UNLV | University Libraries
University of Nevada, Las Vegas

Electrical and Computer Engineering Faculty Publications | Electrical & Computer Engineering

6-2009

A Graph-based Approach to Symbolic Functional Decomposition of Finite State Machines

Piotr Szotkowski
Warsaw University of Technology, p.szotkowski@tele.pw.edu.pl

Mariusz Rawski
Warsaw University of Technology, rawski@tele.pw.edu.pl

Henry Selvaraj
University of Nevada, Las Vegas, henry.selvaraj@unlv.edu

Follow this and additional works at: https://digitalscholarship.unlv.edu/ece_fac_articles

Part of the Computer Engineering Commons, Electrical and Electronics Commons, Signal Processing Commons, and the Systems and Communications Commons

Repository Citation

Szotkowski, P., Rawski, M., Selvaraj, H. (2009). A Graph-based Approach to Symbolic Functional Decomposition of Finite State Machines. *Systems Science, 35*(2), 41-47.
https://digitalscholarship.unlv.edu/ece_fac_articles/281

This Article is brought to you for free and open access by the Electrical & Computer Engineering at Digital Scholarship@UNLV. It has been accepted for inclusion in Electrical and Computer Engineering Faculty Publications by an authorized administrator of Digital Scholarship@UNLV. For more information, please contact digitalscholarship@unlv.edu.

LINDA HALL LIBRARY
5109 CHERRY STREET
KANSAS CITY, MISSOURI 64110-2498
PHONE **(816) 363-4600**
FAX: **(816) 926-8785**

7/8/13DocServ #: 704151

9:09

SHIP TO:

University of Nevada Las Vegas
Libraries
ILL/LLB DOCK - BOX 457006
4505 MARYLAND PKY
LAS VEGAS NV 89154-7006

UNL
Patron: Keller, Katherine

Odyssey: 206.107.42.153
Phone:
Ariel:

Regular

Odyssey

CCG

Max Cost: 35.00IFM

Notes: Borrowing Notes; (maxCost; $35)

Shelved as:

Location: LHL

Title: Systems science.

Volume: 35

Issue: 2

Date: 2009

P. Szotkowski, M. Rawski, H. Selvaraj,

Article/Paper: A graph-based approach to symbolic functional decomposition of finite state machines

Pages: 41-48 ~~48~~ 7

OCLC #: 2580682
ISSN: 0137-1223

Lender string:
AZU,IWA,*LHL,BRI,CUY,CUY

IL: 106703308

7

DocServ / OCLC / Pull Slip

Systems Science, Vol. 35, No. 2, 2009

PIOTR SZOTKOWSKI*, MARIUSZ RAWSKI*,
HENRY SELVARAJ**

A GRAPH-BASED APPROACH TO SYMBOLIC FUNCTIONAL DECOMPOSITION OF FINITE STATE MACHINES

This paper discusses the symbolic functional decomposition method for implementing finite state machines in field-programmable gate array devices. This method is a viable alternative to the presently widespread two-step approaches to the problem, which consist of separate encoding and mapping stages; the proposed method does not have a separate decomposition step – instead, the state's final encoding is introduced gradually on every decomposition iteration.

Along with general description of the functional symbolic decomposition method's steps, the paper discusses various algorithms implementing the method and presents an example realisation of the most interesting algorithm. In the end, the paper compares the results obtained using this method on standard benchmark FSMs and shows the advantages of this method over other state-of-the-art solutions.

Keywords: *finite state machine, FSM, FPGA, functional decomposition*

1. Introduction

The implementation of finite state machines (FSMs) in field-programmable gate array (FPGA) devices requires a transformation of the initial, possibly multi-input and multi-output FSM into a set of "small" (for example: four-input and one-output) Boolean functions, implementable directly in the FPGA's logic cells. In particular, this means that the Q and Q' multi-value variables (which represent the current and next state of the machine) have to be encoded into binary values as part of the implementation process.

