AREA OPTIMIZATION OF ROM-BASED CONTROLLERS DEDICATED TO DIGITAL SIGNAL PROCESSING APPLICATIONS

Bertrand LE GAL, Aurélien RIBON, Lilian BOSSUET and Dominique DALLET

IMS Laboratory - UMR CNRS 5218
University of Bordeaux - Talence, FRANCE
e-mail: {firstname.lastname}@ims-bordeaux.fr

ABSTRACT

The interest in using High-Level Synthesis flows to design Digital Signal Processing (DSP) circuits greatly increased in the last years. This is primarily due to the growing processing complexity combined with the limitations of the time-to-market constraint. Dedicated processor design is a complex process, and tools have to optimize processor datapath and controller. In this paper, we propose a controller design flow based on mapping Finite-State Machines into Memory Blocks in order to limit the controller critical path. Our design flow approach takes into account DSP circuit singularities providing efficient area saving compared to other approaches (more than 5%, and up to 62% on real life applications).

1. INTRODUCTION

Custom circuits generated using High-Level Synthesis (HLS) design methodologies [1, 2] are based on generic architectures. Actually, these custom circuits, usually dedicated to Digital Signal Processing (DSP) applications [3, 4], are composed of two parts: a datapath to perform computations and a controller to control the hardware resources. Generated circuit complexity increases with application functionalities [5, 6], performance and system constraints [7]. Design complexity heavily impacts on the circuit controller size. Therefore, this controller may become a performance bottleneck for the circuit due to increasing critical path delay (this delay limits the circuit maximum clock frequency and throughput).

Many controller related issues have been addressed in control-intensive researches. It has been demonstrated that implementing a controller using a ROM based design provides interesting characteristics [8]. However, existing techniques have been developed for control-intensive applications and must be adapted to efficiently manage computation-intensive application specificities.

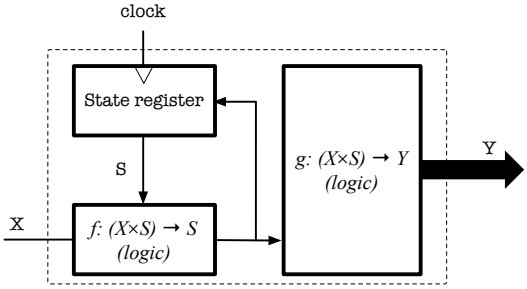


Figure 1 - Logic-based implementation of a FSM controller

In this paper, we present a design flow to save memory area for ROM-based controllers dedicated to custom DSP circuits. These circuits – hand-written or automatically generated using HLS tools – have two main characteristics: (1) every datapath resource is not used on each clock cycle [9, 10] (this can permit command signal optimizations) and (2) the controller can be split into smaller parts for better area reduction.

This paper approach is different from literature ones, indeed we do not consider that the next state computation part of the controller is the most complex one. In the case of DSP circuits, the controller complexity is located in the output decoder part of the design. The main issue is in this case to factorize efficiently the output command signals to save design area.

Article is organized as follows. Section 2 presents the literature approaches for ROM based controller implementation optimizations and explains the motivation for studying this type of solution. Section 3 details the area optimization algorithm used and extend literature approaches to handle DSP singularities. Experimental results validating our methodology are reported in Section 4. Finally, Section 5 concludes this paper.

2. RELATED WORKS

Controller optimization techniques [11, 12] have been developed considering logic-based implementation (Figure 1). Logic based controller design has been proved inefficient for controllers with: large number of states [13, 14] and huge number of output signals.

One way to cope the relation between the critical path and the number of FSM states is to implement the design controller in a ROM-based design (Figure 2). Using such controller architecture, the output values and the transition conditions are pre-computed before logical synthesis and stored in ROM element. In this controller architecture, the critical

