Synthesis of Externally Synchronous, Internally Asynchronous Circuits

Bassam Tabbara* Luciano Lavagno† Alberto Sangiovanni-Vincentelli‡

Abstract

The purpose of this work is to take an arbitrary 2-phase synchronous circuit and eliminate its global clock while preserving its externally visible behavior. This is done by converting it into a set of self-timed combinational circuits with latches and a clock generation network that uses only a local handshake. We discuss the overhead required by this approach to self-timing and outline opportunities for future research.

1 Introduction

The problem of clock distribution is becoming increasingly difficult to manage with standard synchronous digital design techniques. The methods for tackling it are moving far from the original simplicity and degree of automation that has made synchronous synthesis-based design so successful.

One possible mechanism for tackling these problems is the technique of *self-timing* ([4, 6]). Unfortunately, traditional Self-Timed Circuit (STC) design requires different specification methods (like Signal Transition Graphs, STGs, [1]) that are unfamiliar to designers. Moreover, understanding the complex timing patterns of true STCs is an obstacle to a widespread diffusion of this design style. In this work, we follow an idea originally presented in [5] to build a *local synchronization network* that, starting from a *two-phase* synchronous circuit, derives the clock of each latch in it from the signals that indicate termination of computation in the latches that drive the inputs of its cone of logic.

The original paper of Varshavsky *et al.*, though, did not discuss the method to automate the partitioning of the logic into two phases and the design of the clock generation circuitry. It did not provide, either, a general-purpose, cost-effective mechanism to derive the "Done" signal for each block of combinational logic, that is required to properly clock each latch. Our work tackles both problems, and discusses the advantages and disadvantages of the proposed solution.

For the first problem, we provide a first-cut, simple technique for partitioning an arbitrary synchronous circuit into two phases, and discuss methods to improve this partition. Moreover, we give an architecture in which the combinational logic, latches and clock generation circuitry cooperate correctly. Finally, we describe by means of a Signal Transition Graph the specification, to be synthesized by state-of-the-art asynchronous design tools ([2]), of the clock generation circuit.

For the second problem, we show that the Timed-Shannon Circuit (TSC) (introduced in [3] for low-power circuit design) can be modified to produce an acknowledge (Ack) signal when it has completed the computation of a Boolean function. Hence it can be used to implement the combinational part of the proposed architecture.

Let us consider first the problem of implementing a piece of combinational logic with an Ack signal.

2 Timed Shannon Circuits for the combinational logic

The problem of generating an Ack signal from a piece of combinational logic has been solved in various ways, all depending on the single path propagation property, that is sufficient (but not necessary) to guarantee correct completion detection. In this paper we consider an automated solution, that uses only static CMOS logic, and that has a reasonable overhead with respect to the synchronous case. It is based on Timed Shannon Circuits, that are conceptually similar to dynamic CMOS logic, but are less sensitive than dynamic circuits to noise problems.

Recall that a Timed Shannon Circuit (TSC, [3]) is derived by traversing a Binary Decision Diagram of the Boolean function to be implemented. The traversal is performed from the root to the leaves of the TSC, and this guarantees that computation proceeds

*University of California, Berkeley
†Politecnico di Torino, Cadence Berkeley Labs
‡University of California, Berkeley

along *only one path*. Thus it becomes easy to detect its completion, by checking when the transition of the "Enable" input to the TSC reaches one of the leaves.

Figure 3 shows the construction of the TSC at node labeled E of the BDD appearing on the left (bold edges are associated with the 0 value of the labeling variables xi, thin edges are associated with the 1 value). It also shows the final complete TSC. It should be obvious that the top Enable signal starts the computation when it rises from 0 to 1. The output F of the circuit is the OR of all the edges leading to the 1 terminal in the BDD. Similarly, the output Ack can be derived by ORing together all the edges leading both to the 1 and to the 0 terminal.

The idea is that during the passive stage both F and not-F are 0 (because the circuit is disabled by pulling the Enable low) i.e. the circuit is performing no computation and is passive. When the circuit is Enabled, either F or not-F is 1 (and the other is 0) since only one and only one path will be active. Ack is 0 during the passive stage. Therefore the Timed Shannon Circuit (TSC) is indeed self-timed, since every transition on the Enable signal is acknowledged by a transition on the Ack signal.

In order to build the STC that will be substituted for the initial synchronous design, we first build the BDD of each combinational circuit whose output function feeds a latch input, and then construct the TSC.

Let us consider now the complete architecture using the TSC to implement the combinational logic.

3 The proposed Self-Timed Circuit architecture

In order to preserve the external equivalence between the synchronous and the asynchronous circuit, we need to divide the set of memory elements of the original specification into two classes, and guarantee that two adjacent (with respect to combinational logic true paths) memory elements always belong to different classes. In this way, at each point in time one class can be in the *active* phase, and the other one can be in the *passive* phase. Passive latches keep their outputs stable, active latches update their outputs with the new values computed by their combinational logic cones.

