STG Optimisation in the Direct Mapping of Asynchronous Circuits*

D. Sokolov, A. Bystrov, A. Yakovlev

School of Electrical, Electronic and Computer Engineering,
University of Newcastle upon Tyne, UK

Abstract

Direct mapping from Petri nets (PN) and Signal Transition Graphs (STG) avoids algorithmic complexity inherent in logic synthesis methods based on optimal state encoding. However, it may lead to inefficient implementation, both in size and performance, due to excessive use of state-holding elements.This paper presents a set of tools that optimise logic produced by the direct mapping technique by means of: exposure of outputs, detection and elimination of redundant places. Output exposure is an approach to explicitly model output signals as STG places, which can be directly mapped into output flip-flops. The STG can be simplified after output exposure. The detection of redundant places is a computationally hard problem with multiple solutions. The tool solves this problem by using several heuristics aimed at speed and size. All operations preserve behavioural equivalence. The efficiency of the overall algorithm and individual heuristics is analysed using a number of benchmarks.

1. Introduction

There exist two paradigms of logic circuit design: synchronous and asynchronous. Synchronous design is based on two major assumptions: all signals are binary and the time is discrete. In the asynchronous methodology the signals are binary and the time is continuous. While the majority of modern circuits are synchronous, such properties as robustness to variations in operation conditions (e.g. power supply voltage, environment temperature) and lower power consumption attract interest to asynchronous circuit design.

Two main approaches to asynchronous design are logic synthesis [2] and direct mapping [4, 5]. Whilst the former is well developed and supported by tools, the latter is insufficiently studied. Existing techniques for direct mapping often produce large circuits with an inefficient interface to the environment. A two-level approach for design of low-latency *speed-independent* circuits [8] has been proposed [1]. The subject of this paper is a description of the tools implementing the above approach and several new heuristics for the optimisation of the STG used for direct mapping.

*EPSRC: grant GR/R16754 (BESST), grant GR/M94366 (MOVIE)

2. Background

2.1. Model

In this paper *1-safe Petri Nets* (PN) are used to capture concurrent behaviour of an asynchronous system and *Signal Transition Graphs* (STG) as a circuit specification.

Traditionally, a PN 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 $M : P \to N$ denoting the number of *tokens* in each place ($N = \{0,1\}$ for 1-safe PNs). It is assumed that $\bullet t \neq \emptyset \neq t\bullet, \forall t \in T$. 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 $i = 0, \ldots, n-1$. M_n is called a *reachable marking*.

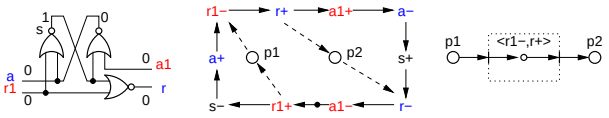
An extension of a PN model is a *contextual net* [7]. It uses additional elements such as *non-consuming arcs*, which only control the enabling of a transition and do not consume tokens. In this paper only one type of non-consuming arcs is used, namely *read-arcs*. A set of read-arcs R can be defined as follows: $R \subseteq (P \times T), R \cap F = \emptyset$. There is an arc between p and t iff $(p,t) \in R$.

An 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. A set of signals A can be divided into a set of *input signals* I and a set of *output and internal signals* O, $I \cup O = A, I \cap O = \emptyset$. Note, that a set of read-arcs R has been included into the model of STG, which is an enhancement w.r.t. [9].

2.2. Direct mapping

The tools presented in this paper rely on the method of direct mapping described in [5]. In this method every place of an STG is associated with a *David cell* (DC) [3], whose circuit diagram is shown in Figure 1(a). DCs can be coupled using four-phase handshake interfaces, so that the interface $\langle r, a1 \rangle$ of the previous stage DC is connected to the interface $\langle r1, a \rangle$ of the next stage. The operation of a single DC is illustrated in Figure 1(b). Places $p1$ and $p2$ correspond to active levels of signals $r1$ and r respectively. They can

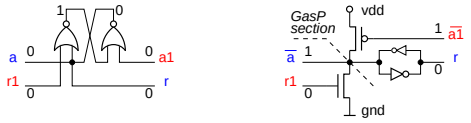
be used to model places of a PN as shown in Figure 1(c). The dotted rectangle depicts the transition between $p1$ and $p2$. This transition contains an internal place, where a token 'disappears' for the time $t_{r1-\rightarrow r+}$. In most cases this time can be considered as negligible, because it corresponds to a single gate delay. The limitation of the method using DCs is that any loop in the specification PN must contain at least three places [5].



