Early Output Logic using Anti-Tokens

C.F. Brej and J.D. Garside
Dept. of Computer Science, The University of Manchester, Oxford Road, Manchester, M13 9PL, UK.

{cb,jdg}@cs.man.ac.uk

Abstract

Delay-insensitive dual-rail and bundled data design methodologies are the two main approaches used for the creation of asynchronous circuits. Bundled data allows the creation of fast, low overhead circuits, whereas dual-rail allows bit-level pipelining and true average case performance. This paper presents 'Early output' logic, which combines the positive features of the two methods to create faster asynchronous circuits. This method allows the creation of circuits yielding performance faster than synchronous counterparts.

Early output implementations allow logic to create results before all inputs are presented. The results move to the next stage, but the current stage stalls while waiting for the late inputs to arrive simply to acknowledge them. This unnecessary wait can be removed by allowing backwards propagating 'Anti-Tokens' to remove the late inputs. The use of anti-tokens and improved semi-decoupled latches allows the removal of many stalls due to unnecessary synchronisations, thus improving the performance of the circuit.

1. Introduction

The four-phase dual-rail [4] approach returns both wires to zero after each transaction. This allows fully quasi delay insensitive circuits to be created which do not need to hold state. Delay-Insensitive Minterm Synthesis (DIMS) is the approach usually taken to create QDI circuits. It allows logic to be constructed without the need for matched delays. Unfortunately DIMS gates are large, slow and power-hungry (fig. 1). Additionally, the four phase protocol forces each stage to waste as much time returning to zero as it used to calculate the data. Although DIMS works very well in creating bit-level pipelined and perfect average timed circuits, the gates are too slow to compete with other design styles.

2. Early output logic

Early output logic addresses some of the problems in DIMS logic to create fast circuits whilst preserving many of its beneficial properties.

Many of the problems in the DIMS approach are caused by the gates trying to manage the timing as well as the logic of the function. By separating the timing and logic parts of the circuit (and losing the 'QDI-ness'), a faster circuit can be created, as demonstrated with Phased Logic. The communication protocol remains QDI and timing assumptions are only applied in local logic.

DIMS gates are large and slow due to the C-elements required to ensure the output only rises when all inputs are valid and falls only when they both return to NULL. If these restrictions are moved into separate guarding logic the gates can be much smaller and faster.

Effectively all the gate is now required to do is to execute the logical operation on the dual-rail inputs. Figure 1 shows an early output OR gate compared with its DIMS counterpart. The delay through the gate is equal to that of a single gate stage.

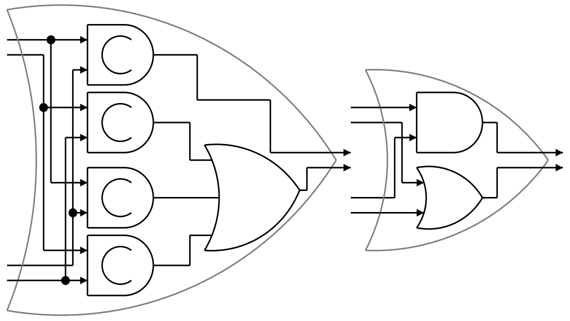


Figure 1: DIMS and Early output OR gates

Figure 2 shows the output of DIMS and early output OR gates. The early output gate outputs early in two cases (marked with a *). Although this is beneficial because the result arrives at its destination sooner it does not ensure that all inputs have arrived.

DIMS OR

	0	N	1
0	0	N	1
N	N	N	N
1	1	N	1

Early Output OR

	0	N	1
0	0	N	1
N	N	N	1*
1	1	1*	1

Figure 2: DIMS vs early output behaviour

A DIMS gate:

1) outputs NULL when all inputs are NULL.
2) executes the required logical operation.
3) only outputs a valid value when all inputs are valid.
4) only returns to NULL when all inputs are NULL

The early output only has properties 1 and 2, so some other mechanism must be provided to ensure correct operation.

