Clockless Implementation Structure and Methodology for DSM Implementation

Yinghua Li[1] Alex Kondratyev[2] Robert K. Brayton[1]

[1]UC Berkeley
[2]Cadence Berkeley Labs

ABSTRACT

We explore a new implementation method for ***clockless*** *designs based on (1) dynamic PLAs, (2) a synchronous design flow, and (3) timing assumptions realistic for DSM technologies. We argue for the relative invulnerability of our implementations with respect to manufacturing and environmental variations. Initial experiments quantify area and delay comparisons relative to standard cell (SC) implementations.*

1. Introduction

Managing synchronization and control of computation/communication in deep sub-micron (DSM) integrated circuits using a global clock is becoming increasingly difficult. Accounting for delay variations due to signal integrity (SI) violations (crosstalk noise and IR drop) and process variations, results in a significant increase of timing margin requirements. These margins for modern DSM designs often reach 100%, leading to huge power and area penalties caused by over-design, as well as an increase in the design cycle time because of difficulties in achieving timing closure. The impact of signal integrity violations may be reduced by using EDA tools for analysis (see e.g. [1]), which flag where fixes in the layout could be made; however this will become increasingly difficult also. Addressing process variations in a statistical way is a possibility, but such methods are immature, not yet supported by commercial tools, and may be too probabilistic in their results. Thus implementing a synchronous paradigm in DSM will become an increasingly costly proposition. Hence new design techniques are called for that have better implementation predictability and are more tolerant to delay variations coming from SI and manufacturing variations. At the same time, it is desirable to use synchronous specifications and a design flow familiar to designers.

To address the above issues, we propose a design method based on an initial synchronous specification, generation of a clockless control, and the use of dynamic PLAs as the main implementation features. The approach closely resembles a conventional design flow familiar to ASIC engineers and relies on standard tools originally developed for synchronous circuits. As an additional advantage, the proposed approach achieves very low electromagnetic interference (EMI [2]) by eliminating the clock and distributing circuit-switching activities more evenly in time.

In any clocked or non-clocked logic, both the correctness of the functionality needs to be guaranteed, as well as certain timing relations among signals. Delay insensitive (DI) asynchronous logic guarantees the correctness of the operation of the circuit regardless of delays on both the gates and connections. Such circuits are impractical because of their complexity. Thus, in asynchronous synthesis, timing assumptions are usually made about the relative delays of wires or components to reduce logic complexity. In quasi-delay-insensitive (QDI) logic, timing assumptions are made only on the skew of wire delays after forks, while the components can have arbitrary but bounded delays. Informally, QDI assumes that the skew caused by forks in critical wires is negligible or isochronic [3]. In DSM applications, QDI is highly problematic since wire delays are becoming more dominant and unpredictable due to SI and manufacturing variations. To address these shortcomings, different extensions to QDI assumptions were suggested. They impose less demanding assumptions on the skew of wire delays either bounding it by the time to propagate through several levels of logic [4] or even through the whole combinational cloud between registers [5]. However these methods suffer from a high area overhead and usually involve non-conventional design methods to derive implementations.

Area and power penalties of asynchronous implementations can be reduced using more aggressive timing assumptions. For example, the bundled data approach [6] infers the completion of computation or communication by observing transitions on special dedicated outputs (called matched delays) that must change last, relative to other signals. In general, the bundled data assumption may be difficult to guarantee because: 1) matched delays in standard cell designs are implemented by gates (inverters commonly) that are different from those used in the datapath and 2) there is no guarantee that delay elements are placed closely to critical paths to which they are to match. Thus using conventional bundled data implementations in DSM designs is problematic due to high variations in wire and gate delays.

An important step in producing completion signals for every computation block reliably was made in architectures based on dynamic PLAs [7,9]. A delay of a single PLA is easy to match using an extra row and column of its internal planes. This delay is highly predictable because of PLA regularity and short local wiring. When an environment condition (voltage, temperature) changes, this delay scales well with the delays of other PLA outputs because of the locality of variations. Though good progress in gaining timing reliability was achieved in [7,9], these works suggested only a partial cure because they relied heavily on timing assumptions when coordinating different PLAs and used conventional clocking to synchronize registers.

Our work benefits from combining a clockless synchronization scheme borrowed from micropipelines [6,15] with the advantages stemming from the predictability of PLA network implementations. We also extend the results from [7,9] by developing a new routing (bundled routing) scheme that easily satisfies the bundled data assumption. Our work leads to two timing assumptions that must be guaranteed for correct behavior. We discuss how these are relatively easy to guarantee even in a DSM environment.