(a) Circuit (b) STG (c) Model

Figure 1. David cell

Faster and more compact solutions for a DC implementation were developed in [1] by introducing timing assumptions. In DC implementations shown in Figure 2 the reset phase of state holding element happens concurrently with the token move into the next stage DC. An interesting feature of the implementation in Figure 2(b) is that it internally contains GasP interface [11], which uses a single wire to transmit a request in one direction and an acknowledgement in the other.



(a) Gate−level (b) Transistor−level

Figure 2. Fast David cell implementation

2.3. Device-environment interface

As proposed in [1], in order to transform the initially closed (with both input and output transitions) system specification into the open system specification, the concepts of environment tracking and output exposure should be applied. This concept can be applied to STG that is consistent, output persistent and delay insensitive to inputs.

An STG is *consistent* if in any transition sequence from the initial marking, rising and falling transitions of each signal alternate.

A signal x is *persistent* if there is no event $x*$ of signal x disabled by another event $y*$. An STG is *output persistent* if all output signals are persistent and input signals cannot be disabled by outputs.

An STG is *delay insensitive (DI) to inputs* if no event $x*$ of input signal x is switched by another event $y*$ of input signal y. If this condition is not satisfied, then the method of *order relaxation* described in [10] can be used. This condition seems to be too strong because an input sequence may exist which, for example, contains multiple transitions of the same signal, but still uniquely identifies the end of the sequence. However, for simplicity in this paper only the case of DI inputs is considered.

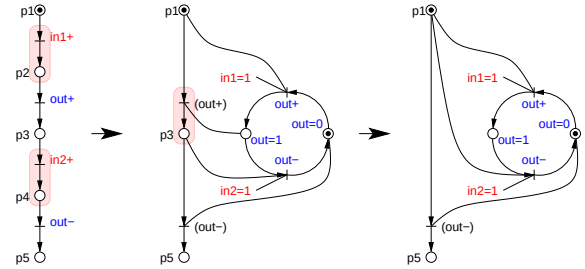
The first step in constructing our model is splitting the system into device and environment. For this the original STG is duplicated as shown in Figures 3(a,b). Then, in the first copy, corresponding to the *device*, input events are replaced by dummies and in the second copy, corresponding to the *environment*, output events are replaced by dummies. Behaviour of the device and environment is synchronised by means of read-arcs between dummy transitions and successor places of their prototypes in the counterpart as shown in Figure 3(b).

At the second step the outputs of both device and environment are exposed by the following technique. Every interface signal is associated with a pair of complementary places representing the low and high levels of the signal. These places are inserted as transitive places between the positive and negative transitions of the signal, expressing the property of signal consistency. *Trackers* of device and environment use these *exposed outputs* to follow (or *track*) the behaviour of the counterparts as shown in Figure 3(c).

After that, *elementary cycles* are formed and read-arcs are introduced to represent the signals as shown in Figure 3(d). Read-arcs from the predecessor places of dummies to signal transitions and from the successor places of signal transitions to dummies preserve the behaviour of the system. The resultant system specification is *weakly bisimular* [6] to the original. The elementary cycles are subsequently implemented as set-reset Flip-Flops (FF) and the places of the tracker as DCs.

