Integrated Resynthesis for Low Power

Olivier Coudert† Ramsey Haddad†

Synopsys Inc., 700 East Middlefield Rd.
Mountain View, CA 94043, USA

Abstract

Research on synthesis for low power has been done in all three stages of logic synthesis: technology independent optimization, technology mapping, and technology dependent optimization. This paper presents an integrated method, using remapping and technology dependent optimizations, to minimize the power of a mapped circuit under the given delay constraints. It produces 24% savings in power.

1 Introduction

Due to a growing market in portable devices, modern day electronic systems require a high degree of performance and portability at a low cost. In the past, logic synthesis tools have addressed the problem of performance and cost by optimizing for area under a given delay constraint. However, portability requires the system to also be energy efficient. Optimizing for area does not necessarily reduce the power dissipation, and increasing performance does not necessarily increase the power. Hence synthesis tools must now operate in the 3-dimensional, non-linear, non-convex, space of delay, power, and area.

Synthesis for low power dissipation may be applied at the three steps in logic synthesis: technology independent optimization, technology mapping, and technology dependent optimization. Technology independent optimization consists of applying boolean and algebraic transformations to the boolean representation of the circuit. This can be used to reduce the switching activity of the circuit [3, 11], which can save dynamic power. The main problem is that technology independent optimization may not be reflected in the physical circuit, because the former is far removed from the final mapping of the circuit.

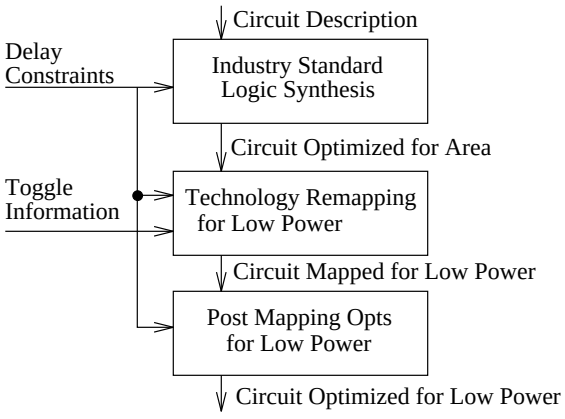

Figure 1: Overall flow of resynthesis for low power.

This paper attacks the power optimization problem at two different stages of synthesis: technology mapping and technology dependent optimization. Nothing excludes this technique from being merged with technology independent optimization algorithms to produce a complete logic synthesis package for low power. The overall methodology is shown in Fig. 1. A circuit is first mapped with an industry standard synthesis tool to meet delay constraints and optimize area. It is then remapped and a number of post-mapping optimizations are applied to minimize the power consumption under the given delay constraints.

Section 2 discusses the cost models used in the optimization procedures. Section 3 addresses the technology mapping algorithm. It is an extension of the algorithm presented in [14] to optimize for low power. This enables us to perform technology mapping for low power while being able to reconsider some of the decisions that are normally frozen during the technology independent stage. Section 4 covers the post-mapping optimization, which involves gate sizing, phase assignment, buffering, and pin swapping. Experimental results are discussed in section 5.

†This work was partially funded by ARPA under contract no. F33615-95-C1627.

ISLPED 1996 Monterey CA USA
0-7803-3571-8/96/$5.00 © 1996

2 Cost models

This section discusses the three cost models a logic synthesis has to face: area, power, and delay. Area is the easiest to model, since the area of each gate is known, and the routing area can be accurately estimated from statistical data.

The power dissipated in a gate is:

$$P_{total} = P_{load} + P_{internal} + P_{leakage}$$

The term P_{load}, the net dynamic power, is due to the charging and discharging of the output load of the gate. It depends on the toggle rate (number of transitions per unit of time) of the output net, and on the output load. The term $P_{internal}$, the internal gate dynamic power, depends on the toggle rates and slope of the inputs, and on the internal loads. The term $P_{leakage}$, the static leakage power, represents the static power dissipation in CMOS devices due to the leakage current.

Some work [24, 2, 25] has only focused on the term P_{load} since it is generally the dominant factor of the total power dissipation. But reducing the load of a gate decreases the slope of its outputs, thus increasing the internal power of its fanout gates. Because of this complex dependency between P_{load} and $P_{internal}$, we cannot ignore $P_{internal}$. Moreover, although the leakage power $P_{leakage}$ is usually smaller than the dynamic power, i.e., $P_{load}+P_{internal}$, it is a dominant factor in products such as pagers that remain idle for long periods of time.

