Synthia: Synthesis of Interacting Automata Targeting LUT-Based FPGAs

George A. Constantinides[1], Peter Y. K. Cheung[1], and Wayne Luk[2]

[1] Electrical and Electronic Engineering Dept., Imperial College, London, U.K.
george.constantinides@ieee.org
p.cheung@ic.ac.uk
[2] Department of Computing, Imperial College, London, U.K.
wl@doc.ic.ac.uk

Abstract. This paper details the development, implementation, and results of *Synthia*, a system for the synthesis of Finite State Machines (FSMs) to field-programmable logic. Our approach uses a novel FSM decomposition technique, which partitions both the states of a machine and its inputs between several sub-machines. The technique developed exploits incomplete output specifications in order to minimize the interconnect complexity of the resulting network, and uses a custom Genetic Algorithm to explore the space of possible partitions. User-controlled trade-off between logic depth and logic area is allowed, and the algorithm itself during execution determines the number of sub-FSMs in the resulting decomposition. The results from MCNC benchmarks applied to Xilinx XC4000 and Altera FLEX8000 devices are presented.

1. Introduction

Finite State Machine (FSM) decomposition is the implementation of a large FSM as a network of smaller interacting FSMs, a problem studied since the late 1960s [1]. In VLSI architectures, FSM decomposition is useful for a number of reasons. By controlling the topology and manner in which the machine is decomposed, it is possible to aim for an implementation with characteristics such as: high speed, as decomposition can usually lead to a reduction in logic depth [2]; low area, as state-encoding heuristics will often cope more efficiently with each of the smaller sub-FSMs than with the large lump-FSM [3]; reduction in interconnect complexity [4], and I/O minimization [5].

This paper describes one approach to FSM decomposition, which specifically targets LUT-based FPGA implementations. The technique partitions the states of a large FSM between several sub-machines, while also partitioning its inputs. Output don't-care conditions are exploited in order to minimize the interconnect complexity of the resulting network, and a custom Genetic Algorithm is used to explore the space of possible input partitions. A user-controlled area/speed trade-off is allowed, and the algorithm itself determines the number of sub-FSMs in the decomposition.

P. Lysaght, J. Irvine, and R. Hartenstein (Eds.): FPL'99, LNCS 1673, pp. 323-332, 1999.

2. Background

Feske et al. [3], have developed an approach to FSM decomposition which operates at the State Transition Graph level. Decomposition strategies that partition the STG allow a wider solution space to be searched by the following phases of synthesis [2]. They proceed by breaking the STG into a number of sub-STGs, each containing an extra 'wait' state representing all states outside the sub-STG's domain. The sub-FSMs pass messages to each other when a transition occurs which would cross a sub-STG boundary, indicating which state to enter. Unlike most techniques for FSM decomposition, [3] not only allows an arbitrary number of sub-FSMs to be formed, but decides on that number itself. The simplicity of the messages passed means that complex don't care conditions, arising from the association of particular inputs with particular groups of states, are not exploited. The authors report improvements compared to a one-hot implementation: reduction in circuit depth (29%) and number of logic blocks (38%) when circuits were mapped to a Xilinx XC4000 FPGA.

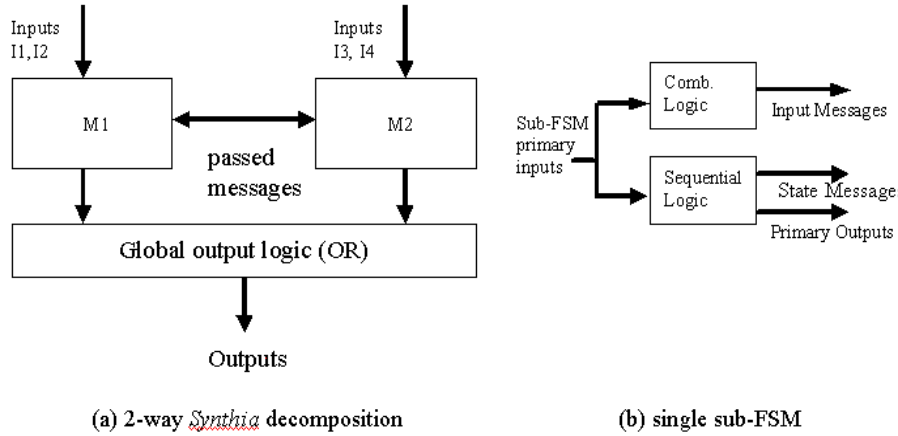
Yang et al. [4], have developed an FSM decomposition to minimize the complexity of VLSI interconnection. The first step is to partition the set of inputs between *n* sub-FSMs, each containing all the states of the lump FSM. The communication between sub-FSMs necessary for each sub-FSM to calculate its correct (next-state, output) combination is then found. Outputs are partitioned between sub-FSMs, leading to possible redundancy in states. This is followed by a state-minimization on each sub-FSM. Because the messages passed between sub-FSMs are only functions of machine inputs, the logic to generate them is combinational and simple compared to the sequential logic for the sub-FSM. This technique exploits more complex don't care conditions, leading to a significant reduction in interconnect complexity. This is important for FPGA design, as routing resources tend to be limited. However, as developed in [4], little advantage is taken of incompletely specified FSMs. In addition, the user is required to specify the number of sub-FSMs in advance of algorithm execution. In a later paper [8], the authors apply their technique to FPGA implementations, claiming a significant speedup at the cost of a 44% increase in the number of logic blocks.