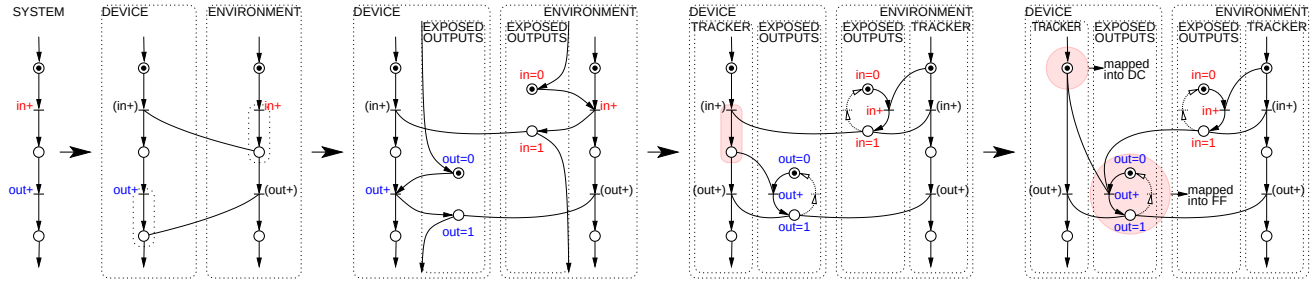
It is often possible to control outputs by the directly preceding interface signals without using intermediate states. Many places and preceding dummies can thus be removed, provided that the system behaviour is preserved w.r.t. input-output interface (weak bisimulation). Such places are called *redundant*. Their elimination is restricted, however, by potential *coding conflicts*.

Coding conflicts may cause tracking errors. For example, the STG in Figure 4(a) can be transformed as shown in Figure 4(b) by output exposure and removal of redundant places $p2$ and $p4$. However, if the place $p3$ is eliminated as shown in Figure 4(c), then the tracker cannot distinguish between the output being not yet set and the output being already reset. Note the specifics of our direct mapping approach: only those signals whose switching directly precedes the given output or tracker transition are used in its support.



(a) STG (b) Output exposure and optimisation (c) Coding conflict

Figure 4. Preventing coding conflicts



(a) STG (b) Device−environment (c) Complementary places (d) Output exposure (e) Optimisation

Figure 3. Device-environment interface

3. Tools

This section describes the set of tools implementing the above method. The package contains programs for:

- detection of redundant places;
- exposure of the outputs;
- elimination of redundant places;
- mapping of optimised specification into circuit.

The following subsections describe these tools in detail. The example STG used throughout the following subsections is shown in Figure 5.

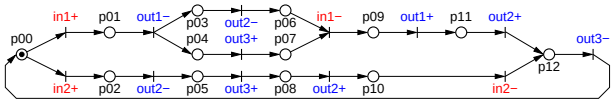


Figure 5. Example STG

3.1. Detection of redundant places

After the exposure of outputs some of the places in the tracker part together with preceding dummy transitions can be removed. It is easier, however, to detect redundant places by processing the original specification. For this each place is given a tag having one of three values: UNDEFINED (the place has not been tested yet), REDUNDANT (the place can be safely removed after output exposure) or MANDATORY (the place should be preserved). The sets of UNDEFINED, REDUNDANT and MANDATORY places are denoted as P_U, P_R and P_M respectively. First, all places are tagged as UNDEFINED. Then, if place p satisfies the Condition 1 and Condition 2, it is tagged as REDUNDANT, otherwise as MANDATORY**.** The order in which the places are processed affects the result and follows the heuristics presented in Algorithm 1.

Condition 1. *There must be more than three* UNDEFINED *or* MANDATORY *places in every loop containing place p.*

This restriction on place redundancy is imposed by the minimal number of DCs in a loop, which is three [5].

Condition 2. *Removal of place p must not cause a coding conflict.*

A coding conflict is detected by intersecting two sets of signals. The first set contains the signals whose transitions are fired in the *forward neighbourhood* $\varepsilon_f(p)$ of place p is defined as the minimal (w.r.t.$\subseteq$) set such that:

$$\varepsilon_f(p): \quad \begin{array}{l} p \in \varepsilon_f(p); \\ \forall\, x \in T \cup P_R \; if \; \exists\, y \in \varepsilon_f(p) \;\mid\; x \in y\bullet, \\ \quad then \; x \in \varepsilon_f(p) \end{array} \qquad (1)$$

The set of signals in question can now be defined using labelling function λ:

$$A_f(p) = \lambda(\varepsilon_f(p) \cap T) \qquad (2)$$

The second set is defined similarly to (1), 2 and contains the signals whose transitions are fired in the *backward neighbourhood* $\varepsilon_b(p)$ of place p:

$$\varepsilon_b(p): \quad \begin{array}{l} p \in \varepsilon_b(p); \\ \forall\, x \in T \cup P_R \; if \; \exists\, y \in \varepsilon_b(p) \;\mid\; x \in \bullet y, \\ \quad then \; x \in \varepsilon_b(p) \end{array} \qquad (3)$$

$$A_b(p) = \lambda(\varepsilon_b(p) \cap T) \qquad (4)$$

If $A_f \cap A_b = \emptyset$ then removal of the place p does not cause coding conflicts. Note that the above argument is based on the fact that the neighbourhoods are acyclic due to the enforcement of Condition 1.

