a control unit. To prevent long lines on chip we want to use optical interconnections. Long lines are necessary for the realization of the bit shift operation, the transfer of the carry bits and the sign bit. Then we need less area and can operate with a higher clock cycle.

The PEs require one optical receiver (OR) and one optical transmitter (OT) for the shift and for the control unit. The adder takes two OT and two OR for its functionality (see Fig. 6). Each row is responsible for one iteration. So after finishing the calculation of one iteration the result is transferred to the next neighbored PE in the north. Since this is a next neighborhood communication this can be done electronically. Due to the principles of a CSA we have to transfer the carry bits $C0$ and $C1$ (see (14), (15), and Fig. 4) within a row and not only to the next neighbors. The communication paths, running always from a less significant bit to a more significant bit (see Fig. 7), shall be optically realized. We also need a shift operation. The characteristic within our approach is, that there is always within one row the same distance of shift positions.

Depending on the sign bit of y (see Tab. 1) the control unit of each PE has the task to select the right operands for the add and shift unit. Due to the 2-complement representation of the numbers the leftmost PE of a row can determine whether the value of y is less than zero. So this PE can broadcast the sign bit over the whole row to the right to the control units of the other PEs (see Fig. 7). Then all other PEs know the result of the selection condition. This allows to determine for the functions (1..5) of Tab. 1 if the old or the new values are valid. For all the other functions the operator if we have to add or to subtract is inverted or can be kept. This was already described in more details in Section 2 A.

The necessary optical wiring for the bit shifting, the transfer of the carry bits and the sign bit is shown in

Fig. 7 schematically. The optical interconnections can be realized for example in a glass substrate or waveguide, which is located above the electronic PE array. Using optical interconnections has the following advantages. The transfer of the carry bits in a kind of segmented busses allows to save hardware because the multiplexer's hardware for each PE (see Fig. 4) is needed only once. Furthermore realizing the bit shift and the sign bit transfer in optics saves area since long lines are not necessary anymore.

As mentioned above we need $(1 + \mathrm{ld}\, n)$ steps for an addition. To carry out the shift operation we need two further steps. Hence, we get finally for the whole calculation

$$\text{steps} = n \cdot (\mathrm{ld}\, n + 3) \quad . \tag{17}$$

B. Bit serial method

Next we explain in more detail the setup of the PE of the bit serial architecture approach (see Fig. 8).

The PE needs 20 electrical ports for I/O, 10 inputs and 10 outputs, and 3 optical inputs serving the PE with the global clock and the values of precalculated constants for $\log(1 + 2^{-i})$ and $\arctan(2^{-i})$ (see Section 2 A). During the first clock cycle the input ports are used to select the function, which the PE should do on the following bit stream. When the complete stream is read, the PE generates at its output ports the code, which programs a following PE to select the right function. In the next cycle the calculated operands are shifted out. Since the compatibility between input and output ports all PEs can be consecutively connected to build the complete iteration scheme as shown in Fig. 8. The PEs are announced 'fixed' because every PE only works at the iteration stage it was build for. Hence, we also need no dynamic shift operations due to the fixed number of bits to be shifted. Since the shift operation can be implemented with fixed interconnections it will not take more time than a single ripple carry step. The shift operation is hardwired instead of using a multiplexer with the number of inputs corresponding to the number of bits in the mantissa. So, PE fixed to stage 1 will not calculate correct results for any other iteration stage than number 1. The advantage of fixing the PE to an iteration stage is an optimization leading to significantly less chip area.

Additional to the 'fixed' attribute all PEs work synchronously. This approach allows the usage of one table generator for all PE vectors shown in Fig. 5. Internally the PE mainly consists of eight shift registers holding the operands α, β and x, y and z with their corresponding values of the previous iteration. The shift registers for the new calculated operands x, y and z get their input from an add/sub unit. This unit adds or subtracts the two input signals. A multiplexer with 8 inputs selects the operand according to the programed function. The operands are shifted in with the LSB first. After the MSB initiates the last add/sub operation, the controller checks the selection condition and switches the multiplexers, so that they select the right value shifted out to the next PE. The *sel* port is used to select the add or sub operation for the CORDIC algorithms in the next stage. The *sync* port works like a reset for the controller and is activated everytime the PE receives the start of a new calculation. The PEs operate fully pipelined meaning that the input stream feeds new data into the PE without any wait states. Therefore, the calculation of a function needs one clock cycle for each bit in the mantissa plus one additional clock cycle for the programming of the PE. The result has a latency of the number of iterations n times the number of bits in the mantissa n plus 1.

$$\text{steps} = n(n + 1) \tag{18}$$

5. Performance comparison of the three architecture models

As we have seen there are different possibilities for the realization of architectures in which the presented low-level algorithms are hardwired implemented. The maximum throughput Θ we can achieve in such an architecture is expressed by the relation of the number of parallel operations p versus the number of necessary calculation steps s for one iteration.

$$\text{throughput}\Theta = \frac{\#\ \text{parallel calculations}}{\#\ \text{steps per one iteration}} = \frac{p}{s} \tag{19}$$

Our ultimate aim is to find that model that delivers the highest computing performance with regard to throughput. To find an answer we determine the necessary number of steps for one iteration at first. Tab. 2 shows this for the different models. We assumed an effective wordlength of $n + \mathrm{ld}\, n$ instead of n to get a bit precision of 2^{-n}. Hence, we have to replace all n in (13), (17) and (18), which correspond to the wordlength n, by $n + \mathrm{ld}\, n$ to get the number of steps for the different architecture models.

