Fast Electrical Correction Using Resizing and Buffering

Shrirang K. Karandikar,[†] Charles J. Alpert,[†] Mehmet C. Yildiz,[‡]
Paul Villarrubia,[‡] Steve Quay[‡] and Tuhin Mahmud[‡]
[†]IBM Austin Research Laboratory [‡]IBM EDA

Abstract— Current design methodologies are geared towards meeting different design criteria, such as delay, area or power. However, in order to correctly identify the critical parts of a circuit for optimization, the circuit has to be electrically clean – i.e., slews on each pin have to be within certain limits, a gate cannot drive more than a certain amount of capacitance, etc. Thus far, this requirement has largely been ignored in the literature. Instead, existing methods which optimize delay are used to fix electrical violations. This leads to solutions that are unnecessarily expensive, and still leave violations that remain unfixed. There is therefore a need for an area-efficient strategy that targets the electrical state of a circuit and fixes all violations quickly. This paper explicitly defines "electrical violations" and presents a flexible approach (called EVE, the Electrical Violation Eliminator) for fixing these. Experimental results validate our approach.

I. Introduction

In an typical physical synthesis flow, placement is followed by electrical correction and various optimizations, such as gate sizing, buffer insertion and resynthesis, among others, as shown in Figure 1. After each step, critical parts of the design are determined using a static timing analyzer, and subsequent steps focus on these critical portions. Placement [1, 2] and optimization (gate sizing using techniques of logical effort [3–5] and Lagrangian relaxation [6] and buffering [7–10]) are well studied problems, both in academia and industry, and remain an area of active research. However, electrical correction has largely been ignored in the literature, though its importance is well recognized [8, 11–14], and industrial design flows include steps to address it.

Most optimization techniques rely on the correct operation of the timer, to guide them in the right direction. However, any timer, even a sophisticated one, can work correctly only if the design is in a good electrical state. For example, if capacitive loads are outside the range that a gate model has been characterized for, the timer will give results that do not reflect the true performance of the gate. Consequently, the timing analysis of an entire design can be wrong, and critical areas may not be correctly identified. In order to get meaningful results from a timer, these characterization issues need to be addressed. This is done by the "electrical correction" phase, whose role is to make sure that slew and capacitance violations are fixed.

The role of this step is to fix violations *quickly*. This will naturally need area, but electrical correction should minimize this area overhead, thereby reducing unnecessary power consumption and silicon real estate. The need for reducing area usage is obvious for area-constrained designs. However, even in designs where the total area may not be at a premium, local regions may be congested. Further, in delay-constrained designs, the area saving can be used by subsequent optimizations to improve the performance of critical regions.

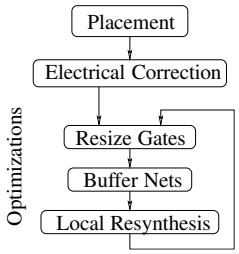


Fig. 1.: Implementation Flow

While the importance of this step has been recognized previously, the solution has been to use existing resizing and buffering methods, which optimize the delay of a circuit. This is expensive, since the solution used solves the wrong problem, and utilizes valuable silicon area while doing so. In todays high-performance area-constrained designs, this area is better managed for improving delay or reducing power consumption.

This paper presents a new approach that specifically addresses electrical correction. Our goal is to develop an efficient framework that fixes violations without unnecessary overhead, i.e, we are interested in the least cost solution that eliminates electrical violations. This approach, called EVE (Electrical Violation Eliminator) has been implemented under an industry toolset, and integrates several extensions that can be used, depending on the needs of the design.

II. Background

A. Electrical Violations

Timing analyzers utilize models for gate delays and slews, which are pre-characterized. Each gate is characterized with a maximum capacitive load that it can drive and a maximum input slew rate. The operation of the timer is valid only within these ranges. If these conditions are violated, timers usually extrapolate to obtain 'best guess' values. However, values calculated in this manner may very well be wildly inaccurate. This leads to the limits that define electrical violations. There are three principal "rules" that a design has to pass for it to be electrically clean, as follows.

