Synthesis Methodology for Built-In At-Speed Testing

Yinghua Li, Alex Kondratyev*, and Robert Brayton
U. C. Berkeley and *Cadence Berkeley Labs

Abstract

We discuss a new synthesis flow, which offers the ability to do easy delay testing almost free in terms of its impact on speed and area compared to corresponding implementations with standard cells. The methodology uses matched delays in pre-charged PLAs and bundled routing to produce a completion signal, which is guaranteed to lie on all critical paths. We give a non-delay testing method for ensuring matched delays are correct, i.e. that all completion signals arrive after their corresponding data signals. The design margins of the matched delays can be small since they are internal to the PLAs, which are regular structures and therefore more predictable.

1. Introduction

As chip dimensions continue to shrink, the variations in process parameters such as lengths, widths, L_{eff}, V_{th} and environment parameters such as temperature, V_{dd}, etc. are predicted to increase dramatically [1]. Using worst-case conditions (usually at 3σ) on all the parameters will be much too conservative. It has been estimated for ASICs that these margins are already 50-100% [3] of the actual delay. Although grading chips by speed (binning) may help in reducing margins, in ASIC designs, testing for delay is an expensive proposition with very limited coverage. Two-vector test sequences are required to stimulate slow paths, and special circuitry (beyond LSSD latches) is needed to apply the test at speed and scan out the computed results to check if an error occurred. Therefore binning is applied mostly to microprocessors.

Another method to reduce delay uncertainty and tighten delay margins is to use statistical timing analysis [9]. However, accurate process statistics are difficult to obtain. and even then, it has been estimated that such methods can only improve the delay margins about 10-15% [2].

Razor [5] is an architectural technique, which applies mainly to microprocessors. It uses shadow latches with a delayed clock to store late arriving signals. There are some issues with metastability as well as its applicability to ASICs, where the number of critical paths may be significant.

Another possibility is to create a completion signal using either delay insensitive encoding (like dual-rail) [10] or using matched-delay computations (as used in micro-pipelining [11]) where the matched-delay line is guaranteed to be slower than any associated data computation. Experiments with dual-rail encoding, indicates about a 2 x penalty in total area over its single rail counterpart [4]. Using a single-rail implementation, such as normal standard cells, with matched-delay lines is problematic since it is difficult to guarantee the delay inequality between the data computations and the matched-delay lines. This is especially difficult because of wiring routes and hard-to-predict cross-talk effects.

In this paper, we propose a method by which logic modules are synthesized exploiting single rail encoding and matched delays, but using a multi-level network of PLAs rather than standard cells. The PLAs are pre-charged NOR-NOR structures, chosen because of their speed and efficient area usage. The matched-delay signals come in two varieties, those inside PLAs and those outside, between PLAs. For the inside matched-delay signal, we rely on the regular structure of a PLA to accurately predict a delay margin. The matched-delay line inside the PLA is actually at no cost since it is already needed to control the relative timing of the two NOR structures; a NOR should not begin computation (discharge) until all its inputs are ready. For the outside matched delay, we use a "bundled routing" scheme with the matched-delay line interior to a set of corresponding data lines. We prove that the delay of the matched-delay line in the bundle is always greater than for any data wire in the bundle *under all cross-talk conditions*.

At the primary outputs (PO) of a logic module, a completion signal is formed by ANDing all matched-delay lines of any PLA that feeds a PO. This signal is compared with the clock edge, and if it is 0 when the clock edge arrives, an *error* signal is set to 1, indicating that the module was too slow. This signal can be used in a number of ways; for easy delay testing, binning chips, or adjusting V_{dd} to reduce power.

We will show how all the matched delay inequalities can be tested using normal single-vector (non-delay) testing hardware and a modified version of ATPG for single stuck-at testing. After this testing, for any passing circuit, all the matched delay inequalities are satisfied for all slow modes. Further, this implies that the output completion signal lies on all critical paths. Since such paths can be only along matched-delay lines, the number of delay failure modes for any module is drastically smaller than for its standard cell implementation. With process and environmental variations increasing dramatically, this should provide a much more rugged method for synthesizing logic since there are many fewer modes of failure.

