Application of Fast Layout Synthesis Environment to Dividers Evaluation

A. Houelle, H. Mehrez, N. Vaucher, L. Montalvo* and A. Guyot*

MASI CAO-VLSI laboratory, Université Pierre et Marie Curie
4 Place Jussieu, 75252 Paris Cedex 05, France
* Integrated Systems Design Group, TIMA/INPG,
46 Avenue Félix Viallet, 38031 Grenoble Cedex, France.

Abstract

Experience has shown that generator programs are quite often written by VLSI designers, as they hold the empirical knowledge better than anyone. However, their ability does not necessarily include programming and debugging skills: these designers have to focus on the problem at hand, not on the tools or the language they use to solve it. GenOptim has been created to quickly design efficient IEEE 754 floating-point macro-cell generators that do not rely on particular target technologies. Whereas, according to [1], the design of fast and efficient adders, multipliers and shifters is well understood, division and square root remain a serious design challenge. GenOptim was used to quickly evaluate new divider architectures

1. Introduction

The aim of this paper is twofold: first to introduce a new layout synthesis tool developped at Paris Pierre and Marie Curie University and then to report on how it can be used to select a "good" division algorithm. As noted in [1], going to higher order radixes seems a promising research method to improve the speed of division and square root extraction. Nevertheless higher order radixes decrease the circuit regularity and increase the number of wires. Since the implication of regularity and connectivity on speed and area are difficult to address analytically and manual electrical optimisation (fan-out, transistor sizing, ...) is tedious, exploration through synthesis becomes a necessity.

The paper is organised as follows: first GenOptim functionalities are described, then it is applied to a sequence of three dividers developped at Grenoble University, that are detailed from the algorithms down to the logic equations and the layout. As a conclusion, speed and area measurements are compared.

2. GenOptim description

GenOptim is a library of C functions which allows the user to describe the netlist, the layout, the test vectors and the behavioural views as shown in figure 1. The C language was chosen for its popularity and portability.

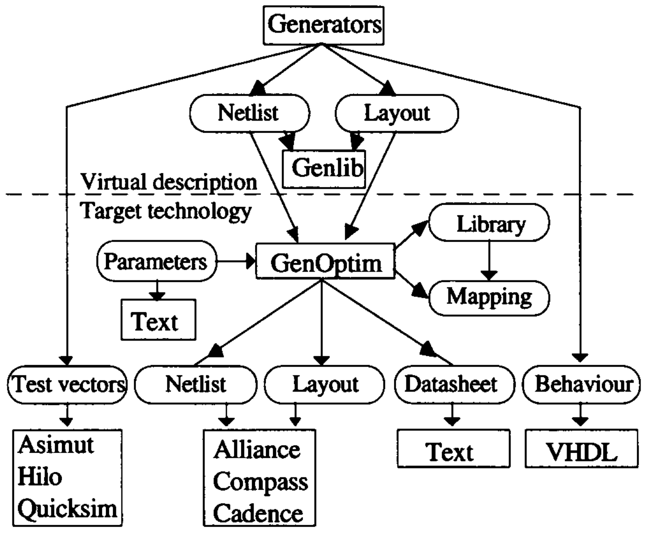


Figure 1: GenOptim description.

Generators created with GenOptim can be mapped on any standard cell library (actually we have tried three kinds of library: CMOS standard cell, CMOS datapath cell and GaAs standard cell). As depicted in figure 1, the circuit generation starts with a virtual description (netlist and layout of virtual cell) of the circuit. Next, GenOptim maps the virtual description to the target library and optimises the circuit. Each generator is able to generate five different views which are:

The netlist view: It describes hierarchical interconnections among cells.

The layout view: It provides an optimised preplacement of all real instanciated cells.

These two views contain enough information to route the final circuit. Each CAD system has its own router and its proper description of netlist and layout. To resolve the

1063-6889/95 $4.00 © 1995 IEEE

proper description of netlist and layout. To resolve the problem of compatibility, GenOptim uses multiple drivers and is able to generate netlist and layout views for several systems. We have already created drivers for Cadence, Compass and Alliance [2] systems.

The test vector view: This view must be able to test 100% of the generated circuits with a minimum set of test vectors. These vectors remain interesting for complex operators using scan path. Test vectors are highly dependent on the circuit architecture and profit by the knowledge of the designer.

The behavioural view: The language of description is a subset of VHDL [3] which may be synthesised. It allows correctness verification of the mapping netlist.

The datasheet view: It is a report of all timing and critical paths of the generated circuit. It is very useful when designing a generator. The user can easily and quickly see where the critical paths are and the evolution in performance/surface following the activated optimisations. Other information can be found in the datasheet e.g.: Cell positioning (useful when compression is used), mapping, activated optimisations, input capacitance, output resistance.

GenOptim uses two files. The first one is the mapping file and describes how the virtual cells will be mapped following the cells existing in the library. The second file contains the options enabled by the user. These options concern mainly the optimisations related to time performance and positioning, target technology, CAD system output.

3. GenOptim methodology of mapping and optimisations

The process of transforming a circuit description into a technology dependent realisation is called technology mapping. During this step, the size of active elements, speed of the circuit and power consumption are determined. By using virtual definition of cells in netlist and placement views, complete independence can be easily obtained. As shown in figure 2, a Multiplexer can be directly mapped, if it exists in the target library, otherwise, it will be built with basic cells such as and, or, etc.... GenOptim profits from the mapping step to optimise the circuit. A direct replacement of the logic circuit gates by their physical realisations chosen from the given library does not lead to satisfactory results. It is usually the case that such a realisation does not fulfil the timing constraints. In order to correct the situation, the initial realisation undergoes an optimisation process which takes into account specifics of the available technology.

All these strategies take into account the placement of cells and the consequent area. Thus, GenOptim chooses the best trade-off between performance optimisation and area complexity. Furthermore, all these strategies can be applied to different critical paths or simply to specific paths which are given in the option file.