The power model we use includes all the three terms given above[1]. Hence we take into consideration the complex relationship between power and delay optimization. The toggle rates can be found by either simulation, or probabilistic analysis [18]. We use toggle rates computed by the commercial probabilistic/simulation based power estimation tool DesignPower [7].

To each point v of a network is associated an *arrival time* $AT(v)$, which is the time at which the signal is propagated from the primary inputs to v, and a *required time* $RT(v)$, which is the time at which the signal must arrive to meet point-to-point delay constraints. The *slack* is defined as $RT(v) - AT(v)$. The set of points that has the minimal slack value constitute the *critical path* of the circuit, i.e., the slowest topological path. If the smallest slack is non-negative, the delay constraints are met. Unlike [2], we do not take false paths into account when calculating the slack because it is too expensive a process for large circuits. Therefore, the delay computed will be conservative. The reader is referred to [8, pp. 225–289] for more details on delay computation, path sensitization, and false paths.

The time needed for a signal to propagate from an input of a gate to an input of the next gate depends on the output load (the output capacitance seen at the output of the gate) and on the input transition time (the time needed by the input signal to achieve its transition). The reader is referred to [1, 27, 19, 16, 22, 17] for the presentation of some delay models. An extensive study of different input transition time sensitive delay models shows that a table lookup approach is more accurate than most of the multi-coefficient (linear, polynomial, or posynomial) approximations [17]. We will use such a table lookup based approach, which produces results within 3% of SPICE.

[1] We neglect power dissipation due to glitching, because measuring the glitching requires event-driven simulations, which are too costly to run each time one modifies the circuit.

3 Technology Mapping

Technology mapping consists of, given (1) a technology independent directed acyclic graph (DAG) of basic logic functions, (2) a specific technology library, and (3) a cost model, finding an implementation of the DAG using the library elements that optimizes the cost. An effective approach [12] amounts to breaking up the DAG into multiple fanout-free trees, optimizing them independently using dynamic programming, and performing some local optimizations at multiple fanout nodes to produce the resultant mapped DAG.

Dynamic programming finds the optimal mapping of a tree as long as the solution at a node can be constructed from the solutions of its sons. This is true when the cost function is the area or the number of gates. When the cost function is the delay, the solution is no longer optimal since the delay of gate g is not just a function of g and its sons in the tree, but is also influenced by the fanout of g. Therefore, a bottom-up dynamic programming approach is not guaranteed to be optimal, and more complex approaches must be used [20]. The same argument holds when the cost function is the power.

Technology mapping for low power has been proposed in [24, 25, 26, 15]. But it suffers from the same problem as technology mapping for delay, which is that the circuit is initially decomposed into some fixed structure, generally a NAND-INV structure, before being mapped. Therefore, the quality of the mapped circuit is highly dependent on this structure.

Recently, it has been shown in [14] that it is possible to simultaneously redecompose and map the circuit, i.e., previous decomposition decisions can be reconsidering during technology mapping. We use an extension of [14] to optimize for low power. As mentioned above, technology mapping is done one tree at a time. So in the following discussion, the input circuit is always a tree.

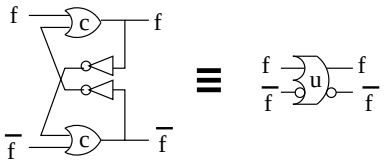

Figure 2: A unode

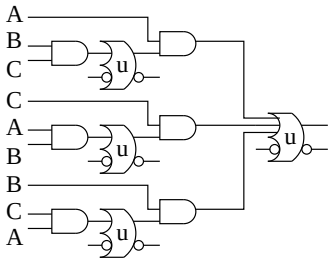

Figure 3: A mapping graph

3.1 Mapping Graphs

We use the *mapping graph* data structure introduced in [14] to reconsider the original logic decomposition while performing the technology mapping: in addition with the usual AND2/INV gates, the network is also described in terms of *unode* gates. A unode makes use of two *choice* gates (a choice gate is denoted as an OR gate with a 'c'). A choice gate is an artificial gate that is used to represent several implementations of the same logical function in the same graph. Thus the unode, as defined by Fig. 2, denotes several implementations of f and $\overline{f}$. A unode is used to capture several implementations of a function, and the phase assignment of it. In Fig. 3, a 3-way AND initially implemented as $(a \cdot (b \cdot c))$ can also be implemented as $(c \cdot (a \cdot b))$ or $(b \cdot (c \cdot a))$.

