An Effective Solution of Benchmarking Problem
FSM Benchmark Generator and Its Application to Analysis of State Assignment Methods

Lech Jóźwiak, Dominik Gawłowski and Aleksander Ślusarczyk

Eindhoven University of Technology
Faculty of Electrical Engineering
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
L.Jozwiak@tue.nl

Abstract

This paper focuses on the benchmarking that is one of the main issues of the synthesis method and tool development, analysis, characterization and evaluation. To solve several serious problems related to the usage of practical industrial benchmarks, we developed and implemented an FSM Benchmark Generator (BenGen). BenGen makes us possible to efficiently construct FSMs with various known characteristics, including FSMs representative to various typical industrial application areas, greatly reduces the necessity of having the actual industrial benchmarks, and enables research, comparison, evaluation and fine tuning of circuit synthesis methods largely independent of the industry, and much more effectively and efficiently than having only some industrial benchmarks. Using a large set of benchmarks generated with BenGen, we performed an experiment aiming at the effectiveness characterization of some popular academic and industrial FSM state assignment approaches. The experimental results enabled us among others to demonstrate that the pragmatic assignment approaches used commonly in today's commercial tools are only effective for some special classes of circuits or not effective at all.

1. Introduction

Microelectronic circuits and systems have important applications in virtually all fields of human activity, and more and more influence the performance of society and our everyday life. Many of them, embedded in systems for critical applications, impose extremely high quality requirements. Progress in microelectronic technology is extremely rapid and results in unusual silicon and system complexity. It is outstripping the designers' abilities to make use of the created opportunities. In this situation, quality-driven design, and development and application of new more suitable design methods and tools are of crucial importance for the modern system industry.

The circuit synthesis level is located somewhere in the middle of the top-down system design trajectory, between the architectural and physical levels. The main task of circuit (logic) synthesis is the translation of the symbolic, functional description of an architecture module into the binary logic description of a network of circuit components that realizes the behavior of the module. This network must satisfy specific constraints and optimize certain objectives. In other words, logic synthesis decomposes a given architecture module in a (near-) optimal network of logic building blocks that implements the module's function.

For large, complex, multi-aspect and "irregular" circuit synthesis problems, it is impractical or even impossible to generate and examine all possible solutions. It is necessary to find only the most promising solutions, when using heuristic search, and to choose the best of these. One of the basic requirements for usage of the heuristic search methods in practical applications of circuit and system synthesis is their adequate effectiveness and efficiency. The **effectiveness** is understood here as the **robustness of a method in the sense of steadily delivering of the high-quality solutions in the whole range of its possible input data**. This corresponds to the construction of the high-quality circuits for the whole range of possible functions and values of the structural and parametric requirements. The "high quality" does not necessary mean strictly optimal, but steadily optimal or close to optimal. The efficiency is understood in the sense of a short enough computation time and practical memory usage.

The automatic hardware synthesis started in 1970s, with automatic physical and layout synthesis, and continued in 1980s and 1990s, with automatic logic and RTL synthesis. Subsequent extension of the automatic synthesis to the system levels surpassed many unsolved important theoretical and methodological problems of high practical importance in logic synthesis.

In this paper, we will demonstrate that some popular contemporary circuit synthesis approaches are not robust, and often produce circuits that are far from the best possible. Thus, further research and development in circuit synthesis are necessary. **This paper focuses on the benchmarking** that is one of the main issues of the synthesis method and tool development, analysis, characterization and evaluation.

2. The problems of benchmarking.

The circuit and system synthesis methods and tools can be thoroughly analyzed, characterized, compared, and evaluated only through applying them to some sets of (practical or hypothetical) designs (benchmarks), and analyzing the synthesis results produced. Unfortunately, the **standard synthesis benchmarks are not representative** enough, and there are several **serious problems related to the usage of practical industrial benchmarks**:

– the industry does not want to make their benchmarks accessible, motivating that these are their or their clients' important designs, and thus, they cannot be disclosed;

– having even a large set of the actual industrial circuit specifications, it is difficult to see how big and divergent parts of the actual circuit space do they cover: are the circuits very different or almost the same; do they cover the circuit space regularly or are just cumulated in some particular regions; what are their characteristics;

– to be representative, the industrial benchmarks have to involve very different functions typical to diverse application fields; they have thus to be collected from many industrial partners, but this results in a multitude of their formats, standards and/or representations.

