The Best of Both Worlds: The Efficient Asynchronous Implementation of Synchronous Specifications

Abhijit Davare
Dept. of EECS, UC Berkeley
Berkeley, CA 94720
davare@eecs.berkeley.edu

Kelvin Lwin
Cadence Design Systems
San Jose, CA 95134
klwin@cadence.com

Alex Kondratyev
Cadence Berkeley Labs
Berkeley, CA 94704
kalex@cadence.com

Alberto Sangiovanni-Vincentelli
Dept. of EECS, UC Berkeley
Berkeley, CA 94720
alberto@eecs.berkeley.edu

ABSTRACT

The desynchronization approach combines a traditional synchronous specification style with a robust asynchronous implementation model. The main contribution of this paper is the description of two optimizations that decrease the overhead of desynchronization. First, we investigate the use of clustering to vary the granularity of desynchronization. Second, by applying temporal analysis on a formal execution model of the desynchronized design, we uncover significant amounts of timing slack. These methods are successfully applied to industrial RTL designs.

Categories and Subject Descriptors

J.6 [**Computer-Aided Engineering**]: Computer-aided design (CAD)

General Terms

Algorithms, Performance, Design, Experimentation

Keywords

Desynchronization, Separation Analysis, Clustering

1. INTRODUCTION

The synchronous paradigm is useful from a specification perspective since it allows the designer to separate timing and functionality. Implementing this paradigm using a global clock, however, is becoming more difficult. Since the global clock signal has one of the highest switching activities of all signals and since it spans a vast amount of area on the chip, it is subject to many signal integrity and process variation issues [2]. It also consumes a significant amount of power, and has high EMI [3].

In an asynchronous design style, timing is coupled with functionality. This design style is resilient to timing variation, but also involves high overhead, since each block must also handle timing through the use of handshake (i.e. request and acknowledgment) signals. For most mainstream applications, the additional overhead and lack of commercial tool support makes asynchronous design unattractive.

An ideal solution would allow the decoupling of design specification from design implementation to balance the benefits of the synchronous and asynchronous design styles. Through the use of desynchronization and additional optimizations, this decoupling can be achieved.

Desynchronization [1] automatically replaces the global clock with local latch controllers that communicate via handshake signals. It can be implemented in current commercial design flows and is transparent to the RTL designer. The desynchronization approach described here is a specific instance of the more general method described in [4].

This paper proposes two methods that significantly reduce the overhead of desynchronization. First, clustering allows a single latch controller to be responsible for multiple latches. Second, separation analysis of events in the desynchronization execution model allows us to discard certain handshake signals, reducing the complexity of latch controllers.

In Section 2, we introduce the desynchronization framework. In Section 3, an abstraction for representing the synchronous design is described and the clustering and separation analysis problems are formulated in further detail. In Section 4, experimental results are presented from the application of desynchronization with clustering and separation analysis to a number of industrial RTL designs.

2. DESYNCHRONIZATION

The desynchronization procedure begins with a synchronous circuit description consisting of combinational blocks, registers, and a global clock net. First, the registers are split into master and slave latches and retiming across latch boundaries is performed if required. Note that this first step of desynchronization is often used in standard synchronous design [5].

DAC 2004, June 7–11, 2004, San Diego, California, USA.

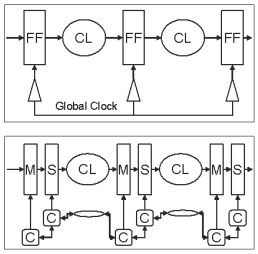

Figure 1: Desynchronization applied to a synchronous circuit

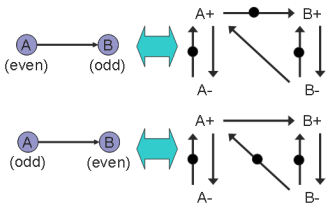

Figure 2: Templates for latch graph to signal transition graph conversion

Next, for each latch, a latch controller component is inserted to replace the global clock signal. The controller communicates using request and acknowledge signals with the successor and predecessor latches respectively using a four-phase "return to zero" handshake. In case of multiple successors (predecessors), the request (acknowledgment) signals are generated with AND-causality. An example of desynchronization applied to a simple pipeline is shown in Figure 1. All request and acknowledgment signals pass to neighboring controllers through a delay element, represented by the ovals in Figure 1. This *matched delay* corresponds to the longest path delay in the combinational block between the two latches and serves the purpose of a conservative completion detector [8].

The execution semantics of a desynchronized circuit can be modeled using a Petri Net [7] representation known as a Signal Transition Graph (STG) [6]. The events in the STG are the rising (+) and falling (-) edges of the controller clock signals. The STG model for desynchronization is live, safe, and conflict-free [1]. The rules for building such a STG for transparent high latches are specified in Figure 2. Two templates are required depending on the parity (master or slave) of the latches.

