Asynchronous SRT Dividers: The Real Cost

H. Boutamine*, A. Guyot*, B. Elhassan**, M. Renaudin**
*TIMA Laboratory
Institut National Polytechnique de Grenoble
46, Avenue Félix Viallet - F38031 Grenoble Cedex, France
**CNET, France Telecom
BP98, Chemin du Vieux Chêne F38243 Meylan Cedex, France
E-mail: Hicham.Boutamine@imag.fr

Abstract

Synchronous systems represent the majority of digital circuits built, essentially because they are easier to design and test. However asynchronous approaches are becoming more attractive to designers because of the potential advantages brought in terms of power consumption and delay, and also favoured by the increased sophistication of today's CAD tools. An important topic in asynchronous research is the SRT (Sweeny, Robertson, Tocher) self-timed divider. In this paper we compare two versions of a 32-bit SRT divider, synchronous and asynchronous. Our results show that the asynchronous circuit is faster, but consumes more power over an increased area.

1. Introduction

The major parts of digital circuits now designed are synchronous and the main reason for their widespread use is the simplicity of the design and test. This has allowed CAD tool designers to develop powerful tools for this purpose. In a synchronous system, a designer can simply define the combinational logic necessary to compute the given functions and surround it with latches. By setting the clock rate to a large enough period, all worries about hazards (undesired signal transitions) and the dynamic state of the circuit are removed.

Asynchronous design has been an active area of research since at least the mid 1950's but has yet to achieve widespread use. The motivations for using asynchronous systems are many and can be summarised as follows :

•No clock skew: Since asynchronous circuits by definition have no globally distributed clock, there is no need to worry about clock skew. In contrast, synchronous systems often slow down their circuits to accommodate the skew. As feature sizes decrease, clock skew becomes a much greater concern.

•Lower power: Standard synchronous circuits have to toggle clock lines, and possibly precharge and discharge signals, also in portions of the circuit that are unused in the current computation. Although asynchronous circuits often require more transitions on the computation path than synchronous circuits, they generally have transitions only in areas involved in the current computation.

•Average case instead of worst-case performance.

•Easing of global timing issues : In systems such as a synchronous microprocessor, the system clock and thus system performance is dictated by the slowest (critical) path. Thus several portions of a circuit must be carefully optimized to achieve the highest clock rate, including the rarely used portions of the system. Since many asynchronous systems operate at the speed of the circuit path currently in operation, rarely used portions of the circuit can be left unoptimized without adversely affecting system performance

•Automatic adaptation to physical properties: The delay through a circuit can change with variations in fabrications, temperature, and power supply voltage. Synchronous circuits must assume that the worst possible combination of factors is present to clock the system accordingly. Many asynchronous circuits sense computation completion, and will run as quickly as the current physical properties allow [1].

The subtractive divide algorithms used in current microprocessors fall into the broad category of SRT (Sweeny, Robertson, Tocher) methods [2] [3]. These algorithms use redundant representation of values and treat groups of consecutive bits as single higher radix digits in order to enhance performance [4]. All known implementations are synchronous [5]. Research in the asynchronous SRT divider subject is very important though, recursive nature of the algorithm makes it very appropriate to self-timed architectures. This study evaluates the cost of both the asynchronous and the synchronous SRT dividers in terms of power consumption, area and speed. For this purpose we have layed out a 32-bit

ED&TC '96
0-89791-821/96 $5.00

synchronous and asynchronous SRT divider and the comparison reported here has been done using the electrical simulator Eldo for performance, number of transistors and power dissipation . The area is directly measured on the layout.

Section II of this paper reviews the digit-recurrence division algorithm, the implementation of the basic cells and its extension to the synchronous divider. Section III presents the structures used to implement the asynchronous divider. Section IV presents the comparison results and discusses them. Finally, Section V presents the conclusions.

2. The division algorithm

The digit-recurrence approach of the division Q = A ÷ D relies on subtraction and on multiplication by the radix b and by the quotient digit q_j.

Step 0 *(initialisation)*

$R^{(0)} = A - D$; $Q^{(0)} = 1$

Step j *(iteration)* $1 \le j \le n$

$R^{(j)} = R^{(j-1)} - q_j * D * b^{-j}$;

$Q^{(j)} = Q^{(j-1)} + q_j * b^{-j}$

It is easy to verify that $A = Q^{(j)} * D + R^{(j)} \ \forall$ j.