It should be stressed that usage of a non-characterized set of (industrial) benchmarks to research, compare or evaluate the synthesis methods or tools can very easily lead to completely wrong conclusions. In particular, the thousands benchmarks used can be almost the same or trivial, and from the fact that a method or tool worked excellently for all of them it can completely not be concluded that it will work reasonably for a one more circuit, but different.

To resolve the problems related to the usage of practical industrial benchmarks in research, comparison and evaluation of circuit synthesis methods, we developed and implemented an FSM Benchmark Generator (*BenGen*). The benchmark generator makes us possible to efficiently construct FSMs with various known characteristics. Among others, FSMs representative to various typical industrial application areas can be generated using *BenGen*. Availability of the benchmark generator greatly reduces the necessity of having the actual industrial benchmarks, and enables research, comparison and evaluation of circuit synthesis methods largely independent of the industry, and much more effectively and efficiently than having only some industrial benchmarks.

3. FSM Benchmark Generator (*BenGen*).

The FSM benchmark generator *BenGen* makes us possible to efficiently construct FSMs with various characteristics. This includes:

– FSMs with different number of states and various transition patterns between the states (e.g. chains of states with forward and/or backward transitions, loops, conditional "case" structures etc. and their combinations);

– FSMs with different numbers of inputs and outputs, and different proportions between the next-state and output logic (state-dominated, balanced or output-dominated), as well as, between the primary-input and state-input (input-dominated, balanced, state-dominated), and their mixtures;

– FSMs with various dependence of particular transitions and output variables on the number of inputs and input conditions;

– completely, incompletely and weakly specified FSMs.

This also includes FSMs representative to various typical industrial application areas, as for instance, having typical structure of controllers from various application areas, or representing various sequential data-path circuits (e.g. counters).

BenGen makes us possible an efficient FSM construction, but also modification of the constructed or industrial FSMs, and very precise "fine-tuning". This last feature is extremely useful in sensitivity analysis of EDA methods and tools to the changes in their input data, i.e. changes in the FSM characteristics.

The FSM benchmarks from this generator enable us the following:

– analysis of a wide spectrum of design cases and problems related to the FSM architecture and logic synthesis, in the space spanned by various FSM structures and different implementation options, and in the light of various optimization constraints and objectives,

– thorough and very precise testing of EDA methods and tools,

– reliable comparison and quality evaluation of different EDA methods and tools,

– extensive research and precise characterization of EDA methods and tools in relation to FSMs with various known characteristics (e.g. state/input/output dominated, branch/chain/ loop dominated, completely/incompletely/ weakly specified, small/medium/large etc.)

– very precise sensitivity analysis of EDA methods and tools to changes in the FSM characteristics.

Consequently, the **FSM benchmark generator is a very useful tool. It enables us to perform the research, comparison, evaluation, characterization and fine-tuning of the ADA methods and tools relatively independent of the industrial benchmarks, and much better than having only the industrial benchmarks**. Of course, the generated benchmarks do not completely replace the actual industrial benchmarks, but very much reduce the necessity of using them. The industrial benchmarks can still be useful to ensure that the FSM types represented by them are covered by the generated benchmarks, and to perform the final confirmation checks.

BenGen has **two work modes**: batch and interactive mode. In the **batch mode**, the parameters of the FSM to generate are supplied in a script file. These parameters include: the number of inputs and outputs of the FSM; the number and length of the FSM's branches; the characteristics of a branch, such as the number of backward transitions or loops; the number of inputs and outputs active for a given branch, etc. Most of these parameters can be specified in the form of a probability distribution to randomize their values in the specific instances of the generated FSMs. As a result, in the batch mode BenGen can be used to easily generate large sets of FSM benchmarks with certain characteristics using the same script file. The script file can be easily modified to generate a next batch of somewhat different FSMs. The **interactive mode** of *BenGen* provides more control over the generation process. The user can interactively enter any of the parameters available in the batch mode. In addition to that, operations allowing modifications of single branches, or transitions are provided. This makes the interactive mode *BenGen* an ideal tool for fine-tuning of the generated or industrial FSMs for the specific characteristics, required for instance to check the behaviour or sensitivity of a method or tool in relation to a certain aspect.