Slew Limits These rules define the maximum slews permissible on all nets of the design. If the slew (defined here

1-4244-0630-7/07/$20.00 ©2007 IEEE.

as the 10%-90% rise or fall time of a signal, other definitions can be used as well) at the input of a logic gate is too large, a gate may not switch at the target speed, or may not switch at all, leading to incorrect operation.

Capacitance Limits These define the maximum effective capacitance that a gate or a macro input can drive. A large capacitance on the output of a gate directly affects its switching speed and power dissipation. Additionally, gates are typically characterized for a limited range of output capacitance, and delay calculation can be inaccurate if the output capacitance is greater than the maximum value.

Fanout Limits These rules limit the number of fanouts that can be driven by different gates. Critical nets can be limited to have lower fanouts. These limits are easily fixed, and are not addressed further in this work.

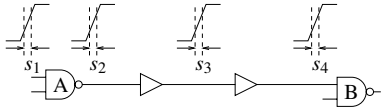
Violations of these three rules (referred to as slew violations, capacitance violations and fanout violations, respectively) taken together are called electrical violations. These limits are principally determined during gate characterization, but designers may choose to tighten these constraints further. For example, a tighter slew constraint can guardband the design against variability or noise [9]. High performance designs, such as microprocessors typically have much tighter slew limits than ASICs.

B. Causes of Slew Violations

Figure 2 shows the main causes of slew violations, and how these can be fixed. Consider a net having source gate A and sink gate B. The capacitive load seen by gate A is the sum of the interconnect capacitance of the net and the input capacitance of gate B. Assume that a signal with slew s_1 is applied at the input of gate A. Due to the load that it has to drive, the slew s_2 at the output of gate A may be more than s_1. Thus, one cause of degradation is the source gate not being capable of driving the load at its output. Next, even if the slew at the output of A, s_2, is within the specified limits, it could degrade as the signal traverses the net to the sink. Thus, at the sink, the signal could have an even larger slew of s_3. This is the second contribution to slew degradation.

(a) Slew degradation due to gate and interconnect

(b) Fixing slew violation by sizing the source



(c) Fixing slew violation by buffering

Fig. 2.: Causes of and fixes for slew violations

There are two main methods of fixing slew violations. First, the source gate of the net can be sized up, so that the new gate can drive the load present (as in Figure 2(b)). While this may fix violations on the net in question, the obvious disadvantage is that the problem has been moved to the input of the source gate, where the input nets now have larger capacitances. This may or may not create violations on the input nets.

Second, keeping the source at its original size, buffers can be inserted on the net in question, as shown in Figure 2(c). These isolate the load capacitance of the sink, and repower the signal on the net, so that slews are within the specified limits. Unlike resizing, this method does not affect the electrical state of any other nets, but the area overhead can be much higher. Additionally, the time required to determine where to best insert buffers, is much greater than the time required to resize a gate.

C. Capacitance and Fanout Violations

In this paper, the effective capacitance is approximated by the sum of the interconnect capacitance and sink capacitances. The causes of capacitance violations are similar to those of slew violations: sink and interconnect capacitance both contribute to the existence of a violation. The fixes, too are similar, using resizing and buffering. However, it is possible to have capacitance violations on a net that does not have slew violations, and vice versa. Therefore, both capacitance and slew violations have to be taken into consideration individually.

Capacitance violations are also related to fanout violations, since large fanouts can lead to high loads. Fanout constraints are artificial, but are imposed by designers as a guard band.

D. Unfixable violations

There are violations that cannot be fixed due to a number of reasons, such as constraints and limits that are too aggressive for the design, gates (such as registers) that cannot be resized, buffer blockages, or poor floorplanning. For example, consider the case of a primary input driving a sink, with a large blockage present right at the primary input. The length of the net connecting the primary input to its sink will guarantee a slew or capacitance violation. Even routing around the blockage may not be feasible, leading to an unfixed violation.

