Journal of VLSI Signal Processing 16, 117–130 (1997)
© 1997 Kluwer Academic Publishers. Manufactured in The Netherlands.

Clock Skew Optimization for Peak Current Reduction

L. BENINI, P. VUILLOD,* A. BOGLIOLO,† AND G. DE MICHELI
Computer Systems Laboratory, Stanford University, Stanford, CA 94305-9030

Received August 1, 1996; Revised October 21, 1996

Abstract. The presence of large current peaks on the power and ground lines is a serious concern for designers of synchronous digital circuits. Current peaks are caused by the simultaneous switching of highly loaded clock lines and by the signal propagation through the sequential logic elements. In this work we propose a methodology for reducing the amplitude of the current peaks. This result is obtained by clock skew optimization. We propose an algorithm that, for a given clock cycle time, determines the clock arrival time at each flip-flop in order to minimize the current peaks while respecting timing constraint. Our results on benchmark circuits show that current peaks can be reduced without penalty on cycle time and average power dissipation. Our methodology is therefore well-suited for low-power systems with reduced supply voltage, where low noise margins are a primary concern.

1. Introduction

Clock skew is usually described as an undesirable phenomenon occurring in synchronous circuits. If clock skew is not properly controlled, unexpected timing violations and system failures are possible. Mainly for this reason, research and engineering effort has been devoted to tightly control the misalignment in the arrival times of the clock [1]. Although clock-skew control is still an open issue for extremely large chip-level and board-level designs, recently proposed algorithms for skew minimization have reported satisfying results [1–4]. For a large class of systems skew control can therefore be achieved with sufficient confidence margin.

Conservative design styles (such as those adopted for FPGAs) explicitly discourage "tampering with the clock" [5]. Nevertheless, the arrival time of the clock is often purposely skewed to achieve high performance in more aggressive design styles. In the past, several algorithms for *cycle-time minimization* have been proposed [6–10]. The common purpose of these methods was to find an *optimum clock-skewing strategy* that allows the circuit to run globally faster. Average power dissipation can also be reduced by clock skewing coupled with gate resizing [11].

In this work, we discuss the productive use of clock skew in a radically new context. We target the minimization of the peak power supply current. Peak current is a primary concern in the design of power distribution networks. In state-of-the-art VLSI systems, power and ground lines must be over-dimensioned in order to account for large current peaks. Such peaks determine the maximum voltage drop and the probability of failure due to electromigration [12]. In synchronous systems, this problem is particularly serious. Since *all sequential elements* are clocked, huge current peaks are observed in correspondence of the clock edges. These peaks are caused not only by the large clock capacitance, but also by the switching activity in the sequential elements and by the propagation of the signals to the first levels of combinational logic.

In this paper, we focus application specific integrated circuits implemented with semi-custom technology. We do not address the complex issues arising in custom-designed chips with clock frequencies over 150 MHz. For such high-end circuits, achieving adequate skew control is already a challenging task. We assume a single-clock edge-triggered clocking style, because it represents the worst case condition for current peaks. We propose an algorithm that determines the clock

*On leave from INPG—CSI, Grenoble, France.
†Also with DEIS, Università di Bologna, Italy.

arrival times at the flip-flops in order to minimize the maximum current on the power supply lines, while satisfying timing constraints for correct operation.

In addition, we propose *a clustering* technique that groups flip-flops so that they can be driven by the same clock driver. Since the number of sequential elements is generally large, it would not be practically feasible to specify a skew value for each one of them. In our tool, the user can specify the maximum number of clock drivers, and the algorithm will find a clustering that always satisfies the timing constraints while minimizing the peak current.

Any optimization technique based on clock control cannot neglect the structure and the performance of the clock distribution network and clock buffers [13]. Implementing skewed clocks with traditional buffer architectures imposes sizable power costs that may swamp the advantages obtained by clock skew. Our clocking strategy is based on a customized driver that achieves good skew control with negligible cost in power, area and performance.

Our technique is particularly relevant for low-power systems with reduced supply voltage, where the noise margins on power and ground are extremely low. Experimental results show that our method not only reduces the current peaks, but it does not increase the average power consumption of the system. We tested our approach on several benchmark circuits. On average, current peak reduction of more than 30% has been observed. Average power dissipation is unchanged and timing constraints are satisfied.

The results were further validated by accurate *post-layout* electrical simulation of circuits of practical size (over 100 flip-flops). The power dissipation due to the clock network and buffers was taken into account. The post-layout results confirm the practical interest of our method and the effectiveness of our clustering heuristic.

