FSM IMPLEMENTATION IN EMBEDDED MEMORY BLOCKS OF PROGRAMMABLE LOGIC DEVICES USING FUNCTIONAL DECOMPOSITION

Henry Selvaraj*, Mariusz Rawski, Tadeusz Łuba

*****University of Nevada, Las Vegas*
4505 Maryland Parkway, Las Vegas, NV 89154-4026, USA
email: selvaraj@unlv.edu

Warsaw University of Technology, Institute of Telecommunications
Nowowiejska 15/19, 00-665 Warsaw, Poland

Abstract: Since modern programmable devices contain embedded memory blocks, there exists a possibility to implement Finite State Machines (FSM) using such blocks. The size of the memory available in programmable devices is limited, though. The paper presents a general method for the synthesis of sequential circuits using embedded memory blocks. The method is based on the serial decomposition concept and relies on decomposing the memory block into two blocks: a combinational address modifier and a smaller memory block. An appropriately chosen decomposition strategy may allow reducing the required memory size at the cost of additional logic cells for address modifier implementation. This makes possible implementation of FSMs that exceed available memory by using embedded memory blocks and additional programmable logic.

Keywords: digital circuits, logic minimization, implementation, sequential machines, programmable read only memory, Boolean functions

1. INTRODUCTION

Decomposition has become an important tool in the analysis and design of digital systems. It is fundamental to many fields in modern engineering and science (Brzozowski and Luba 1997; Hartmanis and Stearns 1966; Luba 1994; Zupan and Bohenec 1966a, b; Ross, *et al.*, 1991). Functional decomposition relies on breaking down a complex system into a network of smaller and relatively independent co-operating sub-systems, in such a way that the original system's behavior is preserved. A system is decomposed into a set of smaller subsystems, such that each of them is easier to analyze, understand and synthesize.

By taking advantage of the opportunities the modern microelectronic technology provides us with, we are in a position to build very complex digital circuits and systems at relatively low cost. There is a large variety of logic building blocks that can be exploited. The library of elements contains various types of gates; a lot of complex gates can be generated in (semi-)custom CMOS design; and the field programmable logic families include different types of (C)PLDs and FPGAs. However, the opportunities created by modern microelectronic technology are not fully exploited because of weaknesses in traditional logic design methods.

Recently, new methods of logic synthesis based on functional decomposition have been developed (Luba, *et al.*, 1995; Chang, *et al.*, 1996; Burns, *et al.*, 1998; Jozwiak and Chojnacki 1999; Qiao, *et al.*, 2000). Unfortunately decomposition-based methods are considered as methods suitable mainly for implementation of combinational functions.

Modern FPGA architectures contain embedded memory blocks. In many cases, designers do not need to use these resources. However, such memory blocks allow implementation of sequential machines in a way that requires less logic cells than the traditional flip-flop based implementation. This may be used to implement "non-vital" sequential parts of the design, saving logic cell resources for more important sections. However such an implementation may require more memory than available in a device. To reduce memory usage in ROM-based sequential machine implementations, decomposition-based methods can be successfully used (Luba, *et al.*, 1992).

In this paper, basic information is introduced first. Secondly, application of decomposition in the implementation of sequential machines is presented. Subsequently, some experimental results, obtained with a prototype tool that implements functional decomposition, are discussed.

The experimental results demonstrate that decomposition is capable of constructing solutions (utilizing embedded memory blocks) of comparable or even better quality than the methods implemented in commercial systems.

2. BASIC NOTIONS

2.1 Functional decomposition

Let A and B be two subsets of X such that $A \cup B = X$. Assume that the variables $x_1,...,x_n$ have been relabeled in such way that:

$A = \{x_1,...,x_r\}$ and

$B = \{x_{n-s+1},...,x_n\}$.

Consequently, for an n-tuple x, the first r components are denoted by x^A and the last s components by x^B.

Let F be a Boolean function, with $n > 0$ inputs and $m > 0$ outputs. Let (A, B) be as defined above. Assume that F is specified by a set **F** of the function's cubes. Let G be a function with s inputs and p outputs; let H be a function with $r + p$ inputs and m outputs. The pair (G, H) represents a serial decomposition of F with respect to (A, B), if for every minterm b relevant to F, $G(b^B)$ is defined, $G(b^B) \in \{0, 1\}^p$, and $F(b) = H(b^A, G(b^B))$. G and H are called blocks of the decomposition.