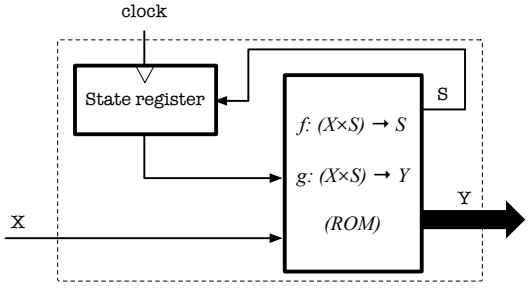


Figure 2 - ROM-based implementation of a FSM controller

 ISSN 2076-1465

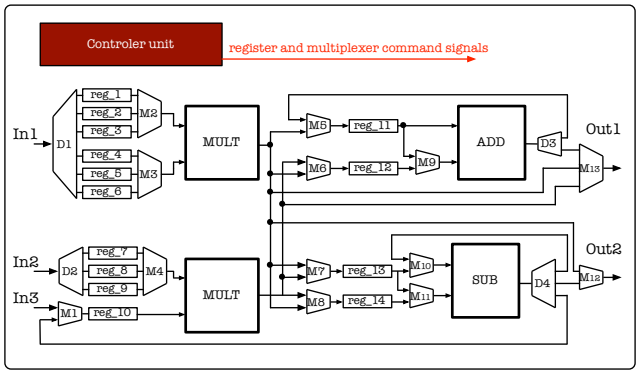


Figure 3 - Circuit composed of a datapath and its controller

path is almost constant whatever the number of states and the number of resources (depending only on resource characteristics, place and route choices and logical synthesis tool options [15]).

General methods for ROM-based controller synthesis targeting implementation of sequential circuits using embedded memory blocks have been proposed in [8, 16]. These methods dedicated to control-intensive applications, save ROM area by decomposing the memory block (corresponding to the controller) into two blocks: a semi-combinational address modifier and a smaller memory block to store the output values. An appropriately chosen decomposition strategy may reduce the required memory size at the cost of additional logic cells. These optimizations focus only on the next address computation part of the controller implementation. A similar approach was proposed in [17] which considers the controller power consumption problem. Finally, in [18, 19], the author uses *don't care* value to simplify state transition equations. This simplification reduces the memory size as well as multiplexer complexity of the address modifier part only of the controller.

Literature approaches consider general FSM models with uncorrelated output signals. These works consider that the next state computation part of the controller is more complex that the output decoding one. DSP circuit controllers do not have such characteristics:

1. The FSM models are mainly linear (the next state decoding equations and conditions are simples)
2. The output signal set is complex (huge numbers of states and output signals).

In custom DSP circuits, the FSM output signals are used to control the datapath resources like ones shown in Figure 3. The controller next state computation part is elementary

Controller State	Ouput signals						
	1	2	3	4	5	6	7
1	X	X	1	1	1	1	1
2	X	1	X	0	X	0	0
3	X	1	X	1	X	0	0
4	0	1	1	1	1	0	0
5	1	0	1	X	X	1	1
6	X	0	X	1	X	0	0
7	X	X	1	1	X	1	1

Controller State	Ouput signals				
	1	2/5	3	4	6/7
1	X	1	1	1	1
2	X	1	X	0	0
3	X	1	X	1	0
4	0	1	1	1	0
5	1	0	1	X	1
6	X	0	X	1	0
7	X	X	1	1	1

Figure 4 - Column compaction examples.

Controller State	Decoder ouput signals					
	1	2	3	4	5	6
1	X	0	1	1	X	1
2	X	1	X	0	1	0
3	X	1	X	1	X	0
4	0	1	1	1	X	0
5	1	0	1	X	1	0
6	X	0	X	1	0	X
7	0	X	1	1	X	1

Controller State	Ouput Instr.
1	1
2	2
3	3
4	3
5	4
6	1
7	1

Instr dec.	Decoder ouput signals					
	1	2	3	4	5	6
1	0	0	1	1	0	1
2	X	1	X	0	1	0
3	0	1	1	1	X	0
4	1	0	1	X	1	0

Figure 5 - Instruction compaction examples.

since its execution path is linear.

Proposed approach - An efficient way to design this kind of controllers is to implement the next state computation function *f* using an adder, a register and a multiplexer resource. The output decoding function *g* is implemented using a ROM memory. Each word of the ROM stores the output commands associated to state *S*. In this paper, we propose a cluster-based methodology, efficiently reducing the ROM area associated to output signal generation.

3. AREA SAVING TECHNIQUES

ROM area increases with the number of resources and the number of FSM states. Depending on circuit complexity, these requirements can become huge. Two literature techniques have been proposed, removing spatial and temporal redundancy. These techniques use the fact that command signals (controller outputs) are not required for each hardware resources at each clock cycle. Undefined command values named *don't care* values are represented using *X* in the truth table (example in Figure 4). Don't care values help the ROM area reduction process, as they can be modified without any design functionality impact.

Spatial redundancy - The first approach to save ROM area is to realize column compaction [20, 21]. This technique aims at removing output signals (columns) which are logically equivalent, or can be made equivalent through assignment of *don't cares*. Given a set of output columns, the problem of finding the smallest column set to drive the overall datapath resources can be obtained by compacting the given set. Figure 4 present the thrust table before and after the optimization process. This problem is related to the maximum clique-partitioning problem, which is *NP*-complete.

