Parallel Controller Synthesis from a Petri Net Specification

K. Biliński, M. Adamski †, J. M. Saul, E. L. Dagless

Department of Electrical and Electronic Engineering,
University of Bristol, Bristol BS8 1TR, United Kingdom

†Department of Computer Engineering and Electronics,
Higher College of Engineering, 65-246 Zielona Gora, Poland

Abstract

This paper presents new algorithms for the synthesis of parallel controllers that operate on a Petri net. This net is first simplified by reduction, then coloured and finally used to generate a state assignment with which the controller can be synthesised. The new concept of using colours for detecting and representing concurrency within the Petri net is presented.

1 Introduction

The synthesis of a controller which drives parallel interacting processes can be carried out using one of two main approaches. In the first one FSM synthesis techniques are used. The specification of a controller is divided into a number of concurrent processes, so that a set of sequential controllers is produced. Next, the controllers are implemented as FSMs and linked together with cross–connected control lines and/or semaphore bits subject to the initial parallel specification. The second approach uses a Petri net to specify the parallel controller. Controller behaviour can be analysed using well defined concepts from Petri net theory, which involve reachability graphs and net invariants methods. In addition, a graphical representation of concurrency is often easier to understand, and so it can reduce the likelihood of parallel synchronisation errors.

In [2] and [5] some techniques for synthesising parallel controllers from Petri net specifications were introduced. However, they have a major disadvantage. All of them need the construction of a reachability graph from the Petri net. This operation requires a long computation time and lots of memory, particularly when complex controllers are synthesised. In this paper a new approach to parallel controller synthesis is given and illustrated with examples. The main idea is to use colours to detect and represent concurrency within the net instead of reachability graph generation. It should be pointed out that the concept of colouring a Petri net presented here is different to the one presented in [3], known as CP–nets. The main point in CP–nets is that token represents information and the data value which is attached to it is referred to as the token colour. CP–nets explicitly attach a set of possible token–colours to each place and a set of possible occurrence–colours to each transition. The basis of the approach investigated in this research was to use colours in order to detect and represent concurrency in Petri nets. It is shown in Section 6.2 that information about concurrency within the net, which can be obtained from a coloured Petri net, is similar to that contained in its reachability graph.

The paper is organised as follows: Section 2 gives a brief introduction to Petri nets and parallel controllers. An overview of the proposed methodology is given in Section 3. In Section 4, an algorithm for structural reduction is presented. In the following section a set of colouring rules and a net colouring algorithm is shown. Two methods for place encoding are given in Section 6. Experimental results are discussed in Section 7, and concluding remarks are presented in Section 8.

2 Preliminaries

A Petri net is a bipartite, weighted, directed graph, which has two types of nodes called *places*, represented by circles and *transitions*, represented by bars. Directed arcs connect the places and the transitions, with some arcs leading from the places to the transitions and others vice versa. To each arc a weight, a positive integer represented by a label, is assigned, where the m–weight arc can be viewed as the set of m parallel arcs. When $m = 1$ the label is usually omitted. A marking is an assignment of *tokens*, represented as black dots, to the places. The position and the number of tokens changes during the net execution. Formally a Petri net PN is defined as a 5-tuple [4]:

$$PN = (P, T, F, W, M_0)$$

where: $P = \{p_1, p_2, ..., p_m\}$ is a finite non-empty set of places; $T = \{t_1, t_2, ..., t_n\}$ is a finite non-empty set of transitions; $F \subseteq (P \mathrm{x} T) \cup (T \mathrm{x} P)$ is a finite non-empty set of arcs; $W : F \rightarrow \{1, 2, 3, ...\}$ is a weight function; $M_o : P \rightarrow \{0, 1, 2, ...\}$ is the initial marking; $P \cap T = \emptyset$ and $P \cup T \neq \emptyset$.

The net execution is performed according to simple rule for transition enabling and firing [4]. A Petri net is said to be an ordinary Petri net if all of its arc

