# Integrating Logic-level Power Management Techniques

L. Benini # G. De Micheli # E. Macii ‡ M. Poncino ‡ R. Scarsi ‡

# Stanford University
Computer Systems Laboratory
Stanford, CA 94305

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

**Abstract— Pre-computation and gated clocks are well-known design solutions for reducing power dissipation in sequential circuits. Although it is clear that the two techniques are based on similar principles, no definite answer has been given on whether they are equivalent formulations or two alternative approaches. In this paper, we address this open question. We show that pre-computation and gated clocks are *not* equivalent, and that they can be concurrently applied. We describe an architecture for merging the two techniques, and we analyze its applicability to power optimization of different classes of sequential circuits.**

## I. Introduction

Complex digital systems usually contain portions of logic which are not performing useful computations at each clock cycle. Think, for example, of arithmetic units and register files within a microprocessor or, more simply, to state registers of an ordinary synchronous sequential circuit. The idea, known since a long time in the community of IC designers, is to shut down the logic which is not in use during some particular clock cycles, with the objective of limiting power consumption. In fact, preventing gates from performing useless transitions causes a decrease in the overall switching activity of the circuit.

In the past, the implementation of power-down strategies was carried out manually by the designers. With the increased complexity of modern VLSI systems, the difficulty of performing this task has grown as well. This has motivated the development of automatic techniques for inserting power management mechanisms into the designs starting directly from the logic/RT-level descriptions of the systems being synthesized.

Two successful solutions to the problem of reducing the power dissipated by an IC through logic shut-down have recently become available. The first one, identified as *pre-computation*, is due to Alidina and co-workers [1, 2, 3, 4]. The method relies on the idea of duplicating part of the logic with the purpose of pre-computing the circuit output values one clock cycle before they are required, and then use these values to reduce the total amount of switching in the circuit during the next clock cycle. In fact, knowing the output values one clock cycle in advance allows the original logic to be turned off during the next time frame, thus eliminating any charging and discharging of the internal capacitances.

The second alternative, called *gated clocks*, has been proposed by Benini *et al.* in [5, 6, 7]. The fundamental idea of the method is to selectively stop the clock, and thus force the combinational logic of the circuit to make no transition, whenever the computation to be performed at the next clock cycle is useless. In other words, the clock signal is disabled in correspondence to the idle conditions of the synchronous network. For reactive circuits, such as most of the logic controllers employed in telecommunication systems, the number of clock cycles in which the design is stuck into a wait state (for example, monitoring for a given event to occur) is usually large. Therefore, avoiding the power waste corresponding to the states in which the circuit is waiting may provide a significant improvement in the power figure of the overall system.

The use of pre-computation-based architectures has shown to be particularly effective for the optimization of pipelined circuits, whose structure consists of a combinational logic block with latched inputs and outputs. Concerning synchronous sequential circuits, that is, circuits with latched state feedback whose behavior is usually modeled through a finite state machine, the theoretical applicability of the method has not been supported by adequate experimental data. Gated clocks, on the other hand, have proved to be particularly useful for power optimization of circuits with state feedback. To the best of our knowledge, however, no results have been reported on the application of the technique to pipelined circuits.

Although the two techniques have been applied to different classes of circuits, they have striking similarities. It may be possible to think that pre-computation and gated clocks are indeed the same optimization technique applied to different types of circuits.

This paper brings three contributions. First, we show that pre-computation and gated clocks are *not* equivalent,

and they are *not* mutually exclusive. Second, we have collected new data on the applicability of the two shut-down strategies discussed above, and the outcome of the experiments, presented here, has clearly indicated that the use of gated clocks is beneficial also for pipelined circuits. On the contrary, power management based on pre-computation has produced relatively disappointing results in the case of sequential circuits. Third, we propose an integrated power-down architecture which tries to exploit the benefits of both pre-computation and gated clocks, and we highlight the results we have obtained on standard benchmark circuits through the application of such architecture.

## II. A Comparative Analysis

In this section we summarize the main characteristics of the pre-computation and the gated clocks power-down strategies; in addition, we present a comparative analysis of the two methods.

### A. Pre-Computation

The pre-computation approach has been proposed with the objective of optimizing pipelined designs, that is, circuits having the structure depicted in Figure 1. The combinational block $A$ implements a $N$-input Boolean function, $f$, and it has the I/O pins connected to registers $R_1$ and $R_2$. Notice that, for the sake of simplicity, but without loss of generality, in the following we assume $f$ to be a single-output function.