In both modes, *BenGen* takes away the burden of tedious specification of single transitions, or checking the consistency of the constructed FSM. Instead, the user can focus on specifying high-level characteristics of the machine. Given the global machine's characteristics, BenGen generates the required number of state chains, with the requested number of states, and appropriate backward transitions between and self-loops in the states, if needed. Given the state-transition behaviour defined in abstract terms, *BenGen* generates the input conditions for particular transitions, etc. In this process not only does it consider user's requirements concerning the active inputs for a given branch, but also ascertains that the machine is consistent, i.e. distinct transitions have disjoint input conditions and all possible input conditions are specified (for completely specified FSMs). The generator also determines the output values for the transitions, taking into account the outputs active for a given branch.

With the above characteristics, BenGen is a very useful tool allowing efficient generation of an arbitrary number of various sorts of well-characterized FSM benchmarks, with the minimum effort of the user. On the other hand, it also enables fine-grained control over the generation process and editing of the generated or industrial FSMs.

Currently, *BenGen* is extensively used in our research related to the analysis of design problems and usage of various EDA methods and tools in controller architecture and circuit synthesis for implementation with modern FPGAs and re-configurable SoC platforms. In particular, we used *BenGen* to perform analysis of problems related to the existing academic and commercial methods and tools for FSM state assignment. A part of this research is discussed in the following sections of this paper.

4. State Assignment: Approaches and Quality Evaluation

4.1 State assignment

The main task of logic synthesis is to translate the symbolic functional description of an architectural module into a binary-logic network structure consisting of binary signals and operators. The circuit structure, being the result of such a translation, must fulfill certain requirements and must be (near-)optimal for some objectives. In most cases, a certain tradeoff of area, speed and power-dissipation is optimized.

For sequential logic, this translation process consists in the simplest case of the following two main sub-processes:

– assignment of an adequate binary representation for the FSM's symbolic internal states, and

– optimal synthesis of the FSM's combinational binary-logic component that results from the state assignment.

In the sequel, we will deal with the FSM state assignment, but its results will be estimated through the combinational component synthesis.

Definition 4.1. A *sequential machine/finite state machine* (FSM) M is an algebraic system defined as follows: $M = (I,S,O, \delta, \lambda)$

where:

I - a finite nonempty set of inputs,

S - a finite nonempty set of internal state,

O - a finite set of outputs,

δ - the next-state function: δ:S x I $\rightarrow$ S
λ - the output function:
λ: S x I $\rightarrow$ O (a Mealy machine), or
λ: S $\rightarrow$ O (a Moore machine)

As opposed to inputs and outputs of a sequential machine, which are usually pre-assigned, because they represent some external world signals, the internal states are initially in a symbolic form in most cases.

State assignment consists of choosing a binary representation for the symbolic internal states of a sequential machine, so that the resulting binary logic is optimal for a given objective.

Hardware implementation of a sequential machine requires two sorts of components: combinational logic (implementing the binary representation of the next-state function δ and output function λ), and binary memory elements (implementing the state memory). The choice of codes for the states finally decides the binary-level structure of both components, and thus, greatly affects all or most of their main characteristics (area, speed, power dissipation). The advantage of finding an optimal assignment is significant [3][4]. Therefore, optimal state assignment is one of the most important problems in the FSM synthesis. The problem of finding an optimal state assignment is however computationally complex (NP-hard). In a strict sense, it has never been solved, except for exhaustive search that is impractical or impossible for large machines, even with a computer. In such a situation, some approximate heuristic assignment approaches must be used.

4.2 Heuristic assignment approaches

For large FSMs, it becomes necessary to find some heuristics to find a high quality assignment with reasonable computation resources.

