Department of Computing Science,
University of Newcastle upon Tyne

# Synthesis of Asynchronous Circuits with Predictable Latency

A. Bystrov, A. Yakovlev

Technical TR Series

CS-TR-754

7th December 2001

Contact:
A.Bystrov@ncl.ac.uk
Alex.Yakovlev@ncl.ac.uk

EPSRC grant GR/M94366


Published by the University of Newcastle upon Tyne,
Department of Computing Science, Claremont Tower, Claremont Road,
Newcastle upon Tyne, NE1 7RU, UK

# Synthesis of Asynchronous Circuits with Predictable Latency

A. Bystrov, A. Yakovlev

7th December 2001

**Abstract**

A new method for low-latency asynchronous circuit design uses a two-level architecture. It consists of the explicit context logic and output flip-flops. Explicit context logic computes a single context signal for each output concurrently to the environment operation. Every flip-flop generates an output from the corresponding context and trigger signals. The flip-flop latency for every transition is defined by the number of corresponding trigger transitions and can be predicted at an early stage of the design. Explicit context logic is generated by a logic synthesis tool, which produces a near logarithmic state encoding. This is especially beneficial for the designs from specifications having implicit (hidden) counters.

## 1 Introduction

Constantly growing speed and integration level of modern VLSI circuits makes the issues of power consumption, electromagnetic compatibility, clock skew, robustness and scalability increasingly important. Asynchronous circuits compare favourably to synchronous designs with respect to the above criteria, however their support in industrial design tools is very poor. One of many reasons for this is it is virtually impossible to predict the latency of an asynchronous circuit at the stage of specification. This is different from the specification of synchronous automata, whose performance is rigidly defined by clock.

In this paper we describe a design method for asynchronous circuits with predictably low latency. This is achieved by introducing a fixed two-level architecture with a standard implementation of the interface circuitry. The latency of an output in this approach depends only on the number of inputs directly preceding the output, which can be easily estimated at an early stage of circuit specification.

This method is heavily based on the ideas of explicit context implementation used in [3], where a direct syntax mapping [8, 10, 2, 11, 18] was used to implement internal context signals. Benchmark study has shown that the above method produces very fast circuits with predictable latency, though the size of such circuits can be large due to the 1-hot encoding of local states (Petri net places) in the direct mapping methods. For example, if a specification contains a counter and the counter is not represented as an explicit interface to a known counter implementation, then the direct mapping approach would generate a memory cell for every state of such a counter, which is not economical. Logic synthesis methods, such as [15, 7, 9], are better suited for such specifications as they produce a 'near logarithmic' state encoding and use significantly less memory elements (internal signals). This quality is often achieved at the expense of the lower speed, as at every stage of operation the circuit has to 'decode' the state by observing a large number of signals. The effect of speed reduction, being extremely undesirable in many applications, has a minimal impact on the latency of circuits designed by the proposed method. The reason for this is that the logic synthesis approach is applied only to the synthesis of context internal signals, which are calculated concurrently to the environment operation. Once such a signal has been computed, the circuit interface waits for the environment to issue the 'trigger' signals (the inputs directly preceding the output) and generates the output. The interface part has a standard flip-flop implementation and its latency, being very small, depends only on the number of 'triggers'.

To summarise the above, in the proposed method we combine the low latency of interface circuitry and the compactness of 'near logarithmic' state encoding within the two-level circuit architecture.

## 2 Model

In this paper we use *1-safe* PNs to capture concurrent behaviour of an asynchronous system and *signal transition graphs* (STG) as a circuit specification. Traditionally, a PN (a *net system*) is defined as a tuple $\Sigma = \langle P, T, F, M_0 \rangle$ comprising finite disjoint sets of places $P$ and transitions $T$, flow relation $F \subseteq (P \times T) \cup (T \times P)$ and initial marking $M_0$. There is an arc between $x$ and $y$ iff $(x, y) \in F$. The *preset* of a node $x$ is defined as $\bullet x = \{y \mid (y, x) \in F\}$, and the *postset* as $x\bullet = \{y \mid (x, y) \in F\}$. A *marking* is a mapping $M : P \to N = \{0, 1\}$ ($N$ is binary for 1-safe PNs). It is assumed that $\bullet t \neq \emptyset \neq t\bullet$ for every transition $t \in T$. The evolution of a PN from the initial marking $M_0$ to a marking $M_n$ by executing transitions results in a *firing sequence* $\sigma = t_1 t_2 \ldots t_n$, where $t_i$ are such that $M_i [t_{i+1}\rangle M_{i+1}$, for $i = 0, \ldots, n-1$. Moreover, $M_n$ is called a *reachable marking*.

A Signal Transition Graph (STG) is a PN whose transitions are labelled by signal events, i.e. $STG = \langle P, T, F, R, M_0, \lambda \rangle$, where $\lambda : T \to A \times \{+, -\}$ is a labelling function and $A$ is a set of signals. Usually, the labelling assigns the same label to several transitions. Note, that a set of read-arcs $R$ has been included into the model of STG, which is an enhancement w.r.t. [16, 4]. Another specific property of an STG is signal consistency, i.e. when the net is executed no signal can have two transitions of the same polarity without having a transition of the opposite polarity between them.