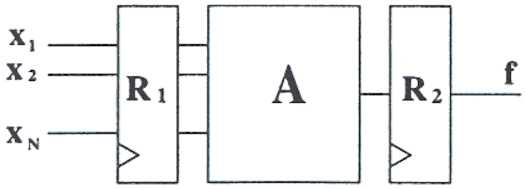


Fig. 1. Pipelined Circuit.

The method is based on the idea of duplicating part of the original circuitry with the purpose of pre-computing the circuit output values one clock cycle before they must appear, and then use these values to decrease the total amount of switching within the logic during the next clock cycle. In fact, knowing the output values one clock cycle in advance allows the original logic to be turned off during the next time frame, thus eliminating any charging and discharging of the internal capacitances.

Clearly, the size of the pre-computation logic must be kept under control, since its contribution to the total power balance may offset the savings achieved by blocking the switching inside the original circuit. To avoid this problem, it is possible to select only a subset of the input conditions for which the output values are pre-calculated. However, the task of selecting such subset, which basically corresponds to synthesizing the pre-computation logic, is not trivial, the ideal target being a very compact circuit whose corresponding Boolean function covers a large part of the original function. Moreover, the synthesized pre-computation logic must meet other design constraints, like area and speed.

Two pre-computation architectures have been introduced. The schematic of the first one is reported in Figure 2.

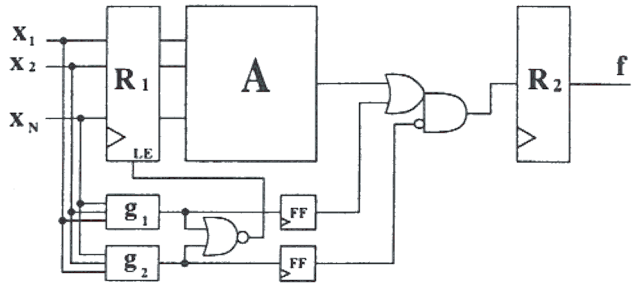


Fig. 2. First Pre-Computation Architecture.

The distinguishing features of the architecture are the two $N$-input, single-output predictor functions, $g_1$ and $g_2$, whose behavior is required to satisfy the following constraints:

$$g_1 = 1 \Rightarrow f = 1 \quad \text{and} \quad g_2 = 1 \Rightarrow f = 0 \qquad (1$$

If, at the present clock cycle, either $g_1$ or $g_2$ evaluates to 1, the register load enable signal $LE$ goes to 0, and the inputs to block $A$ at the next clock cycle are forced to retain the current values. Hence, the number of gate output transitions inside block $A$ gets zeroed, while the correct output value for the next time frame is provided by the two registers located on the outputs of $g_1$ and $g_2$. Notice that, in this first pre-computation architecture, functions $g_1$ and $g_2$ may depend on all the inputs to block $A$.

Obviously, the choice of the predictor functions is a key issue. The ideal target would be to have $g_1 \equiv f$ and $g_2 \equiv f'$. From the practical stand-point, however, this solution would not give any advantage in terms of power consumption over the original circuit, since it would require the complete duplication of block $A$, and thus it would provide the same number of switchings as before, but with an area twice as large as the original network. Consequently, the objective to be reached is the realization of two functions for which the probability of their logical sum (i.e., $g_1 + g_2$) to be 1 is as high as possible, but for which the area penalty due to their implementations is very limited.

One way of guaranteeing functions $g_1$ and $g_2$ to be much less complex than function $f$ is to enforce the dependency of the two predictor functions on a limited number of inputs if compared to $f$. We have then the second pre-computation architecture, shown in Figure 3, which differs from the first one in two main aspects: First, the set of inputs to $A$ is partitioned into two subsets, and only one of them feeds the predictor functions $g_1$ and $g_2$. Second,

We can divide the states of a Mealy-type FSM into two classes [6]. States where self-loops are not idle conditions (unless taken twice), are called *Mealy-states*, while states where self-loops are idle conditions are called *Moore-states*. For Mealy-states it is not possible to stop the clock of the circuit by just observing the state and input lines. It is then possible to apply an algorithm that operates on the state transition graph (STG) of the FSM to transform Mealy-states into Moore-states, thus allowing the exploitation of more self-loops as idle conditions where the clock can be stopped. Since it is generally computationally infeasible to extract the STG representation for large sequential circuits, the transformation from Mealy-states to Moore-states is not applicable and we must restrict ourselves to the Moore-states of the Mealy FSMs.

