8th International Conference on DEVELOPMENT AND APPLICATION SYSTEMS
Suceava, Romania, May 25 – 27, 2006

APPLICATIONS OF FINITE STATE MACHINES GENERAL DECOMPOSITION METHOD WITH OPTIMIZATION

Codrin PRUTEANU
"Gh. Asachi" Technical University
Computer Engineering Department
Bld. Mangeron, 53A, 70050, IASI
codrin.pruteanu@gmail.com

Dan GÂLEA
"Gh. Asachi" Technical University
Computer Engineering Department
Bld. Mangeron, 53A, 70050, IASI
dangalea@wtci.ro

Cristian-Győző HABA
"Gh. Asachi" Technical University
Electrical Engineering Department
Bld. Mangeron, 53, 70050, IASI
cghaba@yahoo.com

Abstract. *The reduction of FSM's in modern circuits during the synthesis flow is an NP complete problem. Heuristics are used to divide single state machines into networks of state machines. The resulting network of state machines is composed of interacting submachines. We implemented a pre-synthesis EDA tool called fsmtool which uses the General Decomposition Method (GDM) of FSM's. This tool will divide a single FSM into a network of interacting FSM's by reducing each submachine's complexity while attempting to minimize the number of the obtained submachines. The application of GDM to any FSM will generate results which will cover other decomposition methods like: cascade decomposition, serial decomposition, parallel decomposition, and factorization.*
Keywords: *Logic Synthesis, General Decomposition, Optimization, System-Level Design.*

1. Introduction

Moore's law predicts that the capacities of the integrated circuits will double every 18 months. As a result the clock periods are shrinking which makes it more difficult for complex circuits such as FSM's to meet timing. Larger chips can result in more complex FSM's. The synthesis of complex FSM's is an NP complete problem. Current synthesis tools are unable to keep up with the increased complexity of the FSM's. Designer productivity and EDA tools have not kept the step with the increased clock frequencies and transistor counts. The design of digital chips as a result is becoming more and more dependent on the back end timing closure loop. On projects with large transistor counts the back end timing closure can result in many gate levels engineering change orders (ECO's) to meet timing. Electronic Design Automation (EDA) tools currently produce suboptimal circuit realizations for complex circuits. Timing closure of designs with complex circuits is an area of ongoing research in EDA companies and research labs. Functional decomposition has been successfully used among others for logic synthesis for FPGA platforms [7]. General decomposition can be used to analyze and decompose any discrete, binary, multi-value or symbolic system in the fields of modern engineering and science. L. Józwiak formulated the theory and methodology of general decomposition that deals with realization of the behavior of large FSMs by networks of interconnected, smaller FSMs operating in parallel (so called partial machines). In this paper we developed a software tool which applies the **GDM** to FSM's to produce circuits optimized for timing and area [10]. We tested the effectiveness of fsmtool on a set of standard benchmarks and a number of generated FSMs in typical circuits. The results of our experiments indicate that the approach proposed in this paper compares favorably to the results obtained from other FSM decomposition methods.

The resulting interacting FSM has a smaller gate count and higher operating frequency than the benchmark circuits. The resulting netlists were compared with the pre-decomposed netlists using timing analysis. The main subject of this paper is to optimize the circuit synthesis of finite state machines (FSMs). The FSM is one of the basic digital components in a modern computer chip. A FSM describes the behavior of the system in terms of distinct states that the system can be in and transitions between the states.

The combination of the current state of the FSM and the input signals determine the output produced by the FSM and its next state. We will

use the **GDM** method to decompose and transform the original FSM into a network of interacting FSM's (submachines). We estimated the decomposed circuit area using the gate level netlist produced from the technology mapping step. The target libraries used for benchmarking fsmtool were some Field Programmable Gate Array (FPGA) libraries from Xilinx and Altera. The selection of LUT's in the target FPGA library will be based on the ***fsmtool*** encoding of states for each of the interacting submachines. Decomposition is a natural method of synthesizing complex circuits. The **GDM** method splits a complex circuit into smaller sub circuits which are connected to each other.

