THE FELIN ARITHMETIC COPROCESSOR CHIP

M. Cosnard, A. Guyot, B. Hochet, J.-M. Muller, H. Ouaouicha, P. Paul and E. Zysman

CNRS, Lab. TIM3, INPG, 46 Av. Félix-Viallet, 38031 Grenoble Cedex, FRANCE.

ABSTRACT

We describe a general VLSI architecture for the computation of arithmetic expressions including floating-point trancendental functions. This architecture is divided in three parts : a communication machine, the control part of a computation machine and the operative part of this computation machine. In order to compute the most usual trancendental functions, we introduced some general algorithms, presented briefly here, including as a particular case the CORDIC scheme. Our major architecture goals were regularity, parametrization and automatic design. The final chip is designed in a 2-Alu CMOS technology, and its name is FELIN ("Fonctions ELémentaires INtégrées is the french for "integrated elementary functions").

This work was supported in part by the GRECO C^3 and the GCIS of the French CNRS.

INTRODUCTION.

The FELIN chip is composed of two machines : a communication machine and a computation machine. The computation machine stores the operators and the corresponding operands of the arithmetic expressions to be computed. It then generates an execution profile, sends an mathematical function and its arguments to the computation machine and stores intermediate results. The computation machine is divided in two parts. The control part decomposes the mathematical function into a sequence of several functions of an elementary level, and then generates the corresponding procedure chain and for each procedure the associated algorithm. The operative part executes the program generated by the control part.

FELIN contains a directory of 25 elementary functions, that includes trigonometric and hyperbolic sine, cosine and tangent with their reverse, logarithm and exponential in base 2, e and 10, square root, x to the power y, and of course add, substract, multiply, divide, and remainder as defined by the IEEE standard. The chip handles single (32b), double (64b) and extended precision (80b) floating point numbers. It supports neither integers nor decimal. All trancendental instructions are computed with **unlimited argument range.**

The circuit is designed for a 2μ gate double metal CMOS process and packed in a 40 pin dil. It measures 7.1 x 5.9 mm and contains over 90,000 transistors.

I - Algorithms.

The computation of an elementary function f needs generally 3 steps. In the first step, the given argument is reduced to a related argument in a restricted domain : this step is called the **range reduction**. In the second step the function value (or a related one) is computed for the reduced argument. In the third step, the function value for the original argument is deduced from the result of the second step.

Example : Let us suppose that we want to compute *sin* (10000), with an algorithm which works on the interval $[-\pi/2 , \pi/2]$.

first step. We compute N (integer) and $y \in [-\pi/2 , \pi/2]$ such that $10000 = N\pi + y$.

We obtain $N = 3183$
$y = 0.3105836\ldots$

second step. We compute *sin* (y) = 0.305614…

third step. Since N is odd, we have *sin* (x) = - *sin* (y).

Here, we shall present only the algorithms used to compute functions values for the reduced arguments. For range reduction, FELIN uses an algorithm introduced in [6],[7].

Some hardware algorithms for computating the elementary functions have been previously proposed. 300 years ago, Briggs, a contemporary of Neper, introduced an algorithm for computing the logarithmic function. In 1959, J. Volder [8] introduced the CORDIC scheme (COordinate Rotations on a DIgital Computer), which enables us to compute the trigonometric functions using only additions and shifts. In 1971, J. Walther [9] showed that CORDIC can be extended to the hyperbolic and arithmetic fun-

ctions. The CORDIC scheme is based upon the following iteration :

$$x_{n+1} = x_n - m d_n y_n 2^{-n}$$
$$y_{n+1} = y_n + d_n x_n 2^{-n}$$
$$z_{n+1} = z_n - d_n e_n$$

Where the choices of m, d_n and e_n are presented in fig1.

$$\begin{cases} K = \prod_{i=0}^{\infty} \mathrm{Cos}\, e_i \\ K' = \prod_{i=0}^{\infty} \mathrm{Ch}\, e_i \end{cases}^{**} \qquad \begin{cases} x_{n+1} = x_n - m d_n y_n 2^{-n} \\ y_{n+1} = y_n + d_n x_n 2^{-n} \\ z_{n+1} = z_n - d_n e_n \end{cases}$$