Some advantages of the error signal for delay testing are:

1. delay testing can be done with single vector inputs, because the pre-charge state always serves as the initial vector.
2. it is not necessary to observe the data computed by the test vector input since only the error signal needs to be observed to infer a delay fault; hence delay fault coverage is very high, unlike regular delay fault testing.

In terms of implementation efficiency, we show on a set of benchmark modules that our synthesis process is, on average, within 1% of the corresponding standard cell (SC) implementations in both delay and area. This experiment was done, using 180nm technology, by comparing a commercial SC tool against our PLA-oriented tool. In both cases, the modules were synthesized using typical-case parameters, targeting high-speed implementations.

The paper is organized with Section 2 discussing the overall structure of the modules, the implementation of the pre-charged PLAs, and the delay inequalities, which are required for correct functionality. Section 3 presents bundled routing and proves the theorem that the matched delay (*done*) signal is always greater than the delay of any data line. Section 4 details our synthesis flow including the algorithm used for decomposition and clustering into PLAs. Section 5 discusses the physical design aspects including block placement and our bundled (global) routing algorithm.

Section 6 presents our testing methodology, which ensures that the necessary delay inequalities hold. Section 7 gives the details of the experiments comparing PLA implementations with standard cells, and an experiment on a sufficient delay margin required for the matched delay in the first NOR structure of a PLA. Section 8 concludes.

2. Network of PLAs

The structure in which the networks of PLAs will be used is illustrated in Figure 1.

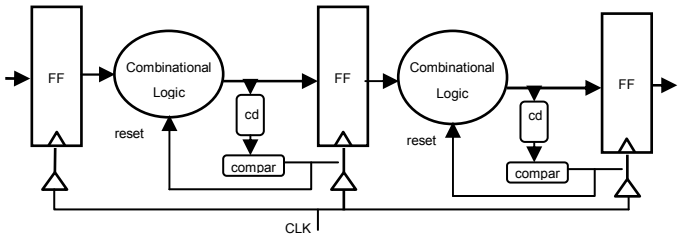

Figure 1: *Synchronous circuit using completion detection.*

The clock *CLK* is the normal clock used in a regular implementation. The *CD* (completion detection) signal is the result of all the matched delays lines (*done* lines of all PLAs producing a PO of the module). *CD* is pre-charged to 0 and rises to 1 when the PLA computation is completed. It is compared with the clock edge and produces *error* = 1 if ever *CLK* rises to 1 but *CD* is still 0. The combinational logic blocks CL are implemented as a network of bundle-routed pre-charged PLAs.

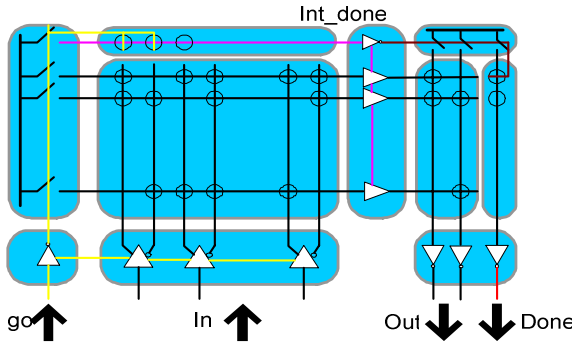

Figure 2: *NOR-NOR structure dynamic PLA*

Figure 2 shows the details of a single NOR-NOR dynamic PLA structure, where the AND and OR planes of the PLA are both implemented by dynamic NOR gates. Each plane needs a signal for controlling precharging transistors and input buffers. This signal should always come later than any input data signal in order to make a correct evaluation. The controlling signal for the OR plane (*int_done*) is obtained by a matched delay line for the AND plane. Its delay is designed so that it arrives to open the intermediate buffers between the two NOR arrays only after all NOR gates of the AND plane have completed their computations. Only then does the second NOR structure (OR plane) begin evaluation. The OR structure produces an extra output, which is the *done* signal output (right-most column) of the PLA.