3.2 Power Updating

Technology mapping is done after the circuit has already been characterized for toggle rates. However technology remapping may introduce new nodes into the circuits, e.g., we might introduce the new node $(a \cdot b)$ or $(c \cdot a)$ in the example given in Fig. 3. The toggle rates of these new nodes are needed in order to evaluate their power implications. The toggle rates only depend on the logical functions of these new nodes. Thus we only need to extract the logical functions from the mapping graph.

To do so, let us consider a unode, and assume that there is one input in the choice gate which defines f (the case where only $\overline{f}$ is defined is symmetric). Then the unode is replaced with a simple circuit making use of an inverter, as shown in Fig. 4. After all unodes have been

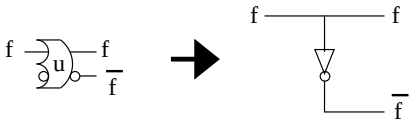

Figure 4: Unode replacement for simulation

translated this way, simulation or probabilistic based methods can be used to compute the toggle rates needed to explore the different mappings for the input circuit.

Let us discuss the probabilistic analysis. It is true that, at the very beginning, the remapping process takes a tree of gates as input. Hence, if the assumption of independence among the inputs of the tree is reasonable, then the computation of the toggle rates of the interior nodes is simple, since we have no reconvergent paths within the tree to destroy the independence assumption. However, if the tree contains XOR's or MUX'es, its expansion into a AND/INV graph yields a DAG, but not necessarily a tree. Thus, even if one assumes the independence of the inputs, a straightforward probabilistic propagation of toggle rates in a bottom up manner does not take into account the functional dependencies introduced by XOR's or MUX's gates. One can cope with this problem by using BDDs [18], but it can be too costly.

As an alternative, we use a direct probabilistic propagation that assumes the independence of the inputs of a tree, but that uses the following second-order method to limit the inaccuracies due to the signal reconvergences produced by XOR and MUX gates. Before computing the toggle rates for an AND2 gate, we look at its predecessor AND2 gates to detect whether this collection of gates constitutes a XOR2 or a MUX2. If so, then the toggle rate computation is performed using equations that are specific to XOR2's and MUX2's, and which take into account their internal reconvergences.

3.3 Dynamic Programming

Dynamic programming is performed on a mapping graph (which is a DAG) by considering the choice gates in topological order starting from the leaves. Our dynamic programming method has binning for output loads [20]; that is, for each choice gate, we keep track of multiple best solutions that are a function of the output load that the gate would drive.

When a library gate g can be mapped so that its output is the output of the current choice gate c, we need to evaluate the cost of the solution. To do this, we recursively build a solution network that uses g. The gates in this network that drive the inputs of g are chosen by looking up the previously computed best solutions for the input nodes of g that match the appropriate out-

put loads. We compute the cost of each such network driving the variety of output loads that g may drive and determine the best solution for each load.

In our system, building and evaluating the cost of the networks dominates the performance of the technology mapping. Thus it helps to prune the search tree of networks considered. For example, we assume that all the inputs of a multi-input AND gate are symmetric, even though we use more exact (and *non*-symmetric) models when determining the cost. This leads to some loss in optimality in exchange for huge CPU savings. This loss in optimality is compensated for in the post-mapping optimization described in Section 4, which does examine pin swapping.

3.4 Objective function

Dynamic programming uses an objective function to select one implementation over other ones. Ideally, instead of just remembering the best solutions at a choice gate as a function of output load, we should remember them as a function of both output load and some delay related cost, such as arrival time (e.g., [24, 25, 26, 15]). However it is impossible to determine exactly the arrival time at a node in the subject tree since it depends on the choice of the gates at the fanin (input slopes) and fanout (output load) of this node. Thus the best solutions depend also on the input slopes, but remembering this dependency is too costly.