function	initial values	m	e_n	d_n	results
sine/cosine	$x_o = K$ $y_o = 0$	1	$\mathrm{Arctg}\, 2^{-n}$	$\mathrm{sign}\, z_n$	$x_n \longrightarrow \cos z_o$ $y_n \longrightarrow \sin z_o$
Arctg	$z_o = 0$	1	----	$-\mathrm{sign}\, y_n$	$z_n \longrightarrow \mathrm{arctg}\, y_o / x_o$
sh/ch **	$x_o = K'$ $y_o = 0$	-1	$\mathrm{Argth}\, 2^{-n}$	$\mathrm{sign}\, z_n$	$x_n \longrightarrow \mathrm{ch}\, z_o$ $y_n \longrightarrow \mathrm{sh}\, z_o$
Argth **	$z_o = 0$	-1	----	$-\mathrm{sign}\, y_n$	$z_n \longrightarrow \mathrm{argth}\, y_o / x_o$
multiplication	$y_o = 0$	0	2^{-n}	$\mathrm{sign}\, z_n$	$y_n \longrightarrow x_o z_o$
division	$z_o = 0$	0	2^{-n}	$-\mathrm{sign}\, y_n$	$z_n \longrightarrow y_o / x_o$

N.B. The symbol ** means that in the iteration and in the infinite product the terms 4, 13, 40, 121, ... , k , 3k+1 have to be repeated.

Fig. 1 The CORDIC algorithm.

The algorithms of Briggs, Volder and Walther are particular cases of a class of algorithms which can be build using a new mathematical tool which generalizes the numeration basis : the **discrete basis** [5], [6]. We shall now describe briefly this notion.

The 2 following theorems have been proved in [5].

Definition 1 : let B = (e_n) be a decreasing infinite sequence of positive reals such that

$$\sum_{n=0}^{\infty} e_n < +\infty$$

B is called a **discrete basis of order p** if the set :

$$\left\{ \sum_{n=0}^{\infty} d_n e_n \ , \quad d_n \in \{0,1, \ldots ,p\} \right\}$$

is an interval.

For instance, the sequence (10^{-n}) is a discrete basis of order 9 since each number x included in the interval [0 , 10] can be written :

$$x = \sum_{n=0}^{\infty} d_n 10^{-n}$$

the d_i are the digits of the classical radix 10 decomposition of x. This example shows that the notion of discrete basis generalizes that of numeration basis.

Theorem 1. B is a discrete basis if and only if for any integer n :

$$e_n \le p \cdot \sum_{k=n+1}^{\infty} e_k$$

This result shows that, for instance, the sequences (a^{-n}), $(\mathrm{Arctg}\ (a^{-n}))$ and $\ln(1+a^{-n})$, $a > 1$, are discrete basis of order $\lceil a-1 \rceil$, where $\lceil u \rceil$ is the lowest integer greater or equal than u. for instance, since the sequence (π^{-n}) is a discrete basis of order 3, one can write in "base π" :

"1/2 = 0.11211202..."

since

$$\frac{1}{2} = \sum_{n=0}^{\infty} d_i \pi^{-i} \quad \text{with} \quad d_0 = 0, d_1 = d_2 = 1, \ldots$$

Theorem 2 : if B = (e_n) is a discrete basis of order p, then the two following algorithms give

$$t = \sum_{n=0}^{\infty} d_n e_n = \lim_{n \to \infty} t_n$$

1-unidirectional algorithm.

$t_o = 0$

$d_n = \max \{ j \le p \ , \ t_n + j e_n \le t \}$

$t_{n+1} = t_n + d_n e_n$

2-bidirectional algorithm.

$t_o = 0$

if $t_n \le t$

then $d_n = \max \{ 1 \le j \le p \ , \ t_n + (j-1) e_n \le t \}$

else $d_n = \min \{ -p \le j \le -1 \ , \ t_n + (j+1) e_n \ge t \}$

$t_{n+1} = t_n + d_n e_n$

Example of application : computation of the exponential function.

Let us suppose that we want to compute the exponential function, with a computing system of numeration basis B. Since the sequence $(e_n) = (ln\ (1 + B^{-n}))$ is a discrete basis of order B-1 , using the unidirectional algorithm, one can write any number x included in $[0, \Sigma e_n]$ in the form :

$$x = \sum_{n=0}^{\infty} d_n \ln (1 + B^{-n})$$

and obtain :

$$e^x = \prod_{n=0}^{\infty} (1 + B^{-n})^{d_n}$$

This last result is easily computable since, in a radix-B system, a multiplication by $(1 + B^{-n})$ reduces to a shift and an addition. The infinite sums and product are truncated to an order N, according to the desired accuracy. Thus, we obtain the following algorithm :

```
input : the argument t ; output : exp.
e_n = ln (1+B^-n) are precomputed and stored
constants.
Begin
    x := 0 ;
    exp := 1 ;
    For k := 0 to N do
    begin
      d := 0 ;
       u := 0 ;
       while (u<t) and (d<B-1) do
            begin
            u := x + e_n ;
            if u ≤ t then
              begin
                 x := u ;
                 exp := exp + exp.B^-k
              end ;
            d := d + 1
       end
   end
End.
```

The relative error on the result *exp* is approximately B^{-N}.

