ASS†UCE - An Exploratory Environment for Finite State Machines

Ney Laert Vilar CALAZANS & André Duque MADEIRA
Instituto de Informática - PUCRS
Av. Ipiranga, 6681 - 90619-900 - Porto Alegre - RS - BRAZIL
Phone: +55 (51) 339-1511 Ext: 3211 Fax: +55 (51) 339-1564
e-mail addresses: (calazans, madeira)@andros.inf.pucrs.br

Abstract

This paper describes ASS†UCE, an exploratory environment for Finite State Machines. The environment is intended as an aid in the field of digital systems design with research and educational purposes in mind. Currently, ASS†UCE is already applicable as a set of computer aided design tools for the synthesis of digital systems. We describe the general structure of the environment, stressing its use as an educational tool. We also present the available aids for performing state minimization, and simultaneous state assignment and state minimization. Next, we discuss ongoing works in the application of the same formal concepts underlying the available tools to the solution of other hardware design problems, like functional decomposition and asynchronous design. Benchmark results are presented comparing ASS†UCE component programs to available state of the art research tools.

Keywords: digital systems, computer aided design, finite state machines.

1 Introduction

Finite State Machines (FSMs) are widely employed in Computer Science in general, and in Digital Systems Design in particular. Indeed, any digital system can be described in the logic level of abstraction using this model. Such description can be synthesized as a digital circuit, through the use of well-known logic design techniques for sequential circuits, e.g. those described in textbooks like [12].

However, some facts prevent this approach to be useful in building state of the art digital systems. First, present Very Large Scale Integration (VLSI) digital circuits are typically composed by a number of electronic components in the order of 10^6 to 10^7, and should thus be represented by FSMs counting maybe as many as 10^{500} states[28]. Of course, applying logic design techniques to implement such FSMs in hardware is unthinkable, even if powerful supercomputers are used to accomplish the design task. Second, since FSMs are a general model, its efficiency to represent hardware modules that display some regular internal organization (like datapaths, register files and memories) is very low compared to specifically designed models. The only way to deal with the first fact is to rely upon the extensive use of hierarchical decomposition of the *design process*. This leads to the proposition of higher level models such as Communicating FSMs[11] and Communicating Sequential Processes[13]. The solution to the second problem normally relies on the use of specific models whenever they are applicable, leaving the FSM model to the design of unstructured hardware modules only.

Even with the above mentioned restrictions to its use, FSMs are by far the most useful model for describing many important classes of hardware modules like sequence detectors, embedded controllers, industrial process controllers and processor and plant control units[6]. Thus, we consider justified to propose an exploratory environment for manipulating FSMs in view of its implementation in hardware. While stemming directly from research work in integrated circuits (ICs) design [3, 5], the ASS†UCE environment is evolving to become both a research and educational tool for digital systems at undergraduate and graduate levels.

The rest of the paper is organized as follows. The next Section states the addressed problem and gives an overview of the ASS†UCE environment characteristics, including a brief description of a first tool, XASS†UCE, designed to act as a textual/graphical interactive interface to the environment. Closing this Section, there is a discussion on the current work in enhancing the educational characteristics of ASS†UCE. Section 3 presents the internal structure of ASS†UCE, together with some of the main concepts used in the construction of the research tools integrated in the environment. The discussion stresses the generality of the underlying formal framework. The same Section comprises a brief discussion of the two currently available research tools, a heuristic

encoder/minimizer for FSMs, and an exact state minimizer. Also in this Section is an overview of ongoing work in present and new research tools of the environment. Section 4 quantitatively compares our research tools to others available in the public domain. In fact, these three sections present the environment separately in its two aspects, as an educational tool (Section 2) and as a research resource (Sections 3 and 4). Finally, Section 5 presents some conclusions on the use and implementation of the environment.

