International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:8, No:10, 2014

A High Level Implementation of a High Performance Data Transfer Interface for NoC

Mansi Jhamb, R. K. Sharma, A. K. Gupta

***Abstract*—**The distribution of a single global clock across a chip has become the major design bottleneck for high performance VLSI systems owing to the power dissipation, process variability and multi-cycle cross-chip signaling. A Network-on-Chip (NoC) architecture partitioned into several synchronous blocks has become a promising approach for attaining fine-grain power management at the system level. In a NoC architecture the communication between the blocks is handled asynchronously. To interface these blocks on a chip operating at different frequencies, an asynchronous FIFO interface is inevitable. However, these asynchronous FIFOs are not required if adjacent blocks belong to the same clock domain. In this paper, we have designed and analyzed a 16-bit asynchronous micropipelined FIFO of depth four, with the awareness of place and route on an FPGA device. We have used a commercially available Spartan 3 device and designed a high speed implementation of the asynchronous 4-phase micropipeline. The asynchronous FIFO implemented on the FPGA device shows 76 Mb/s throughput and a handshake cycle of 109 ns for write and 101.3 ns for read at the simulation under the worst case operating conditions (voltage = 0.95V) on a working chip at the room temperature.

***Keywords*—**Asynchronous, FIFO, FPGA, GALS, Network-on-Chip (NoC), VHDL.

I. Introduction

THE asynchronous circuit design techniques are a promising design alternative for resolving the design issues such as circuit reliability problems caused by thermal variation and voltage occurring in CMOS design technology [1]. Generally, ASIC or full custom design techniques are widely used for implementing asynchronous circuits. Nowadays, the FPGA device is gaining popularity by being a reconfigurable device. These reconfigurable devices have a bright scope in the future IC market. Not much work is done in the direction of implementing asynchronous circuits on FPGA devices. For exploiting the benefits of asynchronous circuits such as low power consumption, average case performance, low electromagnetic interference and delay insensitivity, it is important to implement them on these reconfigurable devices. Implementing asynchronous circuits on commercial FPGA is a rarity owing to the hardness of timing control for the signal propagation delays. To implement the asynchronous circuits on an FPGA device, there are two directions of research: (a) The design of new FPGA architecture for easy adaptation to asynchronous circuits, [2]-[4] (b) Implementing the asynchronous circuits on presently available FPGA devices [5]. Recently, in [6], an asynchronous pipelined FIFO is extended to an asynchronous MIPS processor using 65 nm Xilinx Virtex-5 devices for implementation.

Asynchronous FIFO is a fundamental building block for system-on-chip designs that employ Globally-Asynchronous-Locally-Synchronous (GALS) [7], [8] paradigm. The work presented in [9] shows the GALS on-chip network can be constructed using a network node structure. In the network node, the data packets are globally transferred using the asynchronous handshaking protocol. Thus in order to buffer the output or input data packets of the network node, an efficient asynchronous FIFO is required. Till date, the asynchronous FIFO designs are classified into the two categories with respect to the movement of data. The first class comprises of flow-through FIFOs [10], [11] based on the micropipeline structure proposed in [10]. In this class before hitting the output port, the data has to propagate across all the data cells of the FIFO. This contributes to a high throughput and poor latency. To avoid the movement of data within the FIFOs, the second class of asynchronous FIFOs use token passing, counter control logic [12] and common bus structure [13], [14]. In this case the increased complexity of control logic is the price paid for eliminating the latency incurred due to the propagation of data in a flow through FIFO. The control pipelines based on micropipeline structure are proposed in [15]. These pipelines are used as control logic for the asynchronous FIFO. An asynchronous arbiter and C-element RTL structures are also used in proposed asynchronous FIFO [15]. An asynchronous implementation of the ARM microprocessor is designed in [16]. The work presented in [16] assesses the effects of different design styles on the micropipeline latch structures.