Let us consider the example shown in Figure 5. First, all the places $p00$-$p12$ are tagged as UNDEFINED. At Heuristic A stage the places $p01$, $p02$, $p09$ are tagged as REDUNDANT, because all their predecessors are transitions of input signals and all their successors are transitions of output signals (those places are shown as small circles). The result of this tagging is shown in Figure 6(a).

Places $p05$, $p11$, $p12$ are tagged as REDUNDANT and $p03$, $p04$, $p08$ as MANDATORY at Heuristic B stage. It works as follows. There are four chains of UNDEFINED places: $\{p11, p12, p00\}$; $\{p05, p08, p10, p12, p00\}$; $\{p03, p06\}$; $\{p04, p07\}$. According to our heuristics the chains should be processed in the order shown in roman numerals in Figure 6(b). First, the places in the chain $\{p11, p12, p00\}$ are tested for being redundant and places $p11$, $p12$ are tagged as REDUNDANT because they satisfy Condition 1 and Condition 2. The place $p00$ remains UNDEFINED because it is the last place in the chain. Next,

Algorithm 1 Detection of redundant places

input: STG, tagging of places
output: new tagging of places
{ Heuristic A: output latency reduction }
for each $p \in P_U \mid \bullet p \subseteq I$ and $p\bullet \subseteq O$ **do**
  if p satisfies Condition 1 and Condition 2 **then**
    tag p as REDUNDANT;
  else
    tag p as MANDATORY;
  end if
end for
{ Heuristic B: size reduction }
generate a set G of chains containing more then one place $p \in P_U$ in sequence;
use heuristics to select the order of chains processing;
for each chain $g \in G$ **do**
  associate p with the first place of g;
  while p is not the last place in g **do**
    if p satisfies Condition 2 within chain g and Condition 1 **then**
      tag p as REDUNDANT;
    else
      tag p as MANDATORY;
    end if
    associate p with the next place in g;
  end while
end for
{ Heuristic C: size reduction }
for each $p \in P_U$ **do**
  if p satisfies Condition 1 and Condition 2 **then**
    tag p as MANDATORY;
  else
    tag p as REDUNDANT;
  end if
end for

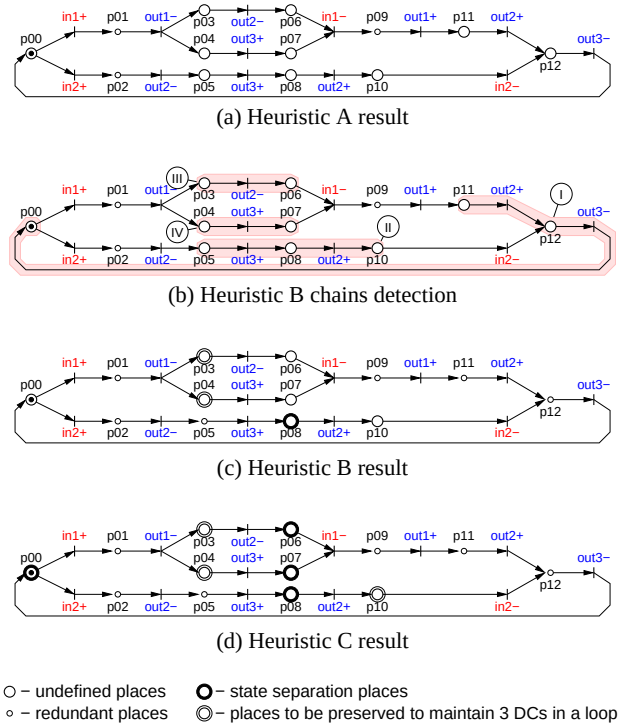
the chain $\{p05, p08, p10\}$ (places $p12$ and $p00$ are cut, as they have already been processed) is considered. Place $p05$ is tagged as REDUNDANT. Place $p08$ separates transitions of signal $out2$ and is tagged as MANDATORY. The place $p10$ remains UNDEFINED as the last in the chain. Finally, in the chains $\{p03, p06\}$ and $\{p04, p07\}$ places $p03$ and $p04$ are tagged as MANDATORY because there are only three non-REDUNDANT places left in the loops containing them. Places $p06$ and $p07$ remain UNDEFINED as the last in the chains. The result is shown in Figure 6(c).