It may happen that some gates are not strong enough to drive many outputs. Those situations may produce excessive delay due to loading factors coming from the driven gates. In order to solve the problem, the original gates can be replaced by stronger copies from the existing cell library (buffered gates). We call this technique multi-mapping, because a virtual cell is mappable with several cells of the target library (as shown in figure 2). GenOptim also uses techniques such as buffer insertion and cell duplication in order to reduce the propagation time of the circuit.

By construction, the main task of the GenOptim generator end-user is to map his own standard cells library on the virtual generator standard cells library. However, this task is rather simple, as a direct mapping between the generator cell model and its "real" standard cell equivalent is straightforward. It is clear that the resulting mapping netlist is crucial and its correctness relies on an adequate comparison with the corresponding behaviour generator (view). This validation sequence is performed by the end-user in its own design environment and is called site test.

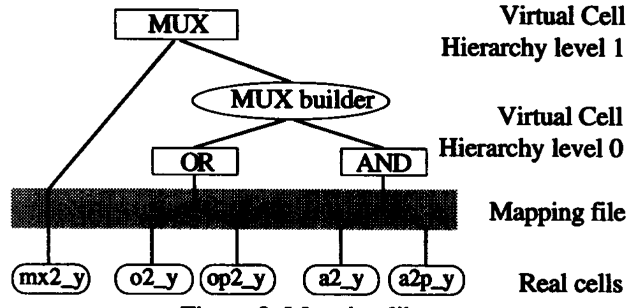


Figure 2: Mapping library

3.1 Generator validation

In order to validate the divider generators, a specific simulator has been used. This simulator has been created to perform IEEE addition, subtraction, multiplication, division and square root with parametrised operands. The results of the simulator have been compared with the results of a SPARC coprocessor (which is IEEE compatible) in order to verify its correctness (as shown in Figure 3). The simulator creates test vectors which are used to validate the behavioural view of the generator. The netlist and the placement views generated by the divider generator are routed by the user router. Thus, the core layout can be validated thanks to a hierarchical Design Rule Checker (DRC). An extracted netlist of transistors can be obtained from the resulting layout and validated with the simulator test pattern. The netlist view is the input of a functional abstractor which provides a VHDL Data-Flow behavioural description. The resulting behaviour can be compared to the initial behavioural view, thanks to a formal proof analyser.

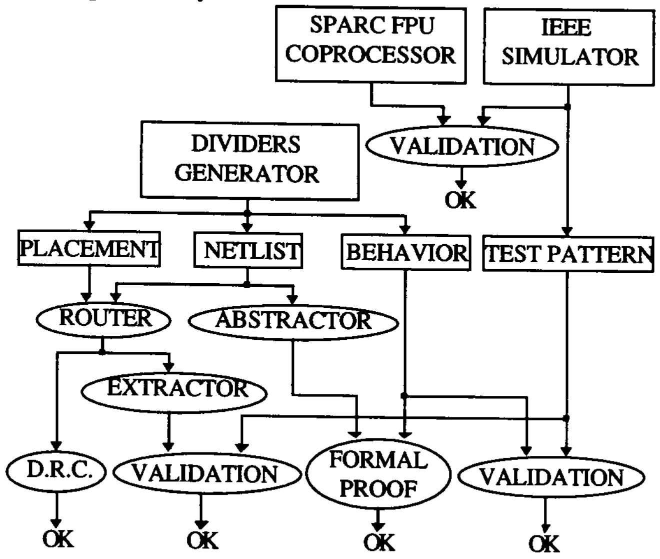


Figure 3: Validation process

Batch files are used to test the divider generators for a significant number of precisions including those defined by the IEEE floating-point standard.

4. Hardware division implementation

There are mainly two approaches for the hardware implementation of division Q = A÷D. The Newton-Raphson algorithm uses a series of multiplications and additions to develop an increasingly accurate approximation of the desired quotient Q. The digit-recurrence approach relies on subtraction and multiplication by the radix b and by the quotient digits q_j.

$$R^{(0)} = A$$
$$R^{(j+1)} = b * (R^{(j)} - q_{j+1} * D)$$
$$Q^{(j+1)} = Q^{(j)} + q_{j+1} * b^{-j-1}$$

The iteration maintains the property $A = D*Q^{(j)} + b^{-j}* R^{(j)}$ and $b^{-j} * R^{(j)} \rightarrow 0$ as $j \rightarrow \infty$. For the same quotient accuracy, the higher the radix b the less iterations are required. The multiplication by the radix b is a simple shift-left, and the multiplication by the quotient digit q_{j+1} is either simple or the product is precomputed. If both $Q^{(j)}$ and $R^{(j)}$ are in redundant notations, then the addition/subtraction can be made carry-propagation-free [4,5,6].

In this paper, we compare the synthesis of a radix-2 combinatorial realisation, with $q_j \in \{-1,0,1\}$, one pseudo-radix-4 with $q_j \in \{-3,-2,-1,0,1,2,3\}$ and one radix-4 with $q_j \in \{-2,-1,0,1,2\}$. The last two divisions require half as many iterations hence half as many slices.

4.1 Notations

In order to avoid ambiguity, the (+,∗) symbols are used to indicate arithmetic addition and multiplication, and the ($\wedge,\vee,\oplus$) for logical AND, OR and XOR and to avoid confusion between sign and subtraction, -1 will be noted $\bar{1}$ when needed. A and D are mantissas of IEEE-754 numbers, so A is written $1+\Sigma_{i=1}^{n} a_i * 2^{-i}$ and $D = 1+\Sigma_{i=1}^{n} d_i * 2^{-i}$, $a_i,\ d_i \in \{0,1\}$. So $1/2 < Q < 2$. Since $-d_i = \overline{d_i} -1$,

$-D = -2 +\Sigma_{i=1}^{n} \overline{d_i} * 2^{-i} + 2^{-n}$.

The value of the quotient is the weighted sum of its digits $Q = \Sigma_{i=1}^{n} q_i * b^{-i}$.