Heuristic state assignment approaches can be subdivided into two main categories:

– **structural** (constructive) **approaches**, that construct (near-)optimal assignment using some knowledge on the internal structure of a sequential machine to satisfy the optimization objectives; and

– **statistical** (generative) **approaches** that generate and check the assignments but do not use any information about the internal structure of a sequential machine.

The most prominent examples in the first category are SECODE [3], Jedi[7], MAXAD[6] and NOVA[8]. The best-known approaches of the second category are the best-random approach and the generative genetic algorithms (in contrast to constructive) that do not stepwise construct state assignments, but generate, test, modify and combine some complete assignments [1][2].

The best-random approach, which consists in choosing the best from n randomly generated assignments, is a trivial heuristic approach, because it is in fact an unguided "trial and error" method. Genetic algorithms, on the other hand, create consecutive “generations” of solution as a mutation or combination of the best solutions from the previous generation. The statistical character of this approach consists in the semi-random generation of the initial population, selection of parents for the next generation of solutions, and mutation of the solutions to introduce some extra variation to the population.

Yet another **pragmatic approach** used commonly in today’s commercial tools is to apply an assignment that is “known” as possessing some “nice” properties. The mostly used encodings in this category are: one-hot, Grey-code and natural binary encoding.

4.3. State assignment solution space and probabilistic quality measures

An extensive analysis of the state assignment solution space was presented by us in [1][2], and therefore, only the most important definitions and conclusions are recalled here. The distribution of solutions for state assignment can be well estimated by a normal (Gaussian) distribution. For non-trivial machines, and especially for medium (from 9 through 32 states, and a substantial number of inputs or outputs) and large machines (with more than 32 states, and a substantial number of inputs or outputs), many factors influence the state assignment quality, but none of them is a deciding factor in separation. In this situation, that is typical to all natural or technical processes that show a normal distribution, there are many more constellations of factors that lead to medium value assignments (different mixtures of "good" and "bad" factors, arrangements of medium-value factors with "good" and "bad" factors, etc.) than constellations that lead to very good or very bad assignments (arrangements of only the best or only the worst factors). Of course, for larger machines, on average, more factors decide an assignment's quality, there are more constellations of these factors, while each of them in isolation can decide less about the quality of the assignment. Therefore, the estimation of the distribution of solutions by the normal distribution is on average better for larger machines than for smaller ones. Very small machines show substantial deviations from the normal distribution. Numerous examples of typical empirical distributions are given in [3][4].

A *normal distribution N (m, σ)* is defined as a distribution with the probability density function:

$$f(x) = (\frac{1}{\sigma\sqrt{2\pi}}) * e^{-(x-m)^2/(2\sigma^2)}$$

where: m - expected value; σ - mean square deviation.

The cumulative distribution function F(x)= P(t ≤ x) for a normal distribution is given by:

$$F(x) = \begin{cases} 0{,}5 + \Phi(\frac{x-m}{\sigma})\, for\, x > m \\ 0{,}5 - \Phi(\frac{x-m}{\sigma})\, for\, x \le m \end{cases}$$

where: $\Phi(x)$ is an integral from the probability density function of the normalized Gaussian distribution *N(0,1)*in the integration range *<0,x>*. The values of $\Phi(x)$ are provided in tabular form in mathematical guides (also at Internet).

We must remember that the random variable *X* describing the quality of a solution is a discrete variable. So, we must calculate $P(X=X_i)$ as an integral from the probability density function *f(x)* in the integration range $<X_i-0.5, X_i+0.5>$:

$$P(X = X_i) = \int_{X_i-0,5}^{X_i+0,5} f(x)dx =$$

$$= F(X_i + 0{,}5) - F(X_i - 0{,}5)$$

and $P(X \le X_i) = F(X+0{,}5)$

To calculate the probability of a number of successes ***k*** in ***n*** random choices, the **binominal distribution** can be used:

$$P(S_n = k) = \binom{n}{k} p^k (1-p)^{n-k}\ and$$

$$P(S_n > 0) = 1 - P\binom{n}{k} p(S_n = 0) = 1 - (1-p)^n \,,$$

where: p – the probability of success for one random choice.