E. Verifying Correctness with the Timer

The timing analyzer that is used, EinsTimer [15], has two modes of operation, with respect to slew calculation and propagation. In the accurate mode, when determining the output slew of a gate, the *actual* input slew is used. However, at the beginning of physical design, gates are at arbitrary sizes, long wires are not yet buffered, and slew violations abound. In this scenario, actual slew values at gate inputs are easily outside the limits for which gate models have been characterized. Hence, there is also a less accurate mode, where a *default* input slew is used, in order to determine the output slew rate. When the circuit is in an electrically poor state, this mode of operation is preferred, in comparison to the accurate mode, where calculated values are quite possibly meaningless.

In the context of fast electrical correction, using the default slew is also advantageous, since any change in the design (gate resizing or buffer insertion) has a local effect on the timing graph, and is easy to calculate. In contrast, in the accurate

mode, a change in gate size, affecting the output slew value has to be propagated in the output cone of logic, which can be computationally expensive.

III. The Iterative Approach

The literature on techniques for electrical correction is sparse, as most works focus on optimization for delay. However, one approach that implements an iterative flow is [16,17]. This is a combination of resizing gates and adding buffers to nets, in order to fix violations. Integrating these optimizations into a single pass is difficult, and hence multiple iterations over the entire design are necessary, with each pass consisting of either resizing or buffering.

The first step is gate sizing to fix any slew and capacitance violations. This gate sizing is not timing driven, but instead tries to match the correct gate size with the load being driven. i.e., given the input slew, output load, and the required output slew, it determines the best gate size. Since nets with violations are considered in this step, the usual result is gates being sized up. Larger gates have greater drive capabilities, and therefore can drive larger loads. However, this results in an increased capacitance at the inputs of the sized gates, and therefore can introduce new violations at the inputs. There are possibly quite a few violations that cannot be fixed using even the largest gate sizes available; buffer insertion is used to fix these. In order to conserve runtime, the most aggressive buffering algorithms may not be used, and this (and other reasons) can lead to violations that are still unfixed. Additionally, when there are loops in the circuit, fixing violations on one net can lead to creating violations on nets that were previously thought fixed. To address these remaining violations, another round of resizing and buffering is used. Subsequent calls can be made more aggressive than before, in the hope that the remaining violations can be fixed. This process is iterated, till either all violations are fixed, or the runtime becomes excessive.

The most important drawback of this approach is that sizing and buffering used to fix violations are applied sequentially, with no communication, or indeed, knowledge of each others' capabilities. Thus, each pass of resizing or buffering tries to fix the violations that it sees, and assumes that the the other will be able to handle the violations that it cannot fix. Thus, if resizing is applied to a net to fix a slew violation on a sink, it may decide that buffering is the best solution, for a variety of reasons. However, in the next pass, when the net is passed to the buffer insertion routine, there may be conditions that prohibit the insertion of buffers, such as blockages. Subsequent passes of resizing and buffering are then needed with different settings, to overcome this situation, and there is no guarantee that any of these passes will fix the existing violation.

As will be shown in section Section V, this approach is unable to fix many violations. During gate sizing, it typically only sizes gates that have violations, and therefore sizes them up. Combined with buffer insertion, this can lead to circuits that have up to 27% larger area than the initial versions. It is this behavior that EVE is intended to rectify.

IV. Electrical Violation Eliminator

In this section, we present the framework developed for efficiently fixing violations, called EVE. We first present the basic framework, which carries out resizing and buffering on a net by net basis. Analysis of unfixed violations using this framework suggests numerous enhancements, which are presented next. The basic framework is flexible, and allows for easy integration of these enhancements. Rather than multiple passes over a design, alternating between resizing and buffering, this framework integrates the selection of the two optimizations, allowing for the use of the correct optimization in a single pass over the design. Since only operations that are needed for fixing violations are used, there is less overhead in terms of area and runtime.

