7th International Conference on DEVELOPMENT AND APPLICATION SYSTEMS
Suceava, Romania, May 27 – 29, 2004

COMPLEX CIRCUITS DESIGN. FINITE AUTOMATA DECOMPOSITION

Codrin PRUTEANU
"Gheorghe Asachi" Technical University Iasi
Bld. Mangeron no. 53A, RO-6600 Iasi
codrinp2001@yahoo.com

Abstract. *In order to simplify a synthesis process for certain complex structures it is sometimes inevitable to decompose the function in a pre-processing step. By using the general decomposition method of finite state automata on the basis of a specified states partition we may synthesize functions that are unable to be processed with standard tools which will fail otherwise.*
Keywords: *logic synthesis, finite automata, decomposition.*

1. Introduction

Today's technology has reached such a level that the high level synthesis is integrated with logic synthesis. The problem of complex circuits' decomposition date back to the very early days of computer aided design of digital circuits. By studying the properties of the complex functions it is possible to represent them by means of several simpler functions.
By using these functional properties we may decompose large and complex circuits into a system of smaller circuits which may be readily available and easily maintained. In the case of multi level logic synthesis programs, like MVSIS, we may deal with specific problems as generating low performance circuits when facing with complex functions. By decomposing the circuit into several sub circuits the synthesis procedure may be simplified. The decomposition of a sequential machine implies the obtaining of two or more partitions of the original machine, every partition corresponding to a submachine which operates concurrently with the others in order to respect the initial behavior of the circuit. The partitions resulted through the decomposition process from combining the states of the initial machine are becoming states in the newly created machines. The necessary decomposition algorithms are specific for certain types of two way or multi way, parallel, cascade, general or arbitrary decomposition topologies.

2. Preliminaries

A finite state machine M can be described by a five-tuple $M = (S, I, O, \delta, \lambda)$, where S is a set of state symbols, I is a set of primary inputs, O is a set of primary outputs, $\delta : I \times S \rightarrow S$ is the next state function, and $\lambda : I \times S \rightarrow O$ is the output function (Mealy machine). A FSM can be represented by its State Transition Graph (**STG**) or equivalently, by its State Transition Table (**STT**).
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 **sub machines**. The machine obtained as a result of the decomposition is called the **decomposed machine** and is implemented as a network of interconnected sub machines. The general structure of such a network is shown in Fig.1. 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 $\Pi_1 . \Pi_2 \ldots \Pi_n = \Pi(0)$
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 $\Pi(0)$.