For each of the FSMs analyzed in this paper, we estimated m and σ based on the experimental results form 20 randomly generated assignments.

Since a strictly optimal assignment is relatively unknown for large machines, it is impossible to define the terms "good" or "suboptimal" for assignments in relation to the optimal assignments. Therefore, we proposed some probabilistic quality measures for assignments and assignments methods that are based on estimating of the distribution of assignments in the solution space with a normal distribution.

Definition 4.2: An *assignment is good* if and only if its quality X_G is no worse than the average quality: $X_G \le m$ (*i.e.* $P(X \le X_G) \le 0.5$).

Definition 4.3: An *assignment is near-optimal* if and only if the probability of randomly generating an assignment, with a quality X no worse than X_S of the near-optimal assignment, is not higher than *0.05* ($P(X \le X_N) \le 0.05$).

Definition 4.4: An *assignment is unique* if and only if the probability of randomly generating an assignment, with a quality X no worse than X_U of the unique assignment, is not higher than *0.005* ($P(X \le X_U) \le 0.005$).

Definition 4.5: A *state assignment method is good (near-optimal, unique)* if and only if it computes good (near-optimal, unique) assignment in most cases.

4.4. Difficulties with the statistical approach

Since the distribution of state assignment solutions for medium and large machines is close to a normal distribution, assignments generated by statistical methods will be concentrated around the average quality assignments.

For instance, by ten random choices, the probability of obtaining a solution that is no worse than the average, is *0,999*, but the probability of obtaining a near-optimal solution X_N is *0,4012*(less than *0,5*!) and the probability of obtaining a unique solution X_U is only *0.0489*. Thus, assignments better than average can be generated with high probability by statistical methods, but the probability of generating very good assignments is low, while, generating unique assignments is very unlikely.

In contrast, the structural methods, that construct assignments using relevant information, are able to construct near-optimal or even unique assignments with good probability [3][4][6][7][8].

Moreover, in order to choose the best from n random assignments, the resulting multiple output combinational logic must be synthesized for each encoding and the synthesis results compared. The computation time for one run of the combinational synthesis program is comparable to the computation time for the structural state assignment program, but the synthesis program must be run n times for random encoding. This problem is even more pronounced for methods based on generative genetic algorithms. To find a good solution, these methods usually need to evaluate tens or hundreds of generations, each composed of tens of solution. For each of the solutions, the quality function, e.g. the size of the synthesized circuit implementation, needs to be computed. In consequence, the generative genetic algorithms deliver solutions of comparable quality orders of magnitude slower than the structural methods.

The difficulty of evaluating the quality of an assignment, in combination with the normal distribution of solutions, it is the main reason why the structural approaches can be more efficient than statistical ones.

The quality of a statistical approach is restricted mainly by the structure of the problem itself (normal distribution of solutions, time-consuming evaluation of the solutions' quality). The quality of a structural approach is restricted mainly by the capacity of the human brain to think heuristically. It is decided by the ability to effectively and efficiently use the relevant knowledge on all the important factors influencing the assignment quality to control the constructive search process.

5. Experimental results

Using a large set of benchmarks generated with *BenGen*, we performed an experiment aiming at the effectiveness characterization of some popular academic and industrial FSM state assignment approaches. In this section, the summary data corresponding to a part of this experiment are presented.

This **characterization experiment involved** among others:

– **1-hot, gray** and **natural binary encoding**: representing **popular, pragmatic industrial assignment approaches** used in today's commercial tools, and

– **jedi**: one of the most advanced academic tools representing the **constructive structural state assignment approach**.

The encoded machines were synthesized for the LUT-FPGA architecture using two combinational synthesis methods: SIS and IRMA2FPGA. In this brief paper, we only present the synthesis results from SIS related to the circuit area expressed in terms of the number of 5-input LUTs, because the results from IRMA2FPGA support the same general conclusions.