Timing information is added to the STG by associating a delay *interval* with each event and transition as shown in Figure 3. By treating all delays as intervals, we can take into account any sources of timing uncertainty. In this case, it is important to note that most substantial source of timing uncertainty is the discrepancy between the actual combinational delay and the matched delay on the same chip, not across all chips.

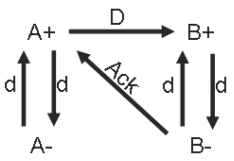

Figure 3: Timing in the desynchronization model

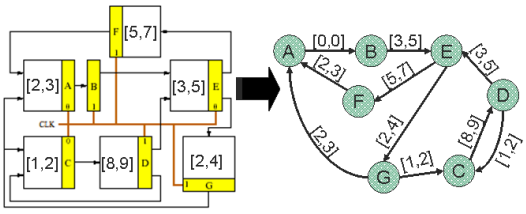

Figure 4: Constructing the Latch Graph from a Synchronous Circuit

The rising (falling) transitions $A+$ ($A-$) and $B+$ ($B-$) are assigned a delay equal to that of the rising (falling) edge delay for the respective latch controllers. The transitions marked d represent the pulse width delay for the latch controller. The *Ack* delay represents the delay for communicating the acknowledgment signal to the predecessor latch. Finally the matched delay D is calculated such that the quantity $D + delay(B+) + d + delay(B-)$ is greater than or equal to the delay of the combinational block between latches A and B. Setup and hold constraints are both satisfied with this timing model. The delay overhead introduced by the latch controller (the sum of the delays from $B+$, d, and $B-$) does not influence the overall timing of the circuit as long as the combinational delay is larger than the latch controller delay.

3. OPTIMIZATION

The desynchronization process has two major sources of design freedom: the number of latch controllers and their complexity. These can be explored using clustering and separation analysis respectively. We first describe the latch graph model used to represent a desynchronized circuit.

3.1 Constructing the Latch Graph

The circuit, consisting of combinational blocks and master or slave (parity odd or even) latches, is represented as a weighted directed bipartite graph called the *latch graph*. The vertices in this graph represent the latch controllers in the desynchronized design. Each vertex is marked with the appropriate parity. Initially, the RTL designer may have already grouped together multiple latches in a single hierarchical unit (e.g. bus). In the sequel, we choose to consider this hierarchical unit as atomic. So, a vertex in our latch graph may actually correspond to a designer-specified combination of latches.

The edges in the latch graph represent the request and acknowledgment wires that connect the latch controllers. The weights on the edges represent the delay intervals of the corresponding combinational blocks. An example of the transformation of a simple sequential circuit to a latch graph is shown in Figure 4, where the white rectangles represent combinational blocks, while the shaded rectangles are latches.

3.2 Separation Analysis

If the relative occurrence times of request and acknowledge events for controllers in the circuit can be calculated, some latch controllers can be simplified. An example of this type of controller simplification is given in Figure 5. The

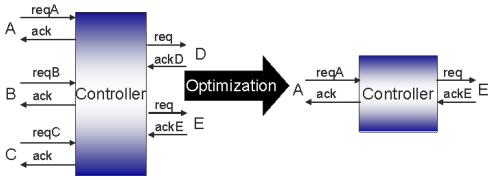

Figure 5: Control simplification under the guarantee that $arrival(reqB_k) \leq arrival(reqA_k) \wedge arrival(reqC_k) \leq arrival(reqA_k) \wedge arrival(ackD_k) \leq arrival(ackE_k)$ for all occurrences k

```
findSlack(event u, event v)
{
   a := MaxSep(u, v);
   b := MaxSep(v, u);
   if (a > 0 AND b < 0) return min(a, |b|);
   else if (a < 0 AND b > 0) return -1 * min(|a|, b);
   else return 0;
}
```

Figure 6: Determination of Timing Slack using maximum separation analysis

key concept in this type of simplification is the assignment of partial orders to the request and acknowledge signals for each of the controllers based on the timing characteristics of the circuit. The dominated control signals can then be eliminated. With precise timing information - corresponding to zero width delay intervals - separation analysis will remove all control edges not in the critical cycle.

Relative occurrence times of control signals can be determined by solving the maximum separation analysis problem using the TSE algorithm [9]. Given events u and v in a timed STG, let u_k and v_k represent the k^{th} occurrence of these events in the execution of the STG. Then, the maximum separation time between u and v, $\Delta = MaxSep(u, v)$ is defined such that:

$$\tau(u_k) - \tau(v_k) \leq \Delta \quad \forall k \geq 0$$

where $\tau(u_k)$ and $\tau(v_k)$ are the times associated with the specified occurrence of the events. The TSE algorithm finds a conservative bound on Δ with pseudo-polynomial complexity.

