Power Optimization of Combinational Modules Using Self-Timed Precomputation

António Mota
IST-INESC
mota@algos.inesc.pt

José Monteiro
IST-INESC
jcm@inesc.pt

Arlindo Oliveira
Cadence European Labs / IST-INESC
aml@inesc.pt

Abstract

Precomputation has recently been proposed as a very effective power management technique. Precomputation works by preventing some of the inputs from being loaded into the input registers, thus significantly reducing the switching activity in the circuit. In this paper, we present a self-timed approach for the precomputation of combinational logic circuits. This technique allows for maximum power savings without the need of a clock signal. However, we may incur in some delay penalty. We describe how to achieve significant power reductions without increasing the maximum delay, by choosing a judicious placement of the latches in the combinational logic circuit. Experimental results are presented for arithmetic modules, confirming that power dissipation can be greatly reduced with marginal increases in circuit area and almost zero delay increase.

1 Introduction

The reduction of the average power consumption in digital circuits is becoming an increasingly important task, due to the generalization of portable equipments and concerns with heat dissipation in fast integrated circuits. Less power consumption means longer battery life and better device reliability.

In static CMOS circuits, the average power dissipation is directly related to the average switching activity of the nodes in the circuit. Recently, precomputation [1] has been proposed as a powerful sequential logic optimization method for low-power. This method allows for the reduction of the average power dissipation in an arbitrary sequential synchronous circuit by stopping transitions at the inputs to the circuit. The method described in the present paper describes a self-timed approach for precomputation, applicable to the design of asynchronous modules.

The asynchronous environment presents several advantages, namely:

1. **low power operation**: compared to a synchronous circuit in a synchronous system, where there are always switching nodes due to the presence of the clock, an asynchronous circuit presents a much lower average power dissipation because there is switching activity in the circuit only when performing computations;
2. **high speed operation**: in a synchronous system with a considerable dimension, there is the problem of clock skew (delay in the arrival times of the clock to the different parts of the circuit), which frequently forces the clock frequency to be reduced in order to maintain correct functionality. In a typical asynchronous system, the data flowing is locally controlled by sequencers or other devices, enabling high speed operation even on large systems. Certain asynchronous implementations, for instance using dual-rail codes [2, 4], can signal the end of computations, thus achieving an average delay smaller than a synchronous equivalent where the maximum frequency of the clock is determined by the critical path of the circuit.

Hence, the realization of asynchronous modules using self-timed precomputation stands as a very desirable technique. Not only do we have reduced switching activity within active modules due to the use of precomputation, but also the use of asynchronous modules dispenses a clock signal.

The protocol related to the operation of the modules requires that a "request for computation" signal be asserted after the circuit's primary inputs become stable. No assumption is made about the time interval involved in the process, resulting in a zero-wait-states mechanism or self-timed operation. The modules can then be controlled by the traditional sequencers or other controllers that usually implement handshaking protocols [2].

The paper is organized as follows. Section 2 presents the precomputation architecture. In Section 3, we describe how self-timed logic can be effectively used to perform precomputation on asynchronous circuits. We discuss how to trade power for delay by latch positioning in Section 4. In Section 5, we apply our approach to some arithmetic modules and present experimental results showing the validity of the developed work.

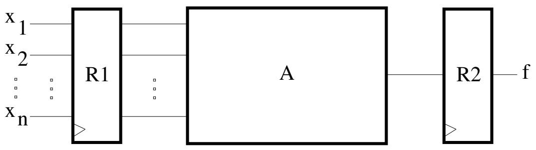

Fig. 1: Original circuit.

2 Precomputation Architecture

The precomputation technique first proposed in [1] achieves the reduction of the switching activity in a logic circuit by preventing transitions at the inputs from propagating. For this purpose, a simple combinational circuit (the precomputation logic) is added to the original circuit. Under certain input conditions, the precomputation logic disables the loading of all or a subset of the input registers. Under these input conditions, no power is dissipated in the portions of the original circuit with only disabled registers as inputs.

To illustrate this method, consider the 1-stage pipeline circuit in Figure 1, where the combinational logic block A is bounded by blocks of registers R_1 and R_2. Figure 2 shows this circuit after precomputation has been applied. In this figure, g_1 and g_2 constitute the precomputation logic. Power dissipation in the original circuit A is reduced when the outputs of either g_1 or g_2 evaluate to 1.