2.1. Guarding

Property 3 (above) ensures that only when all inputs into a stage are valid will the result become valid. In an early output logic system this is undesirable. Instead the requirement is merely that all the inputs must have been asserted before they can be acknowledged.

Figure 3 shows an example of an early output pipeline stage. Two levels of C-element are used to guarantee operation. The first C-elements (C1 and C2) are adjacent to the output latches; these 'guarding' C-elements produce an acknowledge when the output latches have captured the input data – the timing of which is implicit in the (grey) data signals – and all the contributing input stages have output valid data. These then signal an acknowledgement to the input latches. This ensures that a latch will not receive an acknowledge until it is ready and raises it has output data.

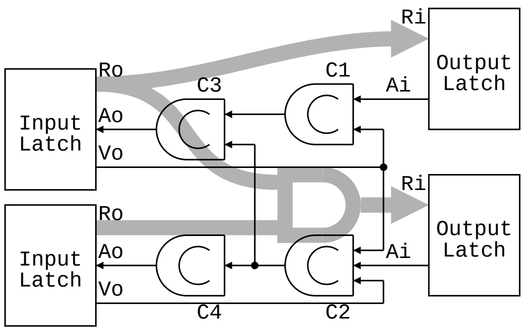


Figure 3: Guarding example

The second set of C-elements ensures that all the latches the data has been sent to have acknowledged before the input latch may change. In many cases (such as C4 in fig. 3) these elements may be degenerate and can be removed and in other cases optimisations are possible.

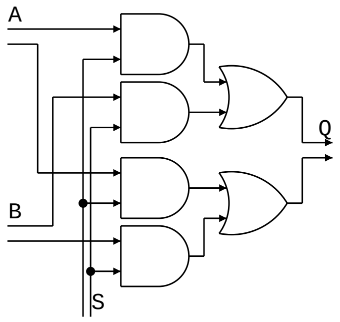
These C-elements also ensure that all relevant latches achieve a NULL state in between data values (property 4, above).

This mechanism requires an additional 'valid' (Vo) signal to accompany the data assertion. In a four-phase, dual-rail system this may simply be provided by an OR of the input bits. This signal may already be available within the latch controller (see fig. 6).

There is a hazard introduced by this approach in that late-arriving (unnecessary) data will begin to ripple through the logic in parallel with its 'valid' signal triggering its removal. It is therefore necessary for the logic designer to ensure that this 'runt' data does not both survive and have a sufficient delay for it to reach the an early output gate after the other inputs have returned to NULL. In this circumstance a false data packet could be introduced. However the timing constraints on this appear to be fairly easy to meet.

2.2. Early output states

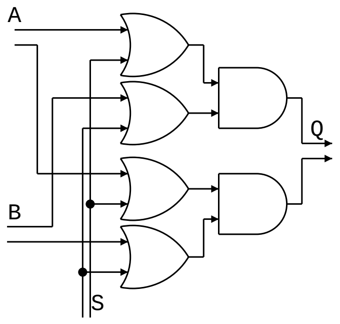
Early output gates may output a value before all inputs are valid. This increases the speed of computation. These early output states allow the result to move to the next stage while current stage waits for all inputs to arrive before acknowledging them. A good early output design will use as many of these early output cases as possible. Figure 4 shows a possible design for a dual-rail 2:1 multiplexer along with its truth table. Although the design is correct it does not capture all early output states.



		S		
A	**B**	**N**	**0**	**1**
N	N	N	N	N
0	N	N	0*	N
1	N	N	1*	N
N	0	N	N	0*
0	0	N	0	0
1	0	N	1	0
N	1	N	N	1*
0	1	N	0	1
1	1	N	1	1

Figure 4: Early output multiplexer design

Figure 5 shows an improved design which catches two extra early output cases. This circuit will be able to create a valid output if both data inputs are equal and thus the select input is irrelevant.



