GALS Networks on Chip: A New Solution for Asynchronous Delay-Insensitive Links

G. Campobello[1], M. Castano[1], C. Ciofi[1] and D. Mangano[2]
[1] *Dipartimento di Fisica della Materia e Tecnologie Fisiche Avanzate, University of Messina, Italy.*
[2] *Dipartimento di Ingegneria dell'Informazione, University of Pisa, Via Caruso, I-5122 Pisa, Italy.*
e-mail: daniele.mangano@iet.unipi.it

Abstract

In this paper a cost effective solution for asynchronous delay-insensitive on-chip communication is proposed. Our solution is based on the Berger coding scheme and allows to obtain a very low wire overhead. For instance, the results of our evaluation show that a 64-bit link can be built paying a wire overhead of 10% and 30 equivalent two-input gates per wire. As a general rule, when the number of bits to be transmitted increases, the wire overhead decreases and the gate overhead remains almost the same.

1. Introduction

With the System-on-Chip era, in order to provide enough on-chip bandwidth, the shared-bus approach was replaced by more complex hierarchical bus structures (i.e. AMBA, STBus, etc.). However, current interconnection systems will soon be inadequate in terms of scalability, flexibility, performances and energy efficiency. Recently, a new method for building a communication infrastructure interconnecting the IP-cores in a packet-switched fashion has been proposed [1]. This communication infrastructure is generally known as Network-on-Chip. As it is shown in Fig.1, the main components of the NoCs are: the routers (or switches), the Network Interfaces (NI) and the links (or channels). However, in order to provide the future NoC systems with adequate scalability degree, a method for decoupling the clocks of the IP modules is needed. In fact, the synchronization with a single clock source is another important issue which limits the SoCs development ([2], [3]). A fully asynchronous approach may be used to overcome the clock synchronization problem, but because of the investment in the clocked tools and techniques, synchronous design will continue to be attractive in industry [4]. Therefore, a solution based on GALS paradigm is expected to be the a really attractive approach to overcome the problem [1]. Such argumentations lead to elect the GALS-NoC paradigm as target for developing the future complex SoC-based systems. In the last years, many efforts have been done to develop efficient NoC, and several NoC architectures, emulation frameworks and synthesis flows, have been proposed (e.g., [5], [6], [7]). Coppola at al. proposed a framework for design exploration which enables fast modeling and simulation through a layering approach [8]. A synthesis flow, known as NetChip, targeting fully synchronous NoC architectures, has been recently proposed [9]. Xpipes is a NoC architecture obtained by means of NetChip. This NoC architecture, based on the pipelined links, uses latency insensitive operation to overcome the interconnect-delay problem. Solutions based on GALS-NoC paradigm could allow to overcome several design problems by avoiding the necessity of the timing constraints on the clock signal. In fact, by partitioning the system into different decoupled synchronous modules, clock distribution among the synchronous IP-cores is not needed and locally clock generation is possible. At present, it is known that in order to achieve asynchronous communication, expensive coding and decoding circuits are needed. Furthermore, an overhead in terms of wires has to be paid.

In this work we propose an asynchronous delay-insensitive link implementation supporting the GALS-NoC paradigm. Such a link is based on the Berger coding scheme [10]. Our proposal enables to build links of any width with low wire and logic overheads and removes the constraints on wire propagation delays. In this way, the back-end SoC design could be simplified and a higher productivity and a lower time-to-market could be achievable. Such a solution is based on the idea to exploit the temporal order of the events for accomplishing the completion detection. The main advantages of our approach are the completely self-timed property, the high degree of modularity, and the possibility of an implementation that uses only standard and highly optimized blocks.

3-9810801-0-6/DATE06 © 2006 EDAA

The reminder of this paper is organized as follows. Section 2 presents the overview of our proposal and tha way in which asynchronous links can be used in actual SoCs. Section 3 introduces the theoretical basis of the DI coding and provide details on our implementation. Section 4 reports a few interesting simulation results and a complexity evaluation. In section 5, a cost comparison with previous solutions is presented. Finally, in section 6, some conclusions are reported.