At Heuristic C stage the places $p00$, $p06$, $p07$, $p10$ are tagged as MANDATORY because $p00$ separates $out1$ transitions, $p06$ separates transitions of $out2$, $p07$ separates $out3$ transitions. Place $p10$ should be preserved to maintain three places in the loop. The resultant STG is shown in Figure 6(d).

3.2. Exposure of outputs

The procedure of output exposure is defined in Algorithm 2. It converts the closed system specification (environment behaviour is included in the model) into the open system specification (environment is excluded and assumed to work correctly). This transformation should be applied to the specification after the detection of redundant places.

The result of output exposure applied to the STG of Figure 6 is depicted in Figure 7. First, the initial levels of signals $in1 = 0$, $in2 = 0$, $out1 = 1$, $out2 = 1$, $out3 = 0$ are calculated from the initial marking. Next, pairs of places representing low and high levels of input signals ($in1 = 0$, $in1 = 1$; $in2 = 0$, $in2 = 1$) and output signals ($out1 = 0$, $out1 = 1$; $out2 = 0$, $out2 = 1$; $out3 = 0$, $out3 = 1$) are



(a) Heuristic A result

(b) Heuristic B chains detection

(c) Heuristic B result

(d) Heuristic C result

(e) Legend of place tagging

Figure 6. Detection of redundant places

Algorithm 2 Exposure of outputs

input: STG
output: new STG with exposed outputs
for each $x \in A$ **do**
  create places '$x = 0$' and '$x = 1$';
  if initial level of x is low **then**
    mark '$x = 0$' with a token;
  else
    mark '$x = 1$' with a token;
  end if
end for
for each $x+ \in T$ **do**
  replaces $x+$ with dummy '$(x+)$';
  create transition $x+$;
  create consuming arc '$x = 0$' $\rightarrow x+$;
  create consuming arc $x+ \rightarrow$ '$x = 1$';
  create read-arcs $\bullet$ '$(x+)$' $\leftrightarrow x+$;
  create read-arc '$x = 1$' $\leftrightarrow$ '$(x+)$';
end for
for each $x- \in T$ **do**
  replaces $x-$ with dummy '$(x-)$';
  create transition $x-$;
  create consuming arc '$x = 1$' $\rightarrow x-$;
  create consuming arc $x- \rightarrow$ '$x = 0$';
  create read-arcs $\bullet$ '$(x-)$' $\leftrightarrow x-$;
  create read-arc '$x = 0$' $\leftrightarrow$ '$(x-)$';
end for

created. They are marked according to the initial levels of the signals. Finally, transitions of these signals are moved from the tracker to the interface and are replaced by dummies in the tracker. Corresponding read-arcs preserving behavioural equivalence are created between tracker and exposed outputs.

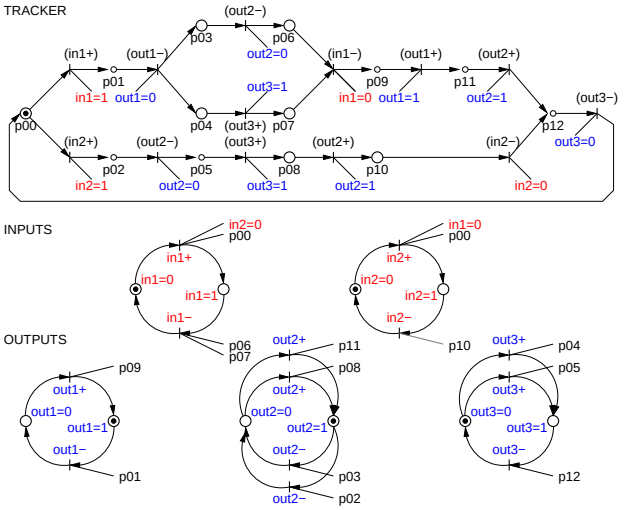


Figure 7. Exposure of outputs

3.3. Elimination of redundant places

Algorithm 3 describes the process of eliminating the redundant places. This procedure should be used after detection of redundant places and exposure of outputs.