All of the currently widespread approaches to implementation of FSMs in FPGA devices, such as the ones proposed in [1, 4, 13], consist of two separate steps: the encoding of the FSM's states into fixed, predefined binary representations, followed by mapping of the resulting binary function into the FPGA's logic cells. While the process of binary functional decomposition is considered to give the best results for the mapping step, all these approaches share the common disadvantage of introducing a definitive, final encoding of the machine's states in advance, before the mapping process begins. The complexity of the (most often – multi-level) mapping step makes it very hard to create a universal state encoding method; all of the presently used methods are optimised for certain FSM types or certain FPGA architectures, and do not yield optimal results in any general sense. The topics of different coding styles and possible optimizations of the state encoding algorithms with regard to performance and resource utilization are discussed in many current papers, such as [5, 12].

A completely different method of FSM implementation has been proposed in [6]. This novel approach, called symbolic functional decomposition, eliminates the need to break the implementation process into two steps – definitive binary state encoding followed by the excitation of the binary function's synthesis. This new method performs the decomposition without having to pre-encode the state variables; instead, the multi-value nature of these variables is

* Warsaw University of Technology, Institute of Telecommunications, Nowowiejska 15/19, 00-665 Warszawa, Poland, e-mail: p.szotkowski@tele.pw.edu.pl, rawski@tele.pw.edu.pl

** University of Nevada, Department of Electrical and Computer Engineering, 4505 Maryland Pkwy., Las Vegas, NV 89154, USA, e-mail: selvaraj@ee.unlv.edu

maintained through the process of functional decomposition, while a partial encoding of the states is gradually introduced – which leads to a final state encoding yielding good overall quality (indicated by low number of logic cells used in the final implementation of the FSM). The approach of symbolic functional decomposition for implementation of finite state machines in FPGA devices, proposed in [7, 8], does not suffer from the disadvantage of separate encoding and mapping steps, as it does not have a separate state encoding step – instead, the initial finite state machine (with its multi-valued Q and Q' state variables) is subjected to a symbolic decomposition process that encodes the states gradually at every consecutive decomposition iteration; the states are encoded partially, only as much as required (and useful) in the given iteration of the process.

This paper presents algorithms implementing the method of symbolic functional decomposition. The presented algorithms give better results than the above-mentioned, currently widespread two-step approaches; the results are also on par with the ones obtained using the state assignment method described in [3]. Further improvements to the algorithm are expected to yield even better results in the future.

2. Symbolic functional decomposition

Similarly to serial decomposition of a Boolean function, symbolic functional decomposition of an FSM can be described in terms of blankets induced by its inputs, outputs and state variables.

Let X be the set of primary inputs, Y be the set of primary outputs of a certain FSM specified by a state transition table. Let Q and Q' be multi-valued variables representing present and next state of this FSM. Let U and V be two subsets of X, such that $U \cup V = X$. Let Q_V and Q_U be multi-valued variables encoding variable Q. Let β_V and β_U be blankets induced by the primary input subsets V and U. Let β_{Q_V} and β_{Q_U} be blankets induced by the multi-valued variables Q_V and Q_U. Let β_Y and $\beta_{Q'}$ be blankets induced by the primary output sets and by next state multi-valued variable Q'.

Theorem 1. *Existence of a symbolic functional decomposition* [8].

An FSM has a symbolic functional decomposition with respect to (U, Q_U, Q_V, V) *if there exists a blanket* β_G *such that* $\beta_V \bullet \beta_{Q_V} \leq \beta_G$ *and* $\beta_U \bullet \beta_{Q_U} \bullet \beta_G \leq \beta_F$, *where* $\beta_F = \beta_Y \bullet \beta_{Q'}$.