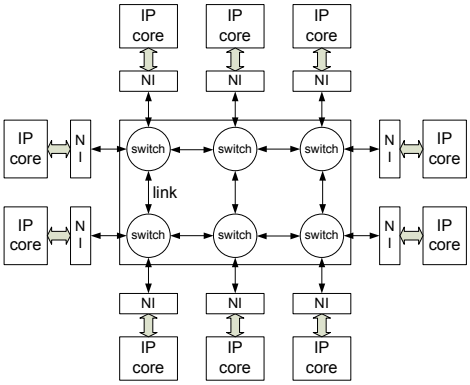

Figure 1: Simple Network-on-Chip block diagram

2. Overview

In GALS systems, synchronous units belonging to different clock domains communicate to each other by means of asynchronous channels. Data synchronization is the main issue in GALS circuits, and several solutions have been recently proposed [11], [12]. There are two main approaches for building GALS Network-on-Chips: one where the links are the only asynchronous elements, and one where the whole communication infrastructure is asynchronous and the IP-cores are the only fully synchronous units. The asynchronous links presented in this paper can be used in both organizations. From the point of view of the NoC designer, an asynchronous link can be considered as the provider for a transport service across the chip. The top-level view of the proposed link is reported in Fig.2.

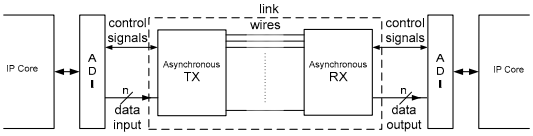

Fig. 2: links top-level view.

In general, two asynchronous design methodologies, and therefore two asynchronous communication schemes, are possible. These are *Bounded-Delay* (BD) and *Delay-Insensitive* (DI). In the former, information on wires and gates delays is needed, whereas in the latter is not. Since the proposed asynchronous link implements a DI communication, our solution requires no constraints on the wires delays, and it guarantees the correctness of the communications in any case.

We have chosen to provide the links with a four-phase handshake signaling protocol.

In a GALS-NoC system, data synchronization is needed both at the border of the network and at the switch input ports. In order to integrate the asynchronous links in a GALS-NoC system, suitable interfacing modules have to manage the communication with both the link ports and the synchronous units. We will refer to such an interface as *Asynchronous Domain Interface* (ADI). Therefore, an ADI module manages:

- The communication with either the synchronous IP-core or the synchronous router module by means of a suitable protocol.
- The communication with the link by means of a four-phase handshake protocol.
- Data synchronization.

Recently, an ADI implementation for STBus protocol has been proposed [13]. In this case, the ADI module enables full-duplex communication with generic STBus synchronous modules. The four-phase handshake protocol which is used with the view to interfacing the link with the external, works as follows:

- When no transaction is in progress, all output signals of the TX are at 0 logic level.
- When the Transmitter IP Core sends a request and the TX wants to start a transaction, it accordingly drives the outputs.
- When the RX side recognizes the condition of "data valid", it samples the data and rises an Ack signal to the TX.
- When the TX receives an Ack signal, it de-asserts all its outputs.
- When the RX detects all 0s at its inputs, it de-asserts an Ack signal.
- Finally, when the TX recognizes that an Ack signal goes back to 0, the system comes back to the initial condition and a new transaction can start.

3. DI codes and proposed links

DI codes have been used in many applications for error detection and delay-insensitive communication. Their main feature is the ability of allowing the correct interpretation of the code word independently of the delay of individual bits. Several delay-insensitive coding schemes have been proposed, but, effective CMOS implementations are needed in order to make feasible asynchronous DI on-chip communication [14], [2]. In fact, delay-insensitive coding schemes imply an overhead in terms of area and wire number to be paid. This is the main

drawback of the DI codes, and many efforts in the research area of the asynchronous circuits aim to devise feasible solutions for delay-insensitive communication.
Verhoeff [15] proposed a technique for modeling DI codes. Such a method take into considerations three main factors: efficiency, membership test and encoding/decoding complexity. Efficiency (WH) measures the wire overhead with respect to the optimum binary encoding. Such a parameter is defined as follows:

$$WH = \frac{\log_2 N}{n}$$

where N is the number of values represented and n is the number of wires which are needed.
The membership test is related to the completion detection function and the encoding/decoding complexity keeps into account for the logic overhead due to the encoder and decoder circuits. Also costs of implementations based on the Delay-Insensitive Minterm Synthesis (DIMS) have been analyzed [14].
Dual rail and *m-of-n* are well known coding schemes which can be used for on-chip delay-insensitive communication. In [16], *Molina et al.* proposed an implementation of a delay-insensitive chip area interconnect using dual rail encoding. *Bainbridge et al.* proposed a *m-of-n* scheme for point-to-point on-chip links [14].
In dual rail coding scheme, n wires are needed for transmitting a $n/2$ bit data word. This coding scheme is the simplest possible and it requires a minimum amount of additional logic. When dual rail scheme is applied to large and complex interconnection systems, the cost of doubling the width of the data-path (both in terms of chip area and, as a consequence, of propagation delays) may be unacceptable. Therefore, different coding schemes are needed, which may allow a reduction in the number of additional wires, even at the cost of a significant complication in the coding scheme. The *m-of-n* scheme is a coding scheme where exactly m lines out of the n available for transmission are at logic level one. In a sense, dual rail coding is nothing but a particular case of *m-of-n* coding when one recognizes that a 1 of 2 coding is applied to each bit. Therefore, one may think of the dual rail coding as a 1 of 2 coding which is applied to 1-width "groups" of lines of the original data to be transmitted. The concept of grouping, which may appear to be forced in the case of dual rail coding, will have an important impact on the actual possibility of realizing *m-of-n* coders for wider link in our approach.

3.1. Link using Berger codes

The Berger code is a systematic code which is purposely designed for error detection in data transmission. It is composed of two parts: the information bits (*D,* data bits) and check bits (*CR*), i.e. the binary representation for the count of the information bits which are to 0. Obviously, the width of R part is that needed in order to provide a correct binary representation of the number D:

$$R = \lceil \log_2 (D+1) \rceil$$

A detailed discussion of the properties of the Berger code can be found in [10]. The wires overhead for a system based on the Berger code is:

$$WH = \frac{R}{R+D}$$

Therefore, when D increases, WH decreases and one gets a greater advantage in using this coding scheme.
Berger code has been already proposed as a means for obtaining DI codes [17]. Our work, however, proposes a specific architectural implementation for asynchronous, completely self-timing, on-chip communication system based on the Berger code.
Fig.3 shows the top-level block diagram for the new proposed architecture.
Berger code consists of the data bits (information bits) and the check bits. Before starting with the description of the new architecture, it is important to stress the fact that this asynchronous link has been designed for employing the conventional 4-phase handshake protocol. In the block diagram shown in Fig.3, no pipeline stage is included. Clearly, optional pipeline stages can be used in order to increase the throughput.

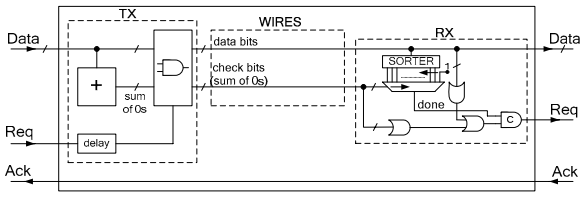

Fig. 3: Top-level architecture for the link based on the Berger coding scheme.

The convenience in using a pipeline mechanism is strongly dependent on several parameters such as the transmitter clock frequency, the receiver clock frequency, the wire delays, the communication protocol between the transmitter and the receiver, etc. Although this is a quite complex scenario, it is possible to state that the uniform distribution of a few pipeline stages along the link between the transmitter and the receiver does improve the throughput by a factor approximately equal to the number of pipeline stages. The situation is quite different when the pipeline stages are located very close to one another at the receiving end of the link, as it could be easily obtained with our design. In such a case, the question arises

whether it is useful or not to have more than one pipeline stage just in front of the receiver. Simulations have shown that the improvement that can be obtained is not worth the complication and additional area. Both the completion detector and the barrier of c-elements which are needed for implementing a pipeline stage, makes the implementation of two pipeline stages too much expensive with respect to the possible benefits that are obtained.

3.2 Detailed description

With reference to Fig.3, let us number the inputs of the multiplexer starting from 0 to D (D is the number of wires carrying data bits) from left to right. Let us also assume that the multiplexer input which is selected is the one that, according to the previous numbering, corresponds to the value CR which is encoded in natural binary on the *R* check bits.

In order to better understand the proposed asynchronous RX block, let us first analyze how the circuit in Fig.4 works. The goal of such a circuit is to detect when all the data and check bits have been received. Specifically, the circuit sets the signal *done* only when all the bits have been received.

As stated above, detecting when all the signals are stable is the main problem to solve in DI communications because delays are not known and can be different for each different wire.

The idea is that the outputs of the sorter and the check bits drive the multiplexer according to different directions. For the sake of clarity, let us take into consideration the following example. Let us suppose that at the initial state all the bits (both check and data bits) are zero and that the string "11101010" represents the data bits to be sent. The output of the sorter will be initially composed of all zeros ("00000000") and changes to "00000001" after the first data bit is propagated.