Figure 3 shows a network of PLAs (interconnected with bundled routing). Note that each PLA produces a *done* signal, which is routed to all of its fanout PLAs. At the input of any PLA, all the *done* signals of its fanins are ANDed to form a *go* signal for the PLA. The inverse of this controls the pre-charging of the PLA through PMOS transistors, shown on the left side of Figure 2. The *go* signal controls the opening of the buffers at the input data signals. The buffers must not be opened until the correct data values from the inputs have arrived. That is why all input *done* signals are ANDed to form the PLA *go*.

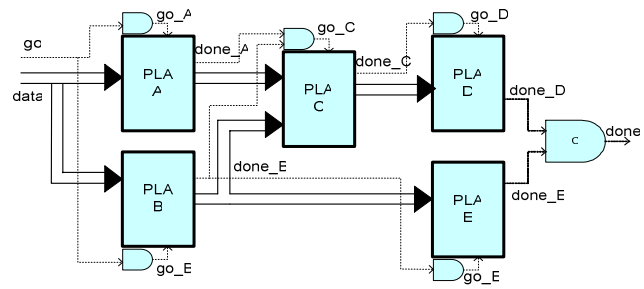

Figure 3: *Architecture for bundled data communication*

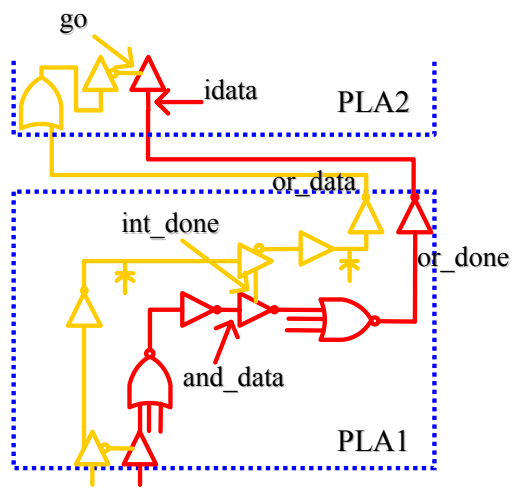

Figure 4. *Interconnection of two PLAs*

Figure 4 shows symbolically the signals on which the relative delays have to be guaranteed to obtain correct functionality. These are

1. $D_{\text{int\_done}} \geq D_{\text{and\_data}}$, shown in PLA1, and
2. $D_{\text{or\_done}} + D_{\text{wire\_done}} + D_{\text{go}} \geq D_{\text{or\_data}} + D_{\text{wire\_data}}$,

which compares the arrival times of the signals at the input to PLA2. Delay Inequality 1 relies on the internal matched delay of a PLA and arises from the requirement that the OR array should not begin evaluating until all data signals are valid. Since this is a relative delay requirement between signals internal to the same regular structure, it is easier to design for this with a relatively tight margin. Nevertheless, to absolutely guarantee this, we will test for this condition using the testing methods described in Section 6. Inequality 2 concerns delays along wires and would not exist if dual-rail encoding were used. However, with single-rail encoding, a matched delay along the routes to the fanouts is required, and the delay-matching requirement is Inequality 2. Note that the implementation of the *go* signal at the input of PLA provides some extra delay margin, D_{go} , here. In Section 3 it is proved that $D_{\text{wire\_done}} \geq D_{\text{wire\_data}}$ is guaranteed by our bundled routing scheme *under all cross talk*

conditions. Therefore, we need to design for $D_{\text{or\_data}} - D_{\text{or\_done}} \le D_{\text{go}}$, which can be tightly controlled in the regular structure PLA1. Again, we will design for this, but to absolutely guarantee this, we will test it with the strategy described in Section 6.