Removing temporal redundancy – The second approach is used to remove the inter-instruction redundancy, reducing the ROM height. Removing the temporal redundancies modifies output computation function $g:X\times S\rightarrow Y$ updating it

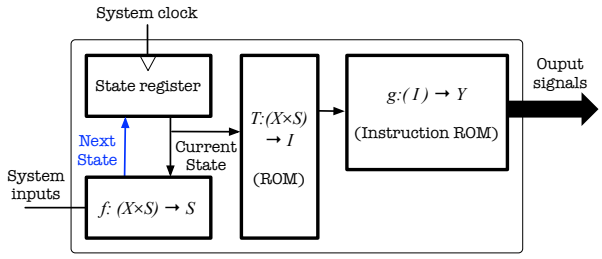
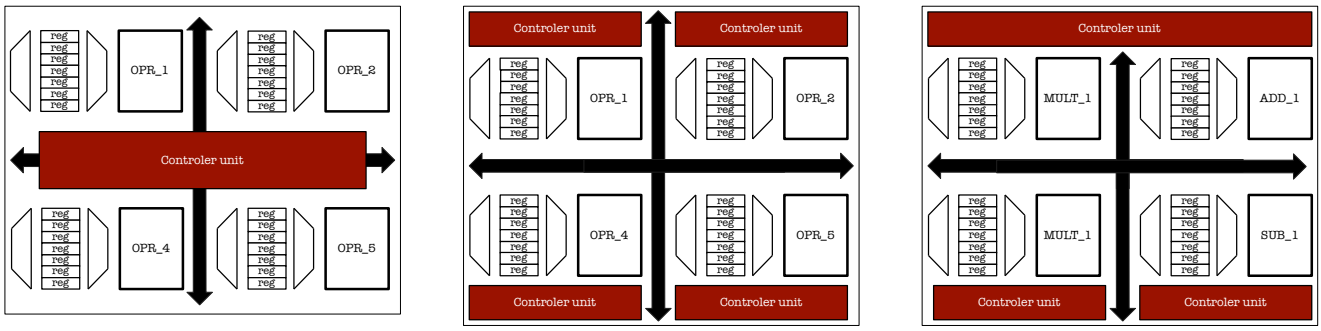


Figure 6 - Architecture for instruction indexed controllers



(a) A single controller manages the overall elements (b) Each cluster has its own controller (c) Mixed approach where controllers are factorized to minimize area

Figure 7 - Possible controller designs in custom DSP circuits

by $g':T(X\times S)\rightarrow Y$ where $T:(X\times S)\rightarrow I$ is the function which associates an instruction for each controller state. Figure 5 presents an example of instruction compaction result. This indexed relation between current controller state and the output signals required an architectural modification: a new ROM memory is required to implement the T relation. Modified controller architecture is shown in Figure 6.

However, this optimization technique may leads to ROM size increasing in some circumstancies i.e. when their exist a low instruction redundancy in the controller (indexing ROM may be more expensive than memory saved compacting instructions).

4. PROPOSED CLUSTER-BASED APPROACH

Using a single controller to manage the overall datapath resources (Figure 7a) is a bottleneck during the optimization process, i.e. merging rows or columns can be forbidden by only one bit value over hundreds. To solve this optimization issue, dedicated processing circuits can be divided into independent synchronous elements. These elements named clusters are atomic groups composed of:

- A computation resource (arithmetic or logic resource),
- Associated storage elements (registers),
- Required steering logic resources (multiplexers).

Each cluster has its own characteristics (i.e. computation starting and ending states which depends on resource usages specified during the HLS scheduling step [9, 10]). In example, architecture shown in Figure 3 is composed of 4 clusters: one for each multiplier (MULT), one for the substractor (SUB) and one for the adder (ADD). With such circuit decomposition (Figure 7b), each cluster controller can be optimized without considering others. This approach improves results obtained using instruction compaction technique.
Unfortunately, the drawback of duplicating controllers using an island-styled approach (Figure 7b) is design area increase. Fully clustered appraoch reduces the column compaction opportunities during area optimization step and it requires indexing ROMs in each cluster.
Efficient datapath controller design is located between the clustered approach and single ROM one (Figure 7c). The cluster-merging problem is an optimization problem where the objective function can be described as follow:

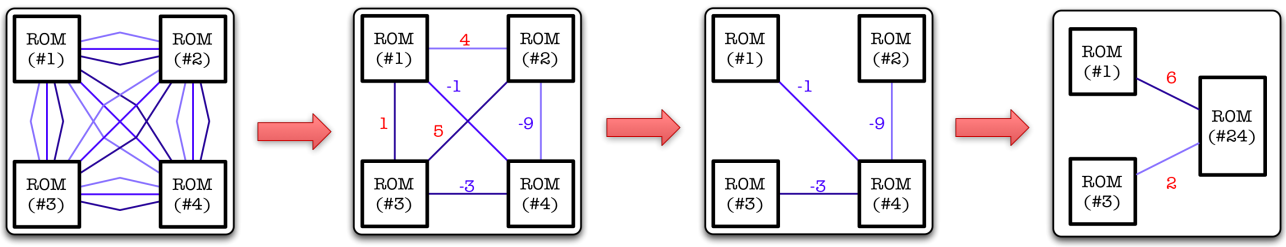
$$\min : \sum_{i=1}^{N} Area(c_i)$$

with N the number of controllers; $Area(c_i)$ the controller memory size of the i^{th} controller.

To find an efficient controller solution, a weighted graph $B=(C, E)$ is built. Each vertex $c_l \in C$ represents a cluster controller. Node c_i is weighted with v_i which represents the minimum memory cost of the i^{th} controller. $E \in (C \times C)$ is the set of weighted edges $e_{l,m}$ between c_l and c_m. Edges represent the merging possibility between the linked controller nodes. Weight $w_{l,m}$ associated with edge $e_{l,m}$ corresponds to the area saving (or lost) obtained while merging both linked controllers.

4.1 First Step: Creating the weighted graph

For each cluster c_i with $i \in [1, N]$ in the architecture, we



(a) Initial graph model (b) Redundant edge removing (c) Inefficient edge removing (d) ROM factorization result

Figure 8 - Bi-partite graph models obtained during the proposed optimization process.

Application	Technique	# of execution states	# of resources (controller output bits)	# of clusters in the design	ROM decoder output width (# of bits)	# of different instructions	# of ROM bits	ROM size (kByte)	EXP5 method saving
2d DCT	EXP1	80	646	1	646	80	52326	6,4	47,8 %
	EXP2			1	354	80	28674	3,5	4,8 %
	EXP3			1	354	80	28674	3,5	4,8 %
	EXP4			35	515	[2, 65]	34330	4,2	20,5 %
	EXP5			4	396	[16, 80]	27297	3,3	------
	EXP1	120	640	1	640	120	77440	9,5	51,9 %
	EXP2			1	330	120	39930	4,9	6,7 %
	EXP3			1	330	118	39787	4,9	6,4 %
	EXP4			26	481	[3, 97]	49792	6,1	25,2 %
	EXP5			3	348	[59, 117]	37253	4,5	------
64 taps FFT	EXP1	91	1247	1	1247	91	114724	14,0	67,3 %
	EXP2			1	692	91	63664	7,8	41,1 %
	EXP3			1	692	69	48392	5,9	22,5 %
	EXP4			61	1043	[6, 35]	45057	5,5	16,8 %
	EXP5			13	852	[29, 44]	37509	4,6	------
	EXP1	274	1069	1	1069	274	293975	35,9	80,6 %
	EXP2			1	551	274	151525	18,5	62,3 %
	EXP3			1	551	157	88707	10,8	35,6 %
	EXP4			54	873	[3, 65]	74305	9,1	23,2 %
	EXP5			10	677	[53, 89]	57095	7,0	------
								0,0	
inverse JPEG	EXP1	80	2805	1	2805	80	227205	27,7	77,7 %
	EXP2			1	1221	80	98901	12,1	48,7 %
	EXP3			1	1221	72	88479	10,8	42,6 %
	EXP4			256	2295	[4, 30]	78307	9,6	35,2 %
	EXP5			37	1674	[13, 33]	50749	6,2	------
	EXP1	140	2472	1	2472	140	348552	42,5	69,4 %
	EXP2			1	1110	140	156510	19,1	31,9 %
	EXP3			1	1110	125	139737	17,1	23,7 %
	EXP4			119	1963	[2, 76]	148909	18,2	28,4 %
	EXP5			4	1136	[91, 94]	106635	13,0	------

Table 1 - ROM area saving for two applications synthesized under different timing (latency) constraints

create a node c_i in B. For each node c_i we compute the associated controller minimum memory cost v_i. The optimal v_i weight is obtained after applying the overall optimization schemes. There exist three distinct possible ways to obtain the best ROM design:

1. Using column compaction only
2. Using column and instruction packing
3. Using the same optimizations as (2) but executed in reverse order.