The number of ones at the output of the sorter will grow, from the rights to the left, until reaching "00011111" when all the ones have been propagated. Therefore, the 4-th output of the sorter will be one only when all the data bits will be received.

Let us now consider the check bits. At the beginning, the check bits are all zero and therefore the multiplexer input which is selected is the leftmost one on the left and the *done* signal is 0. The signal *done* does not change even when a few information bits have been already received.

For the above example, the corresponding check bits will be "0011". If only one of the 1 bits is received, the second or the third output of the sorter will be selected and the done signal will remain to zero. Finally, when all the 1 check bits are received, the 4-th output of the sorter will be selected and the done signal will be set to one thus indicating that all the signals have been received.

On the basis of these observations, the operation of the entire link can be described in some detail. The operation of the TX section is quite simple:

- The adder has to count the sum of 0s, which is needed to build the Berger code. Obviously, the sum of 0s is obtained by inverting the logic state of the input bits.
- When the *Req* signal at the input port is at the low logic state, all the bits of the Berger code are low because of the presence of the AND barrier.
- The delay on the *Req* signal guarantees that all the bits of the Berger code are forwarded only after that they are stable at the input of the AND barrier (this because the adder could not be hazard free).

The RX must implement a few mechanisms:

1. Recognizing the situation when all the information bits are stable, in order to generate one request to the output port (completion detection).
2. Recognizing the situation when all the information and check bits are de-asserted, thus allowing to de-assert the request at the output port.

For implementing the mechanism 1, the use of the two standard circuits named sorting network (or sorter) and multiplexing network (or multiplexer) are proposed. The sorter enables to detect how many 1s are present among input bits, and several implementations have been proposed in the literature [18]. With reference to Fig.4, the operation of the completion detector for the Berger code can be summarized as follows:

- As soon as the information bits start going from the logic state 0 to the logic state 1, the same number of output bits of the sorter goes from the logic state 0 to the logic state 1 in the order that goes from the right to the left.
- For each check bit going from logic state 0 to logic state 1, an input of the multiplexer at the right of the previous one will be selected and the logic state of this input will be the one present on the done signal.
- In any case, when either information bits or check bits are not stable, the selected input of the multiplexer is at the logic state 0.
- The *done* signal will go to the logic state 1 only after that all the 1s contained in the information bits and in the check bits become stable.

For mechanism 2 implementation, two multi-inputs OR gates have been used, one for recognizing the situation when all the information bits are at logic state 0, and one for recognizing when all the check bits are at the logic state 1. The outputs of these gates are connected to the input of another OR gate; this port generates a signal (namely return in Fig.3) which rises when at least one

information or check bit is high and falls when all the information bits and check bits are low.
If the *done* signal goes high, then the *Req* signal goes high; on the other hand, if the *done* signal goes low, the *Req* signal cannot return to logic state 0 before the return signal goes low. For implementing this behaviour, we have connected the *return* and *done* signals to the inputs of the c-element shown in Fig.3. C-element (or Muller cell) is a key component for implementing asynchronous systems [2].

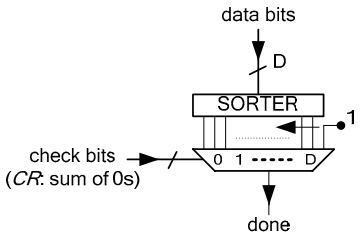

Fig. 4: Completion detector for the link based on the Berger coding scheme.