Algorithm 3 Elimination of redundant places

input: STG with exposed outputs, tagging of places
output: new STG with redundant places removed
for each $p \in P_R \mid p$ is marked **do**
  roll back the token to the nearest MANDATORY place or places;
end for
for each $p \in P_R$ **do**
  for each $a \in F \mid a$ is a read-arc from p **do**
    create a set of read-arcs from the MANDATORY places preceding p;
    delete a;
  end for
end for
for each $p \in P_R$ **do**
  delete $\bullet p$ and p;
  locally change the structure;
end for
simplify the spec removing duplicated arcs, places and transitions;

Figure 8 shows the optimised specification of the STG given in Figure 6. Redundant places $p01$, $p09$, $p11$, $p02$, $p05$ are removed, together with preceding dummies, and read-arc origins are moved to the preceding MANDATORY places. The removal of place $p12$ causes the splitting of transition '$(out3-)$'.

It should be noted that splitting MANDATORY places increases the fanin of all the following elements and the fanout of the preceding elements, as all arcs from such places are duplicated. Similarly, transition splitting increases the fanout of the preceding elements and the fanin of the following elements, because arcs to such transitions are also duplicated.

4. Benchmarks

The benchmarks shown in Table 1 are studied with four levels of optimisation (corresponding to levels of redundant

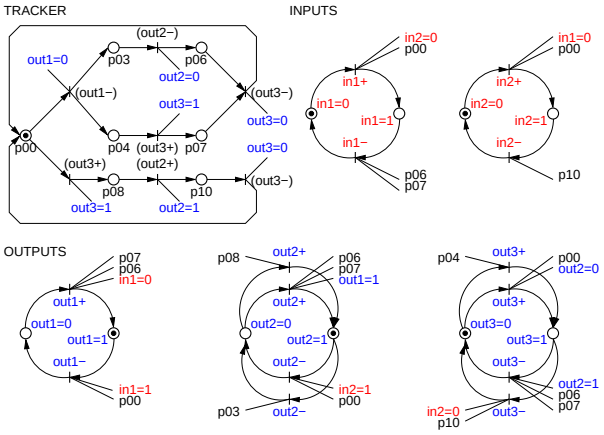


Figure 8. Elimination of redundant places

places detection):
LEVEL 0 - specification is not optimised;
LEVEL 1 - heuristic A is applied (decreases output latency, further size optimisation is possible);
LEVEL 2 - heuristics A and B are used;
LEVEL 3 - heuristics A, B and C are applied.

The number of transistors is counted for the case of places being implemented as fast DCs shown in Figure 2(b), request-acknowledgement logic of DCs and set-reset logic of FFs being implemented at transistor level. The condition of having at least three DCs in a loop is met. Latency is counted as the number of transistors in the longest chain of switching transistors.

Tests of *toggle* and *imec-alloc-outbound* show that half of DCs can be eliminated in the first level of optimisation, which is most important in terms of latency. The number of transistors for device implementation decreases up to 30%. The fanins and fanouts of DCs do not grow. Further optimisation of *toggle* does not decrease the number of DCs and transistors, because the alternation of input and output signals makes it possible to detect all redundant places in the first level.

The case of *pe-rcv-ifc* should be analysed separately. Even the first level of optimisation increases the number of switching transistors in a chain to 7 and the next levels make it even worse - 14. This can be explained by the splitting of a MANDATORY place before the deep hierarchy of forks and the recalculation of read-arcs from redundant places to the split places. The solution to this problem is to evaluate the complexity of elements by calculating the maximum number of incoming arcs into a transition (which corresponds to maximum fanin of DCs or FFs) before and after removal of every place. If this number is increasing (or at least is increasing beyond the maximum implementable value), then the place should be kept in the specification. This optimisation reduces the size of the circuit by preventing complication of logic rather than reducing the number of DCs. This is the *element complexity optimisation* technique, whose results are shown in Table 1 in parentheses.

