

# Towards Synthesis of Monotonic Asynchronous Circuits from Signal Transition Graphs

N. Starodoubtsev \*    S.Bystrov    M. Goncharov    I.Klotchkov  
A. Smirnov  
Institute for Analytical Instrumentation  
Russian Academy of Sciences,  
Rizhskii Prospect 26 St. Petersburg, 198103 Russia  
synth@iai.rssi.ru

## Abstract

*An approach to synthesis of asynchronous speed-independent circuits in monotonic logic gates (e.g. ASIC gate array library IBM SA-12E) is discussed. It is based on the normalcy conditions for STG behavioural specification, which guarantees implementability in monotonic and negative gates. The paper presents techniques for refining the STG specification to an implementable form. It is crucial that in comparison with other speed-independent synthesis methods, e.g. those in the Petrify CAD tool, this approach does not require the use of inverters with negligible delays in order to guarantee the absence of glitches or hazards in the circuits. Experiments with STG benchmarks, involving our new refinement techniques and VHDL simulation, indicate an average reduction of 28% in area and 23% in power consumption against solutions based on non-monotonic logic decomposition. These savings are more noticeable for more complex STG specifications. Such gains are however paid by an average of 6% decrease in circuit speed.*

## 1 Introduction

### 1.1 Preliminary notices

In general, concurrent systems consist of software and hardware parts. While the former is often implemented as a set of sequential processes, the latter has a more natural tendency for concurrency: each gate operates concurrently to other gates unless there

is a connection between them (either immediate or through other gates). Synthesis of asynchronous circuits is often considered as a task of building a circuit for which causal relations between signal transitions are specified by an interpreted Petri net called Signal Transition Graph (STG).

The well-known Petrify synthesis tool [1] is often used for synthesis of control-dominated asynchronous circuits from STGs. It produces logical equations and then performs technology mapping, based on decomposition, into a target gate library. As a result a lot of internal gates may appear in addition to those specified in the given STG. Some causal relations should bind them in order to avoid a hazardous switching as a result of concurrency between newly introduced internal signals.

The above approach has however two drawbacks. Firstly, it does not make use of the inherent property of CMOS gates to implement a monotonic (in particular, negative) Boolean functions. The result of decomposition is a non-monotonic circuit involving inverters connected to gate inputs. Those inverters are often assumed to have zero delay for the circuit to function correctly. Indeed, if they were seen as ordinary gates with delays, there could be situations when such an inverter might switch concurrently with the gate it is connected to. The latter is a dangerous condition as it may lead to glitches.

Secondly, the above approach tries to minimize the number of inserted signals in the (non-monotonic) STG implementation (in order to minimize the set of reachable states). This results in a circuit where some gates can be rather complex, i.e. have a rather large inputs number (fan-in). With a greater variety of types of multi-input gates in the circuit, there is a higher chance that one has more variations in thresholds. Besides, huge fan-in typically results in huge

---

\*This work was supported by EPSRC Visiting Fellowship (project BREACH), GR/M94359

fan-out, which makes such threshold variations dangerous because the hypothesis about isochronic forks in wires (being key for speed-independent circuits that we consider below) might not work somewhere in the circuit [2]. Although technology mapping can decompose large gates into smaller ones it still does not solve the problem because the decomposition does not reduce the fan-out of a decomposed gate. One of the gates always inherits the large fan-out of the decomposed gate as well as the problem of the implementation of the relevant isochronic fork.

We suggest an alternative approach to synthesis, which can eliminate input inverters. As a side effect, such circuits usually have less fork ends than their non-negative counterparts.

As mentioned above, modern asynchronous circuits are usually built of CMOS gates, with ASIC gate array libraries often being used as the basis for their implementation. Such libraries consist of a restricted number of rather simple gates realising the narrow set of logic functions. These functions are typically monotonic and depend on four variables or less (see example in Figure 1 and explanations below).

Whatever design methodology is used, it results in the circuit consisting of such rather primitive building blocks. The success or failure of the design process is eventually the success or failure of our effort to implement the required behaviour of control or interface circuit in a way that is in compliance with the properties inherent to the gates of the circuit. The earlier is the stage of design process at which we take into account the generic properties of the gates, the more is the chance to reach a successful solution.

In this work we try to consider the monotony property of CMOS gates, as early as the first steps of refinement of a given behavioural specification. As the second property (restricted fan-in), we still use the monotonic decomposition of complex gates.

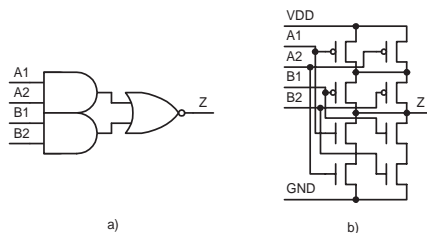


Figure 1: An example of ASIC library gate AOI22: logic symbol (a) and its transistor-level realisation (b)

We will show by experiments that such an approach

leads to a significant reduction in the area and power consumption, as well as improvement of the circuit robustness, compared to non-monotonic solutions. It is important for future sophisticated handshake protocols that the more complex the behaviour is, the greater gain the approach brings.