5. Division without operand scaling

Here the value of each quotient Q signed digit $q_j \in \{-1,0,1\}$ is the difference of two bits $q_j = (q_j^+ - q_j^-)$. By analogy with the "Carry save" notation (where each digit is the sum of two bits), this notation is called "Borrow save" [7,8]. The same redundant notation is used to add/subtract the divisor D to the partial remainder R without carry propagation. The iteration of the division is $R^{(j+1)} := 2 * (R^{(j)} - q_j * D)$. The test of the sign of $R^{(j)}$ truncated to the 3 most significant digits is sufficient to select q_j in order to ensure that $-2D < R^{(j+1)} < 2D$.

Let $\Sigma_{in} = \Sigma_{i=-2}^{0} r_i * 2^{-i}$. Let us examine the 3 only possibilities:

If $\Sigma_{in} < 0$ then $-2D < R^{(j)} < 0$.
Let $R^{(j+1)} := 2 * (R^{(j)} + D)$. Then $-2D < R^{(j+1)} < +2D$.
If $\Sigma_{in} > 0$ then $0 < R^{(j)} < 2D$.
Let $R^{(j+1)} := 2 * (R^{(j)} - D)$. Then $-2D < R^{(j+1)} < +2D$.
If $\Sigma_{in} = 0$ then $-1 < R^{(j)} < +1$.
Let $R^{(j+1)} := 2* R^{(j)}$. Then $-2D \le -2 < R^{(j+1)} < +2 \le +2D$.

For clarity, let us call r (r_i^-, r_i^+) the input digit of a bit slice and s (s_i^-, s_i^+) the output digits.

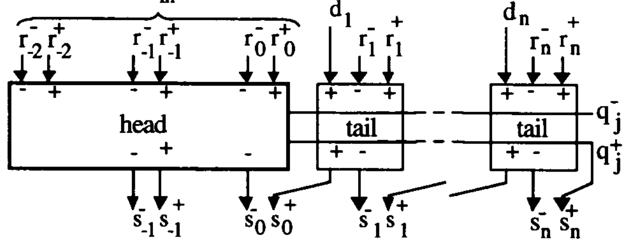


Figure 4: A slice of an hybrid radix-2 divider

5.1 Equations of the tail cell

The tail cells (fig. 4) preserve the equation:

$(2*s_{i-1}^+ - s_i^-) = (r_i^+ - r_i^- + (q_j^+ \wedge \overline{d_i} \vee q_j^- \wedge d_i))$.

The assembly of n tail cells forms an hybrid carry propagation free adder/subtracter. The last digit of S: $s_n^+ * 2^{-n}$ is given by q_j^+.

5.2 Equations of the head cell

As it was first proposed in [9], the head (fig. 4) selects the quotient digit and simultaneously performs the operation on the three most significant digits. In order to prove that the

slice output S has one digit less than the input R, we prove that the head needs only three binary outputs.
Let us first find the range of Σ_{in}.
$1 \le D < 2 \Rightarrow -4 < R < +4 \Rightarrow -5 < \Sigma_{in} < +5$ because R is in BS notation, and $\Rightarrow -4 \le \Sigma_{in} \le +4$ since Σ_{in} is an integer. $-4 \le \Sigma_{in} \le +4$ implies for the head outputs
$-3 \le 2 * (s^+_{-1} - s^-_{-1}) - s^-_0 \le +2$ and the head is consistent.
The head logic equations follow:

$$q^+_j := r^+_{-2} \vee \overline{r^-_{-2}} \wedge (r^+_{-1} \wedge \overline{r^-_{-1}} \vee \overline{r^-_{-1}} \wedge r^+_0 \wedge \overline{r^-_0} \vee r^+_{-1} \wedge r^+_0 \wedge \overline{r^-_0}) ;$$

$$q^-_j := r^+_{-2} \vee \overline{r^+_{-2}} \wedge (\overline{r^+_{-1}} \wedge r^-_{-1} \vee \overline{r^+_{-1}} \wedge \overline{r^+_0} \wedge r^-_0 \vee r^-_{-1} \wedge \overline{r^+_0} \wedge r^-_0) ;$$

$$s^+_{-1} := r^+_{-2} \wedge (r^+_{-1} \vee \overline{r^-_{-1}} \vee r^+_0 \wedge \overline{r^-_0}) \vee \overline{r^-_{-2}} \wedge r^+_{-1} \wedge \overline{r^-_{-1}} \wedge r^+_0 \wedge \overline{r^-_0};$$

$$s^-_{-1} := r^-_{-2} \wedge (\overline{r^+_{-1}} \vee r^-_{-1} + \overline{r^+_0} \wedge r^-_0) \vee \overline{r^+_{-2}} \wedge \overline{r^+_{-1}} \wedge r^-_{-1} \wedge \overline{r^+_0} \wedge r^-_0 ;$$

$$s^-_0 := q^+_j \oplus r^+_0 \oplus r^-_0 ;$$

6. Radix-2 division with operand scaling

Instead of computing $Q = A+D$ with $1 \le A < 2$ and $1 \le D < 2$, we will compute $Q = \frac{K * A}{K * D} = \frac{R_0}{Y}$, where K is the scaling factor.
In 1991, Burgess [10] proposed to reduce the range of D in the following way:
if $(d_1 = 1)$ **then** $K = 1 - 1/4$ **else** $K = 1$.
Obviously the range reduction is easy to perform, it does not change the quotient and leads to $1 \le Y < 1.5$.
Now we have $Y = 1 + \Sigma_{i=2}^{n} y_i * 2^{-i}$ and