This work implements an asynchronous FIFO belonging to the class of flow-through FIFO. Though the FIFO architecture implemented in this work is meant for network-on-chip (NoC) applications, it may be employed as a general purpose asynchronous FIFO. The FIFO design presented in this work has an additional advantage of being constructed at Register Transfer Level using commercially adopted VHSIC-HDL. This leads to an easy integration with different synchronous modules for construction of a commercial asynchronous-synchronous mixed design for e.g., Globally Asynchronous Locally Synchronous NoC. In this paper, the asynchronous FIFO is designed with the following key features:

Mansi Jhamb is with the University School Of Information and Communication Technology, GGSIPU, Sector-16C, Dwarka, New Delhi, India (e-mail: mansi.jhamb@gmail.com)

Prof. R.K. Sharma and Prof. A.K. Gupta are with the Department of Electronics and Communication Engineering, NIT Kurukshetra, India (e-mail: mail2drrks@gmail.com, anilg699@rediffmail.com)

- The preceding and the succeeding modules on either side of the FIFO can put and get the data using their own respective clocks across simple synchronous interfaces, leading to an easy NoC integration.
- The design can be synthesized employing standard cell based flow.
- Full/Empty conditions are detected and handled in a very simple manner.

The rest of the paper is organized as follows. In Section II, the detailed design of asynchronous FIFO is presented. Section III describes the implementation of the asynchronous FIFO design. Section IV deals with the performance evaluation of eight-stage, 16-bit wide asynchronous pipelined FIFO. Finally, the conclusions are drawn in Section V.

II. The Design of Asynchronous FIFO

A. The Design of Asynchronous Interface

The asynchronous interface facilitates the data movement from one module to another module using the concept of handshaking as shown in Fig. 1.

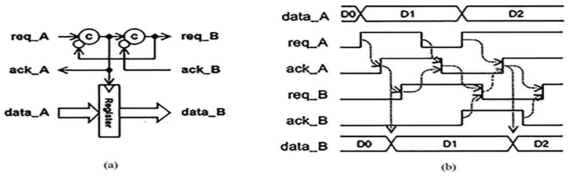

Fig. 1 Muller pipeline using C- Elements (a) Structure (b) 4- Phase Bundled Data Handshaking Protocol

The process of Handshaking is explained as following steps:

- Initially all four handshake signals are low.
- The block A sends out data and in the meantime sets req_A high.
- Now, ack_A will go high since req_B is low. The positive edge of ack_A acts as the clock signal for data register. Thus, data_A is latched to the stage register.
- The ack_A signal acts as the input to the next C-element. When ack_A is made high, it pulls req_B high.
- When req_A becomes low, ack_A will also become low.
- When the ack_B signal from next stage is asserted high, the second C-element conveys that next stage has read the output data of the register. This causes the req_B to go low.
- Finally, corresponding to req_B going low, the block B pulls the ack_B signal low.

The implemented asynchronous FIFO design is based on four-phase handshaking protocol. The design can support dual-rail protocol and bundled-data (on changing the interface descriptions). The block diagram of the implemented asynchronous FIFO based on 4 phase bundled-data handshaking protocol is shown in Fig. 2. The block diagram is functionally segregated into data processing section and control logic section. The control section comprises of control pipelines based on the concept of micro pipeline. In the control pipeline, every stage controls read/write operations of data section. The phenomenon of pushing the data into the asynchronous FIFO is as following:

- Once the data to be sent (Data_in (15:0)) is ready, the sender asserts the request signal (Rin)
- The FIFO will set the acknowledge signal (Ain) after the successful reception of the data
- Subsequently, the sender resets the Rin signal in response to Ain signal
- Once the Rin signal is reset, the FIFO resets the Ain signal.

The phenomenon of popping the data is same as the process of push despite the fact that the FIFO supplies the data and receiver receives it.