All the results are based on the *speed-independent (SI) circuit* [3] model, which is known to produce asynchronous circuits free from glitches and hazards due to the fact that each signal transition is acknowledged by another transition.

## 1.2 Gate libraries

The logic structure and transistor-level CMOS circuit of a typical element AOI22 (AND-OR-INVERT, both AND having two inputs) of such libraries are depicted in Figure 1 (p-channel CMOS's being in upper half of the circuit, while n-channel CMOS's being in lower one). This element has four inputs what is a maximum for the library under consideration [4]. For any combination of inputs  $A1, A2, B1, B2$  output  $Z$  is connected to either supply voltage  $Vdd$  or ground  $Gnd$  level (corresponding to logical 1 and 0 respectively) realising Boolean function  $Z$  of four arguments  $A1, A2, B1, B2$ . Notice some properties of the gate being typical for most of gate array libraries:

- There are no input inverters and, hence, the gate realises a *monotonic Boolean function*. Notice that input bubbles in upper half of the transistor-level circuit just depict the inherent property of p-channel CMOS to be opened by value of 0.
- Function  $Z$  is *negative* monotonic because level 0(1) can be provided only by 1(0) on some inputs. In other words, every stage of CMOS circuit has an inherent output inverter.
- As mentioned above, gates in the ASIC library are very simple, e.g., there are no gates more complex than one-stage one depicted in Figure 1 in IBM's SA-12E gate array sub-library [4].
- Such one-stage complex gates can be considered as atomic just like simple gates NOT, NAND and NOR and "gates" AND and NOR depicted in Figure 1a are no more than just logic symbols corresponding to serial and parallel CMOS chains.
- Most of gate array libraries include only one kind of non-monotonic gates: XOR and XNOR, implementing modulo 2 sum and its complement. If other non-monotonic function is to be implemented, then use of two or more atomic gates is

necessary. In this case every gate, even simple inverter, has its own delay not to be ignored. E.g. NOT(INVERT) delay is 66 ps while NAND2 one is 77 ps for equal conditions (see [4]). Hence for SI circuits, each inverter’s transition must be acknowledged by some other gate’s transition.

And finally notice that most of the results of the paper are applicable to wider range of libraries such as standard cell ones (which for standard cell SA-12E sub-library includes as large gate as AO2222) or dynamic gate library including generalised C-gates [5] (without input inverters). The only point being important for our approach is that each atomic gate must implement monotonic Boolean function, what always takes place for CMOS gates with separated p- and n-channel, such as in Figure 1. Nevertheless, we will consider in this paper only the gate array library, in which AOI22, AND4, OR4 are the most complex gates, in order to point out all main stages of synthesis avoiding too complex examples.

### 1.3 Comparison with previous works

Examples of asynchronous controllers that can tolerate variations in timing parameters of components are given in [6]. The class of asynchronous circuits that are insensitive to gate delay variations are Muller’s speed-independent (SI) circuits, introduced in [3]. An extensive research effort has been put in the last decade [7] into methods and algorithms for synthesis of SI circuits. A CAD tool, Petrify [1], can synthesise a SI circuit from its Signal Transition Graph (STG) specification [7] if the latter satisfies the basic implementability conditions [7] and map the solution to a particular ASIC library.

In order to preserve SI-ence during the logic decomposition, Petrify seeks for the newly emerging gate outputs to be acknowledged by other signals, or in the case of complemented signals often assumes the delay of input inverters (“bubbles”) to be equal to zero. This is the main limitation that we want to overcome.

As mentioned above, an additional inverter requires more area, power and causes a signal delay. Note, e.g., the implementation of signal  $Ao$  in Figure 2b, where an inverter  $\overline{Ro}$  is added to input  $Ro$ . The upper bound on the delay of the detached inverter  $\overline{Ro}$  should be restricted, otherwise it may result in a hazardous behaviour in the case of inverter  $\overline{Ro}$  keeping its value of 1 until the transition of  $X+$ .

For comparison, let us consider *monotonic implementation* of the same signal  $Ao$  in Figure 2d without input inverters. This circuit has two advantages compared to that of Figure 2b: it has no hazards under

any delays of the  $Ao$  and  $n0$  gates and it’s area is less than that of its counterpart – 12 cell units versus 18 if implemented in the logic gate array library SA-12E. Negative circuits (composed of gates implementing only negative Boolean functions) attracted attention about three decades ago when it was noticed that basic CMOS gates have an inherent output inverter and thus, implement a negative monotonic Boolean function [8, 9]. Next, the interest to the negative asynchronous circuits arose when it became clear that they have a power consumption advantage over their non-negative counterparts [10, 11, 12].

Negative speed-independent circuits are more robust because inverter delays are not assumed to be negligible, what is more realistic. Synthesis of speed-independent negative circuits was first investigated in [13, 14]. A formal method of behavioural refinement, minimising the number of inserted auxiliary signals was suggested in [13]. However, minimizing the number of signals does not necessarily mean the circuit minimisation.

Representative experimental results for small, middle and large sized negative control-dominated asynchronous circuits with behaviour given by STGs were presented in [15] and [16]. New experimental results obtained by simulation in VHDL environment are represented in this paper.