These three optimization ways are explored and the best one (having the minimum cost value) is selected and used as node weight (v_i).

Once all nodes have been created, edges can be inserted. Each graph node is linked to all others. Each edge $e_{i,j}$ models the area saving obtained while merging controller c_i with c_j. For each node couple (c_i, c_j) we create three distinct edges, weighted using the merged controller cost obtained using the three optimization processes (similar as nodes weight). An example of such graph is presented in Figure 8a.

4.2 Second Step: Removing inefficient opportunities

Once the overall, weighted graph B has been constructed, we first eliminate the redundant edges linking node couples: for each couple (c_i, c_j) we only keep one edge $e_{i,j}$ corresponding to the best area saving. In case of area saving equivalence, column compacted only solution is preferred to other ones for critical path reason. This step result is presented in Figure 8b.

Finally, the inefficient merging possibilities are removed (merging opportunities which increase the controller design area). Each edge is evaluated and ones with $w_{i,j} > 0$ are removed. This model transformation is illustrated in Figure 8c.

4.3 Third Step: Incremental graph compaction

Graph compaction problem is solved using greedy approach to limit the algorithmic complexity. The B graph is analyzed to find the maximum weight $w_{i,j}$ value. This weight corresponds to best controller merging opportunity (area saving). We merge the two controllers associated to $e_{i,j}$, removing nodes c_i and c_j from B. A new node c_k is inserted in B. Edges linking c_k to c_m with $c_m \in B/\{c_k\}$ are created and weighted as described in algorithm Step 1. Newly created edges are optimized and the procedure performed again until there is no more available merging (Figure 8d).

Saving results may be improved considering smaller clusters at the optimization start i.e. one dedicated controller for each icrcuit element (registers and multiplexers). Unfortunately, increasing the number of clusters will increase in the same time drastically the optimization complexity and runtime.

5. EXPERIMENTAL RESULTS

In this section, we present experimental results. The ROM based optimization techniques have been integrated in the VHDL backend of the *GraphLab HLS* tool [22]. Experiments are based on usually used *digital signal processing* applications. The optimization process results have been

obtained for different synthesis constraint sets applied to the applications. Generated circuits have different controllers architectures (different number of resources to manage, different number of states, different decoder filling). This procedure helps us to provide a fairly evaluation of the decoder area saving obtained. Five methodologies have been compared:

- **EXP1**: ROM-based implementation of the controller without any optimization.
- **EXP2**: ROM-based implementation of the controller optimized using column compaction technique.
- **EXP3**: ROM-based implementation of the controller optimized using column and instruction compaction techniques. Best solution obtained using both optimization orders is provided.
- **EXP4**: ROM-based implementation of the controller using a full clustered-based approach: each cluster has its own ROM decoder. ROM decoders are column and instruction compacted.
- **EXP5**: ROM-based implementation of the controller is obtained using the proposed approach. Clusters are optimized and merged according to area saving opportunities as described in Section 4.

Results presented in Table 1 show ROM area saving obtained for benchmark applications. Results highlight that the proposed technique always provide better area saving (from 5% to 62% saving, average=27%) compared to other methodologies. Experiments also prove that method efficiency depends on controller characteristics and content. Finally, proposed approach helps in finding interesting tradeoffs between column and instruction compaction.

6. CONCLUSION

In this paper, we have presented a new ROM area saving methodology which takes care of DSP circuit singularities to extend literature approaches. The controller architecture is generated using an efficient trade-off between single and fully clustered controller approaches. Proposed methodology is integrated to a high-level synthesis tool. As the experimental results show, the controllers generated using our design flow have significantly less area: 5% up to 62% compared to a single based controller design and 17% up to 36% compared to a fully clustered approach.

REFERENCES

[1] J. P. Elliott, *Understanding Behavioral Synthesis*. A Practical Guide to High-Level Design. Kluwer Academic Publishers, 2000.

[2] *Special issue on High-Level Synthesis*, P. Coussy and A. Takach Ed., IEEE Design & Test of Computers, vol. 26, issue 4, 2009.

[3] E. Casseau, B. Le Gal, P. Bomel, C. Jégo, S. Huet, and E. Martin. *"C-based rapid prototyping for digital signal processing"*. In Proceedings of the European Signal Processing Conference (EUSIPCO), Antalya, Turquie, 4-8 September 2005.