The **GDM** method creates a new state mapping of each submachine based on the fan-out of each of the state variables and the inputs to the submachines. The problem of state encoding is also considered a special case of general decomposition. One particular special case of general decomposition is functional decomposition. It deals with state machines having a single state and trivial state behavior, i.e. with combinational functions. In this approach, a large combinational function is realized by a network of smaller sub functions.

2. Problem Formulation

A FSM can be classified as deterministic or non-deterministic, and completely or incompletely specified. A completely specified state machine M can be described by a tuple $\boldsymbol{M} = (I, S, O, \delta, \lambda)$, where $\boldsymbol{I}$ is a set of primary inputs, $\boldsymbol{S}$ is a set o state symbols, $\boldsymbol{O}$ is a set of primary outputs, $\boldsymbol{\delta} : \boldsymbol{I} \times \boldsymbol{S} \rightarrow \boldsymbol{S}$ is the next state function, and $\boldsymbol{\lambda} : \boldsymbol{I} \times \boldsymbol{S} \rightarrow \boldsymbol{O}$ is the output function for Mealy machines or $\boldsymbol{\lambda} : \boldsymbol{S} \rightarrow \boldsymbol{O}$ is the output function for Moore machines. Moore machines can be considered a special case of Mealy machines. Therefore the set of theories and methods used for Mealy machines will be also applicable to Moore machines. For this reason in this paper we will focus on Mealy FSMs. In the case of incompletely specified FSMs, $\boldsymbol{I}$ represents a finite set of inputs, $\boldsymbol{S}$ is a finite non-empty set of internal states, $\boldsymbol{O}$ is a finite set of outputs, $\boldsymbol{\delta} : \boldsymbol{I} \times \boldsymbol{S} \rightarrow \boldsymbol{2^S}$ is the next-state function, and $\boldsymbol{\lambda} : \boldsymbol{I} \times \boldsymbol{S} \rightarrow \boldsymbol{2^O}$ is the output function. A FSM can be described in a tabular form by a state transition table (STT), in a graphical form by a state transition graph (STG), or by a RTL code. ***fsmtool*** reads FSMs described in kiss input format and generates kiss or verilog output code. The states of an FSM are listed in the prototype FSM as symbolic state names. Each symbolic state name has a corresponding binary value. The decomposition method will change the number of states and will re-encode the state values. Symbolic minimization performs the logic minimization phase before the FSM state encoding.

Symbolic minimization was implemented by De Micheli in KISS [3] in 1985. Some of the shortcomings of KISS are addressed in its successor NOVA [13]. NOVA takes more efficient and flexible approach to constraints satisfaction, representing it as a graph embedding problem and solving in several, heuristic strategies producing superior results and offering quality/runtime trade-offs. In this paper we will describe the FSMs in a STG format and we will apply the general decomposition method for FSM optimization. The general sequential circuit is presented in Figure 1 and the general decomposition topology is presented in Figure 2.

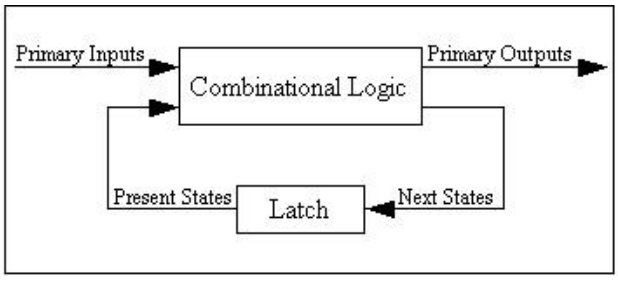

Figure 1. General sequential circuit.

Definition: A **partition** Π on a set of states S is a collection of disjoint subsets of S, called blocks, whose set union is S. A partition Π on the set of states S of a machine M is said to be a **closed partition** if and only if for any two states s and t which are in the same block of Π, and for any input $i \in I$, the next states $\delta(s,i)$ and $\delta(t,i)$

are in a common block of Π. A partition is a **general partition** if it is not closed.

