Asynchronous Circuit Design on Reconfigurable Devices

R.U.R.Mocho[1], G.H.Sartori[1], R.P.Ribas[1], A.I.Reis[2]

1 - PPGC, PGMICRO - UFRGS
Caixa Postal 15064
Porto Alegre, Brazil
rpribas@inf.ufrgs.br

2 - NANGATE
Smedeholm 10, 2 tv
DK-2730 Herlev
are@nangate.com

ABSTRACT

This paper presents the design of asynchronous circuits on synchronous FPGAs and CPLDs. Different design styles have been investigated through the implementation of dual-rail full adders and ripple carry adders, as well as self-timed ring based applications. The comparison analysis has been carried out by prototyping the circuits on standard programmable logic devices, and using the development tools provided by vendors. Although the feasibility of asynchronous circuits has been demonstrated in such devices, the experimental results clearly show the inefficiency of such a kind of digital system implementation. This is mainly due to the architecture characteristics of the programmable devices and the logic synthesis realized by the development environments. Remarks and suggestions are derived from this study for a new FPGA architecture devoted to asynchronous design.

Categories and Subject Descriptors
B.6.1 [**Logic Design**]: Design Styles – *combinational logic, logic arrays, sequential circuits.*

General Terms
Performance, Design, Reliability, Experimentation, Theory.

Keywords
Asynchronous circuits, FPGAs.

1. INTRODUCTION

The advent of FPGAs and CPLDs circuits provided effective platforms for fast prototyping of VLSI digital synchronous integrated circuits [1]. This technology has focused on synchronous designs, and more recently on globally asynchronous locally synchronous – GALS systems. Asynchronous design, on the other hand, lacks of a well established FPGA/CPLD like alternative [2].

The programmable logic solutions specific for asynchronous circuits are based on large granularity blocks that do not have the same flexibility and degree of configurability provided by LUT based FPGAs and AND-OR array based CPLDs. This way, it is hard to start the design flow from well established hardware description languages (HDLs), like VHDL and Verilog. Most proposed architectures for asynchronous FPGAs are closely associated to a given design style. For instance, MONTAGE [4] is based on arbiters and synchronizer cells. The approach in [5] is based on Null Convention Logic (NCL [6]). In [7], a dataflow based architecture for asynchronous circuit is proposed. The main drawbacks are: the designer should start from a dataflow specification and the granularity of the logic blocks is designed to make them compatible with dataflow constructs. The approach in [8] is based on micropipeline implementations, while the work in [9] presents test results for a highly pipelined asynchronous FPGA. A flexible FPGA that can be targeted to several different design styles is proposed in [10]. However, the logic block presented there is somewhat expensive as it requires a matrix of 11x14 connection points internally to the logic block, as well as two LUT-7 structures.

Some approaches prefer to implement asynchronous designs on top of synchronous FPGAs. For instance, the approach in [11] presents a comparison among implementations including an asynchronous circuit design from schematics on top of an Actel FPGA device [12]. An asynchronous co-processor partially implemented on a FPGA, partially on an ASIC is presented in [13], where the project adopts a dataflow architecture and it is described structurally. The design of a self timed ALU on a FPGA platform is discussed in [14], but again the circuit is described in schematic level, not using hardware description languages. The work in [15] proposes the design of asynchronous circuits using regular FPGAs, and its main contribution is an informal proof that the C-element is hazard-free if implemented in a single LUT. However, only 2- and 3-input C-elements are mentioned and hardware description languages are not applied.

In this work, synchronous FPGAs/CPLDs available in the market are targeted to implement asynchronous or self-timed circuits from VHDL specifications. Different design styles, considering dual-rail encoding for computation completion detection, have been investigated. Note that differential and dynamic CMOS structures, very popular in asynchronous ASIC design, are not suitable in programmable components [2]. Moreover, bundle data implementations, like micropipelines, are also not compatible with such devices.

This paper is organized as follows. Section 2 discusses the functionality and VHDL description of basic cells for asynchronous design. Four dual-rail design styles are discussed in Section 3 through adder circuits. Section 4 presents the experimental results and analysis. Finally, the remarks for a dedicated asynchronous FPGA and conclusions are given in Section 5.

2. CELLS FOR SELF-TIMED DESIGN

Self-timed designs comprise specific circuits generally not used in synchronous design. The ones used in this work are the C-element or Muller cell, the M-out-of-N cell and the unique dual-rail latch. These three components are briefly described bellow [2].

