A Tool for the Synthesis of Asynchronous Speed-Independent Circuits

Ondrej Gallo, Tomáš Nečas, Fedor Lehocki

Faculty of Electrical Engineering and Information Technology,
Slovak University of Technology,
Ilkovičova 3, 812 19 Bratislava, Slovak Republic
ondrej.gallo@stuba.sk, xnecas@is.stuba.sk, fedor.lehocki@stuba.sk

Abstract. The present work is devoted to the development of software tool written in Java for synthesis of asynchronous speed-independent circuits. A special type of Petri nets - Signal Transition Graph, was used for the synthesis. Using the algorithm based on the theory of regions, a logic function is derived from this graph. In order to reduce the complexity of the resulting asynchronous circuit the number of gates should be minimized by optimization of logical function with a Quine-McCluskey algorithm.

Keywords: Synthesis Asynchronous Speed-Independent Circuit, Signal Transition Graph, State Graph.

1 Introduction

Asynchronous circuits have gained in importance along with the expansion of the production of high integration circuits. By that time, mostly synchronous systems were designed. Decreasing the dimensions and increasing the operating frequency, however, made the synchronisation of individual function blocks on a chip more difficult. The time delay is generally caused by the increased cycle of timing signal. This delay (for the individual function blocks) differs locally to such a degree that it results in the synchronisation failure of the individual subsystems and the malfunction of the circuit. This issue can be solved by adding a special regulation circuit providing a constant clock frequency in the whole chip. Such a circuit would occupy relatively much space on a chip (approx. 10%) and consume much power (approximately 40% of the input power). This would result in the increasing cost and energy demand of such chips. The application of asynchronous circuits provides a different approach. These circuits do not need a clock signal because their operation is controlled by events and not by time. As no clock signal generation is required and no regulation circuitry is needed, such circuits are smaller and consume less energy. The only issue that is common for both synchronous and asynchronous circuits is the presence of so-called hazardous states.

The aim of this work is to provide a tool for the synthesis of asynchronous circuits that generates a circuit diagram as a result of the circuit behavior. The field of digital circuit synthesis is complex and provides many solution approaches. One of them is represented by the application of the Petri nets formalism as a description tool for the behavior of an arbitrary digital circuit. Application of algorithms for STG synthesis

Recent Advances in Petri Nets and Concurrency, S. Donatelli, J. Kleijn, R.J. Machado, J.M. Fernandes (eds.), CEUR Workshop Proceedings, ISSN 1613-0073, Jan/2012, pp. 207–211.

(e.g. based on the region theory) a logical function is derived (using Quine-McCluskey algorithm) and presented in the form of a circuit diagram.

2 Basic definitions

The signal transition graph (STG) [1] is a specially labelled Petri net that is defined as a heptad $(P, T, F, M_0, N, s_0, \lambda_T)$, where P is set of places, T is the set of transitions, F is the flow relation $F \subseteq (P \times T) \cup (T \times P)$, M_0 is the initial marking of Petri net, $N = I \cup O$ is a set of signals, I is a set of input signals and O is a set of output signals. s_0 represents the initial value for each signal in its initial state and λ_T: $T \rightarrow N \times \{+, -\}$ is the transition labelling function.

The state graph SG [1] is basically a reachability graph for the STG. Compared with the reachability graph, states in the state graph are labelled more functionally. It can be designed only in the case that the reachability graph is bounded, i.e. the number of reachable labels is finite. This graph in its final form is being used for the derivation of logical functions in the synthesis of asynchronous circuits. The state graph is formally defined as a triplet (S, δ, λ_S). S is a set of all the states, $\delta \subseteq S \times T \times S$ is a set of state transitions and $\lambda_S : S \rightarrow (N \rightarrow \{0,1\})$ is a function of state labelling.

Each state is labelled by a binary vector $(s(0), s(1), ..., s(n))$, where each signal s(i) $i \in \{0, 1, ... n\}$ can acquire values from the set $\{0, 1\}$. Although for a better illustration of the graphical representation of possible states, individual signals can also acquire possible values from the set {0, 1, R, F}. This form of labelling also provides information as to whether the respective signal is excited or not. The excited signal represents a change in its value from 0 to 1 or vice versa. This is illustrated by the characters R or F. The R character denotes the signal value in the SG being equal to 0 but where its value has changed to 1 in the subsequent state s(i). The signal is able to reach the next state because the respective transition could be started. This transition is labelled u_i+ by using the labelling function $\lambda_T(t)$. Similarly, the F character denotes the signal value being equal to 1 and to 0 in the subsequent state. The excited state can be formally defined as follows:

$$\exists(s_i, t, s_j) \in \delta \,.\, \lambda_T(t) = u_i+ \vee \lambda_T(t) = u_i-. \qquad (1)$$

Each signal is labelled in the STG as a transition representing the front edge of the signal u_i+ or the decay of the signal u_i-. As stated hereinabove, STG is a special Petri net fulfilling the following properties:

Input free-choice: The starting sequence of the respective transition (signal) is controlled by the so-called mutual exclusion of input signals. It is indicated by a special transition in the transition graph.