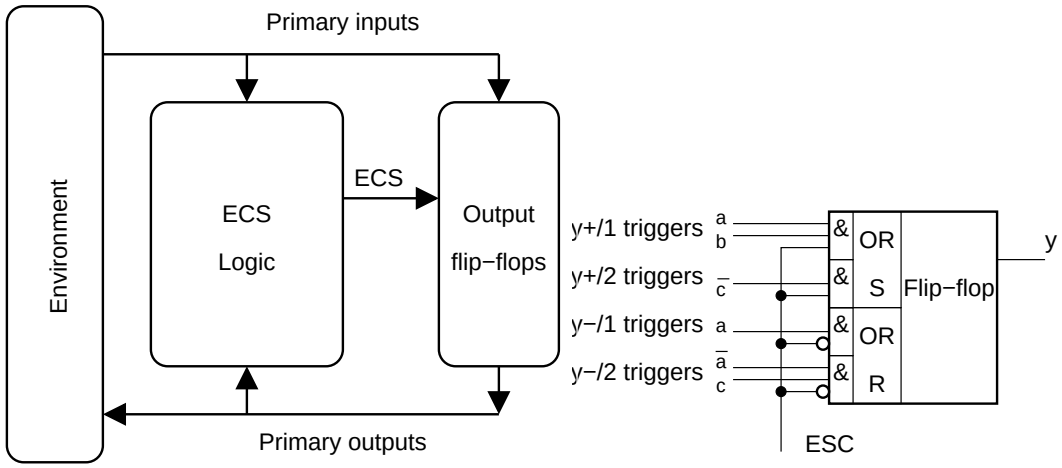
## 3 Two-level circuit architecture

The latency of a circuit is a delay between an input signal event and the output event directly succeeding it. Circuits generated by logic synthesis tools such as Petrify [7] compute an output as a function of the global state. They do not differentiate between the signals whose transitions directly precede the given output (trigger signals) and the other signals used in the output function (context signals). The number of context signals for the given output can be large, which increases the latency of the logic gate implementing the function.

In our approach we modify the circuit specification by introducing additional internal 'explicit context' signals (ECS), one per output signal, thus creating a two-level architecture shown in Figure 1(a). The first level is the ECS logic, which 'predicts' the output and computes the corresponding ECS concurrently to the environment operation. The second level is formed by output flip-flops, such as shown in Figure 1(b). A *single* ECS replaces possibly multiple context signals in the output function. It is extremely important that the function implementing an ECS does not include trigger signals and, hence, can be computed in advance, concurrently to the environment calculating the trigger signals. This is the key to the latency reduction. The ECSs can be synthesised by Petrify or by any other tool, their latency is not important.

The specification of an ECS circuit is illustrated in Figure 2. A fragment of the original specification shown in Figure 2(a) contains inputs $\{a, d\}$ and outputs $\{b, c\}$. For each output signal an ECS is created and named by adding 'r_' in front of the corresponding output signal name (the letter 'r' stands for the request of the output). The STG specifying the ECS signals is derived from the original STG by the following algorithm illustrated in Figure 2(b):

1. Make two copies of the original STG.
2. Rename all output transitions of the second STG into the output request transitions using, for example, the naming convention described above.




(a) Overall structure

(b) Output flip-flop

Figure 1: Circuit architecture

3. Replace all input and internal transitions in the second STG by dummy transitions (transitions without signal labels). This allows the ECS model to 'run ahead' of the original specification, thus forming the ECSs before the trigger events have arrived from the environment. The ECSs inherit the property of signal consistency of the corresponding outputs.

4. Create a handshake connection between every output request transition and the corresponding output. For this do the following:

   (a) Insert an additional dummy after each output request transition in the second STG. At this point we assume that all outputs are persistent and deterministic signals, which permits 'transition splitting'.

   (b) Connect every output request transition of the second STG to the corresponding output transition of the first STG by a consuming arc.

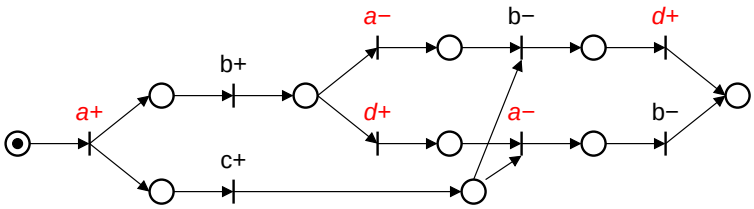
   (c) Connect every output transition of the first STG to the dummy directly succeeding the corresponding output request transition of the second STG by a consuming arc.

5. Rename all output transitions in the first copy into inputs. This corresponds to the flip-flop outputs being treated as inputs by the ECS logic in Figure 1(a).