2.1 C-Element

The C-element is widely used in asynchronous designs for implementing the handshake control circuit and for computing completion detection. The truth table for a C-element is given by Table 1. Notice that the output signal follows the input ones when these are equal, or the cell acts as a memory of the previous value. A dedicated configurable block for this logic function is not available in synchronous FPGA and CPLD. The way used to implement C-elements is through their description as combinational circuits with the output reconnected to one of the inputs. A VHDL description of C-element implementation using this strategy is presented in Fig. 1. Another way to implement the C-element functionality is the behavioral description presented in Fig. 2. C-elements with more than 2-inputs can be described either considering both strategies mentioned above or by combining 2-input cells.

Table 1: Truth table of 2-input C-element

I1	I2	Out
0	0	0
0	1	keep previous value
1	0	keep previous value
1	1	1

```
ARCHITECTURE inst OF CC IS
SIGNAL ctemp: std_logic;
        COMPONENT modcc
        PORT (a, b, cin : IN std_logic;
                 cout   : OUT std_logic);
        END component;
BEGIN
INSTCC: modcc PORT MAP (a, b, ctemp, ctemp);
        c <= ctemp;
END inst;

ARCHITECTURE equation OF modcc IS
BEGIN
cout <= (a or b) and (a or cin) and (b or cin);
END equation;
```

Figure 1 : VHDL C-element as instance of an equation

2.2 M-out-of-N cell

The behavior of an M-out-of-N cell is described through an example. The behavior of a 2-out-of-3 cell is shown in the VHDL code presented in Fig. 3. The output goes to high logic level if 2 of the 3 inputs present the logic value '1'. The output is reset when all inputs are set to logic '0'. Otherwise, the output is memorized. Notice that, when M is equal to N, a C-element is obtained, while making M equal to 1 result in a combinational OR cell.

2.3 Dual Rail Latch

The dual-rail latches used in self-timed rings and asynchronous pipelines, considering the 4-phase protocol [2], have the behavior described in Fig. 5. The output is reset when the enable signal 'En' is equal to '0'. When the latch is enabled, it either acts as a memory when no valid data is available in the inputs (It = If = 0) or the output follows the input values when complemented data (It ≠ If) are present in the input. The implementation of such a kind of latch applied in this work is shown in Fig. 4.

```
ARCHITECTURE behavioral OF ncl_3_2 IS
SIGNAL s_temp: std_logic;
SIGNAL aux: std_logic_vector (2 downto 0);
BEGIN
aux <= i1 & i2 & i3;
s_temp <=   '0' when aux = "000" else
    '1' when aux = "110" else
    '1' when aux = "101" else
    '1' when aux = "011" else
    '1' when aux = "111" else
s_temp;
s <= s_temp;
END behavioral;
```

Figure 3 - VHDL behavioral description of 2-out-of-3 cell.Figure 1.

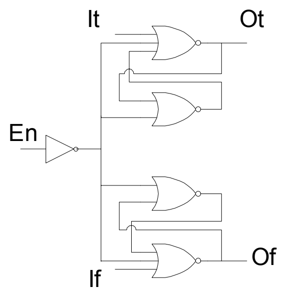

Figure 4 – Dual rail latch for self timed circuits

```
ARCHITECTURE behavior OF latch_async IS
    SIGNAL Ot_temp, Of_temp: std_logic;
BEGIN
PROCESS (It, If, En, Ot_temp, Of_temp)
BEGIN
    IF (En='0') THEN
    Ot_temp <= '0'; Of_temp <= '0';
    ELSIF (It='0') and (If='0') THEN
                Ot_temp <= Ot_temp;
                Of_temp <= Of_temp;
        ELSE
                Ot_temp <= Ot;
                Of_temp <= Of;
        END IF;
        Ot <= Ot_temp; Of <= Of_temp;
END PROCESS;
END behavior;
```

Figure 5 – Behavioral description of dual rail latch

3. DUAL RAIL DESIGN STYLES

Four different design styles for the combinatorial blocks have been investigated, taking into account the completion detection through dual-rail signaling (Dt,Df), that means: (1,0) = ‘1’; (0,1) = ‘0’; (0,0) = no valid data or waiting state; (1,1) = not used .

The logic styles described bellow were used to implement full adders, which have then cascaded to build ripple carry adders (RCA), ranging from 4 to 32 bits. The RCA circuits were then applied in the self-timed ring based applications such as least common multiple, greatest common divider, square root, counter, integer division and remainder.