3. An overview of the method

The terms used in the above definitions and theorem are best illustrated by Fig. 1. The idea behind the symbolic functional decomposition method is to operate on the blankets induced by the FSM's state transition table and to construct the three blankets from the theorem's inequalities: β_{Q_U}, β_{Q_V} and β_G. These blankets have to fulfill the above theorem, as well as ideally be small enough for the G block, in its final realization, to not have more binary inputs than the largest Boolean function realised by a logic cell of the FPGA device; such a G block can be implemented in a single logic cell (or several parallel ones).

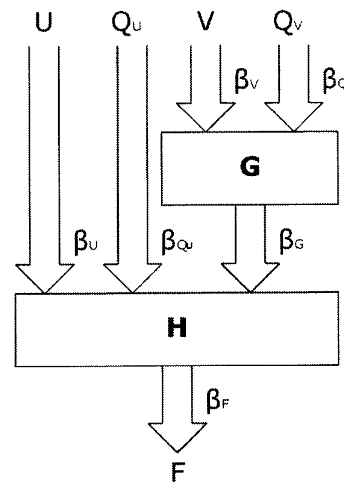


Fig. 1. Symbolic functional decomposition with blankets.

3.1. Construction of the β_{Q_U} blanket

Once the U and V subsets of the binary inputs have been chosen, the β_{Q_U} blanket can be constructed. On the one hand, the blanket has to satisfy $\beta_Q \leq \beta_{Q_U}$ (i.e., be constructed from the blanket induced by the FSM's state variable); on the other, the $\beta_U \bullet \beta_{Q_U} \bullet \beta_G \leq \beta_F$ requirement of the decomposition's theorem has to be fulfilled.

The construction of this blanket is very important for the quality of the decomposition (measured as the number of logic cells required to implement the given FSM in the target FPGA device). The fewer blocks this blanket has, the fewer binary inputs will be required in the final implementation of the FSM – the number of binary inputs required to implement a symbolic variable is equal to the base-two logarithm of the number of the blanket's blocks, rounded up.

Ideally, the β_{Q_U} blanket ought to have fewer blocks than the β_Q blanket – fewer enough to make it "fit" in a smaller number of binary inputs of the final implementation. For example, if the FSM has ten states (and, thus, the β_Q blanket has ten blocks and requires at least four bits to be implemented), making the β_{Q_U} blanket have at most eight blocks means it can be implemented using just three binary inputs; likewise, if the β_{Q_U} blanket is constructed so that it has four of fewer blocks, it becomes implementable using just two inputs.

Unfortunately, the fewer blocks the β_{Q_U} blanket has, the less separations (required by the FSM's outputs) it can provide. All of the separations required by the F function and not provided by the β_U blanket have to be provided by either β_{Q_U} or β_G; the less of them come from β_{Q_U}, the more will have to come from β_G, the more blocks will β_G have and, in the end, the more binary inputs will be required to implement it (thus offsetting the advantage of a "small" β_{Q_U} blanket).

3.2. Construction of the β_G blanket

Once the β_{Q_U} blanket is constructed, the β_G blanket has to provide all of the separations required by the F function which are not provided by either β_U or β_{Q_U} (the $\beta_U \bullet \beta_{Q_U} \bullet \beta_G \leq \beta_F$ part of the theorem has to be satisfied). The β_G blanket, which describes the output of the G block, is constructed from the blocks of the $\beta_V \bullet \beta_{Q_V}$ blanket. Given that the β_{Q_V} blanket is constructed from the FSM's state variable (much like β_{Q_U}), the β_G blanket by definition must fulfill the $\beta_V \bullet \beta_Q \leq \beta_G$ requirement – in other words, be constructed from the blocks of the $\beta_V \bullet \beta_Q$ blanket.

Again, the smaller the number of blocks in the β_G blanket the fewer binary inputs will be required in the final implementation.

