Comparison of Standard Cell based Non-linear Asynchronous Pipelines

Chammika MANNAKKARA† and Tomohiro YONEDA†

† National Institute of Informatics, Tokyo, Japan

Abstract Two types of non-linear asynchronous pipeline constructs, namely Conditional Branch and Asynchronous were compared for 2-phase, 4-phase and Early Acknowledgement handshake protocols. On the course, we have developed linear and non-linear controllers for Early Acknowledgement protocol and the Asynchronous Join controller for 2-phase protocol. All the different types of controllers were implemented in standard cells enabling their use in FPGA based designs. The overheads of performing the complex operations of Conditional Branch and Asynchronous Join, were estimated in comparison to the simple linear controller of the corresponding protocol. The results show that the 2-phase protocol performs better than the other two even in non-linear pipelines. The interesting result that we obtained was that the controllers for Early Acknowledgement protocol have shown the performance of 2-phase controllers when pipelines have processing elements in-between stages which allows the overhead of the controller to be hidden in the matched delay between stages.

Key words Asynchronous pipelines, Non-linear pipelines, 2-phase protocol, 4-phase protocol, Early Acknowledgement protocol

1. Introduction

In this paper we focused on two non-linear pipeline constructs: Conditional Branch and Asynchronous Join. We have compared the performance of the Conditional Branch and Asynchronous Join controllers for three different signaling protocols, namely, 2-phase signaling, 4-phase signaling and Early Acknowledgement Protocol. Altogether, six different types of standard cell based pipeline controllers were designed and compared the performance against the linear pipeline performance of corresponding protocol. In this paper, we have proposed a new standard cell based linear pipeline controller for Early Acknowledgement protocol. The Asynchronous Join and Conditional Branch controllers for Early Acknowledgement protocol are based on this linear controller. Moreover, we propose a new Asynchronous Join controller for 2-phase signaling based on the transition arbiter. Implementation details for all the controllers are given in Section 3.

The rest of the paper is organized into 4 sections as follows. Section 2 covers operation of the Conditional Branch and Asynchronous Join controllers. Implementation and operation of both linear and non-linear controllers which were compared in this paper is presented in section 3. Preliminary simulation results are presented in Section 4. Final section derives the conclusions.

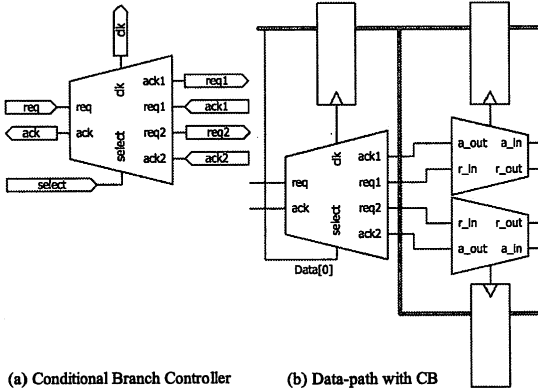
2. Non-linear Asynchronous Pipeline Constructs

In this paper, we considered two non-linear asynchronous pipeline constructs, Conditional Branch and Asynchronous Join. This section details the abstract operation of each of the construct without any particular reference to a signalling protocol.

2.1 Conditional Branch

In contrast to Fork construct, Conditional Branch divert the data to only *one* branch depending on *selection* signal to the controller. The interface of a two way Conditional Branch construct and its application in a data dependent two way conditional branching datapath is shown in Fig. 1.

Conditional Branch controller communicates with input stage with *req* and *ack* signals where as the two output stage control signals are *req1*, *ack1* and *req2*, *ack2* respectively. When a request *req* is made from the previous stage of pipeline, data is latched by clocking *clk* signal. Acknowledgement *ack* is sent to requesting stage when the data is latched. Depending on the *select* signal request is routed on either the first branch *req1* or the second branch *req2*. Controller is responsible for blocking further requests from input stage while waiting for an acknowledgement from next stage which could otherwise over-write the data being latched. When the corresponding branch path acknowledges the request using either *ack1* or *ack2* the cycle completes and Conditional Branch stage is ready to process the next request on *req*.