To find the partial order between two events u and v, we analyze both $MaxSep(u, v)$ and $MaxSep(v, u)$ as shown in Figure 6. If the function returns 0, no timing slack exists. For any non-zero return value, the timing slack is annotated on the appropriate edge and corresponding inputs may be removed during controller implementation. The timing slack on control edges can also be used to simplify the corresponding combinational block or be propagated backward to other combinational blocks.

3.3 Clustering

The desynchronization technique as described assigns a latch controller to each vertex in the latch graph. This may cause an excessive area and power overhead due to the large number of controllers needed. Clustering groups together multiple latches under the control of a single latch controller.

In the latch graph representation, clustering merges multiple vertices into a single vertex. As a result, we may have to merge edges as well. A conservative edge merging procedure is used to preserve the semantics of the desynchronization model. If edges A_1 to A_n are merged into a new edge B, $B.min$ is $min(A_1.min, ..., A_n.min)$ and $B.max$ is $max(A_1.max, ..., A_n.max)$.

3.3.1 A Heuristic Algorithm

In this section, we describe a polynomial-time heuristic algorithm for clustering. The heuristic first calculates the merge fitness between each pair of vertices in the circuit and merges the pair with the best score. This procedure is repeated iteratively until no more mergings can take place.

During this process, a clustering tree – known as a dendrogram [10] – is constructed. Leaves in this tree are the original vertices from the latch graph. Interior nodes represent merged subsets of the original vertices. The children of any interior node are the sets of vertices merged to create that node. Once the dendrogram has been constructed, a cut specifies a particular clustering. This cut can be made depending on a number of different criteria – the number of latch controllers, the limit on the number of latches controlled by a single controller, the tolerable loss of timing precision, or any combination of these. We use the *merge tolerance* parameter to indicate the maximum allowable amount of timing imprecision by only merging vertices with merge fitness scores above this parameter. The calculation of the merge fitness between a pair of latch controllers is based on two factors: the *shared neighbors criterion* and the *edge delay correspondence criterion*. Any weighting of the two factors can be used; an equally weighted linear sum is used here.

If a circuit contains a structure where a group of controllers have a large percentage of shared predecessor and successor controllers, we can substantially reduce both the number of controllers and the edges (matched delay and acknowledgment wires) between them by merging these controllers. The *shared neighbors criterion* attempts to find such a structure in the graph. If the shared percentage is zero (i.e. there are no shared predecessors or successors), no edge delay correspondence calculation is possible and a negative merge fitness score is returned. This prevents unrelated vertices from being merged.

The *edge delay correspondence criterion* measures the similarity in delay intervals between edges that will be merged. If the two edges being merged have substantially different delay intervals, the merging step introduces a relatively large degree of conservatism. The rationale behind tracking the similarity of delay intervals is to ensure that timing precision is not lost when merging edges.

Clustering is NP-hard and an exact solution can be found using a MILP formulation. To determine the performance of the heuristic, it was evaluated against an MILP implementation for small circuits. The heuristic performed quite well, with loss in timing precision being within 10%.

4. EXPERIMENTAL RESULTS

Experiments were conducted on five industrial RTL designs to determine the effectiveness of the clustering and separation analysis optimizations. Design 1 is a simple pipelined processor implementation with 8,500 cells. Designs 2 and 3 are video transform applications with 5,000 cells each. Designs 4 and 5 are image processing applications with 19,000 cells each.

The first set of experiments measured the amount and ex-

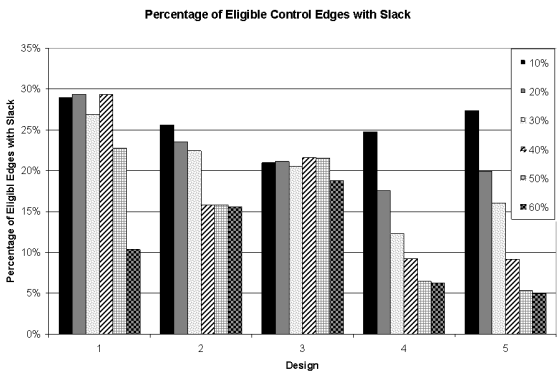

Figure 7: Generation of slack with separation analysis on industrial RTL designs

Clust.	Latches per Cont.	Metric	No Sep. Ana.		Sep. Ana.		Savings	
			Cont.	DataP.	Cont.	DataP	Cont.	Total
None	1	Power	33.9	62.7	26.4	59.4	22%	11%
		Inst.	2246	9991	1899	9841	15%	4%
Med.	2.35	Power	12.3	58.0	10.8	58.3	12%	3%
		Inst.	861	9977	699	9972	19%	4%
Full	9	Power	3.55	54.1	3.17	52.4	11%	3%
		Inst.	254	9998	235	9891	7%	2%

Figure 8: Desynchronization overhead after Place & Route with different optimizations

tent of timing slack generated by the separation analysis on the RTL designs under different delay variation percentages. These experiments were conducted after performing clustering with the *merge_tolerance* parameter set halfway between its minimum and maximum values. Note that since retiming was not performed, a percentage of edges in any given design were the single edges from the master to the slave latches. These edges were not eligible for separation analysis and could never be allocated slack.

