Reducing Power Consumption of Dedicated Processors Through Instruction Set Encoding

Luca Benini [#] Giovanni De Micheli [#] Alberto Macii [‡] Enrico Macii [‡] Massimo Poncino [‡]

[#] Stanford University
Computer Systems Laboratory
Stanford, CA 94305

[‡] Politecnico di Torino
Dip. di Automatica e Informatica
Torino, ITALY 10129

Abstract

With the increased clock frequency of modern, high-performance processors (over 500 MHz, in some cases), limiting the power dissipation has become the most stringent design target. It is thus mandatory for processor engineers to resort to a large variety of optimization techniques to reduce the power requirements in the hot zones *of the chip. In this paper, we focus on the power dissipated by the instruction fetch and decode logic, a portion of the processor architecture where a lot of capacitance switching normally takes place. We propose a methodology for determining an encoding of the instruction set that guarantees the minimization of the number of bit transitions occurring inside the registers of the pipeline stages involved in instruction fetching and decoding. The assignment of the binary patterns to the op-codes is driven by the statistics concerning instruction adjacency collected through instruction-level simulation of typical software applications; therefore, the technique is best exploited when applied to encode the instruction set of core processors and microcontrollers, since components of these types are commonly used to execute fixed portions of machine code within embedded systems. We illustrate the effectiveness of the methodology through the experimental data we have obtained on an existing microprocessor.*

1 Introduction

Power consumption is becoming one of the most relevant design constraints for modern digital systems. Obviously, the need of low-power electronics is even more stringent for portable applications, such as laptop computers, mobile phones, and personal digital assistants (PDAs). Usually, applications of this type carry on board devices such as general-purpose microprocessors, dedicated core processors, microcontrollers, and DSP processors that, besides reduced power dissipation, must ensure reasonably high performance, and thus run at fairly high clock frequencies [1].

There are several solutions that can be adopted at the architectural-level to limit the power required by processor-based systems. These include modifications of the processor's organization [2, 3], careful design of the memory and I/O subsystems [4, 5, 6, 7], proper choice of the data representation [8], adoption of power-management strategies [9, 10, 11], and exploitation of bus encoding/decoding techniques [12, 13, 14, 15, 16, 17].

In this paper, we focus our attention on the power dissipation within a processor. More precisely, we target the minimization of the power consumed by the instruction fetching and decoding circuitry. We move from the observation that sequences of op-codes are stored into some registers of the CPU's pipeline stages when a stream of machine instructions is fetched from the main memory and decoded. Loading a binary pattern into an N-bit register is usually a power consuming operation, since it may require to charge or discharge the capacitances associated to the cells of the register. To be more specific, up to N bit transitions can occur inside the register any time a new value is stored. Properly choosing the binary patterns to be assigned to the instruction op-codes is then key to reduce the number of transitions, and thus the switching component of the power dissipated by the fetch and decode logic of a CMOS processor.

The method for determining a near-optimal, low-power instruction set encoding we propose works as follows. First, instruction-level simulation is performed on traces of typical software programs, and the information about the number of adjacencies between pairs of instructions is recorded. Then, $\lceil \log_2(K) \rceil$-bit binary codes are assigned to the various instructions using an encoding algorithm which targets the minimization of the Hamming distance between the codes which are very likely to be adjacent in typical machine instruction streams (K is the number of op-codes in the processor's instruction set).

We show the effectiveness of our methodology through a case study. In particular, we have taken the MIPS R4000 [18] as the target processor, and we have collected the information on instruction adjacency when some general-purpose programs, such as compilers, DBMSs, word processors, and data compression tools are executed. Then, we have used the low-power encoding schemes of [19] and [20] to re-encode the binary op-codes implemented in reality. Finally, we have calculated the reduction in switching activity of the op-code bits in each register of the CPU pipeline by simulating the execution of the re-encoded instruction streams. Savings are between 15% and 42%, depending on the encoding algorithm and the benchmark program.