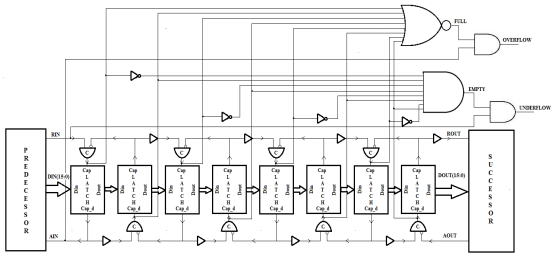

Fig. 2 A 4 Phase micropipelined FIFO structure

B. FIFO Characteristics

The top structural architecture of the implemented asynchronous micropipelined FIFO is shown in Fig. 3.

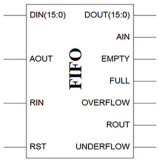

Fig. 3 Micropipelined FIFO: interfacing signals

The parameters characterizing the implemented FIFO are summarized in Table I.

TABLE I
FIFO Characteristics

Parameter name	Description
RST	Resets the FIFO
RIN	Input request to the FIFO from the sender which inputs data to the FIFO
AIN	Input acknowledgement sent by the FIFO to the sender after receiving data from it
DIN (15:0)	16 bit data input to the FIFO
DOUT (15:0)	16 bit data output from the FIFO
ROUT	Output request from the FIFO to the receiver which receives data from the FIFO
AOUT	Output Acknowledge sent by the receiver to the FIFO after it has received data from it
FULL	Sets when FIFO is full
EMPTY	Sets when FIFO is empty
OVERFLOW	Sets when there is an input request while the FIFO is full
UNDERFLOW	Sets when there is an output acknowledge while the FIFO is empty

C. FIFO Architecture

Fig. 4 shows the architecture of asynchronous pipelined FIFO implemented in this work with the bold and regular lines symbolizing the data bus and control signals respectively. The architecture is segregated into individual blocks, each performing a dedicated task, coordinated by the C elements. These blocks send signals to the handshake controller indicating their state i.e. request, acknowledge. The architecture is based on 4-phase micropipeline circuit. The latch capture signal 'Capture' of a single-phase transparent latch is directly driven by the C-elements. The completion is detected by the signal 'Cap_done'. Initially, the latch stages exhibit transparency and control signals are asserted low. Assertion of signal Din leads to the generation of a request on pipeline input and signal Rin is asserted high. After initialization, the C gate of the first stage is primed and therefore the capture signal 'Capture' goes high. Once the latch completion occurs, 'Capture' is asserted high and sent to the C element of the subsequent stage and also sent back to the sender through 'Ain'. This indicates the completion of the latching of the data. Now, the data can be removed and a reset phase can be initiated by resetting 'Rin'. The subsequent stage will also latch similarly, forwarding the latch completion signal to consecutive stage C element [16]. This will correspond to Rin being reset, setting the transparency in the first latch and on completion, Ain is reset. This completes the 4-phase handshaking.

Though FIFO design is simple 4-phase micropipeline circuit based on Sutherland's strategy [10], this circuit has the following limitation; a latch can only be occupied if the subsequent adjacent latch is transparent. The micropipeline backlogs occur if the data is injected to the pipeline at higher rate than the rate of data removal. This implies that only every other pipeline stage can be occupied, resulting in halving the depth of pipeline [16]. The FIFO is full when alternate latches hold data. Therefore, when capture signals to the alternate latches and inversion of the capture signals of the remaining latches give '1' as the output when passed through a multiple AND gate, it means that the FIFO is full.

Since propagation delay of the latch is greater than that of the Muller C elements, a delay has to be introduced in the path of the request signals in order to ensure proper synchronization between data and request signals. This delay is implemented using buffers having propagation delay comparable to that of the latches. This is because Muller C elements have negligible delay.

In order to guarantee the control and to synchronize the execution order of individual modules, the implemented FIFO architecture is interpreted using a five-state Mealy Finite State Machine, as shown in Fig. 5. The functions of each of the five states i.e. empty, partially filled, full, overflow and underflow is described in Table II.