The number of calculations we can carry out in parallel depends on the number of PEs that are needed in each architecture model to carry out one calculation. This number is different for the three architecture models because we need one PE for one iteration in the bit serial case (*bs*) and we work with one PE per bit in both bit parallel architecture models (*bpRCA* and *bpCS*). Of course the PEs are different in size. Hence, we get the formula in the right most column of Tab. 2 for the number of parallel calculations p, if A_{bs}, A_{bpRCA} and A_{bpCS} are the corresponding sizes for one PE and A_{chip} is the available circuit area.

With the expressions of Tab. 2 it is now possible to calculate the throughputs for the different architecture models. But this presumes that we know the necessary sizes for the PEs. These are technology depending figures we don't know so far. To compare the throughputs independently of the technology we consider the relations of the throughputs for the *bpRCA* case versus the *bs* case (20a), the *bpCS* case versus the *bs* case (20b), and the *bpRCA* case versus the *bpCS* case (20c).

$$\begin{aligned}\Theta_{bpRCA/bs} &= \frac{\Theta_{bpRCA}}{\Theta_{bs}} = \frac{A_{bs}}{A_{bpRCA}} \cdot \frac{1}{\mathrm{ld}\,n + n} \qquad (20a)\\ &= \varrho \cdot \frac{1}{n + \mathrm{ld}\,n}\\ \Theta_{bpCS/bs} &= \frac{\Theta_{bpCS}}{\Theta_{bs}} \qquad (20b)\\ &= \frac{A_{bs}}{A_{bpCS}} \cdot \frac{\mathrm{ld}\,n + n + 1}{(n + \mathrm{ld}\,n)(\mathrm{ld}(n + \mathrm{ld}\,n) + 3)}\\ &= \sigma \cdot \frac{\mathrm{ld}\,n + n + 1}{(n + \mathrm{ld}\,n)(\mathrm{ld}(n + \mathrm{ld}\,n) + 3)}\\ \Theta_{bpRCA/bpCS} &= \frac{\Theta_{bpRCA}}{\Theta_{bpCS}} \qquad (20c)\\ &= \frac{A_{bpCS}}{A_{bpRCA}} \cdot \frac{\mathrm{ld}(n + \mathrm{ld}\,n) + 3}{\mathrm{ld}\,n + n + 1}\\ &= \tau \cdot \frac{\mathrm{ld}(n + \mathrm{ld}\,n) + 3}{\mathrm{ld}\,n + n + 1}\end{aligned}$$

For that, these relational throughput values are expressed in dependence on the relational PE sizes ϱ, σ and τ. Although these relational PE sizes are still technology dependent figures, they help us to find a first answer to the question, which architecture delivers the highest throughput. This is demonstrated with the curves for $\Theta_{bpRCA/bs}, \Theta_{bpCS/bs}$, and $\Theta_{bpRCA/bpCS}$, shown in Fig. 9 for a wordlength $n = 32$.

For example, the *bpRCA* architecture model offers only a higher throughput versus the *bs* architecture, if the PE size of the *bs* architecture will be about 40 times greater than the PE in the *bpRCA* model. A different situation is given for the comparison between the *bpCS* and the *bs* model. As long as the PE size in the bs case is not about eight times greater than the PE in the *bpCS* model, the *bs* architecture is predominant versus the *bpCS* architecture model. A similar relationship yields for the comparison *bpCS* to *bpRCA*. Only if the size of the PE, which exploits the conditional sum algorithm, is more than five times greater than the PE in the *bpRCA* model, the more simple ripple carry architecture *bpRCA* is advantageous.

Hence, by using the relational PE sizes we have reduced the question, which architecture is the most efficient one, to a comparison of PE sizes. This comparison we carry through with the corresponding gate numbers for each PE. These gate numbers were determined by a high-level synthesis of a VHDL description of our PEs. Due to the bad theoretical results of the bit parallel method using a ripple carry adder, this model is not taken into account any more. A synthesis for the bit serial (*bs*) approach with the design analyzer of Synopsys[9] generated a gate level model with 267 cells of a standard library. For the bit parallel architecture model using the CSA (*bpCSA*) this number was 1558 cells.

Consequently we favour the bit serial approach since it delivers the highest throughput.

6. Mapping onto optoelectronic hardware

In this section we discuss how the *bs* approach can be realized with real optoelectronic hardware. We are interested in the costs for the logic measured by an estimated transistor number. The task of the optical interconnection scheme is the broadcasting of the necessary constant values and the clock signal. There are two reasons why we would realize these links by optical means. The first is to avoid long on-chip interconnections because they increase the clock cycle time. This concerns the clock signal line itself and the distribution of the constant bits within one row of the PE array. Furthermore chip area is saved if the memory for the constant is realized outside the circuit and the data can be fetched by a parallel data access.