Clearly, the choice of an existing processor, such as the MIPS R4000, as our case study has been made only for the sake of illustration of the goodness and usefulness of our approach, whose applicability is thought for the early phases of the design flow of new processors. In particular, it is self-evident that the output of the op-code assignment procedure is heavily influenced by the specific software programs which are used to compute the frequency of instruction adjacency; therefore, our technique is best suited for dedicated processors, since they are commonly used within embedded systems to execute the same portion of code over and over. The use of such components, commonly identified as intellectual proprietary (IP) cores, as basic blocks for the development of special-purpose digital systems is becoming a well-established design strategy in the microelectronics industry. This is the reason why we believe that the proposed power optimization strategy could be of interest for IP core engineers and vendors.

2 Low-Power Instruction Set Encoding

As mentioned in the introduction, the procedure for the assignment of the binary patterns to the various op-codes consists of two phases. In the first one, instruction-level simulation is executed on typical instruction streams to collect the information concerning the adjacency of pairs of op-codes. Assuming an instruction set consisting of a total of K distinct op-codes, we store the adjacency statistics into a $K \times K$ matrix, called A, whose entries $a_{i,j}$ represent the number of times op-code i immediately precedes op-code j.

From matrix A, we can derive a weighted undirected graph, $G_A(V, E, W)$, with $|V| = K$ vertices. Each vertex, v_i, represents op-code i, and each edge $e_{i,j}$ is labeled $w_{i,j} = a_{i,j} + a_{j,i}$ and connects vertex v_i to vertex v_j. If $w_{i,j} \neq 0$, $\forall i \neq j$, $G_A(V, E, W)$ is completely connected, i.e., it is a clique.

Given graph $G_A(V, E, W)$, the target is to assign to each vertex v_i a binary code of length $\lceil \log_2(K) \rceil$ in such a way that the following cost function gets minimized:

$$C(G_A(V, E, W)) = \sum_{e_{i,j}} w_{i,j} \cdot H_{i,j} \qquad (1)$$

where $H_{i,j}$ is the Hamming distance between the two codes of vertices v_i and v_j. The ultimate objective is to assign closer (in the Hamming sense) codes to vertices joined by "heavy" edges. In this way, pairs of op-codes which are likely to be adjacent in the instruction stream, when stored in the same register at two consecutive clock cycles will require a small number of switchings within the register's bits, thus reducing the total switching power.

Example 1

Consider the graph $G_A(V, E, W)$ of Figure 1-a. If we encode the vertices $v_0, \ldots, v_3$ as in Figure 1-b, the cost is 1710. *On the other hand, with the encoding of Figure 1-c, the value of the cost function is* 1300.

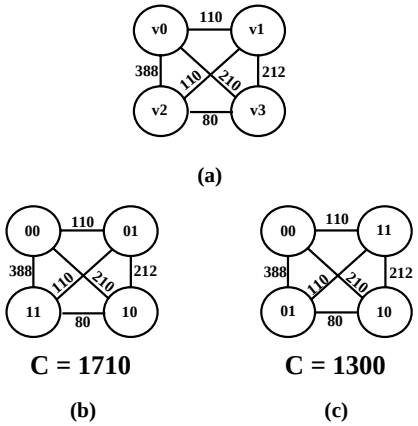

Figure 1: Example of $G_A(V, E, W)$ Encoding.

The problem we are facing (i.e., finding a minimum-length encoding for the vertices of graph $G_A(V, E, W)$ which minimizes the cost function of Equation 1) has been deeply investigated in the context of FSM encoding for low power, and efficient solutions can be found in the recent literature (see [21] for a comprehensive survey).

Exact encoding algorithms only work for very small graphs (a few tens of vertices), since the encoding problem is NP-hard. This is the reason why we propose to adopt two heuristic procedures. The first one [19], more accurate, is based on the solution of an integer liner programming problem, it relies on explicit enumeration of all the vertices in the graph, and it is applicable to mid-sized examples. The second procedure [20], on the other hand, is less accurate, but it is fully based on implicit representations of Boolean and pseudo-Boolean (i.e., real-valued) functions by means of BDDs [22] and ADDs [23], and solves the encoding problem as a maximum weighted matching problem; therefore, it is of interest for larger graphs (more than a few hundreds of vertices).

3 A Case Study

3.1 MIPS R4000 Architecture

The MIPS R4000 is a 64-bit microprocessor which provides a 64-bit on-chip Floating-Point Unit (FPU), a 64-bit integer Arithmetic Logic Unit (ALU), 64-bit integer registers, a 64-bit virtual address space, and a 64-bit system bus. Figure 2 shows a block diagram of the R4000 processor internal.