3.3. Construction of the β_{Q_V} blanket

In the algorithms proposed in [10, 11], the β_{Q_V} blanket is constructed in a separate step – based on the separations required by the β_G blanket and not provided by the β_V blanket (so that the $\beta_V \bullet \beta_{Q_V} < \beta_G$ part of the theorem is satisfied). In the algorithm proposed below, the β_G and β_{Q_V} blankets are constructed concurrently; the way the β_G blanket is constructed impacts heavily the requirements for the β_{Q_V} blanket's construction, and so their concurrent construction yields better results (while being more complicated to compute).

As with the β_{Q_U} blanket, the β_{Q_V} blanket is constructed from the FSM's state variable, and thus fulfills $\beta_Q \leq \beta_{Q_V}$. The goal of the β_{Q_V} construction step is to make this blanket as small as possible; in particular, the number of binary inputs used to implement it, combined with the size of the V set (i.e., the number of indirect inputs) selected for the decomposition should be less than or equal to the number of inputs of the largest (input-wise) logic cells in the target FPGA device – otherwise the G block will have to undergo a separate decomposition process (because it will not "fit" into a single logic cell).

3.4. Creation of the G and H tables

Once the β_{Q_U}, β_G and β_{Q_V} blankets are constructed, it is possible to create the G and H tables.

The G table, which implements the G block "torn off" of the initial FSM, contains a typical binary function, ready to be implemented in the target FPGA device.

The H table describes a new, smaller (input- and/or state-wise) FSM – the states of which are described by the blocks of the β_{Q_U} blanket. If encoding this machine's states to binary values would make it "fit" into the target architecture (e.g., a two-input, eight-state FSM being implemented in five-input logic cells), it can be implemented directly in the target FPGA device (much like the G table). Otherwise, the whole process of symbolic functional decomposition is repeated, with the machine described in the H table acting as the new base FSM.

4. Algorithms implementing symbolic functional decomposition

4.1. Algorithms for construction of the β_{Q_U} blanket

So far, two algorithms were designed to implement the construction of the β_{Q_U} blanket.

[10] presents a relatively fast algorithm based on the concept of r-admissibility. As mentioned above, once the separations provided by the β_U blanket are subtracted from the separations required by the β_F

blanket, the remaining separations must be provided by the β_{Q_U} and β_G blankets. The more of these "missing" separations are provided by the β_{Q_U} blanket, the fewer of them must be provided by the β_G blanket and so fewer target logic blocks are required to implement the G block. On the other hand, if the target architecture has multiple outputs – let us assume a 4/2 architecture – it would be a waste of resources to create a β_G block implementable on an odd number of binary outputs; in this case a smaller β_{Q_U} blanket might be implementable on fewer direct binary inputs with a larger β_G blanket still leading to the same number of logic cells used by the G block. This relationship between the construction of the β_{Q_U} blanket and the number of target architecture's outputs is represented by r-admissibility, which for a given β_{Q_U} blanket denotes the minimal number of binary outputs required by the β_G blanket.

An alternative algorithm for β_{Q_U} construction proposed in [11] uses the concept of an incompatibility graph created from the β_Q blanket's blocks, and merges the graph's vertices until the β_{Q_U} blanket is implementable in a fewer number of binary inputs. This approach yields better average results than the one based on r-admissibility, but at the cost of longer computation.

4.2. Algorithms for construction of the β_G and β_{Q_V} blankets

So far, three algorithms were designed to implement the construction of the β_G and β_{Q_V} blankets. Two of them, proposed in [10] and [11], construct these blankets independently of each other; the third one, described in detail below, constructs both blankets concurrently.

4.2.1. Separate β_G and β_{Q_V} creation

[10] presents an algorithm that first constructs the β_G blanket by creating an incompatibility graph; the vertices represent the blocks of the $\beta_V \bullet \beta_Q$ blanket, while the edges connect the vertices that must be kept separated so that the β_C blanket provides all the separations required by the β_F blanket and not provided by the β_U and β_{Q_U} blankets. This graph is subsequently colored; the β_G blanket contains blocks created by merging same-colored vertices.