[4] Arvind et al. *"High-Level Synthesis: an essential ingredient for designing complex ASICs,"* In Proceedings of the Internationnal Conference on Computer-Aided Design, pp. 775–782, 2004.

[5] W. Kaijie, and R. Karri, *"Fault Secure Datapath Synthesis using Hybrid Time and Hardware Redundancy,"* IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems, vol 23, No. 10, pp. 1476-1484, October 2004.

[6] B. Le Gal and E. Casseau, *"Automated Multimode System Design for High Performance DSP Applications,"* In Proceedings of the European Signal Processing Conference (EUSIPCO), Glasgow, Scotland, August 24-29, 2009.

[7] S. Ahuja, W. Zhang, L. Avinash, S. K. Shukla, *"A Methodology for Power Aware High-Level Synthesis of Co-processors from Software Algorithms,"* In Proceedings of the VLSI Design Conference, pp. 282-287, 2010.

[8] H. Selvaraj, M. Rawski, T. Łuba, *"FSM implementation in embedded memory blocks of programmable logic devices using functional decomposition,"* in Proceedings of International Conference on Information Technology: Coding and Computing (ITCC), p. 355, 2002.

[9] G. Lakshminarayana, A. Raghunathan, N.K. Jha, S. Dey, *"Transforming control-flow intensive designs to facilitate power management,"* In Proceedings of the IEEE/ACM international Conference on Computer-Aided Design (DAC), pp. 657-664, 1998.

[10] E. Musoll and J. Cortadella, *"High-level synthesis techniques for reducing the activity of functional units."* In Proceedings of the International Symposium on Low Power Design, pp. 99-104, 1995.

[11] W. Shiue, *"Power/Area/Delay aware FSM synthesis and optimization,"* Microelectronics Journal, vol. 36, no. 2, pp. 147–162, February 2005.

[12] K. Kuusilinna, V. Lahtinen, T. Hamalainen, and J. Saarinen, *"Finite state machine encoding for VHDL synthesis,"* In Computers and Digital Techniques, vol. 148, no. 1, pp. 23–30, 2001.

[13] P. Bomel, E. Martin, and E. Boutillon, *"Synchronization processor synthesis for latency insensitive systems,"* in Proceedings of the Design Automation and Test in Europe Conference, p. 896, 2005.

[14] M.P. Desai, H. Narayanan, S. Patkar, *"The Realization of Finite State Machines by Decomposition and the Principal Lattice of Partitions of a Submodular Function,"* Special Issue on Submodular Functions, Discrete Applied Maths, vol. 131, pp. 299-310, 2003.

[15] Xilinx corporation, *"Block Memory Generator, "* version 3.3, September 16, 2009.

[16] M. Rawski, H. Selvaraj, and T. Luba, *"An application of functional decomposition in ROM-based FSM implementation in FPGA devices,"* Journal of Systems Architecture, vol. 51, p. 424, 2005.

[17] A.Tiwari and K.Tomko, *"Saving power by mapping finite-state machines into embedded memory blocks in FPGAs,"* in Proceedings of the conference on Design, Automation and Test in Europe (DATE), page 20916, 2004.

[18] R. Senhadji-Navarro, I. Garcia-Vargas, G. Jimenez-Moreno, A. Civit-Ballcels, *"ROM-based FSM implementation using input multiplexing in FPGA devices,"* Electronics Letters, Volume 40, Issue 20, pages 1249-1251, Octobre 2004

[19] I. García-Vargas, R. Senhadji-Navarro, G. Jiménez-Moreno and A. Civit-Balcells, P. Guerra-Gutiérrez, *"ROM-Based Finite State Machine Implementation in Low Cost FPGAs"*. In the Proceedings of the International Symposium on Industrial Electronics (ISIE), pp. 2342-2347. Vigo, Spain, June 4-7, 2007.

[20] D. Binger and D.W. Knapp, *"Encoding multiple outputs for improved column compaction," in Proceedings of the International Conference on Computer Aided Design (ICCAD),* pp 230–233, 1991.

[21] S. Mitra, L. Avra and E. Mc Cluskey, *"An output encoding problem and a solution technique,"* in Proceedings of the International Conference on Computer-Aided Design (ICCAD),pp 304–307, 1997.

[22] E. Casseau and B. Le Gal, *" High-Level Synthesis for the Design of FPGA-based Signal Processing Systems"*, In the Proceedings of SAMOS'09, Samos, Greece, July 20-23, 2009