2.1. SRT radix two division: Borrow-save division

Let us give a table of the notations and variables used along this part: ⊕, ∨, . denote XOR, OR and AND; +, –, *, ÷ denote respectively add, subtract, multiply and divide. i, j, n represent integers.

Number	Digit	
A	a_i	dividend or radicand
D	d_i	divisor (0 for square root)
$Q^{(j)}$	q_i $q_i^+ - q_i^-$	j^{th} quotient or square root in redundant notation
$U^{(j)}$	u_i	$= Q^{(j)}$ in standard notation
$V^{(j)}$	v_i	$=U^{(j)} - 2^{-j}$ in standard notation
$R^{(j)}$	r_i $s_i^{(j)} - t_i^{(j)}$	j^{th} partial remainder in redundant notation
$\hat{R}^{(j)}$	$r_{-2} r_{-1} r_0$	a 3-digit coarse estimate of $R^{(j)}$
$LSB^{(j)}$	lsb_i	$LSB^{(j)} = j$, i.e. $lsb_1^{(j)} = 1$ (i = j)

j is the recurrence index and i is the position of a digit in an n-digit number.

$a_i , d_i , q_j^+ , q_j^- , u_i , v_i , s_i, t_i \in \{0,1\}$,

$q_j , r_i \in \{-1,0,+1\}$, $q_j = q_j^+ - q_j^-$, $r_i = s_i - t_i$.

Let $R^{(j)}$ be $\sum_{i=-2}^{n} r_i^{(j)} * 2^{-i} = \sum_{i=-2}^{n} (s_i^{(j)} - t_i^{(j)}) * 2^{-i}$.

$R^{(j)}$ has two most significant digit more than D.

The first one is because the range of the partial remainder is twice as large as D, the other one is due to the overflow characteristic of redundant notations [4] [6] [7] [8]. The two roles of the head cell (Fig. 1) are:

- 1: to select q_j and the operation to execute,
- 2: to execute this operation on the head digits.

The tail cells role (Fig. 1) is to execute the operation dictated by the head on the tail digits.

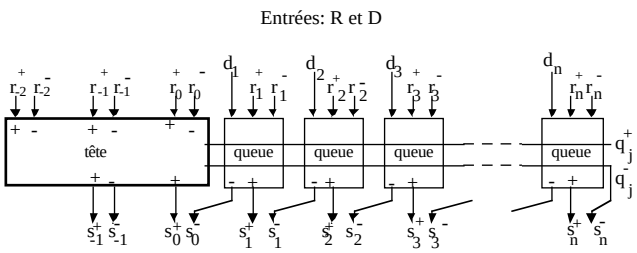

Figure 1: A row of head and tail cells for division

2.2. Equations of the Head and Tail cells

Since $D = 1 + \sum_{i=1}^{n} d_i * 2^{-i}$ then

$-D = -2 + \sum_{i=1}^{n} \overline{d_i} * 2^{-i} + 2^{-n}$ because D is in 2's complement notation.

So the iteration $R^{(j)} = R^{(j-1)} - q_j * D * 2^{-j}$ becomes:

$R^{(j)} = R^{(j-1)} + q_j^- * (1 + \sum_{i=1}^{n} d_i * 2^{-i}) * 2^{-j} + q_j^+ * (-2 + \sum_{i=1}^{n} \overline{d_i} * 2^{-i} + 2^{-n}) * 2^{-j}$

The head cell preserves the identity:

$\hat{S}^{(j)} = \hat{R}^{(j-1)} + q_j^- + q_j^+ * (-2)$

and the tail cells preserve the identity :

$(2 * s_{i-1}^{(j)} — t_1^{(j)}) = (s_i^{(j-1)} — t_i^{(j-1)} + q_j^- . d_i + q_j^+ . \overline{d_i})$.

The bit $s_{n-1}^{(j)}$ weighting $(+2^{-n-j})$ is wired to q_j^+ .