The R4000 processor realizes instruction parallelism by using an 8-stage superpipeline; each stage takes one *PCycle* (one cycle of *PClock*, which operates at twice the frequency of the *MasterClock*). The execution of each instruction takes at least eight *PCycles*. Normally, two instructions are issued at each *MasterClock* cycle. Once the pipeline has been filled, eight instructions are executed simultaneously. The processor achieves high throughput by pipelining cache accesses, reducing register access times, and allowing the latency of functional units to span more than one pipeline cycle. Figure 3 shows the 8-stages of the instruction pipeline.

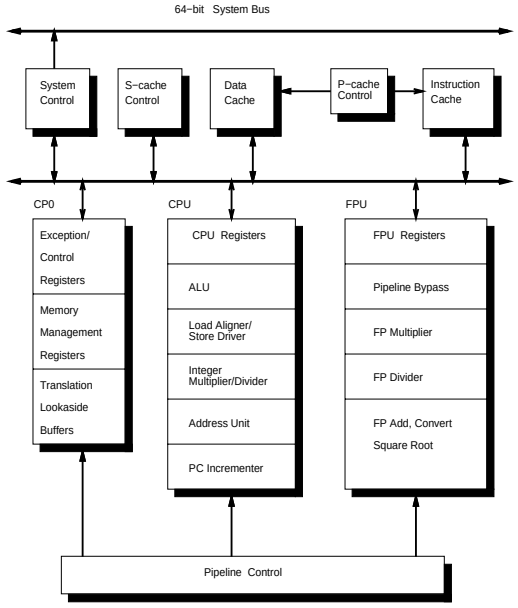

Figure 2: R4000 Processor Internal Block Diagram.

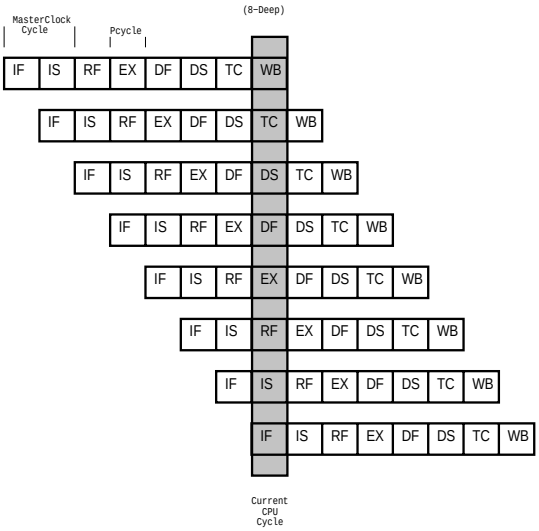

Figure 3: Instruction Pipeline Stages.

3.2 Instruction Set Summary

Each CPU instruction is 32 bits long. There are three instruction formats, shown in Figure 4:

- *Immediate* (I-type);
- *Jump* (J-type);
- *Register* (R-type).

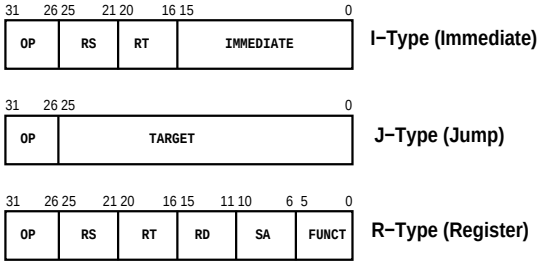

Figure 4: CPU Instruction Formats.

Instruction decoding is greatly simplified by limiting the number of formats to these three. This limitation means that the more complicated and less frequently used operations and addressing modes can be synthesized by the compiler, using sequences of these same simple instructions.
Concerning the functions they perform, instructions can be grouped into seven categories: *Load and Store*, *Computation*, *Jump and Branch*, *Coprocessor*, *Coprocessor 0*, *Special*, and *Exceptions*.
The instruction op-codes are 6-bit long; their binary encodings are shown in Table 1 (rows are labeled with bits [31..29], and columns are labeled with bits [28..26]).