The original machine is called the **prototype machine** and the individual machines that make up the overall realization are called the **submachines**. The machine obtained as a result of the decomposition is called the **decomposed machine** and is implemented as a network of interconnected submachines [6]. The general structure of such a network is shown in Figure 2. Zero partition $\Pi(0)$ denote a partition with |S| blocks such that each block contains exactly one state. It can be shown that a machine M can be decomposed into a set of n interacting machines that perform the same function as M if and only if there exists a set of nontrivial partitions:

$\Pi_1, \Pi_2, \ldots, \Pi_n$ such that $\Pi_1 . \Pi_2 \ldots \Pi_n = \Pi(0)$ [15]

Definition: A legal decomposition of a machine M exists if for a given set of partitions $\Pi_1, \Pi_2, \ldots, \Pi_n$, their product is equal to $\Pi(0)$.

Definition: An **unordered dichotomy** on S is a set of two disjoint subsets of S.

Definition: An **ordered dichotomy** on S is an ordered pair of two disjoint subsets of S. The first subset is referred to as left or zero-block, while the second is a right or one-block.

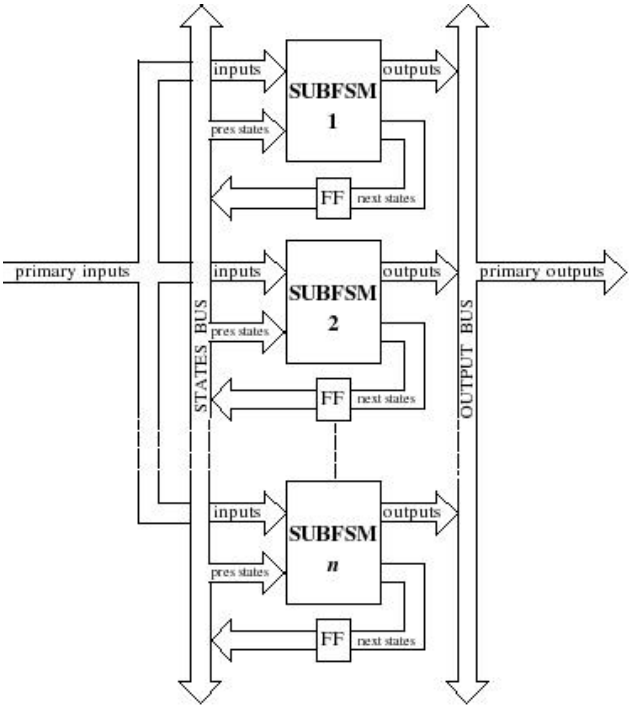

Figure 2. General Decomposition topology.

3. Problem Solution

The **general decomposition method,** or arbitrary decomposition, can have various topologies and covers any type of decomposition. The GDM input constraint is that each state in the prototype machine should have a unique state encoding. The GDM output constraint is that each state encoding in the decomposed machine will also be unique. The next set of constraints which the GDM considers during the optimization step are the outputs from the other submachines which are connected to a particular submachine from the FSM currently being optimized. The type of constraints imposed is briefly illustrated by means of the example topology in Figure 2. For example we use a FSM which has the behavior of a shift register. The internal description could be represented in standard kiss format, by its top-level schematic in Figure 3, or by its state transition graph (**STG**) [1] in Figure 4.

Figure 3. Top-level schematic for the prototype machine.

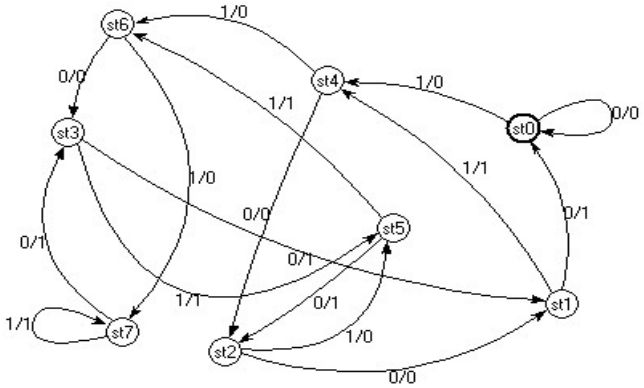

Figure 4. STG for the prototype machine.

Suppose we have a Top-level representation of the shift register known as the prototype machine and a set of partitions randomly generated with ***fsmtool*** tool in an initial step as follows:

{ (*st0 st1*) , (*st2 st3*) , (*st4 st5*) , (*st6 st7*) }

{ (*st0 st2*) , (*st1 st3*) , (*st4 st6*) , (*st5 st7*) }

Then we obtain two submachines that satisfy the general decomposition rules of the prototype machine as we can see in Figure 5 in a RTL schematic top view, or each one of them detailed in Figure 6 and Figure 7 in STG format [8].

