

Synthesis of asynchronous interface circuits by STG refinement

N. Starodoubtsev, IEEE member M. Goncharov I. Klotchkov
A. Smirnov
Institute for Analytical Instrumentation
Russian Academy of Sciences, St. Petersburg, Russia

Abstract

An approach to the synthesis of asynchronous speed-independent interface circuits composed of very simple gates, typical for the ASIC gate array libraries such as IBM's SA-12E, is considered. Basic properties of signal transition graphs (STG) implementable by monotonic gates and heuristics for arbitrary STG refinement to implementable form are shown by example. No gates or inverters are supposed to have a negligible delay in order to guarantee absence of glitches or hazards in the circuits. Experimental results indicate an average reduction in circuit area (24%) and power consumption (18%) against solutions based on non-monotonic decomposition made by the Petrify tool, with these reductions being greater for more complex STG specifications.

1 Introduction

Modern asynchronous interfaces are usually realised as CMOS circuits. ASIC gate array libraries are often the basis for their implementation. Such libraries consist of a restricted number of rather simple gates realising narrow set of logic functions often being monotonic one and depending of 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 an 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 take into account the monotonic properties of CMOS gates, as early as the first steps of refinement of a given behavioural specification. As the second property (restricted fan-in), we suggest a way to the monotonic decomposition of complex gate.

We will show by experiments that such an approach leads to a significant reduction in area and power consumption, as well as improvement of 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* 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.

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). For any combination of inputs $A1, A2, B1, B2$ output Z is connected to either

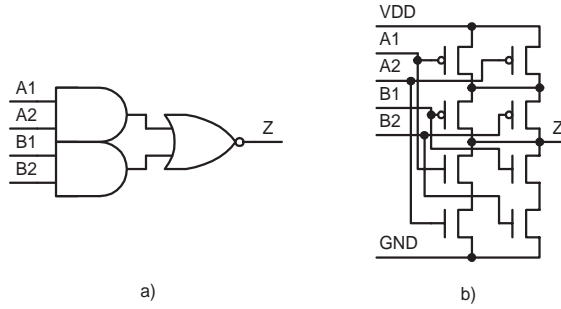


Figure 1: An example of ASIC library gate AOI22: logic symbol (a) and its transistor-level realisation (b)

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 is not gate more complex than one-stage one depicted in Figure 1 in IBM's SA-12E gate array sub-library [1].
- Such one-stage complex gate just as well as primitive gates NOT, NAND and NOR can be considered as atomic one and "gates" AND and NOR depicted in Figure 1a are no more than just logic symbols corresponding to serial and parallel CMOS chains.
- As a rule for each AOI nm gate ($0 \leq m \leq n \leq 2$), an ASIC library includes also AO nm gate having second stage - an output inverter and, hence, realising monotonic *positive* function. The latest takes more area, power and is slower than negative one.
- 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 [1]). 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 [2] (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.

Comparison with previous works. Examples of asynchronous interface controllers that can tolerate variations in timing parameters of components are given in [3]. The class of asynchronous circuits that are insensitive to gate delay variations are Muller's speed-independent

(SI) circuits, introduced in [4]. An extensive research effort has been put in the last decade [5] into methods and algorithms for synthesis of SI circuits. A software tool, Petrify [6], can synthesise a SI circuit from its Signal Transition Graph (STG) specification [5] if the latter satisfies the basic implementability conditions [5] and map the solution to a particular ASIC library.

In order to preserve SI-ence after 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 a 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 transition $X+$.

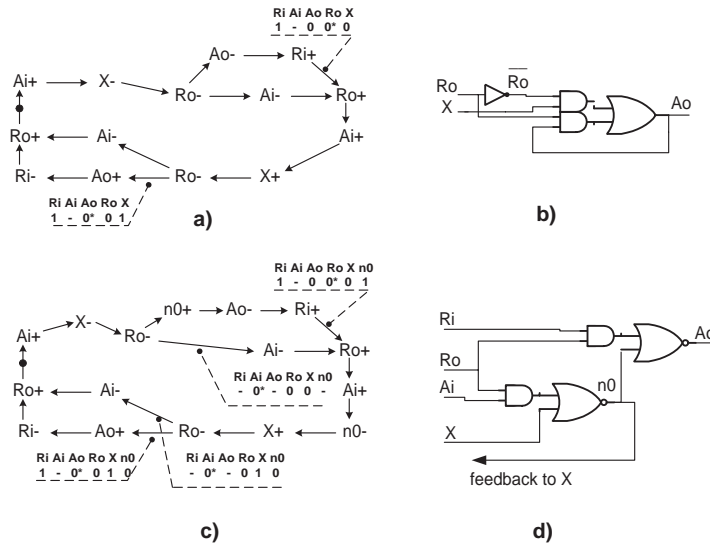


Figure 2: An example of implementations of signal Ao for Converta circuit:

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 negative monotonic Boolean function [7, 8]. 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 [9, 10, 11].