2. Skew Optimization

It is known that clock skew can be productively exploited for obtaining faster circuits. *Cycle borrowing* is an example of such practice: if the critical path delay between two consecutive pipeline stages is not balanced, it is possible to skew the clock in such a way that the slower logic has more time to complete its computation, at the expense of the time available for the faster logic. For large and unstructured sequential networks, finding the best cycle borrowing strategy is a complex task that requires the aid of automatic tools.

2.1. Background

We will briefly review the basic concepts needed for the formal definition of the skew optimization problem. The interested reader can refer to [1, 7, 9] for further information. Clock-skew optimization is achieved by assigning an arrival time to the local clock signals of each sequential element in the circuit. We consider rising-edge-triggered flip-flops and single clock. The clock period is T_{clk}. For the generic flip-flop i ($i = 1, 2, \dots, N$, where N is the number of flip-flops in the network) we define its *arrival time* T_i, $0 \leq T_i < T_{\text{clk}}$. The arrival time represents the amount of skew between the reference clock and the local clock signal of flip-flop i. A *clock schedule* is obtained by specifying all arrival times T_i. Obviously not all clock schedules are valid. The combinational logic between the flip-flops has finite delay. The presence of delays imposes constraints on the relative position of the arrival times.

The classical clock-skew optimization problem can be stated as follow: *find the optimal clock schedule* $\mathbf{T} = [T_1, T_2, \dots, T_N]$ *such that no timing constraint is violated and the cycle time* T_{clk} *is minimized*. This problem has been analyzed in detail and many solutions have been proposed. Here we follow the approach presented in [7] where edge-triggered flip-flops are considered.

We assume for simplicity that all flip-flops have the same setup and hold times, respectively called T_{SU} and T_{HO}. If there is at least one combinational path from the output of flip-flop i to the input of flip-flop j, we call the maximum delay on these paths $\delta_{i,j}^{\max}$. The minimum delay $\delta_{i,j}^{\min}$ is similarly defined. If no combinational path exists between the two flip-flops, $\delta_{i,j}^{\max} = -\infty$ and $\delta_{i,j}^{\min} = +\infty$. For each pair of flip-flops i and j, two constraints must be satisfied.

First, if a signal propagating from the output of i reaches the input of j before the clock signal for j is arrived, the data will propagate through two consecutive sequential elements in the same clock cycle. This problem is called *double clocking* and causes failure. The first kind of constraints prevents double clocking:

$$T_i + \delta_{i,j}^{\min} \geq T_j + T_{\text{HO}} \tag{1}$$

On the other hand, if a signal propagating from i to j arrives with a delay larger than the time difference between the next clock edge on j and the current clock edge on i, the circuit will fail as well. This phenomenon is called *zero clocking*. Zero clocking avoidance is enforced by the following constraint:

$$T_i + T_{SU} + \delta_{i,j}^{\max} \leq T_j + T_{clk} \qquad (2)$$

Input and output impose constraints as well. Input constraints have the same format as regular constraints, where the constant value of the input arrival time T_{in} replaces the variable T_i. For output constraints the variable T_j is replaced by the constant output required time T_{out}.

The total number of constraint inequalities constructed by this method is $O(N^2 + I + O)$, where, I and O are the number of inputs and outputs respectively. In practice, this number can be greatly reduced. Techniques for the reduction of the number of constraints are described in [6, 8] and are not discussed here for space reasons.

Example. We obtain the constraint equations for the circuit in Fig. 1. There are two variables T_1 and T_2, representing the skew of the clocks CLK1 and CLK2. The clock period is T_{clk}. We assume that $T_{SU} = T_{HO} = 0$. The constraints for variable T_1 are the following:

$$\begin{aligned} T_1 + \delta_{1,2}^{\max} &\leq T_2 + T_{clk} \\ T_1 + \delta_{1,2}^{\min} &\geq T_2 \\ T_1 + \delta_{1,out}^{\max} &\leq T_{out} + T_{clk} \\ T_{in} + \delta_{in,1}^{\min} &\geq T_1 \end{aligned}$$

Moreover, $0 \leq T_1 \leq T_{clk}$. Similar constraints hold for T_2. We have eliminated one input constraint and one output constraint because we assume that skews are positive and that the circuit with no skews was originally satisfying all input and output constraints. Notice that all constraints are linear. The feasibility of a set of linear constraints can be checked in polynomial time by the Bellman-Ford algorithm [14].

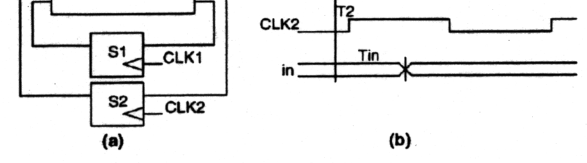

Figure 1. (a) Example circuit, with two flip-flops. (b) Timing waveform representing the skewed clocks.