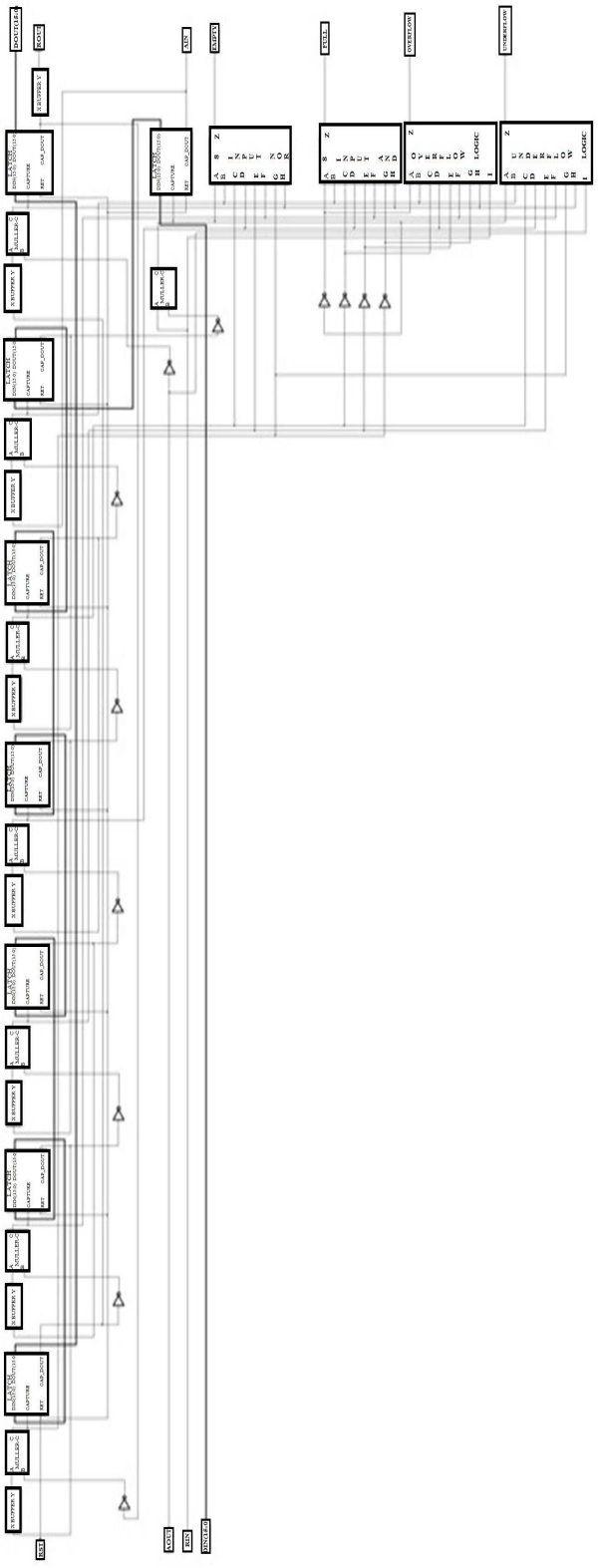

Fig. 4 Micropipelined FIFO architecture

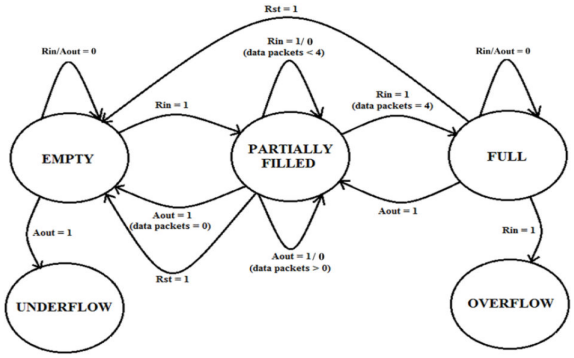

Fig. 5 FIFO state diagram

TABLE II
DIVERSE STATES OF IMPLEMENTED DESIGN

State	Task Performed
EMPTY	Initial state of the FIFO having no data
PARTIALLY FILLED	When number of writes is greater than number of reads, FIFO is partially filled with data
FULL	Since the design is of asynchronous FIFO with four words depth, it is considered full when filled with four data packets
UNDERFLOW	When there is a read request to an empty FIFO, underflow occurs
OVERFLOW	When there is a write request to a full FIFO, overflow occurs