		S		
A	**B**	**N**	**0**	**1**
N	N	N	N	N
0	N	N	0*	N
1	N	N	1*	N
N	0	N	N	0*
0	0	0*	0	0
1	0	N	1	0
N	1	N	N	1*
0	1	N	0	1
1	1	1*	1	1

Figure 5: Alternative multiplexer design

By rearranging logic it is often possible to create circuits which capture more of the early output cases. Circuits with more early output cases will stop and unnecessarily wait for late inputs less often.

3. Semi-decoupled latches

Although gates can output results early, guarding logic ensures that a pipeline stage waits for the late inputs before acknowledging. During this time the data on the output is valid and will remain so until the last input arrives. This stops any subsequent stages from moving more than half a cycle ahead. The later stages can calculate the result with the input just fed to them but, after entering the reset phase, they will wait for the input to return to zero. This takes time and, ideally, the stage should be allowed to work on the next set of data.

A latch between the stages could remove the value from the input to allow the stage to complete the acknowledgement. This latch would have to wait until the input has returned to zero before continuing to pass data. These are the properties of a semi-decoupled latch.

Figure 6 shows the standard dual-rail latch design [3] which has been adapted for use in early output logic. The OR gate is used to both provide an acknowledge backwards and a validity forwards.

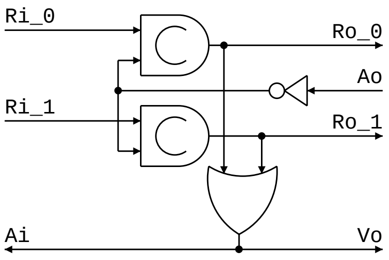


Figure 6: Standard dual-rail latch

Figure 7 shows a commonly used dual-rail semi-decoupled latch. The data outputs of a semi-decoupled latch will fall once the acknowledge signal reaches it, even if the input data has not returned to zero. The component is much more complex than the standard dual-rail latch, and it requires a C-element to store some state. The state holding C-element fires when the output signal is acknowledged. It remains on until the data C-elements have returned to zero.

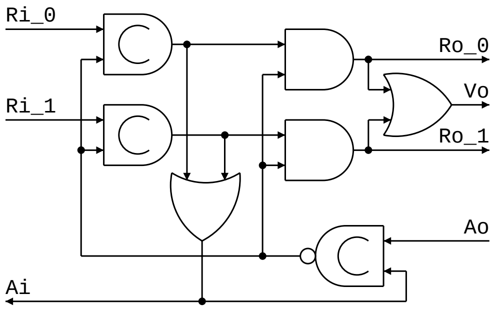


Figure 7: Standard semi-decoupled latch

The semi-decoupled property of the latch is required to make circuits with less internal synchronisations. Unfortunately, the cost in performance and area of the above design is very high.

3.1. Early output semi-decoupled latch

A cheaper semi-decoupled latch is required to allow circuits to operate with fewer synchronisations. All latches used by early output circuits described in this paper output a validity signal. The Vo (Valid out) line is connected through C-elements to the Ao (Acknowledge out). This allows the data signals to drop but the Ao line will remain high until Vo has dropped. This stops the stage from completing the acknowledge but later stages can complete the cycle and start a new one.

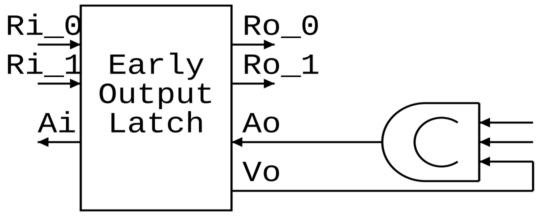


Figure 8: Vo to Ao connection

Using early output logic it is safe to force data to zero and still ensure the stage does not complete the acknowledge phase. Figure 9 shows a much cheaper semi-decoupled latch design. The cost of the two AND gates is very low. The latch also is faster to react to acknowledge signals. Due to the sequencing of the transitions, each of the AND gates can be implemented using only two transistors.