3.3 Low-Power Instruction Set

To show the effectiveness of the low-power instruction set encoding methodology of this paper, we present results we have obtained on the MIPS R4000 microprocessor.
We have selected a total of eight software applications; for each of them we have built matrix A and the corresponding graph G_A; finally, we have constructed the global graph, $G_A^G = \sum G_A$, and we have run on it the explicit and the implicit encoding algorithms of [19] and [20]. Tables 2 and 3 show the new binary patterns for all the op-codes.
By inspection of the two tables, it can be observed that the application of the explicit encoding algorithm has modified almost all the op-codes; on the other hand, a lower number of changes has been introduced by the implicit algorithm. This behavior was somehow expected, since the explicit algorithm introduces fewer approximations in the computation of the near-optimal codes, and therefore it more heavily modifies the initial encoding. The simulation results of the next section demonstrate that the op-codes of Table 2 are substantially better than the ones of Table 3. On the other hand, as already mentioned, the use of the algorithm of [19] may not be feasible in the case of op-codes longer than 6 bits.

[31..29] \ [28..26]	000	001	010	011	100	101	110	111
000	SPECIAL	REGIMM	J	JAL	BEQ	BNE	BLEZ	BGTZ
001	ADDI	ADDIU	SLTI	SLTIU	ANDI	ORI	XORI	LUI
010	COP0	COP1	COP2	RESRVD1	BEQL	BNEL	BLEZL	BGTZL
011	DADDI	DADDIU	LDL	LDR	RESRVD2	RESRVD3	RESERVD4	RESRVD5
100	LB	LH	LWL	LW	LBU	LHU	LWR	LWU
101	SB	SH	SWL	SW	SDL	SDR	SWR	CACHE
110	LL	LWC1	LWC2	RESRVD6	LLD	LDC1	LDC2	LD
111	SC	SWC1	SWC2	RESRVD7	SCD	SDC1	SDC2	SD

Table 1: Original Instruction Set Encoding.

[31..29] \ [28..26]	000	001	010	011	100	101	110	111
000	LW	SB	J	XORI	REGIMM	LDR	LL	BLEZL
001	LUI	LWU	SWL	SDR	LWR	RESRVD3	BGTZL	BEQL
010	SW	RESRVD6	ADDI	SDC1	SLTIU	LDC1	RESRVD1	SCD
011	LWC1	SDC2	COP2	RESRVD7	LWL	RESRVD5	LDC2	LDL
100	SPECIAL	ANDI	JAL	ORI	BNE	BGTZ	LH	LWC2
101	ADDIU	SH	BLEZ	SC	LBU	SDL	DADDIU	RESRVD2
110	BEQ	SLTI	LB	LD	LHU	COP0	SWC2	CACHE
111	COP1	SD	SWR	RESRVD4	SWC1	LLD	BNEL	DADDI

Table 2: Low-Power Instruction Set Encoding (Explicit Algorithm).

[31..29] \ [28..26]	000	001	010	011	100	101	110	111
000	BLEZ	BGTZ	REGIMM	SLTIU	BNE	SLTI	BEQ	ANDI
001	ORI	XORI	JAL	LUI	SPECIAL	LW	ADDIU	SW
010	ADDI	LB	LWL	LWR	SWL	SWR	LH	CACHE
011	J	SB	LHU	SH	COP1	LWC1	SC	SWC1
100	DADDI	DADDIU	LDL	LDR	RESRVD2	RESRVD3	RESRVD4	RESRVD5
101	BEQL	BNEL	BLELZ	BGTZL	LBU	LWU	SDL	SDR
110	COP0	LL	LWC2	RESRVD6	LLD	LDC1	LDC2	LD
111	COP2	RESRVD1	SWC2	RESRVD7	SCD	SDC1	SDC2	SD

Table 3: Low-Power Instruction Set Encoding (Implicit Algorithm).