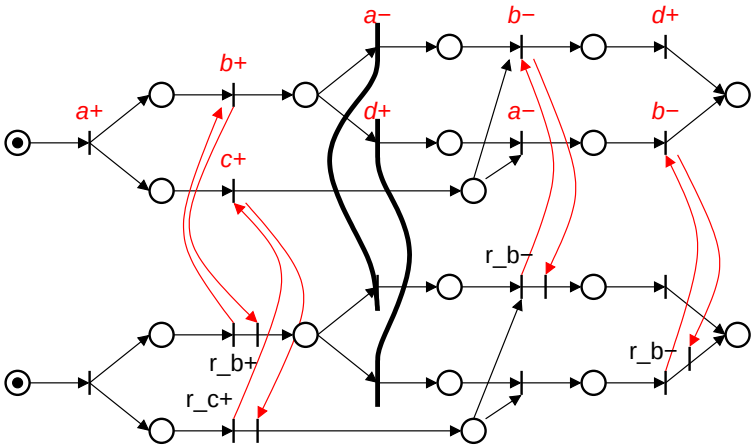
6. Merge every transition following a non-deterministic choice (free choice, arbitration, confusion) in the first STG and the corresponding dummy of the second STG into a single transition labelled as the input transition. Merging means that two transitions are replaced by a new transition (depicted as a thick curved line), their incoming and outgoing arcs are connected to the new transition. All such non-deterministic signals must be inputs.

7. Similarly merge every transition on internal signals of the first STG and the corresponding dummy of the second STG into a single transition labelled as the corresponding internal transition.

This algorithm assumes that the original STG has a complete state coding (CSC) property as defined in [6]. This means that the appropriate tool, such as Petrify, should be run to check this property and, if needed, to insert CSC internal signals before the application of the above algorithm. The original specification conforming CSC guarantees that the algorithm produces the STG having the property of CSC.




(a) Original STG




(b) ECS specification

Figure 2: ECS insertion

The algorithm is applicable only to specifications having persistent output signals, i.e. any transition on such a signal must fire once it has been enabled. There is also a requirement of the immediate non-deterministic choice identification, which means that the transitions directly succeeding the non-deterministic choice place must have different labels. These restrictions are required to guarantee strong equivalence between the original STG and the ECS STG, which is the *observational equivalence* or *weak bisimulation* [14] w.r.t. the signals present in the original STG.

# 4 Case study

In this section we compare the proposed method against the direct application of logic synthesis tool Petrify [7] to the well known benchmark of a VME bus controller. The first stage in both approaches in CSC checking and solving. In both cases it is done by using Petrify. The resultant CSC specification is shown in Figure 3.

The second stage in the proposed method is STG transformation by the above algorithm. Its result is shown in Figure 4. All primary input and output signals are transformed into the inputs of ECS specification and labelled in italics. Output request signals are outputs of this level and internal CSC signals are kept as internal. This STG has a single non-deterministic (free) choice, whose options are input transitions $dsr+$ and $dsw+$. These are used as choice identifiers in the right part of STG.

This STG is weakly bisimilar to the original STG w.r.t. the all signals of the original STG. It also satisfies the properties of signal consistency and CSC. The circuit in Figure 5(a) is a complex gate implementation of this model. The low latency flip-flops in Figure 5(b) form the primary outputs out of output request signals. Their set and reset transistor meshes have been optimised for the minimum number of transistors. The output

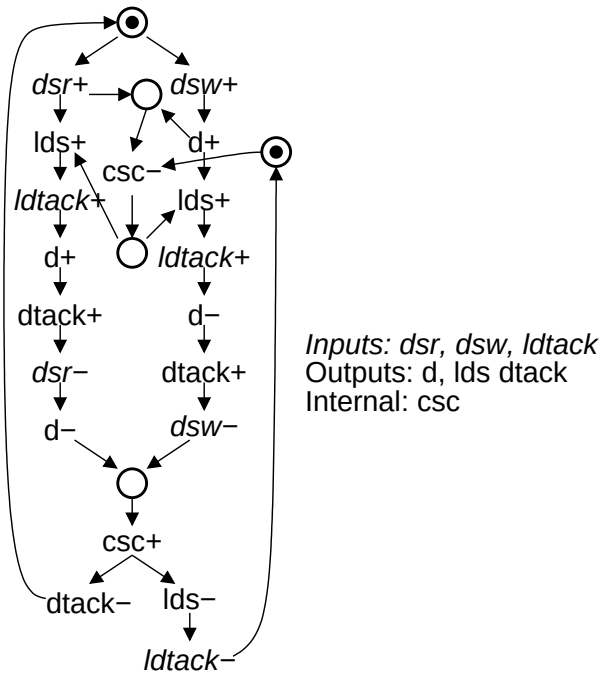



Figure 3: VME bus controller STG

signals *dtack* and *lds* coincide with *r_dtack* and *lds*, hence flip-flops are not required.

The analysis of the schematic shows that in spite of the use of slow complex gates, the latency of $d+$ and $d-$ output transitions is minimal (one gate delay and, depending on the combination of polarities, an input inverter). Other output transitions, however, are still controlled by complex gates. This is due to the nature of the original specification and to the way Petrify solves CSC conflicts. Indeed, transitions $dtack-$ and $lds-$ happen directly after the internal signal transition $csc+$, inserted by the tool, which defines their latency. Possibly, a smarter placement of CSC signals may reduce their latency. This issue, however does not belong to the subject of this paper. Transition $dtack+$ also has no input triggers, hence its latency is defined by complex gates implementing $r\_dtack+$. Reduction of its latency is only possible by modification of the original specification.

The reference circuit in Figure 6 has been directly generated by the Petrify tool out of the specification shown in Figure 3. All its outputs are formed by complex gates, which define their latency.

The latencies of these two designs are compared in Table 1. Both implementations use AMS-CUQ-0.6$\mu$ process. The comparison shows a significant latency reduction for the output $d$, implemented using an output flip-flop. Other outputs have the same latency.