Once the β_G blanket is defined, the β_{Q_V} blanket can be constructed in the same way – by coloring an incompatibility graph built from blocks of the β_Q blanket and separations required by the β_G blanket that are not provided by the β_V blanket. This method of separate construction of the β_G and β_{Q_V} blankets is relatively fast; the speed (and the quality of results) depends on the algorithm used for coloring the graph vertices.

A different approach to construction of the β_G and β_{Q_V} blankets was proposed in [11]. The algorithm constructs the two blankets in the same way it constructs the β_{Q_U} blanket; once the incompatibility graph for either of the two blankets is created, the vertices are merged until the whole blanket is implementable on a fewer binary inputs. As for the construction of β_{Q_U}, this approach yields better results than the one proposed in [10], but at a cost of longer computation.

4.2.2. The issue with separate β_G and β_{Q_V} creation steps

While the separate construction of the β_G and β_{G_V} blankets is very convenient for implementation of the above algorithms (both blankets are constructed using the same process, the only difference is in the parameters – β_G is built from blocks of $\beta_V \bullet \beta_Q$ and must provide separations required by β_F which are not provided by β_U and β_{Q_U}, while β_{Q_V} is built from blocks of β_Q and must provide separations required by β_G which are not provided by β_V), the constructions of these blankets are far from being independent of each other.

The β_G blanket can be constructed in many different ways from the blocks of the $\beta_V \bullet \beta_Q$ blanket while still satisfying the $\beta_U \bullet \beta_{Q_U} \bullet \beta_G < \beta_F$ part of the theorem. If the creation of the β_G blanket is done independently of the creation of β_{Q_V}, it will most probably require many separations that are not provided by β_V (but only by β_Q); these separations will have to be provided by β_{Q_V} to satisfy the $\beta_V \bullet \beta_{Q_V} < \beta_G$ requirement. This leads to large β_{Q_V} blankets, which in turn means the resulting G block might not be directly implementable in the target architecture (in such a case, either the whole decomposition is discarded or the G table is created and undergoes a separate decomposition step).

To solve the above issue, the construction of the β_G and β_{Q_V} blankets must happen in concurrency. In the algorithm presented below, the dual coloring of the incompatibility graph causes the two blankets to be more compatible with each other – and, in the end, yield a better decomposition.

4.2.3. An example separate construction of β_G and β_{Q_V}

Assume

$$\beta_V = \{\overline{11,21,31}; \overline{12,22,32}; \overline{13,23,33}; \overline{14,24,34}\},$$

$$\beta_Q = \{\overline{11,12,13,14}; \overline{21,22,23,24}; \overline{31,32,33,34}\},$$

$$\beta_V \bullet \beta_Q = \{\overline{11}; \overline{12}; \overline{13}; \overline{14}; \overline{21}; \overline{22}; \overline{23}; \overline{24}; \overline{31}; \overline{32}; \overline{33}; \overline{34}\},$$

and the incompatibility graph for construction of the β_G blanket presented in Fig. 2.

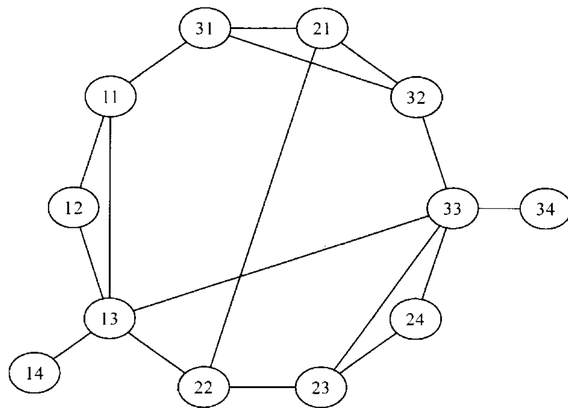


Fig. 2. An example incompatibility graph.

If this graph is subsequently colored (for example, by using the classic Welsh–Powell algorithm [14]), it might end up being colored as in Fig. 3.