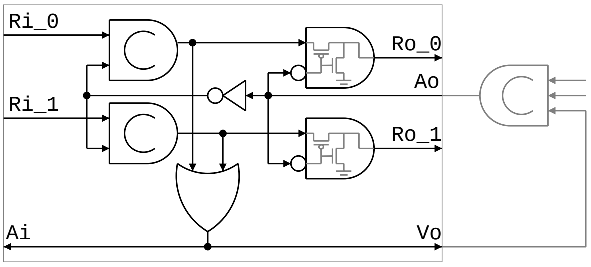


Figure 9: Early output semi-decoupled latch

Data out lines (Ro_0 and Ro_1) are driven high when the C-elements switch high, and low when the acknowledge becomes high. Due to this strict sequencing, the AND gate can be created with a P-type pass transistor, to propagate the data when the acknowledge is low and an N-type transistor to force the signal to ground when the acknowledge is high.

4. Anti-Tokens

In early output circuits the 'valid out' (Vo) signal can delay a transition on the acknowledgement signal until the latch is ready to accept it. Normally it is used to delay the acknowledgement until the latch has some data to destroy, or to prevent the acknowledgement from being released until the latch has finished returning to zero. In the semi-decoupled latch the direct relation of the data output to the validity signal is broken. The valid signal no longer states the condition of the outputs but is used to control the acknowledge signal.

By asserting the valid signal before outputting any data the latch opens itself to receive an acknowledgement. This will happen in early output cases. Normally latches are not ready to receive an early acknowledgement as they do not have any data to destroy. If the latch were to deliberately raise the valid signal and receive an early acknowledgement, the correct action by the latch would be to destroy the next token it receives. It can then release the validity signal and allow the stage to continue working on the next set of data tokens. The latch is now in a state where it will to destroy one token and then resume operating normally. This can be thought of as holding an “Anti-Token”. As a latch can only hold one anti-token it has to prevent the following stage from giving it another early acknowledgement. This is done by simply not raising the valid line early.

A latch can raise its acknowledge signal even before it receives any data, as the acknowledgement will not reach the input latches until they all raise their valid lines. A latch holding an anti-token can acknowledge early but has to keep the acknowledge high until all inputs raise their valid lines and the guarded acknowledge has activated. This is because the latch cannot sense if the stage moved to the reset phase by only observing the data lines. The data lines never went high so the latch can’t wait for then to return to zero.

To allow the latch to snoop on the state of the previous stage the guarded acknowledge line (Vi) is fed into the latch. This allows the latch to send and receive early acknowledgments.

4.1. Anti-tokens and logic

Anti-tokens can move backwards through logic as well as FIFOs. Figure 10 shows a situation where an anti-token has arrived at a logic stage where some of the inputs have arrived. In such a case, if the remaining latches which have no output data are anti-token latches, they will receive an early acknowledge and accept an anti-token. The latches with data will receive an acknowledge and reset to zero. If any of the latches which do not hold a data token are not anti-token latches and have not raised their validity early then an acknowledgement cannot be completed until they receive a data token. In early output logic, it is safe to mix anti-token and non-anti-token propagating latches together, as only latches which accept anti-tokens will be able to raise their output validity early.

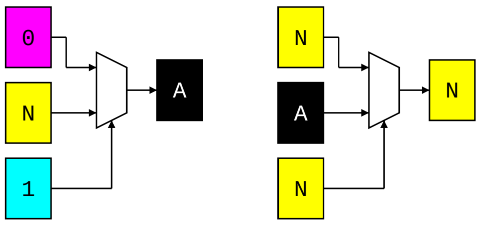


Figure 10: Anti-token propagation through logic

Early output logic generates anti-tokens in early output cases. The example in figure 11 shows a multiplexer with only the data inputs valid. As shown in section 2.2, a 2:1 multiplexer can be designed so that in some cases it can create a valid output before the select signal arrives. Once the result arrives at the output latch, it can then acknowledge. If the acknowledgement passes through the guarding logic then all the latches which do not hold any data will accept anti-tokens and the latches that do will receive an acknowledge.