Table 1: Comparison of VME bus controller implementations

| Transition | Petrify | Low latency design |
|---|---|---|
| $ldtack+ \rightarrow d+$ | 0.89ns | 0.29ns* |
| $ldtack+ \rightarrow d-$ | 0.94ns | 0.16ns* |
| $d+ \rightarrow dtack+$ | 0.38ns | 0.38ns |
| $dsw- \rightarrow dtack-$ | 1.19ns | 1.19ns |
| $ldtack- \rightarrow lds+$(read mode) | 0.61ns | 0.61ns |
| $ldtack- \rightarrow lds+$(write mode) | 0.60ns | 0.59ns |
| $dsr- \rightarrow d-$ | 0.59ns | 0.22ns* |

The above example has been specifically chosen for the analysis of different problems related to the proposed method. It has many output transitions directly preceded by an output or by an internal signal, the situations when the delay cannot be reduced. However, it is questionable whether such a delay can be treated as latency.

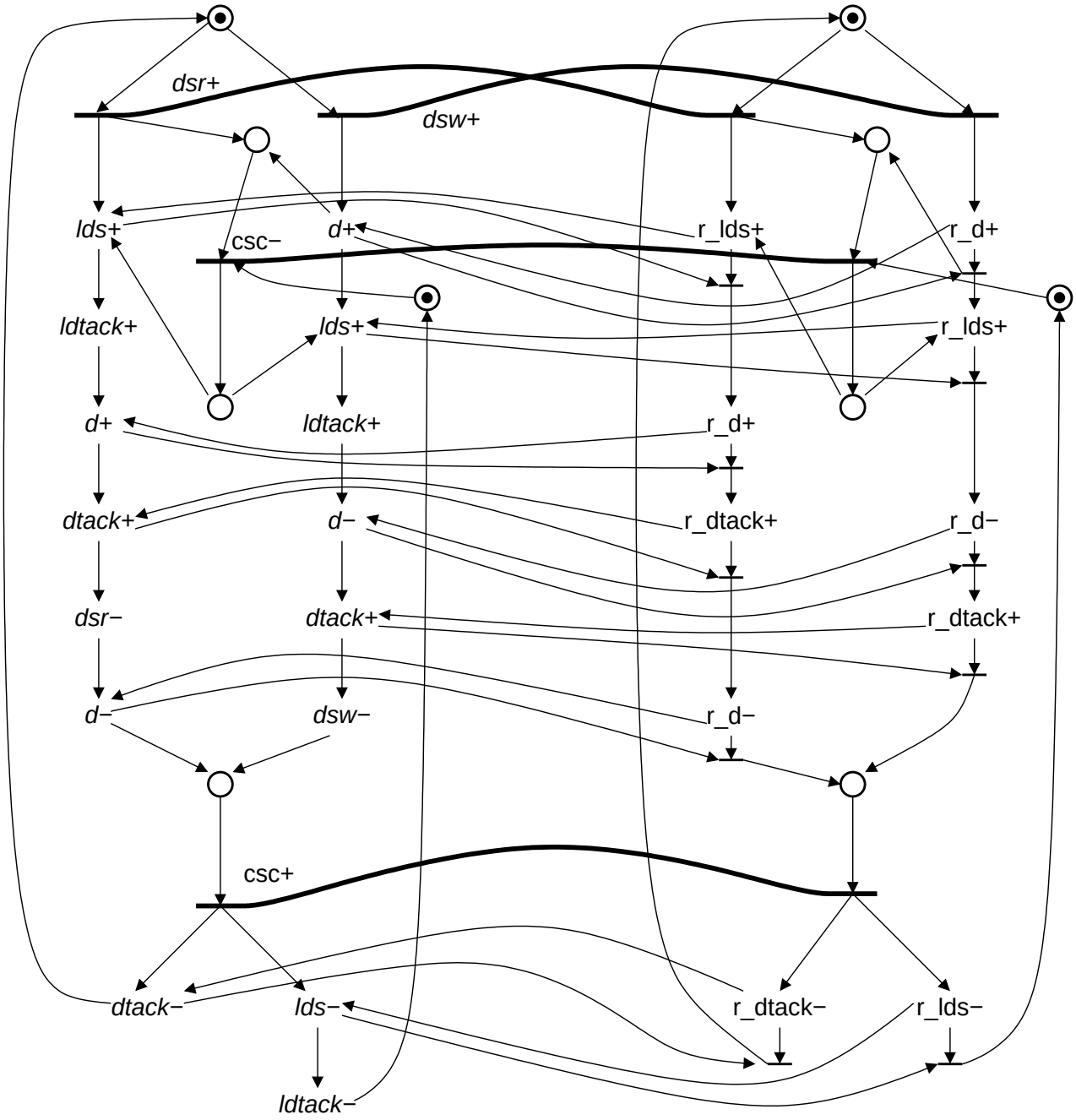
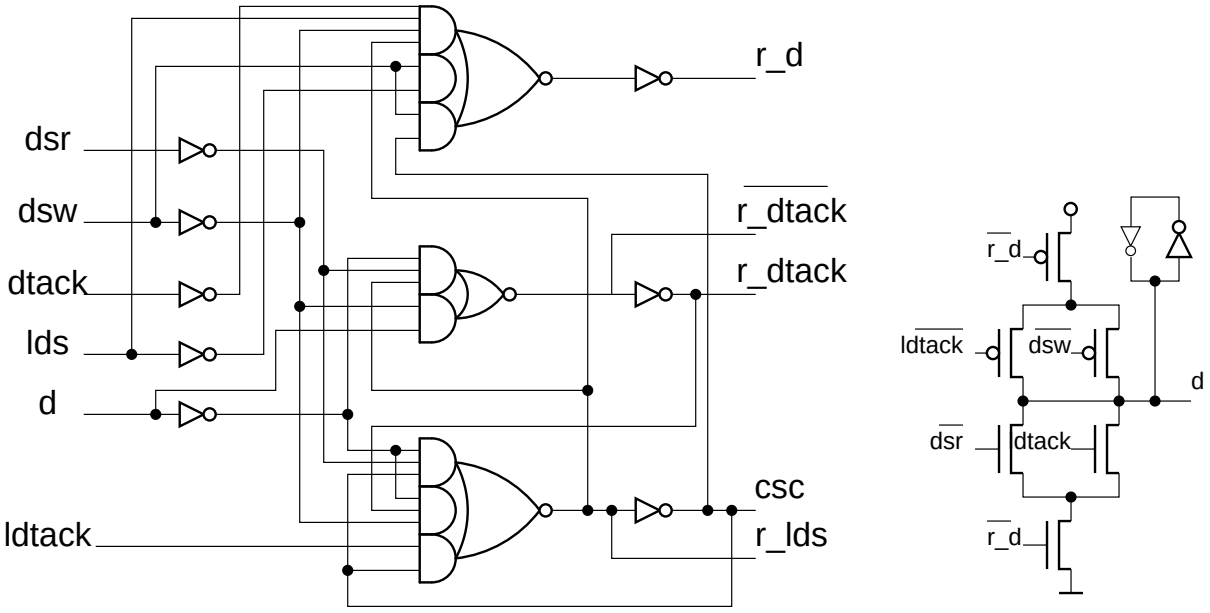