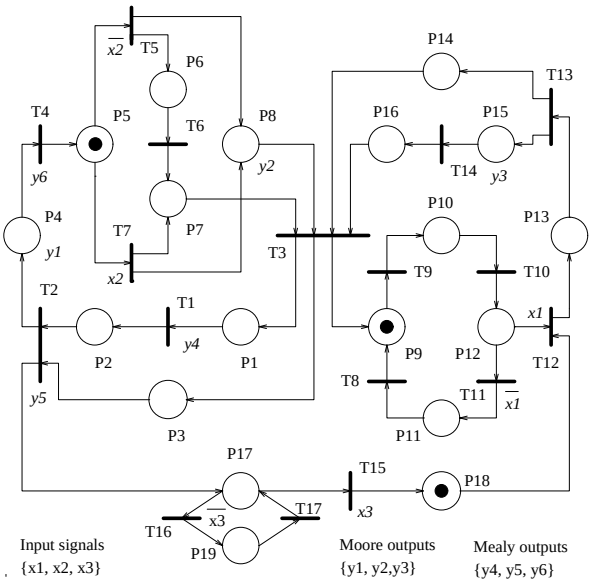

Figure 1: An example of an interpreted Petri net.

weights are set to one. An ordinary Petri net is said to be a State Machine (SM) if each its transition has exactly one input and one output place. An ordinary Petri net is said to be a State Machine Decomposable (SMD) if and only if there exist a collection of state machines $SM_i = (P_i, T_i, F_i)$, $1 \leq i \leq N^+$ such that $P = \cup P_i, T = \cup T_i, F = \cup F_i$.

When there are many parallel, interacting subprocesses to be controlled, using a finite state machine can be awkward, since each state has to control all of them. A better solution is to divide each of the global sequential states into a number of concurrently active local states, with each local state controlling a different subprocess [5]. The main difference between a finite state machine and parallel controller is that whilst a finite state machine has only one state active at any time, a parallel controller can have several states active simultaneously.

For a parallel controller specification a high level Petri net is used, which is called an interpreted synchronous Petri net. An example of an interpreted Petri net is shown in Figure 1. A predicate may be attached to each transition, which is a Boolean function of the controller's input signals. Moore outputs are associated with places and Mealy outputs with transitions. Each place represents a local state of the controller, and so the global state of the controller is equivalent to the current marking of the net. New rules for transition enabling and firing are introduced. A transition is enabled when all of its input places are marked and its predicate, if present, is asserted. All transitions are synchronised by a global clock and so all enabled transitions fire simultaneously.

In this paper it is assumed that a parallel controller is synchronised using a global clock, and is described using an ordinary SMD net which is synchronous and interpreted.

3 Synthesis methodology

The synthesis methodology that is presented in this paper consists of the following steps:

- First, a parallel controller is specified using a Petri net; then the net is reduced and a macronet is achieved; this module performs also an internal place encoding.
- Next, the macronet is coloured according to the colouring rules that are presented in Section 5. Places that are marked with the same colour are used to generate a state machine component of the macronet. The major idea behind further net analysis is based on the fact that places which are marked with disjoint sets of colours are concurrent to each other. The next steps of the synthesis can be performed following one of two possible approaches.
  - The coloured macronet can be decomposed into a set of state machine components. After decomposition each component is encoded separately using a unique set of latches.
  - An alternative approach is to encode the entire macronet according to some heuristic rules which have been developed within this research.
- Finally, a logic–level description of a parallel controller is achieved in a standard format that can be accepted by logic synthesis tools.

The following sections discuss the above methodology and illustrating it with appropriate examples.

4 Structural reduction

4.1 Reduction algorithm

To simplify the process of the analysis of a Petri net a reduction of the net is performed. During reduction a Petri net representation is converted into a more general description, called a macronet.

There are many reduction techniques for Petri nets. Some of the reduction methods were introduced in [2] and [4], three of which are implemented into the system.

- Fusion of Series Places (FSP) — as depicted in Figure 2 (a).
- Fusion of Parallel Places (FPP) — as depicted in Figure 2 (b).
- Fusion of Distributed Transitions (FDT) — as depicted in Figure 2 (c).

In [2] the concept of the Nth–order macronet is introduced. In this approach to reducing the net, both FSP and FPP techniques are used recursively until the macronet becomes irreducible. In the system presented here this idea is enlarged by adding the next

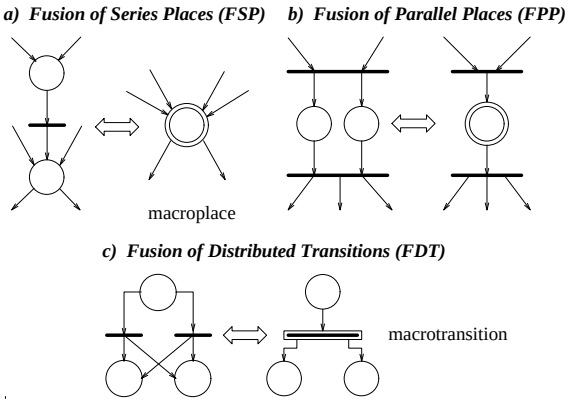

Figure 2: Petri net reduction techniques.

reduction technique — FDT. For a large set of nets this extension gives a distinct improvement in the results of the reduction. The algorithm for the Petri net reduction is shown in Figure 3[1]. The algorithm consists of three main steps.