Partition-based representation of Boolean functions can be used to describe functional decomposition algorithms (Brzozowski and Luba 1997; Luba and Selvaraj 1995; Luba 1994; Luba, *et al.*, 1996; Rawski, *et al.*, 1997).

If there exists a r-partition Π_G on **F** such that $P(B) \leq \Pi_G$, and $P(A) \bullet \Pi_G \leq P_F$, then F has a serial decomposition with respect to (A, B).

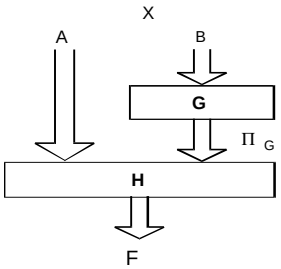

Fig. 1. Schematic representation of the serial decomposition

The serial decomposition process consists of the following steps: an input support selection (the most time-consuming part of the process), calculation of partitions $P(A)$, $P(B)$ and P_F, construction of partition Π_G, and creation of functions H and G (Luba and Selvaraj 1995).

2.2. Finite state machine

Let $A = \langle V, S, \delta \rangle$ be an FSM (completely or incompletely specified) with no outputs (outputs are omitted as they have insignificant impact on the method), where:

V – set of input symbols,
S – set of internal states,
δ – state transition function,

and $m = \lceil \log_2 |V| \rceil$, $n = \lceil \log_2 |S| \rceil$ denote the number of input and state variables respectively.

To describe logic dependencies in such an FSM special partition description (Brzozowski and Luba 1997) and special partition algebra (Hartmanis and Stearns 1966) are employed.

Let ***K*** be a one-to-one correspondence between the domain D_δ of transition function and $K = \{1, \ldots, p\}$, where $p = |D_\delta|$. The characteristic partition P_c of an FSM is defined in the following way:

$(k_1, k_2) \in B_{Pc}$ iff $\delta(K^{-1}(k_1)) = \delta(K^{-1}(k_2))$

Thus, each block B_{Pc} of the characteristic partition includes these elements from K which correspond to pairs (v, s) from the domain D_δ such that the transition function $\delta(v, s) = s'$ maps them onto the same next state s'.

A partition P on K is compatible with partition π on S iff for any inputs v_a, v_b the condition that s_i, s_j belong to one block of partition π implies that the elements from K corresponding to pairs (v_a, s_i) and (v_b, s_j) belong to one block of the partition P.

A partition P on K is compatible with partition θ on V iff for any state s_a, s_b the condition that v_i, v_j belong to one block of partition θ implies that the elements from K corresponding to pairs (v_i, s_a) and (v_j, s_b) belong to one block of the partition P.

In particular, a partition P on K is compatible with the set $\{\pi, \theta\}$ if it is compatible with both π and θ, while it is compatible with set $\{\pi_1, \ldots, \pi_\alpha\}$ of partitions on S (or set $\{\theta_1, \ldots, \theta_\alpha\}$ of partitions on V) iff it is compatible with $\pi = \pi_1 \bullet \pi_2 \bullet \pi_3 \bullet \ldots \bullet \pi_\alpha$ ($\theta = \theta_1 \bullet \theta_2 \bullet \theta_3 \bullet \ldots \bullet \theta_\alpha$).

3. ROM IMPLEMENTATION OF FINITE STATE MACHINES

FSM can be implemented using ROM (*Read Only Memory*) (Luba, *et al*., 1992). Figure 2 shows the general architecture of such an implementation. State and input variables ($q_1, q_2, \ldots, q_n$ and $x_1, x_2, \ldots, x_m$) constitute ROM address variables ($a_1, a_2, \ldots, a_{m+n}$). The ROM would consist of words, each storing the encoded present state (control field) and output values (information field). The next state would be determined by the input values and the present-state information fedback from memory.

This kind of implementation requires much fewer logic cells than the traditional flip-flop implementation (or does not require them at all, if memory can be controlled by clock signal – no register required); therefore, it can be used to implement "non-vital" FSMs of the design, saving LC resources for more important sections of the design. However, a large FSM may require too much of buried memory resources.