Initial experimental results show a 23% area-delay product penalty compared to standard cells (SC) implementations. SC implementations were used for reference comparisons because: 1) both methods are based on a synchronous design specification, and

hence can be compared directly, sharing much of the front-end of an automatic synthesis flow, 2) SC implementations are often used as a standard reference for comparing area-delay tradeoff potential of different methodologies applied to ASICs. Our implementation ensures correct communication using "bundled routing" between the PLAs, and relies, in the PLA, on very local timing assumptions that are much easier to satisfy than the ones needed for global clocks.[1]

The main contributions of the paper are:

1. It combines dynamic PLAs [7,9] which tolerate variations in computation delays, with bundled routing of the (*done*,*data*) signals of the PLAs which tolerates variations in communication delays.
2. Our bundled routing proposal makes the global communication insensitive to delay variations due to cross-talk. PLAs can be built to be insensitive to cross-talk as well [7].
3. All delay assumptions needed for correct operation depend only on realistic local process and environmental variations, but are insensitive to global variations.
4. The proposed approach is fully automated. It starts from conventional synchronous specifications and proceeds to implementations using standard EDA tools.

We believe that this methodology will be an appealing alternative to conventional ASIC implementations for DSM technologies because of its superior tolerance to manufacturing and environmental variations and the absence of a clock.

This paper is organized as follows. Section 2 gives some relevant background about global architectures (outside the combinational cloud) that could be used in our clockless design. In Section 3, the PLA-based design flow, the PLA structure, its delay and area computations, and the synchronization method used are discussed. In Section 4, we describe the bundled routing method, give delay inequalities required for correct operation, and argue for their continued validity under realistic local parametric variations. Experimental results are shown in Section 5 and Section 6 concludes.

2. Background

We propose to build modules using our clockless methodology and to use asynchronous micro-pipelines for communication between modules. This paper is concerned mainly with the synthesis of the modules. Micro-pipeline architectures [6], of which Figure 1 is only one example, have detectors (D) which sense if the computation of the previous stage has completed. The registers (R) allow data to pass, while holding it until the fanout computation is done. Data between the modules propagates like a wavefront. Each register maintains the local wavefront until the computation result is captured and held by the registers of the next stage. The fanout registers acknowledge receipt of information by changing the outputs of their detectors. The acknowledgements from all fanout registers are collected by a synchronizing latch, a C-element, which rises only when all its inputs are 1 and falls only when all are 0. Then the next wavefront is allowed to propagate through and is maintained until accepted by the next stage. In asynchronous logic, registers are controlled by locally generated signals instead of a global clock. More complex pipelining structures become possible, since clock routing and attenuation problems are absent.

Our implementation of a module is superficially like this, but no pipelining between PLAs is done; a new computation does not start until the whole combinational block has evaluated. Each module in the leaves of a design hierarchy will be implemented using a clockless implementation of the combinational portions of a modules, which will be implemented by a multi-level network of PLAs; latch boundaries are preserved. In this way our clockless implementation of a module behaves similarly to a synchronous circuit with the only difference that synchronization signals for registers are derived internally rather than by using a common clock source; hence each module creates its own delay.

Communication Between Modules:

Due to the absence of timing assumptions about the wavefront propagation, communication between modules of the architecture from Figure 1 usually relies on a two-phase operation. Data changes from the spacer phase (reset) to a proper data codeword during the set phase, and then back to the spacer in the reset phase.

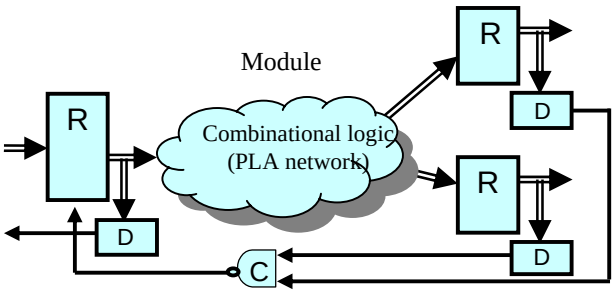


Figure 1. One stage (module) of a pipeline

In both phases of such an asynchronous pipeline, the completion of the data propagation can be logically determined by the data content, which can be ensured through the use of so-called delay-insensitive (DI) encoding. This would be appropriate for DSM at the global stage, since no timing assumptions are required and the extra overhead of the required acknowledgement circuitry would be amortized by the larger area of the combinational logic implementation (network of PLAs) of each module. One approach to DI encoding uses multi-valued encoding of the communicating signals. The simplest scheme for multi-valued encoding is dual-rail encoding, in which each signal a is represented by two wires $a.0$ and $a.1$ (i.e. $a = 1$ encoded as $a.0 = 0$, $a.1 = 1$, and $a = 0$ encoded as $a.0 = 1$, $a.1 = 0$). A spacer for the reset phase is encoded by all signals reset to 0 in both approaches.