- *FSP_scanning* (FSP–S), each of the transitions is checked to see whether it is a transition connected with a single input and a single output place. For each transition that satisfies the above condition the FSP technique is used.
- *FPP_scanning* (FPP–S), for all places which have the same input and output transitions the FPP algorithm is performed.
- *FDT_scanning* (FDT–S), all transitions are checked to see whether they share exactly the same sets of input and output places. For all transitions that satisfy this condition the FDT reduction method is used.

In Figure 4 the macronet for the Petri net presented in Figure 1 is given.

4.2 Place encoding during reduction

To reduce the number of latches needed to implement the Petri net and to simplify the main encoding algorithm, a method of place encoding inside a macroplace is used. After FSP–S a technique for sequential encoding is performed and after FPP–S a parallel encoding method is used. Only places which belong to macroplaces are encoded. Each macroplace is encoded separately.

For sequential place encoding — orthogonal codes are assigned to all places; for parallel place encoding — renaming of latches is performed to make codes of concurrent places non-orthogonal to each other. Two codes are said to be orthogonal if they share all of their state variables.

[1] All algorithms which are presented in this paper are written in C–like pseudo–code.

```
int net_reduction (petri_net)
 {
  do {
     do {
        if (( error =Structure_checker (petri_net))==0) {
           pn_error (error);
           return 0; } /* there is a structural error inside the net */
        FSP_scanning (petri_net);
        if (( error =Structure_checker (petri_net))==0) {
           pn_error (error);
           return 0; } /* there is a structural error inside the net */
        Series_place_encoding (petri_net);
        FPP_scanning (petri_net);
        Parallel_place_renaming (petri_net);
     }
     while ( FPP algorithm was used at least once );
     FPT_scanning (petri_net);
  }
  while (FPT algorithm was used at least once );
  return 1;  /* reduction complete */
 }
```

Figure 3: Petri net reduction algorithm.

5 Coloured net

To determine the relationship between places in the macronet a colouring algorithm is used. It is assumed that if a net is coloured it means that the net is covered by a set of subnets, which all are State Machines. Since each place in the State Machine is sequential to the other places, implementation of such a net is very simple.

After the Petri net colouring colours are associated with places and transitions. Each colour represents one state machine subnet. Places in each of the subnets are sequentially related to each other and concurrently related to places of any other subnet. Thus if two places share any colour it is said that these two places are sequentially related to each other.

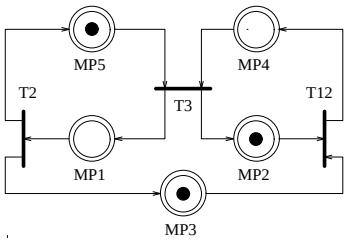

Figure 4: The net from Figure 1 after reduction.

There is a set of rules which must be satisfied during a net colouring:

a. If a place has a colour each of its input and output transitions must have the same colour.
Each live state machine subnet has to be strongly connected (together with point **e***).*

b. Each place and transition must have at least one colour.
The entire net must be covered using state machines, so that it cannot be a place or a transition which does not have a colour (i.e. which does not belong to any state machine subnet).

c. The input places of each transition must hold different colours.
A state machine does not has any transition with two or more input places.

d. The output places of each transition must hold different colours.
A state machine does not has any transition with two or more output places.

e. The input and output places of a transition must share the same set of colours.
Each live state machine subnet has to be strongly connected (together with point **a***).*

f. There are not two or more initially marked places which share exactly the same set of colours.
Each of the state machine subnets must contain at most one token.

g. The number of different colours which are shared by the places initially marked is equal to the total number of colours.
Each safe state machine has exactly one token.

Theorem 1: Let a Petri net be coloured. Then the net is safe and live.

The formal proof of the above theorem can be found in [1]. Unfortunately, the converse of this result is not true.[2] If the colouring algorithm fails the standard method for analysis and synthesis of parallel controllers is performed [5]. The colouring algorithm is shown in Figure 5.

6 Place encoding

Two algorithms for place encoding of the macronet have been implemented. In the first approach parallel decomposition is first performed and then each of the components is encoded. In the second approach a new method has been developed that uses a hierarchical algorithm to encode places straight from the coloured net. These two methods are described in Sections 6.1 and 6.2 respectively.