The approach we use here to simulate in VHDL the circuit with its environment specified by STG was represented in [17]. The use of this approach to obtain the simulation results is briefly described below in the section 4.

### 1.4 Technology mapping and STG refinement

A synthesis procedure is often considered as a black box whose input is a required behaviour and whose output is the realisation of this behaviour by a circuit (Figure 3a). Many modern approaches to the synthesis of asynchronous control-dominated logic circuits use Signal Transition Graph (STG) [7] as a model for specifying the initial circuit behaviour. We use STG model because it is convenient for the transformation of required behaviour to some final implementable form. The required behaviour is often given by an STG that cannot be implemented by a circuit consisting *only* of the gates which implement the STG signals. An STG is not properly encoded (CSC, see [7] and section 2) can serve as example. In such cases the initial specification is to be extended with auxiliary state signals.

It is typical to subdivide synthesis from STG spec-

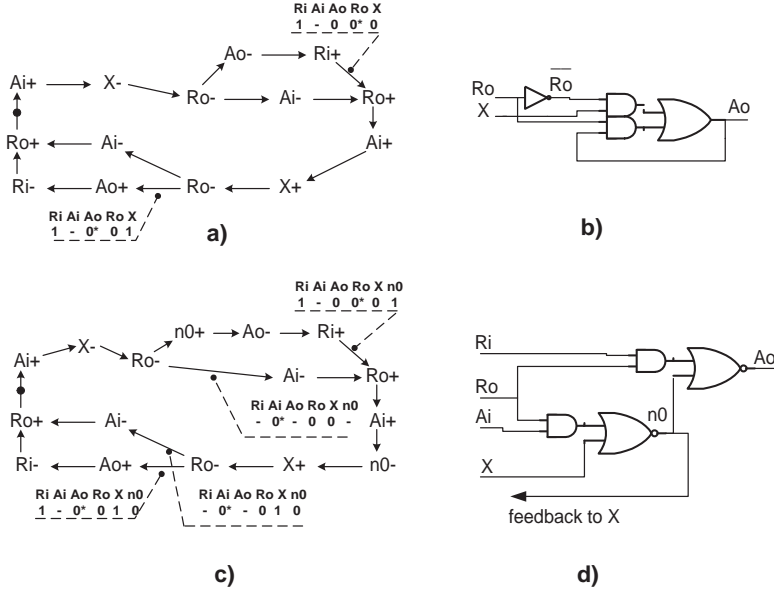


Figure 2: An example of implementations of signal  $Ao$  for Converta circuit: (a) - STG satisfying CSC, (b) - non-monotonic implementation in monotonic library, (c) -  $Ao$ -normal STG, (d) - monotonic implementation

ification into two stages: transformation of a given STG in order to satisfy, e.g., CSC conditions (*refinement* stage) and deriving the Boolean equations from the refined STG. It is known from [7, 14] that the insertion of auxiliary signals is necessary in general, otherwise an equation for some signal may not exist. A similar approach can be applied to synthesis of monotonic circuits. The latter can be easier mapped into real ASIC library. Equations for monotonic gates could be derived from an STG if the latter satisfies certain properties that are stronger than the CSC conditions. An approach to the refinement of an arbitrary STG to the one which has a monotonic circuit solution (called normal STG) is suggested in the section 3.

Thus, in general, we first need to perform an analysis, then STG refinement (in the case of violation of CSC or normalcy properties, introduced in the section 2), and only after that, to derive equations (see Figure 3b). Our approach virtually shifts technology mapping to an earlier stage of synthesis – to the STG refinement. Section 4 presents experimental data and the way they were obtained while section 5 gives conclusions.

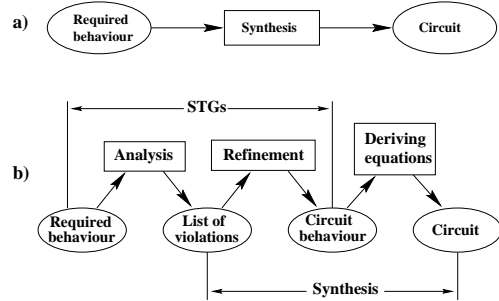


Figure 3: The first (a) and more detailed (b) views on synthesis

## 2 Theoretical background

### 2.1 Basic definitions

STG is a sort of interpreted Petry net in which nodes of the event type are interpreted as signal transitions. An example of an STG specifying a handshake controller **Converta** is given in Figure 2a, where the *initial marking* enables only one transition  $Ai+$ , which has to have all of its input arcs (in STG without choice arcs are associated with *implicit places*) marked and is thus able to fire. Each firing removes one token from every input (implicit) place of the transition and adds

one token to each of its output places. After the firing of  $Ai+$  the net moves to a new marking, from which transitions  $X-$  and  $Ro-$  fire in sequence, which is followed by  $Ao-$  and  $Ai-$  firing concurrently, and so on. Playing the token game, one can generate all reachable markings of the STG, if the latter is *bounded*, i.e. each STG place can only have a bounded number of tokens. (For obvious reasons, we will only consider bounded STGs.) Each marking of an STG can be associated with a single binary code (called *binary state* or simply *state*) of signal values.