3. Bundled Routing Structure

A *bundle* of signals is defined to be the set of all outputs of a single PLA, including its *done* signal. Thus, each PLA has a single output bundle. Bundled routing refers to the fact that a bundle is routed as a single entity, except that if a data signal is not needed by a fanout PLA, then that data signal is not routed there. Thus, as the bundled net meets a Steiner point, the number of signals in the bundle may change. At the sink-points of the bundled net, i.e. each fanout PLA, the signals in the bundle are just the *done* signal and the *data* signals that are required by that PLA. Along any route from a PLA to a fanout, the done signals and data signals are side-by-side using the same route and the same wiring layers. This guarantees that the nominal delay (without cross-talk effects) along *done* and any data wire in the bundle will satisfy $D_{\text{nom\_wire\_done}} \ge D_{\text{nom\_wire\_data}}$. Also, since *done* is routed to all fanouts while a *data* signal may go to only a subset of fanouts, the data net may see less capacitive loading than *done*. This only strengthens the above inequality.

Cross-Talk effects: To insure that cross-talk is not a factor in Delay Inequality 2, we use two mechanisms,

1. *done* is pre-charged high while *data* is pre-charged low,
2. *done* is made internal to its bundle.

Therefore, *done* has either two *data* signals as its neighbors or a *data* and a shield line as its neighbors. Thus, the neighbors of *done* are known and cross-talk can only slow *done* down during execution since its neighbors are changing in the opposite direction. On the other hand, data signals can be sped up by other data signals, but can be slowed down only if next to *done*. However, in that case *done* is slowed down by at least the same amount as any data, since each is slowing the other down by the same amount. *done* can be slowed down even more by having two rising neighbors.

Theorem: *done* is slowed down more that *data* under all cross talk conditions, i.e. $D_{wire\_done} \ge D_{wire\_data}$.

Proof: We use the fact that $D_{\text{nom\_wire\_done}} \ge D_{\text{nom\_wire\_data}}$. The proof is by case analysis. We show that the slowest *data* case is always faster than the fastest *done* case.

slow data:

1. *data* has no falling neighbor: $D_{\text{wire\_data}} = D_{\text{nom\_wire\_data}}$
2. *data* has one falling neighbor (*done*): $D_{\text{wire\_data}} = D_{\text{nom\_wire\_data}} + \delta$
3. other cases can only speed up *data*

fast done:

1. *done* has no rising neighbors: $D_{\text{wire\_done}} = D_{\text{nom\_wire\_done}}$. This corresponds to 1 above since in this case *data* has no falling neighbors. Hence, $D_{\text{wire\_data}} = D_{\text{nom\_wire\_data}}$, and combining with $D_{\text{nom\_wire\_done}} \ge D_{\text{nom\_wire\_data}}$, we have $D_{wire\_done} \ge D_{wire\_data}$.
2. *done* has one rising neighbor: $D_{\text{wire\_done}} = D_{\text{nom\_wire\_done}} + \delta$. This can be compared with case 2 above since *data* can have at most one falling neighbor. Thus $D_{\text{wire\_data}} \le D_{\text{nom\_wire\_data}} + \delta$ and combining with $D_{\text{nom\_wire\_done}} \ge D_{\text{nom\_wire\_data}}$, we have $D_{wire\_done} \ge D_{wire\_data}$.
3. *done* has two rising neighbors; $D_{\text{wire\_done}} = D_{\text{nom\_wire\_done}} + 2\delta$. This is slower than any possible *data*.

QED

Note that we assume that all wires in a bundle have the same set of parameters at any cross-section of the bundle. However, parametric variations along the length of the net are allowed and can affect total delays but not the relative delays.

4. PLA Synthesis Flow

The synthesis flow for constructing a network of PLAs is:

1. Optimize the network using a technology independent procedure.
2. Decompose individual node functions into smaller ones such that each satisfies a given upper bound on its size.
3. Cluster the nodes to form PLAs, upper-bounded by given size parameters, and such that the resulting PLA network is acyclic.
4. Remove redundancies in the resulting network of PLAs (the network can be treated as a regular network of NOR gates with a classical redundancy removal algorithm being used).
5. Place the PLAs using special block placement with estimated bundled wiring congestion.
6. Globally route the bundled nets.
7. Post-process to improve congestion and delay.