As a first step to achieve this goal, we assume to start from a synchronous circuit and we split each latch into two latches. This does not change the behavior, but only requires two clock cycles to perform one step of computation. We can then use retiming techniques to move those latches around while preserving the behavior, and partially recover the loss in performance due to the doubling of the number of cycles per computation. Note that the self-timing technique does not require to perform timing analysis of this retimed circuit, because the combinational logic itself signals the completion of computation. This new circuit with split latches by construction yields a latch adjacency graph that can be colored with 2 colors.

In the future we will look at better partitioning schemes, that allow to share the clock generation logic between latches, based on connectivity analysis. Latches with a high number of common fanins to their respective combinational logic cones should be kept together in this approach.

Once we have partitioned the circuit into latches driven by cones of logic, we can look at the architecture that ties them together. It is depicted in Figure 1, assuming that the combinational logic cone for the latch implementing signal F2 is driven by signals F1 and F3, with associated Done signals X1 and X3. The notation in the figure is as follows:

TSC	Timed Shannon Circuit
F2	Input of latch 2 (from TSC)
T2	Synchro-element output for latch 2
A2	Acknowledge signal for F2 (from TSC)
X2	Done signal for latch 2
EN	Enable for the TSC

The Done signal rises to indicate that this portion of the circuit is entering the *passive* phase, and its fanout blocks of logic can start computing (by entering the *active* phase).

Let us examine the behavior of this architecture by starting with the TSC. This is derived from the original circuit (as described in Section 3) to implement the combinational circuit feeding the input of the latch. The output latch is used to allow other portions of the circuit to compute while keeping F2 stable. The Enable of the TSC is hooked to the output T2 of the synchro-element. This element can be thought of as a gateway that is initially disabled. Therefore the TSC is disabled and the Ack signal (A2 in Figure 1) is at 0. This means that the Done signal (X2 in Figure 1) is at 0, and this portion of the circuit is passive (i.e. it is not doing any computation). Other portions can freely look at the output stored in the latch, that is waiting to be enabled by its neighbors when they are done computing.

This brings us back to the top of Figure 1. The synchro-element can be activated (opened) only by all the Done signals from its neighbors going to 1 (X1 and X3 in Figure 1).