Direct mapping of *par5* and *count* (with element complexity optimisation) generates circuits with lower output

optim level	DC num	fi n	fout	trans count	max latency	comp time
toggle						
0	8	2	2	70	2	0.064s
1-3	4	2	2	46	2	0.075s
petrify				24	2	0.114s
imec-alloc-outbound						
0	17	2	2	151	1	0.089s
1, 2	8	2	4	97	2	0.126s
3	7	2	4	90	2	0.116s
petrify				32	3	2.023s
pe-rcv-ifc						
0	58	4	4	537	4	0.192s
1	45 (51)	6 (4)	7 (4)	521 (495)	7 (3)	0.350s
2	36 (38)	6 (4)	12 (4)	524 (412)	14 (2)	1.713s
3	30 (36)	6 (4)	13 (4)	478 (400)	14 (2)	1.928s
petrify				114	4	6.592s
par5						
0	28	6	5	256	6	0.128s
1-3	16 (17)	6 (6)	6 (5)	208 (180)	10 (5)	0.184s
petrify				108	10	7m52s
count						
0	21	3	4	168	1	0.092s
1	18 (19)	4 (3)	6 (4)	192 (156)	5 (2)	0.074s
2	15 (14)	4 (3)	5 (4)	167 (132)	5 (2)	0.088s
3	13 (14)	4 (3)	5 (4)	153 (132)	5 (2)	0.108s
petrify (manual CSC solution)				80	5	1.412s

() - results of element complexity optimisation

Table 1. Benchmark results

latency than logic synthesis. Scaling of *par5* example to have 4, 5 and 6 concurrent branches has shown exponential growth of *petrify* computation time *(1m15s*, *7m52s* and *33m12s* respectively*)*, whilst our algorithm produced the result in *0.165s*, *0.184s* and *0.206s* respectively. In some cases (e.g *count*) *petrify* failed to resolve a CSC conflict (even if it was reducible) and our algorithm has completed the job.

In general, these benchmarks show the high efficiency of the optimisation method. Circuits produced by the proposed technique are usually larger than *petrify* solutions, but they often have lower output latency. Our method can also process large specifications, which are not computable by *petrify* in acceptable time.

5. Summary

The main part of the toolkit for the low-latency asynchronous circuit design by direct mapping is developed. The adopted architecture allows the minimisation of state-holding elements and reduction of latency. The characteristic feature of the method is that the optimisation is achieved at the specification level (as opposed to optimisation of logic circuits after the stage of synthesis). The approach exploits the two-level architecture where a circuit consists of two blocks: the tracker and the block of output flip-flops. The tracker computes context signals for outputs concurrently with the environment operation, thus achieving the latency reduction effect. The output flip-flops generate outputs from context and trigger signals. The tools work in three stages: first, redundant places are detected; then, the output signals are exposed; finally, redundant places together with preceding dummies are deleted. The benchmark study indicates the high efficiency of optimisation heuristics. Significant size reduction can be achieved at the first level alongside with latency reduction. Further optimisation still decreases the number of circuit elements but sometimes increases their fanin. In such cases the element complexity should be optimised.

References

[1] A. Bystrov and A. Yakovlev. Asynchronous circuit synthesis by direct mapping: Interfacing to environment. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 127–136, Manchester, UK, Apr. 2002. IEEE Computer Society Press.

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

[3] R. David. Modular design of asynchronous circuits defi ned by graphs. *IEEE Transactions on Computers*, 26(8):727–737, Aug. 1977.

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

[5] M. Kishinevsky, A. Kondratyev, A. Taubin, and V. Varshavsky. *Concurrent hardware: the theory and practice of self-timed design*. Series in Parallel Computing. Wiley-Interscience, John Wiley & Sons, Inc., 1994.

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

[7] U. Montanari and F. Rossi. Acta informacia. Technical report, 1995.

[8] D. E. Muller and W. S. Bartky. A theory of asynchronous circuits. In *Proceedings of an International Symposium on the Theory of Switching*, pages 204–243. Harvard University Press, Apr. 1959.

[9] L. Rosenblum and A. 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.

[10] H. Saito, A. Kondratyev, J. Cortadella, L. Lavagno, and A. Yakovlev. What is the cost of delay insensitivity? In *Proc. International Conf. Computer-Aided Design (ICCAD)*, pages 316–323, Nov. 1999.

[11] I. Sutherland and S. Fairbanks. GasP: a minimal FIFO control. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 46–53. IEEE Computer Society Press, Mar. 2001.