Algorithm 1 Cluster a Circuit into PLAs

```
C = simplify_network(C);
InitializeHeap(clusterPair);
while(HeapNotEmpty(clusterPair)&&
        ((HeapMaxKey(clusterPair)>0||NumPLA(C)>Limit))
{   (p1,p2) = HeapGetMax(clusterPair);
    pnew = cluster(p1,p2);
    Espresso(pnew);
    RemoveRelatedPairs(clusterPair,p1);
    RemoveRelatedPairs(clusterPair,p2);
    AddClusterablePairs(clusterPair,pnew);
}
```

Figure 5. *Clustering Algorithm*

We start with normal technology independent synthesis using a logic synthesis system like SIS and optimize the network, e.g. using a script like SIS's *script.rugged*. Then, each node is decomposed into sub-nodes, which are smaller than a given bound. This is the starting point of a simple greedy clustering algorithm. All pairs of nodes (p_1,p_2) are put in a heap, ranking them with a heuristic, measuring their desirability to be merged into a single PLA. If a pair would form a PLA violating the given bounds on the PLA sizes, that pair is not considered. This measure uses the estimated number of AND terms, number of inputs, number of outputs, etc. The top ranked pair (p_i,p_j) is selected, and its combined set of functions are minimized, as a PLA, using ESPRESSO. All pairs from the heap associated with either p_i or p_j are removed from the heap. The new PLA, p_{n+1}, becomes a new node, is paired with all remaining nodes, and the new pairs are ranked with all remaining pairs in the heap. At any time, a

pair is rejected if merging it would cause the resulting network of PLAs to become cyclic. The merging of nodes continues until no pair, remaining in the heap, can be merged to bring any improvement. Although, relatively straightforward, the algorithm is very effective; in comparison with the algorithm used by Khatri [8], ours was superior in area and delay by ratios of .79 and .67 respectively.

After PLA clustering, the network may have some redundancies in the literals of the AND and OR terms, even though ESPRESSO guarantees that its result as a PLA is prime and irredundant. This is because we have a multi-level network of PLA nodes, which may still have some redundancies due to unobservability conditions. We will see that removing all redundancies is desirable for 100% test coverage of the delay inequalities, as discussed in Section 6.

Finally, the network of PLAs is placed and bundle routed, discussed in the next section.

5. Physical Design

The physical design step consists of placement, global bundled routing and a post-processing step to improve congestion and delay. We first discuss our algorithm for global bundled routing.

MRSA-Based Bundled Routing Algorithm

The bundled communication scheme poses a new kind of routing problem. Since the *data* signals in the bundle are routed only to modules where needed, a bundled net can have different widths along its different segments. A naive algorithm using constant-width thick wire routing, although simple, would overestimate the space needed for most segments and could cause unnecessary rip-up and reroute later. Our experiments showed a large penalty in terms of wire length and interconnect delay for using a thick routing scheme.

We generalize the MRSA (Minimum Rectangular Steiner Arborescence) tree routing algorithm of [6] for such a variable-width wire routing problems. The MRSA algorithm was chosen because it constructs the routing net from sinks to the source, merging different segments along the way. At any merging point, the widths of the sinks connected to it are known (it is the union of the signals going to those sinks), so the actual width of the segment continuing from this point to the next merging point is known precisely.

The MRSA algorithm of [6] is based on branch-and-bound. The points in the routing graph are ranked by their distances from the source terminal. Possible merging points are divided into two categories: Steiner merging points, which have no sink terminal at it, and terminal merging points, which have a sink terminal at it. A point p is said to be dominated by another point q if $|p\text{-}s|=|p\text{-}q| + |q\text{-}s|$, where s is the source terminal and $|p\text{-}s|$ is the Manhattan distance between p and s. Points are scanned in decreasing distance from the source. At each scan level i, sink terminals more distant than this level have all been connected into subtrees and the roots of all subtrees form a *peer* set P. Possible merging points on the current level are examined. If some point is chosen, then all points in P dominated by it are connected to it to construct a larger subtree and P is updated. Branching happens when a possible Steiner merging point is met, where the Steiner point is chosen (not chosen) as a merging point. However, it is proved that terminal merging points should always be chosen. Finally a MRSA tree is constructed from S, the set of selected merging points.