III. IMPLEMENTATION OF THE DESIGN

In this paper, we design and analyze a simple asynchronous 16-bit micropipelined FIFO of depth four with the awareness of place and route (PAR) on FPGA device in order to show the feasibility of high-speed implementation of asynchronous circuits. We use Xilinx Spartan 3 devices to design and implement the FIFO with layout adjustments to fulfill the timing constraints which are mandatory for the correct operations of the circuits. Firstly, to verify the functionality of the asynchronous pipelined FIFO, the gate-level simulation of the eight-stage, 16-bit wide pipeline is as shown in Fig. 6. The Figs. 6 (a) and (b) show the status of the asynchronous pipelined FIFO indicating the empty/full conditions.

The top-down approach reported in this work is presented in Fig. 7. In this approach, the prime concern is the specification and design functionality. The process flow starts with the system level modeling of implemented architecture. The implemented model is analyzed and values of the test vector are utilized in Timing and RTL simulation. Very High Speed Integrated Circuits Hardware Description Language (VHDL) is employed for circuit design at RTL level. Special consideration is given in the design of the constituent modules of the FIFO using the concept of handshaking because we want to have a completely parameterized code for our design. Once the correct functionality is ensured through RTL simulation, the logic synthesis is performed. At this level, Xilinx generates a schematic which is independent of technology and optimizes the circuit to the FPGA specific library chosen (Spartan 3, XC3S200- 5FT256).

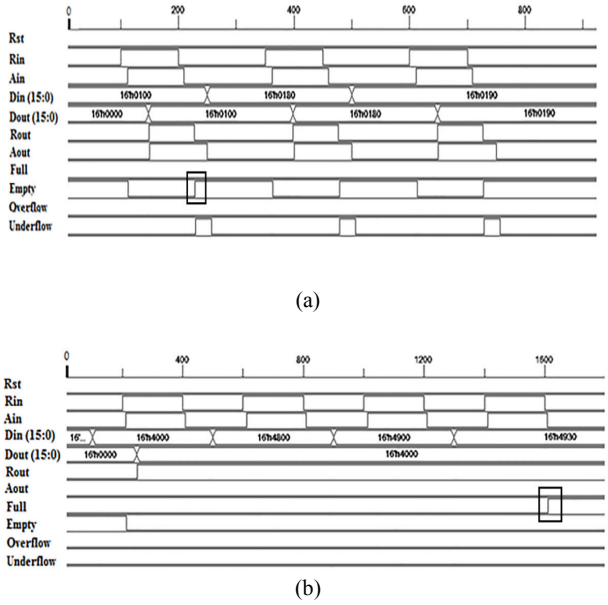

Fig. 6 Gate-level simulation results for a 8-stage asynchronous pipelined FIFO: (a) Depicting EMPTY condition (b) Depicting FULL condition

This is the stage at which specific design requirements; area and timing constraints are to be defined. Subsequently, the Xilinx ISE (Xilinx 9.1i) place and route (PAR) tool accepts input netlist file (.edf) generated by XST in the synthesis process. After that, the translation program translates the input netlist along with the design constraints onto a Xilinx database file.