$$-Y = -1.5 + \Sigma_{i=2}^{n} \overline{y_i} * 2^{-i} + 2^{-n}.$$

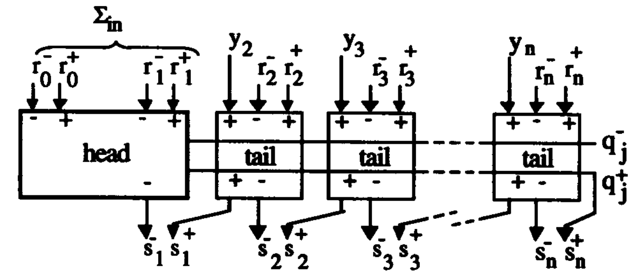


Figure 5: Radix-2 divider slice with operand scaling

input value Σ_{in}	outputs				operation (head and tail)
$(r^+_0 - r^-_0) + 2^{-1} * (r^+_1 - r^-_1)$	s^-_1	q^+_j	q^-_j	$-2^{-1} * s^-_1$	$R^{(j+1)} := 2 * R^{(j)} - q_j * Y$
- 1,5	1	0	1	-0,5	$R^{(j+1)} := 2 * R^{(j)} + 1 + \sum_{i=2}^{n} y_i * 2^{-i}$
- 1	0	0	1	0	$R^{(j+1)} := 2 * R^{(j)} + 1 + \sum_{i=2}^{n} y_i * 2^{-i}$
- 0,5	1	0	0	-0,5	$R^{(j+1)} := 2 * R^{(j)} + \sum_{i=2}^{n} 0 * 2^{-i}$
0	0	0	0	0	$R^{(j+1)} := 2 * R^{(j)} + \sum_{i=2}^{n} 0 * 2^{-i}$
0,5	0	1	1	0	$R^{(j+1)} := 2 * R^{(j)} - 0{,}5 + \sum_{i=2}^{n} 1 * 2^{-i} + 2^{-n}$
1	1	1	0	-0,5	$R^{(j+1)} := 2 * R^{(j)} - 1{,}5 + \sum_{i=2}^{n} \overline{y_i} * 2^{-i} + 2^{-n}$
1,5	0	1	0	0	$R^{(j+1)} := 2 * R^{(j)} - 1{,}5 + \sum_{i=2}^{n} \overline{y_i} * 2^{-i} + 2^{-n}$

Table I

6.1 Operation table

Like Vandemeulebroeke [8], we write zero in two forms: either $0 = \Sigma_{i=2}^{n} 0 * 2^{-i}$ or $-0 = -0.5 + \Sigma_{i=2}^{n} 1 * 2^{-i} + 2^{-n}$. As before, to prove the division we check that the head needs just one bit to express the output values associated with all the possible head input values. Note that no range consideration is necessary.

6.2 Equations of the head

The head cell (fig. 5) has the following logic equations:

$$q^+_j := r^+_0 \wedge \overline{r^-_0} \vee (r^+_0 \vee \overline{r^-_0}) \wedge r^+_1 \wedge \overline{r^-_1} ;$$

$$q^-_j := \overline{r^+_0} \wedge r^-_0 \wedge (\overline{r^+_1} \vee r^-_1) \vee r^+_0 \wedge \overline{r^-_0} \wedge \overline{r^+_1} \wedge r^-_1 \vee (r^+_0 \wedge r^-_0 \vee \overline{r^+_0} \wedge \overline{r^-_0}) \wedge r^+_1 \wedge \overline{r^-_1} ;$$

$s^-_1 := q^+_j \oplus r^+_1 \oplus r^-_1$ { only digits with weight 2^{-1}} ;

The fact that the two first digits of $Y = K * D$ are constant, namely $y_0 = 1$ and $y_1 = 0$, simplifies the head equations, nevertheless their evaluation is not significantly faster than without range reduction. No speed improvement can be expected from this approach, but this approach can be generalised if the dividend is further reduced to force $y_2 = 0$ as well.

7. Pseudo radix-4 division with operand scaling

In this approach, we need 2*Y and 3*Y to have constant values (independent of the value of D) for the digits at the left of the decimal point and one digit at the right of the decimal point (in **bold**).
$2*Y = \mathbf{10.0} \; y_3 \, y_4 \, y_5 \ldots \; y_n$ ($2 \le 2*Y < 2 + 1/2$).
$3*Y = \mathbf{11.0} \; f_2 \, f_3 \, f_4 \, f_5 \ldots f_n$ ($3 \le 3*Y < 3 + 1/2$).

Those conditions lead to $1 \le Y < 1 + 1/6$. But $1 \le Y < 1 + 1/8$ seems easier to achieve.

To get $Y = K * D$, the scaling factor K is chosen according to the following Figure 6:

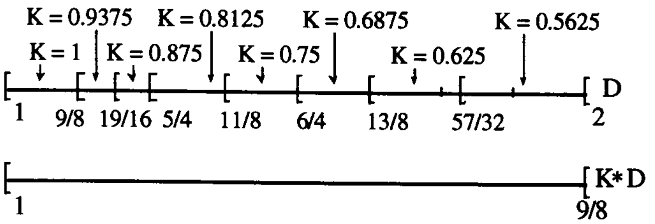


Figure 6: Selection of the scaling factor values

$Y = K*D$ is obtained in redundant notation by two 4-input multiplexers and 1 layer of FA, and $3*K*D$ by two additional FA layers. Then in parallel the carries for $K*D$ and $3*K*D$ are propagated, giving Y and $3*Y$ (Figure 10). So the computation of $3*Y$ after Y does not require any extra time.

In parallel $K*A$ is computed, that is the first partial remainder $R^{(0)}$. No carry has to be propagated since $R^{(0)}$ is in redundant notation.

On Figure 7, as on figure 5, the signed binary digits of $R^{(j)}$ are noted (r_i^- , r_i^+) and signed binary digits of $(R^{(j)} - q_j * Y)$ are noted (s_i^- , s_i^+). Note that $(R^{(j)} - q_j*Y)$ has 2 most significant digits less than $R^{(j)}$.

The values of $q_j \in [-3, +3]$ are coded on 3 bits in the form *sign, absolute value*. The *sign* is fed to s_n^+.