This identity translates into the following logic equations for the five outputs of the head cell (Fig.1):

$$q_j^+ = s_{-2}^{(j-1)} \vee \overline{t_{-2}^{(j-1)}} (s_{-1}^{(j-1)} . \overline{t_{-1}^{(j-1)}} \vee \overline{t_{-1}^{(j-1)}} . s_0^{(j-1)} . \overline{t_0^{(j-1)}} \vee s_{-1}^{(j-1)} . s_0^{(j-1)} . \overline{t_0^{(j-1)}}) ;$$

$$q_j^- = t_{-2}^{(j-1)} \vee \overline{s_{-2}^{(j-1)}} (\overline{s_{-1}^{(j-1)}} . t_{-1}^{(j-1)} \vee \overline{s_{-1}^{(j-1)}} . \overline{s_0^{(j-1)}} . t_0^{(j-1)} \vee t_{-1}^{(j-1)} . \overline{s_0^{(j-1)}} . t_0^{(j-1)}) ;$$

$$s_{-2}^{(j)} = s_{-2}^{(j-1)} (s_{-1}^{(j-1)} \vee \overline{t_{-1}^{(j-1)}} \vee s_0^{(j-1)} . \overline{t_0^{(j-1)}}) \vee \overline{t_{-2}^{(j-1)}} . s_{-1}^{(j-1)} . \overline{t_{-1}^{(j-1)}} . s_0^{(j-1)} . \overline{t_0^{(j-1)}} ;$$

$t_{-2}^{(j)} = t_{-2}^{(j-1)} (\overline{\overline{s_{-1}^{(j-1)}}} \vee t_{-1}^{(j-1)} \vee \overline{\overline{s_{0}^{(j-1)}}} . t_{0}^{(j-1)}) \vee \overline{\overline{s_{-2}^{(j-1)}}} .$

$\overline{\overline{s_{-1}^{(j-1)}}} . t_{-1}^{(j-1)} . \overline{\overline{s_{0}^{(j-1)}}} . t_{0}^{(j-1)}$;

$s_{-1}^{(j)} = \bar{q_j} \oplus s_{0}^{(j-1)} \oplus t_{0}^{(j-1)}$ {only digits with weight 2^0}

2.3. Synchronous division

The synchronous divider can be built easily from the slice described above. We only have to add a Flip-Flop range and a multiplexer range. The role of the Flip-Flops is to keep stable the outputs of the slice which become the inputs of the divider, for calculating the next quotient, and the role of the multiplexers is to toggle between the first dividend which is external and the next dividends which are generated by the slice itself (the rest of the division). We can also use one or more slices, and the choice of the number of stages depends on the goals fixed by the dsigner.

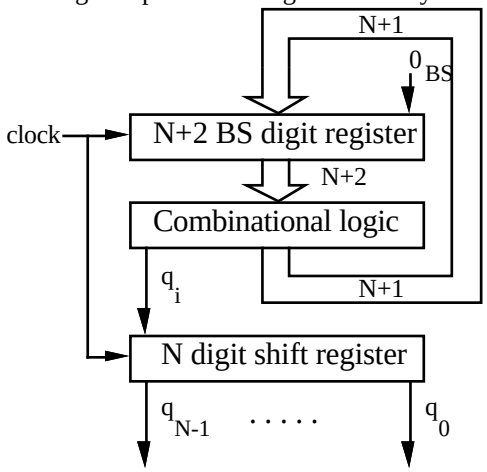

Figure 2: The synchronous divider

3. Asynchronous division

The asynchronous divider uses a self timed ring as proposed by Williams to control a sequence of function blocks described in the slice above [10]. Following his methodology, an elementary stage was first designed using DCVS Logic cells in order to minimise the forward latency Lf. An elementary stage with its local controller is described in Figure 3. The structure of the head and tail cells (Fig 4) was motivated by the use of elementary DCVSL cells available from an asynchronous standard-cells library previously designed [14].

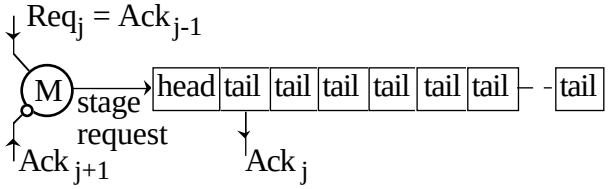

Figure 3: An elementary stage. The M component is a Muller C element.

3.1. Timing assumptions

As shown in Fig. 3, the acknowledge signal Ack_j is issued from the first tail cell. Sensing only the output of the first tail cell relies on the assumption that the Ack_j signal is active only after the outputs of the head cell and of all the tail cells are valid. This assumption is realistic since the Ack_j signal takes some time to be generated (a nand gate) and its effect on the next stage is delayed by a Muller C element (Fig. 3 & 5).