In contrast, within a module, we use single-rail signals and a bundled data approach for communication between the PLAs. For implementations using dynamic PLAs, reset can be mapped naturally to the precharge phase while set corresponds to PLA evaluation. Earlier, we had experimented, within a module, with using different QDI encodings, multi-valued logic, and different acknowledgement circuitry, but concluded that all were too expensive. These experiments led us to the methodology of this paper.

[1] *Although these ideas can be applied to pipelining of the network of PLAs to achieve a (roughly) 40% performance improvement, our focus in this paper is on a non-pipelined clockless implementation, which employs exactly the same latches (clockless) at the combinational boundary as given in the specification.*

3. Design Flow for a PLA-Based Implementation

3.1 Overview

The choice of dynamic PLAs as the main building block of asynchronous implementation is motivated by the following:

1. Dynamic PLAs typically have a designated signal "done" that changes last in the evaluation and precharge phases. By using this signal, detection of completion need not be made logically. Implementing this signal within a PLA is easy and does not introduce significant timing overhead since PLA structures are regular and their delays are highly predictable [9].
2. PLAs can realize much more complicated logic than standard cells. This increases the ratio of computation to communication, which is desirable in DSM where wire and gate delays are predicted to have equal effects on circuit performance.
3. PLAs can be made insensitive to cross-talk, if necessary, with little area penalty [7].
4. Synthesis tools for synchronous designs can be used and very little extra effort is needed to transform synthesis results to clockless designs using dynamic PLAs.
5. Timing overhead of the precharge phase can be reduced to a minimum.

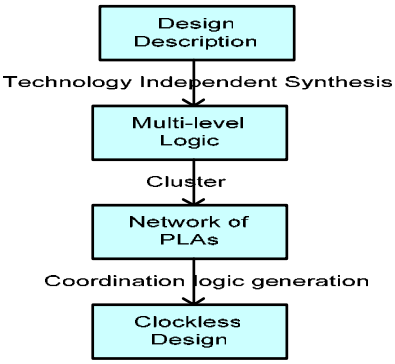


Figure 2. Clockless Design Flow for Dynamic PLA Implementation

```
Algorithm 1 Cluster a Circuit into NPLA
C = simplify_network(C);
InitializeHeap(clusterPair);
while(HeapNoEmpty(clusterPair)&&
(HeapMaxKey(clusterPair)>0||NumPLA(C)>numLimit))
{
   (p1,p2) = HeapGetMax(clusterPair);
   pnew = cluster(p1,p2);
   Espresso(pnew);
   RemoveRelatedPairs(clusterPair,p1);
   RemoveRelatedPairs(clusterPair,p2);
   AddClusterablePairs(clusterPair,pnew);
}
```

Figure 2 shows the design flow for a clockless implementation using dynamic PLAs. It starts with technology independent synthesis using conventional synthesis tools for synchronous designs. Synthesis parameters can be tuned to target a good decomposition into a network of PLAs. Then a clustering algorithm is applied to map the synthesis result to a network of PLAs [7]. Finally, the resulting network of PLAs is augmented with extra logic to properly coordinate the precharge and evaluation phases.

Thus, only the final step is special to the clockless design flow. Moreover, because of the regularity of PLA structures, the clustering algorithm used as the technology mapping step is much simpler (and hence faster) than the corresponding step for standard cell designs.

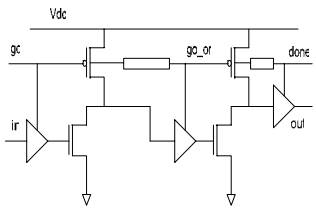
Our clustering algorithm currently targets minimizing the total active area of the PLA network, while delay of the whole circuit is controlled only roughly by restricting the number of PLA-levels of logic. Starting with a SIS optimized network, we treat each node as a PLA. Heap *clusterPair* contains all pairs of PLAs that can be clustered in the current network. A *cost*, assigned to each pair, is related to the PLA area change and net-number change. The pair that can bring the maximum saving to cluster into a new PLA is chosen next. Then the heap is updated by deleting non-existent pairs and adding pairs introduced by this new PLA. The procedure continues until no pair exists, or no cost saving is possible.