2 The ASS†UCE Environment

2.1 The FSM Design Problem

An FSM is an abstract model that can be formally described as follows:

Definition 1 (Finite State Machine) *A **finite state machine** (FSM) is an algebraic structure of the form $\mathcal{A} = \langle I, S, O, \delta, \lambda \rangle$ where:*

1. *$I = \{i_{p-1}, i_{p-2}, \ldots, i_0\}$ is the **input alphabet**, $S = \{s_{q-1}, s_{q-2}, \ldots, s_0\}$ is the finite **state set**, and $O = \{o_{r-1}, o_{r-2}, \ldots, o_0\}$ is the **output alphabet**;*
2. *δ is a discrete function: $\delta : I \times S \longrightarrow S$; called **next state or transition function** of $\mathcal{A}$; given a pair $(i_j, s_k) \in I \times S$, if $\delta(i_j, s_k)$ is specified, $s_l = \delta(i_j, s_k)$ is the **next state** of the FSM $\mathcal{A}$ corresponding to the input i_j and to the **present state** s_k;*
3. *λ is a discrete function: $\lambda : I \times S \longrightarrow O$, called **output function** of $\mathcal{A}$; given a pair $(i_j, s_k) \in I \times S$, if $\lambda(i_j, s_k)$ is specified, $o_m = \lambda(i_j, s_k)$ is the **output** of the FSM $\mathcal{A}$ corresponding to the input i_j and to the **present state** s_k;*

*The pair $(\delta(i_j, s_k), \lambda(i_j, s_k))$ is called a **transition** of FSM $\mathcal{A}$. If both, δ and λ are completely specified functions, $\mathcal{A}$ is called a Completely Specified FSM (CSFSM), otherwise $\mathcal{A}$ is called an Incompletely Specified FSM (ISFSM).*

In order to implement a digital system starting from an FSM description, several steps are required. Each of these steps consists normally in solving some complex design optimization problem, whose solution leads to the best implementation in hardware of the FSM functionality. Hardware implementations are considered optimal if they satisfy one or more objective criteria. These criteria most often conflict with each other. The most commonly agreed relevant criteria are speed, silicon area, power dissipation and testability. If we restrict attention to the logic level of abstraction only, the most important design problems related to FSMs are:

- decomposition - given an FSM, break it into a set of smaller, communicating FSMs whose input/output behavior is equivalent to or compatible with the original description, and such that from the resulting network we can obtain the best hardware implementation;
- state minimization or reduction - given an FSM A, find another equivalent or compatible FSM B, but which has the set of states S_B with the least cardinality;
- input, state and output assignment or encoding - given a *symbolic* FSM, find a binary encoding for the sets of inputs, states and outputs (I, S, O), generating an *encoded* FSM; the encoding must be such that the physical implementation is the best and, at the same time, equivalent to or compatible with the symbolic description;
- mapping or binding - given an encoded FSM, the kind of hardware device to use in the implementation and a library of hardware modules supported by the device, translate the FSM description into a set of interconnected modules that is equivalent to or compatible with the encoded FSM, and which satisfies optimality criteria.

2.2 Environment Basic Principles

Engineering and Computer Science students are taught to deal with FSM logic design representations such as state transition graphs and tables (STGs/STTs), truth tables and Karnaugh maps. However, production tools for FSM design are never based on these, due to their associated computational complexity.

Production and research Computer Aided Design (CAD) tools make available very complex and very efficient methods and representations to solve each FSM hardware design problem. Although all such tools allow automatic

execution, the best designs can only be obtained after careful experimentation with each tool, iteratively using distinct parameterizations. This can only be achieved if the concepts underlying each tool are well understood. More critical than the designer role is that of the computer scientist, which must be able to enhance and/or propose alternate or missing design methods. Their task is further complicated by the fact that the design tools are normally inserted inside a commercial CAD system framework. These frameworks contain sometimes more than a hundred tightly coupled tools, and the design techniques used by each tool are considered as intellectual property, i.e. are not publicly available.

Training students to become clever tool users and tool designers is a hard task, since each FSM design problem implies mastering both, numerous formal concepts and convoluted data structures with associated manipulation algorithms.

As an example, let us consider the problem of performing FSM state minimization, shown by Pfleeger to be NP-complete in the general case of ISFSMs[22] (this is the model we assume here). The mathematical concepts that need to be mastered include equivalence and compatibility classes, set covering and set closure. From these, it is necessary to develop notions such as state compatibility, compatibility classes of states, prime compatibility classes and state class sets[8]. Then, adequate data structures and techniques must be sought to develop state compatibility analysis, and finding the minimum cardinality closed cover of compatibility classes of states. Complex techniques like *binate covering* were recently proposed for efficiently minimizing large FSMs[9], and this adds to harden the understanding of the design problem resolution.

The main objective of developing ASS†UCE as an educational resource is to provide an environment that significantly increases the controllability and observability of state of the art algorithms that implement the solution of FSM design optimization problems. This is obtained in ASS†UCE by allowing step by step execution coupled with aids to inspect data generated and manipulated in each step, either in qualitative or in quantitative form. For instance, using ASS†UCE, a user may choose to execute state minimization of some FSM. Alternately, he may read in the FSM description, perform state compatibility analysis and then, based on the state structure and quantitative data obtained from this step, decide for the setting of some specific optimization options for the following steps of the state minimization procedure.