In the experiment, **250 FSMs** were used that exhibit various characteristics typical to FSMs encountered in different real-life applications. We identified a number of typical schemes of sequential behavior and for each scheme, generated a set of benchmarks of different size and with differing proportions of input, state and output logic. For instance, the sequential behavior schemes included: a single loop of states (typical for a counter or simple sequencer); a number of loops starting from a common initial state (typical for a controller realizing a few different control programs for different operation modes); a single loop with sub-loops attached to states along the main loop (a main control "program" with "subroutines" - the subroutine loops may have their own sub-subroutine loops); and more complicated cases of

Table 1. Experimental results – summary

jedi	b	g	n	u	quality
small	6	37	18	12	g
medium	3	7	17	88	u
large	0	1	0	61	u
input dom.	1	10	15	58	u
balanced	5	31	19	36	n
state dom.	3	4	1	67	u
output dom.	2	20	8	69	u
balanced	4	18	23	49	u
state dom.	3	7	4	43	u
Total	9	45	35	161	u

1-hot	b	g	n	u	quality
small	53	20	0	0	b
medium	50	25	7	33	g
large	8	2	3	49	u
input dom.	15	32	5	32	g
balanced	64	10	3	14	b
state dom.	32	5	2	36	n
output dom.	32	11	4	52	u
balanced	49	24	4	17	b
state dom.	30	12	2	13	b
Total	111	47	10	82	g

gray	b	g	n	u	quality
small	26	36	7	4	g
medium	5	37	20	53	n
large	0	3	0	59	u
input dom.	10	28	17	29	n
balanced	20	39	9	23	g
state dom.	1	9	1	64	u
output dom.	10	25	5	59	u
balanced	20	36	17	21	g
state dom.	1	15	5	36	u
Total	31	76	27	116	n

bin	b	g	n	u	quality
small	24	42	3	4	g
medium	13	39	18	45	n
large	0	4	2	56	u
input dom.	12	32	10	30	g
balanced	21	44	10	16	g
state dom.	4	9	3	59	u
output dom.	10	27	9	53	u
balanced	20	44	12	18	g
state dom.	7	14	2	34	u
Total	37	85	23	105	n

sequential behavior. Within each scheme, we varied proportions of backward transitions and state self-loops within loops and branches generated. This way a representative benchmark set has been created.

We evaluated the effectiveness of different state assignment methods on particular benchmarks, by computing the probability of achieving the result produced by the method for a given benchmark using the random encoding. For this purpose, we assumed a normal distribution $N(m, \sigma)$ of the random encoding results for each benchmark regarding area, with the parameters m and σ of the distribution derived from the results of 20 random min-length encodings. Based on the probability of achieving the particular encoding result on random, the results were categorized according to the classification proposed in Section 4.3. Due to space limitations, we did not include the large tables with complete results for all FSMs in this brief paper. Instead, we categorized the machines according to three criteria: the size, the proportion of the number of primary input bits to the number of state bits, and the proportion the number of primary output bits to the number of state bits. The size criterion divides FSMs into *small* (max. 8 states), *medium* (9 to 32 states) and *large* (more than 32 states). The proportion of the number of primary input/output bits to state bits categorizes the FSMs as *input/output dominated* if the number of input/output bits is 50% larger than the number of state bits, *state dominated* if the number of state bits is larger than the number of input/output bits and *balanced* otherwise. In Table 1, the numbers of bad (*b*), only good (but not near-optimal or unique) (*g*), only near-optimal (but not unique) (*n*) and unique (*u*) encodings are presented produced by each assignment approach for particular classes of benchmarks and for all benchmarks (*Total*), as well as, the resulting quality of each state assignment approach (*quality*) for each FSM class and for the total benchmark set.

6. Conclusions

This paper focused on the benchmarking that is one of the main issues of the synthesis method and tool development, analysis, characterization and evaluation. To resolve several serious problems related to the usage of practical industrial benchmarks, we developed and implemented an FSM Benchmark Generator (*BenGen*). *BenGen* makes us possible to efficiently construct FSMs with various known characteristics, including FSMs representative to various typical industrial application areas. The usage of *BenGen* to perform analysis of problems related to the academic and commercial methods and tools for FSM state assignment fully confirms that its availability greatly reduces the necessity of having the actual industrial benchmarks, and that this is a very efficient tool. The benchmarks from *BenGen* enable research, comparison and evaluation of circuit synthesis methods largely independent of the industry, and much more effectively and efficiently than having only some industrial benchmarks.