3.1 DIMS

Delay Insensitive Minterm Synthesis - DIMS is strongly based on canonical (minterm based) sum-of-products, where each minterm is recognized through a C-element [2]. The FPGA and CPLD implementations have been done through the instantiation of the C-elements, described previously. Fig. 6 shows an exclusive-NOR cell based on DIMS technique.

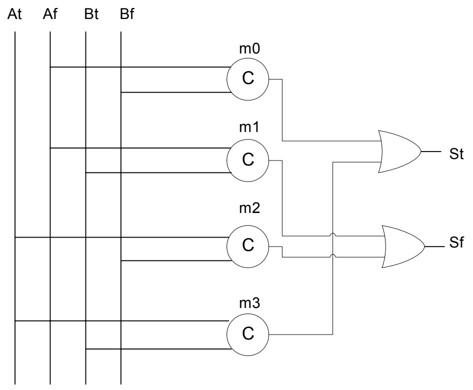

Figure 6 – Exclusive-NOR based on DIMS

3.2 NCL

Null Convention Logic – NCL, in turn is a design style derived from threshold logic [5] [6]. The adopted implementation has been done through instantiation of the M-out-of-N elements described previously. The NCL based full-adder is depicted in Fig. 7. It contains two 2-out-of-3 cells and two 3-out-of-5 cells. NCL logic can be synthesized with similar methods to threshold logic [6].

3.3 Derivation from Combinational Circuits

In this implementation, the circuit is derived from a standard single-rail combinational circuit. All the equations are made positive unate by using the dual-rail encoding with the adequate polarity to avoid inverted literals. The intermediate signals that are needed in both polarities will result in duplicated logic gates. Fig. 8 illustrates this principle. Notice that additional circuitry is needed to ensure that: a) the output is reset only when all the inputs are null; and b) a valid value is produced in the output only when all the inputs present a valid data.

3.4 Behavioral Description with Strong Indication

In this approach, the design of dual-rail combinational blocks is done through behavioral description including the desired characteristics. For instance, strong indication for valid data and for input reset was described in a behavioral way, but not in the cell level. The circuit is specified from a behavioral point-of-view and the strong indication conditions are added behaviorally. These conditions include: a) reset the output only when all the inputs are reset; and b) output receives a valid value only when all the inputs are defined, that means, when they present a valid data.

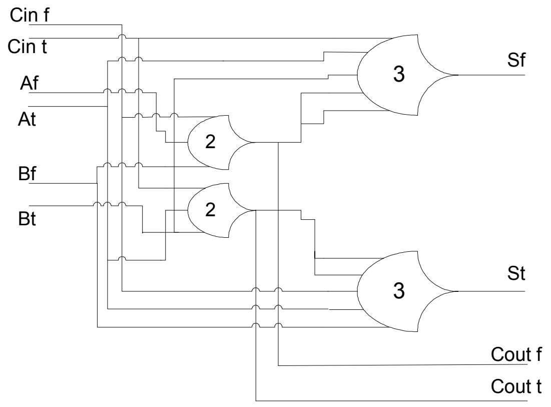

Figure 7 – Full-adder based on NCL

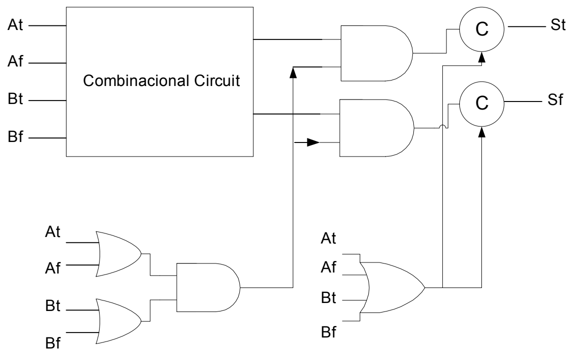

Figure 8 – Derivation from combinational logic

4. EXPERIMENTAL RESULTS

The experiments have been carried out always considering VHDL circuit descriptions. Six programmable devices available commercially were targeted, being FPGAs and CPLDs from the major vendors:

- PLD#1 - Altera SRAM-based FPGA FLEX10KE
- PLD#2 - Altera Flash-based CPLD MAX7000AE
- PLD#3 - Xilinx SRAM-based FPGA SPARTAN2
- PLD#4 - Xilinx Flash-based CPLD XC9500XV
- PLD#5 - Actel antifuse-based FPGA AXELERATOR
- PLD#6 - Actel SRAM-based FPGA 500K family

Each implementation was made through the particular development tool provided by the vendor. The statistics about the number of configurable cells (macrocells, logic cells or LUTs) that represent circuit complexity were also extracted from the vendor's tool.