An important practical consideration that is often overlooked in the literatures is the generation of the skewed clocks. Although generating delays is a relatively straightforward task, the cost (in power, area and signal quality degradation) of the delay elements is an important factor in the evaluation of optimization techniques based on clock skewing. We will first concentrate on the theory of clock skew optimization for the sake of simplicity. Circuits for the generation of skewed clocks will be discussed in a later section.

Cycle time minimization is an optimization problem targeting the minimization of a linear cost function (i.e., $F(T_1, T_2, \ldots, T_N, T_{clk}) = [0, 0, \ldots, 0, 1] \cdot [T_1, T_2, \ldots, T_N, T_{clk}]$) of linearly constrained variables. It is therefore an instance of the well-known *linear programming* (LP) problem. Several efficient algorithms for the solution of LP have been proposed in the past [15]. Our problem is radically different and substantially harder. It can be stated as follows: *find a clock schedule such that the peak current of the circuit is minimum.* The cost function that we want to minimize is *not* linear in the variables T_i. In the following subsection, we discuss this issue in greater detail.

2.2. *Cost Function*

In peak current minimization, the constraints are exactly the same as for the traditional cycle time minimization, the only difference being that we consider T_{clk} as a constant. Unfortunately, our cost function is much more complex. Ideally, we would like to minimize the maximum current peak that the circuit can produce. This is however a formidable task, because such peak can be found by exhaustively simulating the system for all possible input sequences (and a circuit level simulation would be required, because traditional gate-level simulators do not give information on current waveforms). To simplify the problem, we make two important assumptions. First, we only minimize the current peak directly caused by clock edges (i.e., caused by the switching of clock lines and sequential elements' internal nodes and outputs). This approximation is justified by experimental evidence. In all circuits we have tested, the largest current peaks are observed in proximity of the clock edges. The current profile produced by the propagation of signals through the combinational logic is usually spread out and its maximum value is sensibly smaller.

Notice that we are *not* neglecting the combinational logic, but we consider its current as a phenomenon on which we have no control. Again, this choice is motivated by experimental evidence: our tests show that in most cases, the current profile of the combinational logic is not very sensitive to the clock schedule. For some circuits, the combinational logic may be dominant and strongly influenced by the clock schedule. We will discuss this case in a later section.

The second approximation regards the shape of the current waveform. Each sequential element produces two peaks, one related to the rising edge of the clock, and the other to the falling edge. For a given flip-flop, the shape of the current peaks is weakly pattern dependent. We approximate the current peaks produced by each sequential element (or group of sequential elements) with two triangular shapes, that are fully characterized by four parameters: starting time t_s, maximum time t_m, maximum value current I_m and final time t_f.

To compute these parameters we run several current simulations [16] (see Section 4) and we obtain current waveform envelopes $I_{\text{av}}(t)$ ($I_{\text{av}}(t)$ is obtained by averaging the current at t on different input patterns). For each peak of the curve I_{av}, we define the four parameters as shown in Fig. 2: t_s is the time at which the current first reaches 1% of the maximum value, t_f is the time at which the current decreases below 1% of the maximum value, I_m and t_m are respectively the maximum current value and the time when it is reached. Experimentally we observed that the triangular approximation is satisfactory for the current profiles of the sequential elements. For combinational logic, this approximation is generally inaccurate. The current profile of combinational logic is more adequately modeled by a *piecewise linear* approximation. Fortunately, any piecewise linear function can be decomposed in the sum of one or more triangular functions.

The total current is the sum of the current contributions represented as triangular shapes. Every flip-flop i has two associated contributions $\Delta_i^r(t, T_i)$ and $\Delta_i^f(t, T_i)$, representing respectively the current drawn on the raising and falling edge of the clock. Notice that such contributions are functions of time t and of the clock arrival time T_i. In fact, the curve translates rigidly with T_i. The current drawn by the combinational logic is approximated with a sum of triangles (i.e., a piecewise linear waveform) $\Delta_C(t)$. Note that $\Delta_C(t)$ is *not* a function of the arrival time of any clock. The total current is the sum of the contributions due to flip-flops and combinational logic:

$$I_{\text{tot}}(t, \mathbf{T}) = \Delta_C(t) + \sum_{i=1}^{N} \Delta_i^r(t, T_i) + \sum_{i=1}^{N} \Delta_i^f(t, T_i) \tag{3}$$

We clarify this equation through an example.

Example. The current profiles for the flip-flops of the circuit in Fig. 1 are shown in Fig. 3 for one assignment of T_1 and T_2. The current profile of the combinational logic for this example is shown in Fig. 4 with its approximation.