The activation function $F_a(x, s)$ is then given by the union of all self-loops of Moore-states ($x$ and $s$ are the input and state variables). The set of all self-loops in the FSM includes $F_a$, since it contains also the self-loops of Mealy-states. Explicit and implicit (i.e., BDD-based) algorithms have been proposed to automatically determine $F_a(x, s)$ from either the STG or the gate-level descriptions of the original circuit.

Obviously, if a circuit is an implementation of a Mealy FSM with no Moore-states, the activation function will be empty, thus implying no chance of stopping the clock at any time. Fortunately, it may still be possible to determine some stopping conditions associated to Mealy-states. Intuitively, self-loops on such states are not idle conditions because it cannot be guaranteed that output transitions will not be required, even if the next state does not change. However, if the outputs of the sequential circuit are taken as inputs of the activation function as well as the state and primary inputs, the problem can be solved. The gated clock architecture can be modified as shown in Figure 6.

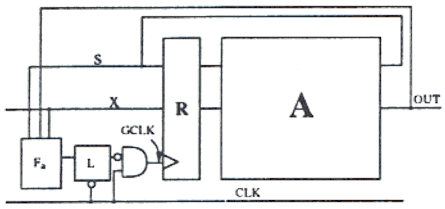


Fig. 6. Modified Gated Clock Architecture.

If all outputs are taken as inputs of the activation function, all self-loops can be exploited to stop the clock [7]. It may be observed that, since the number of outputs in a sequential circuit is often very large, the size of the activation logic may increase too much. However, it may be the case that we do not need to use all the outputs as inputs of $F_a$ [7].

There is clearly a trade-off between the number of additional self-loops that can be considered in the activation function by including one or more outputs to its support and its size (and power dissipation). The selection of an optimal subset of outputs for inclusion in the support of the activation function is usually performed heuristically [7].

After the Boolean expression of the activation function is determined, it must be synthesized and optimized. The problem to be faced here is that the size (and the power dissipation) of the combinational block corresponding to $F_a$ may be unacceptably large. The idea is then to synthesize a simplified function, $f_a$, contained into the original $F_a$, which dissipates the minimum possible power, and stops the clock with maximum efficiency.

Formally speaking, the problem of computing the simplified activation function can be formulated as follows. Given a Boolean function, $F_a$, find a new function $f_a \subseteq F_a$ such that its probability, $P(f_a)$, satisfies the inequality $P(f_a) \geq \alpha P(F_a)$ (where $0 \leq \alpha \leq 1$ is a user-specified scale factor and $\alpha P(F_a)$ represents the probability constraint) and the number of literals in a two-level implementation of function $f_a$ is minimum.

An effective approach to the simplification of the activation function is based on symbolic Boolean functions manipulation. The algorithm works as follows. First, a pseudo-Boolean function, $P_{F_a}$, is constructed, which implicitly represents the probability of the minterms in the ON-set of $F_a$. Then, some of the minterms of $F_a$ are removed until a given cost criterion breaks the loop. Clearly, both the minterm removal and the stopping condition must be guided by a combination of the size improvement in the implementation of $F_a$ and the probability decrease of the ON-set of $F_a$.

### C. Extending Pre-Computation to Sequential Circuits

In [3], the authors have made the claim that the applicability of the pre-computation architectures of Figures 2 and 3 is not restricted to the case of pipelined circuits but, instead, it can be easily extended to designs having an arbitrary structure. In particular, they have shown in detail how the second pre-computation architecture can be adapted to synchronous sequential circuits with traditional topology (see Figure 4). The proposed modifications are depicted in Figure 7.

In spite of the nice theoretical formulation of the solution, no experimental evidence of its practical usefulness has been presented in the literature. We have thus conducted a deeper investigation on this subject. In Sections III and IV we report considerations and experimental data on the outcome of our analysis.