Once a network of PLAs is obtained, the key issue is to derive an efficient and reliable way for coordinating the evaluation and precharge phases. In [7], the "*go*" signal for a particular PLA is obtained from the "*done*" of the slowest PLA in its fanins. However, this mechanism is vulnerable to process variation and delay changes due to signal integrity problems. We use a more robust structure. The timing assumptions for this structure are stated and analyzed.

3.2 Implementation of Dynamic PLAs

PLA Structures

Figure 3 shows the dynamic PLA structure that we will use [9]. *go* controls precharge of the AND plane and input buffers. The rectangle with an output *go_or* denotes the delay that matches the maximum delay of AND. *go_or* controls the intermediate buffer and precharge of the OR plane. Another matched delay for the OR plane is used to produce *done*. Output buffers are controlled by *done*. The matched delay is implemented by an extra row in AND plane and an extra column in OR plane.



NOR-NOR with controlled intermediate buffer

Figure 3. Schematic diagrams of PLA structures

Delay Computation

We use the delay model of [8], which we briefly describe here. The delay of a buffer and one programmable bit in a PLA is based on a linear delay model, e.g. for a programmable bit, we have

$$D = D_{i\_bit} + D_{l\_bit} * load \qquad (3.1)$$

where D_{i_bit} is the intrinsic delay and D_{l_bit} is the load dependent delay factor.

The following timing assumptions are needed to ensure correct functioning of the PLAs.

(1) For the evaluation phase, *go* should not rise until the input data signals are ready. So the evaluation phase delay for the structure in Figure 3 is:

$$D_{eval} = D_{go} + D_{AND} + D_{OR} + D_{OBuffer} \quad (3.2)$$

where $D_{go} = D_{i\_gobuffer} + D_{l\_gobuffer}*(m + 1)*pLoad,$

$D_{AND} = \max_v\{ D_{i\_inbuffer} + D_{l\_inbuffer}*m_v*bitLoad + D_{i\_bit} + D_{l\_bit}*(\max_h(m_{v,h})*bitLoad + L))\},$

$D_{OBuffer} = D_{i\_outbuffer} + D_{l\_outbuffer}*n_{max} *inBufferLoad.$

m is the number of product terms in this PLA. n_{max} is the maximum number of fanout PLAs among outputs generated by the PLA for which the delay is evaluated. *pLoad* is the capacitance of a single AND-plane precharge transistor. *bitLoad* is the capacitance of each programmable bit. To compute the AND-plane delay [9], each input literal is examined. m_v is the number of product terms where literal v is used. $\max_h(m_{v,h})$ gives the maximum number of literals in such product terms. L is the external load attached to the product term row, which is usually the input capacitance of an intermediate buffer plus the capacitance of a precharging circuit. The OR-plane delay computation is the same as the AND-plane, except that the driver is an intermediate buffer while the load buffer is an output buffer. (2) The delay for precharging is estimated as

$$D_{precharge} = D_{go}+ D_{i\_pre} + D_{l\_pre}*(\max_h*bitLoad + L) \quad (3.3)$$

D_{i_pre} and D_{l_pre} are the intrinsic delay and load dependent factor of the precharging transistor for AND plane. The precharging delay for the OR plane is not included because (a) precharging for AND and OR planes can be started at the same time (b) the precharging for the OR plane can end later, as long as it can finish before the evaluation of the AND-plane is done. As long as the sum of the precharging delay and evaluation delay of the AND-plane is larger than the OR-plane precharging delay, which is the usual case, Equation (3.3) is valid.

Area Computation

The area for this structure is computed as in [9]:

$$Area = (H_{InBuffer}+H_{OutBuffer}+(m+1)*H_{Bit}) \quad (3.4)$$

$$*(W_{Precharge}+ W_{IntBuffer} + v*W_{ANDBit} +(n+1)*W_{ORBit})$$

where m is the number of product terms, v is the number of input literals (a binary variable may have 1 or 2 literals feeding the PLA), n is the number of output signals. $H_{InBuffer}$ and $H_{OutBuffer}$ are the heights of the input and output buffers, $W_{Precharge}$ is the width of the precharging circuit for the AND-plane, $W_{IntBuffer}$ is the width of an intermediate buffer, H_{Bit} is the height of a programmable bit and W_{ANDBit} and W_{ORBit} are the widths of the programmable bit in the AND and OR planes respectively. Thus, the area and delay of a PLA can be calculated immediately once the logic realized by the PLA is known.