3.2. Ring structure optimisation

From electrical simulations of a netlist extracted from the layout of a single stage, the values of "P" and "Lf" defined in [10] were computed. Then the optimum number of stages to be used in the ring is easily derived. For this circuit, the optimum number is four.

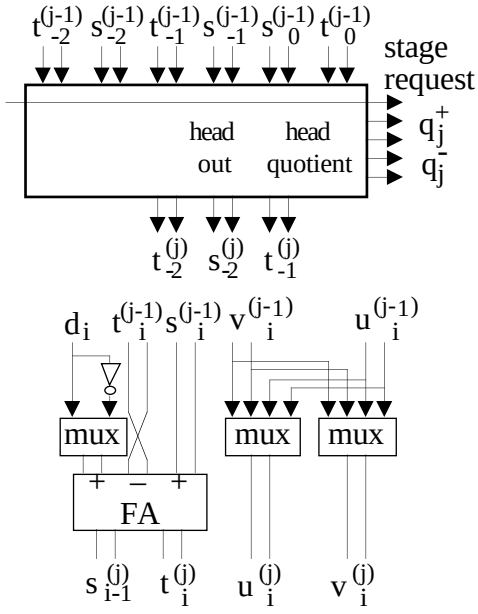

Figure 4: Structure of the head and tail cells for division. Bits are dual-rail coded.

3.3. Self-timed ring structure

Thus, the self-timed ring was build by assembling four rows (Fig. 1,3,5). Self synchronisation in the ring is obtained through four looped Muller C elements (Fig. 5). The Muller C elements make sure that while stage j is computing, stage (j-1) $_{\text{mod } 4}$ in the ring is holding its output data, while stages (j-2) mod 4 and (j-3) mod 4 are precharging.

The four-stage structure in conjunction with this simple control scheme insures that the best possible throughput is achieved because under these conditions the ring operates with a single token in the Data-Limited region [10]. The token always flows forward into an unoccupied stage.

3.4. Interface controller

An interface controller has to be added to the ring to cleanly initiate and stop the computation. Signal Req_{in} tells the divider that a new operand is ready, and Ack_{out} signals the completion of the division (Fig. 5). The interface controller CTRL was first modelled using a Signal Transition Graph, and then synthesized following a classical method [13]. Finally it was verified that the logic implementation of the controller was hazard free [14].

When Req_{in} is active, the controller allows the operand to be inserted in the ring via the input multiplexer, and then starts the computation using the Req_{en} signal. The circuit is initialised as follows:
$R^{(0)} = A - D$, $s_i^{(0)} = a_i$, $t_i^{(0)} = d_i$, $V^{(0)} = 0$, $U^{(0)} = 1$, $LSB^{(0)} = 1$. After two computation steps, the controller changes the Mux selection to close the ring (Fig. 5). Then the inserted token turns round inside the ring until completion [11] [12].

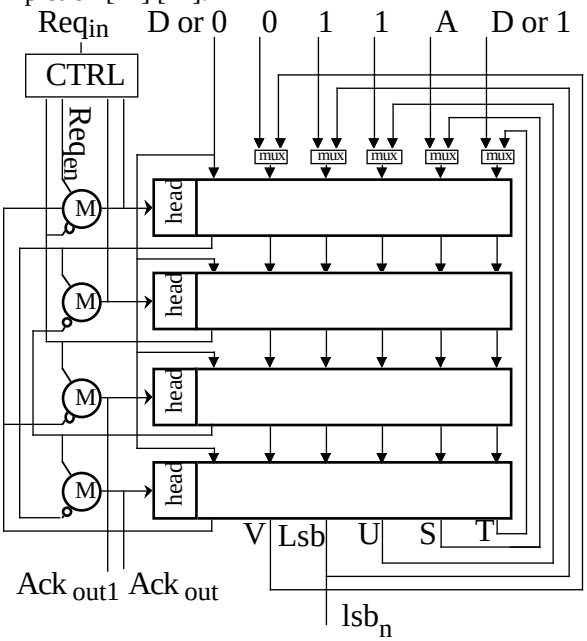

Figure 5 : Architecture of the four-stage ring divider.