### D. Extending Gated Clocks to Pipelined Circuits

Gated clocks can obviously be applied to pipelined circuits. In fact, this kind of designs differentiate from the finite state machine model of Figure 4 only by the absence of the state feedback. Hence, the only constraint for the gated clock architecture to work properly is that all output lines must be considered simultaneously. Also for this

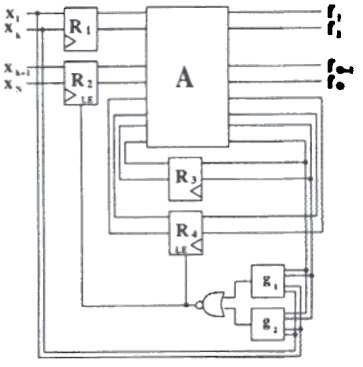


Fig. 7. Second Pre-Computation Architecture for a Synchronous Sequential Circuit.

extension, a discussion and some experimental results are provided in Sections IIIand IV..

### E. Comparison and Discussion

Pre-computation and gated clocks are both based on the concept of synthesizing a logic function that stops the clock of a sequential circuit in a subset of all possible input conditions. Although the clock stopping circuitry for pre-computation (i.e., enable signals) and gated clocks (i.e., latch-based clock shut-down) appear to be different, it is easy to realize that the difference is only a matter of implementation style. Moreover, we have just shown in the previous sections that the two techniques can be applied to the same classes of circuits.

These observations may lead to the conclusion that pre-computation and gated clocks are equivalent. In fact, they are not. Pre-computation detects clock-stopping opportunities based on lack of observability. If for some input conditions a large number of inputs is not observable at the outputs, there is no need to clock the flip-flops associated to the un-observable inputs. The purpose of the pre-computation logic in the first architecture is precisely to detect when the pre-calculated inputs are unobservable.

On the contrary, gated clocks are based on the intrinsic memory-retaining property of CMOS circuitry. If there are conditions for which we can pre-determine that the outputs are not going to change, there is no need to clock the circuit, because the input activity does not propagate to the outputs. Moreover, we can stop the clock indefinitely, as long as the outputs do not need to be updated.

From these observations, we may also conclude that there are circuits for which pre-computation is completely ineffective, namely those for which a large fraction of the inputs is always observable. On the other hand, gated clocks are not effective for systems where at least one output changes every clock cycle. The non-equivalence of gated clocks and pre-computation, and thus the possibility of merging the two techniques, is investigated in the next sections.

## III. Applying Pre-Computation and Gated Clocks

In this section, we describe an architecture that integrates the pre-computation and the gated clock mechanisms into a unique shut-down scheme. The motivation for this solution is that the conditions that enable the power management of some or all circuit (primary and/or present-state) inputs are not necessarily the same for the two approaches, and thus some advantages may result from the simultaneous use of the two techniques. Figure 8 shows the integrated architecture for the case of pipelined designs.

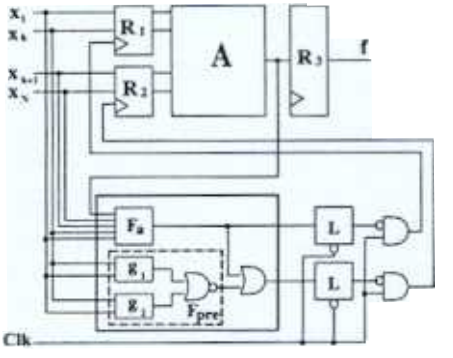


Fig. 8. Integrated Architecture.

The solid box identifies the global shut-down logic: Block $F_a$ implements the activation function of the gated clock scheme, and block $F_{pre}$ implements the pre-computation conditions, that is, $(g_1 + g_2)'$.

From Figure 8 it seems clear that, whenever the conditions for which $F_{pre}$ is 1 are a subset of those for which $F_a$ is 1, the integrated architecture provides no benefit with respect to the pure gated clock scheme. However, if some of the pre-computation conditions are not included in those of the gated clock, joining the two networks always results in a potential power improvement. As a matter of fact, this situation indicates that a subset of the circuit inputs can be stopped more often if the integrated approach is taken.

For better area optimization, functions $F_a$ and $F_{pre}$ should be synthesized and optimized concurrently as a unique, two-output circuit. This is because the amount of logic shared among the two blocks may be substantial. Notice that, in the proposed scheme, we did not resort to registers with an enable input. Rather, we have used the output of the pre-computation circuitry to gate the clock of registers $R_2$. The advantage of this scheme is that it avoids useless transitions on the gated clock distribution network when $F_{pre}$ takes on the value 1. Consequently, further power savings can be achieved.