6.1 Net decomposition encoding methods

To simplify the net encoding a parallel decomposition is performed. Let us consider the macronet of Figure 4. It is easy to see that after the net colouring each colour clearly defines one of the net components (Figure 6 (a), (b)). However, some of the places (macroplaces) belong to more than one component, e.g. MP4 in Figure 6 (b). For the encoding algorithm each place must belong to only one component. The most important part of the algorithm presented here is to choose the correct strategy to decide which places will be removed and which will be kept by the component. All places removed from any component are replaced by a special macroplace, e.g. PR1 in Figure 6 (b).

The heuristic algorithm which finds the first good irredundant disposition of places with the disposition criterion — minimal number of latches needed to encode the entire net — can be defined as follows:

[2] The presented method can only be applied to SMD nets.

```
int Net_Colouring (petri_net)
  {
    while ( Is not coloured pleace in net ) {
        start_place = choose_free_place (petri_net);
        for (all output transition) {
            if ( transition_colouring (start_place.output_transitions)==0) {
                return 0;   /* Net can not be coloured */
            }
        }
        erase not allowed place and transitions;
    }
    return 1;   /* Net colouring complete */
  }

int transition_colouring (list_output_transitions)
  {
    while ( Is free allowed transition in list_output_transitions) {
        transition = free_transition (list_output_transitions);
        if ( place_colouring (transitions.output_place)==0) {
            mark transition as not allowed;
        }
        else {
            return 1;
        }
    }
    return 0;
  }

int place_colouring (list_output_places)
  {
    if ( Is start_place in lista_output_places) {
        return 1;
    }
    while ( Is free allowed place in list_output_places) {
        place = free_place (list_output_places);
        assign colour to each in/out transitions for place;
        if ( transition_colouring (place.output_place)==0) {
            mark place as not allowed;
            erase colour from each in/out transition for place;
        }
    }
    return 0;
  }
```

Figure 5: The Petri net colouring algorithm.

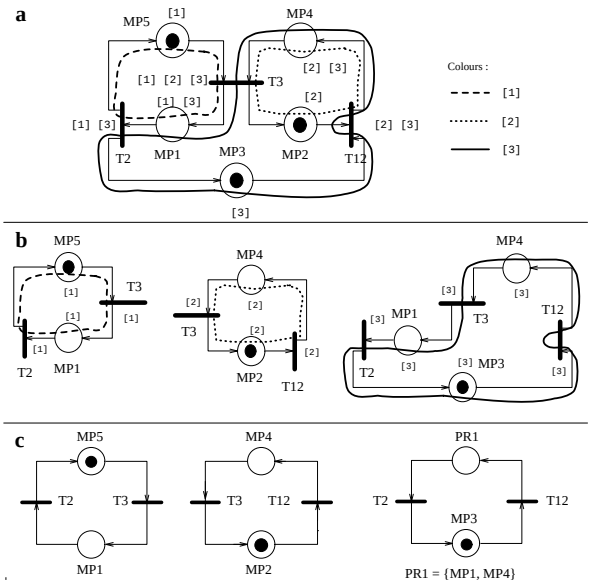

Figure 6: A net decomposition example.

1. *The colour (component) which involves a macroplace (place) that uses a maximum number of latches for internal encoding is taken first.*
2. *If two or more colours (components) have the same number of latches the colour (component) with the larger number of places is chosen.*
3. *If there is still more than one colour the order of choosing the colour is arbitrary.*
4. *The chosen colour (component) holds all of its places (macroplaces), and shared places are removed from the other colours (components).*
5. *The previous steps are repeated as long as all components are created.*

In Figure 6 (b) and (c) the decomposition phases are shown.

As we can see from Figure 6 (c), each component is represented by the state machine, and the standard place encoding algorithm can be used. There is only one restriction: to encode each of the components a disjoint of set of latches has to be used. After place encoding a code that is assigned to the special macroplace is added to code of each place that was previously replaced by this special macroplace.