Negative speed-independent circuits are more robust because inverter delays are not assumed to be negligible, which is more realistic. Synthesis of speed-independent negative circuits was first investigated in [12, 13]. A formal method of behavioural refinement, minimising the number of inserted auxiliary signals was suggested in [12]. However, minimising the number of signals does not necessarily mean circuit minimisation. First experimental results for small and middle sized negative control-dominated asynchronous circuits with behaviour given by STGs were presented in [14].

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) [5] 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. For example, a well-known property of complete state encoding (CSC condition, see [5] and section 2) requires the insertion of auxiliary state signals if this property is violated.

It is typical to subdivide synthesis from STG specification 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 [5, 13] 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. 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 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 STG refinement. Section 4 presents experimental data and section 5 gives conclusions.

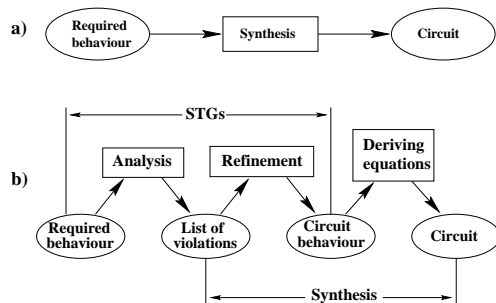


Figure 3: The first (a) and more detailed (b) views on synthesis

2 Theoretical background

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* if 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.

STGs 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 monotonic and in particular for 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 STGs of monotonic circuits can be derived from the definition of a monotonic Boolean function f for *negative* (1) and *positive* (2) cases:

$$r \leq s \Rightarrow f(r) \geq f(s) \tag{1}$$

$$r \leq s \Rightarrow f(r) \leq f(s), \tag{2}$$

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) or (2), which monotonic gates satisfy by definition. An STG is *normal* if it is normal wrt every non-input signal x . It is known [13, 14] that monotonic circuit realises a normal STG. It is shown in [14] that an STG that is normal wrt x automatically satisfies the 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 [13, 14] that normal behaviour is coherent.

Sufficient conditions of STG implementability by monotonic circuit. The behaviour of a monotonic circuit can be described by a consistent STG that satisfies CSC, coherence and normalcy. Thus, all of the above properties are necessary conditions of 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. To be short we will only show our main refinement steps and heuristics using the **converta** example.

Elimination of CSC and coherency conflicts. The first step, elimination of CSC violation, is well known (see [5]). 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 coherency definition, some positive transition should be inserted before $Ao-$ and some 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 becomes unordered (see Figure 2c).

Elimination of normalcy conflict. 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.

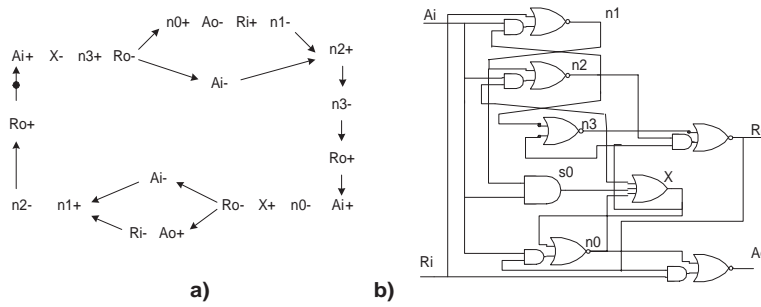


Figure 4: An example of implementations of signal Ao for Converta circuit

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 one. This methodology gives the designer a lot of opportunities for potential improvement of a particular circuit. Using this technique the designer effectively replaces the creation of a circuit by the creation of its STG, which is much easier as a rule. Flip-flop insertion is a heuristics which often brings good results.

Decomposition for technology mapping. In the case when an equation derived from the refined STG cannot be implemented by an atomic library gate, as e.g. signal $X = a_i n_1 + n_0 + n_3$ in our example, some kind of decomposition of a monotonic equation similar to the one from [5] 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*, for which case transitions of atomic gates realising equations 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 $X\pm$.