Both commercial and academic environments for digital systems design already exist. Examples are SIS, a research environment developed at the University of Berkeley[25] and the educational environment proposed at the University of Arizona, also well-known[19]. However, our proposition differs from both examples, as will be depicted in the following Sections.

2.3 The Environment Interface

The ASS†UCE environment provides a framework where to integrate new design tools in a form adequate to use them with teaching purposes. It comprises an interactive, textual/graphics interface with an execution shell accepting commands. The shell allows the execution of command scripts, so that using a combination of batch and on-line commands is possible.

The philosophy underlying the interface is to provide either fine-grained and coarse-grained interaction. In fine-grained interaction, each atomic command is a single transformation of data, such as state compatibility analysis or input constraints generation. At each step, the user can visualize results of the computation performed up to that moment and decide what to do next. Flags that control the execution of fine-grained commands can be set at any moment, provided their settings were not used in previous commands. In coarse-grained interaction, commands can represent sets of fine-grained commands or some design tool execution with specific parameters.

Commands can be classified in three groups:

- input/output - for reading and writing files, including the input and execution of scripts;
- status manipulation - used to visualize results and setting flags to control execution of some design tool step;
- FSM manipulation commands - employed to execute some design tool step or some analysis step.

The execution of commands and setting of flags is controlled through the use of dependency graphs to avoid running commands which require execution of other commands not issued yet or setting flags after having used the previous settings. Figure 1 shows a sample screen of the XASS†UCE program[26] that implements the environment interface after the execution of commands for loading and listing an FSM in the KISS2[7] format.

XASS†UCE was implemented using a mix of tools, Tcl/TK[20] for graphics and the shell, and C++ and the LEDA class library [17] for implementing computationally intensive tasks.

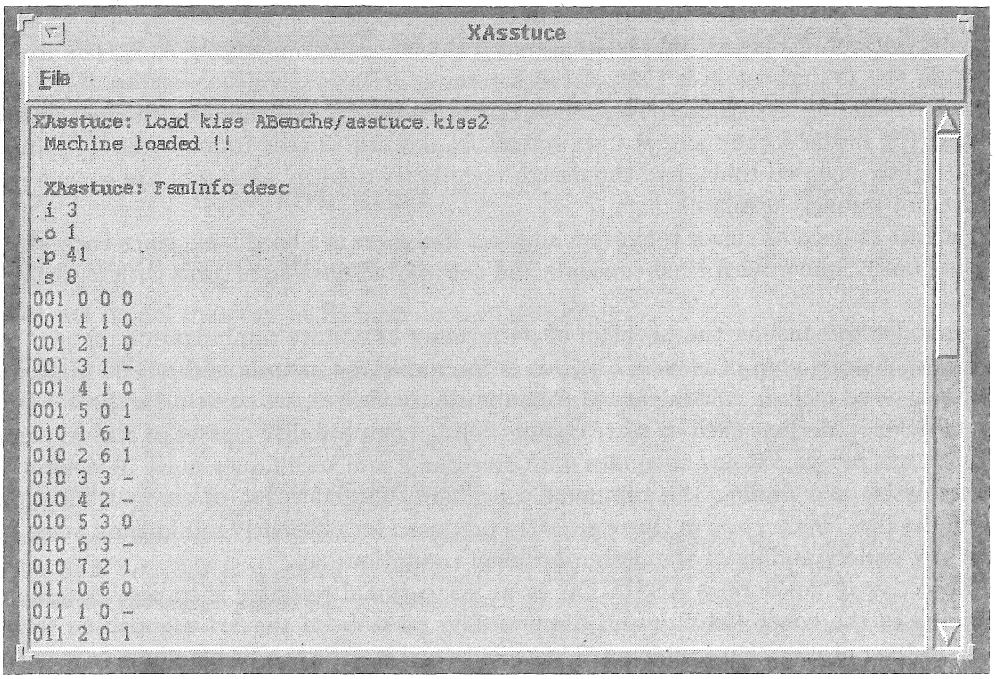

Figure 1: Sample Screen of XASS†UCE

2.4 Ongoing and Future Work

XASS†UCE already comprises several capabilities, as described in [26]. However, we are working in improving the user-friendliness and functionality of the interface. Future versions will make available the following set of features:

- modular addition of either new tools or new algorithms inside some tool already in the environment;
- enhanced graphical display of data structures, execution traces and results;
- script on-line generation using background log of commands;
- data exchanging capability with XSIS, the graphical interface to SIS;

The interface is already operational and is used in Computer Science graduate courses. With an enhanced version, we shall introduce it in undergraduate laboratory courses as well. As has happened in graduate courses, we intend to employ the teaching experience acquired to further the implementation of the environment.