3.4 Results for Real Software Applications

The end result we expect from the application of our technique is a reduction of the switching activity in the op-codes bits of some registers of the pipeline stages when sequences of machine instructions are executed. To make sure that this is actually what happens, we have taken the machine code of eigth different software programs, and for each of them we have monitored the average switching activity per op-code bit that occurs inside the fetch and decode registers when the program is executed. Then, for each application, we have determined the low-power instruction set encoding using both the explicit and the implicit algorithm, we have re-encoded the original instruction stream using the new op-codes, and we have calculated the average switching activity per op-code bit that would occur inside the fetch and decode registers when the re-encoded instruction stream were executed. Table 4 shows the results of the comparison. Savings are considerably high: Between 30% and 42% for the explicit encoding algorithm, and between 15% and 33% for the implicit one.
As mentioned, the results in the table are obtained using *ad-hoc* instruction encodings. Therefore, they clearly show the usefulness of the proposed approach as a tool for helping in determining the most suitable encoding for a special-purpose machine on which a well-established piece of embedded code will be repeatedly executed. However, the proposed methodology can be beneficial also to designers of general-purpose microprocessors. For devices of this type, the goal would be to determine the *best average* encoding, that is, the one which minimizes the power for most of the applications whose execution on the processor is the most likely to happen. The approach to be followed is then that of collecting the statistics on instruction adjacency for all such applications, and then use this information to determine the new encoding. To show the applicability of our technique also to the case of general-purpose machines, we have re-encoded the instruction stream of each program using the op-codes of Tables 2 and 3, and we have determined the average switching activity per op-code bit before and after re-encoding. The last row of Table 4, named **Global**, reports the average of these values taken over the eigth programs we have considered. Savings are larger than 30%.

Program	*Average Switching Activity Per Op-Code Bit*				
		Explicit Algorithm		*Implicit Algorithm*	
	Before	*After*	*Savings*	*After*	*Savings*
espresso	0.3025	0.1752	42.06%	0.2045	32.39%
gs	0.2995	0.2017	32.64%	0.2449	18.22%
gunzip	0.2989	0.1941	35.07%	0.2337	21.80%
gzip	0.3206	0.2062	35.67%	0.2582	19.45%
jedi	0.2861	0.1779	37.80%	0.1898	33.65%
latex	0.3036	0.2097	30.91%	0.2576	15.14%
matlab	0.3340	0.2062	38.25%	0.2576	22.86%
oracle	0.3443	0.2117	36.83%	0.2684	22.05%
Global	0.3012	0.1950	35.23%	0.2091	30.57%

Table 4: Average Switching Activity Reduction.

4 Conclusions

It is well established that modern microprocessors, including application-specific products (e.g., embedded cores and microcontrollers), are performance-critical devices, since they tend to run at very high clock frequencies; consequently, they normally consume a considerable amount of power. Designers are thus constrained to resort to optimization techniques to keep the available power budget under control.

In this paper, we have directed our attention to the power dissipated by the fetch and decoding logic of a processor. We have demonstrated that the choice of the instruction binary codes plays a key role in the minimization of the power consumed by this portion of the chip. We have therefore presented a methodology that can be fruitfully exploited by processor engineers to determine a near-optimal, low-power assignment of the op-codes for special-purpose machines, and we have supported our claims concerning the viability and the effectiveness of the proposed technique through experimental results collected on a real-life microprocessor, namely, the MIPS R4000.

Acknowledgments

This work is supported, in part, by a grant from SGS-Thomson Microelectronics.

References

[1] A. P. Chandrakasan, S. Sheng, R. W. Brodersen, "Low-Power CMOS Digital Design," *IEEE Journal of Solid-State Circuits*, Vol. 27, No. 4, pp. 473-484, April 1992.

[2] S. Gary, "Low-Power Microprocessor Design," *Low Power Design Methodologies*, J. M. Rabaey and M. Pedram Editors, Chapter 9, Kluwer Academic Publishers, 1996.

[3] D. Dobberpuhl, "The Design of a High Performance Low-Power Microprocessor," *ISLPED-96: ACM/IEEE International Symposium on Low-Power Electronics and Design*, pp. 11-16, Monterey, CA, August 1996.

[4] S. Wuytack, F. Catthoor, L. Nachtergaele, H. De Man, "Global Communication and Memory Optimizing Transformations for Low Power Design," *IWLPD-94: ACM/IEEE International Workshop on Low Power Design*, pp. 203-208, Napa Valley, CA, April 1994.

[5] P. R. Panda, N. D. Dutt, "Reducing Address Bus Transitions for Low Power Memory Mapping," *EDTC-96: IEEE European Design and test Conference*, pp. 63-67, Paris, France, March 1996.

[6] S. Wuytack, F. Catthoor, L. Nachtergaele, H. De Man, "Power Exploration for Data Dominated Video Applications," *ISLPED-96: ACM/IEEE International Symposium on Low Power Electronics and Design*, pp. 359-364, Monterey, CA, August 1996.