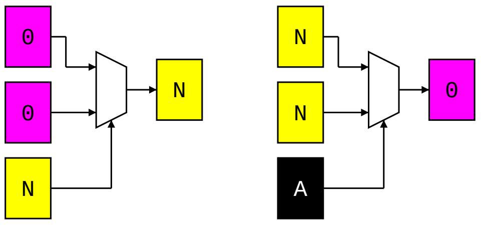


Figure 11: Anti-token generation

4.2. Anti-Token pipeline

Figure 12 shows an example of an early output circuit. In this circuit the two data C-elements are abstracted to just one and are left outside the latch to allow easier description. The C-element is driven by signal S from the latch control unit. The C-element is asymmetric because while passing an anti-token the S signal is withdrawn before the data arrives. This can cause the C-element to become metastable and so the S signal instead resets the C-element when it drops. The Ai signal is combined with Vo signal to create the guarded acknowledge. This can then be passed to the input latch as the Ao and to the output latch as Vi. The latch also snoops on Ri and Ro signals. The Ri signal is observed to ensure that the data C-elements are not re-enabled while there is still data from the previous operation.

Figure 13 shows a Burst-mode machine description of the latch controller. State 0 is the initial state, in which outputs S and Vo are high while Vo and Ai are low. In state 0 the latch is waiting for the sign of a token or an anti token. If the Ro+ transition happens first the latch will go through states 1, 2 and 3 while passing a token. And if the Ao+ arrives first then the latch will pass an anti-token by going

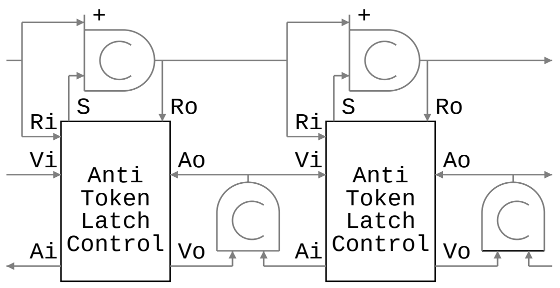


Figure 12: Anti-token FIFO

through states 4 and 3.

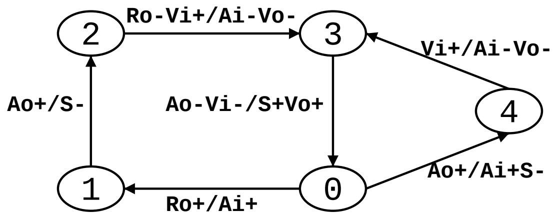


Figure 13: Description of the anti-token latch Burst-mode machine

4.3. Anti-token latch behaviour

Figure 14 depicts the operation of the anti-token latch when passing a token. In state 0 both S and Vo signals are high and so both the data and the guarding C-elements are primed to fire. In this example the data C-element will fire first and raise Ro. The latch then moves through a token passing action similar to that of standard dual-rail latches. The only difference being that dropping the S signal directly forces Ro to drop, even if Ri is still high. Instead of waiting for Ro to drop, Ri is tested directly to check it has returned to zero before releasing the acknowledge. This gives the latch a semi-decoupled behaviour.