The necessary fan-out for the optical broadcast links is determined by the number of PEs that can be integrated in one row. To determine this number it is necessary to estimate the area requirements for one PE. In the *bs* model we need six shift registers to store the values for x, y, z and their addends. For a mantissa length with

24 bits and an accuracy of 2^{-24} we would need 29 bits. Two more bits are necessary to avoid overflow and an additional one to store the sign bit. Hence, we need 6x32=192 flip flops, which can be realized as dynamic delay flip flops (D-FF). A single dynamic D-FF needs in CMOS 8 transistors, if they are cycled continuously with a high clock rate. Then we need 1536 transistors for the memory.

Besides, we have to integrate the addsub, the control and the two 2-to-1 multiplexers, and the 8-to-1 multiplexer (see Fig. 8). Realized as complex gate the full adder in the addsub unit needs 28 transistors. Including the costs for generating the complement and a further multiplexer component we need about 50 transistors for the addsub unit. Using transmission gates the 2-to-1 multiplexers require 8 and the 8-to-1 multiplexer needs 28 transistors. The control unit consists mainly of a counter and a finite state machine to reconfigure the addsub unit depending on the function we would like to execute. The control unit requires about 120 AND and OR gates in the VHDL specification. The cost for that is estimated with about 500 transistors. So, let us summarize the total transistor costs. Since the addsub and the multiplexer units exist three times, $3 \cdot (36 + 50) = 258$ transistors are necessary, plus the transistor costs for control unit and memory, we receive finally about $258 + 500 + 1536 = 2294$ transistors. Furthermore we assume about 10% additional area for wiring. So we get about 2500 transistors for one PE.

In a current design[20], which was made for a H-SEED circuit offered by the CO-OP consortium, we obtained an average transistor size of about $42\mu m^2$. This current design represents a good mix, because 310 transistors were included in an area of $120 \times 110\mu m$ containing memory, logic gates, wires and also unused chip area. The technology was a $0.5\mu m$ CMOS process. Assuming a $0.35\mu m$ technology, which is offered by the current CO-OP foundry run, we can double our transistor number. This is possible according to the relation, that the number of devices on chip increases by the square of the scaling factor $\alpha = \frac{0.5}{0.35} = 1.43$. This would lead to $21\mu m^2$ average transistor size. For our first order estimation we will calculate with an average size of $25\mu m^2$ per transistor. Hence, the area for one PE is then $2500 \cdot 25\mu m^2 = 62500\mu m^2$. This would fit exactly in an area of $167 \times 375\mu m$ size, which allows to integrate an array of 27×60 PEs on a chip with $1.01 \times 1cm$ area. That means, we can execute 60 calculations simultaneously. We calculate only with 27 PEs in the vertical direction, since this is the minimum number of iterations we have to accomplish. The further 5 bits are only present to get the required accuracy. Assuming a completely filled pipeline the maximum performance is equal to the calculation time for one iteration, i.e. the time one PE needs. This time is 33 (32 bits + 1 step for control code) times the reciprocal of the used clock frequency. Hence, the maximum computing performance we can achieve is shown in (21).

$$P = 60 \cdot \frac{1}{33 \cdot \frac{1}{\text{clock cycle}}} \tag{21}$$

In Fig. 10 we compare the computing performance with pure electronic systems.
We can see, that for a clock cycle of 100 MHz our single optoelectronic chip performs 182 Mega operations per second (MOPS). This outperforms systems like CISC, RISC and signal processors with the exception of the Cray 2 supercomputer. We think, that 100 MHz are easy to achieve, since we need not more than 10 gate delays in our PE and 1ns gate delay is a reserved assumption. The results show that our approach offers enough potential to get a competitive edge versus pure electronic designs.

Let us now take a closer look to the required optical interconnection scheme. As already mentioned only three optical inputs are needed per PE, two for the constant values and one for the clock. If we refer to the defaults of the CO-OP workshop again, we can assume a vertical and horizontal distance of $62.5 \times 125\mu m$ for the self-electrooptic effect device (SEED) elements used as optical "pins". So, we can arrange the three optical inputs in vertical direction with a pitch size of $125\mu m$. Fig. 11 shows schematically the optical "wiring" and the setup for our bit serial parallel signal processor.

In addition to the processor array we need a memory circuit containing the constant values. The memory itself can be realized again as a H-SEED circuit. As alternative also an one-dimensional array of VCSELs flip-chip bonded on a silicon circuit is conceivable. The silicon circuit is structured in rows. In the i^{th} row the constant values for the i^{th} iteration is stored. The bits of the constants are retained in a ring shifter. So, in each clock period the corresponding bit is shifted out and is used to switch on the VCSEL. The emitted power must be sufficient to drive 60 receivers, which are assigned to the PEs in one row of the processor circuit. The light has to be broadened horizontally to realize the fan-out. Such an interconnection scheme had to be realized two times for each PE to broadcast the two constant bits. In the same way we organize the clock distribution for all PEs in one row. The corresponding circuit to generate the clock can be integrated directly in the memory chip. We emphasize that the number of optical links running between the memory and processor circuit is $60 \times 3 \times 27 = 4860$, what is impossible to achieve with current electronic technology.