Boundedness: This property of the transition graph provides that the state graph (to be defined later in the text) shall acquire a finite number of states. The transition graph is single-bounded when just a single label is in each place. The transition graph must be a safe Petri net.

Liveness: The STG must be free from deadlocks.

State consistency: All transitions providing front edge and a decay of the signal must strictly alter between + and – in any execution of the STG.

Complete state coding (CSC): This property is checked in SG. This means that a pair of states of S has a unique state coding defined by the labelling function λ_S, or it does not have the unique state coding but it does contain the same output signal excited in each state. If this property is not fulfilled, a new signal or signals are to be inserted into the SG.

Persistency: If a transition is enabled, it is fired and the label is transferred from the place ahead of the transition to the place or places behind, provided that this start shall not be deactivated by another transition. This property must be provided for the input and internal signals. The persistency of the input signals must be ensured by the environment of the designed circuit.

3 Software description

The model of the circuit behavior represents an input for our tool (ACDesigner [2]). In the process of the circuit's design, designers mostly prefer the timing diagram of the circuit's behavior. In this diagram, all input and output signals, and causalities between the signals, are recorded. Thus if a timing diagram is available a special Petri net (Signal Transition Graph) can be formed, representing the input for our software. By means of the algorithm presented by Cortadella et al. [3], we derive an asynchronous speed–independent circuit whose behavior is characterized by the STG. Speed independence is a property ensuring the correct circuit operation considering that all logical gates have unbounded delay [1]. The graphical representation of the circuit synthesis process is shown in Fig. 1.

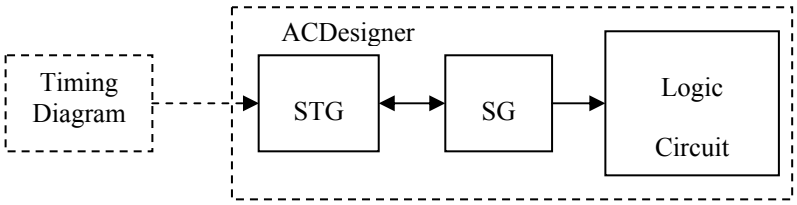

Fig. 1. Graphical representation of the particular steps in synthesis.

After loading STG in the software, the required properties (boundedness, consistency, persistency, input free-choice, liveness, and complete state coding) are verified and the SG is derived. In some cases, the state graph does not comply with the CSC property. The solution of this issue is based on the insertion of new states into the SG according to the algorithm published in [4] and [5]. An example of the SG fulfilling the CSC (new signal *csc0*) property is shown in Fig. 2b. SG is an intermediate product in the process of synthesis and it is not visualised in our tool.

The state graph is divided into regions and intersections of some regions. The best regions covering conflicts are selected (for each iteration only one conflict is selected). After selection of the conflict the environment is added (consisting of

neighbor regions) and the cost function is calculated [4]. This function serves as a basis to determine direction of the SG search to find the most suitable place for the insertion of the new signal. After reduction of the number of possible insertion points, the best one is selected for which the least complicated circuit is formed. This iteration should also be repeated several times, because single signal insertion may not solve all the conflicts, and new ones could even appear. The algorithm, however, converges to a solution that was confirmed experimentally in the reference [4]. If SG fulfils the CSC property, a logic function is derived for each output and new (internal) signal. The minimisation of the logic function, which is required with respect to the resulting number of gates, is the next step. The Quine-McCluskey algorithm [6] was applied in our software to minimise the logic function. At this level, requirements can also be laid on the application of particular gate types. In our case, we focused on the use of the standard 2-input gates of AND, OR type and on the NOT gate.

A Petri net in PNML format [7], which can be designed by using other tools, e.g. PNEditor [8] or VipTool [9], is the input file in this ACDesigner version. The net must contain a label indentifying the signal type (input or output signal). In a Petri net, signals are represented by transitions. Therefore, they must be labelled as the input or output signals. In Fig. 2a, a demonstration of the Petri net is shown including the labelling of transitions with the keywords "in" and "out". The name is arbitrary provided that it ends either with the + or − sign. For the designer, it is an indication of the signal change from 0 to 1 (+ sign) or from 1 to 0 (− sign), respectively.