It is important to observe that, even though, in principle, the combined architecture may be applicable to synchronous sequential networks, the reduced efficiency of the pre-computation paradigm for this type of circuits makes

in addition to the combinational blocks implementing $g_1$ and $g_2$, only one extra NOR gate is required, as opposed to the first architecture, where two further registers, plus a NOR gate and an OR-AND gate were necessary.

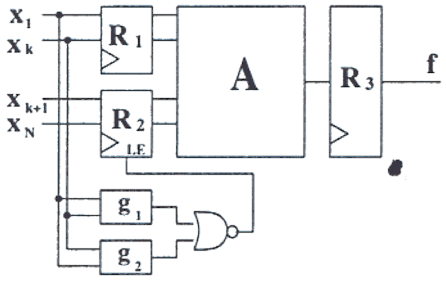


Fig. 3. Second Pre-Computation Architecture.

Similarly to the first architecture, the predictor functions $g_1$ and $g_2$ must satisfy the constraints posed by Equations 1. When either $g_1$ or $g_2$ evaluates to 1, the outputs of register $R_2$ are not allowed to make any switching at the next clock cycle. However, since the outputs of register $R_1$ can be updated (the load enable signal, $LE$, does not feed register $R_1$), some activity is allowed within block $A$, and function $f$ will evaluate to the correct logic value. In contrast with the first pre-computation architecture, the second solution never calculates and stores in advance the output values for the subsequent clock cycle. The circuits implementing the predictor functions $g_1$ and $g_2$ only determine whether the selected subset of inputs to block $A$ can be frozen, thus possibly reducing the total number of transitions inside the combinational logic. Therefore, we can say that the operating mode of this structure is somehow similar to the one of the gated clock architecture that will be covered in detail in the next section.

Two problems still have to be addressed. First, how to compute the subset $x_1, \ldots, x_k$ of inputs to block $A$ to be fed to the predictor functions. Second, how to synthesize functions $g_1$ and $g_2$ in such a way that Equations 1 are satisfied and that the probability of $g_1 + g_2 = 1$ is high.

Exact and heuristic procedures have been devised for the purpose of computing the $x_1, \ldots, x_k$ inputs which are most convenient for this particular application. The common assumption to all these methods is that such subset is the one for which the probability of $g_1 + g_2 = 1$ gets the maximum value for a fixed $k$. Since, by definition, $g_1$ and $g_2$ cannot be 1 for the same input vector, the maximum probability of $g_1 + g_2$ to be one occurs only when the probabilities of $g_1 = 1$ and $g_2 = 1$ simultaneously get their maximum values. This cost function is used to recursively explore the (possibly pruned, when the heuristic methods are used) search space of all possible subsets of cardinality $k$.

After the subset $x_1, \ldots, x_k$ has been determined, functions $g_1$ and $g_2$, which only depend on $x_1, \ldots, x_k$ and which have the maximum probability of being 1, can be computed as:

$$g_1 = \forall_{x_{k+1},\ldots,x_N} f \quad \text{and} \quad g_2 = \forall_{x_{k+1},\ldots,x_N} f' \qquad (2)$$

where $\forall_{x_i} f = f_{x_i} \cdot f_{x_i'}$ is the universal quantification of $f$ with respect to variable $x_i$, and $f_{x_i} = f(x_i = 1)$ and $f_{x_i'} = f(x_i = 0)$ are the positive and the negative cofactors of $f$ with respect to $x_i$.

### B. Gated Clocks

Gated clocks [6, 7] are an effective alternative to the pre-computation-based power-down strategy described in the previous section. They have been proposed as a power-management strategy for synchronous sequential circuits having the structure of Figure 4.

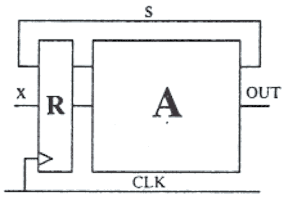


Fig. 4. Synchronous Sequential Circuit.

The fundamental idea of the method is to selectively stop the clock, and thus avoid transitions within the combinational logic, anytime the computation to be performed at the next clock cycle is useless. In other words, the clock signal is disabled in correspondence to the idle conditions of the synchronous network.

The gated clock version of the circuit of Figure 4 is depicted in Figure 5. Signal $F_a$, called *activation function*, selectively stops the local clock (i.e., $F_a = 1$) when the circuit does not perform state or output transitions.