If we assume an optical output power of 1–2mW per VCSEL, the dissipated power in the one-dimensional VCSEL array should be handled without any problems. An important question on the receiver side concerns the necessity for an amplification in the receiver circuit. Using SPICE simulations for a 0.8 μm CMOS process we showed that about 100μA photo current are sufficient to switch an inverter with 200MHz directly by the optical input signal. Assuming a responsivity of about 0.4 A/W, as it is characteristic for SEED diodes, we need 250μW optical input power to get a photo current of 100μA. Then each VCSEL had to emit unrealistic 15mW for a fan-out of 60. To operate with reasonable values a photo current of 10μA must be sufficient. Hence, in this case an amplification of the photo current should be provided to avoid mistakes caused by noise currents. Then the optical power budget available with VCSELs is sufficient to realize the broadcast connection.

For the light coupling onto the chip we propose the use of multiple beam splitters[21]. Special binary phase hologramms with a period number of N, known as Dammann gratings, can equally distribute an input light intensity onto 2N+1 points on a line. As alternative to Dammann gratings devices of the Talbot type are of interest[22] for two- or one-dimensional array illumination, too. They avoid Dammann handicaps as an additional Fourier lens and the lambda drift sensivity. Details concerning the realization are to discuss with the optics community. We hope, that in the mid-term future, optical systems exploiting the Talbot effect for large 1-to-N interconnections, can be realized either in planar optics techniques or with stacked optics to integrate broadcast interconnectioncs with OE-VLSI circuits.

7. Remarks on fault tolerance

Since we aspire a massively parallel system fault tolerated mechanisms are essential. Therefore we developed an approach to simulate fault models for optoelectronic circuits using VHDL. VHDL is well known for the design of digital systems. The complete architectures presented in Section 3 and 4 was validated by VHDL simulation. A testbench generates stimuli for the input ports of the PE array. Comparing the output of the system with the expected results was the indicator for a bugfree system. The VHDL model of the system is able to be processed by the Synopsys[9] synthesis tool. This tool maps the low-level behavior description of the PEs to a cell library of a chip manufacturer integrating the functionality in silicon.

The new aspect of this section is a VHDL model for the optical communication and the optoelectronic transmitters and receivers. Our model integrates the possibilities to simulate the system with faulty transmitters/receivers and cross talk between optical communication channels with the aim to investigate the impact of these faults to the system.

To simulate different fault sources in receivers we can consider either a stuck at 0 or a stuck at 1 fault model. If the receiver delivers always a zero, even it is illuminated, a stuck at 0 model can be used. This holds also for an optical sender that emits no light although the threshold current is sufficient. To consider also fault models in differential coded optical inputs, which are mostly used in SEED based systems, a stuck at 1 fault model is to use in the simulation.

Since cross talk depends on the layout of the optical channels, models of the optics are coupled very closely to the architecture. By carrying out simulations based on an architectural specific fault model we want to derivate tolerances concerning the optical devices. For example we want to find out at which level of cross talk the system fails. As a result of our simulations we want to define specifications for quality of optical devices. These values can serve as inputs for physicists.

For the simulation and evaluation of the above mentioned faults we use the tool VERIFY[23,24]. VERIFY is a tool which injects single faults during the simulation of a system, e.g. stuck at 0/1 or cross talk. In addition, VERIFY evaluates the impact of the injected faults by comparing the waveforms of the output signals of a golden run (no faults are injected) with every run belonging to a specific fault. VERIFY was used for this task because it works on VHDL code and the extension of the VHDL models with the above mentioned fault models is very easy.

The model of the optoelectronic system can be seperated into two parts. One part covers the model of the electronic components, the PEs. This part is designed without respect to the optics. Hence, the design of the PEs itself is exactly the same process like conventional circuit design. The other part, the optical model, covers the model of the optical interconnections and tranformation of the optical interfaces to digital interfaces, the optical sender and receiver. In the following paragraphs only the optical model will be

described.
The optical interconnections are realized with integer signals. For each possible optical link exists a parameter describing the amount of light coming in if the sender is on. Connections representing a real connection from a sender to a receiver are weighted high, full line in Fig. 12, connections representing cross talk are weighted low, dashed lines in Fig. 12. Summing up all weights from one optical receiver to every active sender is the amount of light reaching the receiver. The receiver only sends a 1 to the digital interface if the sum is greater than a threshold, else it sends a 0. State of the art is that we generate the interconnection parameters with a program, calculating the parameter depending on the distance from the emitter to the receiver where cross talk is less weighted than desired communication. Working together with physicists more fitting parameters for this model will be searched to program algorithms for automatic generation of the parameter matrix.

8. Summary and outlook

Our goal was the design of a reconfigurable parallel signal processor using optical interconnections to avoid long on-chip lines. Fine grained architectures are preferable to exploit the potential of high space bandwidth in optics. Therefore we developed uniform low-level algorithms from the add-and-shift class to calculate typical operations for signal processing. These are the eight operations *sine, cosine, exponential function, logarithm, arctangent, square root, multiplication* and *division*. This can be achieved by using only simple operations like adding, shifting and an access to a table. No sophisticated multiplication units are necessary.