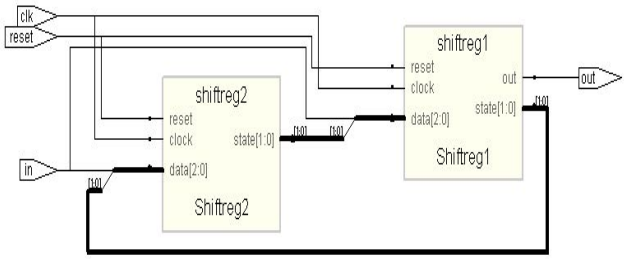

Figure 5. RTL schematic for the decomposed machine.

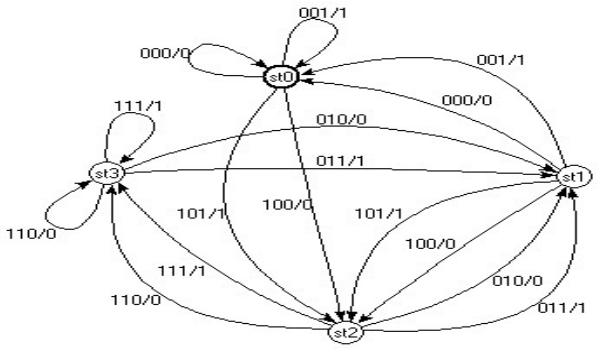

Figure 6. STG for first submachine.

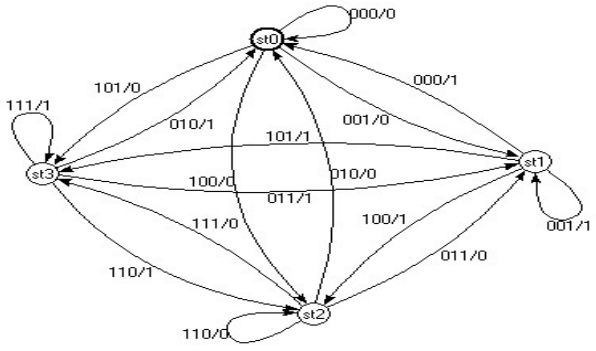

Figure 7. STG for second submachine.

We refer to this topology as the **general decomposition** topology since it can represent any arbitrary decomposition. Parallel and cascade decompositions are special cases of the general decomposition. In this topology each submachine M_i has 3 sets of symbolic inputs.

1. a set of primary inputs I_{Pi}; these inputs can be considered either symbolic or binary, depending on the initial specification

2. a set of state inputs I_{Si}; derived from the states of other submachines; for a machine corresponding to a closed partition, $I_{Si} = 0$.

3. a set of internal states S_i.

The encoding of each submachine can be arbitrary, as long as its internal states are given distinct binary codes. The primary inputs are made available to every submachine. The states of one machine are made available to the others if necessary. The outputs are generated directly by individual submachines. The state information is shared by the interconnected submachines which are not independent of each other. In our example we have a prototype machine *M* with one primary input and one primary output, and 8 internal states S = (*st0, st1, st2, st3, st4, st5, st6, st7*). We decompose this machine into 2 submachines M_1 and M_2 as follows:

M_1: {(*st0 st1*), (*st2 st3*), (*st4 st5*), (*st6 st7*)}

M_2: {(*st0 st2*), (*st1 st3*), (*st4 st6*), (*st5 st7*)}

These two partitions are satisfying the legal decomposition properties since their product equals the unity partition *Π(0)*. The blocks of states of the two orthogonal obtained partitions are then encoded as:

M_1: { x_1 , x_2 , x_3 , x_4 } = { *00, 01, 10, 11* }

M_2: { y_1 , y_2 , y_3 , y_4 } = { *00, 01, 10, 11* }

The newly created states $x_{1..4}$ and $y_{1..4}$ represent the encoded blocks of states from the prototype machine. In this case each submachine requires a $log_2(n) = 2$ flip-flops, where $n = 4$ (the number of states). The corresponding machine M_1 has the input alphabet: $\sum x \prod_B = \{(0,y_1), (1,y_1), (0,y_2), (1,y_2), (0,y_3), (1,y_3), (0,y_4), (1,y_4)\}$

and machine M_2 has the input alphabet:

$\sum x \prod_A = \{(0,x_1), (1,x_1), (0,x_2), (1,x_2), (0,x_3), (1,x_3), (0,x_4), (1,x_4)\}$

The next state functions δ_{M1} and δ_{M2} are defined by the following expression:

$$\delta_M: \sum x \prod_{M2} x \prod_{M1} \rightarrow \prod_{M1} \quad (1)$$

$\delta_{M1}((0, y_1), x_1) = x_1$ $\delta_{M2}((0, x_1), y_1) = y_1$

$\delta_{M1}((1, y_1), x_1) = x_3$ $\delta_{M2}((1, x_1), y_1) = y_3$

$\delta_{M1}((0, y_1), x_2) = x_1$ $\delta_{M2}((0, x_2), y_1) = y_1$

$\delta_{M1}((1, y_1), x_2) = x_3$ $\delta_{M2}((1, x_2), y_1) = y_3$

$\delta_{M1}((0, y_2), x_1) = x_2$ $\delta_{M2}((0, x_1), y_2) = y_1$

$\delta_{M1}((1, y_2), x_1) = x_4$ $\delta_{M2}((1, x_1), y_2) = y_3$

$\delta_{M1}((0, y_2), x_2) = x_2$ $\delta_{M2}((0, x_2), y_2) = y_1$

$\delta_{M1}((1, y_2), x_2) = x_4$ $\delta_{M2}((1, x_2), y_2) = y_3$

$\delta_{M1}((0, y_3), x_3) = x_1$ $\delta_{M2}((0, x_3), y_3) = y_2$

$\delta_{M1}((1, y_3), x_3) = x_3$ $\delta_{M2}((1, x_3), y_3) = y_4$

$\delta_{M1}((0, y_3), x_4) = x_1$ $\delta_{M2}((0, x_4), y_3) = y_2$

$\delta_{M1}((1, y_3), x_4) = x_3$ $\delta_{M2}((1, x_4), y_3) = y_4$

$\delta_{M1}((0, y_4), x_3) = x_2$ $\delta_{M2}((0, x_3), y_4) = y_2$

$\delta_{M1}((1, y_4), x_3) = x_4$ $\delta_{M2}((1, x_3), y_4) = y_4$

$\delta_{M1}((0, y_4), x_4) = x_2$ $\delta_{M2}((0, x_4), y_4) = y_2$

$\delta_{M1}((1, y_4), x_4) = x_4$ $\delta_{M2}((1, x_4), y_4) = y_4$

The output functions f_{M1} and f_{M2} are defined by the following expression:

$$f_M: \sum x \prod_{M1} x \prod_{M2} \rightarrow \lambda \quad (2)$$

$f_{M1}(0, x_1, y_1) = 0$ $f_{M2}(0, y_1, x_1) = 0$

$f_{M1}(1, x_1, y_1) = 0$ $f_{M2}(1, y_1, x_1) = 0$

$f_{M1}(0, x_2, y_1) = 1$ $f_{M2}(0, y_1, x_2) = 1$

$f_{M1}(1, x_2, y_1) = 1$ $f_{M2}(1, y_1, x_2) = 1$

$f_{M1}(0, x_1, y_2) = 0$ $f_{M2}(0, y_2, x_1) = 0$

$f_{M1}(1, x_1, y_2) = 0$ $f_{M2}(1, y_2, y_1) = 0$

$f_{M1}(0, x_2, y_2) = 1$ $f_{M2}(0, y_2, x_2) = 1$

$f_{M1}(1, x_2, y_2) = 1$ $f_{M2}(1, y_2, x_2) = 1$

$f_{M1}(0, x_3, y_3) = 0$ $f_{M2}(0, y_3, x_3) = 0$

$f_{M1}(1, x_3, y_3) = 0$ $f_{M2}(1, y_3, x_3) = 0$

$f_{M1}(0, x_4, y_3) = 1$ $f_{M2}(0, y_3, x_4) = 1$

$f_{M1}(1, x_4, y_3) = 1$ $f_{M2}(1, y_3, x_4) = 1$

$f_{M1}(0, x_3, y_4) = 0$ $f_{M2}(0, y_4, x_3) = 0$

$f_{M1}(1, x_3, y_4) = 0$ $f_{M2}(1, y_4, x_3) = 0$

$f_{M1}(0, x_4, y_4) = 1$ $f_{M2}(0, y_4, x_4) = 1$

$f_{M1}(1, x_4, y_4) = 1$ $f_{M2}(1, y_4, x_4) = 1$

As we can observe in Figure 6, the state transition graph of a decomposed machine may contain parallel edges. If we replace all the parallel directed edges starting from a state *st1* and ending on a state *st2*, we obtain a digraph G(V,E), where V is the set of states and E is the number of edges. This graph may contain self loops. The number of edges in the obtained digraph could be a cost function. The larger number of edges is in digraph, then the larger number of terms is in the next state functions. This means a higher time to answer, an increased number of gates and a larger final implementation of the circuit [11]. The main interest could be the size of the decomposed submachines, the response time, the critical path or even the arrangement of the submachines on a specified technology. During the ***fsmtool's*** execution time**,** the algorithm is mainly dominated by the process of evaluating the functionality and performance of the obtained decomposed solutions. We apply the general decomposition method by using a set of manually or automatically generated partitions with the possibility to obtain valid submachines in order to reduce each submachine complexity while attempting to keep a small number of submachines. The obtained sumbmachines are described in verilog language as output files using the following basic structure for the sequential loop:

```
always @(posedge clock)
  if (reset)
    state = st0;
  else
    state = nextstate;
```

We are using the state and input values for the sensitivity list:

```
always @(state or input)
  case (state)
            state_x: case (input) …
```

Table 1. Output report for SPARTAN-XL

Mcnc files	PI	PO	sub FSM	spartan-XL			
				DFFs	Area(FGs)	Delay(ns)	Freq(Mhz)
Shiftreg-orig.v	1	1	1	3	6	16	85.9
Shiftreg-top1.v	1	1	2	4	12	17	65.6
Shiftreg-top2.v	1	1	2	4	7	16	85.9
Shiftreg-top3.v	1	1	3	3	3	8	83.2
Shiftreg-top4.v	1	1	2	4	13	17	68.6
Dk14-orig.v	3	5	1	3	32	23	63.8
Dk14-top1.v	3	5	3	3	30	23	63.8
Dk14-top2.v	3	5	3	3	31	23	57.1
Dk15-orig.v	3	5	1	2	19	23	64.6
Dk15-top1.v	3	5	2	2	15	20	81
Dk15-top2.v	3	5	2	2	16	20	64.6
Dk27-orig.v	1	2	1	3	6	16	85.9
Dk27-top1.v	1	2	3	3	8	16	85.9
Dk27-top2.v	1	2	3	3	6	16	87.4
Dk512-orig.v	1	3	1	4	16	19	68.6
Dk512-top1.v	1	3	2	4	15	20	78.5
Dk512-top2.v	1	3	3	6	27	21	54.9

Table 2. Output report for VIRTEX-II

mcnc files	PI	PO	sub FSM	VIRTEX-II			
				DFFs	Area(FGs)	Delay(ns)	Freq(Mhz)
Shiftreg-orig.v	1	1	1	3	5	4	512.6
Shiftreg-top1.v	1	1	2	4	9	4	421.3
Shiftreg-top2.v	1	1	2	4	7	4	512.6
Shiftreg-top3.v	1	1	3	3	2	3	551.9
Shiftreg-top4.v	1	1	2	4	11	4	412.8
Dk14-orig.v	3	5	1	3	29	6	308.2
Dk14-top1.v	3	5	3	3	33	6	276.6
Dk14-top2.v	3	5	3	3	36	6	272.9
Dk15-orig.v	3	5	1	2	17	5	335
Dk15-top1.v	3	5	2	2	14	5	332
Dk15-top2.v	3	5	2	2	16	5	327.5
Dk27-orig.v	1	2	1	3	7	4	408.5
Dk27-top1.v	1	2	3	3	9	4	408.5
Dk27-top2.v	1	2	3	3	6	4	493.8
Dk512-orig.v	1	3	1	4	16	5	358.2
Dk512-top1.v	1	3	2	4	13	4	412.8
Dk512-top2.v	1	3	3	6	29	5	258.2