The contribution of a flip-flop is approximated by two triangular shapes. The first corresponds to the

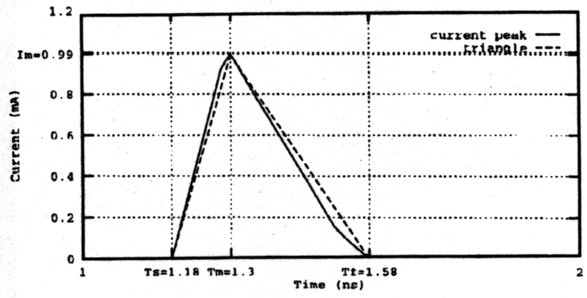

Figure 2. The four parameters characterizing the triangular approximation of the average current profile. t_s and t_e are the times at which the current reaches 1% of its maximum value.

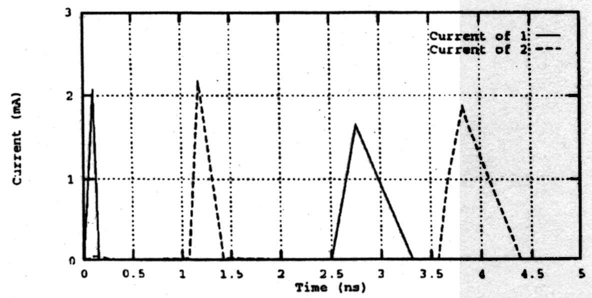

Figure 3. Current profiles for the two flip-flops 1 and 2 from simulation of our example circuit.

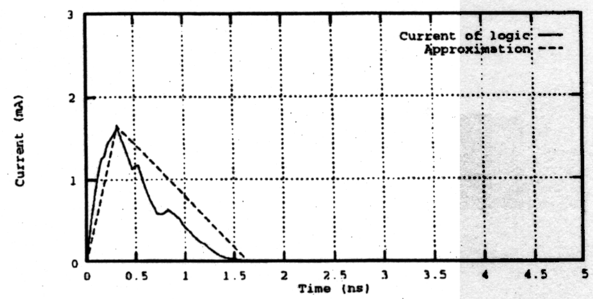

Figure 4. Current profile corresponding to the combinational logic from simulation of our example circuit. The dashed line is its piecewise linear approximation.

rising edge of the clock, the second to the falling edge. Here we have $T_1 = 0$ ns and $T_2 = 1.07$ ns. Notice that the current profile of flip-flop 2 is shifted to the right. The profiles for the two flip-flop do not have exactly the same shape because they are differently loaded. Notice that when $T_1 = T_2$ the two current profiles of the flip-flops are perfectly overlapped. When $T_1 \neq T_2$, the two contributions are skewed.

The cost function F that approximates the peak current is the maximum value of the (approximate) current waveform over the clock period T_{clk}:

$$F(\mathbf{T}) = \max_{t \in [0, T_{\text{clk}}]} \{I_{\text{tot}}(t, \mathbf{T})\} \tag{4}$$

For the above example, the value of the cost function $F(T_1, T_2)$ is the maximum value of the sum of the five triangles over the clock period T. In this case $F(0, 1.07) = 2.7$, whereas initially $F(0, 0) = 4.2$. Our target is to find the optimum clock schedule $\mathbf{T}_{\text{opt}}$ which minimizes the cost function F, while satisfying the timing constraints for correct operation of the circuit.

3. Peak Current Minimization

We now describe our approach to the minimization of the cost function described in the previous section. The first key result of this section is summarized in the following proposition.

Theorem 1. *The cost function F of Eq.* (4) *can be evaluated in quadratic time* (*in the number of triangular contributions*).

Proof: The proof of this theorem is given in a constructive fashion, by describing a $O(N_\Delta^2)$ algorithm (N_Δ is the number of triangular current contributions) for the evaluation of the cost function. The algorithm is based on the observation that the maximum of the cost function can be attained in a finite number of points, namely the points of maximum of the triangles that compose it. In order to evaluate the value of F in one of such points, we must check if the corresponding triangle is overlapping with any of the other contributions. The quadratic complexity stems from this check: for each maximum value v_i (`val` in the pseudo-code), we check if its corresponding triangle Δ_i is overlapping with any other triangle. In case there is overlap, v_i is incremented by the value of the overlapping waveform at the maximum point. Thus, we have two nested loops with iteration bound N_Δ. The pseudo-code of the algorithm is shown in Fig. 5. □

```
/* Let T[i] (i..N) be the variable vector */
/* Delta_orig[i] (i..2N+1) are the 2N+1 contributions when T[i]=0 */
float evaluate (T)
  /* computes the contributions for the vector T */
  Delta = translate_triangles (Delta_orig, T);
  max = 0;
  foreach (c1 in [0..2N])
    val = max(Delta([c1]));
    foreach (c2 in [0..2N])
      if (c2 != c1) then
        if (overlap (Delta[c1], Delta[c2])) then
          /* we look if the 2 triangles overlap and add the value */
          /* of c2 at the maximum point of c1 */
          val += get_value (Delta[c2], time_max (Delta[c1]));
        endif;
      endif;
    endfor;
    if (val ≥ max) then max = val;
   endfor;
   return (max);
 end evaluate;
```

Figure 5. $O(N^2)$ algorithm for the computation of the cost function F.