The low-level algorithms were mapped onto three different optoelectronic processor architectures for a hardwired implementation. By a theoretical analysis we determined that a bit serial architecture approach delivers the highest throughput, but the bit parallel approach based on a conditional sum adder shows the best latency behavior. Since we were interested in maximizing the throughput we mapped the bit serial model onto two OE-VLSI circuits linked together by optical chip to chip interconnections. One circuit contains the fine grained processor array and the other one the memory. Supported by a VHDL synthesis we deduced an size of 27×60 for the processor array. This results in a computing performance of 182 Mega operations per second for a moderate clock speed of 100 MHz. Comparing this to current pure electronic solutions this approach signifies an improvement. Further improvements are possible since our processor architecture offers still potential for optimization. The requirements for the optical interconnection scheme should represent no unsolvable difficulties. We need 81 optical senders with a fanout capability of 60. The emitted light of each sender has to be broadcasted along one row of the array.

Since we aspire massively parallel systems we think fault tolerant mechanisms are necessary. A first step towards this direction is the development of a concept to consider optical cross talk and stuck at 0 fault models for optical transmitters and receivers in the VHDL specification of an optoelectronic system.

One of the next steps we intend to do in the future is to carry out simulation experiments to derivate results concerning the allowable tolerance of the optical system. Furthermore, since the gained performance measures for our architecture are promising and we judge the requirements for optics and optoelectronics as moderate, we think a hardware realization must be one of the next goals. To achieve an efficient solution, especially for the realization of the optical interconnection system, a close cooperation between physics and computer science is indispensable. Such interdisciplinary work is anyway one of the main future tasks to lead optoelectronic computing to success.

9. Acknowledgements

The authors wish to acknowledge support for this work from the *Deutsche Forschungsgemeinschaft (DFG)* .

Figures

```
initialize

for i = 1 to n

 (a) shift_1 shift_2
 (b) ADDSUB_x ADDSUB_y ADDSUB_z
 (c) if (y≥ 0 ) then
         calculation (b) is valid (bit algorithm)
       or
         keep add/sub operation signs (CORDIC)
     else
         use old values for next iteration again
         (bit algorithm)
       or
         invert the add/sub operation signs for
         next iteration (CORDIC)
```

Fig. 1. Digit algorithm structure

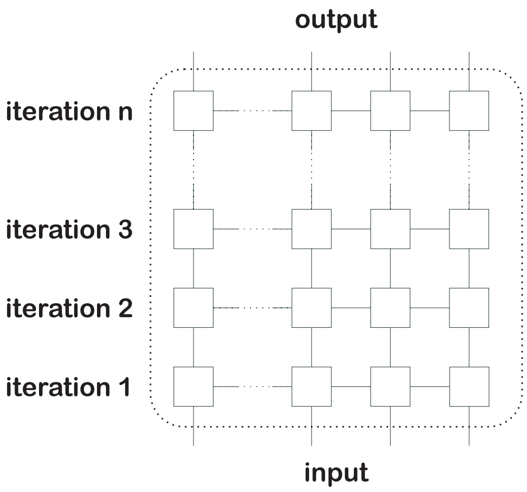


Fig. 2. Processor array of the DSP system.

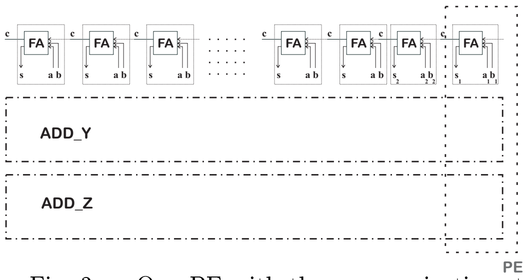


Fig. 3. One PE with the communication structure of a ripple carry adder.

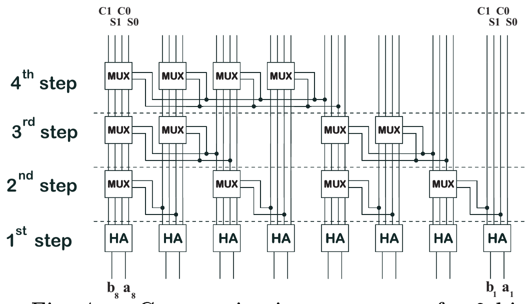
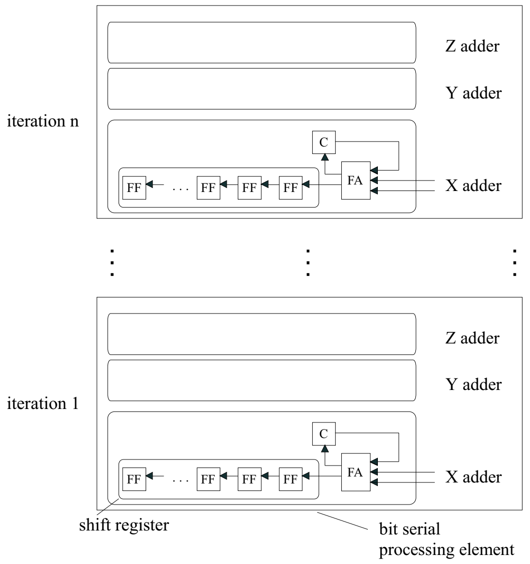


Fig. 4. Communication structure of a 8 bit conditional sum adder in a time/PE diagram.