Figure 4: ECS specification for VME bus controller

If one output follows the other, this means that the designer assumed a certain delay between them, possibly predicting the circuit implementation. Our method does not reduce this delay, which can be treated as a positive feature. A different situation is when an output is preceded by a choice identifier, which is always an input. Latency reduction in this case is not supported by the given method, this is the subject of the future work.

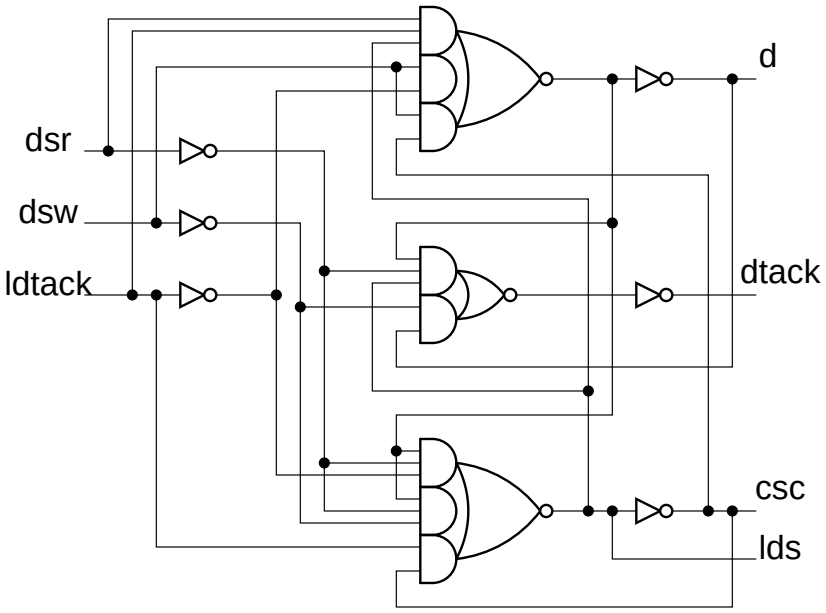
The second design example shown in Figures 7-8 is aimed at emphasising the positive features of the method. It is a modulo four counter defined as the left STG in Figure 7(a). The minimum number of memory elements (CSC internal signals) required to implement a counter is $\log(n)$ rounded to the greater value, where $n$ is the counter modulus. This favourably compares to the low-latency design method based on direct syntax mapping [3], which creates $n$ state separation memory elements for such counters. In the same time it is




(a) ECS logic

(b) Output flip-flop

Figure 5: Low latency implementation of VME bus controller




(a) Generated by Petrify tool

Figure 6: Reference circuit

expected that both methods produce much shorter latency than the direct application of Petrify.

The right part of Figure 7(a) shows insertion of internal signals solving CSC conflicts. It is a standard procedure required by all logic synthesis methods. In this case the signals have been inserted manually in order to make most out of the regular STG structure. An effort was also made to make these signals switch concurrently to the transitions directly preceding the output transitions. This is also a known approach to the

low-latency design. A price to pay for the low-latency of CSC signals is their duplication, so the first bit of memory is represented as $\{a0, a1\}$ signals and the second uses $\{b0, b1\}$. The number of internal signals is now $2\log(n)$, which is still much less than $n$ for large specifications (this specification is scalable for any modulus).