The second key result is summarized by the following theorem:

Theorem 2. *The peak current minimization problem is an instance of the* **constrained DC optimization problem** (*DC optimization problems are those where the cost function can be expressed as the difference of two concave functions* [17]).

Proof: The proof of the theorem is straightforward. The cost function $F(\mathbf{T})$ is the maximum over a finite interval of I_{tot} which is obtained by summing triangular current contributions. Hence, I_{tot} is piecewise-linear. The maximum of a piecewise-linear function is piecewise-linear [17]. The Theorem is therefore proven, because piecewise-linear functions are DC [17]. □

An important consequence of Theorem 2 is the NP-completeness of the current minimization problem (since DC optimization is NP-complete). Our solution strategy is heuristic and it is based on a genetic algorithm (GA) [18]. We will briefly discuss the application of the genetic algorithm for the solution of the problem at hand. Refer to [18] for a more in-depth treatment of genetic search and optimization techniques.

3.1. Heuristic Peak Current Minimization

The minimization of a multi-modal cost function such as the one representing the current peak is a difficult task. Gradient-based techniques [17] are fast and well-established, but they tend to rapidly converge to a local minimum. The *genetic algorithm* is a global optimization technique that mimics the dynamics of natural evolution and survival of the fittest.

A set of initial random solutions (a *population*) is generated. For each solution (an *individual* of the population) the cost function is evaluated. From the initial population a new population is created. The best individuals in the old population have a high probability of either becoming member of the new population or participating in the generation of new solution points. New solutions are created by combining couples of good solutions belonging to the old population. This process is called *crossover*. Weak individuals (i.e., points with a high value of the cost function) have a low probability of being selected for crossover or replication.

The creation and cost evaluation of new sets of solutions is carried on until no improvement is obtained on the best individuals over several successive generations. Alternatively, a maximum number of cost function evaluations is specified as a stopping rule. The basic genetic algorithm and many advanced variations have been applied to a number of hard optimization problems for which local search techniques are not successful. The interested reader can refer to [18] for several examples and theoretical background.

The GA approach is attractive in our case because we have an efficient way to compute the cost function (with low-order polynomial complexity). GA-based functional optimization requires a very large number of function evaluations (proportional to the number of generations multiplied by the size of the population). Since F can be efficiently evaluated, large instances of the problem can be (heuristically) solved.

Notice two important facts. First, our algorithm heavily relies on the triangular approximation. If we relax this assumption, the evaluation of F becomes an extremely complex problem (finding the maximum of a multi-modal function), and the GA approach would not be practical. Second, we consider the contribution of the combinational logic as function of time only (independent from the clock schedule). As a consequence, if the maximum current is produced by the combinational logic, $F(T_1, \dots, T_N)$ is a constant, and no optimization is achievable.

Although the experimental results seem to confirm that the GA is an effective optimization algorithm for peak current minimization, there are margins of improvement. First, the GA does not provide any insight on how far is the best individual from the absolute minimum of the cost function over the feasible region. Moreover, the quality of the results can be improved if the GA is coupled with gradient techniques that are applied starting from the GA-generated solutions and lead to convergence towards local minima.

3.2. Clustering

Up to now, we have assumed that the arrival time T_i of each individual flip-flop can be independently controlled. This is an unrealistic assumption. In VLSI circuits the clock is distributed using regular structures such as *clock trees* [1, 19]. Usually, sub-units of a complex system have local clocks, connected with buffers (drivers) to the main clock tree. The buffers are the ideal insertion points for the delays needed for skew optimization (a practical implementation of such delays will be discussed later). In general it would not be feasible to provide each flip-flop with its own buffer and delay element, for obvious reasons of layout complexity, routability and power dissipation.

Since clock-skew optimization is practical only if applied at a coarser level of granularity, we have developed a strategy that allows the user to specify the number of clusters (i.e., the number of available clock buffers with adjustable delay), and heuristically finds flip-flops that can be clustered without large penalty on the cost function. Here we assume that no constraints on the grouping of flip-flops have been previously specified. This is often the case for circuits generated by automatic synthesis. Structured circuits (data-path, pipelined systems) with pre-existing clustering constraints are discussed later.