The size of the memory needed for such an implementation depends on the lengths of the address and memory word.

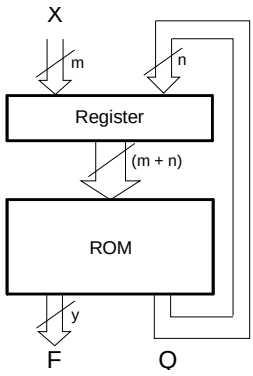

Fig. 2. Implementation of FSM using memory blocks

Let m be the number of inputs, n be the number of state encoding bits and y be the number of output functions of FSM. The size of memory needed for implementation of such an FSM can be expressed by the following formula:

$$M = 2^{(m+n)} \times (n + y),$$

where $m + n$ is the size of the address, and $n + y$ is the size of the memory word.

Since modern programmable devices contain embedded memory blocks, there exists a possibility to implement FSM using these blocks. The size of the memory blocks available in programmable devices is limited, though. For example, Altera's FLEX family EAB (*Embedded Array Block*) has 2048 bits of memory and the device FLEX10K10 consists of 3 such EAB's. Functional decomposition can be used to implement FSMs that exceeded that size.

3.1. Address modifier

Any FSM, say *A*, defined by a given transition table can be implemented as in Fig. 3 using an address modifier.

If $\pi_1, ..., \pi_n$ are partitions on S, $\theta_1, ..., \theta_m$ are partitions on V, and P_k is partition on K compatible with either π_i or θ_j then $P = \{P_1, ..., P_{m+n}\}$ is the set of all partitions compatible with $\{\pi_1, ..., \pi_n, \theta_1, ..., \theta_m\}$. Partitions $\pi_1,..., \pi_n$ correspond to state variables and $\theta_1, ..., \theta_m$ correspond to input variables. To achieve unambiguous encoding of address variables, and at the same time maintaining the consistency relation $\boldsymbol{K}$ with the transition function δ, partitions $P_1, .., P_w$ have to be found, such that:

$$P_1 \bullet P_2 \bullet P_3 \bullet ... \bullet P_w \le P_c.$$

This is the necessary and sufficient condition for $\{P_1, .., P_w\}$ to determine the address variables. This is because each memory cell is associated with a single block of P_c, i.e. with those elements from K which map the corresponding (v, s) pairs onto the same next state.

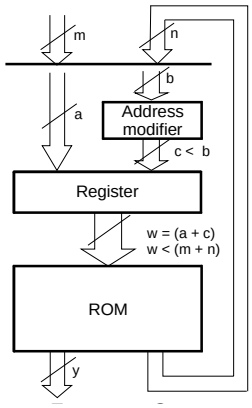

Fig. 3. Implementation of FSM using an address modifier

The selection of w ($w < n + m$) partitions from the set $\{P_1, ..., P_{m+n}\}$ is made such that they produce the simplest addressing unit. Such a selection is possible thanks to the notion of r-admissibility (Luba 1994).

A set $\{P_1, ..., P_k\}$ is r-admissible in relation to partition P iff there is a set $\{P_{k+1}, ..., P_r\}$ of two block partitions such that the following condition holds:

$$P_1 \bullet P_2 \bullet P_3 \bullet ... \bullet P_k \bullet P_{k+1} \bullet ... \bullet P_r \le P_c$$

and no set of $r - k - 1$ partitions exist which meets this requirement.

For partition $\rho \le \sigma$ let $\sigma|\rho$ denote the quotient partition and $\varepsilon(\sigma|\rho)$ the number of elements in the largest block of $\sigma|\rho$. Let $e(\sigma|\rho)$ be the smallest integer equal to or larger than $\log_2\varepsilon(\sigma|\rho)$ (i.e. $e(\sigma|\rho) = \lceil \log_2\varepsilon(\sigma|\rho) \rceil$). Then the r-admissibility of $\{P_1, ..., P_k\}$ is $r = k + e(\pi|\pi_f)$,
where π is the product of $P_1, ..., P_k$ and π_f is the product of π and P.

If $\boldsymbol{P} = \{P_1, ..., P_k\}$ is r-admissible in relation to P then each subset of $\boldsymbol{P}$ is r'-admissible, where $r' \le r$.