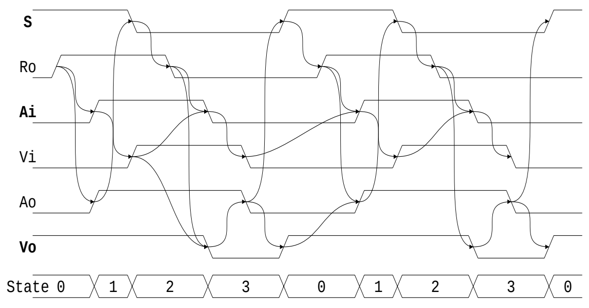


Figure 14: Token passing

The anti-token pass is much simpler than the token pass. In figure 15 the latch receives the Ao+ transition first. Ao+ causes S to be withdrawn. As during the anti-token pass the transitions on S are not observed, and so the circuit is not QDI. Once S is low any data trying to pass into the latch will be ignored. Because Ro could rise just before S drops whilst passing an anti-token, the data C-elements must be asymmetric. This allows the falling of S to force the output low even if Ri is high. As this can cause a short glitch on the data output the system has to rely on some timing assumptions. The depth of the pipeline stage that a glitch can propagate through must be shorter than the reset cycle time.

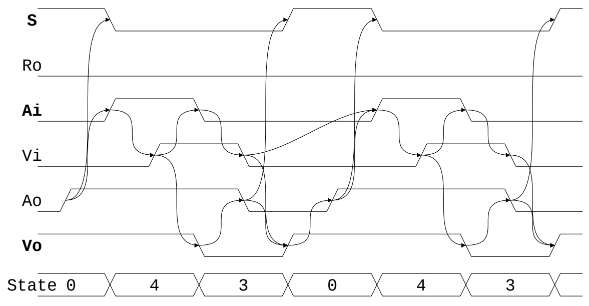


Figure 15: Anti-token passing

There is no need for arbitration in the circuit as either of the two initial transitions (Ro+ or Ao+) will have the same effect (Ai+). The anti-token pass is equivalent to the token pass with the exception that during a token pass the latch waits for the data lines to drop before continuing. As the data lines remain low during an anti-token pass, no stall is required.

4.4. Anti-token latch schematic

The anti-token latch can be synthesized from the Minimalist description or by hand. Figure 16 shows a schematic of the latch, synthesized by Minimalist[9] and then optimised by hand. The circuit is only slightly larger than the standard semi-decoupled latch and still keeps the semi-decoupled property.

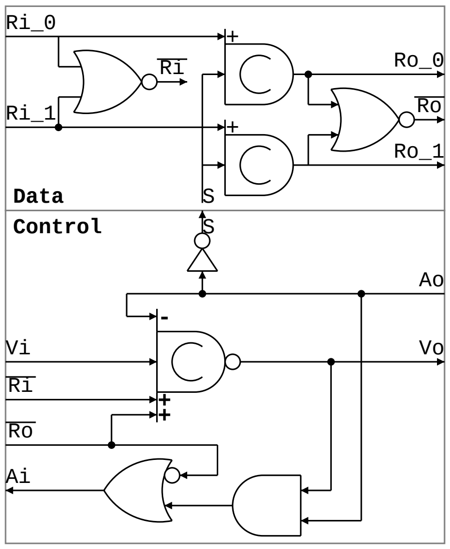


Figure 16: Anti-token latch schematic

4.5. Anti-token/token collisions

Figure 12 showed a simple pipeline along which tokens and anti-tokens can flow. A token and an anti-token can collide across any latch or logic. If they collide across logic then the input latch assumes that its data is being acknowledged and the output latch assumes that the anti-token was being accepted. As the output latch passing the anti-token resets the data C-elements it does not listen to the data lines.

If the collision happens across a latch then there are two different scenarios. If the anti-token arrives first (Ao+) then the S line drops and the latch refuses to listen to the data lines. The latch then proceeds to acknowledge the input so the collision happens across the logic of the previous stage.

If the token arrives at the same time as the anti-token then the Ro lines could raise and soon after fall again. This glitch can last for less than a single gate delay. The rogue signal will then travel through the logic and even if it reaches the output latch then the signal will be ignored. If the logic stage is very deep then timing assumptions need to be met for the circuit to operate correctly.