3 ASS†UCE Internal Structure and Design Tools

Behind the proposal of the ASS†UCE environment there are several original research results. The most relevant of these is that the whole process of logic synthesis of FSMs can be based upon a single, generalized problem statement, called Boolean Constrained Encoding (BCE)[4]. Indeed, several authors have already treated decomposition[11], mapping[18] and encoding[23] using restricted versions of BCE. One of the authors proposed a technique to simultaneously addressing both, the state minimization and state assignment of FSMs[5].

The internal structure of ASS†UCE is depicted in Figure 2. It is centered around a unified framework that can contain any instance of each FSM design problem described by means of design constraints sets. In order to represent design constraints of several distinct problems, we use *pseudo-dichotomies*, an algebraic structure based on the mathematical concept of partitions, proposed in [3]. The concept of pseudo-dichotomy is at the same time simple and powerful. Informally, a pseudo-dichotomy is nothing but a pair formed by a set and a function. The set is a two-block partition of a subset of a set S, and the function tells the conditions under which any encoding satisfies or violates the pseudo-dichotomy. Pseudo-dichotomies can support the representation of virtually all constraints involved in FSM design problems.

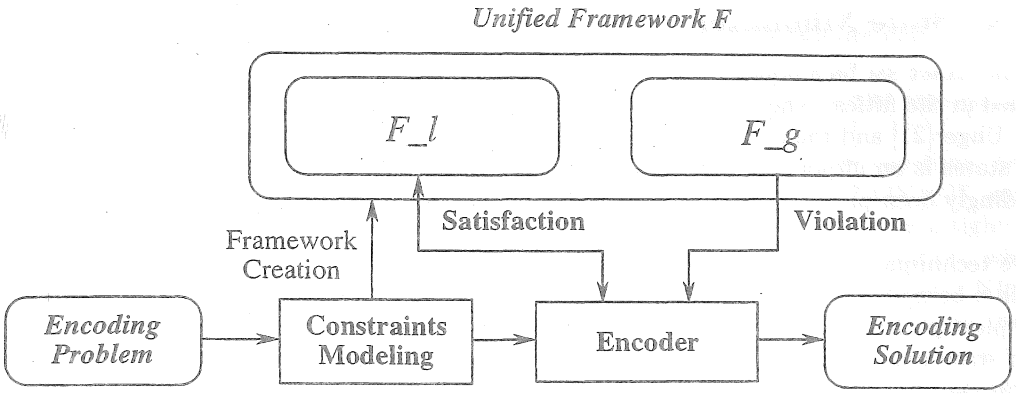

Figure 2: The internal structure of the ASS†UCE environment

As an overall strategy for solving FSM design problems in ASS†UCE, a design tool starts by translating the problem statement into a set of pseudo-dichotomies through the use of modeling techniques proper to the specific problem. After this step, the problem has been expressed and inserted in the unified framework of pseudo-dichotomies and is amenable to treatment by a single, general encoder, that obtains the final solution through the use of efficient constraint satisfaction techniques.

Note that the framework is employed by the encoder to guide the problem solution. Ensuring that *global constraints* (inside F_g) are never violated, guarantees correction of the problem solution, while ensuring that *local constraints* (inside F_l) are satisfied guarantees the optimality of the solution to some extent. More details on the unified framework, such as the formal definition of local and global constraints, and a discussion on how specific problem constraints are mapped to these can be found in [4].

3.1 Heuristic State Minimizer/Encoder

Traditionally, state minimization (SM) and state assignment (SA) are separate procedures of FSM logic synthesis, but using such a *serial strategy* may prevent the obtainment of optimal state assignments [10, 16]. We have proposed a method to address the problem of assigning codes to states of an FSM such that state minimization is taken into account during the encoding process, in what we call a *simultaneous strategy*[5, 1, 15]. Our approach allows creating incompletely specified encodings for the states in the original description. The subsequent combinational logic minimization step can, in this way, merge states such that an implicit state minimization step is performed.