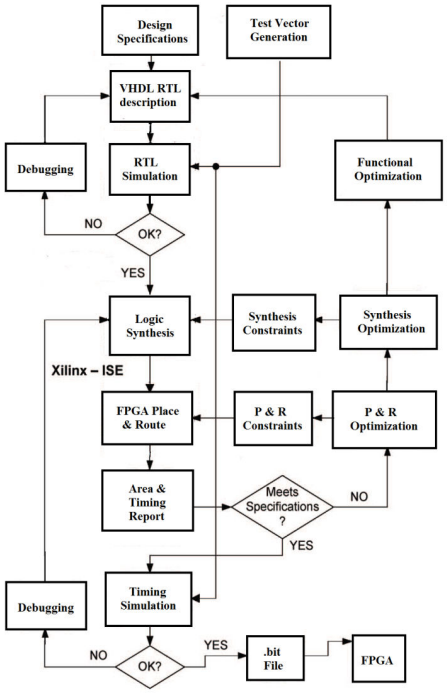

Fig. 7 The design flow

After the successful run of the translation program, the mapping of the logical design onto a Xilinx FPGA device is performed by the MAP program. Once the mapped design is accepted by the PAR program, the output for the bitstream generator (BitGen) is produced. PAR hence places and routes the FPGA. On the reception of the placed and routed design, a bitstream (.bit) is generated according the configuration of Xilinx device. Before the FPGA file is programmed, a timing simulation is carried out to ensure that the design meets the correct functionality and timing constraints.

IV. The Performance Evaluation

The asynchronous FIFO is implemented using VHDL. The event driven simulation tool Xilinx ISE 9.1i is employed for gate level simulation of the asynchronous FIFO presented in this work. The time elapsed between the leading edge of the 'Rin' signal and the lagging edge of 'Aout' signal contributes to the total latency of asynchronous FIFO. It can be calculated by the summation of handshake cycles of read and write requests. This latency is directly proportional to the FIFO size because the data propagates through the FIFO. The timing characterization for the designed asynchronous FIFO is illustrated in Table III.

TABLE III
Timing Characterization for Asynchronous FIFO

	Ack_Rise Latency* (ns)	Req_Hold Time# (ns)	Ack_Fall Latency+ (ns)	Handshake Cycle (ns)
Write Request	11.6	88.3	9.1	109.0
Read Request	1.5	77.0	22.8	101.3

* Time between rising edges of request and acknowledge signals [15]
\# Time between leading edge of acknowledge signal and lagging edge of request signal [15]
\+ Time between falling edges of request and acknowledge signals [15]

It is to be considered that the environment determines the parameter 'Req_Hold' (request hold time) not the FIFO. As per the latencies given in Table III, the implemented asynchronous FIFO is capable of executing 4.75 million 16-bit data read-after-write operations per second. In theory this equals to a throughput of 76 Mb/s.

Once the design is synthesized, the Xilinx ISE translates maps and routes the asynchronous FIFO to the FPGA device. The FPGA utilization produced by the ISE is shown in Table IV.

In an FPGA device, it is difficult to control timing delay among circuit/gate components. To satisfy the timing constraints, special design constraints need to be given to synthesis and PAR optimization processes. Xilinx synthesis and PAR tools support useful constraints like RLOC, LOC and P-block for controlling the layout design [17]. Such a user defined placement increases the speed of the circuits and makes the use of die resources more efficient. RLOC and LOC are the placement constraints specifying the relative and absolute positions of cells. Xilinx Plan Ahead supports P-block constraint and it also allows to constraint circuit modules to a particular area of FPGA device. In order to check the interconnect wire routing, an FPGA editor is used. We have extracted all the delays from our design using ISE Timing Analysis Tool and then only its worst cycle time is statically analyzed.

TABLE IV
FPGA Utilization for the Implemented Asynchronous FIFO

Logic Utilization	Used	Available	Utilization
No. of Slice Latches	112	3840	2%
No. of 4 input LUTs	13	3840	1%
Logic Distribution			
Number of occupied Slices	73	1920	3%
Number of Slices containing only related logic	73	73	100%
Number of Slices containing unrelated logic	0	73	0%
Total number of 4 input LUTs	13	3840	1%
Number of bonded IOBs	41	173	23%
IOB Latches	34		
Number of GCLKs	2	8	25%
Total equivalent gate count for design	820		
Additional JTAG gate count for IOBs	1968		