The specification for ECS, generated by the proposed algorithm, is shown in Figure 7(b). Some transient arcs and redundant dummy transitions are removed for clarity. The logic functions implementing output request signals $\{r\_y, r\_z\}$ and internal CSC signals $\{a0, a1, b0, b1\}$ were generated by Petrify tool. The circuit in Figure 8 implements these functions. The circuit is speed independent in the assumption of zero delay in inverters (a traditional assumption for this tool). No effort was applied to optimise this circuit (e.g. by decomposition of complex gates or by adjusting signal polarities). This is to show that the quality of internal signal implementation in our method does not affect the output latency.

The latency of complex gates in this circuit implemented in AMS-CUQ-0.6$\mu$ technology reaches 1.2ns for the AN3332 gate operating under a small load and 0.35ns for the NA4 gate. This is about 7 and 2 inverter delays correspondingly. The output slopes for these gates are about the double of their latency. Regardless of this, the output latency on both $y$ and $z$ outputs is only 0.16ns for the falling transition and 0.19ns for the raising transition, which is similar to a *single inverter*. The slopes at primary outputs are also similar to those of an inverter.

The direct application of Petrify to the STG in Figure 7(b) produces a circuit comparable in its complexity and speed to the ECS logic part of Figure 8. It is defined by the following logic functions:

1. [y] = c (a0 a1 b0 b1 + a0' a1') + y (b1 b0 a0 + a1' a0' + b1' b0' + c);
2. [z] = c (b1' a0 b0' a1 + z) + z a0;
3. [a0] = z' y' a0 + a1' y;
4. [a1] = y' a0 + a1 y + z;
5. [b0] = y (a1 + b1') + y' b0;
6. [b1] = b0 y' + b1 y;

It implements output $y$ by using a gate AN54332, which cannot be implemented as an atomic gate. The decomposition of this gate destroys the property of speed independence, and results in the latency of $y$ being 1.1ns. Output $z$ is implemented as a gate NA522, which also needs a similar non-speed-independent decomposition, resulting in the latency of 0.52ns. It is clear that the low-latency design compares favourably to these figures.

The comparison of the proposed method to the direct syntax mapping of [3] produces very similar figures on the output latency, which is due to the similarity in the output flip-flop implementation. The latter, however, can have a better throughput if fast (transistor-level, non-speed-independent) cells are used to implement context signals. The number of memory elements generated by the proposed method is less due to the near-logarithmic state encoding. The logic synthesis algorithm is more expensive computationally as it constructs the state space, which can be exponential w.r.t. the STG size.

# 5 Concluding remarks

## 5.1 Relation to other methods

In the context of synthesis of asynchronous control circuits, the role of Petri nets and STGs appears to be very important. Their modelling approach offers the maximum possible level of expressiveness in terms of causality, concurrency and choice. One of the hard problems in the synthesis from STG has traditionally

been the decomposition of a complex gate implementation satisfying the condition of speed-independence (hazard-free operation) [13, 12, 5]. Another crucial issue of decomposition is that of performance. To this end, there has been little work in this area. This issue usually has two sub-issues, one being logic optimisation for input/output latency, the other being optimisation for throughput or cycle time, both addressed in [17] through the use of relative timing. The second sub-issue has been also addressed in the burst-mode synthesis context in [1]. In this paper we approach the logic decomposition, clearly separating these two concerns. While the I/O latency is minimised to its extreme (without changing the specification though) and this minimisation is achieved at minimum cost in terms of logic size, the throughput is left unaffected. Furthermore, it is crucial that with our approach we do not compromise speed-independence in order to achieve our goals.

The second problem can be now tackled independently, only involving the implementation of the ECS logic part, where techniques of logic decomposition [13, 12, 5], relative timing [17] and direct mapping [3] can be employed in order to improve the overall performance of the control circuit.

It should of course be noted that the payment for the speed improvement in our method is made in terms of greater computational effort of the logic synthesis tool Petrify, which has to cope with a considerable increase in the state coding space. We believe that a way to partially alleviate this problem could be to apply the ECS decomposition technique selectively, only to those complex gates (in the 'first-cut' solution from Petrify) where context fan-in is excessively large.

### 5.2 Summary

In this paper a method of latency reduction of asynchronous circuits generated by a logic synthesis tool has been described. The method uses a two-level circuit architecture consisting of the explicit context logic and output flip-flops. This architecture separates transition context from trigger signals. The context, explicitly represented by an additional internal signal, is computed in advance, before the trigger signals arrive. This computation is done concurrently to the environment calculating the triggers, which is the key to the latency reduction. The fanin of the output flip-flop is minimal as it uses a single signal as the context, which reduces the latency. The latency of the output transition is defined by the number of its trigger transitions, which makes it predictable at an early stage of the circuit design.

The explicit context logic is generated by logic synthesis tools, which minimise the number of internal variables. This is very important for the designs of counters and the designs from specifications having implicit (hidden) counters.

The disadvantage of the method is that it does not reduce the latency of output transitions directly following input transitions that identify options in non-deterministic choice. This is the subject for the future work.

## References

[1] P. A. Beerel, W. chun Chou, and K. Y. Yun. A heuristic covering technique for optimizing average-case delay in the technology mapping of asynchronous burst-mode circuits. In *Proc. European Design Automation Conference (EURO-DAC)*, Sept. 1996.