Since coping with the exact delay constraints is impossible, the technology mapping only remembers the best solutions at a choice gate as a function of output load. Also, it concentrates on picking up good gains from trees without tight time constraints. The objective function is such that slack violations are discouraged the most, and then power usage is discouraged. This objective function gives rise to three different behaviors:

- $Slack \ll 0$. If a tree is violating its timing constraints, the technology mapper concentrates on optimizing the timing of the circuit.
- $Slack \gg 0$. If a tree is in no danger of violating the slack, then the technology mapper concentrates on optimizing the power.
- $Slack \approx 0$. Near the leaves of the tree, the technology mapper focuses on minimizing the power, and near the root of the tree the technology mapper concentrates on minimizing the slack. The intuitive explanation of this heuristic is twofold. First, the toggle rates are in general higher at the leaves than the root, giving more opportunities to lower the dynamic power. Second, speeding up gates close to the root is likely to speed up the whole tree. Thus, spending power to gain speed is less expensive close to the root.

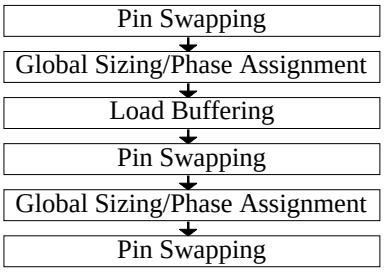

Figure 5: Post-Mapping Optimization Sequence.

4 Post-Mapping Optimization

Technology dependent optimization for low power has focused primarily on gate sizing. It consists of replacing the gates in the mapped circuit with logically equivalent gates so that a user defined cost function is optimized. Early work on gate sizing can be found in [21, 13]. More recent work has focused on optimizing area under a delay constraint [9, 10] and on optimizing power [4].

We kept four local technology dependent optimizations for low power: gate sizing, phase assignment, buffering, and pin swapping. All of them are local optimizations that do not need costly resimulation to update the toggle rates (indeed, 1-complementation is sufficient). These post-mapping optimizations are performed in the sequence shown in Fig. 5. This sequence has been determined experimentally by comparing different scripts over large benchmarks. It achieves good power savings within a reasonable CPU time.

Gate sizing consists of replacing a gate with a logically equivalent one, whose physical parameters decrease the power consumption without violating any delay constraint. Phase assignment takes into consideration that a function can be represented multiple ways using DeMorgan's rules. For example, a NAND function can be mapped to either a NAND gate or a NOT-OR combination of gates. Some phase assignment is implicitly performed during technology mapping, but the technology mapping algorithm described in Section 3 does not look at the consequences of different phase assignments at the multiple fanout nodes.

Some of the previous work in resizing for low power has focused on sizing after elimination of false paths [2], and on using linear programming methods to pick optimal gate sizes to reduce power, area, and or delay [23]. Although the algorithm presented in [2] works with accurate delay numbers by eliminating false paths, it is also forced to resize gates in a greedy manner, from primary outputs to primary inputs. The solution space is highly non-linear and non-convex, and the greedy method has a tendency to get stuck in local minima. The method proposed in [23] produces a solution assuming that the gates can be continuously sized. But

circuit		power				slack					area				CPU			
	g	t	po	tpo	pot	i	t	po	tpo	pot	t	po	tpo	pot	t	po	tpo	pot
a	490	0.97	0.96	0.95	0.95	14.6	14.3	14.6	14.6	14.3	1.01	0.93	0.94	0.97	261	195	488	439
b	1671	0.90	0.99	0.89	0.90	−0.12	0.12	0.00	0.00	0.03	1.02	0.99	1.00	1.03	1801	2500	5403	4097
c	175	0.93	0.91	0.90	0.89	−0.23	0.00	0.01	0.04	0.07	1.15	1.01	1.12	1.12	141	123	220	239
d	139	0.83	0.99	0.82	0.83	71.2	69.3	71.2	70.2	69.3	1.17	1.03	1.17	1.20	77	32	108	101
e	139	0.79	0.78	0.79	0.77	37.9	38.0	38.0	38.0	38.0	1.12	1.00	1.12	1.10	110	34	125	143
f	259	0.85	0.76	0.73	0.72	−0.35	0.01	0.02	0.04	0.00	1.11	0.98	1.07	1.04	198	141	344	344
g	560	0.80	0.92	0.73	0.71	−0.99	0.00	0.01	0.01	0.01	1.21	1.05	1.09	1.16	519	529	864	959
h	684	0.77	0.76	0.70	0.69	0.10	0.04	0.14	0.00	0.01	1.11	1.14	1.12	1.11	380	1954	1088	2137
i	841	0.41	0.85	0.39	0.24	23.1	24.0	24.5	24.6	22.4	1.35	1.18	1.19	1.34	1125	3448	2980	4367