3. *Synthia*

3.1 Decomposition Topology

The *Synthia* system performs a partitioning of both inputs and states between sub-FSMs. The state partitioning is performed by adapting the linear partitioning approach described in [3]. The constraint that each lump-FSM state is assigned to exactly one sub-FSM is retained. However, the constraint that every input must be available to every sub-FSM [3], is relaxed. Instead, our approach introduces two types of messages which may be passed between sub-FSMs. Message from one sub-FSM can either instruct another sub-FSM to enter a particular state, or inform another sub-FSM of the status of its inputs. It is worth noting that the two different types of

message are never needed simultaneously, since if a given sub-FSM is in its wait-state, then it is only waiting for next-state type messages, whereas if it is not in the wait-state, it will never receive next-state type messages. Shown in Fig. 1(b) is a single sub-FSM.



(a) 2-way *Synthia* decomposition **(b) single sub-FSM**

Fig. 1. Synthia decomposition topology

3.2 Input Messages

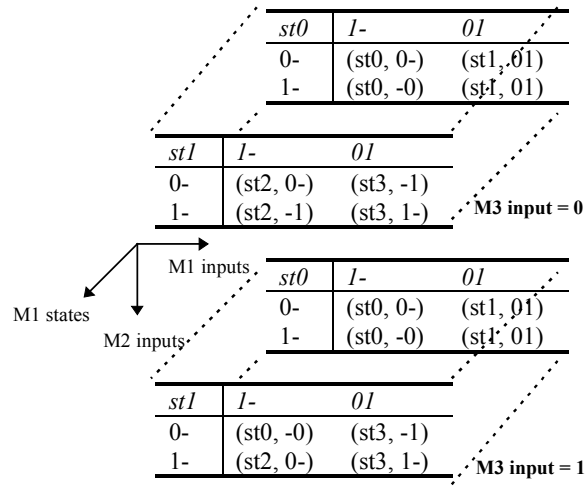
Yang [4] uses *n*-dimensional state-transition matrices in order to find what messages to be pass between *n* sub-FSMs. Here we extend the idea of state-transition matrices to better cope with two phenomena not present in [4]: in our decomposition, states belong to only one sub-FSM and incompletely specified outputs may be present. Assuming that the input and state partitioning has already been performed using the method described in section 3.3, an (*n*+1)-dimensional matrix may then be built for each sub-FSM *M*. One dimension corresponds to the state of *M*. The other *n*-dimensions correspond to possible input cubes seen by each sub-FSM, and the data value at each coordinate represents a next-state/output combination. Consider the state-machine specification shown in Fig. 2(a). Let us assume that st0 and st1 are contained in sub-FSM M1, the other states belonging to other sub-FSMs. Further, assume that the first two inputs are sent to sub-FSM M2, and the final one to sub-FSM M3. A 4-D state-transition matrix is constructed for M1 shown in Fig. 2(b).

It may now be possible to merge axis headings on each of the (*n*-1) dimensions not representing M1, while still retaining all necessary information, thus reducing interconnect complexity. We aim to minimize interconnect complexity for two reasons: routing resources are scarce in FPGAs, and small sub-FSMs are likely to make heavy use of fast local FPGA interconnect, whereas interconnect between sub-FSMs is likely to make more heavy use of slower, non-local routing resources.