For an STG to be implemented by a circuit it should satisfy some basic properties. An STG is *consistent* if all rising and falling transitions alternate for each signal. An STG satisfies the *complete state coding (CSC) condition wrt a signal  $x$*  iff any two *equal binary states* have the same post-behaviour for signal  $x$ , i.e. signal  $x$  is either stable or excited (has an enabled transition) in *both* states. An STG satisfies the *CSC-condition* iff it satisfies CSC wrt all non-input signals. The STG in Figure 2a satisfies both consistency and CSC condition wrt  $Ao$  (cf. states associated with arcs  $(Ri+, Ro+)$  and  $(Ro-, Ao+)$ ). *Speed-independence* (i.e. the acknowledgment of each transition by another one), mentioned above, is another property necessary for an STG to be implemented by a speed-independent circuit. It is implicitly meant throughout the paper.

## 2.2 STG's of monotonic circuits

*Negative circuits* consist of gates that have inverters on their outputs (being inherent to one-stage CMOS gates) and no inverters on their inputs (cf. Figure 2d). An STG of a negative circuit must satisfy some properties that are additional to the ones mentioned above. While the CSC condition concerns only equal states, a more general notion of ordered states is also crucial for the monotonic and in particular for the negative circuits.

**State ordering.** State  $r$  is less or equal to state  $s$  ( $r \leq s$ ) iff for each signal its current value in  $r$  is less or equal to its current value in  $s$ , i.e.  $x(r) \leq x(s)$  for all (including input) signals  $x$ . Two *states*  $r$  and  $s$  are *ordered* or *comparable* if either  $r \leq s$  or  $s \leq r$  takes place.

**Normalcy of STG.** All properties of the STG's of monotonic circuits can be derived from the definition of a monotonic Boolean function  $f$  for *negative* (1) and *positive* (2) cases:

$$r \leq s \Rightarrow f(r) \geq f(s) \quad (1)$$

$$r \leq s \Rightarrow f(r) \leq f(s), \quad (2)$$

where  $r, s$  - ordered reachable states.

An STG is *normal wrt signal  $x$*  iff the *next-state function*  $f_x$  (which gives the implied, i.e. next state, value of signal  $x$  in each state) satisfies one of the properties (1) (*n-normal*) or (2) (*p-normal*), which monotonic gates satisfy by definition. An STG is *normal* if it is normal wrt every non-input signal  $x$ . It is known [14, 15] that monotonic circuit realises a normal STG. Notice that for any two equal states  $r = s$  the equivalence  $f(r) = f(s)$  follows from either properties (1,2) and, hence, an STG normal wrt  $x$  automatically satisfies CSC condition wrt  $x$ . An STG is *coherent wrt  $x$*  if every transition that causes  $x \pm$  has the same (*p-coherent STG*) or opposite (*n-coherent STG*) sign. It was shown in [14, 15] that normal behaviour is coherent.

## 2.3 Sufficient conditions of STG implementability by monotonic circuit

The behaviour of a monotonic circuit can be described by a consistent STG that satisfies CSC, coherency and normalcy. Thus, all of the above properties are necessary conditions of the STG implementability by a monotonic circuit in the sense that if normalcy condition is violated, then a monotonic circuit realising the given STG without auxiliary signals does not exist. Furthermore, normalcy is sufficient for implementability in the sense that for any normal STG there exists a monotonic circuit that realises this STG without auxiliary signal insertion.

## 3 STG Refinement

The normalcy of an STG is shown above to be necessary and sufficient for its implementability by monotonic circuit. If an STG violates normalcy for a pair of ordered states, then the only way to eliminate this conflict would be to insert one or more auxiliary signal transitions, which would make these states unordered. Recall that our final goal is a circuit built of simple library monotonic gates. We subdivide the refinement problem into a set of sub-problems, however, some options of circuit minimisation may be lost after each step. Below we will show our main refinement steps and heuristics using the **converta** example.

Formal synthesis algorithms as well as conditions of their applicability are out of the scope in this paper, which is focused on the STG transformation (refinement) for satisfying normalcy condition. This paper presents ideas underlying the refinement and improvements that can be achieved such a way. We had to use

here manual refinement procedures as long as formal refinement procedures are not developed at the moment.

### 3.1 Elimination of CSC and coherency conflicts

The first step, elimination of CSC violation, is well known (see [7]). The STG in Figure 2a satisfies CSC condition. However it does not satisfy normalcy wrt signal  $Ao$  if the latter is supposed to be negative: ( $1000*0 < 100*01$  and  $f_{Ao}(1000*0) < f_{Ao}(100*01)$ ). Notice that the STG also does not satisfy coherency wrt  $X$  (as  $Ai+ \rightarrow X+$ ) and  $Ao$  (as  $Ro- \rightarrow Ao-$ ) if these two signals are supposed to be negative. As it follows from the coherency definition, a positive transition should be inserted before  $Ao-$  and a negative one before  $X+$ . The insertion of signal  $n0$  eliminates both these conflicts (see STG in Figure 2c) and concomitantly makes STG n-normal wrt  $Ao$ , as states 100001 and 100010 become unordered (see Figure 2c).