This table gives the number of gates **g** of the initial circuits. The **power** and **area** are normalized w.r.t to their values for the initial circuit. The table gives the **slack**, **power**, and **area** of the initial mapped circuit (**i**); after technology mapping alone (**t**); after post-mapping optimization alone (**po**); after technology mapping first, then post-mapping optimization (**tpo**); after post-mapping optimization first, then remapping (**pot**). The **CPU** time is given in seconds on a 60 MHz SuperSparc (85.4 SpecInt).

Table 1: Some experimental results on combinatorial circuits.

gate sizing is a discrete optimization problem, and a continuous solution cannot always be easily projected on a discrete space.

The GS algorithm presented in [5, 6], preliminarily focused on gate sizing, has been extended to incorporate phase assignment. This algorithm optimizes the critical paths and then *relaxes* the circuit using a benefit/penalty function that considers how much power can be saved (by sizing or phase assignment) in regards of the increase or decrease of the delay (the delay constraints are always enforced during the relaxation). The reader is referred to [5, 6] for a description of the GS algorithm. Although technology remapping for low power does a certain amount of gate sizing, it does not have the capability of evaluating the global cost of a decision since it only optimizes a tree at a time. The GS algorithm is more effective at evaluating the global cost of such local transformations.

Buffer insertion takes care of nets with large loads. Inserting one or two buffers can help reduce the delay and/or the power dissipation. It is the case when there is a large load on a gate due to a large fanout or a large wire load. A gate driving a large load increases the delay on this path, and it also increases the power since the signal being propagated to the next stage has a larger rise or fall time, and hence a larger internal power dissipation. The buffer insertion algorithm works in a greedy manner, and buffers are inserted only if they decrease the power and do not violate timing constraints.

When a logical gate involves symmetrical input pins (e.g. AND gate), the input wires can be swapped in such a way that the power can be decreased. This can happen by decreasing the load of a net with a high toggle rate. Experimental results showed that this optimization did not result in dramatic changes in power dissipation, but did prove to be useful for getting out of locally minimal solutions.

5 Experimental Results

We took 92 combinatorial circuits as a benchmark set. These circuits have between 50 and 1700 gates. The starting point of our experiment is a circuit optimized for area under delay constraints. Table 1 summarizes some results.

Since the power optimization is done after the circuits have already been optimized for area without regard for power, any power optimization is likely to increase the area. But it happens that power saving can be achieved without any area penalty, e.g., example *a*.

On average, the technology mapping run alone yielded a 18% power reduction, the post-mapping optimization run alone yielded a 8% power reduction. Applying remapping first, then post-mapping optimization, yields a 21% power saving. Performing post-mapping optimization first, then remapping, yields a 24% power saving. Although the optimization flow is order sensitive, experimental results show that the two optimization steps presented in this paper do not overlap much.

6 Conclusion

This paper has described a power optimization flow using remapping and technology dependent post-mapping optimization. The remapping incorporates the latest improvements in technology mapping with a power estimation of the subject graphs. The post-mapping optimization makes use of gate sizing, phase assignment,

buffering, and pin swapping. This optimization strategy achieves a 24% average power saving.

Acknowledgments

The authors would like to thank Srilatha Manne, who helped in obtaining the experimental results.

References

[1] D. Auvergne, N. Azemard, D. Deschacht, M. Robert, "Input Waveform Slope Effects in CMOS Delays", *IEEE J. on Solid-State Cir.*, **25**-6, pp. 1588–1590, Dec. 1990.

[2] R. Iris Bahar, G. D. Hachtel, E. Macii, F. Somenzi,"A Symbolic Method to Reduce Power Consumption of Circuits Containing False Paths", *IEEE Trans. on CAD*, pp. 368–371, Nov. 1994.

[3] R. Iris Bahar,F. Somenzi,"Boolean Techniques for Low Power Driven Re-Synthesis", *IEEE Trans. on CAD*, pp. 428–432, Nov. 1993.