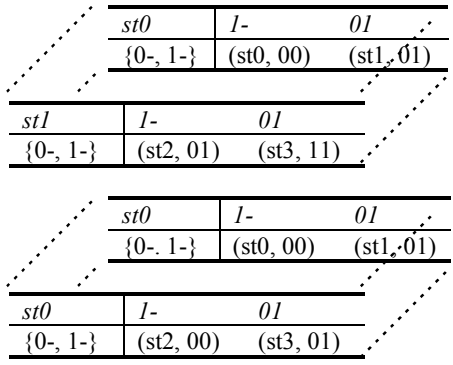
```
.i 5
.s 4
.o 2

0-1-- st0 st0  0-
0-01- st0 st1  01
1-1-- st0 st0  -0
1-01- st0 st1  01
0-1-0 st1 st2  0-
0-1-1 st1 st2  -0
0-01- st1 st3  -1
1-1-0 st1 st2  -1
1-1-1 st1 st2  0-
1-01- st1 st3  1-
  ...
```

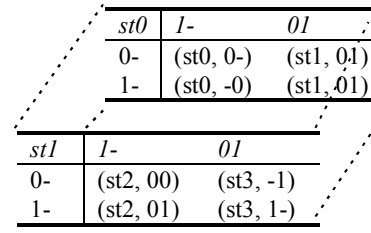
(a) Partial KISS specification



(b) unoptimized state-transition matrix with output don't cares



(c) one optimization of Fig. 2(b)



(d) another optimization of Fig. 2(b)

Fig. 2. State-transition matrices and their optimization

When there are don't care conditions on the output specifications, it is possible that there is no single unique solution to the problem. For example, the reduced matrix could take the form of Fig. 2(c) or 2(d) equally. This situation arises because don't care conditions may be expanded into concrete '0' or '1' specifications as a result of one merge, which would conflict with the concrete value arising from another possible merge. A heuristic has been implemented, as detailed in section 3.4, to select which merges to implement.

3.3 Primary Input and State Partitioning

A Genetic Algorithm (GA) is used to search the space of possible input partitions. We may represent the mapping between primary inputs and sub-FSMs by a vector **I**, indexed by primary input number, and with entries in the range 0 to (n-1), indicating to which sub-FSM each input is assigned. When viewing the problem in this way, a clear demarcation already exists between alleles in a chromosome - the index of the vector **I**. With larger allele size there is not full genetic control over the search-space, whereas with a smaller size (say bits, more typically used in a GA) no extra information is present. The unusual properties of the GA applied are: valid allele values are 0 to (n-1), n may vary during algorithm execution, and mutation consists of randomly choosing an allele value between 0 and (n-1).

Rather than extending the GA approach to the partitioning of states, certain *a-priori* information may be used effectively: states that are successors or predecessors of other states in the STG are more likely to benefit from incorporation within a single sub-FSM than any two states chosen at random. Such sub-FSM networks are likely to have fewer inter-FSM transitions, and in addition, the results of [3] indicate that particular inputs tend to be associated with particular sub-graphs of the STG. Indeed, the algorithm described in [3] is a heuristic incorporating elegantly this prior knowledge. The approach taken in that paper has been modified to incorporate the two different message types, and the presence of an extra phase - that of input assignment.

3.4 Algorithms

At the core of the algorithm employed is the integration of the two phases: an STG-level heuristic for state-partitioning, and a GA for input partitioning. The solution taken is to interleave the two algorithm phases in the following way. The top-level algorithm is shown in pseudo-code below.

```
OptimizeFSM() {
          ga_optimize( start_its );
          do {
            foreach sub-FSM B {
              do {
                gain = Optpart( B, best_chromosome );
              } while( gain > 0.0 );
            }
            cleanup_popul( popul, decomp );
            ga_optimize( popul, next_its );
          } while( newCost < oldCost );
}
```

The algorithm starts with an initial state partition of one state per sub-FSM (a one-hot encoding on the lump-FSM). After performing a GA optimization with respect to that state-partition, the main loop is entered. `Optpart` works on one block of