4. Timing Strategies for Coordination of PLAs

Local synchronization (fanin-based)

The first way of coordinating the propagation of the evaluation and precharge phases reduces to deriving *go* for each PLA from synchronizing *done* signals of all its fanin PLAs using an AND gate (see Figure 4).[2] The correctness of the functioning for this implementation requires the following timing assumptions:

$$delay(go\_done) \geq delay(go\_out) \quad \forall out \in \text{PLA outputs} \quad (4.1)$$

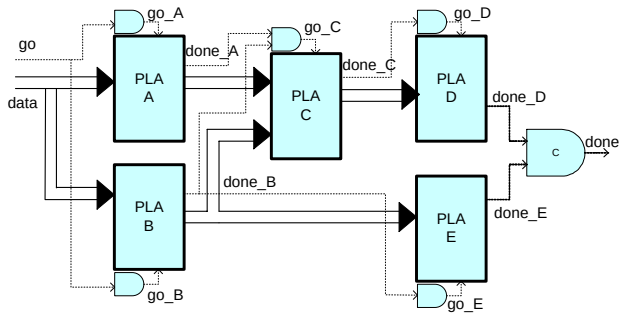
$$delay(wire\_data) \leq delay(wire\_done) + delay(AND\_gate) \quad (4.2)$$

wire_data and *wire_done* represent the nets for data and nets for *done* signal. Assumption (4.1) is simply the bundled data assumption for PLAs. It states that *done* changes not earlier than when all output signals of the PLA settle down. Assumption (4.2) requires that *go* can be asserted only after all data signals settle at the PLA inputs. The guarantee of this assumption is aided by:

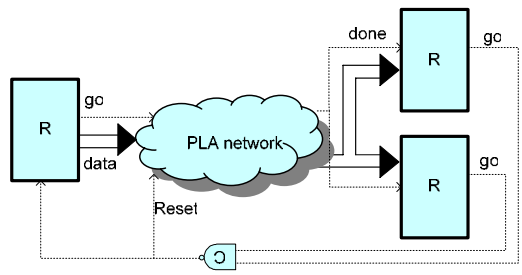
1. *done* for the PLA is generated not earlier than its data output signals, and
2. *done* and the data signals have the same origin (start from the same PLA) and destinations (arrives to the same PLA). By using bundled routing (see next sub-section) for done and data signals, we guarantee that all use the very same metal layers, via locations, and therefore the delay of *done* is scaled well with the rest of wires in the bundle. By shielding *done* in this bundle to insure that it can't be sped up by cross-talk, we can quite easily meet this condition. Additionally, the data signals can't be slowed down by cross-talk because it is either stable or rising and all its neighbors are either stable or rising.

Thus by attaching the *go* signal generator to the PLA and applying bundled routing on *done* and data signals Assumption (4.2) can be met if relative local variations are not too severe.

Finally, the *done* signals of the PLAs whose output signals are only connected to primary outputs of a combinational cloud are synchronized by a C-element (Figure 4).



(a) Combinational logic (PLA network)



(b) One stage of pipeline

[2] *In conventional asynchronous architectures, C-elements are used for synchronization. In our case, AND gates suffice because precharge is applied to all PLAs simultaneously and the completion of the precharge phase relies on timing assumptions that are justified later.*

Figure 4. Fanin-based Local Synchronization

The circuit in Figure 4 works as the follows. The evaluation phase starts from the setting of values at primary inputs and then asserts the corresponding signal *done*. *done* for primary inputs coincides with the *go* signal for PLAs at the first level of a network and their outputs are evaluated. After evaluation, corresponding *done* signals are asserted (see timing assumption (4.1)) and the data wavefront propagates further along the network of PLAs. Note, that if a PLA consumes data inputs from several PLAs (PLA C e.g. in Figure 4a), then it starts its evaluation only after all *done* signals of its fanin PLAs are asserted ($go = \wedge_i done_i$). When *go*=1, all data values at the inputs of a PLA are set (Assumption (4.2)) and evaluation proceeds using the correct input values. Finally, when the evaluation wavefront reaches the primary outputs, the C-element indicates the settling of the data outputs by asserting the output *done* signal.