The smallest partition i.e. one where each element is a separate block, will be denoted as $P(0)$, $\pi(0)$, $\theta(0)$, etc.

3.2. Input/state encoding

The source of the complexity of the address modifier is in the address variables which depend on more than one input/state variables. Therefore it is important to choose such an encoding of input and internal state symbols that we could obtain maximal set of partitions P (compatible with π or θ) whose r-admissibility in relation to P_c is w, where w is the number of address bits of the given ROM block.

Appropriate encoding will be determined by generating partitions with the knowledge that r-admissibility of a partition P compatible with partition $\pi = (B_1; ...; B_i; ...; B_\alpha)$ or compatible with partition $\theta = (B_1; ...; B_i; ...; B_\alpha)$ is:

$$r = \lceil \log_2\alpha \rceil + \lceil \log_2 \max |\delta(B_i)| \rceil,$$

where B_i is a block of partition π or θ, δ is the transition function, $\max |\delta(B_i)|$ denotes the number of elements in the most populous set $\delta(B_i)$, $i \in \{1... \alpha\}$. Because of the one-to-one correspondence between partitions P and π or θ, r-admissibility of π, θ or $\{\pi, \theta\}$ in relation to P_c can be considered.

For a given w, the necessary encoding that allows the implementation of the FSM with the use of address modifier can be found in the following way:

1. find r_1 = r-admissibility of $\theta(0)$; find r_2 = r-admissibility of $\pi(0)$,
2. if $r_1 = w$ (or $r_2 = w$) then $a_1 = x_1$, ..., $a_x = x_x$ (or $a_1 = q_1$, ..., $a_x = q_q$) and further encoding partition are searched among $\pi(0)$ (or $\theta(0)$),
3. if both $r_1 > w$ and $r_2 > w$ then for subsequent steps θ if $|V| < |S|$ or π if $|V| > |S|$ is taken,
4. assume that θ was chosen in the previous step; for $i = 1, 2, ...$ and $\alpha = 2^{m-i}$ find $\theta = (B_1; ...; B_\alpha)$ so that $|B_1| + |B_2| +...+ | B_\alpha| = |V|$ and whose r-admissibility equals w.

In a similar way $\pi = (D_1; ...; B_\beta)$, $\beta = 2^{n-j}$, $j = 1, 2, ...$ are found. The set $\{\pi, \theta\}$ must have r-admissibility of w. Partitions π and θ can be represented as follows:

$$\pi = \pi_1 \bullet \pi_2 \bullet ... \bullet \pi_k,$$
$$\theta = \theta_1 \bullet \theta_2 \bullet ... \bullet \theta_l,$$

where

$$k = \lceil \log_2\beta \rceil,$$
$$l = \lceil \log_2\alpha \rceil.$$

The encoding of the remaining input and state variables can be obtained from the following rules:

$$\pi_1 \bullet \pi_2 \bullet ... \bullet \pi_k \bullet \pi' = \pi(0),$$
$$\theta_1 \bullet \theta_2 \bullet ... \bullet \theta_l \bullet \theta' = \theta(0),$$

where π' and θ' represent partitions induced by those variables.

After all the variables are encoded the process may be considered as a decomposition of the memory block into two blocks: a combinational address modifier and a smaller memory block. Decomposition is computed for partitions:

$P(A) = \pi \bullet \theta = \pi_1 \bullet \pi_2 \bullet ... \bullet \pi_k \bullet \theta_1 \bullet \theta_2 \bullet ... \bullet \theta_l$,

$P(B) = \pi' \bullet \theta'$.

Appropriately chosen strategy of decomposition may allow reducing required memory size at the cost of additional logic cells for address modifier implementation. This makes possible implementation of large FSMs that need more than available memory by making use of the embedded memory blocks and additional programmable logic.