Our clustering algorithm can be summarized as follows. The user specifies the number of clusters N_P. First, we solve the peak current minimization problem without any clustering (every flip-flop may have a different arrival time). We then insert the flip-flops in a list ordered by clock arrival times. The list is partitioned in N_P equal blocks. New constraint equations and new current profiles are obtained for the blocks of the partition. A new peak current minimization is solved where the variables are the arrival times T_j^P, $j = 1, 2, \dots, N_P$, one for each cluster. We also recompute the delays from cluster i to cluster j. The number of equations reduces to $O(N_P^2 + I + O)$. The pseudocode of the clustering algorithm is shown in Fig. 7.

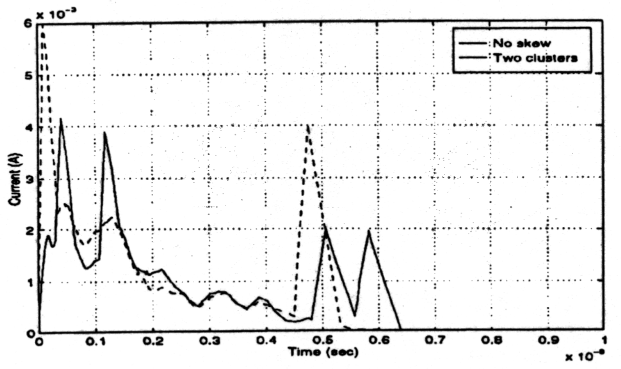

Figure 6. Current profile for benchmark s208 before and after skew optimization with two clusters. The current profiles are obtained by accurate current simulation.

```
/* Let F[i] (i..N) be the instances of the flip-flops */
/* Let T[i] (i..N) be the values given by the MA for instance i */
/* Let N_p be the number of clusters to obtain */
F_sort[i] = sort_by_skew (F[i], T[i]);
size_cluster = N / N_p;
tum_cluster = 0;
foreach (i in F_sort[i])
  if (size (Cluster [num_cluster] == size_cluster)) then
    num_cluster++;
  endif;
  add_in_cluster (Cluster [num_cluster], F_sort[i]);
endfor;
return (Cluster);
```

Figure 7. Clustering algorithm.

The complexity of the clustering algorithm is dominated by the complexity of the ordering of the clock arrival times. Thus, the overall complexity is $O(N \log N)$. Clearly, the overall computational cost of our procedure is not dominated by the clustering step.

Using clustering, we can control the granularity of the clock distribution. The first step of our partitioning strategy is based on the optimal clock schedule found without constraints on the number of partitions. Clustering implies loss in optimality, because some degrees of freedom in the assignment of the arrival times are lost. Our clustering strategy reduces the loss by trying to enforce a natural partitioning. The second iteration of current peak optimization guarantees correctness and further reduces the optimality loss.

Example. Consider the small benchmark s208. It consists of 84 combinational gates and 8 flip-flops. The cycle time is 10 ns, the clock has 50% duty cycle. The current profile for the circuit is shown in Fig. 6 with the dashed line. Observe the two current peaks synchronized with the raising and falling edge of the clock. The irregular shape that follows the first peak shows the current drawn by the combinational logic.

The skew is then optimized with the constraint of 2 partition blocks (i.e., two separate clock drivers allowed). The current profile after skew optimization is shown in Fig. 6 with continuous line. The beneficial effect of our transformation is evident. The two current peaks due to the two skewed clusters of switching flip-flops have approximatively one half of the value of the original peaks. The irregular current profile between peaks is due to the propagation of the switching activity through the combinational logic. Notice that skewing the clock does not have a remarkable impact on the overall current drawn by the combinational logic.

Several different clustering heuristics could be tried. In our experiments we observed that our heuristic produced consistently good results, and did not excessively degrade the quality of the solution with no clustering. However, notice that our heuristic can be applied only if an optimal clock schedule with fine granularity has already been found. For large circuits this preliminary step may become very computationally intensive. In these cases, the user can specify clusters using a different heuristic. In the following sub-section a clustering technique is discussed for dealing with large and structured data-path circuits.

3.3. *Clustering for Staged Circuits*

In the previous discussion, we have solved the current peak optimization problem assuming that we cannot control the current profile of the combinational logic. For many practical circuits this is an overly pessimistic assumption, because the data path of large synchronous systems is often *staged*. In a staged structure, a set of flip-flops A feeds the inputs of a combinational logic block. The outputs of the block are connected to the inputs of a second set of flip-flops B. The sets A and B are disjoint. The flip-flops in A and the block of combinational logic are called a *stage*. Pipelined circuits are staged, and most data paths have this structure, that makes the design easier and the layout much more compact.