6.2 Hierarchical colouring place encoding algorithm

Place encoding can be performed directly from the coloured net. Using colours we can find all relations which occur among places within the net — Figure 7. The algorithm for place encoding from the coloured net can be defined as follows:

1. *The place (macroplace) with the largest number of colours is found and a latch state (e.g. state Qn) is allocated.*
   - *To all places (macroplace) which share the same colours as the chosen place (macroplace), the complement state (i.e. $\bar{Q}n$) is allocated. The latch is not allowed to be used in the encoding of places that do not have any of the colours of the chosen place (i.e. places which are concurrently related to the chosen one).*
   - *For places already removed from the net the symbol '–' (meaning allowed) or '*' (meaning not allowed) are used to represent the latch, depending on whether the place (macroplace) is non–concurrent to all or concurrent to any of the currently encoded places (macroplaces).*
2. *The chosen place (macroplace) is removed from the net.*
3. *If two or more places have an identical set of colours they are merged.*
4. *The above steps are repeated until the net is empty.*

The last step in this algorithm is the renaming of the place codes in macroplaces. To minimise the number of latches used to encode the original net all latches marked in macroplaces codes with the symbol '–' can be reused during the internal macroplace renaming.

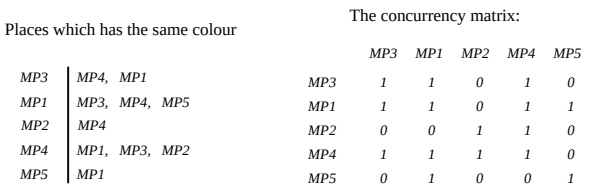

Figure 7: Creating a concurrency matrix from the coloured net of Figure 6 (a).

place	Decomposition algorithm	Hierarchical algorithm
P1	Q10 !Q1 !Q4 !Q3	Q1 !Q6
P2	Q10 Q1 !Q4 !Q3	Q1 Q6
P3	Q10 !Q4 !Q3	Q1
P4	!Q9 !Q8 !Q10	!Q1 !Q3 !Q4
P5	!Q9 Q8 !Q10	!Q1 !Q3 Q4
P6	Q9 !Q8 !Q10	!Q1 Q3 !Q4
P7	Q9 Q8 !Q10	!Q1 Q3 Q4
P8	Q9 !Q10	!Q1 Q3
P9	!Q5 !Q6 !Q7	!Q2 !Q8 !Q7
P10	Q5 !Q6 !Q7	!Q2 Q8 !Q7
P11	!Q5 Q6 !Q7	!Q2 !Q8 Q7
P12	Q5 Q6 !Q7	!Q2 Q8 Q7
P13	Q7 Q2 !Q4 Q3	Q2 !Q1 Q6
P14	Q7 !Q2 !Q4 Q3	Q2 !Q1 !Q6
P15	Q7 !Q2 !Q1 !Q4 Q3	Q2 !Q1 !Q6 !Q5
P16	Q7 !Q2 Q1 !Q4 Q3	Q2 !Q1 !Q6 Q5
P17	!Q2 !Q1 Q4 !Q3	!Q2 !Q1 !Q5 !Q6
P18	Q2 Q4 !Q3	!Q2 !Q1 Q5
P19	!Q2 Q1 Q4 !Q3	!Q2 !Q1 !Q5 Q6

Table 1: The encoding results for the net of Figure 1.

The result of encoding the Petri net of Figure 1 are given in Table 1.

7 Experimental results

In this section, the synthesis results that were achieved by the use of the algorithms presented here are described. The algorithms have been implemented in C and linked to the SIS sequential logic synthesis system [6]. All examples were run on a Sun SPARC–Station2 computer. The benchmarks were taken from published papers and various other sources. The statistics are given in Table 2. The first two columns show the number of places and transitions of the Petri net specification of each controller. The last two columns show the number of states and state transition edges of an equivalent State–Transition–Graph of each controller.

Each of the examples was encoded using the two algorithms that were presented in the preceding sections and then synthesised using the SIS. To evaluate the synthesis results an alternative parallel implementation of each of the examples was created using the methodology presented in [5]. In addition, the functionally equivalent sequential controllers were also generated[3] and then synthesised using state minimi-

[3] The method is based on converting a Petri net description into a single state–transition–graph, from which a sequential controller is next synthesised.

	Petri net		Equivalent STG	
name	#places	#trans.	#states	#edges
fcontr	70	72	-	-
smok4	32	24	331	662
phil10	30	20	81	590
pr_sp	21	22	39	65
zigzag	20	18	40	80
fstore	18	19	48	86
hash	15	17	17	21
graf	19	12	29	79
react	16	13	29	129
fgen	13	14	80	846

Table 2: Statistics of examples: parallel and sequential controller representation.