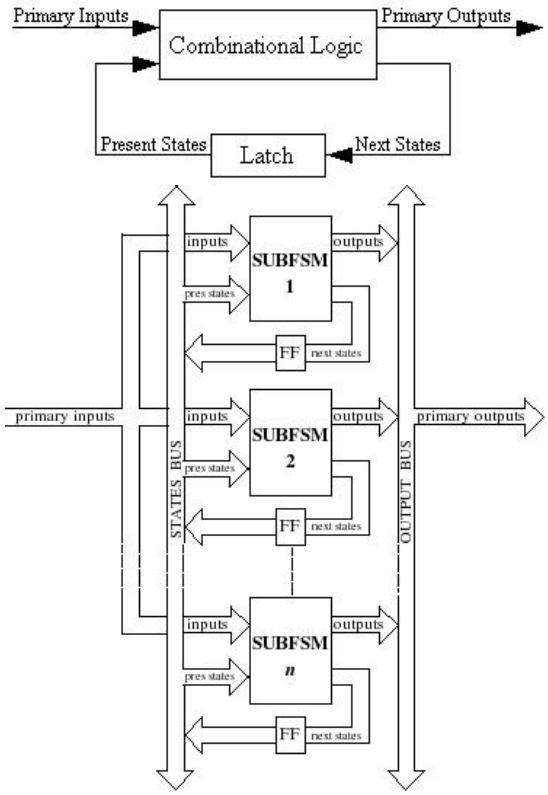

Figure 1. General sequential circuit and general decomposition topology

3. Previous work

The decomposition of sequential machines was first treated in a formal way by Hartmanis and Stearns [5]. They proposed two types of decomposition, parallel and cascade, based on the topology of the decomposed machine. One of the simplest ways in which a machine can be broken up into sub machines is the **parallel decomposition**. The structure of the sub machines is that they are supplied with the same input sequence but operate independently. There is no interaction or exchange of information between the sub machines. This method has limited use in the design of modern finite state machines because practical designs do not usually have good parallel decompositions. Another type of decomposition is the **cascade** or **serial decomposition**, where again each submachine is driven by the same input sequence, but the two machines do not operate independently. A submachine is supplied by means of auxiliary inputs with information about the current internal states of the other sub machines. The possibility of passing state information from a submachine to the next submachine makes cascade decomposition more powerful than parallel decomposition. The transmission of state information is serial and a submachine requires state information of its own state and about the states of its predecessors. It passes its own state information to its successor machines only. Another form of decomposition was presented by Devadas and Newton [14]. Here both components of the decomposed machine interact with each other. This form of decomposition involves identifying **subroutines** of **factors** in the original machine, extracting these factors and representing them as a separate factoring machine. The occurrences of these factors become calls to the factoring machine from the factored machine. The method does not have a definite cost function to optimize and does not guarantee anything about the quality of the decomposition. Finally, the **general decomposition,** or arbitrary decomposition, can have various topologies and covers any type of decomposition. The basic output constraint remains that any pair of states of the prototype machine should have distinct codes in the decomposed machine which remains unchanged. The remaining constraints are dependent on the other sub machines that a particular submachine receives present-state information from. The type of constraints imposed are briefly illustrated by means of the example topology in Fig.1.

4. Implementation

The main objectives in the present paper are to apply general decomposition method to a sequential machine in order to reduce each submachine complexity while attempting to keep a small total number of sub machines. The obtained sub machines during the decomposition procedure are then verified if they respect the

general decomposition properties. The first step is the state partitioning of the prototype machine using a specified mask of states so that it may satisfy the decomposition requirements. The partitioning masks applied to the initial set of states can be manually specified or automatically generated by using random sets of states, grouped in blocks belonging to each partition. If the decomposition mask is manually specified by using predefined partition sets, the standard methods of general decomposition can be applied in order to obtain the results. If a random number of masks for decomposition are automatically generated, there are performed a set of successive steps in which an initial partition is generated first; this will be the first submachine. Then in a series of successive steps are generated the remaining set of partitions that will be the other concurrent sub machines which must respect the decomposition rules. The partitions are generated while their product is not equal to zero partition *Π(0)*, which means there are still blocks of states which are not uniquely identified, or which intersection of states is not equal to ∅. In parallel, the validation procedure is testing the partitions that are not simplifying the set of current states and deletes them by generating a new partition instead.

```
0  st0 st0  0
1  st0 st4  0
0  st1 st0  1
1  st1 st4  1
0  st2 st1  0
1  st2 st5  0
0  st3 st1  1
1  st3 st5  1
0  st4 st2  0
1  st4 st6  0
0  st5 st2  1
1  st5 st6  1
0  st6 st3  0
1  st6 st7  0
0  st7 st3  1
1  st7 st7  1
```

Fig. 2. STT for Prototype Machine

This partition will be tested then if it reduces the current set of states. The cycle is repeated until there have been uniquely identified all blocks of states that will be the new set of states of the resulting sub machines.

For example we will use a FSM which has the behavior of a shift register. The internal description is represented by its state transition table, in standard kiss format in Fig. 2.

Suppose we have a set of partitions manually specified as follows:

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

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

We obtain two sub machines that satisfy the general decomposition rules of the prototype machine as we can see in Fig. 3

```
000  (st0 st0)  0    000  (st0 st0)  0
100  (st0 st1)  0    100  (st0 st0)  0
001  (st1 st0)  1    001  (st1 st0)  1
101  (st1 st1)  1    101  (st1 st0)  1
001  (st0 st1)  0    000  (st1 st1)  0
101  (st0 st2)  0    100  (st1 st1)  0
010  (st0 st1)  1    000  (st2 st1)  1
110  (st0 st2)  1    100  (st2 st1)  1
000  (st1 st0)  0    001  (st0 st1)  0
100  (st1 st1)  0    101  (st0 st2)  0
001  (st2 st0)  1    010  (st1 st1)  1
101  (st2 st1)  1    110  (st1 st2)  1
010  (st1 st0)  0    001  (st2 st2)  0
110  (st1 st2)  0    101  (st2 st0)  0
000  (st2 st0)  1    010  (st0 st2)  1
100  (st2 st2)  1    110  (st0 st0)  1
```