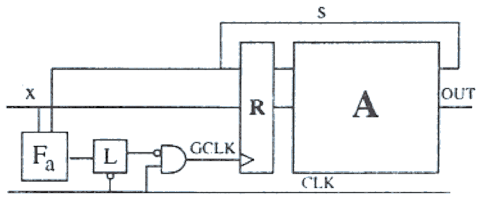


Fig. 5. Synchronous Sequential Circuit with Gated Clock.

Given the gate-level description of the original circuit, we need to identify the conditions under which the clock may be stopped. It is known that the behavior of a synchronous sequential circuit can be modeled by a finite state machine (FSM). Determining the idle conditions is then a simple task for circuits whose FSM models are of Moore-type. In fact, when the present state and the inputs are such that the next state does not change, the Moore FSM is idle [6]. Unfortunately, this property does not hold for Mealy FSMs.

| Circuit | PI | PO | Original Power | Gated Clocks | | Pre-Computation | | Integr. Arch. | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | Power | Savings | Power | Savings | Power | Savings |
| 9sym | 9 | 1 | 123 | 62 | 49% | 134 | −9% | 67 | 45% |
| apex2 | 39 | 3 | 286 | 286 | 0% | 114 | 60% | 114 | 60% |
| cm138 | 6 | 8 | 35 | 13 | 63% | 29 | 17% | 25 | 28% |
| cm150 | 21 | 1 | 94 | 60 | 36% | 74 | 21% | 48 | 49% |
| cmb | 16 | 4 | 66 | 15 | 77% | 45 | 32% | 14 | 79% |
| comp | 32 | 3 | 144 | 116 | 19% | 68 | 53% | 75 | 48% |
| cordic | 23 | 2 | 111 | 38 | 66% | 81 | 27% | 44 | 60% |
| mux | 21 | 1 | 99 | 61 | 38% | 70 | 29% | 60 | 39% |
| sao2 | 10 | 4 | 93 | 58 | 37% | 48 | 48% | 37 | 60% |

TABLE I
RESULTS: PIPELINED CIRCUITS.

| Circuit | PI | PO | FF | Original Power | Gated Clocks | | Pre-Computation | | Integr. Arch. | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | Power | Savings | Power | Savings | Power | Savings |
| s208.1 | 10 | 1 | 8 | 75 | 49 | 34% | 71 | 5% | 52 | 31% |
| s298 | 3 | 6 | 14 | 89 | 72 | 19% | 120 | −35% | 112 | −26% |
| s386 | 7 | 7 | 6 | 63 | 58 | 8% | 70 | −11% | 65 | −3% |
| s400 | 3 | 6 | 21 | 90 | 63 | 30% | 77 | 14% | 71 | 21% |
| s420.1 | 10 | 1 | 16 | 106 | 66 | 36% | 97 | 8% | 72 | 32% |
| s444 | 3 | 6 | 21 | 101 | 76 | 25% | 86 | 15% | 77 | 24% |
| s510 | 19 | 7 | 6 | 95 | 81 | 15% | 99 | −4% | 85 | 11% |
| s526 | 3 | 6 | 21 | 119 | 114 | 4% | 147 | −23% | 136 | −14% |
| **minmax4** | **3** | **5** | **15** | **85** | **40** | **52%** | **66** | **22%** | **34** | **60%** |

TABLE II
RESULTS: SYNCHRONOUS SEQUENTIAL CIRCUITS.

this solution of limited practical interest, as shown by the data presented in Section IV..

## IV. RESULTS

In this section, we present the results we have obtained by applying the three power-down strategies (i.e., second pre-computation architecture, gated clocks, and integrated architecture) discussed in this paper on a set of standard benchmarks. Such results have two consequences. First, they experimentally confirm the indications given by the analysis of Section III. Pre-computation and gated clocks are both efficient for pipelined circuits, but the former fails for most of the sequential examples. Second, they demonstrate the usefulness of the integrated architecture when the target of the optimization are pipelined designs.

Tables I and II report the experimental data. Pipelined circuits have been constructed by adding input and output latches to some combinational circuits taken from the `Mcnc'91` [8] suite. In particular, we have chosen the examples for which the best power savings have been obtained in [4]. Synchronous designs, on the other hand, are taken from the `Iscas'89` set [9], and are the same as the ones we have used for the experiments in [7].