A. Algorithmic framework for the basic approach

Applying resizing and buffering multiple times to a circuit can lead to poor results. Our approach in EVE is to selectively apply these optimizations on a net-by-net basis. We select nets in topological order, from outputs to inputs, and on each net, carry out the following operations.

- If there are no violations on the net, then the source (driving) gate is sized *down* as much as possible, without introducing new violations.
- If slew violations exist on the net, the source gate is sized up as necessary, to fix the violations.
- If the previous step (resizing to fix violations) does not succeed, the net is buffered.

The rationale of this approach is as follows. First, nets are processed in output-to-input order, so that any side-effect of resizing gates only impacts the input nets, which are yet to be processed. Second, sizing gates down when possible has multiple benefits. Area is recovered and reducing the load on input nets potentially removes violations that may exist, or reduces their severity. The area salvaged in this step is better used for improving delay on critical paths of the circuit. Finally, if resizing cannot fix a violation, buffering is used to fix the net. Since buffering is the last resort, this optimization can be as aggressive as required, which is used to our advantage as shown later. This order (resizing followed by buffering) is also advantageous from a runtime standpoint, since buffering a net is computationally more expensive than simply sizing the source gate.

Our approach to gate sizing is straightforward. Given an input slew rate and output load, we iterate through all available sizes, and select the smallest gate size that can deliver the required output slew. Buffering is performed using the slew-based buffering approach proposed in [18], with a library of 1 buffer and 3 inverters for most designs. The algorithm selects the minimum buffer solutions such that the slew constraints are satisfied. The lack of granularity in the buffer library makes the potential to resize the buffer gates possible. Of course, a more fine-grained library can be used, but the extra runtime defeats the purpose of fast electrical correction.

B. Enhancements

The basic framework presented above is flexible, and lends itself to multiple refinements, as follows.

Recursive treatment of nets The run time of the basic approach is dominated by the time spent in buffering, which is quadratic in the size of the buffer library used. In order to improve this runtime, a small buffer library is used for buffering, and the basic approach is applied recursively to the new nets that have been created. Since EVE addresses violations on a net by net basis, it is relatively straightforward to determine which nets have been added, and correctly size the newly added buffers. In contrast, the iterative approach would require multiple passes of the entire circuit.

Rip up existing inverters In the initial stages of the design, the circuit can have a number of inverters that have been added in order to deal with signal polarity requirements. These inverters are usually added at the logic synthesis stage, without any knowledge of the physical layout of the design. However, once the design has been placed, it is possible that the positive and negative polarity sinks are clustered together. A buffering approach that treats the positive and negative sinks separately will treat this problem as two different instances, with each net being buffered separately. leading to a sub-optimal solution. Ripping up inverters and re-implementing the entire tree taking into account sink polarities can result in a solution that uses fewer buffers and less wire length.

In the iterative flow, this would require an additional pass over the circuit, so that inverters that can be ripped up are correctly identified. In EVE, the first time a net with inverters is encountered, it is known that it can be ripped up, and therefore can be seamlessly handled with other enhancements.

Handling loops Most circuits have loops, with registers breaking cycles. This does not create problems in most cases, since registers can be sized only in a small range. However, for a few circuits this can lead to the introduction of new violations. For example, consider the situation in Figure 3. In the output to input traversal, assume that the order in which gates are processed is the inverter, nand and finally the latch. However, if the latch is sized up, the load on the inverter, which has been previously processed increases as well, and if the increase is large enough, can lead to a new slew or capacitance violation. This is handled within the EVE framework as follows. Each time a gate is sized up, we check if the inputs of the gate were previously processed. If so, we ensure that no new violations have been introduced, and fix those that have. Typically, the violation is small, and a small adjustment to the gate size is all that is required. If necessary, the inputs of the gate are resized as well. This is easily done within the EVE framework, since it is known that new violations are because of the resizing. In the iterative flow, this is difficult to keep track of. Multiple passes are not the answer, since it is not known whether a violation is because of loops, because the previous optimization(s) failed, because of blockages, or due to some other reason.