M0	-00	011	010	001
-00				
011	T			
010	T	T		
001	F	F	F	

M1	--0	001
--0		
001	T	

M2	000	100	110	111
000				
100	F			
110	F	T		
111	T	F	F	

Fig. 3. Three compatibility tables

Table 1. *Synthia* run-times

Ckt	Run-time
bbara	4.01s
beecount	1.44s
cse	15m 17s
shiftreg	2m 57s
dk14	6m 47s
dk27	1m 54s
dk512	5m 31s
ex6	11m 59s
opus	5m 42s
s208	13m 45s

the state-partition at a time, trying to suck-in any state which has a predecessor or successor state within another block. If this results in a positive gain, the move is retained. If this the results in an empty block, all references to that block in the chromosome **I** are set to point to the new block containing that state. (A similar function is performed by `cleanup_popul`). Two, possibly different, numbers of iterations are used in the GA - one inside and one outside the main loop body. This is because it is quite likely that the population of input partitions before entering the state-partitioning heuristic is a good *first-guess* for the GA after exiting the heuristic. This insight was confirmed by the results collected.

The delay and area of the resulting network is estimated from the optimized state-transition matrices. After constructing an unoptimized state-transition matrix (as discussed in section 3.2), the algorithm optimizes the matrix by merging axis headings as much as possible. The heuristic iteratively merges axis headings, two at a time. It is a non-backtracking approach: the merge chosen at any given step is the one judged 'most-likely' to result in the largest interconnect complexity reduction, and once that choice is made it will not be changed at a later stage. For each axis in the given state-transition matrix, a *compatibility table* is constructed. This table indicates which axis headings are mergible with which others (to be determined by searching the matrix entries). The example compatibility tables are shown in Fig. 3. Note that these are lower triangular matrices with binary entries (the compatibility relation is bi-directional and Boolean). The heuristic proceeds by counting the number of possible merges on each axis (3 for M0, 1 for M1 and 2 for M2). The axis with the greatest number is chosen for merging, in the hope that some merges will still be possible after the current one. Then the heuristic picks the heading H_1 with the largest number of possible merges (in this case either -00, 011 or 010). Finally, the pair to merge is completed by choosing the heading with the next largest number of possible merges H_2, subject to the constraint that H_1 and H_2 are mergible. After merging the two axis headings, and adjusting their entries (replacing don't cares by '0' or '1') as necessary, the heuristic is ready for its next iteration.

After optimization, the matrices form a complete specification for the network of sub-FSMs, which is then passed to a function in order to calculate a cost estimate. This proceeds in four stages: input message encoding, state message encoding, STG construction for each sub-FSM and finally delay and area estimation.

The message assignments are arbitrary minimum length encodings. The delay and area of each sub-FSM are estimated separately from its neighbours for speed of execution. This is done using SIS [9] procedures for LUT-based FPGAs, as developed by Murgai [10], using a JEDI [11] minimum length state encoding on each individual sub-FSM. The worst-case logic depth (and therefore the estimated worst-case delay) of the resulting network occurs when more than one sub-FSM is involved in a transition. The maximum number of sub-FSMs that may be involved in a transition is two. Hence if all the logic depths of all sub-FSMs arewritten as d_0, d_1, ..., d_n, with $d_0 \geq d_1 \geq ... \geq d_n$, then the worst-case logic depth is bounded above by d_0+d_1, and below by d_0. The returned delay estimate is $d_0 + 0.5d_1$. The area estimate returned for the network is simply the sum of all area estimates for the sub-FSMs. The two costs (area and logic depth) are combined in a linear manner with a user-controlled factor in order to return a single cost-function value.

4. Experimental Results

The algorithm described was implemented and integrated within the SIS logic synthesis package [9]. The implementation takes an STG as input and produces a file of hierarchical VHDL source-code describing the network of interacting FSMs. A diagram illustrating a typical design-flow when using *Synthia* is shown in Fig. 4.

The format for describing FSMs in SIS is the KISS format. This can be acquired either directly from design specification, or through extraction from an HDL description. Synopsys software [12] has such an extraction feature, 'extract'. The state-machine format used by Synopsys is another abstract description, '.st' for which we have written a converter to KISS.