Once the evaluation phase is over, the precharge phase starts from the assertion of the reset signal which is active low. This signal may be derived as an inverted version of the output *done* of the C-element (see Figure 4) after passing registers (we will need an OR gate if there is more than one output register for this combinational logic). Precharge signals (obtained by de-asserting *go*) are applied to all PLAs simultaneously by feeding *reset* to AND gates evaluating the *go* signals in each PLA (*reset* is not shown in Figure 4.a). Completion of the precharge phase is indicated by a falling transition at output *done* of the C-element. Note, that the C-element indicates the completion of the precharge phase for PLAs from the last level only. To ensure that a new evaluation wavefront starts after all PLAs are precharged, the following timing assumption is required:

$$max_{A \in N}\ (A\_precharge) \leq max_{B \in NO}\ (B\_precharge) + delay(C\text{-}el),$$

where *N* and *NO* stand for the PLA network and for the last level of PLAs in the network respectively. This assumption is realistic because the delay of a C-element is usually much larger than the skew of the precharging delays in different PLAs. To be even safer, extra OR gates could be used to collect the *done* signals of lower level PLAs.

Bundled Routing

We propose to make *done* an inverted signal (1 or falling) and to route the *data* and *done* signals, output from the same PLA, as a bundle of varying width. This will guarantee two conditions, first, that there is no vulnerability to delay variations due to cross-talk, and second, that the same metal layers and routing lengths are shared in the bundle, as well as having all vias similar and close. To illustrate the varying width of a bundle, if a fanout requires only 1 or 2 of the data values of a PLA, the width will be 3, 1 for *done* and 2 for the *data*, or 2 for the *done* and *data* signals and 1 for an added shield wire. The bundle is constructed with *done* not on the boundary. The bundle width increases by 1 with each addition of a *data* signal.

Now consider the worst case that could happen. *done* is always falling. *data* signals are either rising or 0. The minimum delay of *done* is obtained when both neighbors are constant 0. The maximum delay of a *data* signal is when one neighbor is 0 and the other is its own *done* signal.[3] In this case, by symmetry, *done* is slowed down by approximately the same amount as the maximally-slowed *data* signal (its neighbor). Assuming that the line lengths, metal dimensions, and via resistances, as functions of the distance along the bundle, are close to the same for all wires in the bundle, we have

$$\mathrm{Delay}_{00}(done) + \delta \geq \delta + \max_i[\mathrm{Delay}_{00}(data_i)]$$

where Delay_{00} is the delay of a line with both neighbors quiescent and δ is the added delay due to a signal and its neighbor changing in opposite directions at about the same time.

Thus we argue informally, that bundled routing and the margin provided by the additional AND-gate in (4.2) guarantee this inequality is not vulnerable to cross-talk as well as to global manufacturing/environmental variations. It can be vulnerable to local variations on the wires of the bundle, such as variations in via resistance. Ultimately one needs to measure, using realistic local variations, what margin (beyond an extra AND-gate delay) (4.2) might be needed for realistic yields.

5. Experimental Results

Our aim in these experiments was to provide a reference point for our clockless implementations in order to give an idea of their relative efficiency in area, delay, and metal usage. We compared the area and "data-to–data" delay of our clockless structure to synchronous implementations based on standard cells. The data-to-data delay is the time interval between which input data first enters the PLA network at the primary inputs, until the next input data can enter the network. This corresponds to the clock cycle time for synchronous designs. Standard cell implementations were chosen for reference because they are used widely in synchronous design and thus provide a good standard reference. A network of dynamic PLAs requires a precharging signal for each PLA. Since it is difficult to generate these signals from a clock, a dynamic PLA network structure is more naturally self-timed.

All designs were implemented in a 0.18-micron technology. This choice was defined by the limited access to SC libraries for smaller feature size technologies. Smaller technologies should favor the clockless approach more. Wire delays were computed from routing results using the Elmore delay model [10]. The data-to-data delays reported for clockless designs include delays for both the evaluation and precharge phases.