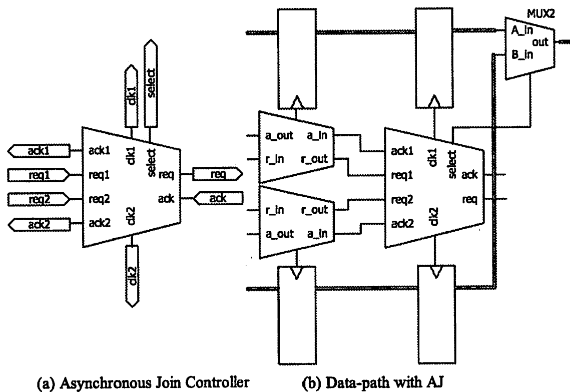
(a) Conditional Branch Controller (b) Data-path with CB

Fig. 1 Conditional Branch Controller

2.2 Asynchronous Join

The top level interface of the Asynchronous Join controller and its usage is shown in Fig. 2. The controller communicates with two input stages through *req1*, *ack1* and *req2*, *ack2* control signals where as output stage controls are *req* and *ack*. The Asynchronous Join module operates two input buffers (clocked by *clk1* and *clk2*) and a MUX (with *select* signal) to capture data from input side and appropriately select an input to the output according to the temporal order of requests. The operation of the controller is as follows.

A request from the input stage is asserted by *req1*(*req2*). The input data is then captured by the controller by clocking *clk1*(*clk2*) and the MUX *select* is appropriately set to *0*(*1*) to select the captured input buffer to the output side. Once the data is captured, an acknowledgement is sent to corresponding input side on *ack1*(*ack2*). Readiness of input data which has been captured, is asserted to the output side using *req*. The transaction completes when the output side acknowledges the completion with *ack*, and the controller is ready to process further request from either side. Similar to Conditional Branch controller, requests from any input side is blocked until the acknowledgement *ack* is received from the output side to avoid data over-write.



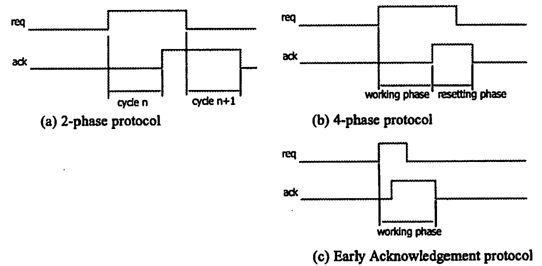
(a) Asynchronous Join Controller (b) Data-path with AJ

Fig. 2 Asynchronous Join Controller

A common situation is when both input sides make requests simultaneously. The controller then arbitrates between two requests and select one request to be processed while putting other on hold.

3. Implementation and Operation

The three different handshake protocols for which the above two types of non-linear controllers were built for the comparison are shown in Fig. 3. The most notable, Early Acknowledgement protocol [4] is an improvement over simple 4-phase protocol which hides the resetting phase of the signaling. In this protocol the acknowledgement is indicated by the falling edge of the acknowledgement signal where as in the 4-phase protocol it is typically indicated with a rising edge. As shown in Fig. 3(c), the acknowledgement signal goes high *as soon as* the request signal goes high there by allowing the request signal to be reset on an 'early acknowledgement' The actual acknowledgement which is indicated by the falling edge of acknowledgement signal demarks the end of the current transaction as well as reset the acknowledgement signal for the next iteration. This protocol eliminates the resetting phase inherent in the 4-phase protocol and yet retain the simplicity of the same.



(a) 2-phase protocol (b) 4-phase protocol (c) Early Acknowledgement protocol

Fig. 3 Handshake protocols

3.1 MOUSETRAP controller

For the transition signaling protocol, MOUSETRAP controller [1] is selected for its simplicity and robustness. As shown in Fig. 4 the controller consists of a D-latch and a XOR gate.