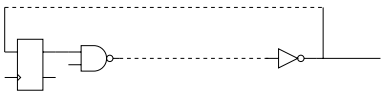


Fig. 3.: Logic Loop

Blockage avoidance Designs can have areas that are off limits to buffers. In this case, a regular buffering approach will fail, due to lack of insertion points for buffers. Existing approaches for blockage avoidance [19, 20] and dealing with difficult instances have higher run times. In the EVE framework, when processing a net that has been visited for the first time, the only reason the basic buffer insertion routine is unsuccessful is due to blockages. Blockage avoidance can then be brought into play.

Select best of resizing and buffering The basic EVE approach can be modified as follows. If a violation can be fixed by either resizing or by buffering, the lowest cost solution is naturally preferred. This can be easily done in the EVE framework, by resizing, measuring the area increase, resetting to the original gate and then buffering. The best solution can then be selected. Consider a woefully small gate of size a driving a net. Sizing it up to a large size, say f, could fix violations. Conversely, a single buffer may be able to fix the violations as well. If the difference in area between a and f is greater than the area of a buffer, the buffer is preferred, else resizing the source is preferred. If area is at a premium in a design, this mode can be used. The trade off is in the runtime required for this analysis.

Minimum perturbation mode EVE can be used in different stages of physical synthesis. In later portions of the flow, a number of optimizations have been applied to improve specific aspects of the design, which may re-introduce a few electrical violations (especially after an incremental placement). We can use EVE to only target these violations, not touching the rest of the design.

Using the timer As mentioned before, default slews are used at the inputs of gates, in order to determine output slews. A more accurate analysis of the design is possible if actual slews are used. In this mode of EVE, the timer is used to determine the actual slews. This mode is much slower, since every change triggers a re-computation of slews and arrival times in the vicinity of the change. This mode is used when only a small number of violations remain.

In the context of the above enhancements, it is interesting to note the advantages of the EVE framework. A few of the enhancements are not unique to EVE, and are available to the iterative approach as well, such as blockage avoidance. However, there is no easy way of determining when to use these features. By its very structure, EVE can easily incorporate recursive treatment of nets, and ripping up inverters and reimplementing buffer trees *as and when* needed. The iterative approach processes the entire circuit in each iteration, and would

Circuit	Initial State		Iterative Flow		EVE-RL	
	Slew	Cap	Slew	Cap	Slew	Cap
Ckt1	30021	2341	372	554	0	220
Ckt2	59789	3834	41	1	3	0
Ckt3	39526	1207	206	0	0	0
Ckt4	62086	3190	541	0	81	0
Ckt5	49954	2473	50	0	0	0
Ckt6	96168	4358	1171	1	17	0
Ckt7	172840	4237	1101	21	58	2
Ckt8	199861	15437	3977	47	7	0
Ckt9	210413	8345	11936	36	36	0
Ckt10	291700	17202	1967	74	96	2
Ckt11	323330	21971	336	4	10	1
Ckt12	543537	42551	1712	192	36	204
Ckt13	1004957	37657	9233	28	13	51
Total Violations Unfixed			32643	958	357	480

TABLE I: Comparing the Iterative Approach and EVE-RL

therefore need much more complicated bookkeeping to achieve the same end. Finally, the designer can make the decision on how to use EVE, keeping in mind his particular constraints, such as number of violations, available area, available CPU time, power budget, etc., and trade off the quality of solution with run time.

V. Results

We use a set of industrial designs in order to determine how EVE performs in comparison with the iterative approach. We apply electrical correction at the beginning of physical synthesis, after the placement step, but before any gate sizing or buffer insertion. Consequently, the design is in an electrically poor state. This is reflected in the number of initial violations, presented in the 'Slew' and 'Cap' columns under 'Initial State' in Table I.