	Hierarchical colouring			Parallel decomposition		
name	area	delay	Tcpu	area	delay	Tcpu
fcontr	513648	11.9	37.8	701080	15.5	87.2
smok4	151728	7.7	28.4	156368	10.4	40.4
phil10	88160	6.4	6.9	120456	9.6	25.2
pr_sp	94656	10.7	10.9	139664	13.8	13.9
zigzag	188384	11.6	28.2	242672	13.2	26.4
fstore	125280	10.8	13.9	174000	16.7	21.9
hash	-	-	-	-	-	-
graf	133168	9.4	14.3	141520	13.1	19.6
react	86304	9.7	8.3	138736	12.2	13.1
fgen	76096	7.6	6.2	84448	8.7	9.1

Table 3: Synthesis results: hierarchical colouring method versus parallel decomposition method.

sation and state assignment programs STAMINA and JEDI and then SIS [6]. For each of the examples the *area*, the *delay*[4] of a final design, and the synthesis cpu time *Tcpu* were considered.

In Table 3 the synthesis results that were obtained by the use of the hierarchical colouring place encoding algorithm described in Section 6.2, and those that were produced by the use of the parallel decomposition implementation, presented in Section 6.1, are shown. In general, when area, delay, and cpu time of the synthesis are considered, results obtained using the hierarchical colouring algorithm are better, and some of them are considerably better. The benchmark **hash** is an example of the Petri net which is safe and live, but which cannot be coloured, thus the algorithms presented in this paper cannot be used in this case.

In Table 4 the synthesis results that were obtained by the use of the methodology described in [5] (reachability graph method), and the equivalent sequential implementation are presented. When reachability graph encoding approach is considered, comparison of the results reveals remarkable improvements, for more complex controllers, when using hierarchical colouring approach. When sequential approach is considered, comparison of the results reveals significant improvements in terms of area, delay and synthesis cpu time when using presented in this paper methodology. For the examples **fcontr** and **smok4** the code length exceeds the limit allowed by the state assignment program JEDI.

[4] Delay denotes the longest feedback time in the circuit.

	Reachability graph			FSM equivalent		
name	area	speed	Tcpu	area	delay	Tcpu
fcontr	725692	28.6	96.1	-	-	-
smok4	198592	8.2	1804.3	-	-	-
phil10	134094	12.7	45.3	915936	106.7	1870.0
pr_sp	109040	11.0	15.1	168896	30.2	61.9
zigzag	205552	16.1	31.8	309024	47.4	601.2
fstore	242208	19.6	26.6	317376	50.5	775.7
hash	15312	4.5	11.0	129920	16.7	23.3
graf	142448	10.9	20.7	151264	15.8	34.5
react	108112	9.2	14.4	119712	17.2	20.2
fgen	76560	7.8	7.1	299744	53.4	1152.7

Table 4: Synthesis results: reachability graph method versus FSM state assignment method.

8 Conclusion

A new methodology for parallel controller synthesis has been introduced. The new concept of using colours to detect and represent concurrency within a Petri net along with the set of colouring rules and a relevant algorithm has been presented. Experimental results clearly demonstrated the advantages of the presented approach, especially when complex controllers are to be synthesised.

The future work in this area includes the development of a more efficient algorithm for net colouring which can deal with all classes of safe and live nets; the set of colouring rules should be thoroughly investigated to facilitate decisions on why a net cannot be coloured.

References

[1] M. Adamski. Petri Net Decomposition. In *Zeszyty Naukowe WSI*, pages 1 – 23, Zielona Gora, 1982. WSI Publisher. In Polish.

[2] M. Adamski. Direct Implementation of Petri Net Specification. *7th International Conference on Control Systems and Computer Science*, pages 74–85, 1987.

[3] K. Jensen. Coloured Petri Nets. In W. Brauer, W. Reisig, and G. Rozengerg, editors, *Petri Nets: Central Models and Their Properties. Advances in Petri Nets 1986, Part I. Proceedings of an Advances Course*, volume 254 of *Lecture Notes in Computer Science*, pages 248 – 299. Springer-Verlag, 1987.

[4] T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):548–580, 1989.

[5] J. Pardey, T. Kozlowski, J. Saul, and M. Bolton. State Assignment Algorithms for Parallel Controller Synthesis. In *Proceedings of the IEEE International Conference on Computer Design*, pages 316–319. IEEE Computer Society Press, 1992.

[6] E.M. Sentovich, K.J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Sadanha, H. Savoj, P.R. Stephan, R.K. Brayton, and A. Sangiovanni-Vincentelli. *SIS: A System for Sequential Circuit Synthesis.* University of California, Berkelay, May 1992. Memorandum No. UCB/ERL M92/41.