In the operation, initially all the control signals are low and the latch is transparent accepting the request as well as the data inputs. Once a request is made by a transition on *req_in*, it is latched along with the data producing a same transition for acknowledgement *ack_out*,request for the next stage *req_out* and made the latch (hence all the data latches of the stage) opaque to prevent over-writing by the new data from previous stage. Acknowledgement transition on *ack_in* from next stage in response to the *req_out*, releases the current stage and makes it transparent again to process the next request from the previous stage.

3.2 4-phase pipeline controller

We have used the 4-phase semi-decoupled controller proposed in [10] for this comparison. The controller and the waveforms for its operation are show in Fig. 5. The operation of the controller is

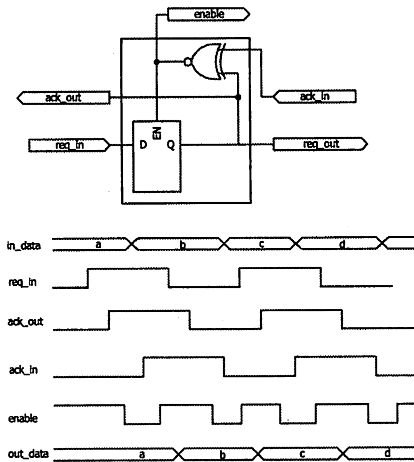


Fig. 4 MOUSETRAP Controller

as follows.

Initially all control signals are low. Request on *req_in* raise the *clk* and *ack_out* signals, latching the input data and acknowledging the request simultaneously. Request for next stage *req_out* is then produced. The inverted inputs of *req_out* and *ack_in* to the *a2* AND gate blocks the subsequent requests from *req_in* until the current data is consumed by the next stage. Feed-back loop from the *ack_out* to the *a1* AND gate helps to retain the logic high level of current acknowledgement until the *req_in* is lowered. Four-phase operation of the controller can be confirmed with the waveforms.

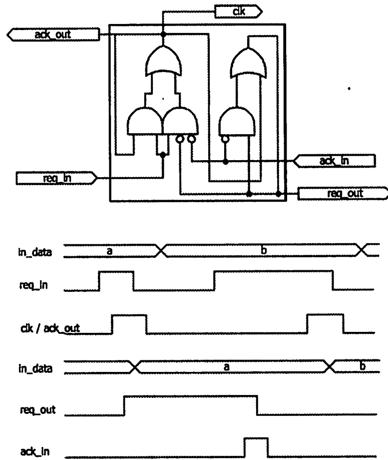


Fig. 5 4-phase Controller

3.3 Early Acknowledgement pipeline controller

The early acknowledgement pipeline controller proposed in this paper is shown in Fig. 6(a). The two additional control signals *start* and *complete* of this controller is to place the matched delay of the pipeline stage, they can be connected to each other in the case of there is no delay between pipeline stage or the overhead of the controller (which is considerably larger compared to MOUSETRAP and Four-phase controller presented previously) is larger than the delay between pipeline stages.

Though the Early Acknowledgement controller is not desirable for high-speed pipelines where the operational overhead of the controller is required to be minimum, in the case of a pipeline where processing between stages is greater than the overhead of controller, we have obtained operational performance near to that of MOUSETRAP controller which makes it very appealing 4-phase controller. We have observed similar performance as well in non-linear controllers based on this linear controller.