In particular, using a representative set of 250 benchmarks generated with *BenGen*, we performed an experiment aiming at the effectiveness characterization of some popular academic and industrial FSM state assignment approaches. In Section 5, the summary data corresponding to a part of this experiment are presented. From this experimental data, it follows that:

– **only the constructive structural state assignment approach, represented by *jedi*, is unique and robust**, and is able to robustly deliver the unique results for all sorts of FSMs, except for the small FSMs (observe however that for small FSMs the normal distribution is degenerated);

– **Gray and binary encodings are near-optimal, but not robust in general**; however, they are unique and robust for the large state-dominated (on the input side) or output-dominated FSMs, and only good for the small or balanced FSMs (i.e. only somewhat better than random). Additionally, binary encoding is only good for the input-dominated FSMs;

– **1-hot is the worst encoding method from all the researched: it is only good and not robust in general**; nevertheless, it is able to reasonably robustly produce the unique results for large, and particularly, large output-dominated FSMs.

Summing up, using *BenGen* we demonstrated that the **pragmatic FSM state assignment approaches used commonly in today's commercial tools are only effective in some special cases** (as the Gray and binary encodings for the large state- or output-dominated FSMs or the 1-hot encoding for the large output-dominated FSMs). In particular, we clearly shown that **the common opinion on the exceptional high quality of the 1-hot encoding in its application to the FPGA-targeted circuit synthesis is just a myth**.

Acknowledgements

The concept and initial design of the FSM Benchmark Generator *BenGen* have been developed by Lech Jóźwiak in his private capacity. The implementation of *BenGen* and subsequent research have been performed at the Faculty of Electrical Engineering of the Eindhoven University of Technology, and were partly supported by the Technology Foundation STW, applied science division of NWO and the Technology programme of the Ministry of Economic Affairs.

References

[1] A.E.A Almaini, et al.: State Assignment of Finite State Machines using a Genetic Algorithm, IEE Proc. on Computers and Digital Techniques, 1995, pp.279-286.
[2] S. Chattopadhyay, P. Pal Chaudhuri: Genetic Algorithm Based Approach for Integrated State Assignment and Flipflop Selection in Finite State Machine Synthesis, Proc. of Int. Conf. on VLSI Design, 1997, pp. 522-527.
[3] L. Jóźwiak: Efficient Suboptimal State Assignment for Large Sequential Machines, EDAC-European Design Automation Conference, Glasgow, Scotland, March 12-15, 1990, IEEE Computer-Society Press, pp. 536 - 541.
[4] L. Jóźwiak: An Efficient Heuristic Method for State Assignment of Large Sequential Machines, Journal of Circuits, Systems and Computers, vol. 2, no.1, 1992.
[5] L. Jóźwiak: Quality-driven Design in the System-on-a-Chip Era: Why and How?, Journal of Systems Architecture, April 2001, ISSN 1383-7621/01165-6074, Elsevier Science, Amsterdam, The Netherlands, 2001, Vol 47/3-4, pp 201-224.
[6] L. Jóźwiak, A. Ślusarczyk, A. Chojnacki: Fast and Compact Sequential Circuits for the FPGA-based Reconfigurable Systems, Journal of Systems Architecture, ISSN 1383-7621, Elsevier Science, Amsterdam, The Netherlands, Vol. 49, No 4-6, September 2003, pp. 227- 246.
[7] B. Lin, A.R. Newton: Synthesis of Multiple Level Logic from Symbolic High-Level Description Languages, Proc. of IFIP Int. Conf. on VLSI, 1989, pp.187-196.
[8] T. Villa, A. Sangiovanni-Vincentelli: NOVA: State Assignment of Finite State Machines for Optimal Two-Level Logic Implementation, IEEE Trans. on CAD, 1990, pp. 905-924
[9] E. Sentovich et al.: SIS: A System for Sequential Circuit Synthesis, Memorandum No. UCB/ERL M92/41, University of California, Berkeley, 1992.