A. Comparing with the iterative approach

In Table I, we first compare the improvement in the electrical state of our benchmark suite obtained using the iterative approach, versus using EVE-RL. This version of EVE includes recursing on all nets created by adding buffers, and handling loops, as described in Section IVB.

The iterative flow employs a number of passes of gate resizing and buffer insertion with a variety of different options, so as to progressively fix more violations in successive passes. The number of violations that remain after applying this approach are presented in the columns under 'Iterative Flow'. The last two columns list the number of remaining violations after applying EVE-RL to the benchmark circuits.

The contrast between the iterative approach and EVE-RL is immediately obvious from Table I. While it may seem that there is a drastic reduction in violations using the iterative approach, there are still 32,643 remaining slew and 958 remaining capacitance violations. After EVE-RL has processed the benchmark circuits, only 357 slew (a reduction of nearly $100x$) and 480 capacitance (nearly half as previous) violations remain.

It is usually the case that improvements such as those seen in Table I require a large overhead in either runtime or area. However, recall that EVE only uses the optimizations necessary to fix violations, and makes the correct decisions *once*. This leads to a much more area-efficient solution for fixing violations, and at the same time requires less time, since multiple passes over the circuit are avoided. The cost of fixing violations using each flow is presented in Table II. For both the iterative approach and EVE-RL, the area increase (or, in some cases, even decrease!) is as shown. The percentage change in area with respect to the original circuit is presented in columns 4 and 7. Column 9 presents the difference in area between the iterative approach and EVE-RL. As can be seen, EVE takes 9.05% less area on average, and fixes more violations than the iterative approach.

On average, EVE-RL runs in 3.46% less time than the iterative approach, which itself is quite fast. Both approaches can process even the largest design, which has more than a million nets. This is another benefit of using techniques specific to electrical correction – sophisticated gate sizing approaches can easily take orders of magnitude more runtime. While these will perform admirably well for optimizing delay, their application to electrical correction wastes runtime, while solving the wrong problem. Logical effort is fast, but is not accurate when dealing with fanouts, and once again, only analyzes delay and ignores slew rates.

B. The basic EVE approach and enhancements

Table III breaks out the effects of the primary enhancements of EVE-RL as described in Section IV. Results of using the basic EVE approach are shown in columns under the heading 'EVE'. The effect of recursively resizing new buffers added is shown under the columns titled 'EVE-R'. Note that this is different from applying a resizing pass on the entire circuit, since the new buffers are a relatively minuscule subset of the circuit. In addition, by sizing the buffers newly added, a lot of area can be recovered. We also resize the source of the original net, and due the buffer insertion, it is usually sized down. This has the dual benefit of once again saving on area, and reducing the load on its inputs. The runtime overhead of this recursion is negligible, but the number of violations are significantly reduced. Next, the result of dealing with loops can be seen under the columns titled 'EVE-RL'. The area and runtime increase, but the number of violations decrease still further.

The usefulness of the other enhancements described in Section IV largely depends on the circuit being corrected. For example, in our test circuits, inverters constitute a small portion of the design. Ripping up and reimplementing trees that have inverters therefore results in relatively small area savings, but a higher runtime overhead. However, in scenarios where there are a larger number of inverters, this feature can be used. Similarly, minimum perturbation mode, and using the timer to determine the best solution are more appropriately used in later stages of the design flow.