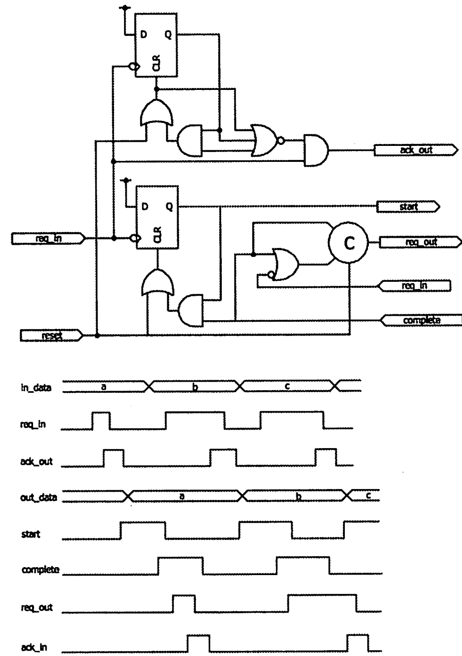


Fig. 6 Early Acknowledgement Controller

In the operation, for which the waveforms are shown in Fig. 6(b), initially all the control signals are low when the *req_in* is raised indicating a request from input side. This can immediately produce a acknowledgement (the 'early acknowledgement') *ack_out* since the *mask*, *mask_clr* and *out* are low opening the masking gate *m1*. The acknowledgement received at the input stage cause the request *req_in* to be lowered which occasion the following four actions to take place in parallel.

- *ack_out* is lowered which complete the hand-shake cycle between input stage
- *clk* positive edge occurs latching the data is to the current stage
- *mask* bit is set blocking further requests from input side
- *start* is raised as the controller goes to waiting state to produce a request to the next stage after the matched delay time is elapsed

When the matched delay time is elapsed and the *complete* is raised the *start* is reset back to zero and the request to the next stage *req_out* is raised. Mask bit is also cleared subsequently, yet *req_out* at high keeps the input request mask effective until the transaction between output stage is complete. When the next stage acknowledges the *req_out* by raising *ack_in*, it causes *req_out* to return to low, enabling the next request from *req_in* of input stage. The handshake at the output stage is completed when the *ack_in* is lowered in response to *req_out* going low.

Following three subsections details the implementation and operation of Conditional Branch controllers for the different type of hand-shake protocols.

3.4 2-phase Conditional Branch

For the transition signaling Conditional Branch controller, we have used a D-flop based controller in contrast to the D-latch based MOUSETRAP controller for linear pipeline. The controller based on [12] is shown in Fig. 7(a).

In the operation, as shown in the waveforms of Fig. 7(b), all control signals are at the same state and *complete* signal is high which indicates the operations of the output side of the controller is complete. The *select* signal can either be at high or low depending on the data or other control information which operates the branching operation. When a request is made with a transition on *req*, difference in states of *req* and *ack* generates the *clk* signal which is gated to *complete*. Since *complete* is high initially, the *clk* signal is raised latching the control and data signals. Once the *req* is latched the same transition occurs in the *ack* which acknowledges the request from input side. The *select* signal will be latched to *s1* and *s2* flip-flops in a way that one of registers will be flipped depending on the *select* signal generating the correct branching request. For example, if the *select* signal is low at the request *s1* is flipped generating a transition on *req1* making request on first branch.

Either of the request event cause the *complete* signal to go low indicating the latched data is being passed to the output stage, which will effectively blocks new requests from the input side. At the acknowledgement of the corresponding branch each pair of request and acknowledgement signals return to the same state, raising the *complete* signal high and re-enabling the requests from the input side.

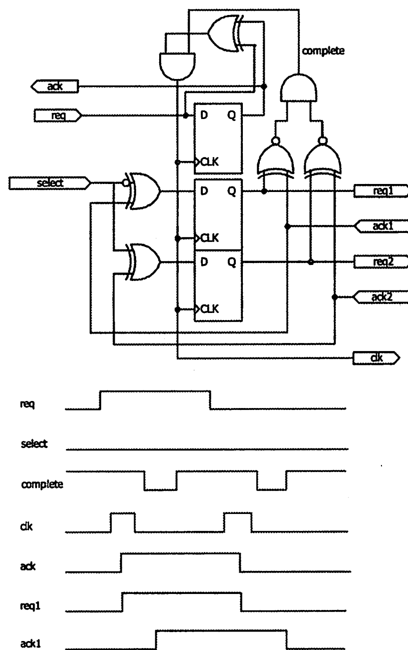


Fig. 7 2-phase Conditional Branch Controller

3.5 4-phase Conditional Branch

The 4-phase Conditional Branch controller is a simple extension of the 4-phase linear controller as shown in Fig. 8. The *select* signal is fed to a MUX which diverts the request *req* to either *req1* or *req2* conditional paths depending on the *select* signal. Since only one request is acknowledged from either *ack1* or *ack2* the acknowledgement to the linear controller from the conditional branches is can be simply ORed.