Fig. 3. STT for first and second submachine

5. Results

For a given machine taken as example and for a specified set of partitions, we may obtain various types of decompositions. The resulting equivalent sub machines have been synthesized on standard tools like Leonardo Spectrum where have been obtained different results that are important from the point of view of technological implementation on various family of circuits and for certain requirements of optimum area and delay. The obtained results are from standard sets of examples implemented on different technological libraries. 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. In the case of running the automatic generation of partitions, the obtained partition number is dynamically reduced to the most efficient possible number. During the decomposition flow we needed some tools like: **genfsm** used to automatically generate FSM's, **vl2mv** used to convert the examples from verilog level to kiss format, **fsmtool** used to

generate the partitioning sets of states and for decomposition, and **kiss2vl**.

6. Conclusions and future work

In this paper we built and tested a number of tools like **genfsm**, **fsmtool**, and **kill2vl** in order to allow us to implement the theoretical part and obtain some practical results. We have shown the potential and efficiency of our tools to generate a finite state machine, to apply a general decomposition method by a set of manually or automatically generated partitions and the possibility to obtain valid sub machines in order to reduce each submachine complexity while attempting to keep a small number of sub machines.

This pre-synthesis step can be easily integrated into a standard synthesis flow and by it's general approach it can be easily adapted to various specific optimization problems like complex function decomposition or optimum implementation on specific target libraries with a variable number of logic gates in their LUT's with different sizes and area of implementations.

This standard flow can be improved in the future by finding an alternative way to test the optimal results even from the generation step. The necessity of using an external tool or standard program for the evaluation of the results can be replaced by a fitness function in a process of improving the quality of the results. This improvement will reduce the execution time and the complexity of the optimization flow.

References

[1] BRZOZOWSKI J.A., LUBA T., *Decomposition of Boolean Functions Specified by Cubes*, University of Waterloo, Canada, Warsaw University of Technology, Poland.

[2] Carlos Llanos Quintero, Marius Strum, *GERPAR and GERTAB2. Two FSM Decomposition Optimization Tools*, Laboratorio de Microelectronica LME-EPUSP, Brasil.

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

[4] Haba C.G. , *Contributions to synthesis of digital circuits*, PhD. Thesis, "Gh. Asachi " Technical University, Iasi, Romania.

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

[6] Hasan Zafar, Maciej J. Ciesielski , *Decomposition and Functional Verification of FSMs*, University of Massachusetts, Amherst.

[7] Herbert Taub, (1982) *Digital Circuits and Microprocessors*, International student edition.

[8] L. Józwiak, A. Chojnacki, (1998) *Application of Information Relationship Measures to Logic Synthesis*, Proc. CSSP98 - 9th Annual Workshop on Circuits, Netherlands

[9] Michael Burns, Marek Perkowski, Lech Jozwiak, and Stanislaw Grygiel, (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 Inst. of Comp. Science, Sept. 17-18.

[10] Muntean I., (1997) *Finite Automata Synthesis*, Editura Tehnica, Bucuresti.

[11] MVSIS, (2001) *Multi-level Logic Synthesis*, University of California, Berkeley, CA.

[12] Partha S. Roop, Sowmya A. , *Functional Decomposition of Composite Finite State Machines*, Univ.of New South Wales, Australia.

[13] Perkowski Marek, *Digital design automation: finite state machine design*, dept. of electrical engineering, Portland State University.

[14] Pranav Ashar, Srinivas Devdas, (1991) *Optimum and heuris-tic algorithms for an approach to finite state machine decomposition*, IEEE Trans. on CAD , Vol. 10, No. 3, March 1991, pp. 296-310.

[15] Roche Emmanuel, (1995) *Factorization of Finite-State Tranducers*, Cambridge.

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

[17] SIS, (1992) *A system for sequential circuit synthesis*, Univ. of California, Berkeley, CA.

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

[19] Stefan Gheorghe, (2000) *Digital systems and Circuits*, Editura Tehnica, Bucuresti.
[20] Zamfirescu A. , *HDL Chip Design*.