The precomputation logic is extra logic that is added to the circuit, offsetting the power savings obtained by shutting down the input registers. In order to keep this logic as simple as possible, g_1 and g_2 are built such they are a function of a small subset of the inputs to A.

The problem then resumes to deciding which inputs to use for the precomputation logic. Increasing the size of the subset increases the probability of switching off inputs to the circuit, but also increases the area and delay of the precomputation logic. Selection of the inputs to use is based on the probability p_i that an output f can be computed without knowing some input i (the observability don't-care set) which is given by:

$$p_i = \text{prob}\,(f_i \cdot f_{i'}) + \text{prob}\,(f'_i \cdot f'_{i'})$$

where f_i is the cofactor of f with respect to i.

In [1], it is also suggested how to apply the precomputation method on combinational circuits. Since there are no registers at the inputs and because of the delay associated with the precomputation logic, signals can only be stopped in the middle of the combinational logic block, through the use of latches. This situation is depicted in Figure 3. The limitation in this approach is that it does not allow the reduction of switching activity in the first levels of logic (block B in Figure 3).

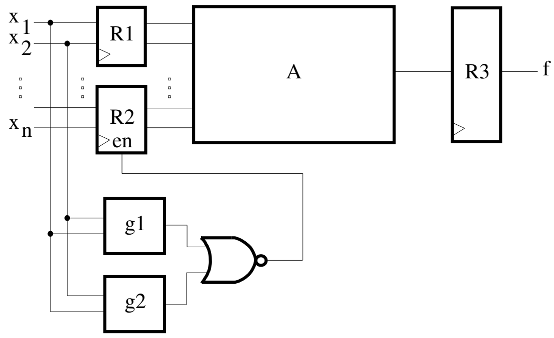

Fig. 2: Synchronous sequential precomputation architecture.

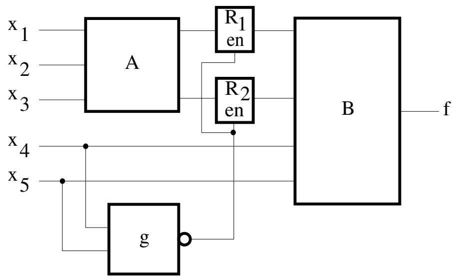

Fig. 3: Combinational precomputation architecture.

In the next section, we describe how self-timed logic can be used to perform precomputation for the whole combinational circuit by stopping transitions at the inputs.

3 Self-timed Precomputation Logic

The algorithm involved in the synthesis of the precomputation logic for a given circuit is explained in [1]. In order to be able to build an asynchronous version of the architecture shown in Figure 2, we will use a self-timed implementation of the precomputation logic. This way we can generate the required signal for enabling or disabling the latches after the assertion of the "request for computation" signal. Further, we must ensure that no glitches are generated for that signal.

We choose to implement the self-timed precomputation logic as a precharged logic gate [4]. The general structure of this logic is presented in Figure 4. While the request signal is at 1, the output of the logic is 1, therefore disabling the latches (active low). To start a new computation, the request signal is lowered and the precomputation logic is evaluated for the current input vector. If precomputation is possible for this input vector, the latch enable signal will stay at 1, thus filtering any transitions at the latches inputs. On the other hand, if a "switch off" condition is not detected, the enable signal goes to 0, enabling the latches and allowing for the new input values to propa-

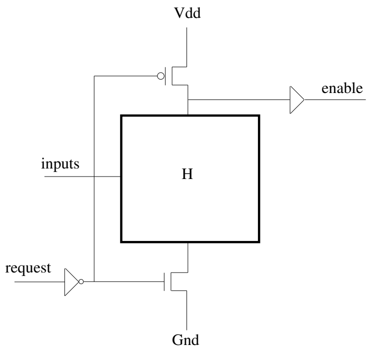

Fig. 4: General structure of the self-timed precomputation logic, implemented as a precharged logic gate.

gate into the circuit. Note that in the combinational precomputation proposed in [1] there was no mechanism of ensuring that the latches were disabled while the precomputation logic is evaluating.