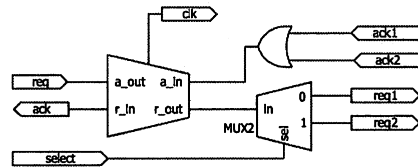


Fig. 8 4-phase Conditional Branch Controller

3.6 Early Acknowledgement Conditional Branch

The Conditional Branch for the Early Acknowledgement protocol is similar to that of 4-phase Conditional Branch controller shown in Fig. 8. Hence, the operation as described in 4-phase Conditional Branch is valid for the Early Acknowledgement Conditional Branch controller as well.

A notable point here is when adding a matched delay to the conditional branch. In a situation where the two branches have two different delays, using a common delay (higher of the two) would reduce the performance of the controller. In that case, branches should be matched individually considering the fact that the overhead of performance of the controller will be same on each branch.

Following three subsections details the Asynchronous Join controller for the different type of protocols. Asynchronous-join operation naturally requires to arbitrate between contending requests from inputs. Standard practice is to use a mutex element to select between such simultaneous requests. Mutex element is best implemented as an analogue circuit which is typically used in custom IC designs. However for standard cell based designs this kind of mutex elements are not available. We have employed the method of [7] which reduces the metastability of the cross-coupled NAND pair using additional circuit for feedback.

3.7 2-phase Asynchronous Join

We propose a new controller to perform the Asynchronous Join operation for 2-phase signaling using the transition arbiter [3] module as shown in Fig. 9.

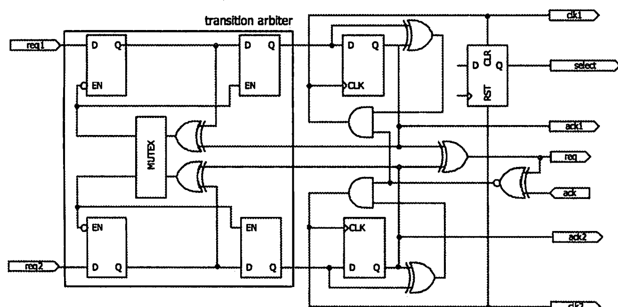


Fig. 9 2-phase Asynchronous Join Controller

The requests *ex_req1* & *ex_req2* are mutually exclusive requests of *req1* and *req2* which are generated using the transition arbiter grant signals. Operation of this circuit can be confirmed to that of intended Join controller behaviors.

Initially, all request, acknowledge signals and clock signals of the controller are low, except for the *complete* which indicate there are no request being processed at output of the controller. The latches and flip-flops are also initialized to zero. When a request is made in *req1*(*req2*) or simultaneously in both lines, arbiter selects one of the request exclusively by raising *ex_req1*(*ex_req2*). The difference in state of *ff_1*(*ff_2*) flip-flop and *ex_req1*(*ex_req2*) signal give raise to the *clk1*(*clk2*) there by capturing data of the selected input. Moreover, the select *signal* is appropriately set to 0(1) correctly selecting the input buffer to the output stage. As result of data capture the output of *ff_1*(*ff_2*) is raised which sends a transition on

- *ack1*(*ack2*) the acknowledgement to the request
- *done1*(*done2*) done signal for transition arbiter
- *req* transition to output notifying the availability of new data

The request *req* transition on the output stage, makes the *complete* signal go high there by lowering clk1(clk2) signals. It can be observed that further requests from the input side is not captured until *req* is acknowledged by *ack* (making *complete* to go high). This prevents the output register being corrupted before the completion by early inputs. Once the acknowledgement *ack* is received from the output stage controller is ready to process new exclusive request on *ex_req1* and *ex_req2* arbitrated by the transition arbiter.