The selected circuits have been initially optimized using the standard SIS `script.rugged` [10], and mapped for delay with the SIS command `map -n 1 -AFG`. The optimized netlists have been used as the starting point for the experiments. The technology library used for mapping includes buffers and inverters with three different strengths, and NAND/NOR gates with up to four inputs. The power estimates, measured in $\mu W$, have been calculated using the IRSIM-CAP simulator [11].

The use of gated clocks on the pipelined circuits has shown to be more effective than pre-computation in six of the nine examples we have considered; in the remaining cases, pre-computation has worked better, with a peak performance on benchmark `apex2`, where gated clocks have not given any advantage. From an absolute point of view, the integrated architecture has produced the best savings in four cases.

Concerning sequential designs, the application of pre-computation has always given worse results than those obtained with the gated clocks alone; in some cases, it has even resulted in a higher power consumption than the original circuit. The reason for this poor behavior lies in that the pre-computation function never attempts to stop the present-state inputs, which represent the majority of the inputs to the combinational logic for sequential circuits with a realistic number of memory elements. As a consequence, also the results obtained with the integrated solution are not satisfactory, except for the case of benchmark `minmax4` (and this was somehow expected, since `minmax4` contains quite many comparators).

## V. Conclusions

It is well known that pre-computation and gated clocks enable substantial power savings through logic shut-down. However, the way they achieve this goal is quite different. In this paper, we have investigated the conditions under which the use of each method is more advantageous from the power stand-point. We have also proposed an integrated architecture which best exploits the characteristics of the two techniques. In fact, it may well be the case that a combination of the two approaches results in a better global optimization. We have presented experimental results showing that the integrated architecture is, for some pipelined circuits, superior to pre-computation and gated clocks. For sequential examples, on the other hand, pre-computation has proved to be ineffective, because of its intrinsic inability of blocking the present-state input lines. Obviously, this has had a negative impact on the integrated solution; in fact, the best power savings have been achieved by resorting to the gated clock alternative.

## References


[1] M. Alidina, J. Monteiro, S. Devadas, A. Ghosh, M. Papaefthymiou, "Precomputation-Based Sequential Logic Optimization for Low Power," *IWLPD-94: IEEE Intl. Workshop on Low-Power Design*, pp. 57-60, Napa Valley, CA, April 1994.

[2] M. Alidina, J. Monteiro, S. Devadas, A. Ghosh, M. Papaefthymiou, "Precomputation-Based Sequential Logic Optimization for Low Power," *ICCAD-94: ACM/IEEE Intl. Conf. on Computer-Aided Design*, pp. 74-81, San Jose, CA, November 1994.

[3] 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.

[4] J. Monteiro, J. Rinderknecht, S. Devadas, A. Ghosh, "Optimization of Combinational and Sequential Circuits for Low Power Using Precomputation," *1995 Chapel Hill Conf. on Advanced Research in VLSI*, pp. 430-444, Chapel Hill, NC, March 1995.

[5] 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.

[6] L. Benini, G. De Micheli, "Automatic synthesis of low-power gated-clock finite-state machines," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. CAD-15, No. 6, pp. 630-643, June 1996.

[7] L. Benini, G. De Micheli, E. Macii, M. Poncino, R. Scarsi, "Symbolic Synthesis of Clock-Gating Logic for Power Optimization of Control-Oriented Synchronous Networks," *EDTC-97: IEEE European Design and Test Conf.*, Paris, France, pp. 124-130, March 1997.

[8] S. Yang, "Logic Synthesis and Optimization Benchmarks User Guide Version 3.0," *Technical report, Microelectronics Center of North Carolina*, Research Triangle Park, NC, January 1991.

[9] F. Brglez, D. Bryan, K. Koźmiński, "Combinational Profiles of Sequential Benchmark Circuits," *ISCAS-89: Intl. Symp. on Circuits and Systems*, pp. 1929-1934, Portland, OR, May 1989.

[10] E. M. Sentovich, K. J. Singh, C. W. Moon, H. Savoj, R. K. Brayton, A. Sangiovanni-Vincentelli, "Sequential Circuits Design Using Synthesis and Optimization," *ICCD-92: IEEE Intl. Conf. on Computer Design*, pp. 328-333, Cambridge, MA, October 1992.

[11] A. Salz, M. Horowitz, "IRSIM: An Incremental MOS Switch-Level Simulator," *DAC-26: ACM/IEEE Design Automation Conf.*, pp. 173-178, Las Vegas, NV, June 1989.