Initially, the implementation of C-elements with different number of inputs was investigated. As mentioned before, this cell is applied in DIMS technique and generally applied to the handshake circuit building and completion detection. Moreover, it is also interesting due to the storage characteristic observed in the cell logic. It would be expected that the C-element implementation up to a certain number of inputs could be made with only one configurable cell. This expectation comes from the fact that a logic cell has features to implement combinational (for instance a LUT) and memory (for instance FFs) internally to the logic block. The experimental results are shown in Table 2. We noticed that the FFs were never used as there is no explicit clock signal in the description. Memorization characteristics are implemented through combinational elements with feedback. The support for C element in a single cell is achieved for C elements three inputs. This was expected as it corresponds to a LUT with three external inputs and an internal feedback. The PLD#4 was able to support a 6 input C element in a single logic element.

Similar exercise was realized with the dual-rail latch to verify the mapping result provided by the tool. This is one of the main drawbacks in implementing asynchronous circuits on top of synchronous programmable devices, where the dual-rail latches are more expensive that standard flip-flops. The results are given in Table 3. Again, only the combinational part was used.

In the next step, the dual-rail full adder approaches were prototyped. The configuration results are given in Table 4. It is clear the inefficiency of design strategy when compared to conventional single-rail full adder.

The least common multiple, build in a thee-stage self-timed ring, is based on the RCAs generating from the full adders evaluated in Table 4. The experimental results of this application are shown in Table 5. The same experiments were done for other circuits (square root, remainder, greatest common divider, integer division and counter) but providing similar results, probably due to the similarities among them.

In general, NCL logic gave the best results for FPGA. This is due to the possible optimizations derived from threshold logic that resulted in the optimized full-adder in Fig. 7. Also the use of low granularity M-out-of-N cell fits well with the FPGA structure. Actel developing platform was able to process well the behavioral description, obtaining the best implementation for this family. DIMS is a style that is not very competitive due to the use of too many C-elements. As shown in Fig. 6, for instance, a 2-input EXOR gate would require four C-elements one for each minterm. The design styles that are not based on the instantiation of fine grain cells (C-elements or M-out-of-N cells), like derivation from combinational logic (Section 3.3) and behavioral description (Section 3.4) present an improvement for CPLD based architectures. This happens because of the tuning between the size of the description and the size of the available logic cells in the architecture.

When compared to normal synchronous versions the asynchronous circuits implemented were 4 to 5 times larger and around 2 to 4 times slower. This was expected as the developing tools (logic synthesis and mapping) and the device architectures have been conceived to implement synchronous systems. Even if this circuit presents memory characteristics, it is usually mapped into the combinational part of the device architecture. Different software tools from different FPGA/CPLD vendors implemented it as a logic element with a feedback from the output to an input of a LUT or another logic element.

5. DIRECTIONS FOR ASYNCHRONOUS FPGAS

The waste of area when using an FPGA can be illustrated by the following simple example. Consider the logic equation for a full adder.

$$sum = \overline{a}\cdot\overline{b}\cdot c + \overline{a}\cdot b\cdot\overline{c} + a\cdot\overline{b}\cdot\overline{c} + a\cdot b\cdot c \quad (1)$$

This equation can fit into a single LUT, as it has only three variables in its support. If it is to be implemented in dual rail, the following two equations are needed for signals sumT and SumF.

$$sumT = aF\cdot bF\cdot cT + aF\cdot bT\cdot cF + aT\cdot bF\cdot cF + aT\cdot bT\cdot cT \quad (2)$$

$$sumF = aT\cdot bT\cdot cF + aT\cdot bF\cdot cT + aF\cdot bT\cdot cT + aF\cdot bF\cdot cF \quad (3)$$

In addition to that, these new equations have six variables and do not fit in a single LUT. Indeed the implementation of these equations requires six LUTs. Besides that, the available flip-flops in every cell are not used in the circuits because we do not create VHDL processes using dependency on the raising edge of a clock signal (as the circuits are not synchronous). Another extra overhead is the routing of the extra dual-rail signals.