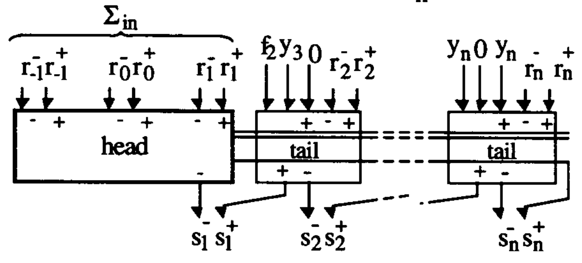


Figure 7: A slice of a pseudo radix-4 divider

input	outputs		operation (head and tail)
Σ_{in}	q_j (3 bits)	$-2^{-1} * s_1^-$	$R^{(j+1)} := 4 * (R^{(j)} - q_j * Y)$
-3.5	-3	**-0.5**	$R^{(j+1)} := 4 * (R^{(j)} + \mathbf{3.0} + \sum_{i=2}^{n} f_i * 2^{-i})$
-3	-3	**0**	$R^{(j+1)} := 4 * (R^{(j)} + \mathbf{3.0} + \sum_{i=2}^{n} f_i * 2^{-i})$
-2.5	-2	**-0.5**	$R^{(j+1)} := 4 * (R^{(j)} + \mathbf{2.0} + \sum_{i=2}^{n} y_{i+1} * 2^{-i})$
-2	-2	**0**	$R^{(j+1)} := 4 * (R^{(j)} + \mathbf{2.0} + \sum_{i=2}^{n-1} y_{i+1} * 2^{-i})$
-1.5	-1	**-0.5**	$R^{(j+1)} := 4 * (R^{(j)} + \mathbf{1.0} + \sum_{i=3}^{n} y_i * 2^{-i})$
-1	-1	**0**	$R^{(j+1)} := 4 * (R^{(j)} + \mathbf{1.0} + \sum_{i=3}^{n} y_i * 2^{-i})$
-0.5	-0	**-0.5**	$R^{(j+1)} := 4 * (R^{(j)} + \mathbf{0.0} + \sum_{i=2}^{n} 0 * 2^{-i})$
0	-0	**0**	$R^{(j+1)} := 4 * (R^{(j)} + \mathbf{0.0} + \sum_{i=2}^{n} 0 * 2^{-i})$
0.5	+0	**0**	$R^{(j+1)} := 4 * (R^{(j)} - \mathbf{0.5} + \sum_{i=2}^{n} 1 * 2^{-i} + 2^{-n})$
1	+1	**-0.5**	$R^{(j+1)} := 4 * (R^{(j)} - \mathbf{1.5} + 2^{-2} + \sum_{i=3}^{n} \overline{y_i} * 2^{-i} + 2^{-n})$
1.5	+1	**0**	$R^{(j+1)} := 4 * (R^{(j)} - \mathbf{1.5} + 2^{-2} + \sum_{i=3}^{n} \overline{y_i} * 2^{-i} + 2^{-n})$
2	+2	**-0.5**	$R^{(j+1)} := 4 * (R^{(j)} - \mathbf{2.5} + \sum_{i=2}^{n-1} \overline{y_{i+1}} * 2^{-i} + 2^{-n} + 2^{-n})$
2.5	+2	**0**	$R^{(j+1)} := 4 * (R^{(j)} - \mathbf{2.5} + \sum_{i=2}^{n-1} \overline{y_{i+1}} * 2^{-i} + 2^{-n} + 2^{-n})$
3	+3	**-0.5**	$R^{(j+1)} := 4 * (R^{(j)} - \mathbf{3.5} + \sum_{i=2}^{n} \overline{f_i} * 2^{-i} + 2^{-n})$
3.5	+3	**0**	$R^{(j+1)} := 4 * (R^{(j)} - \mathbf{3.5} + \sum_{i=2}^{n} \overline{f_i} * 2^{-i} + 2^{-n})$

Table II Note from the table that q_j is the integer part of Σ_{in}, and we need both -0 and +0.

7.1 Operations table

Again we note: $\Sigma_{in} = \Sigma_{i=-1}^{1} r_i * 2^{-i} \in \{-3.5, -3, -2.5, -2, -1.5, -1, -0.5, 0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5\}$.

The table II gives the iteration $R^{(j+1)} := 4 * (R^{(j)} - q_j * Y)$ for all the 15 possible values of Σ_{in} .

Since in fact every operation is carried out in radix 2, but two quotient digits are generated at a time, this division is named pseudo-radix-4.

7.2. Equation of the tail

Let **σ_i = case q_j of** (f_i, y_{i+1}, y_i, 0, 1, $\overline{y_i}$, $\overline{y_{i+1}}$, $\overline{f_i}$). Since q_j is in *sign,absolute-value* σ_i is obtained by a 4-input multiplexer followed by an XOR Then the tail cell preserves the identity $2 * s_{i-1}^+ - s_i^- = r_i^+ - r_i^- + \sigma_i$ (fig. 8).

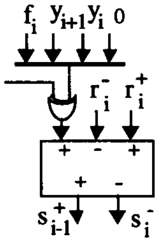


Figure 8

7.3 Equation of the head

q_j is the truncated integer part of Σ_{in}, i.e.
$q_j = \lfloor 2 * (r^+_{-1} - r^-_{-1}) + (r^+_0 - r^-_0) + 2^{-1} * (r^+_1 - r^-_1) \rfloor$ [10, 11].
The ***sign,absolute-value*** is convenient for selection through a multiplexer, and the BS for q_j conversion. s^-_1 is the difference between Σ_{in} and the bold part of $q_j * Y$, so $s^-_1 = r^+_1 \oplus r^-_1 \oplus \mathrm{sign}\ (q_j)$ (fig. 9).