4. EXPERIMENTAL RESULTS

The proposed method was applied to implement several examples from standard benchmark set in FLEX10K10 devices using ALTERA MAX+PlusII system. In Table 1 a comparison of different FSM implementation techniques are presented. In the column named *ROM Implementation,* the number of bits required to implement a given FSM using ROM is presented. FLEX10K10 device is equipped only with 3 EAB memory blocks each consisting of 2048 bits. Most of the presented FSM examples cannot be implemented in this device, because their implementations require much more memory resources than available. In the column called *FF Implementation*, the number of logic cells required to implement the given FSM in the "traditional" way using flip-flops is given. To describe the FSM for this kind of implementation, a special AHDL (*Altera Hardware Description Language*) construction was used. In the column under *AM implementation,* the results of implementation of the given FSM using the concept of address modifier are presented. In this approach, the address modifier was implemented using logic cell resources and ROM was implemented in EAB blocks (Fig. 3). The number of logic cells and the number of memory bits are given in the table as results. It can be easily noticed that the application of decomposition improves the quality of ROM as well as flip-flop implementation.

Table 1. Comparison of FSM implementation results of standard benchmarks in FPGA architecture (EPF10K10LC84-3 device). [1)] Implementation not possible – not enough memory resources, [2)] implementation not possible – not enough CLB resources

Benchmark	ROM Implementation #bits	FF implementation #LCs	AM implementation	
			#LCs	#bits
bbtas	160	10	7	80
beecount	448	32	14	112
d14	512	60	21	256
mc	224	14	2	56
lion9	320	24	1	80
train11	320	25	15	8
bbsse	22528 [1)]	52	3	5632
cse	22528 [1)]	92	2	5632
ex4	13312 [1)]	28	2	3328
mark1	10240 [1)]	40	2	5120
s1	24576 [1)]	137	96	5632
sse	22528 [1)]	52	3	5632
tbk	16384 [1)]	759 [2)]	333	4093
s389	22528 [1)]	64	9	5632
∑	156608	1389	510	41293
%	100 %	100 %	36.7%	26.4%

Table 2. Comparison of FSM implementation results in FPGA architecture (EPF10K10LC84-3 device). [1] FSM described with special AHDL construction; [2] decomposition not possible; [3] not enough memory bits to implement the project

Example	FF_MAX+PlusII		ROM		AM_ROM	
	LCs / Bits	Speed [MHz]	LCs / Bits	Speed [MHz]	LCs / Bits	Speed [MHz]
DESaut	46/0	41,1	8/1792	47,8	7/896	47,1
5B6B	93/0	48,7	6/448	48,0	– [2]	– [2]
count4	72/0 18/0 [1]	44,2 86,2 [1]	16/16384	– [3]	12/1024	39,5

The application of address modifier concept allows implementing FSM in such a way that only about 37 % of logic cell resources required in flip-flop implementation and about 27% of memory resources required in ROM implementation is used. Application of address modifier concept allows implementing all the presented FSMs using available memory and additional parts (address modifier) implemented in CLBs.

In Table 2 results of implementation of several "real life" FSMs are presented. Following examples were used in the experiments:

- DESaut – the state machine used in DES algorithm implementation,
- 5B6B – the 5B-6B coder,
- count4 – 4 bit counter with COUNT UP, COUNT DOWN, HOLD, CLEAR and LOAD.

Each sequential machine was described by a transition table. The results for each method of implementation are presented using the number of logic cells and memory bits required (i.e. area of the circuit) and the maximal frequency of clock signal (i.e. speed of the circuit). The columns under the *FF_MAX+PlusII* heading present results obtained by the Altera MAX+PlusII system for the classical flip-flop implementation of FSM. The *ROM* columns provide the results of ROM implementation; the columns under *AM_ROM* present the results of ROM implementation with the use of address modifier. Especially interesting is the implementation of the 4-bit counter. Its description with a transition table leads to a strongly non-optimal implementation. On the other hand, its description using a special AHDL construct produces very good results. The ROM implementation of this example requires too many memory bits (the size of required memory block exceeds the available memory), thus it cannot be implemented in the given structure. Application of the address modifier concept allows reducing the necessary size of memory, and that makes the implementation possible. The performance of the FSMs implemented with the use of address modifier concept is not significantly degraded.

5. CONCLUSIONS

Balanced decomposition produces very good results for combinational functions implemented using FPGA-based architectures. However, results presented in this paper show that functional decomposition can be efficiently and effectively applied beyond the implementation of combinational circuits. Decomposition can be applied to implement large FSM in an alternate way – using ROM. This kind of implementation requires much fewer logic cells than the traditional flip-flop implementation; therefore, it can be used to implement "non-vital" FSMs of the design, saving LC resources for more important sections of the design. However, large FSMs may require too much buried memory resources. With the concept of address modifier, memory usage can be significantly reduced. The experimental results shown in this paper demonstrate that the synthesis method based on functional decomposition can help in implementing sequential machines using ROM memory.