4. Results and discussion

Two circuits, a synchronous 32-bit divider and a self timed 32-bit divider, have been designed and are currently under fabrication at the SGS-Thomson Crolles plant, using the three-metal 0.5 µm CMOS Technology (3.3 Volts).
The 32-bit synchronous divider has been designed in TIMA laboratory. First it has been implemented at the schematic level. The head cell and the tail cell have been first designed and simulated separately, then an 8-bit slice as described in Fig 1 has been simulated successfully. The necessary buffers have been added and an 8-bit combinatorial divider has been successfully assembled and simulated. After all this preliminary tests, full-custom layouts of the head and tail cells have been designed, the abutment technique was used to build the divider. The different structures described in table II have been layed out, extracted and simulated.
The 32-bit asynchronous divider has been designed in the CNET, a France-Telecom laboratory, they have chosen a different strategy for the layout design. A full-custom method has been used to design the basic cells (head and tail) as standard cells, fully compatible with the SGS-Thomson standard cells library [14]. This means that they will have a fixed height and be locally routed with polysilicon and metal1. A channel-less place&route is done with metal2 and metal3 to obtain the full divider. The power dissipation is computed at the maximum frequency. To achieve our designs, we have used the Cadence Framework and the electrical simulator Eldo. Table III shows the results obtained.

Number of stages of the divider	Division time (ns)	Number of transistors	Area (mm2)	Density (trs/mm2)	Power dissipation (mW/MHz)
1 Stage	288	5500	0,27	20500	1,5
2 Stages	248	7000	0,34		
3 Stages	235,5	8500	0,41		
4 Stages	228	10100	0,49		

Table II: Synchronous division

Number of stages of the divider	Division time (ns)	Number of transistors	Area (mm2)	Density (trs/mm2)	Power dissipation (mW/MHz)
3 Stages	130	12000	0,71	17000	3
4 Stages	100	16000	0,90		

Table III: Asynchronous division

The above results allow us to make the following analysis:
Area analysis: if we compare dividers with the same number of stages, the area of the asynchronous divider is twice that of the synchronous one. We explain it as follows :
-The DCVS logic used is a structure less regular than the standard CMOS one and so takes more area when layed out on silicon [15].
-The control and detection circuit of the asynchronous divider is certainly more important than the clock and Flip-Flop range circuits which represent at most 10 % of the total area.
-Compared to a full-custom design, the use of standard cells encreases the silicon area.
Power dissipation analysis: The DCVS logic used in the asynchronous circuit is responsible for the doubled power consumption as stated in [15], although the cells have been optimized [9]. In addition to this, the clock effect has not a

big influence in this circuit, because the synchronous divider is almost combinatorial and the only synchronous part is the Flip-Flop range controlled by a gated clock. The use of a gated clock resolves this problem without any additionnal control circuits.

Speed analysis: The asynchronous circuit has an average speed greater than the synchronous one because:

-The DCVS logic was used [15].

-The asynchronous circuit control part allows less critical operations, to be executed in a faster time and so the average time is smaller than the synchronous one which uses the critical path to set the clock frequency.

Finally a layout of the divider is presented.

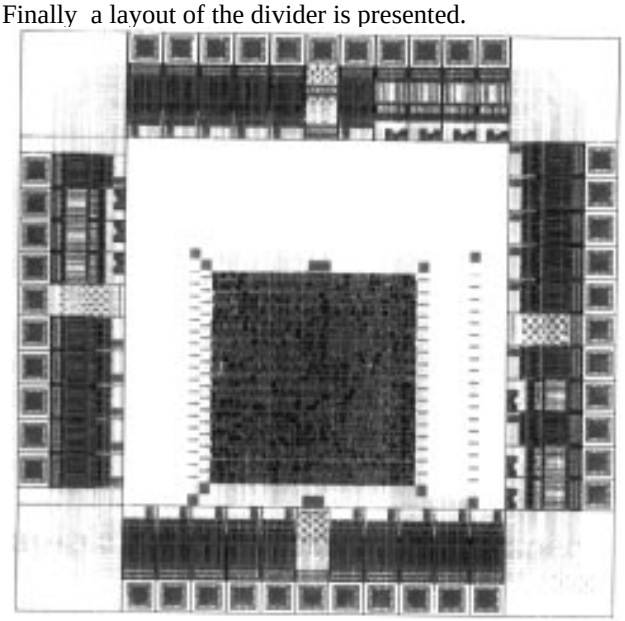

Figure 6: The Layout of the asynchronous divider

5. Conclusions

Asynchronous strategies are techniques for designing, firstly low power designs and secondly faster designs. The main conclusion to be drawn from this work is that asynchronous SRT dividers offer opportunities for realizing faster division but this speed advantage is often gained only at the expense of circuit area and power consumption.