In Figure 4, in the block H that implements the precomputation function there can only be N-type MOS transistors in order to minimize the delay of the logic when the output is changing from 1 to 0. Therefore, some of the inputs may have to be inverted. In this case, we may need to delay the request signal in order to synchronize it with the inputs, thus avoiding glitches at the output. The output of the precharged logic goes through a buffer to ensure that the transition delay of the signal is not affected by his fanout.

Although using the architecture of Figure 2 allows for maximum power savings, there can be some delay penalty when precomputation is not possible. After the signal "request for computation" is asserted, the new input values are deferred until the precomputation logic finishes evaluating. In the next section, we discuss how intermediate power versus delay solutions can be made by choosing appropriate latch positions. Observe that in the case of the synchronous method of [1], this delay overhead is passed to the previous pipeline stage.

4 Trade-off in latch positioning

The position of the latches determines the impact on power dissipation and circuit delay. In the architecture of Figure 2, the position of the latches (at the primary inputs of the circuit) permits the best possible power savings, but exhibits the worst delay characteristics. This delay equals the sum of the delay of the precomputation logic (the time required to generate the enable signal for the latches) with the delay related to the critical path of the function F. It is possible, however, to get the minimum delay possible at the expense of non-optimum power savings. This is illustrated in Figure 3, where some of the latches were placed somewhere between two internal nodes.

The delay penalty is minimized when the ideal position for the latches is found, making possible delays that equal the delay of the critical path of the circuit plus a small contribute due to the latches's response. It is possible, then, to get power savings with virtually no performance loss.

If one chooses to minimize the delay penalty, then the process of determining the ideal position of the latches, i.e., finding how many logic levels must be advanced for a given input, obeys the following algorithm:

1. determine the delay of the latch enable signal, caused by the precharged logic;
2. for those inputs whose slack (relative to the critical path of the circuit) is greater than the delay due to the precharged logic, the latches can be placed directly at the primary inputs of the circuit;
3. for those inputs whose slack corresponds to a delay smaller than the delay due to the precharged logic, one must advance the latches a number of the logical levels such that this condition is no longer verified.

When no hard delay constraints exist, intermediate solutions are possible where some delay is allowed so as to obtain larger power savings. The cost function to minimize will typically be the power-delay product.

5 Experimental Results

Due to the switch-level nature of the precomputation logic, the simulations were made with a switch-level simulator, the SLS of OCEAN [5], and with an electrical-level simulator, the HSPICE of META-SOFTWARE. Two arithmetic modules were tested: a 16-bit comparator and a 16-bit carry-select adder (with carry look-ahead blocks). The results for the comparator are presented in Tables 1 and 2, and the results for the adder appear in Tables 3 and 4.

P_{avg} denotes de average power and all the values are in µW. The delay values (in ns) were obtained with HSPICE, being that t_p 0 -> 1 (1 -> 0) denotes de propagation delay along the critical path of the circuit. An input's change that forced the circuit's output (MSB for the adder) to change from 0 to 1 (from 1 to 0) was used. *N* denotes the number of vectors used for the simulation, obtained from a random sequence. Under the column named SLS/SPICE, the simulations were made with the same vectors, in order to evaluate the discrepancy of the power estimate obtained at the switch-level and at the electrical-level. These discrepancies are very small, thus enabling fast power estimations with a large number of input vectors using the switch-level simulator. The cycle period of the test vectors were 200 ns. All the logical gates have an intrinsic output

circuit	t_p 0 -> 1	t_p 1 -> 0	P_{avg} (N=60) SLS	P_{avg} (N=60) SPICE	P_{avg} (N=4000) SLS
original	29.05	29.5	2015	2029	1762
2 bits	31.9 (+9.8%)	32.25 (+9.3%)	1034 (-49%)	996.2 (-51%)	1017 (-42%)
3 bits	32.35 (+11%)	32.6 (+11%)	898.6 (-55%)	834.8 (-59%)	841.6 (-52%)
4 bits	32.75 (+13%)	32.95 (+12%)	856.3 (-58%)	786.6 (-61%)	830.3 (-53%)

Table 1: 16 bit comparator for maximum power saving.

circuit	t_p 0 -> 1	t_p 1 -> 0	P_{avg} (N=4000) SLS
original	29.05	29.5	1762
2 bits	29.7 (+2.2%)	30.2 (+2.4%)	1112 (-37%)
3 bits	30.2 (+3.9%)	30.8 (+4.2%)	1010 (-43%)
4 bits	29.7 (+2.2%)	30.2 (+2.4%)	1080 (-39%)

Table 2: 16 bit comparator with minimum delay.

load capacitance of 0.5 pF, equal to, approximately, twice the input capacitance of the NMOS transistor gate used in the electrical simulations. The rise/fall time of the input stimuli were 0.1 ns. The switch-level simulation was accomplished with SLS in level 3. The electrical-level simulation was accomplished with SPICE in level 2.