The base of the method is a set of theoretical findings on the relationship between the SM and SA problems that were first described in [3, 4]. These problems were analyzed and each one was reduced to a set of constraints. The relationship among these constraint classes was established, and an encoding method considering both constraint classes were proposed and implemented as a computer program. This tool was called ASS†UCE and the ideas on it were extended to the proposition of the ASS†UCE environment.

For example, consider an FSM where a pair of states is equivalent or compatible. In this case, any state encoding that assigns intersecting sets of binary vectors to these states respects the equivalence or compatibility constraint. Thus, adding compatibility constraints among states to other encoding constraints can be beneficial to finding *better* or the *best* implementations. On the other hand, if two states are incompatible, no encoding can assign intersecting sets of binary values to them. The incompatibility constraints must be respected in order to guarantee a *correct* encoding of states. Compatibility and incompatibility constraints are examples of local and global constraints, respectively.

One advantage of the method behind ASS†UCE over other encoding methods is the possibility of generating sparse codes for symbols, i.e. codes containing don't cares, that constrain to a less extent succeeding tools in the design process.

The algorithm implemented in the ASS†UCE program is a heuristic encoder for FSM states. The runtime complexity of the algorithm has been shown to be $\mathcal{O}(n^2)$, where n is the cardinality of the state set of the FSM. For more details on the implementation of the program we refer to [5].

3.2 Exact State Minimizer

Heuristic and exact SM techniques can be widely found in FSM logic synthesis. Completely specified FSMs were much studied in the fifties. The reduction of the number of states on ISFSMs received a first thorough treatment by Paull & Unger[21] and the complexity analysis of the problem was addressed by Pfleeger[22]. Minimizing the number of states is an important step on the classic process of FSM design. New and more efficient algorithms have accordingly been proposed to deal with FSM VLSI implementations, e.g. the one proposed by Hachtel and others[9].

Intuitive techniques used to exactly minimize the number of states on CSFSMs have a $O(n^2)$ temporal complexity, with n being the cardinality of the set of states. A more efficient technique was proposed by Hopcroft[14], with a complexity of just $O(n \log n)$. Unfortunately, the Hopcroft algorithm works only with Moore machines, where each state has an associated output, i.e. the output depends exclusively on the present state and just indirectly on the primary inputs. Also, this technique was implemented to minimize compiler parser FSMs, i.e. machines with a single, yes/no kind of output.

We have proposed an algorithm which is based on Hopcroft's. Viewing the integration of this algorithm into ASS†UCE and the fact that we need to deal with hardware implementations of FSMs, we have had to generalize the original technique to accept Moore *and* Mealy machines with possibly non-binary output sets. This algorithm was implemented as a computer program which we call MEMCE[24].

The basic internal data structure of MEMCE is a generalization of the Inverse State Table (IST) proposed by Hopcroft. Our algorithms rely on state reachability analysis computed from this data structure, helped by auxiliary lists for data manipulation. More details on the implementation can be obtained in [24].

3.3 Ongoing and Future Work

The tools described in the previous paragraphs are already operational, but new versions are presently under development.

The heuristic minimizer/encoder needs more powerful constraint satisfaction methods, in order to take full advantage of the pseudo-dichotomies unified framework. A recently proposed new approach to implicitly representing functions and sets through Binary Decision Diagrams[2] is under investigation. Also, constraint classes other than those currently manipulated by the program have already been formulated using the unified framework[4], but their generation from the FSM structure and their consideration during encoding is not yet available. Finally, the implementation technology considered today by our tools is limited to two-level logic. Generalizing it to treat multilevel logic is a needed future work to cope with current state of the art production environments.

Another ongoing work is the implementation of a decomposition technique based on ASS†UCE theoretical findings to build FSMs on top of Look-Up-Table RAM-based FPGAs. FPGAs are a new kind of VLSI dynamically programmable device of great success in industry and academy.

An MSc. research work is under way on the subject of mapping asynchronous implementations of FSMs into FPGAs. A new research project involving the authors is starting on the implementation of a general encoder for dealing with several disparate encoding problems in VLSI and other fields as well. This works strongly capitalizes on the experience obtained from the ASS†UCE environment.

Concerning state minimization, we have plans to implement exact and heuristic techniques for minimizing ISFSMs in order to address more thoroughly this problem. Having complete control over state minization and encoding tools will allow a more relevant comparison between the serial and simultaneous strategy for solving both problems.