If we decompose a number x using the discrete basis of order 1 ($Arctg\ 2^{-n}$) in order to compute the trigonometric functions of x on a radix 2 computer, we find the CORDIC algorithm of Volder (see [6] for instance). Some other functions (hyperbolic functions, exponential and logarithmic functions, square root…) can be computed using that notion of discrete basis and a similar notion, defined in [5] : that of **multiplicative discrete basis**, which enables us to write numbers as **products** of precomputed constants.

For example, any number x included in a given interval can be written as :

$$x = \prod_{i=0}^{\infty} (1 + B^{-i})^{d_i} \quad d_i \in \{0,1, \dots, B-1\}$$

and thus we can obtain the logarithm of x as :

$$\ln (x) = \sum_{i=0}^{\infty} d_i \ln (1 + B^{-i})$$

We use radix 2 to implement these algorithms on a chip, in order to get reliability with IEEE floating-point standards. In [7] we show that the range reduction can be effected efficiently using an algorithm very similar to the bidirectional algorithm.

II - The FELIN chip.

The floor plan of the machine is given by fig. 2.

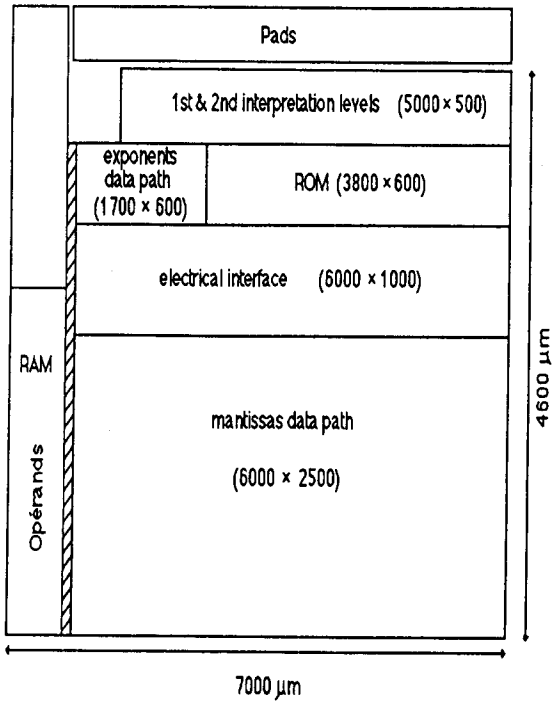

Fig. 2 : Floor plan of FELIN

II - 1 Operative part of the computational machine.

For regularity reasons, we use an internal fixed point format of 128 bits, with 22 bits of integer part and 106 bits of frational part : 64 of them will be given after the computation. Such a format simplifies the computation, because it avoids the normalizations needed by floating point formats. A little 16 bits operative part is associated to it in order to keep and treat the exponents of the arguments, for some functions. The operative part owns some registers and a ROM for the constants of the algorithms, and an adder and a shifter, supplied by two busses. Both operators may work independently for simple computations, which only need addition or shift. But a third local bus allows pipelining between the shifter and the adder, thus performing efficiently the computation $A+B\cdot 2^{-i}$, since the algorithms derived from the discrete basis theory are all based on this primitive. The synoptics is given as follows (fig. 3).

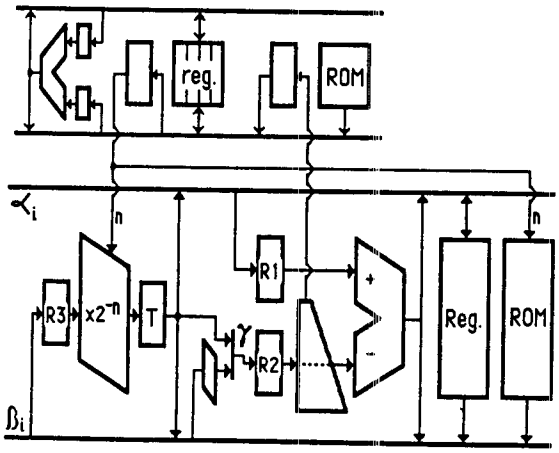

Figure 3

The busses are sequenced in four phases : precharge, read, compute and write. We give in fig. 4 the sequencing of the computation of the well-known CORDIC scheme. Notice that both main busses are used independently during the read phase, and are used together to write back the registers.