If the circuit has a staged structure, the behavior of the combinational logic is much more predictable. If we cluster the flip-flops at the input of each stage, by imposing the same arrival time (i.e., assigning the same clock driver) to their clock signal, we can guarantee that all inputs of the combinational logic of the stage are synchronized. As a consequence, the current profile of the combinational logic translates rigidly with the arrival time of the clock of the flip-flops at its inputs.

For staged circuits our algorithm is more effective, because the clock schedule controls the current profile of the combinational logic as well. The current peak can therefore be reduced even if it is entirely dependent on the combinational logic. Interestingly, the application of clock skew to pipelined circuits has been investigated in [20], where the authors describe a high-performance design style called *counter-flow clocked pipelining* based on multiple skewed clocks. Although the methodology in [20] was not developed to reduce current peaks, the authors observe that clock skewing has beneficial effects on peaks for practical chip level designs.

4. Layout and Clock Distribution

To make our methodology useful in practice, several issues arising in the final steps of the design process need to be addressed. First, pre-layout power and delay estimates are inaccurate and constraints met before layout may be violated in the final circuit. Second, and more importantly, the impact of the clock distribution scheme is not adequately considered when performing pre-layout estimation. Any optimization exploiting clock skew is not practical if the skew cannot be controlled with sufficient accuracy or the cost of generating skewed clocks swamps the reductions that can be obtained.

In the following discussion we assume that the layout of the circuit is automatically generated by placement and routing tools starting from structural gate-level specification. Clusters are specified by providing different names for clock wires coming from different buffers. Flip-flops connected to the same buffer will have the same clock wire name.

To overcome the uncertainty in pre-layout power and delay estimation, two different approaches can be envisioned. We can apply our methodology as a post-processing step after layout. In this case, the constraints can be formulated with high accuracy, and the clock schedule computed with small uncertainty. After finding the optimal clock scheduling and clustering, we need to iterate placement and routing, specifying the new clock clusters and their skews. Alternatively, we can find the clock schedule using pre-layout estimates and allowing a safety margin on the constraint equations. This can be done by increasing the length of the longest paths estimates and decreasing that of the shortest paths, and considering some delay inaccuracy on the computed skews. The effect of the margins is to potentially decrease the effectiveness of the optimization, but in this approach the lay out has to be generated only once.

We chose the second approach for efficiency reasons. For large circuits, the automatic layout generation step dominates the total computation time. The first approach was disregarded because it requires the iteration of the layout step, with an unacceptable computational cost. Notice that this is not always the best choice: if an advanced and efficient layout system is available, which allows incremental modifications (local re-wiring of the clock lines) at low computational cost, the first approach becomes preferable. Moreover, if clustering is user-specified and consistent with the partitioning of the clock distribution implemented in the layout, there would be no need of re-wiring at all, and the first approach would always lead to better results.

4.1. Clock Distribution

After placement and routing, we have complete and accurate information on the load that must be driven by the clock buffer of each cluster. Although many algorithms have been developed for the design of topologically balanced clock trees considering wire lengths and tree structure, for the technology targeted by this work such algorithms are an overkill. Algorithms based on wire length and width balancing become necessary for clock frequencies and die sizes much larger than the ones we deal with [19]. In our case, clock distribution design is simply a buffer design problem.

We assume that we have no control on how the clock tree will be routed, once we specify the clock clusters (i.e., the flip-flops to be connected to the same buffer). From layout we extract the equivalent passive network representing the clock tree for each cluster. We need to design a clock buffer that drives the load with satisfactory clock waveform and skew. The clock waveform must have fast and sharp edges (to avoid short circuit power dissipation on the flip-flops and possible timing violations), and the skew must be as close as possible to the one specified by our algorithm.

Numerous techniques for buffer sizing have been proposed [1, 21] and empirical formulas are available. We used computer-aided optimization methods based on iterative electrical simulation (such as those implemented in HSPICE [22]) that have widespread usage in real-life designs. The main advantage of this approach

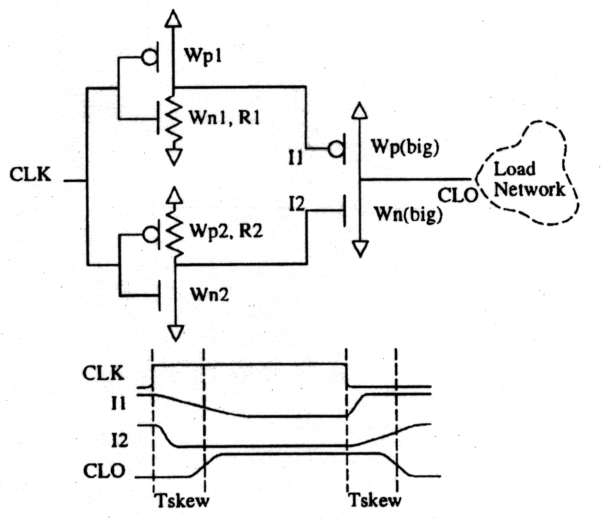

Figure 8. Buffer for generation of skewed clock and signal waveforms.