Figure 7 shows the percentage of eligible control edges annotated with slack under various delay variation percentages. Depending on the design and the percentage variation, the percentage of eligible edges with slack ranges from 5% to 30%. The amount of timing slack annotated on these edges is quite high. On average, even with variation at 60%, the clustered RTL designs still have 11% of eligible control edges annotated with slack equal to 50% of the combinational delay.

The second set of experiments measures the impact of the optimizations on the area and power overhead of the desynchronized version of Design 1 after performing place & route. These experiments are carried out with delay variation set at 30%. Logic synthesis is performed using the RTL compiler from Cadence. The gate-level netlist and matching timing constraints are then imported into SoC Encounter. Floor-planning is done along with creation of power structures inside Encounter. Designs are implemented with the TSMC 0.13 μ library.

Figure 8 shows area (number of instances) and power consumption (in mW) for differently clustered versions of Design 1 before and after separation analysis. The results indicate that clustering is a potent method to decrease desynchronization overhead with a reduction in power (area) from 35% (18%) for the non-clustered design to only 6% (2%) for the fully clustered design. Separation analysis results in further control savings of up to 20% for both power and area.

5. CONCLUSIONS AND FUTURE WORK

We have introduced two strategies for balancing the penalties and benefits of desynchronization. The performance of the clustering algorithm on benchmark circuits suggests that clustering is an effective method of reducing controller overhead while minimizing the amount of resulting timing imprecision. Separation analysis explores the potential for controller simplification based on the calculation of the sequential slack in the desynchronized circuit. The experiments conducted show that industrial RTL designs contain sizable amounts of timing slack, even in the presence of 60% delay variation.

Additional optimizations can be performed to further reduce desynchronization overhead and generate additional timing slack. First, retiming can be carried out on the designs prior to desynchronization, leading to a richer connectivity structure and more potential for assigning slack. Next, partial desynchronization can be investigated where parts of the circuit that stand to gain the most from desynchronization would be desynchronized, while the remaining parts of the circuit would be left under the control of the global clock. Finally, a more relaxed version of maximum separation analysis can be carried out for some designs. For instance, it is easy to show that the execution of the STG for a pipeline structure has a finite prefix for initialization followed by a periodically repeating tail. For such designs, we do not need to consider all k occurrences of events for the separation analysis, but only those occurrences which occur after the initialization phase. This can potentially identify more dominated control edges and generate more slack.

6. ACKNOWLEDGMENTS

We would like to thank Jordi Cortadella, Luciano Lavagno and Christos Sotiriou for numerous discussions on desynchronization and Henrik Hulgaard for providing an implementation of the TSE algorithm.

7. REFERENCES

[1] J. Cortadella, A. Kondratyev, L. Lavagno, C. Sotiriou, "A Concurrent Model for Desynchronization," IWLS 2003.

[2] J. P. Uyemura, "VLSI clocking and System Design" in *Introduction to VLSI Circuits and Systems* John Wiley & Sons, 2002.

[3] J. Rabaey, A. Chandrakasan, B. Nikolic, "Digital Integrated Circuits - Second Edition," Prentice Hall, 2003.

[4] A. Benveniste, L. Carloni, P. Caspi, A. Sangiovanni-Vincentelli. "Heterogeneous Reactive Systems Modeling and Correct-by-Construction Deployment." Proceedings of EmSoft 2003.

[5] D. Chinnery and K. Keutzer, "Reducing the Timing Overhead," in *Closing the Gap between ASIC and Custom: Tools and techniques for High-Performance ASIC design* Kluwer Academic Publishers, 2002.

[6] Chu, Tam-Anh, "On the Models for Designing VLSI Asynchronous Digital Circuits," Integration, 4(2):99–113", June 1986.

[7] T. Murata, "Petri Nets: Properties, analysis and applications," Proceedings of the IEEE, pp. 541–580, Apr. 1989.

[8] I. E. Sutherland. Micropipelines. Communications of the IEEE, Volume 32, Number 6, 1989.

[9] H. Hulgaard, S. M. Burns, T. Amon, and G. Borriello, "An algorithm for exact bounds on the time separation of events in concurrent systems," IEEE Transactions on Computers, 44(11):1306–1317, November 1995.

[10] R.M. Cormack, A review of classification (with Discussion). J. Roy. Stat. Soc. A 134:321–367, 1971.