Our algorithm is a modification of this, so we only discuss differences from the original. The main differences lie in the characterization of merging points and the bounding conditions. To characterize a merging point, we need both the location information and its data signal set. We add a set *SigSet*, for each point in the peer set, which contains all data signals appearing in the subtree rooted at this point. Another difference is the bounding condition when a terminal merging point is met. In the original algorithm, since each wire has the same width, all points that are dominated by this terminal point should be merged with it. This dominance does not always hold when the data signal set is considered. There are two cases:

1. if the data signal set of a terminal point contains the signal set of the dominated point: *SigSet* $v_j \subseteq$ *SigSet* v_i, then the minimum result can be obtained only by connecting v_j to v_i;
2. otherwise, both merging and not merging v_i need to be considered.

The difference in this bounding condition also leads to a difference in the program to generate an arborescence tree from S: when a terminal merging point is met, we need to check if it is contained in S to determine if the points dominated by it, but whose signal set is not covered, should be connected to it.

Block Level Bundled Placer

For placement, we use a block-level placement tool with a global bundled router. The block placement program, shown in Figure 6, uses a simulated annealing (SA) framework with sequence pairs as the layout representation. At each SA step, a new placement is generated, and a *heuristic* is used for constructing a rectilinear Steiner Arborescence tree to generate a topology for each bundled net. This heuristic is just to always merge as early as possible. Then a cost function, which includes area, aspect ratio, routing congestion, and wire delay, is used in SA to evaluate the placement.

```
SA_Place
randomly generate an initial placement
for each scheduled annealing step {
    randomly do one of the following:
        (1) swap a pair in one of the sequence pairs
        (2) flip one of the blocks
    update layout
    for each bundled net, run heuristic RSA tree algorithm
    evaluate area, aspect ratio, congestion and delay
    evaluate cost
    accept or reject }
Signal duplication on critical path for speedup
Postprocess for reducing congestion
```

Figure 6. *Block-level placer and Global Router*

After a final placement is obtained, the full MRSA algorithm is used to obtain the global routes. A post processing step is duplicates some PLA output signals on critical paths to reduce delay. The area cost of duplication in a PLA-based design is small since only an extra column in the OR array is needed to duplicate any signal. Finally, congestion is reduced by moving Steiner merging points away from congested bins to neighboring less-congested bins, while maintaining the net topology of the RSA. Experimental results on additional congestion due to bundled routing were included in [12]. For examples with relative low capacities for communication resources, bundled routing communication results in 1.9% more congested bins compared with single rail communication.

6. Delay Inequality Testing

We start with the observation that the two relative delay inequalities in Section 3 can be tested by *non-delay* testing methods. If either inequality is not met, only a static functional error can occur, which can be observed at the POs of the module. The reason is that

the pre-charged logic can only discharge during evaluation. Therefore, if a *data* signal is late, it may be too late to pull down the appropriate pre-charged line. Thus on that line, the good circuit would have resulted in a 0 (pulled down), but the late signal results in a 1. This happens independent of the clock speed, since the *done-go* signals in the PLA network control the speed of the execution wave inside the PLA network.

In terms of generating a set of test vectors to test for the violation of Delay Inequality 1, $D_{\text{int\_done}} \geq D_{\text{and\_data}}$, we need to find PI minterms (test vectors), which activate all slow modes, which could cause a fault. A slow mode has the property that for any pre-charged NOR gate in the AND-plane, only one transistor pulls it down. If two or more transistors are turned on to pull down the output, evaluation can only be faster. As mentioned in Section 4, the PLA network has been made irredundant in the synthesis flow. Therefore, it is guaranteed that for any transistor of a NOR gate, there exists a PI minterm which turns it on, but no other transistor of the NOR gate is turned on. Further, fault propagation requires that this NOR gate is the unique one that drives a path to the POs. This is by definition a slow mode. Hence, these test vectors are the same set of test vectors as for a normal NOR network. Thus, irredundancy guarantees that any such slow mode can be activated in the AND plane. *Therefore, 100% coverage of the testability of all delay inequalities of type 1 is guaranteed*.