This divider can be used in two different ways:

1-It belongs to a greater system and in this case the asynchronous divider will accelerate the global timing. A good example of such a system is a synchronous microprocessor. In this system the operation of division is the least used operation in the arithmetic unit and thus the speed enhancement is not significant. Also, as nowadays, the trend is towards battery powered portable products, the synchronous divider may be prefered in these systems. However, in an asynchronous microprocessor [16] [17], the use of an asynchronous arithmetic operator is necessary because of the absence of a clock generator.

2-It belongs to a dedicated system which will use intensively the division operation. An example of such a system is a real time cryptography circuit. In this case, the time must be known and we will plan the other functions by following the worst case delay, and here also we prefer the synchronous circuit.

We also notice that at the present time, for competitiveness reason, the trend is for low cost designs and hence the synchronous divider is more suited since it requires less area than the asynchronous one.

Finally, we believe that the SRT divider is a particular case, because it is almost combinational, and for complex synchronous designs, the known limitations will appear and it will be difficult to use some features used in the SRT synchronous divider as the gated clocks, and asynchronous designs will certainly reach some of their goals as demonstrated in the AMULET1 project [16] [17].

References

[1] S. Hauck, " *Asynchronous design methodologies: An overview* ", Proceedings of the IEEE, Vol. 83, No. 1, January 1995.

[2] D. E. Atkins, " *Higher radix division using estimates of the divisor and partial remainders* ", IEEE Transactions on computers, Vol. C-17, No. 10, 1968.

[3] S.E. McQuillan, J.V. McCanny "*New algorithms and VLSI architectures for SRT division and square root* ", proc.11th IEEE symp. on Computer Arithmetic, Windsor, Canada, 1993, pp80-86.

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

[5] P. Soderquist, M. Leeser " *Area and Performance Tradeoffs in Floating-Point Division and Square Root Implementations* ", Technical Report EE-CEG-94-5, School of Electrical Engineering, Cornell University, New York, Decembre 1994.

[6] J.E. Robertson "*The correspondence between methods of digital division and multiplier recoding procedures* ", IEEE Trans. on Comp., vol. C-19 Nb. 8, August 1970.

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

[8] J. William, V.C. Hamacher "*A linear-time divider array* ", Canadian Elec. Eng. Journ. Vol. 6, #4, 1981.

[9] B. El Hassan, A. Guyot, M. Renaudin, "New self timed ring and their application to division and square root", ESSCIRC'95, Lille, France, pp. 226-229, Sept. 1995.

[10] T. E. Williams "*Performances of iterative computation in self-timed rings* ", Journ. of VLSI signal processing, Vol. 7, pp 17-31, February 1994.

[11] T. E. Williams, M. Horowitz ," *A zero overhead self-timed 160 ns 54 bits CMOS divider* " IEEE Journ. of solid state circuit, vol. 26, pp 1651-61, Nov. 1991

[12] K. Choi, KJ. Lee, J.W. Kang, "*A self-timed divider using RSD number system* ", proc. of the ICCD, pp. 504-507, Boston, 1994.

[13] L. Lavagno, A. Sangiovanni-Vincentelli " *Algorithms for synthesis and testing of asynchronous circuits* ", Kluwer Academic Publishers, 1993.

[14] B. El Hassan, "Asynchronous VLSI architectures", PhD thesis, Institut National Polytechnique de Grenoble, Sept. 1995.

[15] K. M. Chu, D. L. Pulfrey " *A comparison of CMOS circuit techniques: Differential cascode voltage switch logic versus conventional logic* ", IEEE Journal of Solid-State Circuits, Vol. sc-22, No. 4, August 1987.

[16] J. D. Garside, " *A CMOS implementation of an asynchronous ALU* ", Proceedings of the IFIP Working Conference on Asynchronous Design Methodologies, Manchester, UK, 1993.

[17] S. Furber, " *Computing without clocks: Micropipelining the ARM processor* ", Internal Report, Departement of computer science, University of Manchester, 1995.