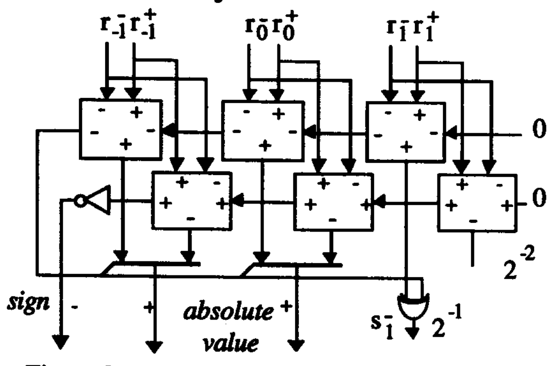


Figure 9: Internal organisation of the Head cell

7.4. Divider organisation

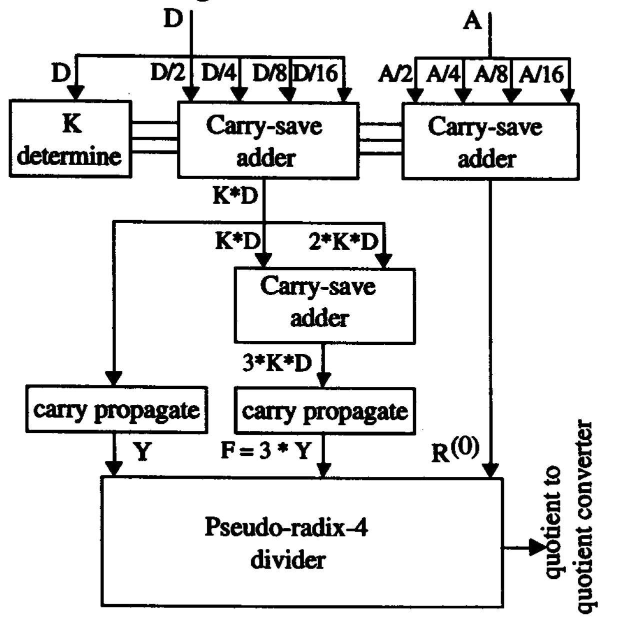


Figure 10: Organisation of the pseudo radix 4 divider

The different blocks are organised as follows: A, D, Y and F are in standard notation, K*D, 2*K*D and 3*K*D as well as $R^{(0)}$ and Q are in BS notations. Every number is in radix-2 [12,13]. From the figure 10 we can see the high structural cost of the range reduction.

8. Radix-4 division

The radix-4 division algorithm is based on the computation of the recurrent equation [12,14,15,16]:

$$R^{(j+1)} = 4 * (R^{(j)} - q_{j+1} * Y).$$

$Y = K*D$ is the divisor, $Q = \Sigma_{j=0}^{n-1} q_j * 4^{-j}$ is the quotient, and $R^{(j)} = \Sigma_{i=0}^{n-1}\ r_i^{(j)} * 4^{-i-j}$ is one of the partial remainders.

At each iteration, the quotient digit q_{j+1} is selected by examining the two most significant digits of the $R^{(j)}$ remainder, according to the rules summarised by figure 11. The arithmetic bounds are given by:
$-Y < R^{(j+1)} < Y$,
and $1 \leq Y < 1 + 1/8$.

$r_1 \backslash r_2$	$\bar{2}$	$\bar{1}$	0	1	2
$\bar{2}$	$\bar{2}$				$\bar{1}$
$\bar{1}$	$\bar{1}$				$\bar{0}$
0	$\bar{0}$	0 or $\bar{0}$			0
1	0	1			
2	1	2			

Figure 11

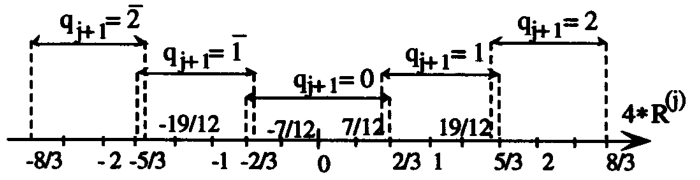


Figure 12: Correspondence between $R^{(j)}$ and q_{j+1}

The dividend X, the quotient Q and the partial remainder $R^{(j)}$ are represented using the signed-digit-set $\{\bar{2}, \bar{1}, 0, 1, 2\}$, similar to the Booth encoding for multiplication. There is no need to compute 3*Y as for the pseudo radix-4. The digits r_i of the partial remainders are encoded on 3 bits according to the equation:
$r_i = -2 * r^-_i + r^+_i + r^{++}_i$.
The $q_{j+1} \in [-2, +2]$ digits of the quotient are encoded by 3 control signals: add, u1, and u2; according to Fig. 13. The radix-4 digits y_i of the divisor Y merely are pairs of odd-even bits of the radix-2 representation encoded according to the equation: $y_i = 2 * y^{++}_i + y^+_i$.

control signals			
add	u1	u2	q_{j+1}
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	2
1	0	0	$\bar{1}$
1	0	1	$\bar{0}$
1	1	0	$\bar{2}$
1	1	1	$\bar{0}$

Figure 13

8.1 Equations of the head

The head (Fig. 17) of the jth slice computes the quotient digit q_{j+1}, encoded by 3 bits: add, u1, and u2, and two bits s^+_1 and s^-_1 of the most significant digit of the remainder. The logic equations of the head cell follow:

$$s^+_1 = r^-_2 \wedge \overline{r^+_2} \wedge \overline{r^{++}_2} ;$$

$$s^-_1 = \overline{r^-_2} \wedge r^+_2 \wedge r^{++}_2 ;$$

$$add = r^-_1 \wedge (\overline{r^{++}_1} \vee \overline{r^+_1}) \vee s^-_1 \wedge (r^-_1 \vee \overline{r^+_1} \wedge \overline{r^{++}_1}) ;$$

$$u1 = \overline{s^+_1} \wedge \overline{r^+_1} \wedge \overline{r^{++}_1} \vee \overline{s^-_1} \wedge r^+_1 \wedge r^{++}_1 ;$$

$$u2 = \overline{s^-_1} \wedge \overline{r^-_1} \wedge (r^{++}_1 \vee r^+_1) \vee$$
$$s^-_1 \wedge (r^+_1 \wedge r^{++}_1 \vee \overline{r^-_1} \wedge \overline{r^+_1} \wedge \overline{r^{++}_1}) \vee s^+_1 \wedge (r^+_1 \oplus r^{++}_1).$$