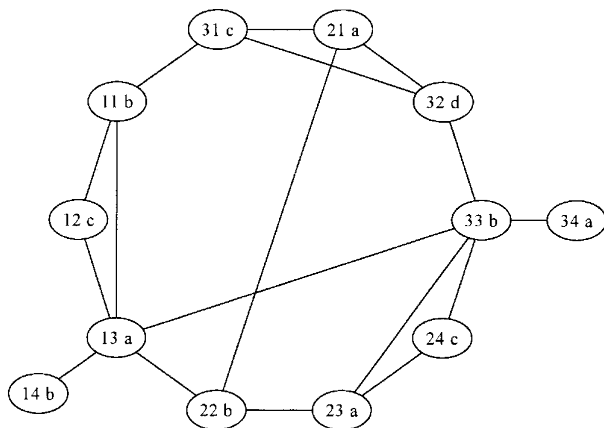


Fig. 3. An example incompatibility graph, colored.

The β_G blanket constructed in this way equals $\{\overline{13,21,23,34}; \overline{11,14,22,33}; \overline{12,24,31}; \overline{32}\}$ and looks like a reasonable result – the graph has three-vertex cliques, which means it is definitely not two-colorable (so cannot possibly be implemented using a single binary input), while using four colors means it will still fit into two binary inputs.

Unfortunately, this coloring means that the β_G blanket requires the separation of the 11 vertex from the 21 vertex and both of them from the 31 vertex – these three separations are not provided by β_V, so each of them must be provided by β_{Q_V}. This, in turn, means that β_{Q_V} must be a three-block blanket (and, in fact, equal β_Q); with the above construction of β_G, β_{Q_V} cannot be implemented with a single binary input.

4.2.4. An example concurrent construction of β_G and β_{Q_V}

Assume, once again, the same β_V and β_Q blankets and the same incompatibility graph from Fig. 2. By concurrently constructing the β_G and β_{Q_V} blankets, a better overall solution can be obtained.

Figure 4 presents the result of concurrent coloring of the graph from Fig. 2 with two kinds of colors (represented by lower- and uppercase letters); the lowercase letters are used to color the vertices representing the blocks of the $\beta_V \bullet \beta_Q$ blanket (to create the β_G blanket), while uppercase letters color groups of vertices which represent the blocks of the β_Q blanket – this coloring creates the β_{Q_V} blanket.

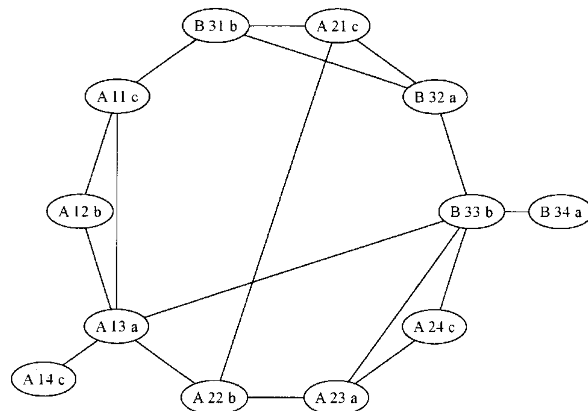


Fig. 4. A better coloring of the incompatibility graph.

As can be seen, in this case the β_G blanket equals $\{\overline{13,23,32,34}; \overline{12,22,31,33}; \overline{11,14,21,24}\}$ and the β_{Q_V} blanket equals $\{\overline{11,12,13,14,21,22,23,24}; \overline{31,32,33,34}\}$ while still providing β_G with all the required separations not provided by β_V.

4.2.5. Steps of the algorithm

The algorithm for concurrent coloring is based on five steps, repeated until all the vertices have both kinds of

colors (and with backtracking if an illegal coloring was obtained at any given time):

1. Choose a vertex for coloring.
2. Color it and all the other vertices belonging to the same β_Q block with the β_{Q_V} ("uppercase") color.
3. Sync colors of all the related vertices.
4. Color it with the β_G ("lowercase") colour.
5. Sync colors of all the related vertices.