[7] J. P. Diguet, S. Wuytack, F. Catthoor, H. De Man, "Formalized Methodology for Data Reuse Exploration in Hierarchical Memory Mappings," *ISLPED-97: ACM/IEEE International Symposium on Low Power Electronics and Design*, pp. 30-35, Monterey, CA, August 1997.

[8] S. Chandrakasan, R. W. Brodersen, "Minimizing Power Consumption in Digital CMOS Circuits," *Proceedings of the IEEE*, Vol. 83, No. 4, pp 498-523, April 1995.

[9] M. Alidina, J. Monteiro, S. Devadas, A. Ghosh, M. Papaefthymiou, "Precomputation-Based Sequential Logic Optimization for Low Power," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. VLSI-2, No. 4, pp. 426-436, December 1994.

[10] L. Benini, P. Siegel, G. De Micheli, "Automatic Synthesis of Gated Clocks for Power Reduction in Sequential Circuits," *IEEE Design and Test of Computers*, Vol. 11, No. 4, pp. 32-40, Winter 1994.

[11] M. B. Srivastava, A. Chandrakasan, R. W. Brodersen, "Predictive System Shutdown and Other Architectural Techniques for Energy Efficient Programmable Computation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. VLSI-4, No. 1, pp. 42-55, January 1996.

[12] C. L. Su, C. Y. Tsui, A. M. Despain, "Saving Power in the Control Path of Embedded Processors," *IEEE Design and Test of Computers*, Vol. 11, No. 4, pp. 24-30, Winter 1994.

[13] M. R. Stan, W. P. Burleson, "Bus-Invert Coding for Low-Power I/O," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 3, No. 1, pp. 49-58, March 1995.

[14] L. Benini, G. De Micheli, E. Macii, D. Sciuto, C. Silvano, "Asymptotic Zero-Transition Activity Encoding for Address Busses in Low-Power Microprocessor-Based Systems", *GLS-VLSI-97: IEEE 7th Great Lakes Symposium on VLSI*, pp. 77-82, Urbana, IL, March 1997.

[15] L. Benini, G. De Micheli, E. Macii, M. Poncino, S. Quer, "System-Level Power Optimization of Special Purpose Applications: The Beach Solution," *ISLPED-97: ACM/IEEE International Symposium on Low Power Electronics and Design*, pp. 24-29, Monterey, CA, August 1997.

[16] E. Musoll, T. Lang, J. Cortadella, "Exploiting the Locality of Memory References to Reduce the Address Bus Energy," *ISLPED-97: ACM/IEEE International Symposium on Low Power Electronics and Design*, pp. 202-207, Monterey, CA, August 1997.

[17] A. Kalambur, M. J. Irwin, "An Extended Addressing Mode for Low Power," *ISLPED-97: ACM/IEEE International Symposium on Low Power Electronics and Design*, pp. 208-213, Monterey, CA, August 1997.

[18] J. Heinrich, *MIPS R4000 Microprocessor User's Manual*, Second Edition, MIPS Technologies, Mountain View, CA, 1994.

[19] L. Benini, G. De Micheli, "State Assignment for Low Power Dissipation," *IEEE Journal of Solid State Circuits*, Vol. 30, No. 3, pp. 258-268, March 1995.

[20] G. D. Hachtel, M. Hermida, A. Pardo, M. Poncino, F. Somenzi, "Re-Encoding Sequential Circuits to Reduce Power Dissipation," *ICCAD-94: IEEE/ACM International Conference on Computer-Aided Design*, pp. 70-73, San Jose, CA, November 1994.

[21] E. Macii, "Sequential Synthesis and Optimization for Low Power," *Low Power Design in Deep Submicron Electronics*, J. Mermet and W. Nebel Editors, Chapter 5.3, Kluwer Academic Publishers, 1997.

[22] R. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Transactions on Computers*, Vol. C-35, No. 8, pp. 79-85, August 1986.

[23] R. I. Bahar, E. Frohm, C. Gaona, G. D. Hachtel, E. Macii, A. Pardo, F. Somenzi, "Algebraic Decision Diagrams and their Applications," *Formal Methods in System Design*, Vol. 10, pp. 171-206, 1997.