An important task we are currently undertaking is the integration of our tools into the SIS environment. While ASS†UCE is good for fine grain manipulation, providing an insight of the inner workings of each algorithm, SIS is in widespread use in research and is good for coarse grain manipulation, such as network restructuring and global optimizations. Integrating our tools within SIS is a good way of allowing fair comparisons between them and SIS corresponding algorithms.

4 Benchmarks Results

Both ASS†UCE and MEMCE were implemented as computer programs using also the C++ language and LEDA, under the UNIX operating system.

The FSM test set used is part of the benchmarks available at the Microelectronics Center of North Carolina (MCNC)[29]. For the ASS†UCE program, 12 machines were taken, corresponding to descriptions where at least one non-trivial pair of compatible states exists. A machine with only trivial compatible pairs of states is

already minimized, implying the uselessness of submitting it to state minimization tools. The test set used for the MEMCE program comprises the only 7 CSFSMs in the benchmark set. ISFSMs are currently not supported by MEMCE.

The FSMs are represented in KISS2, which is the input format accepted by the current versions of ASS†UCE and MEMCE.

Table 1 shows the comparison between ASS†UCE, parameterized with the **-s lw** run-time option, and the serial strategy of running STAMINA[9], to perform state minimization, followed by NOVA[27] which performs state assignment. We have chosen for STAMINA the run-time option **-s 1**, which invokes a tight upper bound heuristic algorithm for performing state minimization avoiding the use of the default option, which performs exact minimization and may lead to an exponential growth in the execution time. Concerning NOVA, we have opted for the run-time option **-e ih**, which invokes a constrained encoding algorithm based on the satisfaction of the input constraints only. We avoided the use of algorithms considering output constraints to maintain a fair comparison with ASS†UCE, since these constraints are not considered in our current implementation.

FSM	i_b	st	o_b	a_cl	sn_cl	a_pt	sn_pt	a_ar	sn_ar	a_t	sn_t	a_spy	sn_spy
s27	4	6	1	3	3	13	12	234	216	0.42	0.1	75.21	76.39
beecount	3	7	4	2	2	9	10	144	160	0.32	0.0	65.28	66.25
lion9	2	9	1	4	2	7	7	119	77	0.71	0.0	72.27	68.83
bbara	4	10	2	4	3	22	20	484	380	0.54	0.11	77.48	75.26
opus	5	10	6	4	4	19	16	532	448	0.54	0.1	67.86	72.54
train11	2	11	1	4	2	6	6	102	66	0.57	0.01	71.57	69.70
sse	7	16	7	7	4	27	31	1134	1023	1.46	0.23	78.75	79.86
bbsse	7	16	7	7	4	27	31	1134	1023	1.39	0.21	78.75	79.86
ex1	9	20	19	8	5	41	41	2501	2132	3.33	0.63	84.57	84.52
tbk	6	32	3	8	4	94	53	3666	1431	25.34	29.27	74.55	61.36
scf	27	121	56	9	7	133	124	18221	16244	167.26	103.54	91.32	91.49
s298	3	218	6	12	8	345	287	16560	10332	815.37	392.68	76.03	74.96
Sum/Avg	-	-	-	72	48	743	638	44831	33532	1017.25	526.88	76.14	75.09

FSM data:	i_b	input bits		
	st	number of states		
	o_b	output bits		
Prefixes:	a_	running ASS†UCE		
	sn_	running STAMINA+NOVA		
Suffixes:	cl	code length		
	pt	product terms	t	CPU time
	ar	estimated area	spy	PLA sparsity

Table 1: ASS†UCE versus Stamina+Nova

Clearly, our first implementation of ASS†UCE has a performance inferior to the state of the art tools used with the serial strategy. The first point to stress is that ASS†UCE solves a harder problem than either NOVA or STAMINA. Second, the heuristics behind the constraint satisfaction of NOVA guarantee that the least code length is obtained, while the current algorithm in ASS†UCE cannot do the same. This results in bigger implementations. The order of magnitude of each of the measured parameters is nonetheless the same, and some gain in the sparsity of the resulting implementations indicate that these are more adapted to further treatment by topological optimization tools like PLA folding.

Table 2 shows the comparison between the MEMCE program and STAMINA. To provide equal comparison conditions we parameterized STAMINA in this case with the **-s 0** to force its algorithm to perform exact minimization.