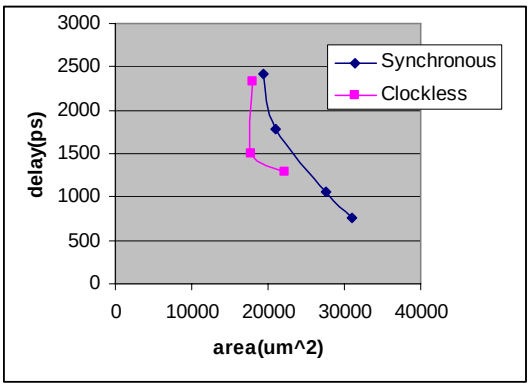
Ten examples from the LGSynth91 benchmark set were tested. For each example, 4 design points for the synchronous implementation and 3 design points for the clockless implementation were obtained. The cycle time and area product averages are shown in Figure 5. Synthesis and technology mapping of the synchronous standard cell designs were performed using SIS [11], while placement and routing were done by the Cadence Silicon Ensemble SEDSM-5.3 [12] tool. For the clockless designs, we started with the technology independent synthesis results of SIS and applied the clustering algorithm shown in Section 3 to generate the network of PLAs. Then a block level placement tool [13] was used. Since currently we have not yet implemented a bundled router, we approximated this by routing "thick" wires using SEDSM-5.3 followed by a trimming procedure. Thus a bundled net is required to have a global route with enough space for its maximum thickness to all sinks of the net. A bundled router specific for this project should lead to much less metal usage and smaller wire delay. Thus, the current experimental results should be viewed as providing conservative bounds on performance and metal usage.

[3] *Since other done signals are in the middle of their bundle, they can't be a neighbor.*

As one can see from Figure 5, for most examples, clockless implementations produce designs with moderate penalties compared to SC synchronous implementations. For some examples such as alu2, apex6, even better results than the synchronous approach were obtained. The average ratio of the area-delay products is 1.23. Note that for large examples a trend seems to be that the clockless approach results in larger penalties. The main reason is due to the inefficiency of handling bundled routing by SEDSM which finally produces large wire delays (about 30% for PLA implementation of C6288 e.g. is taken by wires compared to 10% wire delays for the same circuit in SC implementation). A significant improvement in this is expected with a specifically developed bundled router. We again emphasize that the clockless implementation is targeted for its robustness to delay and process variations. This advantage is not measured by the current experimental setting and needs to be done in the future. However, some idea about the delay penalty due to the more robust communication scheme might be obtained by removing the AND gates for synchronizing done signals at PLA inputs and then using timing assumptions to ensure the correctness of the functionality. A typical delay penalty of 30% stemming from the use of the AND gates was estimated using the C2670 example.

Example	Synchronous SC	Clockless NPLA	Ratio
alu2	2.68e7	2.11e7	0.787
alu4	6.16e7	7.74e7	1.256
C6288	6.89e8	9.27e8	1.346
apex7	9.29e6	9.11e6	0.980
apex6	3.43e7	2.50e7	0.729
C1355	2.16e7	3.31e7	1.532
C3540	1.24e8	1.85e8	1.492
C2670	3.96e7	5.75e7	1.454
C5315	1.12e8	1.71e8	1.526
average			1.236

Table 1. Area-Delay Products(um^2*ps)



(a) Area Delay Curve for apex6.blif

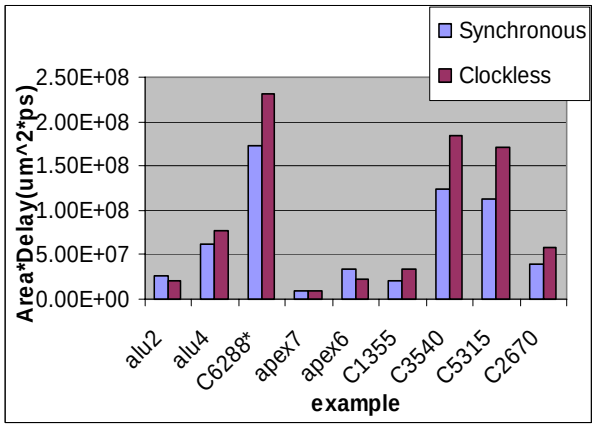


Figure 5. (b) Area Delay Products for Synchronous and Clockless implementations

*: the area delay products of C6288 are devided by 4 to fit in

We note that the SC implementations use only 3 levels of metal, while the PLA network uses 4 levels. The metal utilization of metal 3 for SC implementation and those of metal 4 for PLA implementation are shown in Figure 6. The PLA approach needs one extra metal layer but on the other hand, metal is saved by the clockless design since there is no clock to be routed. Such saving can become substantial for fine-grained pipelined designs.

Whether the presented area and delay numbers are appealing enough for designers to switch to such clockless implementation structures will depend on the application and the severity of difficulties encountered with DSM problems of the future. Even so, to the best of our knowledge, these numbers represent the first time that automatically derived clockless implementations, tolerant to variations in communication and computation delays, resulted in such modest penalties. The common belief about automated methods for asynchronous design is that they result in at least 2 times the area and more than 50% delay penalties, which means the ratio of area-delay products to SC would be around 3 [5]. The only exception that we are aware of is the de-synchronization approach [14], but it requires significantly more stringent timing assumptions and does not address the issues of ruggedness to delay variations in interconnects.