The second delay inequality,

$$D_{\text{or\_done}} + D_{\text{wire\_done}} + D_{\text{go}} \geq D_{\text{or\_data}} + D_{\text{wire\_data}}$$

is slightly more complicated because a slow mode for this would involve a driving PLA, PLA1, propagating a signal to a second one, PLA2, along a bundled route, and causing the arrival at PLA2 to be late. We need to activate all such combinations of a possible slow transistor in PLA1 propagating to PLA2. The slowness may come from a slow transistor in PLA1 propagating to any of its fanout PLAs. Thus, we need to consider all pairs (literal, fanout) where literal is any literal or cube in PLA1, and fanout is any fanout of PLA1. The fanout requirement exists because the functional fault (failure to pull down a NOR gate) happens at the input to PLA2. Therefore, the fault model is a stuck-at-0 (failure to pull down) at the input of PLA2 that must be specifically justified through the selected slow literal of PLA1. The existence of a test vector, that satisfies these conditions, can be formulated using SAT. The SAT clauses are the classical set of SAT CNF clauses for the stuck-at-0 condition at an input to PLA2, *augmented* with additional clauses which force propagation through certain literals in PLA1 to be 0 or 1. For example, if the literal is in an AND term of PLA1, a slow mode would require that every other literal in this AND term to be 0, and further that this AND term should be the only one pulling down the NOR gate it is feeding in the OR plane of PLA1.

A satisfying assignment of the resulting set of SAT clauses is a PI minterm, which tests for a slow mode of the second delay inequality. If SAT returns unsatisfiable, it means that there is no PI minterm which activates the selected transistors and for which the corresponding signal propagates uniquely to the chosen fanout. Thus, this particular combination of modes of slowness is not possible during any evaluation of the circuit. *Therefore, 100% coverage of the testability of all delay inequalities of type 2 is guaranteed*.

7. Experimental Results

The goal of these experiments was to compare implementations against a reference methodology, synthesized using state-of-the-art commercial tools. Standard cells (SCs) were used because they provide a well-known reference and well-developed commercial tools are available. The SC implementations, used for comparison, were fully placed, routed, and buffered using Cadence's *First Encounter* tool set. SC area and delay were compared with our PLA networks, which were synthesized with SIS, placed, and globally routed using our own tools. PLA delays were obtained using SPICE.

All designs were implemented in a 0.18-micron technology using nominal settings on all parameters. The synthesis flows were chosen to obtain optimum delay with area as a secondary objective. Wire delays were computed from routing results using the Elmore delay model [7]. The delays reported for PLA designs include delays for both the evaluation and pre-charge phases. Ten examples taken from the LGSynth91 benchmark set were tested. Figure 7 gives the comparisons.

	SC		PLA(STA)	
Example	Area (um^2)	Delay (ps)	Area (um^2)	Delay (ps)
alu2	11105	1274	7672	1160
alu4	18884	1622	18683	1521
C6288	80304	4401	75218	4235
Apex7	4339	784	4729	818
Apex6	14270	907	15291	838
C1355	14237	1351	12162	1354
C3540	29239	2074	29478	2083
K2	47332	1361	30789	1602
x3	11353	862	14588	845
C5315	31092	2005	42031	2076
Aver.			0.9923	0.9975

Figure 7: *SCs versus PLAs*

We observe from Figure 7 that the PLA implementations are within 1% of both the areas and delays achieved by the Cadence SC tool set.

We conducted another experiment to quantify how small the delay margin can be in the matched delay of the AND plane. Here, we used 90nm technology and, except for V_{th}, set all parameters to constant values that approximated worst case conditions. For V_{th}, we used a Gaussian distribution with a mean equal to a nominal value and σ equal to 5% of the mean. Monte Carlo simulation showed that an 8% delay margin on the delay of the *int_done* line sufficed to guarantee 100% yield. Such a margin was included when the PLA network delay was calculated in the previous experiment.