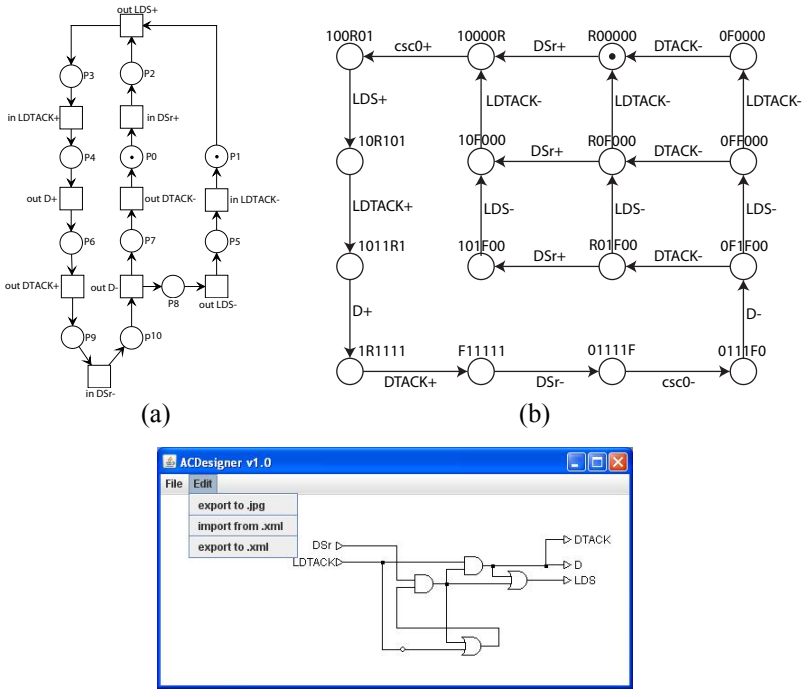

Fig. 2. (a) Input format of Petri nets - STG, (b) SG fulfilling the CSC property, binary vector is <*DSr*, *DTACK*, *LDTACK*, *LDS*, *D, csc0*>, (c) resulting logical circuit.

The software output is shown in Fig. 2c. The user can save the generated circuit in two formats, either as XML or as JPG.

4 Conclusion

The algorithm of the synthesis of asynchronous speed-independent circuits was implemented for the first time in the petrify tool [10]. Due to the absence of the graphical visualisation of the resulting circuit, we decided to programme our own tool, which would include this functionality. The software is written in Java ver. 1.6, which ensures platform independence. The implementation of more methods of logical circuits' synthesis will be the next step. These methods should take advantage of a non-standard memory element (C-element). The utilization of this element may prevent the occurrence of several hazardous states. Moreover, it has a very fast memory and can be easily implemented on a chip. The extension of support to multiple input and output formats is another important step in our development. One of the possible input formats could be the timing diagram, which is easy understandable to many designers. If a logical circuit is designed, it must be verified by the simulation process. This option is provided by another professional simulation software (e.g. Protel, PSPICE, and CADENCE) that requires its own specific file format. From that point of view, our tool will be extended with respective additional functionalities.

References

1. Chris J. Myers, Asynchronous Circuit Design, John Wiley & Sons, July, 2001.
2. ACDesigner: A tool for synthsis of asynchronous circuits: http://gordan.matmas.net/acdesigner/
3. J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno and A. Yakovlev, Hardware and Petri Nets: Application to Asynchronous Circuit Design, Application and Theory of Petri Nets 2000, Lecture Notes in Computer Science vol. 1825, pp. 1-15, Springer Verlag, June 2000.
4. J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno and A. Yakovlev, A region-based theory for state assignment in speed-independent circuits, IEEE Trans. on CAD, vol. 16, no. 8, August 1997, pp. 793-812.
5. J. Cortadella, M. Kishinevsky, L. Lavagno and A. Yakovlev. Deriving Petri nets from finite transition systems. Universitat Politecnica de Catalunya, Tech. Rep. UPC-DAC-96-19, June 1996.
6. Brian Holdsworth, R. Clive Woods, Digital logic design 4th edition, Newnes, 2002.
7. The Petri Net Markup Language: www.informatik.hu-berlin.de/top/pnml/
8. PNeditor: A tool for modeling Petri nets. https://pneditor.matmas.net/
9. VipTool: A tool for modeling Petri nets. http://viptool.ku-eichstaett.de
10. Petrify: A tool for synthesis of Petri nets and asynchronous controllers. http://www.lsi.upc.edu/~jordicf/petrify.