[4] M. Borah, R. M. Owens, M. J. Irwin, "Transistor Sizing for Minimizing Power Consumption of CMOS Circuits under Delay Constraint", Proc. of *Int'l Symp. on Low Power Design*, Monterey CA, pp. 167–172, April 1995.

[5] O. Coudert, "Gate Sizing: a General Purpose Optimization Approach", Proc. of *ED&TC'96*, Paris, France, March 1996.

[6] O. Coudert, R. Haddad, S. Manne, "New Algorithms for Gate Sizing: A Comparative Study", Proc. of *33nd DAC*, Las Vegas NV, June 1996.

[7] DesignPower, *Starter Kit*, Synopsys.

[8] S. Devadas, A. Ghosh, K. Keutzer, *Logic Synthesis*, McGraw-Hill, 1994.

[9] J. P. Fishburn, A. E. Dunlop, "TILOS: a Posynomial Programming Approach to Transistor Sizing", Proc. of *ICCAD'85*, Santa Clara CA, pp. 326–328, Nov. 1985.

[10] J. P. Fishburn, "LATTIS: an Iterative Speedup Heuristics for Mapped Logic", Proc. of *29th DAC*, Anaheim CA, pp. 488–491, June 1992.

[11] S. Iman, M. Pedram, "Logic Extraction and Factorization for Low Power", Proc. of *32nd DAC*, pp. 248–253, San Francisco CA, June 1995.

[12] K. Keutzer, "DAGON: Technology binding and local optimization by DAG matching", Proc. of *24th DAC*, pp. 341–347, June 1987.

[13] C. M. Lee, H. Soukup, "An Algorithm for CMOS Timing and Area Optimization", *IEEE J. of Solid-State Cir.*, **19**-5, pp. 781–787, Oct. 1984.

[14] E. Lehman, Y. Watanabe, J. Grodstein, H. Harkness, "Logic Decomposition during Technology Mapping, Proc. of *ICCAD'95*, San Jose CA, pp. 264–271, Nov. 1995.

[15] B. Lin, H. De Man, "Low-power driven technology mapping under timing constraints", Proc. of *ICCD'93*, pp. 421–427, Oct. 1993.

[16] Motorola HDC Series Design Manual.

[17] A. Martinez, "Automated Library Characterization and Timing Model Accuracy Issues when Interfacing to Different CAD Tools", Hewlett-Packard Company, Santa Clara CA.

[18] F. Najm, "Transition Density, A Stochastic Measure of Activity in Digital Circuits", Proc. of *28th DAC*, San Francisco CA, pp. 644–649, June 1991.

[19] R. W. Phelps, "Advanced Library Characterization for High Performance ASIC", Texas Instruments, Inc., Dallas TX.

[20] R. Rudell, *Logic Synthesis for VLSI Design*, PhD thesis, University of California, Berkeley, 1989.

[21] A. E. Ruehli, P. K. Wolff, G. Goertzel, "Analytical Power/Timing Optimization Technique for Digital System", Proc. of *14th DAC*, pp. 142–146, June 1977.

[22] T. Sakurai, A. R. Newton, "Delay Analysis of Series-Connected MOSFET Circuits", *IEEE J. of Solid-State Cir.*, **26**-2, pp. 122–131, Feb. 1991.

[23] Y. Tamiya, Y. Matsunaga, M. Fujita, "LP based Cell Selection with Constraints of Timing, Area, and Power Consumption", Proc. of *ICCAD'94*, San Jose CA, pp. 378–381, Nov. 1984.

[24] V. Tiwari, P. Ashar, S. Malik, "Technology Mapping for Low Power", Proc. of *30th DAC*, Dallas TX, pp. 74–79, June 1993.

[25] C-Y. Tsui, M. Pedram, A. Despain, "Technology Decomposition and Mapping Targeting Low Power Dissipation", Proc. of *30th DAC*, pp. 68–73, June 1993.

[26] C-Y. Tsui, M. Pedram, A. Despain, "Power Efficient Technology Decomposition and Mapping under an Extended Power Consumption Model", *IEEE Trans. on CAD*, **13**-9, Sept. 1994.

[27] G. Zewi, U. Barkai, Z. Becker, J. Ben-Simon, E. Kadar, "An Accurate Slope-Dependent Delay Model", Proc. of *TAU'90*, Haifa, Israel, 1990.