### 3.2 Elimination of normalcy conflicts

Thus, the STG in Figure 2c became normal wrt  $Ao$ . However it is not normal wrt  $X$  yet, as one can see comparing states associated with two arcs ( $Ro-, Ai-$ ) in Figure 2c :  $100000 < 100010$  and  $f_x(100000) < f_x(100010)$ , which contradicts the negativity of function  $X$ . There are some options to insert auxiliary signals (both in serial and in parallel) in order to make states 100000 and 100010 unordered, as well as there were some options of eliminating CSC conflicts. The simplest way is insertion of two pairs of signals transitions in series, however it is not always acceptable because of increasing latency. In our case let us insert two pairs of transitions ( $n1-, n2+$ ) and ( $n1+, n2-$ ) as shown in Figure 4a, which makes unordered all the states associated with one part of the STG between these two pairs wrt the states associated with the rest of the STG. After that the subsequent insertion of signal transitions  $n3\pm$  is necessary for preserving STG coherency.

Having a look at Figure 4a, one can see that only signals  $Ao$  and  $Ro$  are required outputs. All the other signals were inserted for getting proper realisation:  $X$  - for CSC,  $n0$  and  $n3$  - for coherency,  $n1$  and  $n2$  - for normalcy conflict elimination. We could use other signal insertions, in parallel as well as in serial, what would provide other solutions.

**Flip-flop insertion.** The insertion of complementary signals' pairs ( $y\pm, z\mp$ ) in an STG implicitly adds to the circuit a pair of negative gates, in many cases

realising a flip-flop. Such a flip-flop, when properly inserted, makes one part of the STG unordered with another. This methodology gives the designer a lot of opportunities for potential improvement of a particular circuit. Using this technique the designer effectively replaces the circuit design by more detailed design of its behaviour (STG), what is much easier as a rule. Flip-flop insertion is a heuristics, which often brings good results.

### 3.3 Decomposition for technology mapping

If an equation derived from the refined STG cannot be implemented by an atomic library gate, as e.g. signal  $X = \overline{a_i n_1 + n_0 + n_3}$  in our example, some kind of decomposition of a monotonic equation similar to the one from [7] could be a solution. Only monotonic decomposition is acceptable in our case though. A simple and rather effective way of decomposing logic is a *chained decomposition*, where the transitions of atomic gates realising the parts of the original complex gate have always the same fixed order. E.g., one can see that  $s0-$  causes  $X-$ ,  $s0+$  causes  $X+$  according to the STG and the circuit in Figure 4 and, hence,  $s0\pm$  is always acknowledged by  $X\pm$ . Recall however that decomposition preserves the initial fan-out of decomposed gates along with the problems of isochronic forks (see section 1.1).

### 3.4 A complex example: VME-bus read-write controller

The required behaviour of the controller is depicted in Figure 5. This example is not favourable for negative gate implementation because of few long transition chains with the same signs, such as  $d sr+ \rightarrow l ds+ \rightarrow l dtack+ \rightarrow d+ \rightarrow dtack+$ . There are also such incoherent causalities as  $d+ \rightarrow dtack+$  in the upper part of STG and  $d- \rightarrow dtack+$  in the lower one. Thus, it will require a lot of auxiliary signals in order to refine the STG.

The STG refinement was made separately for read (upper half of STG in Figure 5) and write (lower half) cycles. Positive signal  $ps1$  and negative signals  $n6$  and  $n4$  were inserted in read cycle. Positive signals  $ps3, ps0$  and  $ps2$  and negative signals  $n3, n2, n0$  and  $n5$  were inserted in write cycle. Negative signal  $n1$  was inserted in both cycles.

The resulting monotonic circuit derived from the STG Figure 6 is depicted in Figure 7. It consists of 14 gates. For comparison notice that the circuit derived from non-refined STG from Figure 5 consists of

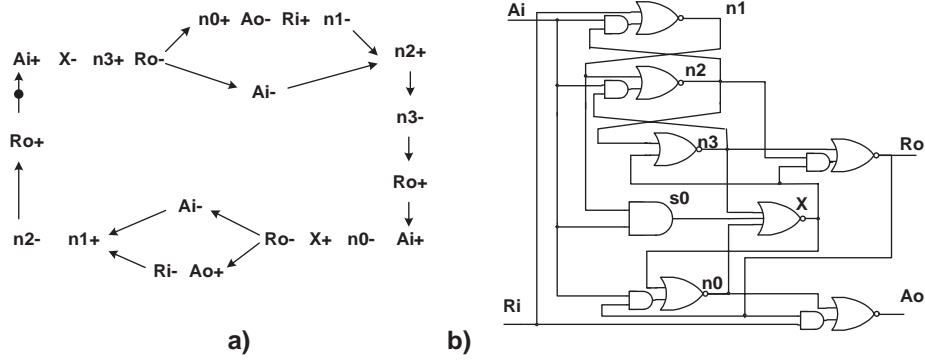


Figure 4: A monotonic fully speed-independent realisation of Converta circuit: (a) - a normal STG, (b) - the circuit