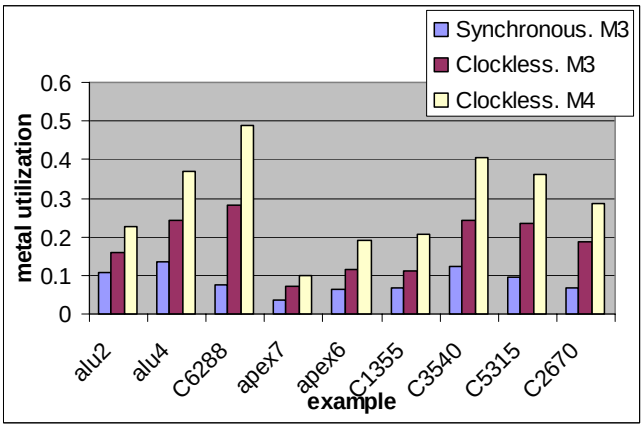


Figure 6. Metal Utilization of Synchronous and Clockless implementations

6. Conclusions

We introduced dynamic PLAs into clockless designs and developed new automatic implementation methods for this, starting from synchronous specifications. Timing assumptions can be restricted to local variations and can be made invulnerable to cross-talk. Thus this method should be very appropriate for DSM, providing designs which are rugged with respect to variations due to manufacturing and signal integrity. In reference to standard cell (SC) clocked designs, initial experiments demonstrate area and delay penalties are moderate.

Future experiments will include building a detailed bundled router, designing PLAs structures for very small DSM technologies, and using SPICE in Monte Carlo experiments to simulate parameter variations in order to quantify the ruggedness of such designs. In addition, we need a testing methodology to capture catastrophic events, like nearly failed vias. We note that delay testing is easier with this methodology, since the test for delay can be controlled by the *done* signal. This uses the fact that unlike clocked designs, a delay fault can only happen if equations (4.1) and (4.2) are violated.

References

[1] *User guide. Celtic User Manual*, Cadence Design Systems, Inc., 2003.

[2] John McCardle, David Chester. *Measuring an Asynchronous Processor's Power and Noise*, SNUG Conference, April 2001.

[3] A.J. Martin. *Programming in VLSI: From communicating processes to delay-insensitive circuits*. In C. A. R. Hoare, ed., Developments in Concurrency and Communication, UT Year of Programming Series, pages 1-64. Addison-Wesley, 1990.

[4] A. Lines, "*Pipelined asynchronous circuits*" Master's Thesis, 1995, Caltech

[5] A.Kondratyev, K. Lwin. *Design of asynchronous circuits by synchronous CAD tools*. In Proceedings of the Design Automation Conference, pp. 411-414, June 2002.

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

[7] S. Khatri, R. Brayton and A. Sangiovanni-Vincentelli, "*Cross-talk immune VLSI design using a network of PLAs embedded in a regular layout fabric*", *Int. Conf. Computer Aided Design*, Nov 2000.

[8] M. Gao, J. Jiang, Y. Jiang, Y. Li, S. Sinha and R. Brayton, "*MVSIS*", International Workshop on Logic Synthesis, June 2001.

[9] Fan Mo, R. Brayton, "*PLA-Based Regular Structures and Their Synthesis*", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Jun 2003.

[10] W.C.Elmore, "*The transient response of dampled linear networks with particular regard to wide band amplifiers*", J. Appl. Phys., vol. 19, no. 1, pp. 55-63, 1948.

[11] E. Sentovich, K. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. Stephan, R. Brayton and A. Sangiovanni-Vincentelli, "*SIS: A system for sequential circuit synthesis*", Tech. Rep., UCB/ERL M92/41, Electronics Research Lab, University of California, Berkeley, May 1992.

[12] *Envisia Silicon Ensemble Place-and-route Reference*. Cadence Design Systems, Inc.. 1999.

[13] F. Mo, A. Tabbara and R. Brayton, "*A Force-Directed Macro-Cell Placer*", International Conference on Computer Aided Design, Nov 2000.

[14] I. Blunno, J. Cortadella, A. Kondratyev, L. Lavagno, K. Lwin and C. Sotiriou, "*Handshake Protocols for De-Synchronization*", Proc. Of the 10th International Symposium on Asynchronous Circuits and Systems, April 2004.

[15] Ted Williams, "*Self-timed Rings and Their Application to Division*", PHD Thesis, Stanford University, 1991.