The values for the original circuits, i.e., with no precomputation, are shown in row *original*. For the comparator, we present results using 2, 3 and 4 input bits for the precomputation logic. For the adder, we used 1 and 2 inputs for precomputation.

As shown in Table 1, in the case of the comparator substantial power savings can be obtained, up to 50%. As more inputs are added to the precomputation logic, the power savings increase. This is true up to a certain number, when the complexity of the precomputation logic offsets the power saved in the original logic block. Also, with more inputs the delay increases. For 3 inputs in the precomputation logic, we get 50% power savings, yet with a 13% delay increase. If the latches are moved to internal nodes, the delay penalty comes down to around 4% and we can still obtain 40% power reductions (Table 2). The results for the carry-select adder follow the same pattern.

circuit	t_p 0 -> 1	t_p 1 -> 0	P_{avg} (N=60) SLS	P_{avg} (N=60) SPICE	P_{avg} (N=4000) SLS
original	15.4	16.7	10972	11681	10010
1 bit	17.1 (+11.0%)	19.0 (+13.7%)	9822.1 (-10.5%)	10529 (-9.9%)	8685.5 (-13%)
2 bits	17.2 (+11.7%)	19.1 (+14.4%)	9008.7 (-18%)	9629.7 (-17.6%)	7904.2 (-21%)

Table 3: 16 bit carry-select adder for maximum power saving.

circuit	t_p 0 -> 1	t_p 1 -> 0	P_{avg} (N=4000) SLS
original	15.4	16.7	10010
1 bit	16.3 (+5.8%)	17.2 (+2.9%)	8924.1 (-11%)
2 bits	16.4 (+6.5%)	17.25 (+3.3%)	8312.0 (-17%)

Table 4: 16 bit carry-select adder with minimum delay.

6 Conclusions

A self-timed logic-level combinational logic optimization method for low power was presented. This method is precomputation based. We add a block that observes the state of some of the circuit's inputs, "switch off" states are forced for certain temporarily redundant blocks within the circuitry, therefore reducing the internal switching activity.

The method can be applied to any combinational circuit in an asynchronous system, the perfect low power environment due to the absence of switching nodes forced by the clock. The precomputation block is self-timed, i.e., it requires the assertion of a "request for computation" signal when the inputs become stable, with zero wait states. Therefore, the control is simplified and it is possible to operate the blocks at high speed.

The size of the precomputation logic determines the amount of power dissipation reduction and the delay. By choosing this size and the correct position of the latches that "block" the propagation of the unnecessary signals, it is possible to trade power savings for delay time. Further, it is possible, when some latches are located in inner parts of the circuit, to get power savings with virtually no increase in the delay of the circuit.

7 References

[1] M. Alidina, J. Monteiro, S. Devadas, A. Ghosh, M. Papaefthymiou, "Precomputation-Based Sequential Logic Optimization for Low Power", in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 2, no. 4, pp. 426-435, December 1994.

[2] L. A. Plana, S. M. Nowick, "Concurrency-Oriented Optimization for Low-Power Asynchronous Systems", in ISLPED 1996, Monterey, CA, pp. 151-156.

[3] J. M. Rabaey, "Digital Integrated Circuits - A Design Perspective", in Prentice-Hall, 1996.

[4] L. David, R. Ginosar, M. Yoeli, "An Efficient Implementation of Boolean Functions as Self-Timed Circuits", in IEEE Transactions on Computers, vol. 41, no. 1, pp. 2-11, January 1992.

[5] A. J. van Genderen, "SLS: An Efficient Switch-Level Timing Simulator Using Min-Max Voltage Waveforms", Proc. VLSI 89 Conference, Munich, pp. 79-88.