3.8 4-phase Asynchronous Join

Asynchronous Join controller for 4-phase signaling is much simpler to implement using two linear controllers and MUTEX element. As shown in Fig. 10, MUTEX is used to arbitrated between possibly contending requests from the input branches. Mutually exclusive requests *ex_req1* and *ex_req2* generated by the MUTEX can be used for two linear controllers to latch the data in each branch and forward to a common output stage. The *select* signal for the demultiplexor is generated using the *clk1* and *clk2* signals as shown which guarantees the selection of appropriate branch of data to the output stage when the data is latched.

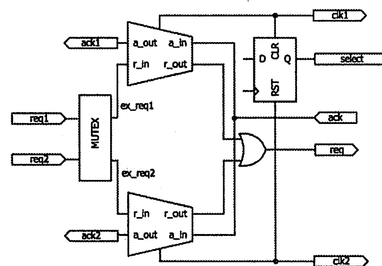


Fig. 10 4-phase Asynchronous Join Controller

Request signals from each controller is ORed to produce the request *req* to the output stage. The acknowledgement *ack* from the output is received by both controllers. The operation of the linear controller can be observed to be resilient to any 'false' acknowledgement (which will be received by one controller when acknowledging the other) received which makes this simple construction possible.

3.9 Early Acknowledgement Asynchronous Join

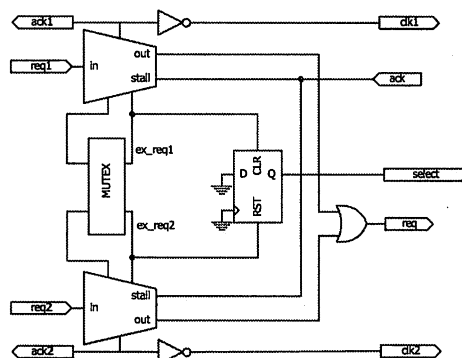


Fig. 11 Early Acknowledgement Asynchronous Join Controller

The construction of the Early Acknowledgement Asynchronous Join controller is similar to that of 4-phase Asynchronous Join controller described in the previous section. It also uses two linear Early Acknowledgement controllers and a MUTEX element to handle the joining paths and arbitrated between two simultaneous requests. As shown in Fig. 11, the matched delay *complete* signals are used for MUTEX inputs which allows each input stage data to be latched-in and acknowledged before arbitration begins. In this controller, the mutually exclusive request grant signals *ex_req1* and *ex_req2* (instead of *clk1* and *clk2* used in the 4-phase controller) are used to set the select signal appropriately.

4. Results

We have evaluated the performance of each of the controllers on Xilinx Vertex-4 FPGA. For linear controllers we have created simple 8-bit 4-stage FIFOs, operated by each type of controller. For the Conditional Branch controllers we have built a 8-bit Y-shaped pipelines with 4-stages where 2-stages are in the stem of the pipeline and 2-stages are branched out. The Conditional Branch controller is placed in the second stage of the pipeline. Similarly, for the Asynchronous Join controllers we have constructed Y-shape pipelines where two branches are joined by the controller to the stem of the pipeline. All pipelines were constructed to be 8-bit.

Environment for the pipelines comprised of input generating shift registers (two registers in the case of Conditional Branch) and output capturing registers (two registers in the case of Asynchronous Join) were operating with minimum overhead which maximize the performance of the controller under test.