[2] K. v. Berkel. *Handshake Circuits: an Asynchronous Architecture for VLSI Programming*, volume 5 of *International Series on Parallel Computation*. Cambridge University Press, 1993.

[3] A. Bystrov and A. Yakovlev. Asynchronous circuit synthesis by direct mapping: Interfacing to environment. Technical Report CS-TR-743 (accepted to ASYNC2002), University of Newcastle upon Tyne, UK, 2001.

[4] T.-A. Chu, C. K. C. Leung, and T. S. Wanuga. A design methodology for concurrent VLSI systems. In *Proc. International Conf. Computer Design (ICCD)*, pages 407–410. IEEE Computer Society Press, 1985.

[5] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, E. Pastor, and A. Yakovlev. Decomposition and technology mapping of speed-independent circuits using Boolean relations. *IEEE Transactions on Computer-Aided Design*, 18(9), Sept. 1999.

[6] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Complete state encoding based on the theory of regions. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*. IEEE Computer Society Press, Mar. 1996.

[7] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. In *XI Conference on Design of Integrated Circuits and Systems*, Barcelona, Nov. 1996.

[8] J. C. Ebergen. *Translating Programs into Delay-Insensitive Circuits*. PhD thesis, Dept. of Math. and C.S., Eindhoven Univ. of Technology, 1987.

[9] R. M. Fuhrer, S. M. Nowick, M. Theobald, N. K. Jha, B. Lin, and L. Plana. Minimalist: An environment for the synthesis, verification and testability of burst-mode asynchronous machines. Technical Report TR CUCS-020-99, Columbia University, NY, July 1999.

[10] L. A. Hollaar. Direct implementation of asynchronous control units. *IEEE Transactions on Computers*, C-31(12):1133–1141, Dec. 1982.

[11] M. Kishinevsky, A. Kondratyev, A. Taubin, and V. Varshavsky. *Concurrent Hardware: The Theory and Practice of Self-Timed Design*. Series in Parallel Computing. John Wiley & Sons, 1994.

[12] A. Kondratyev, J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev. Logic decomposition of speed-independent circuits. *Proceedings of the IEEE*, 87(2):347–362, Feb. 1999.

[13] A. Kondratyev, M. Kishinevsky, and A. Yakovlev. Hazard-free implementation of speed-independent circuits. *IEEE Transactions on Computer-Aided Design*, 17(9):749–771, Sept. 1998.

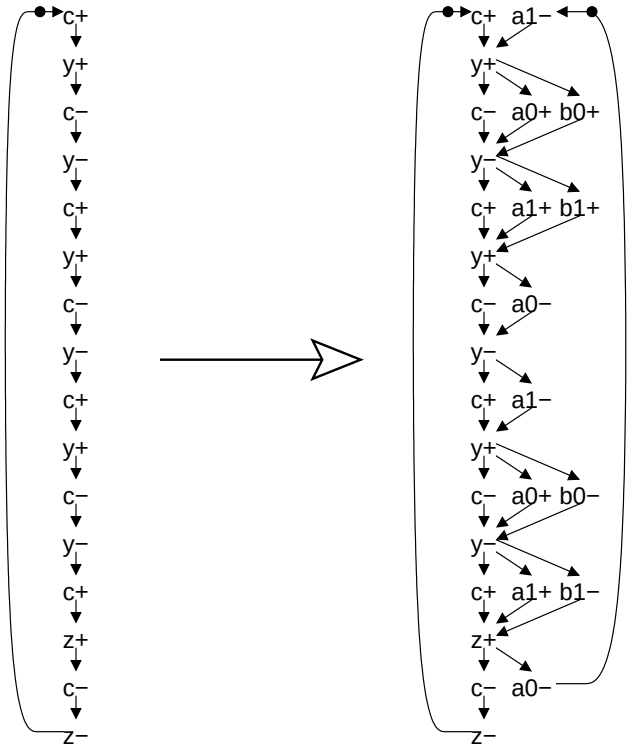
[14] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[15] C. J. Myers, T. G. Rokicki, and T. H.-Y. Meng. Automatic synthesis of gate-level timed circuits with choice. In *Advanced Research in VLSI*, pages 42–58. IEEE Computer Society Press, 1995.

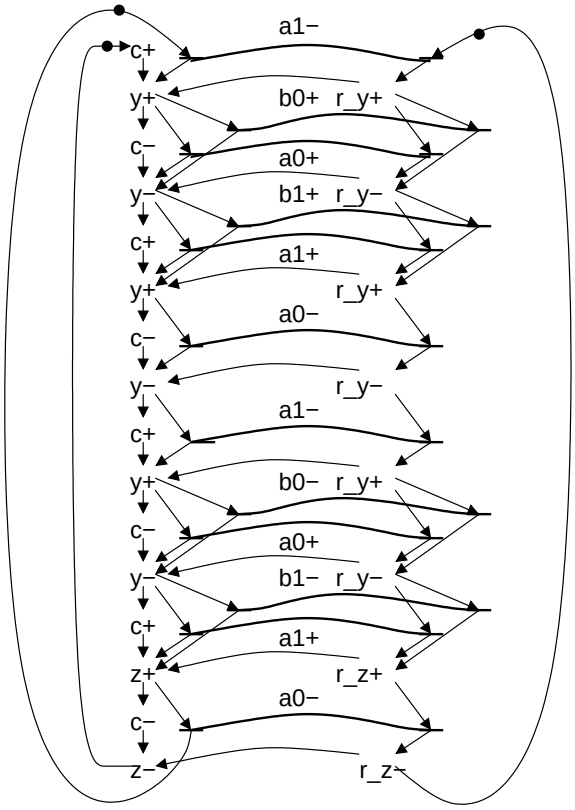
[16] L. Y. Rosenblum and A. V. Yakovlev. Signal graphs: from self-timed to timed ones. In *Proceedings of International Workshop on Timed Petri Nets*, pages 199–207, Torino, Italy, July 1985. IEEE Computer Society Press.