4. Simulation results and complexity

A VHDL model for the proposed link has been developed, in order to perform a functional verification. A test-bench for the design space exploration and the testing of our link has been designed too. Such a test-bench is built on top of the *modelsim* simulator. It includes a transmitter, a receiver and a software module. Such a software module allows to set the simulation parameters and it also enables the acquisition and the evaluation of the results, the visualization and the optional storing of the simulation results when the simulation is running or when it stops.
Some tests have been performed by means of our test-bench. In Fig.5 performance of a link with no pipeline, with a single pipeline stage and with two pipeline stages which are very close to one another are reported. The throughput shown in figure is normalized with respect to the max value. Pipeline stages have been assumed to be near to the receiver, the transmitter frequency was 10MHz, and wire delays spanning from 180 to 310 ns have been used.
An evaluation of the complexity in terms of equivalent two-input logic gates has been performed. Tab. 1 shows the results of our evaluation. In the table the number of bits to be transmitted (n), the cost of the adder (SUM), the cost of the sorter (SORT), the total cost (C), the number of added wires (AW), the wire overhead (WH) and the ratio between the total cost, the number of bits to be transmitted (C/n) and the latency introduced by the asynchronous transmitter and receiver (T) are reported. For the above evaluation we considered that a *n*-bit sorter can be realized as a *Batcher*'s sorting network [18]. Therefore, if the number of inputs is *n*, about $\frac{1}{2}n\log_2^2(n)$ gates are required, placed on $\frac{1}{2}\log_2(n)(\log_2(n)+1)$ levels. For the adder we considered a parallel counter solution. A conventional CMOS realization of a parallel counter involves a number of full adders arranged in a tree.

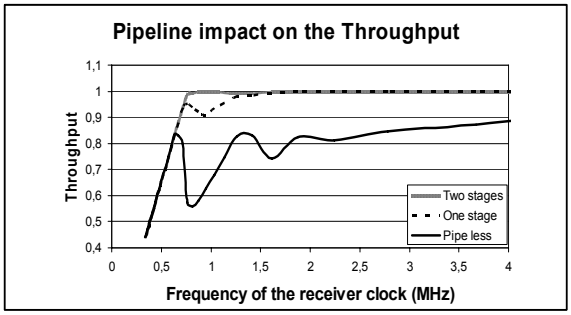

Fig. 5. Throughput vs. receiver clock frequency.

Swartzlander reports that the number of FA's for an *n*-input counter to be in the order of $n\text{-}\log_2(n)$ [19]. We assumed that a full-adder can be implemented with 5 gates and therefore that the adder block needs about 5n gates.
The wire overhead is that due to the sum bits and to the acknowledgment signal, that is $\log_2(n)+1$.
By looking at the table it is possible to conclude that when the number of bits increases, the wire overhead decreases, while the *cost per bit* (C/n) also increases. These results can be exploited for choosing the suitable link width and, possibly, by using a proper strategy for grouping a few links in order to obtain a wider one.

n	SUM	MUX	SORT	C	AW	WH	C/n	T
32	160	192	400	752	6	0.16	23,5	35
64	320	448	1152	1920	7	0.10	30	45
128	640	1024	3136	4800	8	0.06	37,5	56
256	1280	2304	8192	11776	9	0.03	46	68

Tab.1. Results of the cost evaluation.

As far as the latency that is introduced by the asynchronous transmitter and receiver is concerned, it can be estimated considering the sum of the delays due to the adder, sorter and multiplexer (see Fig. 3). If we consider an unitary delay for a gate, a two gates delay for a full adder and a tree based architecture for the multiplexer (i.e. $2\log_2(n)$ gates on its critical paths), the latency can be estimated as:

$$T = \frac{1}{2}\log_2(n)\cdot[\log_2(n)+9]$$

5. Comparisons

In this section we compare the cost of the proposed asynchronous interface with those of other solutions presented in previous works. In [14] the authors report the costs for several asynchronous delay insensitive solutions

expressed as the number of transistors-per-bit for 32-bit bus. Despite the cost model, which is based on the Delay Insensitive Minterms Synthesis, is a bit different with respect to the one used in this paper, we can compare our solution by considering 6 transistors for every gate (that is the same number of transistors which is considered in the above cited paper for two inputs gates). Even if not all the gates used in this paper can be considered as basic two inputs gates, this will simplify the evaluation and will give us results that are on the same order of magnitude of actual values. Tab.2 reports the number of transistors and the wire overhead for the proposed solutions and those presented in [14] in the case of 32-bit bus.

Code	WH	Cost
Dual-rail	0.5	380
1-of-4	0.5	1340
2-of-4	0.35	3120
3-of-6	0.28	9500
2-of-7	0.37	10070
Piestrak (Berger)	0.14	11700
Proposed Solution (Berger)	0.16	4510

Tab.2. Costs and wire overheads for different DI codes.

As it is possible to observe from Tab.2, the proposed solution has a cost which is comparable with that of some *m*-of-*n* solutions but with a wire overhead which is about one half. Furthermore, in comparison with other Berger's based solutions, it has comparable wire overhead with a cost that is less than one half.