Table 1: Experimental results from a complete decomposition and synthesis flow

FILE	TECHNOLOGY	DFF	PI	PO	AREA	DELAY	PORTS	NETS	GATES	INST	ARRIVAL
Example-orig.v	SPARTAN-XL	3	3	1	6	16	4	18	6	15	13.64
Example-top1.v	SPARTAN-XL	3	3	1	5	16	4	17	5	14	13.64
Example-top2.v	SPARTAN-XL	3	3	1	7	16	4	17	7	14	13.64
Example-top3.v	SPARTAN-XL	3	3	1	5	16	4	17	5	14	13.64
Shiftreg-orig.v	SPARTAN-XL	3	3	1	6	16	4	18	6	15	13.64
Shiftreg-top1.v	SPARTAN-XL	4	3	1	12	17	4	25	12	22	13.84
Shiftreg-top2.v	SPARTAN-XL	4	3	1	7	16	4	18	7	15	13.84
Shiftreg-top3.v	SPARTAN-XL	3	3	1	3	8	4	12	3	9	6.90
Example-orig.v	VIRTEX-II	3	3	1	6	7	4	16	6	13	6.73
Example-top1.v	VIRTEX-II	3	3	1	8	7	4	18	8	15	6.73
Example-top2.v	VIRTEX-II	3	3	1	7	7	4	17	7	14	6.79
Example-top3.v	VIRTEX-II	3	3	1	6	7	4	16	6	13	6.79
Shiftreg-orig.v	VIRTEX-II	3	3	1	5	6	4	15	5	12	6.08
Shiftreg-top1.v	VIRTEX-II	4	3	1	9	6	4	25	9	22	6.08
Shiftreg-top2.v	VIRTEX-II	4	3	1	9	6	4	20	9	17	6.03
Shiftreg-top3.v	VIRTEX-II	3	3	1	2	5	4	12	2	9	5.49
Example-orig.v	ALTERA-FLEX10k	3	3	1	9	8	4	18	9	15	8.50
Example-top1.v	ALTERA-FLEX10k	3	3	1	8	10	4	17	8	14	9.60
Example-top2.v	ALTERA-FLEX10k	3	3	1	8	8	4	16	8	13	8.50
Example-top3.v	ALTERA-FLEX10k	3	3	1	8	10	4	16	8	13	9.60
Shiftreg-orig.v	ALTERA-FLEX10k	3	3	1	8	8	4	17	8	14	8.50
Shiftreg-top1.v	ALTERA-FLEX10k	4	3	1	13	11	4	24	13	21	10.70
Shiftreg-top2.v	ALTERA-FLEX10k	4	3	1	7	4	4	15	7	12	4.40
Shiftreg-top3.v	ALTERA-FLEX10k	3	3	1	3	4	4	10	3	7	4.40
Example-orig.v	ASIC-SCL05u	-	-	-	88	-	4	19	88	14	2.80
Example-top1.v	ASIC-SCL05u	-	-	-	82	-	4	17	82	13	2.31
Example-top2.v	ASIC-SCL05u	-	-	-	85	-	4	18	85	13	2.50
Example-top3.v	ASIC-SCL05u	-	-	-	89	-	4	21	89	15	2.43
Shiftreg-orig.v	ASIC-SCL05u	-	-	-	125	-	4	27	125	21	2.34
Shiftreg-top1.v	ASIC-SCL05u	-	-	-	179	-	4	35	179	28	4.29
Shiftreg-top2.v	ASIC-SCL05u	-	-	-	38	-	4	10	38	7	0.80
Shiftreg-top3.v	ASIC-SCL05u	-	-	-	45	-	4	12	47	9	1.45

[21] Wolfgang Thomas, (2003) *Applied Automata Theory*.

Appendix A

We show in the table 1 some examples using the general decomposition methods. They are obtained from a full synthesis flow, starting with a finite state machine automatically generated by GenFSM or specified from an examples set. The input FSMs are described in a standard kiss format, they are parsed and represented in memory and decomposed by the randomly generated or manually specified states masks in FSMtool. The results are represented either in kiss format or verilog output format for integration with standard synthesis tools. The output results depend on the quality of decomposition, the arrangement of internal states in the newly created sub machines and the target technology used for implementation. As we can see in each column, there are specific results for each technological implementation and the best solutions of the decomposition can be easily selected for each target library. These are standard examples and they are used to obtain experimental results for certain types of partitions applied to the prototype machine. The resulting sub machines obtained from the decomposition

with FSMtool are then synthesized with Leonardo Spectrum and the detailed resulting values are pointed in this table.