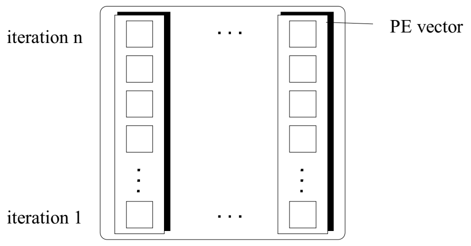


Fig. 5. Bit serial PE vectors for a DSP system arranged to a matrix.

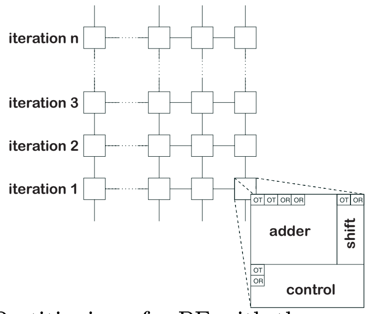


Fig. 6. Partitioning of a PE with the corresponding optical I/Os

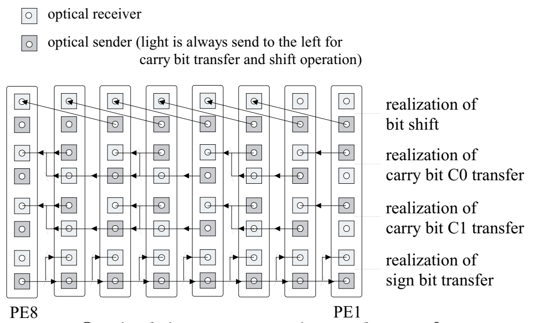

optical receiver
optical sender (light is always send to the left for carry bit transfer and shift operation)
realization of bit shift
realization of carry bit C0 transfer
realization of carry bit C1 transfer
realization of sign bit transfer
PE8
PE1

Fig. 7. Optical interconnection scheme for one row of PEs.

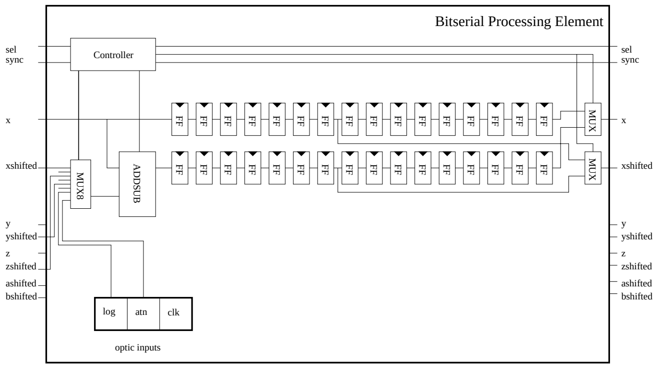


Fig. 8. Setup of a PE of bit serial approach.

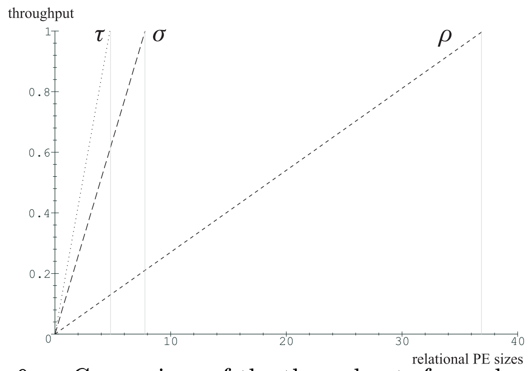


Fig. 9. Comparison of the throughputs for each pair of bit parallel (ripple carry), bit parallel (CSA) and bit serial approach

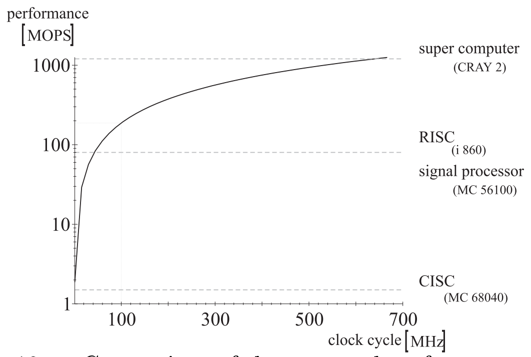


Fig. 10. Comparison of the expected performance of the optoelectronic bit serial DSP system with pure electronic architectures