is that no simplifying assumptions are made on the transistor models and on the buffer architecture. Although the basic clock buffer architecture (a chain of scaled inverters) is well-suited for driving large loads with satisfactory clock waveform, its performance for generating controlled clock skew is poor. There are two standard ways to generate clock skews using the basic buffer: i) add an even number of suitably scaled inverters ii) add capacitance and/or resistance between stages to slow down the output.

Both methods have considerable area and power dissipation overhead. The first method adds stages that dissipate additional power (and use additional area), the second method is probably even worse for both cost measures, because it produces slow transitions inside the buffer, that imply a large amount of short circuit power dissipation. We briefly discuss a clock buffer architecture that has a limited overhead in area and almost no penalty in power dissipation. Our architecture is shown in Fig. 8 for a simple two-stage buffer. The key intuition in this design is that the two large transistors in the output stage are never on at the same time, thus eliminating the short circuit dissipation. The clock skew is obtained by dimensioning the resistances of the two inverters in the first stage.

The transition that controls the output edge is always produced by the transistor in series with the resistance and it can be slowed down using large values $R1$ and $R2$. The penalty is in less sharp output edges (although the gain of the output inverter mitigates this effect) and in the presence of a period when both output transistors are off (the clock line is prone to the damaging effect of cross-talk). Both these effects are greatly reduced by adding another output stage (i.e., two inverters). The complete discussion of this buffer, its dimensioning and its comparison with standard implementation is outside the scope of this paper. However, our HSPICE simulations show that the power overhead of this buffer is negligible and the area overhead is very small.

5. Implementation and Results

The implementation of a program for peak current minimization depends on the availability of a tool that provides accurate current waveforms for circuits of sufficiently large size. Electrical simulators such as SPICE are simply too slow to provide the needed information. In our tool, pre-layout current waveforms are estimated by an enhanced version of PPP [16], a multi-level simulator specifically designed for power and current estimation [23] of digital CMOS circuits. PPP has performance similar to logic level simulators, it is fully compatible with Verilog XL and provides power and current data with accuracy comparable to electrical simulators. Input signal and transition probabilities for all the simulations are set to 50%.

The starting point for our tool is a mapped sequential network (we accept Verilog, SLIF and BLIF netlists). First, the sequential elements are isolated and current profiles are obtained. Alternatively, pre-characterized current models of all flip-flops in the library can be provided. The combinational logic between flip-flops is then simulated and its average current profile is obtained. The first simulation step assumes no skews.

Timing information is extracted from the network. Maximum and minimum delays are estimated with safe approximations (i.e., topological paths). Input arrival times and output required times are provided by the user. The uncertainties in pre-layout estimates are accounted for by specifying a safety margin of 15% on the delay values. The constraint inequalities are generated taking the margin into account. In this step several optimizations, such as those described in [6, 8], are applied to reduce the number of constraint inequalities. Data needed for the evaluation of the cost function are produced: the triangular approximations are extracted from the current profiles and passed to the GA solver [24].

The GA solver is then run to find the optimal schedule that minimizes the peak current. The initial population is generated by perturbing an initial feasible solution (zero skew). The GA execution terminates

Giovanni De Micheli is Professor of Electrical Engineering, and by courtesy, of Computer Science at Stanford University. His research interests include several aspects of the computer-aided design of integrated circuits and systems, with particular emphasis on automated synthesis, optimization and validation. He is author of: Synthesis and Optimization of Digital Circuits, McGraw-Hill, 1994, and co-author or co-editor of three other books. He was co-director of the NATO Advanced Study Institutes on Hardware/Software Co-design, held in Tremezzo, Italy, 1995 and on Logic Synthesis and Silicon Compilation, held in L'Aquila, Italy, 1986.

Dr. De Micheli is a Fellow of IEEE. He was granted a Presidential Young Investigator award in 1988. He received the 1987 IEEE Transactions on CAD/ICAS Best Paper Award and two Best Paper Awards at the Design Automation Conference, in 1983 and in 1993. He is the Program Chair (for Design Tools) of the 1996/97 Design Automation Conference. He was Program and General Chair of International Conference on Computer Design (ICCD) in 1988 and 1989 respectively.
nanni@stanford.edu