CORDIC	Read phase	Operation phase	Write phase
cycle 1	$X_n \rightarrow R1$ $X_n \rightarrow R3$ $T(Y_n 2^{-n}) \rightarrow R2$	$X_n \oplus Y_n 2^{-n}$ Shift (X_n, n)	$X_n \oplus Y_n 2^{-n}$ $\alpha, \beta \rightarrow$ Register
cycle 2	$Y_n \rightarrow R1$ $E_n \rightarrow R3$ $T(X_n 2^{-n}) \rightarrow R2$	$Y_n \oplus X_n 2^{-n}$ Shift (E_n, n)	$Y_n \oplus X_n 2^{-n}$ $\alpha, \beta \rightarrow$ Register
cycle 3	$Z_n \rightarrow R1$ $Y_{n+1} \rightarrow R3$ $T(E_n 2^{-n}) \rightarrow R2$	$Z_n \oplus E_n 2^{-n}$ Shift $(Y_{n+1}, n+1)$	$Z_n \oplus E_n 2^{-n}$ $\alpha, \beta \rightarrow$ Register

⊕ means either + or - depending on the sign of Zn

Fig. 4

For topological efficiency, the operative part is divided in two subparts, each one containing 64 bits, and communicating through the shifter. The input/ output access is made with the Most Significant 64-bits part. The critical component is the 128 bits adder. A two-level "carry-skip" speed-up chain has been implemented with variable length blocks. Theoritical bases and detailed implementation are given in [4]. The two-level carry skip technique is very efficient for large adders, and few space-consuming. Electrical simulations give a computation time less than 30 ns.

II - 2 Control part of the computing machine.

The implementation of the control part takes into account the following specifications :
The operative part is designed to execute some elementary functions, derived from the discrete basis theory. The computation of these functions also requires some preparing and formating operations, such as operand translations, detection of exceptions, …etc, which primitives are additions and/or shifts.
For a given kernel of elementary functions directly executable using our algorithms, we decided to implement composed functions, which can be decomposed in elementary functions. For example :
y^X = exp(X·LogY) is be decomposed in :

computation of LogY
then computation of X·LogY
then exponentiation of the preceding result.

The algorithms are straightforward decomposed in three interpretation levels :

1) The first level decomposes the general function into a sequence of one or several directly executable functions.

2) The second level will generate a list of procedure calls (that we shall call "procedure chain") in the third level, that includes range reduction, exceptions detection, computation, ...

3) All the algorithms executed by the operative part are stored in a ROM that implements the 3rd level.

All the "procedure chains" we must deal with belong to the same family. Thus, in order to simplify the interpretation of the functions, we introduce a high parametrization degree.

Both first levels are implemented in a 6-bits wide bit-slice machine with 10 registers, incrementers and 2 PLAs. The first one selects some of the registers, the second computes the parameters and sequences the machine itself (see fig.5)

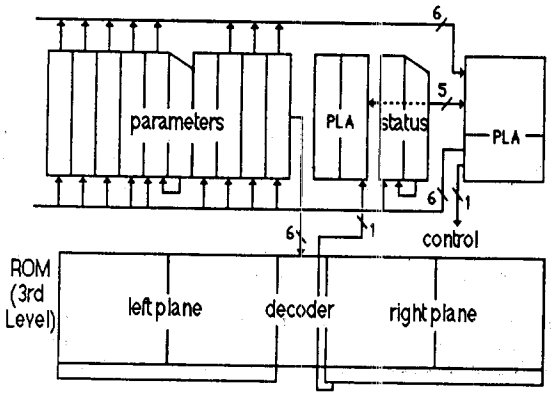

Figure 5

Such a structure is very versatile because its architecture is totally transparent to any evolution of the specifications. Registers may be added or removed and the PLA may be reprogrammed. On the other hand, the parametrization and the regularity of this block make procedural generation possible.