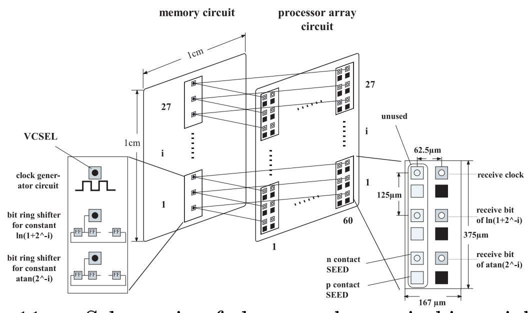


Fig. 11. Schematic of the optoelectronic bit serial DSP architecture

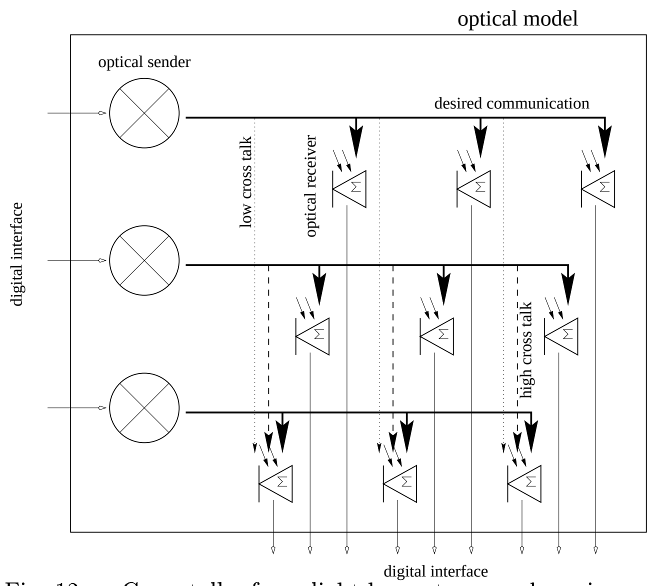


Fig. 12. Cross talk of one light beam to several receivers

Tables

function	iteration formulae	init	condition
$\log(1+\alpha)$	$x_{i+1} = x_i + \log(1+2^{-i})$ $y_{i+1} = y_i - 2^{-i}z_i$ $z_{i+1} = z_i + 2^{-i}z_i$	$x_0 = 0$ $y_0 = \alpha$ $z_0 = 1$	$y_{i+1} \overset{?}{\geq} 0$
e^{α}	$x_{i+1} = x_i + x_i 2^{-i}$ $y_{i+1} = y_i - \log(1+2^{-i})$	$x_0 = 1$ $y_0 = \alpha$	$y_{i+1} \overset{?}{\geq} 0$
$\sqrt{\alpha}$	$x_{i+1} = x_i + 2^{-(i+1)}$ $y_{i+1} = y_i - x_i 2^{-(i)} - 2^{-2(i+1)}$	$x_0 = 0$ $y_0 = \alpha$	$y_{i+1} \overset{?}{\geq} 0$
$\alpha \cdot \beta$	$x_{i+1} = x_i + \alpha 2^{-i}$ $y_{i+1} = y_i - 2^{-i}$	$x_0 = 0$ $y_0 = \beta$	$y_{i+1} \overset{?}{\geq} 0$
$\frac{\alpha}{\beta}$	$x_{i+1} = x_i + \alpha 2^{-i}$ $y_{i+1} = y_i - \beta 2^{-i}$	$x_0 = 0$ $y_0 = 1$	$y_{i+1} \overset{?}{\geq} 0$
$\sin(\alpha)$	$x_{i+1} = x_i \pm 2^{-i}z_i$ $z_{i+1} = z_i \mp 2^{-i}x_i$ $y_{i+1} = y_i \mp \arctan(2^{-i})$	$x_0 = 0$ $y_0 = \alpha$ $z_0 = \kappa$	$y_i \overset{?}{\geq} 0$
$\cos(\alpha)$	$z_{i+1} = z_i \pm 2^{-i}x_i$ $x_{i+1} = x_i \mp 2^{-i}z_i$ $y_{i+1} = y_i \mp \arctan(2^{-i})$	$x_0 = \kappa$ $y_0 = \alpha$ $z_0 = 0$	$y_i \overset{?}{\geq} 0$
$\arctan(\alpha)$	$x_{i+1} = x_i \pm \arctan(2^{-i})$ $y_{i+1} = y_i \mp 2^{-i}z_i$ $z_{i+1} = z_i \pm 2^{-i}y_i$	$x_0 = 0$ $y_0 = \alpha$ $z_0 = 1$	$y_i \overset{?}{\geq} 0$

Table 1. Developed uniform low-level algorithms for standard functions

architecture model	#steps per iteration	#PEs for whole calculation	#parallel calculations
bit serial	$n + \mathrm{ld}\, n + 1$	n	$\frac{A_{\mathrm{Chip}}}{n \cdot A_{bs}}$
bit parallel$_{\text{ripple carry}}$	$n + \mathrm{ld}\, n + 1$	$(n + \mathrm{ld}\, n) \cdot n$	$\frac{A_{\mathrm{Chip}}}{A_{bpRCA} \cdot (n + \mathrm{ld}\, n) \cdot n}$
bit parallel$_{\text{conditional sum}}$	$\mathrm{ld}(n + \mathrm{ld}\, n) + 3$	$(n + \mathrm{ld}\, n) \cdot n$	$\frac{A_{\mathrm{Chip}}}{A_{bpCS} \cdot (n + \mathrm{ld}\, n) \cdot n}$

Table 2. Number of steps for one iteration and for the whole calculation for the different investigated architecture models