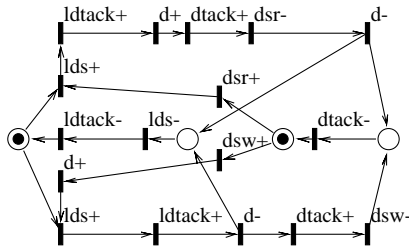


Figure 5: The required VME-bus controller behaviour

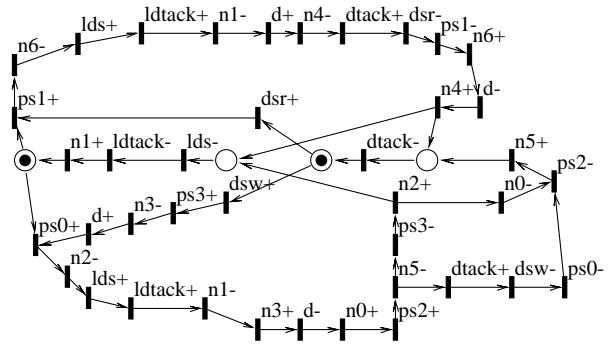


Figure 6: A refined VME-bus controller behaviour

36 gates (14 of them are inverters of restricted delay) for the same gate array library SA-12E. Both solutions were obtained by Petrify tool version 4.0 for our refined and non-refined STG's.

### 3.5 Circuit delays

A proper auxiliary signal insertion results both in simplification of the circuit's gate and in increasing latency (One can see, e.g., in Figure 4 three auxiliary signal transitions  $n1 - n2 + n3 -$  are inserted between  $Ri+$  and  $Ro+$ ). The former trends to reduce circuit delay while the latter trends to increase it.

The problem of synthesis of fast circuits is out of scope in this paper. In experiments described in the section 4 a number of signal transition insertions were made for each STG in order to refine it to a normal one. Among all variety of signal insertions the first one, which brought not bulky solution, was chosen for realisation. Actually we used intuitive criterion of using of a minimal number of auxiliary signals inserted in such a way that each equation was monotonic and

rather simple. If it was not simple, then optionally we either tried another signal insertion or decomposed complex Boolean function. As it is shown in the section 4 such a strategy of signal insertion results in slowing down the circuit speed by 6% in average if compared with non-negative counterpart.

If we had the aim to minimize circuit delay, then quite the different way of signal insertion would be applied (mainly in parallel but not in serial) and quite the different circuits would be obtained. However it is the subject of the future experiments. Meanwhile notice that increasing of the latency (estimated as amount of switching gates) is often inevitable if very simple gate library is used, however it does not necessarily result in the circuit slowing down. E.g., the flip-flop ( $n1, n2$ ) used in Figure 4 for parallel branches synchronisation has latency 2, nevertheless, it works faster than 2-input C-element (often used for branches

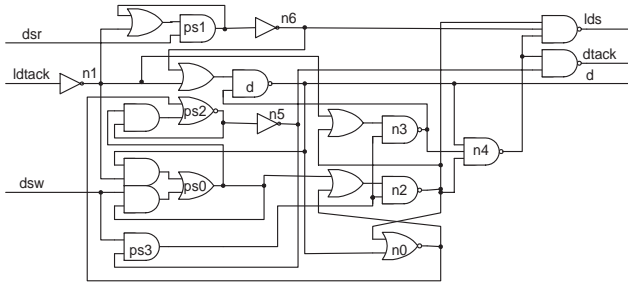


Figure 7: The VME-bus controller monotonic realisation

synchronisation) implemented by AO222 gate having latency 1.

## 4 Experimental results

The monotonic circuits obtained with use of the above methodology were compared against the non-monotonic ones implementing the same behavior.

**Description of experiment.** In order to make sure that monotonic circuits have advantages over their non-monotonic counterparts we have made experiments with a reasonable variety of benchmarks. To mitigate side effects, we had to provide the conditions for experiments to be as equal as possible for both types of circuits: (i) the speed-independent model of the circuit, (ii) the STG specification of behaviour to be implemented, (iii) the same gate array library SA-12E [4] to map the circuits into, (iv) the use of the same tools on all the stages wherever possible. For example, it was natural for us to use Petrify in our experiment [1], not only for obtaining non-monotonic solutions, but also for deriving monotonic Boolean functions from the STGs obtained by our own refinement.

There were, however, two differences: different assumptions for inverter delays (which are always arbitrary for monotonic circuits) and use of quite different procedures of behavioural refinement. STG's implementable by monotonic circuits were mostly obtained manually employing our synthesis tool TaxoSynthesis whenever possible. Table 4 shows results of the Petrify's implementation, after technology mapping (with -tm option), and our monotonic gate implementation, obtained for a set of asynchronous benchmarks. The numbers in Table 4 show: area, energy (characterised by the values of recharged capacitance) and the circuit delay per operational cycle, all obtained for

the implementations in the SA-12E gate array library. These experimental data were obtained by simulating a VHDL code produced by our TaxoSynthesis tool automatically. For all of the implementations of each particular benchmark, both energy and speed were measured for the same input signals' trace provided by STG simulation as it was described in [17].