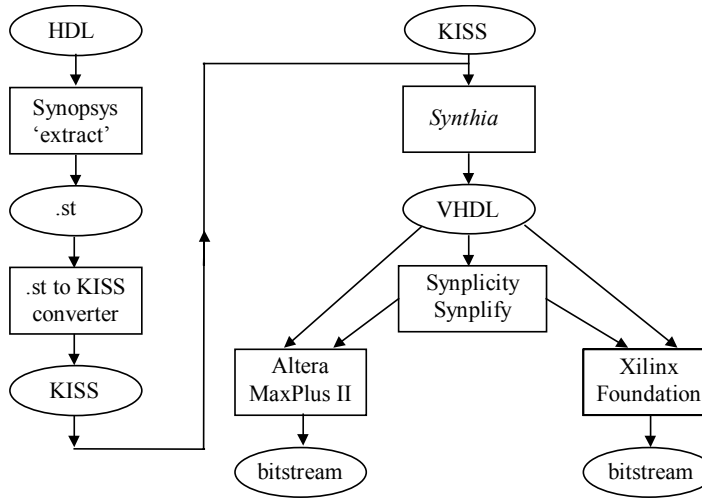


Fig. 4. *Synthia* design flow

FSMs from a subset of the MCNC benchmark set were decomposed using the algorithm. The resulting VHDL code was compiled by Xilinx Foundation tools [13] targeting Xilinx XC4003EPC84-1, and Altera MaxPlus II [14] targeting EPF8282A-2. In addition, the VHDL code was synthesised by Synplify [15], and the resulting network passed through MaxPlus II and Xilinx M1.
The overall number of Altera LCs and Xilinx CLBs were collected, alongside the reported maximum clock frequencies. For the purposes of comparison, each benchmark was also encoded in a one-hot style by JEDI, automatically written as VHDL code, and compiled as above by the FPGA vendor tools. These results are shown graphically in Fig. 5. In addition, *Synthia* run-times on a Pentium II 233 running Linux are reported in Table 1.

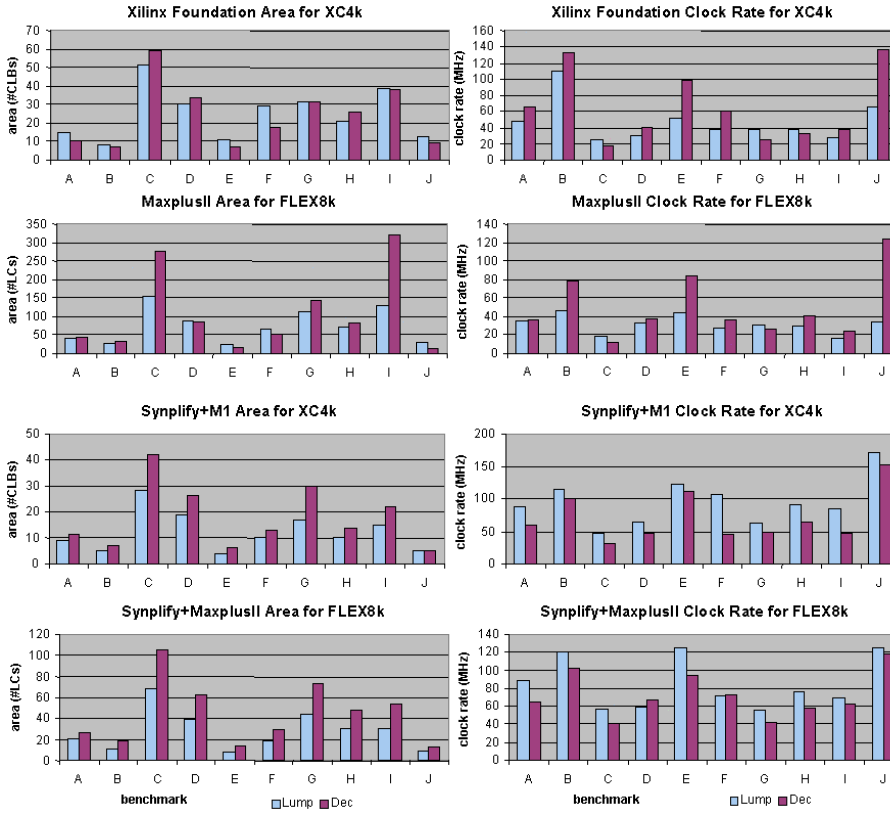
The results shown an average speedup of 29% for XC4k using Xilinx Foundation, and 49.7% for FLEX8k using MaxPlus II. These figures change to speedups of 37% and 55% respectively, when counting negative speedups as 0 (i.e. when either one-hot or *Synthia* is used, whichever is faster). Also shown is an average area reduction of 10% for XC4k using Xilinx Foundation and an average area increase of 16% for FLEX8k using MaxPlus II (becoming a 15% and 12% reduction when modified as above). Also illustrated are the results for XC4k using Synplify, showing an average area increase of 39.13% and clock rate reduction of 27.74% (becoming 0% and 0% when modified as above). Finally the results for FLEX8k using Synplify, show an average area increase of 59% and clock rate reduction of 14.67% (becoming a 0% area reduction and a 1.5% speedup when modified as above).