Table 1 Cycle-times comparison

Cycle time	without processing (ns)	with processing (ns)
Linear		
2-phase (MOUSETRAP)	2.61	9.91
4-phase	4.63	18.43
Early Acknowledgement	5.49	10.62
Conditional Branch		
2-phase	3.96	11.12
4-phase	6.44	19.66
Early Acknowledgement	6.18	10.80
Asynchronous Join		
2-phase	5.98	11.33
4-phase	8.25	22.24
Early Acknowledgement	10.93	13.35

Performance of controllers were evaluated in two cases: without processing and with processing between pipeline stages. In the first case, there is minimum delay between stages without any processing elements in-between which evaluates the maximum performance of the controllers for high-speed pipelines. In the second case, performance of pipeline controllers for a general scenario of pipelines operating with processing in-between stages is tested. In order to emulate the processing elements we have used simple buffers to delay (between 6.9ns to 7.2ns (varied depending on the exact routing) the data-path. The matched delays for controllers were tuned starting from a higher delay to the lowest possible where the proper operation of the pipeline is guaranteed. Post-layout simulation results for Vertex-4 obtained using ModelSim are shown in Table 1.

It can be observed that the 2-phase controllers out perform the 4-phase and Early Acknowledgement controllers in linear and non-linear (Asynchronous Join and Conditional Branch) operations when there are no processing in-between pipeline stages. Conditional Branch for Early Acknowledgement controllers perform better than the 4-phase controllers in the case of Conditional Branch operation. In the cases where, processing elements are present between pipeline stages we observe that the Early Acknowledgement controllers perform better as its overhead got hidden in the required delay between stages. We got the performance comparable to that of 2-phase controllers in linear and Asynchronous Join cases and for the Conditional Branch controllers Early Acknowledgement controller perform slightly better than the 2-phase controller.

5. Conclusion

The performance of 2-phase signaling controllers are observed to perform better in the linear as well as non-linear pipelines, hence desirable for high-speed pipelines without any processing between pipeline stages.

For the Early Acknowledgement protocol based controllers the high operational overhead is apparent in both linear and non-linear operations when there are processing elements present in the pipeline. Yet, their performance is comparable to that of 2-phase controllers in the case of heavy processing element ladened pipelines where their operational overhead get hidden in the pipeline delays.

Reference

[1] M. Singh, S.M. Nowick, *MOUSETRAP: Ultra-High-Speed Transition-Signaling Asynchronous Pipelines*, Proceedings on Computer Design, 2001.

[2] E. Brunvand, *Using FPGAs to Implement Self-Timed Systems*, Journal of VLSI Signal Processing, Vol. 6, Issue 2, pp. 173-190, August 1993.

[3] E. Brunvand, *Translating Concurrent Communicating Programs into Asynchronous Circuits*, Ph.D. Dissertation, Carnegie Mellon University, 1991.

[4] T. Yoneda, et. al. *High Level Synthesis of Timed Asynchronous Circuits*, Proceedings on Asynchronous Circuits and Systems, 2005.

[5] R.O. Ozdag, et. al. *High-Speed Non-Linear Asynchronous Pipelines*, Proceedings on Design, Automation and Test in Europe, 2002.

[6] Quoc Thai Ho, et. al. *Implementing Asynchronous Circuits on LUT Based FPGAs*, Proceedings on The Reconfigurable Computing Is Going Mainstream, 2002.

[7] Y. Sato, et. al. *Systematic Reducing of Metastable Operation Occurred in CMOS D Flip-Flops* Systems and Computers in Japan, 2000.

[8] K.V. Berkel, F. Huberts, A. Peeters, *Streching Quasi Delay Insensitivity by Means of Extended Isochronic Forks* Proceedings on Asynchronous Design Methodologies, 1995

[9] M. Singh, S.M. Nowick, *High-Throughput Asynchronous Pipelines for Fine-Grain Dynamic Datapaths*, Proceedings on Advanced Research in Asynchronous Circuits and Systems, 2000.

[10] I. Blunno, et. al. *Handshake protocols for de-synchronization*, International Symposium on Asynchronous Circuits and Systems, 2004.

[11] M. Ampalam, M. Singh *Counterflow Pipelining: Architectural Support for Preemption in Asynchronous Systems using Anti-Tokens* Proceedings on International Conference on Computer-aided Design, 2006.

[12] Private communication with Montek Singh