We can observe that in all tests sets, the MEMCE program obtained a comparable performance with the STAMINA tool. Exceptions are the S1a and S298 benchmarks. Due to the reduced number of completely specified FSMs in the test set we were not able to obtain a more significant set of results. However, we may see that again, the order of magnitude of the execution time for both STAMINA and MEMCE is the same. Being both exact minimizers, STAMINA and MEMCE generate identical FSMs as to the number of states, the absolute minimum.

The results were obtained running all programs on a Sun Sparcstation 20/60 under Solaris 2.5.1. The programs ASS†UCE and MEMCE were also ported to platforms running Linux OS. Both distributions are freely

FSM	Memce_time	Stamina_time
bbara	0.04	0.01
donfile	0.01	0.01
modulo12	0.01	0.00
opus	0.08	0.00
s1a	0.98	0.06
s27	0.02	0.01
s298	3.01	0.43
SUM	4.15	0.52

Table 2: MEMCE versus Stamina

available, together with the XASS†UCE interface. They can be retrieved via Internet either by FTP or WWW at the addresses ftp://ftp.inf.pucrs.br/pub/groups/gaph/Asstuce/ and http://www.inf.pucrs.br/~gaph/Asstuce/, respectively.

5 Conclusions

We proposed an environment applicable to both, research and teaching. This environment is useful for helping in the design of finite state machines to be implemented in hardware.

This environment is based upon a formal framework that supports the formulation of several VLSI design problems which are being addressed by our research group. The framework provides a unifying principle for solving each individual problem by a constraint modeling step followed by a single constraint satisfaction step.

We consider that the ASS†UCE principles are unique in combining state of the art research results (such as those found in SIS and commercial production environments) with tools that are controllable and observable (such as those available in academic and commercial educational environments). It is the authors' belief, based on current teaching experience, that it is more profitable avoiding the proposal of aids such as languages and tools specifically adapted to digital systems design teaching. Instead, we suggest the careful adaptation of research and/or production tools to pedagogical needs. This has as effect that students are finally better prepared for using real-world tools and solving real-world problems.

Several works are presently under way with the enhancement of the ASS†UCE environment in mind. The work is undertaken in parallel at either educational and research level, with a current greater emphasis in research.

6 Acknowledgements

The authors would like to gratefully acknowledge the continued support of the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), Brazil, to this work. Several scholarship and research grants have been awarded to the authors, among which grants 205411/88-6, 520523/94-6 and 520091/96-5. We would also like to acknowledge the contributions of Fábio Clever Vencato and Guilherme Saueressig in the implementation of the XASS†UCE and MEMCE programs, respectively.

All development tools employed to accomplish this work are freely available for academic purposes. We acknowledge the people and organizations behind every such initiative.

References

[1] M. J. Avedillo, J. M. Quintana, and J. L. Huertas. SMAS: a program for concurrent state reduction and state assignment of finite state machines. In *Proceedings of the IEEE International Symposium on Circuits and Systems - ISCAS*, pages 1781–1784, Singapore, June 1991. The Institute of Electrical and Electronics Engineers.

[2] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, Aug. 1986.

[3] N. L. V. Calazans. *State minimization and state assignment of finite state machines: their relationship and their impact on the implementation.* PhD thesis, Université Catholique de Louvain, Louvain-la-Neuve, Belgium, Oct. 1993.

[4] N. L. V. Calazans. Boolean constrained encoding: a new formulation and a case study. In *Proceedings of the IEEE International Conference on Computer-Aided Design - ICCAD*, pages 702–706, San Jose, Nov. 1994.

[5] N. L. V. Calazans. Considering state minimization during state assignment. In *I Ibero American Microelectronics Conference - X Congress of the Brazilian Microelectronics Society*, pages 49–58, Canela, RS, July 1995.

[6] G. de Micheli. *Synthesis and optimization of digital circuits.* McGraw-Hill Series in Electrical and Computer Engineering. McGraw-Hill, Inc., New York, NY, 1994.

[7] G. de Micheli, R. K. Brayton, and A. Sangiovanni-Vincentelli. KISS: a program for optimal state assignment of finite state machines. In *Proceedings of the IEEE International Conference on Computer-Aided Design - ICCAD*, pages 209–211, Santa Clara, CA, Nov. 1984. The Institute of Electrical and Electronics Engineers.

[8] A. Grasselli and F. Luccio. A method for minimizing the number of internal states in incompletely specified sequential networks. *IRE Transactions on Electronic Computers*, EC-14:350–359, June 1965.