VI. Conclusion and Future Directions

This paper presents the first work (to our knowledge) that focuses on optimization for electrical correction. It describes an integrated buffering and gate sizing framework called EVE that significantly reduces the number of violations, as well as achieving a reduction in the area required. This benefits the power consumption of the design, as well as allowing slack optimizations greater freedom in improving the delay along criti-

Circuit	Initial		Iterative Flow			EVE-RL			Δ Area	Δ Time
	# Nets	Area (x10^3)	Area	Change	Time	Area	Change	Time	(%)	(%)
Ckt1	50,166	631.7	753.6	19.30	104.89	525.9	-16.75	113.87	30.21	-8.56
Ckt2	55,686	531.7	676.2	27.18	165.72	526.3	-1.02	125.00	22.17	24.57
Ckt3	61,087	989.4	1046.0	5.72	108.88	986.0	-0.34	81.47	5.74	25.17
Ckt4	67,043	894.2	1058.0	18.32	213.10	902.8	0.96	209.94	14.67	1.48
Ckt5	67,926	962.0	1008.0	4.78	75.77	979.1	1.78	53.83	2.87	28.96
Ckt6	192,657	15110.0	15270.0	1.06	296.74	15110.0	0.00	266.16	1.05	10.31
Ckt7	242,757	3454.0	3659.0	5.94	496.51	3391.0	-1.82	339.86	7.32	31.55
Ckt8	289,430	26330.0	26810.0	1.82	624.12	26420.0	0.34	655.24	1.45	-4.99
Ckt9	370,203	9671.0	10040.0	3.82	1155.96	9023.0	-6.70	1594.14	10.13	-37.91
Ckt10	383,221	22840.0	23880.0	4.55	882.88	22880.0	0.18	737.29	4.19	16.49
Ckt11	559,398	54580.0	55490.0	1.67	1217.29	54200.0	-0.70	1243.73	2.32	-2.17
Ckt12	984,886	34890.0	36740.0	5.30	2195.34	34030.0	-2.46	2245.27	7.38	-2.27
Ckt13	1,246,820	32710.0	34720.0	6.14	2807.39	31890.0	-2.51	3865.67	8.15	-37.70
Average Improvement									9.05	3.46

TABLE II: Iterative versus EVE-RL : Total Area and Runtime

Circuit	EVE				EVE-R				EVE-RL			
	Slew	Cap	Area	Time	Slew	Cap	Area	Time	Slew	Cap	Area	Time
Ckt1	0	231	529.7	94.01	0	220	525.9	96.06	0	220	525.9	113.87
Ckt2	3	10	539.3	98.52	51	12	526.3	101.65	3	0	526.3	125.00
Ckt3	444	0	992.1	60.58	0	0	986.0	69.89	0	0	986.0	81.47
Ckt4	385	0	919.8	179.64	269	0	902.7	189.08	81	0	902.8	209.94
Ckt5	0	1	988.2	36.81	0	0	978.7	41.14	0	0	979.1	53.83
Ckt6	1431	0	15130.0	211.26	46	0	15110.0	250.57	17	0	15110.0	266.16
Ckt7	1473	12	3422.0	281.32	302	3	3391.0	309.20	58	2	3391.0	339.86
Ckt8	4781	28	26490.0	533.43	33	1	26420.0	615.16	7	0	26420.0	655.24
Ckt9	7469	31	9094.0	1313.12	733	0	9023.0	1369.87	36	0	9023.0	1594.14
Ckt10	2471	24	23020.0	552.51	146	2	22880.0	700.73	96	2	2288.0	737.29
Ckt11	1502	15	55430.0	873.57	230	1	54200.0	934.86	10	1	54200.0	1243.73
Ckt12	2673	229	34240.0	1688.42	36	204	34030.0	1787.75	36	204	3403.0	2245.27
Ckt13	12851	42	32220.0	2504.23	37	52	31890.0	2833.82	13	51	31890.0	3865.67
Total	35483	643			1883	495			357	480		

TABLE III: Comparing EVE Basic, EVE-R and EVE-RL

cal paths. To a large extent, the benefits of EVE are due to its inherent simplicity. More sophisticated techniques for electrical correction may exist, but they are likely to be more expensive.

EVE can be used in different stages of the design flow. Different optimizations may change gate sizes, add buffers, or resynthesize parts of the circuit. After re-placing the design, new violations may be introduced. An approach such as EVE allows for a quick pass that fixes new violations without disturbing the rest of the design.