Table 3. Output report for XILINX-XC400XL

mcnc files	PI	PO	sub FSM	XILINX-XC4000XL			
				DFFs	Area(FGs)	Delay(ns)	Freq(Mhz)
Shiftreg-orig.v	1	1	1	3	6	15	80.1
Shiftreg-top1.v	1	1	2	4	12	17	59.4
Shiftreg-top2.v	1	1	2	4	7	15	89.8
Shiftreg-top3.v	1	1	3	3	3	10	100.7
Shiftreg-top4.v	1	1	2	4	13	16	70.5
Dk14-orig.v	3	5	1	3	32	22	58.2
Dk14-top1.v	3	5	3	3	30	22	58.2
Dk14-top2.v	3	5	3	3	31	22	56.8
Dk15-orig.v	3	5	1	2	19	22	61.1
Dk15-top1.v	3	5	2	2	15	19	74
Dk15-top2.v	3	5	2	2	16	19	61.1
Dk27-orig.v	1	2	1	3	6	15	80.1
Dk27-top1.v	1	2	3	3	8	15	80.1
Dk27-top2.v	1	2	3	3	6	14	82.3
Dk512-orig.v	1	3	1	4	16	17	70.5
Dk512-top1.v	1	3	2	4	15	20	70.5
Dk512-top2.v	1	3	3	6	27	20	61.1

Table 4. Output report for ALTERA FLEX-10K

mcnc files	PI	PO	sub FSM	ALTERA FLEX-10K			
				DFFs	Area(FGs)	Delay(ns)	Freq(Mhz)
Shiftreg-orig.v	1	1	1	3	8	7	96.4
Shiftreg-top1.v	1	1	2	4	13	9	81.7
Shiftreg-top2.v	1	1	2	4	7	4	145.2
Shiftreg-top3.v	1	1	3	3	4	3	145.2
Shiftreg-top4.v	1	1	2	4	16	8	88.5
Dk14-orig.v	3	5	1	3	36	12	67.6
Dk14-top1.v	3	5	3	3	37	10	75.9
Dk14-top2.v	3	5	3	3	30	12	67.6
Dk15-orig.v	3	5	1	2	22	9	83.4
Dk15-top1.v	3	5	2	2	18	8	88.5
Dk15-top2.v	3	5	2	2	20	9	83.4
Dk27-orig.v	1	2	1	3	8	6	88.5
Dk27-top1.v	1	2	3	3	9	6	114.2
Dk27-top2.v	1	2	3	3	7	7	96.4
Dk512-orig.v	1	3	1	4	24	11	72.2
Dk512-top1.v	1	3	2	2	18	8	81.7
Dk512-top2.v	1	3	3	2	20	9	60

4. Results and Conclusions

We may obtain various types of decompositions for a prototype FSM chosen as an example FSM and for a specified set of input partitions. After finishing the decomposition process, we tested the functionality of the obtained submachines and checked their equivalence with the prototype machine by using the ***seq_verify*** **VIS** command. This command checks the sequential equivalence of two flattened networks: the prototype and the decomposed network of submachines [14]. At the end of the optimization process we used the ***kiss2vl*** tool to convert the resulting submachines from the symbolic STG representation into Verilog code.

The Verilog files were read into the Leonardo Spectrum synthesis package which performs timing analysis and area approximation. We have used the MCNC examples for ***fsmtool*** benchmark. The results of the benchmarks are shown in the tables from 1 to 4. By using the extended internal data representation of the decomposed submachines with the GMP library [12], we could virtually approach prototype FSMs with a very large number of states while maintaining a reasonable execution time for the decomposing algorithms. The dynamic allocation of internal data representation (big

numbers) is limited only by the total amount of physical memory (RAM).