[17] K. Stevens, S. Rotem, S. M. Burns, J. Cortadella, R. Ginosar, M. Kishinevsky, and M. Roncken. CAD directions for high performance asynchronous circuits. In *Proc. ACM/IEEE Design Automation Conference*, pages 116–121, 1999.

[18] V. I. Varshavsky and V. B. Marakhovsky. Hardware support of discrete event coordination. In *Workshop on Discrete Event Systems WODES96*, pages 332–339, Edinburgh, 1996.




(a) Solving CSC conflicts




(b) ECS specification

Figure 7: Counter mod4 specification

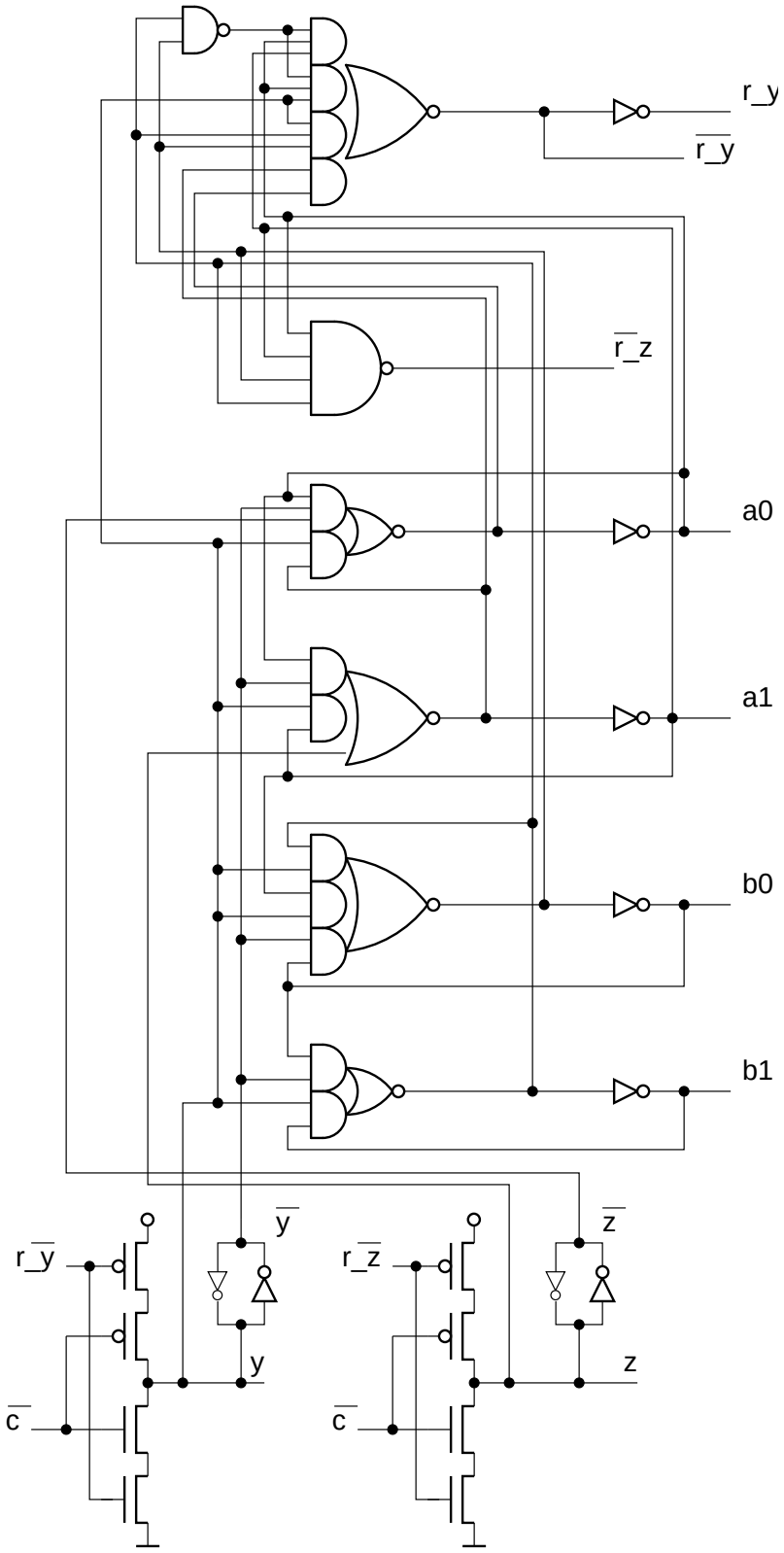



Figure 8: Counter mod4 schematic