One enhancement to EVE is based on the observation that paths optimized for delay have sharp signal transition times. Hence the following strategy can help improve critical paths of the design. After an initial pass where the design has been electrically corrected, a tighter slew limit is assigned to critical regions. The next pass of EVE using these updated slew limits still works on fixing violations, but since it uses the tighter slew limits, the result is a reduced worst slack.

VII. REFERENCES

[1] G.-J. Nam *et al*. The ISPD2005 Placement Contest and Benchmark Suite. In *ACM ISPD*, pages 216–220, April 2005.

[2] A. B. Kahng *et al*. APlace: A General Analytic Placement Framework. In *ACM ISPD*, pages 233–235, April 2005.

[3] R. F. Sproull and I. E. Sutherland. Theory of Logical Effort: Designing for Speed on the Back of an Envelope. In *IEEE Adv. Research in VLSI*, pages 1–16, 1991.

[4] P. Rezvani *et al*. LEOPARD: A Logical Effort-based fanout OPtimizer for ARea and Delay. In *IEEE/ACM ICCAD*, pages 516–519, November 1999.

[5] I. Sutherland *et al*. *Logical Effort: Designing Fast CMOS Circuits*. Morgan Kaufmann, San Fransisco, CA, 1999.

[6] C.-P. Chen *et al*. Fast and Exact Simultaneous Gate and Wire Sizing by Lagragian Relaxation. In *IEEE/ACM ICCAD*, pages 617–624, November 1998.

[7] L. P. P. P. van Ginneken. Buffer Placement in Distributed RC-tree Networks for Minimal Elmore Delay. In *IEEE Int. Symp. on Circuits and Systems*, pages 865–868, May 1990.

[8] C. J. Alpert *et al*. Buffer Library Selection. In *Proc. IEEE ICCD*, pages 221–226, 2000.

[9] C. J. Alpert *et al*. A Practical Methodology for Early Buffer and Wire Resource Allocation. In *Proc. IEEE/ACM DAC*, pages 189–194, 2001.

[10] W. Chen *et al*. Simultaneous Gate Sizing and Fanout Optimization. In *IEEE/ACM ICCAD*, pages 374–378, November 2000.

[11] D. S. Kung. Timing Closure for Low-FO4 Microprocessor Design. In *Proc. IEEE/ACM DAC*, pages 265–266, 2004.

[12] K. L. Shepard *et al*. Design Methodology for the High-Performance G4 S/390 Microprocessor. In *Proc. IEEE ICCD*, pages 232–240, 1997.

[13] C. J. Alpert *et al*. Minimum Buffered Routing with Bounded Capacitive Load for Slew Rate and Reliability Control. *IEEE Trans. on CAD of ICs and Systems*, 22(3):241–253, March 2003.

[14] M. Augarten. Evaluating ASIC Reuse. *EEtimes*, 2000.

[15] L. Stok *et al*. BooleDozer: Logic Synthesis for ASICs. *IBM Journal of R. & D.*, 40(4):407–430, 1996.

[16] L. Trevillyan *et al*. An Integrated Environment for Technology Closure of Deep-Submicron IC Designs. *IEEE Design & Test of Computers*, 21(1):14–22, January 2004.

[17] P. J. Osler. Placement Driven Synthesis Case Studies on Two Sets of Two Chips: Hierarchical and Flat. In *ACM ISPD*, pages 190–197, 2004.

[18] S. Hu *et al*. Fast Algorithms for Slew Constrained Minimum Cost Buffering. In *Proc. IEEE/ACM DAC*, pages 308–313, 2006.

[19] C. J. Alpert *et al*. Steiner Tree Optimization for Buffers, Blockages and Bays. *IEEE Trans. on CAD of ICs and Systems*, 20(4):556–562, April 2001.

[20] C. J. Alpert *et al*. Buffered Steiner Trees for Difficult Instances. In *ACM ISPD*, pages 4–9, 2001.