The third level is implemented in a multiplexed ROM. The main advantage of multiplexing it lies in the fact that each microinstruction links up with several others. So the good one may be choosen by tests at the latest moment. Thus feedback loops are of short (1 clock cycle) and the velocity of the algorithms is enhanced. This architecture is efficient, because it suits perfectly with the topology of the whole circuit. Its regularity allows also an easy procedural generation.

The computing machine of FELIN is able to evaluate directly :

- addition, substraction, multiplication, division
- sine, cosine, arctangent
- logarithm and exponential functions
- square root

and by decomposition :

- arcsin, arcos, tangent
- sh, ch, argsh, argch, th, argth
- y^x , square

Details of computation and range reduction are given in [6], and details of implementation are given in [3].

II - 3 The communication machine.

FELIN evaluates expressions in reverse polish notation. An input file for operands, an output file for results and an evaluation stack all take place in a 16 words 83 bits double access RAM. This RAM is adressed through chaining controlled by a specific 4 bits data path.

In order to increase speed and eliminate initialisation problems, an hardwired garbage collector has been designed. Two other files hold instructions (13 bits) and status (5bits).

CONCLUSION AND PERFORMANCES.

We have build a general class of algorithms and a general architecture designed in order to implement them. The basic difference between FELIN and other co-processors like the INTEL 8087 or the MOTOROLA 68881 is FELIN's ability to evaluate expressions. We are currently testing a first run of FELIN. Tests and electrical simulations enable us to give some approximate performances of FELIN with a 16 MHz clock. In Fig. 6, these performances are compared with those of two commercially available arithmetic co-processors, the MOTOOLA 68881 and the INTEL 8087.

	FELIN	MC68881	8087
Multiplication	6 µs	3.1µs	18.1µs
Division	8.7 µs	3.8µs	25.4µs
Sine/Cosine	14.2 µs	23µs	?
Square root	11.2 µs	?	23.3µs

Fig. 6

In order to facilitate testing, we chosed to run separately the different parts of FELIN. Fig. 7 presents a photomicrograph of the operative part of the computing machine.

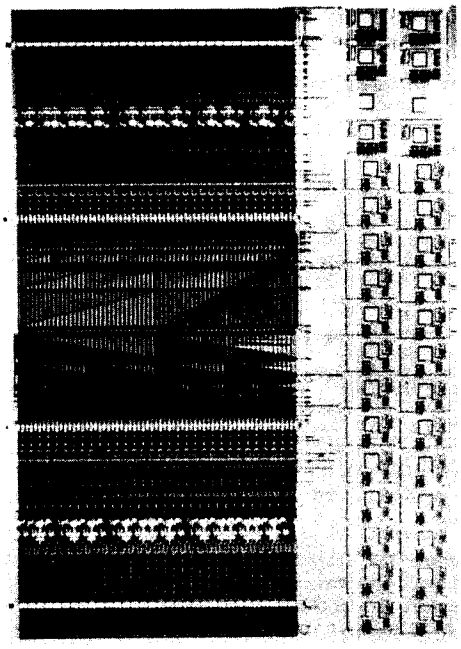

Fig. 7

REFERENCES

[1] J.J. Coonen, "An implementation guide to a proposed standard for floating-point arithmetic", *COMPUTER* , Jan. 1980.

[2] M. Cosnard, A. Guyot, B. Hochet, J.M. Muller, H. Ouaouicha and E. Zysman, "FELIN, an elementary function cruncher", *Future trends on computing* , Dec. 85, Edited by J. Wiley & sons, 1986.

[3] B. Hochet, "Conception de VLSI et applications au calcul numérique", Thèse de doctorat, Institut National Polytechnique de Grenoble, Grenoble, France, Jan. 1987.

[4] B. Hochet and J.M. Muller, "A way to build efficient carry-skip adders", RR. No 583, Institut IMAG, Grenoble, France, 1986.

[5] J.M. Muller, "Discrete basis and computation of elementary functions" , *IEEE Transactions on computers* , Sept. 1985.

[6] J.M. Muller, "Méthodologies du calcul des fonctions élémentaires" , Thèse de doctorat, Institut National Polytechnique de Grenoble, Grenoble, France, Sept. 1985.

[7] J.M. Muller, "Hardware computation of elementary functions including range reduction", to appear in *IEEE Transactions on computers.*

[8] J. Volder, "The CORDIC computing technique", *IRE Trans. on Computers* , Sept. 1959.

[9] J. Walther, "A unified algorithm for elementary functions", *Spring joint computer conference proc.* , Vol. 38, Sept. 1971.