5.1 Configurable Cells

One of the most important area overhead for asynchronous is the duplication of logic for dual rail implementation. The goal here is to make equations 2 and 3 fit in a single dual-rail LUT. This could be achieved as dual rail implementation should focus only on the implementation of positive unate functions. This way, the following assumptions should be made for a LUT devoted to dual rail implementations:

- it has four dual rail inputs;
- it has two dual rail outputs, in a form of a shared programmable selection tree;
- both outputs are reset if all the dual rail inputs are reset;
- one of the outputs is set when all the dual rail inputs present valid data;
- if valid data is not present in the inputs, the output does not need to produce a valid data;
- self timing should be guaranteed through the use of extra C-elements, when needed.

Table 2 – C-element implementations using behavioral (B) and structural (S) VHDL descriptions

	PLD# 1 (B)	PLD# 1 (S)	PLD# 2 (B).	PLD# 2 (S)	PLD# 3 (B)	PLD# 3 (S)	PLD# 4 (B)	PLD# 4 (S)	PLD# 5 (B)	PLD# 5 (S)	PLD# 6 (B)	PLD# 6 (S)
2	2	1	2	1	1	1	1	2	2	5	7/2	9/2
3	2	1	2	1	2	1	1	2	6	8	10/4	12/4
2X2	4	2	4	2	2	2	2	3	4	10	14/4	18/4
4	2	3	2	1	2	2	1	2	9	12	11/4	14/5
2x3	4	2	4	2	3	2	2	3	8	13	17/6	21/6
2x2x2	6	3	6	3	3	3	3	4	6	14	19/6	25/6
5	3	3	2	2	2	3	1	2	15	15	14/6	17/7
2x4	4	4	4	2	3	3	2	3	11	17	18/6	23/7
2x2x3	6	3	6	3	4	3	3	4	10	18	22/8	28/8
2_3x2	6	3	6	3	4	3	3	4	10	18	22/8	28/8
6	3	4	2	2	3	5	1	2	18	18	15/6	19/8
2x5	5	4	4	3	3	4	2	3	17	20	21/8	16/9
2x2x4	6	5	6	3	4	4	3	4	13	21	23/8	30/9
3x2x3	8	4	8	4	5	4	4	5	12	23	27/10	35/10
3_2x3	6	3	6	3	5	3	3	4	14	21	25/10	31/10
4_2x2	6	5	6	3	4	4	3	4	13	21	23/8	30/9

Table 3 – Dual-rail latch implementations using behavioral (B) and structural (S) VHDL descriptions.

	PLD# 1 (B)	PLD# 1 (S)	PLD# 2 (B).	PLD# 2 (S)	PLD# 3 (B)	PLD# 3 (S)	PLD# 4 (B)	PLD# 4 (S)	PLD# 5 (B)	PLD# 5 (S)	PLD#6 (B)	PLD#6 (S)
1 bit	3	2	2	2	1	4	2	2	10	2	15/10	9/2
4 bits	12	8	8	8	4	16	8	8	42	8	60/41	35/8
8 bits	24	16	16	16	8	32	16	16	85	17	117/82	68/17
16 bits	48	32	34	32	16	64	32	32	170	36	233/166	133/34
32 bits	96	64	64	64	32	128	64	64	341	70	462/331	265/70

Table 4 – Full-adder implementations

	PLD#1	PLD#2	PLD#3	PLD#4	PLD#5	PLD#6
DIMS	12	12	12	12	82	78/34
NCL	10	8	8	8	64	50/34
Derived	17	8	13	8	50	48/20
Behavioral	17	10	13	4	42	39/27

Table 5 – 32-bits least common multiple circuit implementations

	PLD# 1	PLD# 2	PLD# 3	PLD# 4	PLD# 5	PLD#6
DIMS	2156	na	2831	na	3485	9165/4848
NCL	1952	na	2587	na	3093	5967/4422
Derived	2318	na	2915	na	3069	5608/3099
Behavioral	2466	na	3214	na	2840	5059/3814

The approach in [10] presents a flexible FPGA that can be targeted to several different design styles. However, the logic block presented there is somewhat expensive as it requires a matrix of 11x14 connection points internally to the logic block, as well as two LUT-7 structures. This happens because they do not apply the unate simplifications arising from dual rail logic. By using unate simplifications, as suggested here, only two LUT-4 are needed and the configurations flip flops can be shared. Besides, the mapping is straightforward (one-to-one) from a given mapping to regular 4-input LUTs.