V. Conclusion

In the asynchronous paradigm, the design of an asynchronous micropipelined FIFO is of great importance since it demonstrates the timing overhead existing in system-on-chips. In this work, we have designed and analyzed a 16-bit asynchronous micropipelined FIFO of depth four, with the awareness of place and route on an FPGA device. We have used a commercially available Spartan 3 device and designed a high speed implementation of the asynchronous 4-phase micropipeline. The asynchronous FIFO implemented on the FPGA device shows 76 Mb/s throughput and a handshake cycle of 109 ns for write and 101.3 ns for read at the simulation under the worst case operating conditions (voltage = 0.95V) on a working chip at the room temperature.

References

[1] Semiconductor Industry Association, International Technology Roadmap for Semiconductor, 2009.
[2] Hauck S., Burns S., Borriello G., Ebeling C. An FPGA for implementing asynchronous circuits. *IEEE Design and Test of Computers* 11 1994; 3: 60-69.
[3] Royal A, Cheung PYK 2003, Globally asynchronous locally synchronous FPGA architectures, *13th International Conference on Field-Programmable Logic and Applications (FPL 2003), SPRINGER-VERLAG BERLIN*, Berlin, Pages:355-364, ISSN:0302-9743
[4] LaFrieda, C., Hill, B., Manohar, R.: An Asynchronous FPGA with Two-Phase Enable-ScaledRouting. In: *Proc. Of IEEE International Symposium on Asynchronous Circuits and Systems*; May 2010.
[5] Brunvand, E., Michell, M., Smith, K. A comparison of self-timed design using FPGA, CMOS,and GaAs technologies. In: *Proc. of International Conf. Computer Design*; October1992: pp. 76-80.
[6] Seung-JoonLee, Deok-Young Lee, Young-WoongKo, Jeong-Gun Lee. Asynchronous Circuit Design on an FPGA: MIPS Processor Case Study. *Communications in Computer and Information Science 2012; vol. 310; Springer.*
[7] Chelsea, T., and Nowick, S. (2004), 'Robust Interfaces for Mixed Timing Systems', *IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 12, 857-873.*
[8] Teehan, P., Greenstreet, M., and Lemieux, G. A Survey and Taxonomy of GALS Design Styles. *IEEE Design and Test of Computers 2007*; 24: pp. 418-428.

[9] X. Wang, D.Sigüenza-Tortosa, T. Ahonen, and J. Nurmi. Asynchronous Node Design for Network-On- Chip. *Proceedings of 2005 International Symposium on Signal Circuits and System; July 2005*.
[10] I.E. Sutherland. Micropipelines. *Communication of the ACM;* June 1989vol. 32; no. 6: pp 720-738.
[11] Brunvand, "Low latency self-timed flow-through FIFOs", *Proceedings of Sixteenth Conference on Advanced Research in VLSI*, pp.76 – 90, March 1995.
[12] A.V. Yakovlev, A.M. Koelmans, and L. Lavagno. High-Level Modeling and Design of Asynchronous Interface Logic. *IEEE Design and Test of Computers*; Spring 1995.
[13] T. Chelcea, and S.M. Nowick. Low-latency asynchronous FIFO's using token rings. *Proceedings of SixthInternational Symposium on Advanced Research in AsynchronousCircuits and Systems*; April 2000: pp. 210 – 220.
[14] K.K. Yi. The Design of a Self–Timed Low Power FIFO Using a Word–Slice Structure. M.Phil, University of Manchester, September 1998.
[15] Xin Wang, Jari Nurmi. A RTL Asynchronous FIFO Design Using Modified Micropipeline. *The10th Biennial Baltic Electronic Conference (BEC 2006), Estonia*; October 2006.
[16] P.day, J.V. Woods. Investigation into Micropipeline Latch Design Styles. *IEEE Transaction on VLSI Sytems* ,Vol. 3; 1995.
[17] Xilinx, ISE Design Software Manuals and Help. Sept, 2010