[9] G. D. Hachtel, J.-K. Rho, F. Somenzi, and R. Jacoby. Exact and heuristic algorithms for the minimization of incompletely specified state machines. In *Proceedings of the European Conference on Design Automation - EDAC*, pages 184–191, Amsterdam, Feb. 1991.

[10] J. Hartmanis and R. E. Stearns. Some dangers in state reduction of sequential machines. *Information and Control*, 5:252–260, Sept. 1962.

[11] Z. Hasan, J.-J. Shen, and M. J. Ciesielski. State assignment for general FSM networks. In *Proceedings of the European Conference on Design Automation - EDAC*, pages 245–249, Brussels, Mar. 1992.

[12] F. J. Hill and G. R. Peterson. *Computer aided logical design with emphasis on* VLSI. John Wiley & Sons, Inc, New York, NY, fourth edition, 1993.

[13] C. A. R. Hoare. *Communicating sequential processes.* Prentice/Hall International, 1985.

[14] J. Hopcroft. *Theory of machines and computations*, chapter An $n \log n$ algorithm for minimizing states in a finite automaton. Academic Press, New York, NY, 1971. Z. Kohavi and A. Paz, eds.

[15] E. B. Lee and M. Perkowski. Concurrent minimization and state assignment of finite state machines. In *Proceedings of the 1984 International Conference on Systems Man and Cybernetics*, pages 248–260, Halifax, Oct. 1984.

[16] B. Lin and A. R. Newton. Implicit manipulation of equivalence classes using binary decision diagrams. In *Proceedings of the International Conference on Computer Design: VLSI in Computers and Processors - ICCD*, pages 81–85, Cambridge, MA, Oct. 1991. The Institute of Electrical and Electronics Engineers.

[17] K. Mehlhorn, S. Näher, and C. Uhrig. *LEDA User Manual - Version 3.5.* Max-Planck-Institut für Informatik, Saarbrücken, Germany, 1997.

[18] R. Murgai, R. K. Brayton, and A. Sangiovanni-Vincentelli. Optimum functional decomposition using encoding. In *Proceedings of the ACM/IEEE Design Automation Conference - DAC*, San Diego, California, June 1994. The Institute of Electrical and Electronics Engineers.

[19] Z. Navabi, R. Swanson, and F. J. Hill. *User Manual for AHPL Simulator (HPSIM2) & AHPL Compiler (HPCOM).* Engineering Experiment Station College of Engineering and Mines, The University of Arizona, Tucson, Arizona, Feb 1993.

[20] J. K. Ousterhout. *Tcl and the Tk Toolkit.* Addison-Wesley Professional Computing Series. Addison-Wesley, Reading, MA, 1994.

[21] M. C. Paull and S. H. Unger. Minimizing the number of states in incompletely specified sequential switching functions. *IRE Transactions on Electronic Computers*, EC-8:356–367, Sept. 1959.

[22] C. P. Pfleeger. State reduction in incompletely specified finite-state machines. *IEEE Transactions on Computers*, C-22(12):1099–1102, Dec. 1973.

[23] A. Saldanha, T. Villa, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. A framework for satisfying input and output encoding constraints. In *Proceedings of the ACM/IEEE Design Automation Conference - DAC*, pages 170–175, San Francisco, CA, June 1991.

[24] G. Saueressig. Memce: Um algoritmo de minimização de estados para o ambiente ASSTUCE. End of term work, Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, 1996. (In Portuguese).

[25] E. M. Sentovich and et al. Sis: A system for sequential circuits synthesis. Memorandum no ucb/erl m92/41, Electronics Research Laboratory, Department of Electrical Engineering and Computer Science, University of California, Berkeley, May 1992.

[26] F. C. Vencato. XASSTUCE: Uma interface gráfico-textual para o ambiente exploratório de máquinas de estados finitas ASSTUCE. End of term work, Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, 1996. (In Portuguese).

[27] T. Villa and A. Sangiovanni-Vincentelli. NOVA: state assignment of finite state machines for optimal two-level logic implementation. *IEEE Transactions on Computer-Aided Design*, 9(9):905–924, Sept. 1990.

[28] Xilinx, Inc. The future of FPGAs. White paper, Xilinx, Inc, 1997. Available at the URL http://www.xilinx.com/prs_rls/5yrwhite.htm.

[29] S. Yang. Logic synthesis and optimization benchmarks. Technical report, Microelectronics center of North Carolina, Research Triangle Park, NC, Jan. 1991. Version 3.0.