5.2 Storage Cells

The flip-flops available in the logic cell were never used. A possibility could be to substitute them by C-elements or asynchronous latches. The best option would be a combination of the two (for instance, 50% of the logic elements would contain C-elements and 50% would contain asynchronous latches instead of FFs). C-elements would be useful to implement distributed control as well as to ensure strong indication of reset and end-of-calculus (see Fig. 8). Asynchronous latches would be the storage elements in pipelines.

5.3 Dual Rail Routing

If a dual-rail LUT is used, the routing of the FPGA should use the concept of dual rail signals. This way, only the wires would be duplicated. All the configuration control does not need to be duplicated as it is assumed that both dual-rail wires are delivered to the same places.

6. CONCLUSION

In this paper, the design of asynchronous circuits on top of synchronous FPGA/CPLD platforms has been evaluated. As expected, the results were not competitive as the device architectures and the developing tools have been conceived for the purpose of prototyping synchronous circuits. However, some good lessons can be taken from these experiments. First, it has been demonstrated that it is possible to fool synthesis tools designed for synchronous circuits to produce working asynchronous design from VHDL code. This can be a low price platform for asynchronous circuit prototyping. Second, guidelines for improving the FPGA architectures, in order to efficiently target asynchronous design, have been derived based on the analysis of the prototyping results.

7. REFERENCES

[1] T.J.Todman, G.A.Constantinides, S.J.E.Wilton, O.Mencer, W.Luk and P.Y.K.Cheung, “Reconfigurable computing: architectures and design methods”, IEE Proc.- Comput. Digit. Tech., Vol. 152, No. 2, March 2005, pp. 193-207.

[2] J.Sparso, S.Furber. Principles of Asynchronous Circuit Design - A system perspective. Kluwer, 2001.

[3] S.Hauck, S.Burns, G.Borriello and C.Ebeling, “An FPGA for implementing asynchronous circuits”, IEEE Design and Test of computers, Fall 1994, pp. 60-69.

[4] R.Payne, “Asynchronous FPGA architectures”, IEE Proc.- Comput. Digit. Tech., Vol. 143, No 5, September 1996, pp. 282-286.

[5] K.Meekins, D.Ferguson and M.Basta, “Delay Insensitive NCL Reconfigurable Logic”, Aerospace Conference Proceedings 2002, pp.4-1961 to 4-1966.

[6] K.M. Fant and S. A. Brandt, “NULL Convention Logic: a complete and consistent logic for asynchronous digital circuit synthesis”, ASAP96, pp.261-273.

[7] J. Teifel and R.Manohar, “An Asynchronous Dataflow FPGA Architecture,” IEEE Transactions on Computers, vol. 53, no. 11, pp. 1376–1392, Nov. 2004.

[8] Y.Zafar and M.Ahmed, “A Novel FPGA Compliant Micropipeline”, IEEE Trans. on CAS II – Express Briefs, Vol 52, No 9, September 2005, pp. 611-615.

[9] D.Fang, J.Teifel and R.Manohar, “A High-Performance Asynchronous FPGA: test results”, FCCM05, pp. 271-272.

[10] N.Huot, H.Dubreuil, L.Fesquet and M.Renaudin, “FPGA architecture for multiple-style asynchronous logic”, DATE 2005, pp. 32-33.

[11] E.Brunvand, N.Michell and K.Smith, “A Comparison of Self-Timed Design using FPGA, CMOS and GaAs Technologies”, ICCD1992, pp. 76-80.

[12] Actel web page. www.actel.com.

[13] J.H.Novak and E.Brunvand, “Using FPGAs to Prototype a Self Timed Floating Point Co-Processor”, CICC 1994, pp. 85-88.

[14] S.Ortega-Cisneros, J.J.Raygoza-Panduro, M.J.Suardíaz and E.Boemo, “Rapid prototyping of a self-timed ALU with FPGAs”, ReConFig 2005, pp. 7-14.

[15] Q.T.Ho, J-B.Rigaud, L.Fesquet, M.Renaudin and R.Rolland, “Implementing Asynchronous Circuits on LUT Based FPGAs”, FPL2002, LNCS2438, pp. 36-46, 2002.

[16] M.L. Dertouzos, “Threshold Logic: a Synthesis Approach”, The MIT Press, 256pp, 1965.

[17] Altera web page. www.altera.com.

[18] Xilinx web page. www.xilinx.com