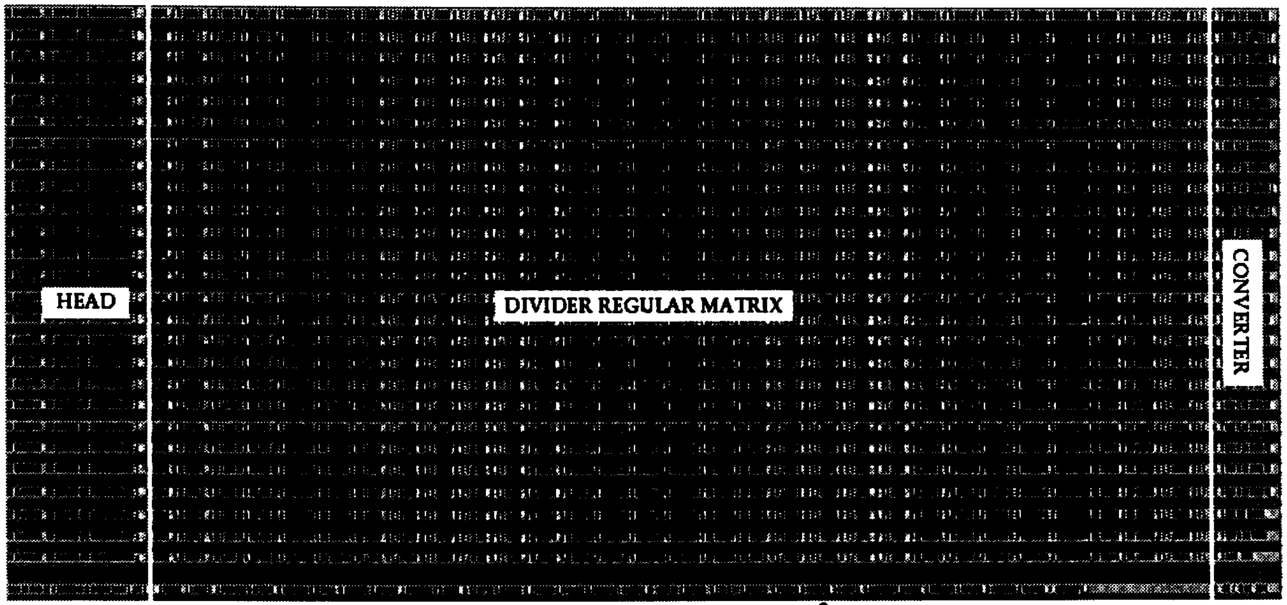


Figure 14: 32-bit radix-2: 28 mm^2

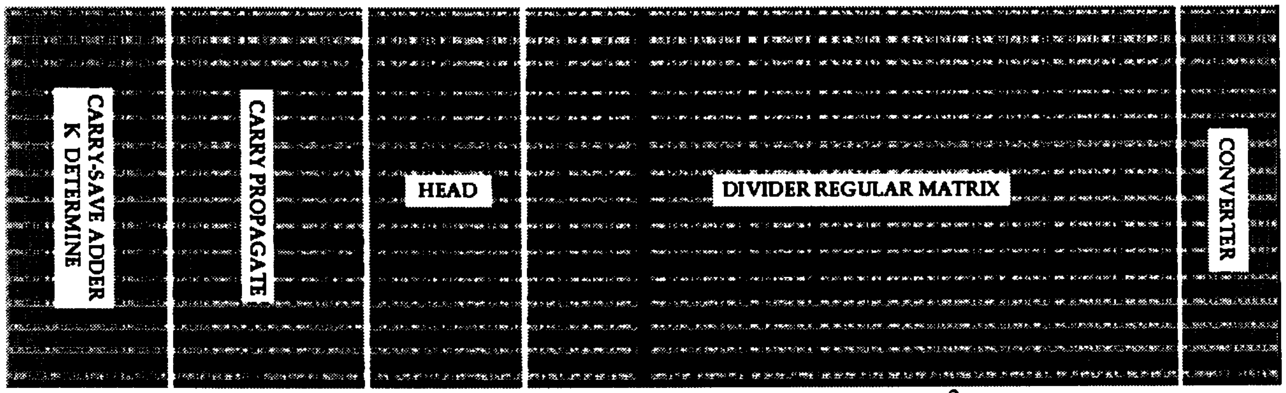


Figure 15: 32-bit pseudo-radix-4 (not to scale): 37.6 mm^2

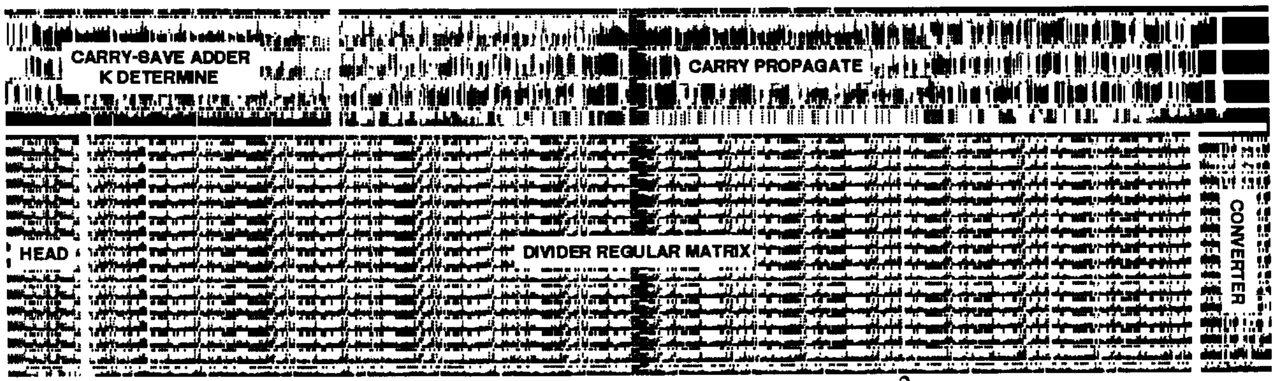
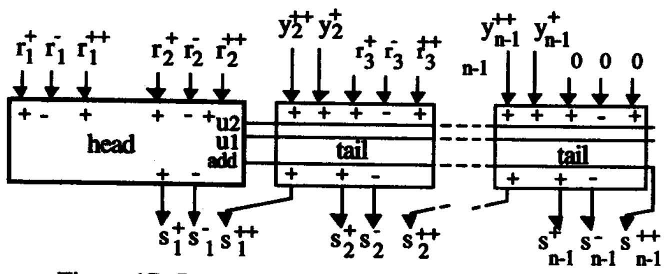