All circuits have been decomposed using the general decomposition method described in this paper. Each table contains an implementation of the examples set for a specific technological library. The first 4 columns in each table give the name of the circuit, the number of primary inputs and primary outputs, and also the number of internal FSMs for each circuit. The last four columns under each technological library name give the number of "D" flip-flops, the area and the delay of each top-level implementation of the decomposed circuit, and the estimated frequency for the chosen technology. During the decomposition process, the set of partitioned states can be specified manually or can be automatically generated by using ***fsmtool***. As we can see, there are situations when the area and the propagation time are smaller than in the prototype machine, but also there are situations when they surmount the initial properties of the machine. During the synthesis step, we obtain better results on all the synthesized libraries for Shiftreg-top2.v and Shiftreg-top3.v because the partitions of states are presenting better implementations for the selected groups of states. This results in a smaller number of internal states and a lower complexity of each submachine. Dk512-top1.v and Dk512-top2.v have a minimum number of flip-flops on ALTERA FLEX-10K library mapping and a maximum number of flip-flops on the other libraries. VIRTEX-II library allows a higher frequency implementation of the FSMs while the SPARTAN-XL results a lower estimated frequency. From the chosen synthesized and mapped examples on different families of circuits we conclude that the general decomposition method can be easily and successfully applied in obtaining better results for the synthesis of large finite state machines, for state encodings and area optimization, critical path reduction, and the estimated frequency optimization of the circuits for a specific ASIC implementation. Since the VLSI circuits of today are generally very large and complex systems, efficient optimization tools are required to serve as powerful development aids.

References

[1] Brayton K. R., Sangiovani-Vincentelli A. (1992) *SIS: A system for sequential circuit synthesis, Electronic Research Laboratory*, Memorandum No. UCB/ERL M92/41, University of California, Berkeley, CA 94720.

[2] Burns M., Perkowski M., Jozwiak L. and Grygiel S., (1998) *An Efficient and Effective Approach to Column-Based Input/Output Encoding in Functional Decomposition*, Proceedings of 3rd International Workshop on Boolean Problems, pp. 19-29, Freiberg University of Mining and Technology, Institute of Computer Science, September 17-18.

[3] Devadas S., Newton A. R. (1989) *Decomposition and factorization of sequential finite state machines*, IEEE Trans. on CAD , Vol 8, No. 11, pp. 1206-1217.

[4] De Micheli G., Brayton R. K., and A. Sangiovanni-Vincentelli (1985) *Optimal state assignment for Finite State Machines*. IEEE Trans. on CAD, pages 269-284.

[5] De Micheli G. (1994) *Synthesis and Optimization of Digital Circuits*, Stanford University, McGraw-Hill.

[6] Hartmanis J. and Stearns R.E. (1966) *Algebraic Structure Theory of Sequential Machines*, Prentice Hall.

[7] Józwiak L. and Chojnacki A. (2003) *Effective and efficient combinational circuit synthesis for the FPGA-based reconfigurable systems*. Special Issue of Journal of Systems Architecture on Reconfigurable Computing, 49(4-6):247-265.

[8] Muntean I. (1997) Finite Automata Synthesis, Editura Tehnica, Bucharest, Romania.

[9] Pranav A., Devadas S. (1991) *Optimum and heuristic algorithms for an approach to finite state machine decomposition*, IEEE Trans. on CAD , Vol. 10, No. 3, pp. 296-310.

[10] Pruteanu C., Galea D., Haba C.G. (2004) *Global Optimization in Complex Circuits Design*, Proceedings on 7th IEEE International Symposium on Signals, Circuits, Systems (ISSCS 2005), vol.2, ISBN 0-7803-9029-6, IEEE Catalog Number: 05EX1038, Iasi, Romania;

[11] Rupesh S. Shelar, Madhav P. Desay, Narayanan H., *Decomposition of Finite State Machines for Area*, Delay Minimization, Indian Institute of Technology, dept. of electrical engineering

[12] Torbjorn G., Swox AB, *The GNU Multiple Precision Arithmetic Library*, Edition 4.1.4, 21 sept 2004

[13] Villa T. and Sangiovanni-Vincentelli (1990) *A. NOVA: state assignment of Finite State Machines for optimal two-level logic implementation*. IEEE Trans. on CAD, pages 905-924

[14] Villa T., Gitanjali S., Shiple T. (1996) *VIS User's Manual, The VIS Group: University of California*, Berkeley, University of Colorado, Boulder, Now at Lattice Semiconductor

[15] Zafar H., Ciesielski M. J. (1991) *Decomposition and Functional Verification of FSMs*, University of Massachusetts, Amherst, MA 01003