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 $d+ \rightarrow dtack+$ and $d- \rightarrow dtack+$. 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 STG depicted in Figure 7 consists of 14 gates.

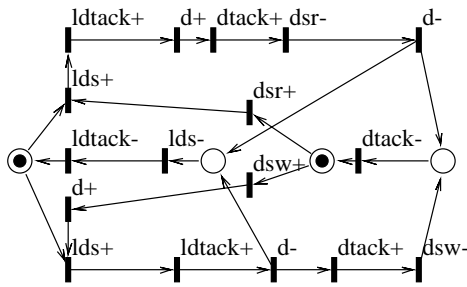


Figure 5: The required VME-bus controller behaviour

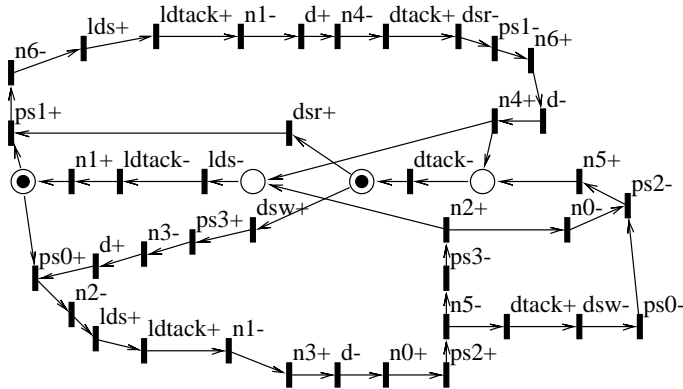


Figure 6: A refined VME-bus controller behaviour

For comparison notice that the circuit derived from non-refined STG from Figure 5 consists of 28 gates (8 of them are inverters of restricted delay) for the same gate array library SA-12E. Both solutions were obtained by Petrify tool.

Circuit delays. A proper auxiliary signal insertion results both in simplification of circuit's gate and in increasing latency (One can see, e.g., in Figure 4 three auxiliary signal transitions $n1 - n2 + n3 -$ 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 section 4 a number of signal transition insertion was made for each STG in order to refine it to 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 minimal number of auxiliary signals inserted 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.

If we had the aim to minimise 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 latency (estimated as amount of switching gates) is often inevitable if very simple gate library is used, however it does not necessarily result in 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 synchronisation) implemented by AO222 gate

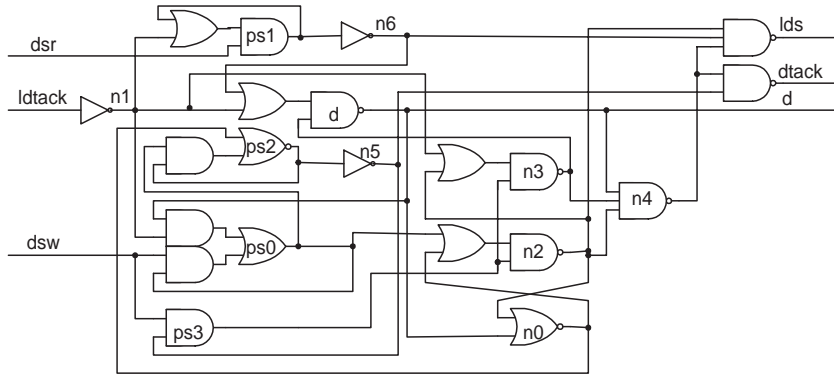


Figure 7: The VME-bus controller realisation

having latency 1.

4 Experimental results

The above methodology of synthesis was used for the experimental comparison of monotonic circuits to non-monotonic ones.

Description of experiment. In order to prove that monotonic circuits have advantages over their non-monotonic counterparts we have made experiments with a reasonable variety of benchmarks. To avoid side effects, we had to provide the conditions for experiments to be as equal as possible for both types of circuits: the speed-independent model of the circuit, the STG specification of behaviour to be implemented, the same gate array library SA-12E [1] to map the circuits into, the use of the same tools on all the stages wherever possible. So, it was natural for us to use Petrify in our experiment [6].

There were, however, two differences: different requirements to inverter delays (which are arbitrary for our circuits) and use of quite different procedures of behavioural refinement, for which our synthesis tool TaxoSynthesis was employed. The table below shows results of the Petrify (with -tm option) and our monotonic gate implementation, obtained for a set of asynchronous benchmarks. The numbers in the table are energy (characterised by the values of recharged capacitance) and area for the implementations in the SA-12E gate array library. For all of implementations of every particular benchmark energy has been measured for the same input signals' trace.