Experimental results presented in Table 4 were obtained by VHDL simulation. The problem of setting circuits, without reset signals, into appropriate initial states and avoiding overriding the initial values by uninitialised ones ('U') was resolved by using values 'Z', 'H' and 'L' for initialisation instead of 'U', '1' and '0' respectively.

The simulation was performed in two modes:

1. Power modelling mode. This mode reflects a static operation of the circuit, with the environment delay being much greater than any circuit delay.
2. Delay modelling mode. In this mode environment delays were much less than any circuit delays.

The delay and power are measured per operation cycle, where one such cycle is the shortest sequence of events between two equal states of the circuit. The cycle length may depend on the presence of choice in the initial specification. In such cases, different kinds of cycles were taken with the same weight, i.e. all choice probabilities are considered equal. The energy was measured by calculating recharged internal and gate capacitance for each gate.

For all three characteristics of the implementation a ratio is given in the third sub-columns, to make it easy to compare different implementations. In all the cases, ratios below 1 show the relative advantage of the monotonic circuit over the generic one, while ratios above 1 demonstrate the opposite.

Finally, the improvements achieved in area, power consumption and speed estimated as  $1/ratio$  from Table 4, for monotonic circuits vs. their non-monotonic counterpart, are shown against circuit complexity in Figure 8. The graph in Figure 8 is a linear approximation of the data from Table 4. One can see that the more complex is the circuit, the greater gain can be achieved. All of the obtained monotonic circuits are speed-independent and free of glitches and hazards under arbitrary gate (including inverter) delays while the non-negative counterpart may apply a zero delay assumption to some inverters.

The experimental results might look paradoxical: dangling inverters were eliminated at no cost. We see the following reason behind it: refinement as a

Table 1: Experimental data: Area, Energy and Delay for monotonic and non-monotonic implementations and corresponding Ratios

Example	Area			Energy			Delay		
	Mono	Non-mono	Ratio	Mono	Non-mono	Ratio	Mono	Non-mono	Ratio
chu133	51	54	0.94	66	60	1.10	1855	1130	1.64
chu150	54	57	0.95	76	68	1.11	1615	1230	1.31
converta	51	60	0.85	75	93	0.80	1843	1950	0.94
mp-forward-pkt	51	72	0.71	43	77	0.56	0907	0790	1.14
nak-pa	69	75	0.92	63	85	0.74	1428	1290	1.11
nowick	42	78	0.54	60	123	0.49	1380	2190	0.63
ram-read-sbuf	63	87	0.72	80	77	1.04	1860	1466	1.27
rpdft	60	78	0.76	66	73	0.90	2325	1730	1.34
sbuf-ram-write	78	111	0.70	84	114	0.73	2575	2087	1.23
sbuf-read-ctl	57	54	1.06	58	53	1.09	1799	1050	1.71
sbuf-send-pkt2	60	78	0.77	41	68	0.60	3765	4767	0.79
seq_mix	63	84	0.75	87	95	0.92	2856	2004	1.42
seq4	42	63	0.67	48	63	0.76	1286	1375	0.93
trimos-send	162	183	0.88	226	210	1.08	3310	3333	0.99
vbe10b	174	282	0.61	319	327	0.98	4250	4625	0.91
vbe6a	120	222	0.54	154	287	0.54	4120	3830	1.08
wrdatab	132	219	0.55	187	258	0.72	3182	3383	0.94
vme-read-write	99	237	0.41	149	431	0.34	3086	6276	0.49
geometric mean			0.72			0.77			1.06

rule increases the number of STG signals, what tends to increase the circuit complexity, however, on the other hand it eliminates dangling inverters and simplify the gates the circuit is built of. The resulting effect can be either positive or negative depending on the properties of the STG to be implemented. Why it is more often positive than negative? We believe that it is because neither our approach nor Petrify’s can produce minimal solutions today for complex STG’s, while proper signal insertion contributes more to the equations complexity reduction than to the increase in the number of extra signals. There also takes place the manual intervention side effect.

## 5 Conclusion

An approach to synthesis of asynchronous hardware implementation of STGs built of simple gates, with a restricted fan-in, realising monotonic Boolean functions has been considered. The approach is based on the use of some generic properties of STGs that describe monotonic circuits’ behaviour. It uses heuristics for refining an arbitrary STG to a form implementable by a speed-independent circuit in a given gate array library, e.g. SA-12E. No gates or inverters are

supposed to have restricted or zero delays, what contributes to the circuit robustness.

The experimental comparison of negative circuit implementations and those having input inverters indicated that the first show an average reduction of 28% in circuit complexity and 23% in power consumption. What is important, the more complex the circuit is, the greater improvement it shows on average. This makes such an approach, based on STG refinement, very promising for the realisation of complex asynchronous circuits with high degree of concurrency.

However, two directions for further improvement of such circuits were not considered here: (i) refinement using decomposition, which could help avoid too large fan-out for some elements and, hence, mitigate the problem of isochronic fork, and (ii) refinement aimed at increasing circuit speed. They are the subjects for further investigation.