Choosing the vertex for coloring

This is the crucial step of the algorithm. The order in which the vertices are colored impacts the final blanket construction tremendously; unfortunately, making this step more complex has an equally tremendous impact on the speed of the algorithm.

In the initial version of the algorithm, the vertices are first selected based on the number of forbidden β_{Q_V} ("uppercase") colors, then on the number of forbidden β_G ("lowercase") colors, and then – if these values are equal – based on their degree.

Coloring with the β_{Q_V} color

Once a vertex without a β_{Q_V} color is selected, it (and all the other vertices belonging to the same β_Q block) is assigned the first "uppercase" color that is not forbidden.

Coloring with the β_G color

Similarly, once a vertex without a β_G color is selected, it is assigned the first not-forbidden "lowercase" color.

Color syncing

Every assignment of a color to a vertex (or a group of vertices) means this color should be forbidden to use by other selected vertices (or groups of them). For example, once the 34 vertex is assigned the *a* color, the 33 cannot be colored with *a*.

For a more complicated example, assume that the $\overline{11,12,13,14}$ β_Q block is colored with A; if, subsequently, the $\overline{21,22,23,24}$ β_Q block gets colored with A, all the vertices from these two β_Q blocks that are in the same β_V block must have the same β_G color (otherwise β_G would require a separation that would not be provided by either β_V nor β_{Q_V}). If at the given time the 12 vertex has the *b* color, the 22 vertex must also have it (and vice-versa).

Thus, after every coloring step, a separate step for syncing both kinds of colors is performed; in case the coloring in the previous steps led to an illegal assignment of colors, the whole algorithm backtraces to the last known-good (and "stable") coloring, forbids the coloring that led to the illegal situation and tries again.

5. Experimental results

Experiments with a prototypical symbolic functional decomposition program *art décomp*, developed to validate the algorithms described above, confirm bet-

Table 1. Experimental results for 4/2+5/1 LCs.

FSM	art décomp	Secode	Jedi	Nova -i	Nova -io	hot-one
bbara	9	7	11	13	14	15
bbtas	4	4	5	3	4	6
beecnt	7	6	9	8	9	12
dk15	7	12	11	13	13	17
dk17	6	10	11	9	11	17
dk27	3	3	3	4	3	6
lion	2	2	2	3	2	3
s8	1	1	1	1	1	9
Σ	39	45	53	54	57	85

Table 2. Experimental results for 5/1 LCs.

FSM	art decomp	Secode	Gray	Jedi	bin
bbara	10	12	11	15	15
bbtas	5	5	5	5	5
beecnt	8	6	8	9	10
dk15	7	7	7	7	7
dk17	6	6	6	6	6
dk27	5	5	5	5	5
lion	3	3	3	3	3
s8	1	1	6	5	5
Σ	45	45	51	55	56

ter results than ones obtained with the common two-step approach, and slightly better than or on par with results from the *Secode* state assignment method described in [3, 9].

Table 1 shows the number of logic cells used by implementing common FSM benchmarks in a Xilinx XC4000 FPGA device. The Jedi [4], Nova [13] and hot-one columns show the number of required logic cells used by the implemented finite state machine after being encoded with the given method and then synthesized with the logic synthesis tool Sis, while the Secode column presents the state assignment method described in [3] (likewise coupled with the Sis tool). Column art decomp shows results obtained with the above algorithms.

Table 2 shows a similar comparison (this time for five-input, single-output cells) between Secode, Jedi and two simple minimal-length encoding methods – sequential encoding (column bin) and Gray encoding. This time the results were synthesized with the IRMA2FPGA combinational synthesis tool [2]. As can be seen from the table, the results obtained with *art décomp* are on par with Secode and better than the ones from classic, two-step approaches.