For both 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 show the opposite.

Finally, the improvements achieved in area and power consumption estimated as $1/ratio$ from the table, for monotonic circuits vs. their non-monotonic counterpart, are shown against circuit complexity in Figure 8. Graph in Figure 8 is linear approximation of the data from table 1. 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.

Example	Area			Energy		
	Monotonic	Non-mono	ratio	Monotonic	Non-mono	ratio
chu133	51	54	0.94	76	71	1.07
chu150	54	57	0.95	80	74	1.09
converta	57	75	0.76	97	138	0.70
mp-forward-pkt	51	72	0.71	63	72	0.88
nak-pa	69	75	0.92	91	94	0.96
nowick	42	78	0.54	71	105	0.68
ram-read-sbuf	63	93	0.68	92	111	0.83
rcv-setup	45	45	1.00	72	70	1.02
rpdft	60	84	0.71	109	172	0.63
sbuf-ram-write	78	117	0.67	108	142	0.76
sbuf-read-ctl	57	54	1.06	71	69	1.03
sbuf-send-pkt2	60	78	0.77	103	124	0.82
seq_mix	63	84	0.75	111	121	0.91
seq4	42	63	0.67	59	77	0.77
trimos-send	174	189	0.92	251	241	1.04
vbe10b	174	270	0.64	250	330	0.76
vbe6a	132	222	0.59	191	270	0.71
wrdatab	132	240	0.55	218	332	0.66
vme-read-write	99	168	0.59	149	232	0.64
middle square			0.76			0.82

5 Conclusion

An approach to synthesis of asynchronous interface circuits that are 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, which guarantee absence of glitches or hazards in such circuits.

The experimental comparison of negative circuit implementations and those having input inverters indicated that the first show an average 24% reduction in circuit complexity and 18% in power consumption. What is important, the more complex the circuit is, the greater improvement it shows in average. This makes such an approach, based on STG refinement, very promising for the realisation of complex asynchronous interfaces.

Acknowledgement. This work was supported by EPSRC Visiting Fellowship (project BREACH), GR/M94359. The authors are also grateful to Dr. A. Yakovlev for very helpful discussion and support of this work as well as for many valuable comments.

References

- [1] <http://www.chips.ibm.com/techlib/products/acics/databooks.html>
- [2] 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.
- [3] M. Kishinevsky, J. Cortadella, A. Kondratyev and L. Lavagno. Asynchronous interface specification, analysis and synthesis, Proc. DAC'98, pp. 2-7.
- [4] 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.

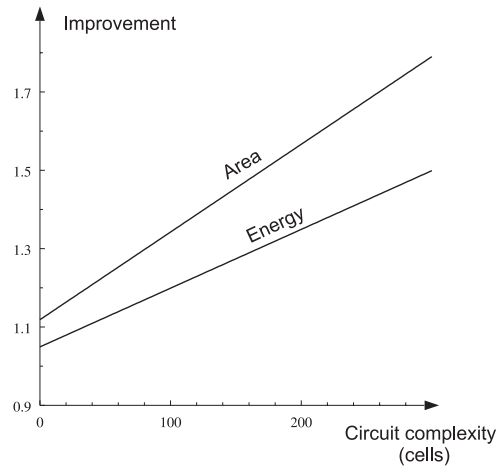


Figure 8: Area and power improvements vs. monotonic circuit complexity

- [5] 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.
- [6] 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.
- [7] G.Mago, "Monotone function in sequential circuits", IEEE Trans. on Computers, Vol. C-22. No. 10, October 1973, pp.928 - 933.
- [8] T.Ibaraki, S.Muroga. "Synthesis network with a minimal number of negative gates", IEEE Trans. on Computers, Vol. C-20. No. 1, January 1971
- [9] C.Piguet. Logic synthesis of race-free asynchronous sequential circuits. IEEE JSSC, vol.26, No 3, March 1991. pp. 371-380.
- [10] C.Piguet. Synthesis of Asynchronous CMOS Circuits with Negative Gates. Journal of Solid State Devices and Circuits, vol.5, No.2, July 1997.
- [11] C.Piguet. Design of Speed-Independent CMOS Cells from Signal Transition Graphs. PATMOS'98. Oct.1998, Copenhagen, pp.357-366.
- [12] 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).
- [13] 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).
- [14] 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.