5. Results

Figure 17 shows the gate delay count of all latches shown. The units are measured in gate delays of the optimised versions of the latches. C-elements require 2 gate delays to switch and the optimisations remove the need to invert the Ao line.

The early-output semi-decoupled latch gives much improved results over the standard semi-detached latch. Ao↓ to Vo↑ delay is two times smaller and the overhead over the original design is so low that it seems beneficial to use the early-output latches as a standard latch throughout most designs. The anti-token latch is also only marginally slower and in the case of Ao↓ to Vo↑ it is actually faster than any other latch. The only problem with using anti-token latches as standard is their size.

6. Conclusion

Early output logic looks very promising when compared with synchronous designs for speed. It is important to remember that although the speed improvement might be sought after, the area and power consumption costs are very high.

Early output logic is the basis to create better features such as improved semi-decoupled latches and anti-token latches. Although these latches look very beneficial, further work must be conducted to show where they should be placed in a design to gain a positive effect.

Early output circuits require some timing assumptions in the logic part of the circuits. This can be easily met if the logic functions are less than 4 inversions deep. This restriction requires very fine grain pipelining. A method of finding these timing requirements and reorganising the logic, placing delay lines or adding extra inputs to guarding C-elements is required to allow a more flexible designs.

From	To	Original	Standard S-D	Early output S-D	Anti-token
Ri_?↑	Ro_?↑	2	3	3	2
Ri_?↑	Ai↑	3	3	3	4
Ri_?↑	Vo↑	3	4	3	N/A
Ao↑	Ro_?↓	2	2	2	2
Ao↑	Vo↓	3	4	3	5
Ao↑	Ai↑	N/A	N/A	N/A	2
Ri_?↓	Ai↓	3	3	3	4
Ri_?↓	Vo↓	3	N/A	3	4
Ao↓	Ro_?↑	2	5	3	2
Ao↓	Vo↑	3	6	3	2
Token Pass		14	20	16	19
Anti-Token Pass		N/A	N/A	N/A	16
Transistor count		24	42	28	41

Figure 17: Transition delays

7. References

[1] J. Sparsø and S. Furber, "Principles of Asynchronous Circuit Design", Kluwer Academic Publishers, 2001, (ISBN 0-7923-7613-7)

[2] K. Van Berkel, F. Huberts and A. Peeters, "Stretching Quasi Delay Insensitivity by Means of Extended Isochronic Forks", Proceedings of the Second Working Conference on Asynchronous Design Methodologies, London, UK, 1995.

[3] S. Furber and P. Day, "Four-phase micropipeline latch control circuits", IEEE Transactions on VLSI Systems, vol. 4, June 1996.

[4] D.E. Muller, "Asynchronous logics and application to information processing", Switching Theory in Space Technology, Stanford, University Press, Stanford, CA, 1963.

[5] D.H. Linder and J.C. Harden, "Phased Logic: Supporting the Synchronous Design Paradigm with Delay-insensitive Circuitry", IEEE Transactions on Computers, Vol. 45, No 9, September 1996.

[6] R. B. Reese, M. A. Thornton and C. Traver, "Arithmetic Logic Circuits using Self-timed Bit-Level Dataflow and Early Evaluation", Proceedings of the 2001 Conference on Computer Design, September 2001.

[7] R.F. Sproull, I.E. Sutherland and C.E. Molnar, "Counterflow Pipe-line Processor Architecture", Sun Microsystems Laboratories Technical Report, April 1994.

[8] C.F. Brej, "An automatic synchronous to asynchronous circuit convertor", 11th UK Asynchronous Forum, 2001.

[9] R. Fuher and S. Nowick, "MINIMALIST: An Environment for the Synthesis, Verification and Testability of Burst Mode Asynchronous Machines", Department of Computer Science, Columbia University Technical Report, 1999.