6. Conclusions

As mentioned in the introduction, the currently widespread two-step approach to the implementation of finite state machines in FPGA devices does not yield optimal results due to the definitive character of assigning final binary encodings to the FSM's states prior to the mapping step; because of the complexity of the (often, multi-level) logic synthesis process, creating a universal algorithm for pre-encoding the state variable is very hard.

The symbolic functional decomposition method, which side-steps the issue by retaining the relevant information about the states of the FSM through the mapping process, yields better results than the two-step approaches; at the same time, the algorithms currently implementing this method still leave room for improvement.

Acknowledgements

This paper was supported by the Ministry of Science and Higher Education of Poland – research grant No. N517 003 32/0583 for 2007–2010.

References

[1] de Micheli G., Brayton R.K., Sangiovanni-Vincentelli A., *Optimal state assignment for finite state machines*, IEEE Trans. on CAD, pp. 269–284.

[2] Jóźwiak L., Chojnacki A., *Effective and efficient combinational circuit synthesis for the FPGA-based reconfigurable systems*, Special Issue of Journal of Systems Architecture on Re-configurable Computing, 49(4–6), pp. 247–265.

[3] Jóźwiak L., Ślusarczyk A., *A new state assignment method targeting FPGA implementations*, Proc. EUROMICRO Symposium on Digital System Design DSD 2000, pp. 50–59.

[4] Lin B., Newton A.R., *Synthesis of multiple level logic from symbolic high-level description languages*, Proc. of IFIP Int. Conf. on VLSI, pp. 187–196.

[5] Rafla N., Davis B., *A study of finite state machine coding styles for implementation in FPGAs*, 49th IEEE International Midwest Symposium on Circuits and Systems MWSCAS 2006, pp. 337–341.

[6] Rawski M., *The novel approach to FSM synthesis targeted FPGA architectures*, Proceedings of IFAC Workshop on Programmable Devices and Systems PDS 2004, pp. 169–174.

[7] Rawski M., Selvaraj H., Luba T., Szotkowski P., *Application of symbolic functional decomposition concept in FSM implementation targeting FPGA devices*, Sixth International Conference on Computational Intelligence and Multimedia Applications ICCIMA 2005, pp. 153–158.

[8] Rawski M., Selvaraj H., Luba T., Szotkowski P., *Multilevel synthesis of finite state machines based on symbolic functional decomposition*, International Journal of Computational Intelligence and Applications, 6(2), 2007, pp. 257–271.

[9] Ślusarczyk A., *Decomposition and Encoding of Finite State Machines for FPGA Implementation*, Technische Universiteit Eindhoven, 2004.

[10] Szotkowski P., Rawski M., *Symbolic functional decomposition algorithm for FSM implementation*, Proceedings of the International Conference on Computer as a Tool EUROCON 2007, p. 484–488.

[11] Szotkowski P., Rawski M., *A graph-based symbolic functional decomposition algorithm for FSM implementation*, to be published in proceedings of the Conference on Human System Interaction HSI 2008.

[12] Titarienko L., Węgrzyn M., *Optimization of Moore FSM on FPGA*, CAD Systems in Microelectronics, 2007, CADSM'07, 9th International Conference – The Experience of Designing and Applications of, February 2007, pp. 246–250.

[13] Villa T., Sangiovanni-Vincentelli A., *Nova: state assignment of finite state machines for optimal two-level logic implementation*, IEEE Trans. on CAD, pp. 905–924.

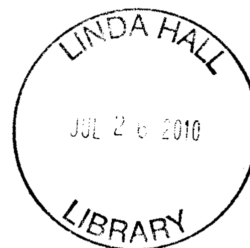
[14] Welsh D.J.A., Powell M.B., *An upper bound for the chromatic number of a graph and its application to timetabling problems*, The Computer Journal, 10(1), 85–86.

Received March 31, 2009

Wrocław University of Technology

Systems Science

published quarterly



WROCŁAW 2009