8. Conclusions and Discussions

A synthesis flow for PLA networks was developed and shown, in terms of both area and delay, to be within 1% of that achieved with classical standard cell implementations using a commercial CAD tool. This is in spite of the relatively immature status of our flow.

It was argued that all the delay inequalities, required for proper functionality, can be tested using non-delay testing methods, i.e. using classical stuck-at testing. This guarantees, for the modules that pass the tests, that the error signal produced at the output of the module is *always* on the critical path. Since such paths consist only of *go-done* signals, the error signal can be used for easy delay testing; single-vector tests can be used instead of vector pairs; only the error signal needs to be monitored instead of monitoring all POs and comparing for the correct computed result.

Although bundled routing guarantees the delay inequalities on the wires hold under all cross talk conditions, cross talk can slow down

the done signals. For more comprehensive delay testing, additional test vectors should be chosen to try to stimulate such slowing down of the done signals.

Ultimately, we would like to demonstrate that PLA networks can be designed with tighter delay margins to obtain high yields while maintaining speed, or high speed while maintaining yield. Tight delay margins should be achievable because delays in the regular structures of the PLAs are highly predictable.

We are currently repeating our experiments at the 90nm node to make sure that our comparisons with SC designs continue to hold. Ultimately, we want to obtain a reasonable set of process variations for 90nm or smaller. We believe that when large variations are present, the ability of this PLA methodology to do easy delay testing can lead to significant performance advantages. Future experiments will include building a detailed bundled router and using SPICE in Monte Carlo experiments to quantify the ruggedness of both the PLA and bundled routing structures. Our static testing methodology needs to be quantified in terms of the relative increase in the size of the test set.

Acknowledgements.

This work was supported partially by the C2S2 Marco research center under contract 2003-CT-888, as well as the California Micro program and our industrial sponsors, Fujitsu, Intel, Magma, and Synplicity.

References

[1] S. Nassif, *Modeling and analysis of manufacturing variations*. In Proc. Of Asia and South Pacific Design Automation Conference, May 2001.

[2] Aseem Agarwal, et. al., *Statistical Timing Analysis for Intra-Die Process Variations with Spatial Correlations*. ICCAD'03, pp. 900-907

[3] *Private communication. Cadence customers*

[4] J. Cortadella, A. Kondratyev, L. Lavagno, C. Sotiriou. Coping with the variability of combinational logic delays, Proc of ICCD, 2004

[5] D. Ernst, et al. Razor: a low-power pipeline based on circuit level timing speculation. In International Symposium on Microarchitecture, 2003

[6] Jason Cong, Andrew B. Kahng, Kwok-Shing Leung. Efficient Algorithms for the Minimum Shortest Path Steiner Arborescence Problem with Applications to VLSI Physical Design. In *IEEE Transactions on CAD,* Jan. 1998.

[7] W.C.Elmore, "*The transient response of dampled linear networks with particular regard to wide band amplifiers*", J. Appl. Phys*,* 55-63, 1948.

[8] S. Khatri, et. al., "*Cross-talk immune VLSI design using a network of PLAs embedded in a regular layout fabric*", ICCAD*,* Nov 2000.

[9] J. Le, X. Li, and L. T. Pileggi, "Stac: Statistical timing analysis with correlation", in *Proceedings of the 41st Annual Conference on Design Automation*, pp. 343-348, ACM Press, 2004

[10] D. Muller, W. Bartky. A theory of asynchronous circuits. In Proceedings of an International Symposium ont the Theory of Switching, 1959, pp. 204-243.

[11] Ivan E. Sutherland. *Micropipelines*. Communications of the ACM, 32(6):720-738, June 1989.

[12] Yinghua Li, Alex Kondrotyev, Robert K. Brayton, "*Gaining Predictability and Noise Immunity in Global Interconnects*", ACSD, 2005.