Figure 16: 32-bit radix-4 (not to scale): 35.2 mm^2

Divider	8 bits		16 bits		32 bits	
	Prop Time	Area mm^2	Prop Time	Area mm^2	Prop Time	Area mm^2
radix-2	18 ns	1,8	33 ns	5,7	65 ns	28
rseudo-radix-4	32 ns	3,3	58 ns	10,3	117 ns	39,5
radix-4	30 ns	3,2	56 ns	9,6	110 ns	35,2

Table III Comparisons of the 3 approaches

The algorithm convergence is formally proved in [16].



r^{+}_{1} r^{-}_{1} r^{++}_{1}
r^{+}_{2} r^{-}_{2} r^{++}_{2}
y^{++}_{2} y^{+}_{2}
r^{+}_{3} r^{-}_{3} r^{++}_{3}
y^{++}_{n-1} y^{+}_{n-1}
n-1
0 0 0
head
u2
u1
add
tail
tail
s^{+}_{1} s^{-}_{1} s^{++}_{1}
s^{+}_{2} s^{-}_{2} s^{++}_{2}
s^{+}_{n-1} s^{-}_{n-1} s^{++}_{n-1}

Figure 17: Structure of a slice of the radix-4 divider.

8.2 Equations of the tail

The TAIL cells (Fig. 17) select a multiple of the divisor Y, according to u1 and u2 then XOR it with add and compute the rest of the s_i digits according to equation

$$4 * s^{++}_{i-1} + s^{+}_{i} - 2 * s^{-}_{i} = -2 * r^{-}_{i+1} + r^{+}_{i+1} + r^{++}_{i+1} + \sigma_j .$$

9. Layout

Divider generators for 4 to 128 bits operands were written with GenOptim. Figures 14, 15 and 16 give the layout of 32 bits dividers generated with a standard cell library [2] in 1.0 µm 2-metal CMOS.

10. Conclusion

Going to higher order radixes seems a promising technique [13,14,15,16] to improve the speed/area trade off for division. It is straightforward to generalise this paper approach to any pseudo power-of-two radixes. The head given in Figure 9 is very regular and can be stretched from three BS input digits to four, five or more, with a linear delay. That means that only marginal speed improvement can be expected.

Radix 8 would require the range reduction to [1, 1+1/16] and the precomputation of 7*D and 5*D besides 3*D, that would more than double the overhead of the pseudo-radix-8 divider as compared to the pseudo-radix-4. Real radix 4 (R and D in radix 4) requires more complex head and tail cells, but no 3*D, resulting in less overhead and less connections. The main problem when going to higher-order radixes is a regularity and connectivity problem: there are many more functional blocks and many more wires. Those problems are difficult to address, modelling is unreliable and exploration through synthesis is a necessity [18]. A fast and convenient environment like GenOptim becomes a must.

11. References

[1] M.D. Ercegovac & T. Lang " *Division and Square Root: Digit-Recurrence Algorithms and Implementations* Kluwer Academic Publishers, The Netherlands, 1994

[2] A. Greiner & F. Pêcheux "*ALLIANCE: A complete Set of CAD Tools for teaching VLSI Design*", proc. 3rd Eurochip Workshop, Grenoble, France, Sept., 1992.

[3] "*The VHDL Language Reference Manual* " , Intermetrics, Inc., version 7.2, Aug. 1985.

[4] A. Avizienis " *Signed-digit number representation for fast parallel arithmetic* " IRE Transaction on Electronic Computer vol. EC-10 September 1961.

[5] J.E. Robertson " *The correspondence between methods of digital division and multiplier recoding procedures* " IEEE Transactions on Computers, Vol. C-19 N°8 August 1970

[6] K.D. Tocher " *Techniques of multiplication and division for automatic binary computers* " Quart. Journ. Mech. Appl. Math. Vol. 11 N° 3 1958

[7] A. Guyot, Y. Herreros & J.M. Muller " *Janus, an on line multiplier-divider for large numbers* " proc. 9th IEEE Symposium on Computer Arithmetic, Santa Monica, September 1989

[8] A. Vandemeulebroecke, E. Vanzieleghem, T. Denayer & P. Jespers "*A new carry free division algorithm and application to a single chip 1024-b RSA processor* " IEEE J. Solid-State Circuits,V.25, 1990

[9] J. William & V.C. Hamacher "*A linear-time divider array* " Canadian Elec.Eng.Jour. Vol. 6 n°4 1981

[10] N. Burgess " *A fast division algorithm for VLSI* " proc. ICCD 91, Cambridge, October 1991

[11] A. Svoboda "*An algorithm for Division* " Information Processing Machine vol. 9, March 1963

[12] L. Montalvo, B. Behnam, T. Vasileva & A. Guyot " *CMOS Implementation of an hybrid radix-4 divider* " proc. ESSCIRC'94, Ulm, Germany, Sept. 1994.

[13] H.R. Srinivas & K.K. Parhi "*A fast Radix 4 division algorithm* " proc. ISCAS 94, London, UK, June 1994

[14] J. Fandriano " *Algorithm for high speed radix 4 division and radix 4 square root* " proc. 8th Symposium on Computer Arithmetic, Como, Italy, June 1987

[15] S.E. McQuillan, J.V. McCanny & R. Hamill " *New algorithm and VLSI Architecture for SRT Division and Square Root* " proc. 11th Symposium on Computer Arithmetic, Windsor, Ontario, June 1993

[16] L. Montalvo & A. Guyot "*A minimally redundant radix-4 divider with operands scaling* " proc. IX congresso de Diseño de Circuitos Integrados, DCIS'94, Gran Canaria, Spain, Nov. 1994

[17] L. Montalvo " *Number systems for hight performance dividers* " PhD Thesis, INPG, Grenoble, March, 1995

[18] A. Guyot, L. Montalvo, A. Houelle, H. Mehrez & N. Vaucher "*Comparison of the layout synthesis of radix-2 and pseudo-radix-4 dividers* " Proc. VLSI Design'95, New Delhi, India, January 1995