List of Figures

List of Tables

1. G. Sai-Halasz. Performance Trends in High-End Processors. *Proceeding of the IEEE*, 83(1):20–36, Jan. 1995.
2. K.W. Goossen, J.A. Walker, L.A. D'Asaro, S.P. Hui, B. Tseng, R. Leibenguth, D. Kossives, D.D. Bacon, D. Dahringer, L.M.F. Chirovsky, A.L. Lentine, and D.A.B. Miller. GaAs MQW Modulators Integrated with Silicon CMOS. *IEEE Phot. Tech. Lett.*, 7:360–362, 1995.
3. A.V. Krishnamoorthy and D.A.B. Miller. Scaling Optoelectronic-VLSI Circuits into the 21st Century: A Technology Roadmap. *IEEE Journal of Selected Topis in Quantum Electronics*, 2(1), April 1996, pages 55-75.
4. T.K. Woodward. VSLI-compatible smart-pixel circuits and technology. In *Smart Pixel Digital Digest*, IEEE/LEOS Summer Topical Meetings, page 65, Keystone Colorado, August 1996.
5. S. Araki, M. Kajita, K. Kubota, K. Kurihara, I. Redmond, E. Schenfeld, and T. Suzaki. Experimental Free-Space Optical Network for Massively Parallel Computers. *Applied Optics*, 35(8):1269–1281, March 1996.
6. J. Jahns. Planar Packaging of Free Space Optical Interconnections. *Proc. of the IEEE*, 82:1623–1631, 1994.
7. F.B. MCCormick et al. Experimental investigation of a free-space optical switching network by using symmetric self-electrooptic-effect devices . *Applied Optics*, 31:5431–5446, 1992.
8. J.A. Neff, C. Chen, T. McLaren, C.-C. Mao, A. Fedor, W. Berseth, Y.C. Lee, and V. Morozov. VCSEL/CMOS Smart Pixel Arrays for Free-Space Optical Interconnects. In *Proceedings of MPPOI'96*, A. Gottlieb, Y. Li, and E. Schenfeld, (eds.), IEEE CS Press Los Alamitos, California, October 1996, pages 282-289.
9. The mention of brand names in this paper is for information purposes only and does not constitude an endorsement of the product by the authors or their institutions [SYNOPSYS Corp.].
10. A.V. Krishnamoorthy, P.J. Marchand, F.E. Kiamilev, and S.C. Esener. Grain-size considerations for optoelectronic multistage interconnection networks. *Applied Optics*, 31(26):5480–5507, Sep. 1992.
11. D. Fey and W. Erhard. Algorithms for High–Performance Computing with Smart Pixels. In *Applications of Photonic Technology*, Lampropoulos, G.A. et al., (eds.), pages 97–100, New York, 1995. Plenum Press.
12. D. Fey, A. Kurschat, B. Kasche, and W. Erhard. A 3D Optoelectronic Processor For Smart Pixel Processing Units. In *Proceedings of MPPOI'96*, A. Gottlieb, Y. Li, and E. Schenfeld, (eds.), IEEE CS Press Los Alamitos, California, October 1996, pages 344-351.
13. M. Ishikawa. System Architecture for Integrated Optoelectronic Computing. *Optoelectronics - Devices and Technology*, 9(1):29–36, 1994.
14. J. Volder. The CORDIC trigonometric computing technique. *IRE Transaction on Electronic Computers EC-8*, No. 3, pages 330-334, Sep. 1959.
15. B. Kasche and D. Fey. *Optimale Algorithmen zur Berechnung von Standardfunktionen mittels Smart Pixel Rechenwerke*, Technical Report, University of Jena, W. Erhard (editor), volume 2(3), February 1996. ISSN 0949-3042.
16. I. Koren. *Computer Arithmetic Algorithms.* Prentice Hall Inc., Englewood Cliffs, New Jersey, 1993.
17. K. Hwang. *Computer Arithmetic – Principles, Architecture and Design.* J. Wiley & Sons Inc., New York, 1979.
18. A. R. Omondi. *Computer Arithmetic Systems: Algorithms, Architecture and Implementation.* Prentice Hall Inc., Englewood Cliffs, New Jersey, 1994.
19. J. Slansky. An Evaluation of Several Two-Summand Binary Adders. In *IRE Trans.*, EC-9, No.2, pages 213–226, June 1960.
20. G. Grimm. *Entwicklung eines VLSI Layouts für op-*

toelektronische programmierbare Schaltkreise, Technical Report, University of Jena, W. Erhard (editor), volume 3(21), April 1997. ISSN 0949-3042.

21. U. Krackhardt, and N. Streibl. Design of Dammann-gratings for array generation, In *Optics communication*, No. 74, pages 31-36 (1989)
22. A.W. Lohmann, and J.A. Thomas. Making an array illuminator based on the Talbot effect, In *Appl. Opt.*, No. 29, pages 4337-4340 (1990)
23. V. Sieh, O. Tschäche, and F. Balbach. Evalutaion of Dependable Systems using VERIFY. In *Preprints of 6th Conference on Dependable Computing for Critical Applications (DCCA-6)*, M. Dal Cin (editor), Grainau, Germany, March 1997, pp. 59-76.
24. V. Sieh, O. Tschäche, and F. Balbach. VERIFY: Evaluation of Reliability Using VHDL-Models with Embeded Fault Descriptions. In *The Twenty-Seventh Annual International Symposium on Fault-Tolerant Computing (FTCS-27)*, P. Storms (editor), IEEE, Seattle, Washington, June 1997, pp. 32-36.