REFERENCES

Burns M., Perkowski M., Jóźwiak L. (1998). An Efficient Approach to Decomposition of Multi-Output Boolean Functions with Large Set of Bound Variables. *Proc. of the Euromicro Conference*, Vasteras.

Brzozowski I., Kos A. (1999). Minimisation of Power Consumption in Digital Integrated Circuits by Reduction of Switching Activity. *Proc. of the Euromicro Conference*, pp.376-380. **Vol. 1**, Milan.

Brzozowski J. A., Luba T. (1997). Decomposition of Boolean Functions Specified by Cubes. *Research Report CS-97-01*, University of Waterloo, Waterloo; REVISED October 1998.

Chang S.C., Marek-Sadowska M., Hwang T.T. (1996). Technology Mapping for TLU FPGAs Based on Decomposition of Binary Decision Diagrams. *IEEE Trans. on CAD*, **Vol. 15**, No. 10, pp. 1226-1236.

De Micheli G. (1994): *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, New York.

Hartmanis J., Stearns R.E. (1966). *Algebraic Structure Theory of Sequential Machines*. Prentice-Hall.

Jozwiak L., Chojnacki A. (1999). Functional Decomposition Based on Information Relationship Measures Extremely Effective and Efficient for Symmetric Functions. *Proc of the Euromicro Conference*, pp.150-159. **Vol. 1**, Milan.

Kravets V. N., Sakallah K. A. (2000). Constructive library-aware synthesis using symmetries. *Proc. Of Design, Automation and Test in Europe Conference*.

Luba T. (1994). Multi-level logic synthesis based on decomposition. *Microprocessors and Microsystems*, **18**, No. 8, pp. 429-437.

Luba T., Gorski K., Wronski L.B. (1992). ROM-based Finite State Machines with PLA address modifiers. *Proc. of EURO-DAC*, Hamburg.

Luba T., Selvaraj H., Nowicka M., Kraśniewski, A. (1995). Balanced multilevel decomposition and its applications in FPGA-based synthesis. In: *Logic and Architecture Synthesis* (G.Saucier, A.Mignotte ed.), Chapman&Hall.

Luba T., Selvaraj H. (1995). A General Approach to Boolean Function Decomposition and its Applications in FPGA-based Synthesis. *VLSI Design, Special Issue on Decompositions in VLSI Design*, **vol. 3**, Nos. 3-4, pp. 289-300.

Luba T., Nowicka M., Rawski M. (1996). Performance-oriented Synthesis for LUT-based FPGAs, *Proc. Mixed Design of Integrated Circuits and Systems*, pp. 96-101, Lodz.

Luba T., Moraga C., Yanushkevich S., Opoka M., Shmerko V. (2000). Evolutionary Multilevel Network Synthesis in Given Design Style. *Proc. IEEE 30th Int. symposium on Multiple-Valued Logic*, pp.253-258.

Qiao J., Ikeda M., Asada K. (2000). Optimum Functional Decomposition for LUT-Based FPGA Synthesis. *Proc. of the FPL'2000 Conference*, pp. 555-564, Villach.

Rawski M., Jozwiak L., Nowicka M., Łuba T. (1997). Non-Disjoint Decomposition of Boolean Functions and Its Application in FPGA-oriented Technology Mapping. *Proc. of the EUROMICRO'97 Conference*, pp.24-30, IEEE Computer Society Press.

Ross T., Noviskey M., Taylor T., Gadd D. (1991). Pattern Theory: An Engineering Paradigm for Algorithm Design. *Final Technical Report*, Wright Laboratories, WL/AART/WPAFB.

Zupan B, Bohanec M. (1966). Experimental Evaluation of Three Partition Selection Criteria for Decision Table Decomposition. *Research Report*, Department of Inteligent Systems, Josef Stefan Institute, Ljubljana.

Zupan B, Bohanec M. (1966). Learning Concept Hierarchies from Examples by Functional Decomposition. *Research Report*, Department of Inteligent Systems, Josef Stefan Institute, Ljubljana.