5. Conclusion

A novel algorithm for the automatic decomposition of FSMs into networks of interacting sub-FSMs has been presented. The technique developed uses a combination of a Genetic Algorithm and a state-partitioning heuristic, along with a further heuristic which exploits output don't-care conditions in order to minimize the interconnect complexity of the resulting network.

In the case of a direct synthesis through Altera MaxPlus II or Xilinx Foundation (see Fig. 4), *Synthia* has significantly improved the performance of the later stages of synthesis, leading to increases in maximum permissible clock frequency, and in the Xilinx case, simultaneous reduction in area. The results from Synplify tend to show the reverse behaviour.

The speedups gained with the Xilinx XC4k, though quite impressive, are relatively small compared to those gained with the Altera FLEX8k. This is almost certainly at least in part due to the approximation made at the cost estimation stage, of a circuit as a network of 4-LUTs. The Xilinx logic block, being a combination for more than one type of LUT, is less suited to this approximation than the Altera device.



Key to benchmarks									
A	bbara	C	cse	E	dk27	G	ex6	I	s208
B	beecount	D	dk14	F	dk512	H	opus	J	shiftreg

Fig. 5. Experimental results

The question of choosing an efficient message encoding for inter-FSM communication stands out as a missing part of this work. This question is related to the ongoing research topic of hierarchical synthesis of sequential circuits.

Acknowledgments

The Authors would like to acknowledge the support of Alan Marshall, Nick Wainwright and John Lumley of Hewlett Packard Laboratories, Bristol, UK. In addition, donations of software tools were gratefully received from Xilinx and Altera.

References

1. Hartmanis, J., Stearns, R. E.: Algebraic Structure Theory of Sequential Machines. Prentice-Hall, Inc., Englewood Cliffs, New Jersey (1966)
2. Ashar, P., Devadas, S., Newton, A. R.: Sequential Logic Synthesis. Kluwer Academic Publishers, Boston Dortrecht London (1992)
3. Feske, K., Mulka, S., Koegst, M., Elst, G.: Technology-Driven FSM Partitioning for Synthesis of Large Sequential Circuits Targeting Lookup-Table Based FPGAs. In: Luk, W., Cheung, P. Y. K., Glesner, M. (eds.): Field-Programmable Logic and Applications. Lecture Notes in Computer Science, Vol. 1304. Springer-Verlag, Berlin Heidelberg New York (1997)
4. Yang, W. L., Owens, R. M., Irwin, M. J.: Multi-way FSM decomposition based on interconnect complexity. Proc. EURO-DAC '93. IEEE, Piscataway, New Jersey (1993) 390-395
5. Kuo, M. T., Liu, L. T., Cheng, C. K.: Finite State Machine Decomposition for I/O Minimization. Proc. ISCAS '95, Vol. 2. IEEE, Piscataway, New Jersey (1995) 1061-1064
6. Xilinx XC4000 Data Book, Xilinx Inc., San Jose (1991)
7. FLEX8000 Handbook, Altera Corp., San Jose (1994)
8. Yang, W. L., Owens, R. M., Irwin, M. J.: FPGA-based synthesis of FSMs through decomposition. Proc. GLSV '94. IEEE, Piscataway, New Jersey (1994) 97-100
9. Sentovich, E. M., Singh, K. J., Lavagno, L., Moon, C., Murgai, R., Saldanha, A., Savoj, H., Stephen, P. R., Brayton, R. K., Sangiovanni-Vincentelli, A.: SIS: A System for Sequential Circuit Synthesis. UCB/ERL M92/41 Memorandum, Berkeley (1992)
10. Murgai, R., Brayton, R. K., Sangiovanni-Vincentelli, A.: Logic Synthesis for Field Programmable Gate Arrays. Kluwer Academic Publishers, Boston Dortrecht London (1995)
11. Lin, B., Newton, A. R.: Synthesis of Multiple Level Logic from Symbolic High-Level Description Languages. Proc. VLSI '89, Munich (1989)
12. FPGA Express Online Help. Synopsys, Inc.
13. Xilinx Foundation Tools Online Help. Xilinx, Inc.
14. Max+Plus II Online Help. Altera Corp.
15. Synplify Users Manual. Synplicity, Inc. Sunnyvale, CA (1998)