6. Conclusions

In this paper a new solution for asynchronous delay-insensitive on-chip communication has been proposed. NoC paradigm appears to be the future architecture of the VLSI systems. At present, a GALS approach seems to be the only one able to support the challenge of building high-scalable systems. However, effective CMOS implementations of asynchronous delay-insensitive links are needed. We proposed a technique for building asynchronous links based on the Berger coding scheme. The results of our evaluation show that a 64-bit link can be built paying a wire overhead by 10% and 30 equivalent two-input gates per wire. As general rule, we discovered that when the number of bits to be transmitted increases, the wire overhead decreases and the gate overhead increases. We also developed a VHDL model of our link and a test-bench for the design space exploration and the testing has been implemented.

References

[1] L. Benini and G. De Micheli, “Networks on chips: A new SoC paradigm”, Computer, 35(1): 70-78, Jan. 2002.

[2] A. Davis and S.M. Nowick, “An Introduction to Asynchronous Circuit Design", Technical Report UUCS-97-013, Computer Science Department, University of Utah, Sep. 1997.

[3] J. Sparso, “Future networks-on-chip; will they be Synchronous or Asynchronous?” SSoCC’04, 13-14 Apr. 2004.

[4] S. Moore, G. Taylor, R. Mullins, P. Robinson, "Point to Point GALS Interconnect", Proc. Of ASYNC'02, Manchester, UK, Apr. 2002.

[5] M. Coppola, R. Locatelli, G. Maruccia, L. Pieralisi, A. Scandurra, “Spidergon: a novel on-chip communication network”, Proc. IEEE International Symposium on System-on-Chip 2004. 16-18, p. 15, Nov. 2004.

[6] P. Guerrier and A. Grenier, “A Generic Architecture for On-Chip Packet-Switched Interconnections,” Proc. IEEE Design Automation and Test in Europe (DATE 2000), IEEE Press, Piscataway, N.J., 2000, pp. 250-256.

[7] A. Adriahantenaina, H. Charlery, A. Greiner, L. Mortiez, and C. A. Zeferino, “SPIN: A scalable, packet switched, on-chip micro-network,” in Proc. Design Automation Test Eur., 2003.

[8] M. Coppola, S. Curaba, M.D. Grammatikakis, G. Maruccia, F. Papariello, “OCCN: A NoC Modeling Framework for Design Exploration”, Journal on System Architecture, vol. 50, pp. 129- 163, February 2004.

[9] L. Benini, G. De Micheli, D. Bertozzi, A. Jalabert, S. Murali, R. Tamhankar, S. Stergiou, “NoC Synthesis Flow for Customized Domain Specific Multiprocessor Systems-on-Chip”, IEEE Transaction on parallel and distributed systems, Vol. 16, No. 2, Feb. 2005.

[10] J.M. Berger, “A note on error detection codes for asymmetric binary channels”, Inform. Contr., vol. 4, pp. 68-73, Mar. 1961.

[11] R. Dobkin, R. Ginosar, C. Sotiriou, “Data Synchronization Issues in GALS SOCs”, Proceedings, Tenth. International Symposium on Asynchronous Circuits and Systems (ASYNC’04), April 2004, pp. 170 - 179.

[12] T. Chelcea, S. M. Nowick, “Robust Interfaces for Mixed-Timing Systems”, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Volume: 12, Issue: 8, Aug. 2004, pp. 857-873.

[13] A. Scandurra, S. Pisasale, D. Mangano, “STBus Asynchronous Decoupler: an answer to the IP integration issues in future technologies", IP-SOC 2004, Dec. 2004.

[14] W.J. Bainbridge, W.B. Toms, D.A. Edwards, S.B. Furber, “Delay-Insensitive, Point-to-Point Interconnect using m-of-n Codes”, Proc. IEEE Ninth International Symposium on Asynchronous Circuits and Systems (ASYNC’03).

[15] T. Verhoeff, “Delay-insensitive codes: an overview”, Distributed Computing, 3(1):1-8, 1988.

[16] P.A. Molina, Peter Y.K. Cheung, “A Quasi Delay-Insensitive Bus Proposal for Asynchronous Systems”, 3rd International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC '97).

[17] S.J. Piestrak, “Membership test logic for Delay-Insensitive codes”, Proc Async ’98, San Diego, California, April 1998, pp194-204.

[18] K.E. Batcher, “Sorting Networks and their Applications”, Spring Joint Computer Conference 1968.

[19] E.E Swartzlander, “Parallel Counters”, IEEE Trans. Computers 1973.