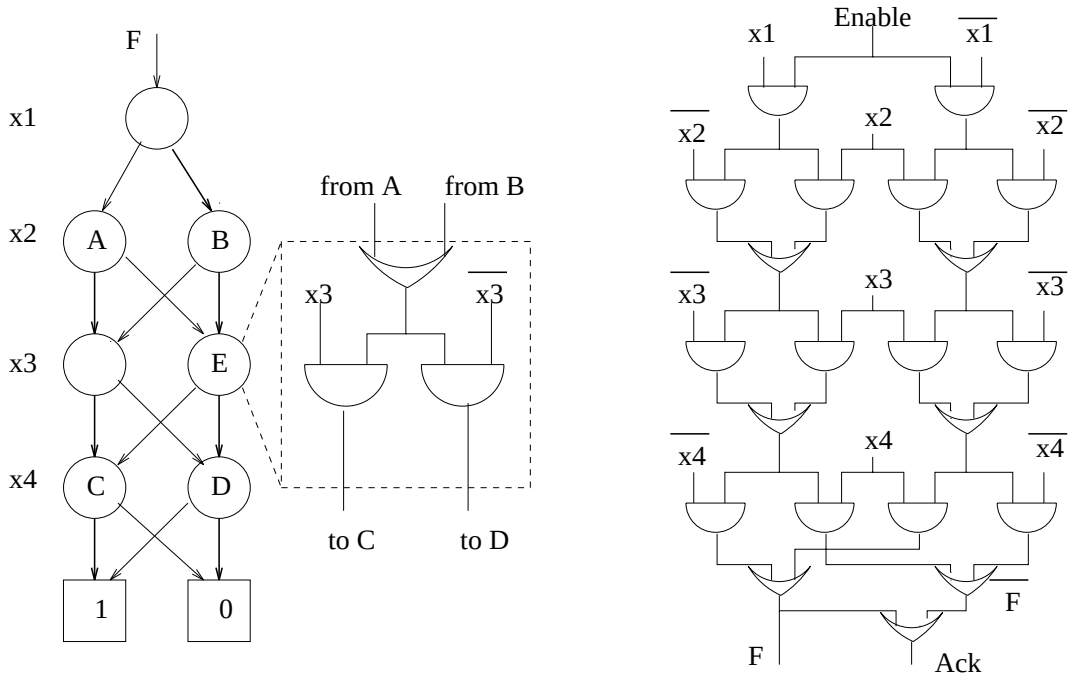

Figure 1: Sample TSC construction from a BDD

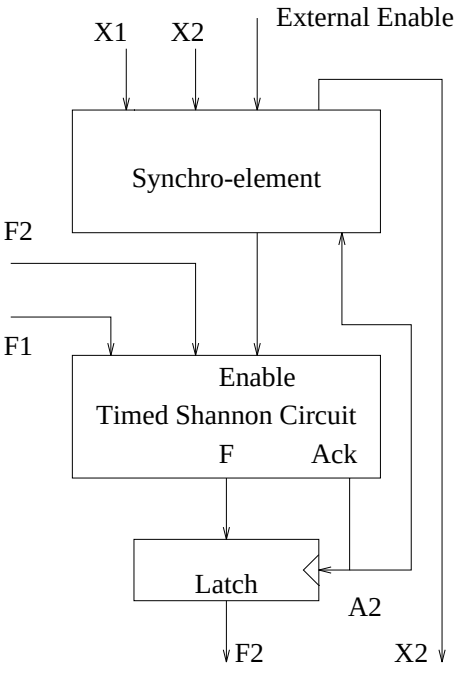

Figure 2: The proposed circuit architecture

The External Enable shown in the figure is used to preserve the external interface of the system. It determines the time at which the primary inputs (if any) are sampled by the TSCs. It corresponds functionally to the clock of the original synchronous circuit, but must be distributed to fewer elements in general. The primary outputs, on the other hand, are ready when the corresponding Ack outputs rise. This means that in order to interface this circuit with a synchronous circuit, some timing analysis is required to verify that the circuit can absorb input transitions fast enough and that its outputs are ready when the other circuits will latch them.

When the neighbors of this circuit are done computing, they enable the TSC. When the TSC is done computing, Ack (A2 in Figure 1) goes to 1, and since it is connected to the edge-triggered clock of the latch, the latch memorizes the result. The computation is now done, and this sub-circuit turns itself off again (i.e. it goes back to its passive stage). The other sub-circuits that depend on F2 have also been notified that this signal is ready to be used, at their earliest convenience.

The STG of the design presented above section is shown in Figure 2. It describes formally the causal dependencies between signal transitions (denoted by + for the rising transition and by - for the falling transition).

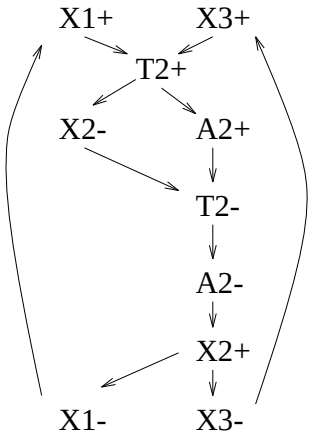

Figure 3: The STG of the synchro-element

4 Conclusions and future work

So far we have experimented with several toy examples, and were able to verify the correct functionality of their STCs. We also built the STCs for the sequential multi-level subset of the MCNC91 and ISCAS89 benchmarks and noticed roughly a two-fold increase in area in general. This can be considered quite high in the current technology, but its importance is going to decrease in the future.

The problem of distributing each signal together with its local clock, on the other hand, is much more serious, since it requires

- a two-fold increase in routing resources as well, and
- a tight control over the relative skews of each such pair of signals.

In a sense, we are replacing a global synchronization and skew problem with several local instances. Placement and routing algorithms, currently targeted at a single critical net (the clock) will have to be modified to take into account this "divide-and-conquer" approach.

In the future we have several directions to explore, including optimizing the multi-level logic of the STC without destroying the property of self-timing, and improving the initial partitioning scheme in order to minimize interaction among the different components and exploit locality of computation.

References

[1] T.-A. Chu. *Synthesis of Self-timed VLSI Circuits from Graph-theoretic Specifications.* PhD thesis, MIT, June 1987.

[2] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Technology mapping of speed-independent circuits based on combinational decomposition and resynthesis. In *Proceedings of the European Design Automation and Testing Conference (EDTC)*, March 1997.

[3] L. Lavagno, P. C. McGeer, A. Saldanha, and Alberto Sangiovanni-Vincentelli. Timed Shannon Circuits: A power-efficient design style and synthesis tool. In *Proceedings of the Design Automation Conference*, pages 254–260, June 1995.

[4] C. L. Seitz. Chapter 7. In C. Mead and L. Conway, editors, *Introduction to VLSI Systems.* Addison Wesley, 1981.

[5] V. Varshavsky., V. Marakhovsky, and T.-A. Chu. Asynchronous timing of arrays with synchronous prototype. To be published, 1996.

[6] V. I. Varshavsky, M. A. Kishinevsky, V. B. Marakhovsky, V. A. Peschansky, L. Y. Rosenblum, A. R. Taubin, and B. S. Tzirlin. *Self-timed Control of Concurrent Processes.* Kluwer Academic Publisher, 1990. (Russian edition: 1986).