The obtained experimental results have shown that both Petrify and our approach have a potential for improving the synthesised circuits.

## Acknowledgement

The authors are grateful to A. Yakovlev for very helpful discussion and support of this work as well as for

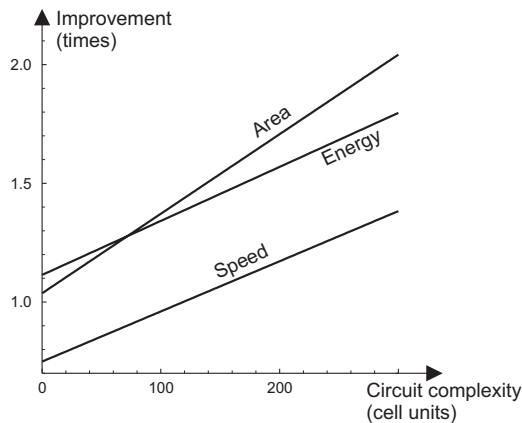


Figure 8: Area, Power and Speed improvements (1/Ratios) vs. non-monotonic circuit complexity

many valuable comments.

## References

- [1] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno and A. Yakovlev. Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers, *IEICE Trans. Inf. and Syst.*, Vol. E80-D, No.3, March 1997, pp. 315-325.
- [2] Kees van Berkel. Beware the isochronic fork. *Integration, the VLSI journal*, 13(2): 103-128, June, 1992.
- [3] D. E. Muller and W. S. Bartky. A theory of asynchronous circuits. In *Proceedings of an International Symposium on the Theory of Switching*, pp. 204-243. Harvard University Press, April 1959.
- [4] <http://www.chips.ibm.com/techlib/products/acics/databooks.html>
- [5] S.B. Furber and J. Liu. Dynamic logic in four-phase micropipelines. *Proc. of the Second Int. Symp. on Advanced Research in Asynchronous Circuits and Systems (ASYNC'96)* March, 1996 Aizu-Wakamatsu, Japan, pp.11-16.
- [6] M. Kishinevsky, J. Cortadella, A. Kondratyev and L. Lavagno. Asynchronous interface specification, analysis and synthesis, *Proc. DAC'98*, pp. 2-7.
- [7] A. Kondratyev, J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev. Logic Decomposition of Speed-Independent Circuits, In *Proceedings of the IEEE/*, Vol.87, No.2, February 1999, pp. 347-362.
- [8] G.Mago, "Monotone function in sequential circuits", *IEEE Trans. on Computers*, Vol. C-22. No. 10, October 1973, pp.928 - 933.
- [9] T.Ibaraki, S.Muroga. "Synthesis network with a minimal number of negative gates", *IEEE Trans. on Computers*, Vol. C-20. No. 1, January 1971
- [10] C.Piguet. Logic synthesis of race-free asynchronous sequential circuits. *IEEE JSSC*, vol.26, No 3, March 1991. pp. 371-380.
- [11] C.Piguet. Synthesis of Asynchronous CMOS Circuits with Negative Gates. *Journal of Solid State Devices and Circuits*, vol.5, No.2, July 1997.
- [12] C.Piguet. Design of Speed-Independent CMOS Cells from Signal Transition Graphs. *PATMOS'98*. Oct.1998, Copenhagen, pp.357-366.
- [13] N.A.Starodoubtsev. Autonomous Antitonic Sequential Circuits. In: *Soviet Journal of Computer and System Science (USA)*, English translation of *Izvestiya Akademii Nauk SSSR. Technicheskaya Kibernetika (USSR)*, 1981, No 4 (Part I. Definitions and Interpretation), No.5 (Part II. Cyclograms and their Properties) and No.6 (Part III. Minimisation).
- [14] N.A.Starodoubtsev. Asynchronous processes and antitonic control circuits, In: *Soviet Journal of Computer and System Science (USA)*, English translation of *Izvestiya Akademii Nauk SSSR. Technicheskaya Kibernetika (USSR)*, 1985, vol.23, No.2, pp.1 12-119 (Part I. Description Language), No.6, pp.81-87 (Part II. Basic properties), 1986, Vol.24, No.2, pp.44-51 (part III. Realisation).
- [15] M.Goncharov, I.Klotchkov, E.Klypkin, A.Smirnov and N.Starodoubtsev. STG refinement for synthesis of negative gates' circuits. *Handouts of the Acid-WG Workshop*, Univ. of Newcastle upon Tyne, Tech. report No.670, April, 1999.
- [16] N.Starodoubtsev, M.Goncharov, I.Klotchkov and A.Smirnov. Synthesis of asynchronous interface circuits by STG refinement. *Proc. AINT'00 (Asynchronous Interfaces)* Delft, the Netherland, July 2000, pp. 65-73.
- [17] M.Goncharov, I.Klotchkov, A.Smirnov and N.Starodoubtsev. Timing extension of STG model and a method to simulate timed STG behaviour in VHDL environment. *Proc of International Conference on Application Concurrency to System Design (IC ACSD)* 1998, Fukushima, Japan, pp.120-129.