A Clock-Tuning Circuit for System-on-Chip

Yaron Elboim, Avinoam Kolodny, *Member, IEEE*, and Ran Ginosar, *Member, IEEE*

***Abstract*—System-on-chip (SoC) design depends heavily on effective reuse of semiconductor intellectual property (IP). Clock distribution has become a problem for integrating IP cores into a single synchronous SoC, because of different clock delays in the IP cores. We propose an onchip clock-tuning circuit, which enhances design flexibility. Programmable delays are inserted in the clock distribution network, such that clock alignment and synchronization are achieved. Design iterations are eliminated with the use of the tuning circuit, saving design effort, and cost. The method is also applicable to compensating for unbalanced clock trees. Hierarchical clock tuning can be implemented and can take advantage of the hierarchical structure of the SoC. Skew analysis has shown that the added programming unit outperforms other clock design options. The method was implemented in a commercial chip, and demonstrated good functionality with high productivity of the design flow.**

***Index Terms*—Circuit tuning, clock distribution, inserted delay, intellectual property (IP) core, system-on-chip (SoC).**

I. Introduction

In system-on-chip (SoC) design, a buffered clock distribution network is typically used to drive the large clock load. Chip design involves a clock alignment step, which equalizes the delay from the clock source to each and every clock target (flip flops, latches, or other memory elements) [7], [15]. Accurate clock alignment is important, because unwanted differences or uncertainties in clock network delays may degrade performance or cause functional errors [3], [4]. Clock distribution and alignment has become an increasingly challenging problem in very large scale integration (VLSI) design, consuming an increasing portion of resources such as wiring area, power, and design time [21].

Ideally, intellectual property (IP) cores ("IPs") should be treated as "black-boxes" to support "plug-and-play" [5], such that IPs can be inserted or removed without affecting other blocks. However, the clock distribution network does not support this concept because each change influences the complete network [7]. Redesign and verification of the global clock distribution network may be required after each change. Such iterations are undesirable and should be minimized. For hard IP cores, clock routing is carried out during the design of the IP core. The timing interface between the hard IP core and the rest of the SoC (signal setup and hold times, input capacitance) must be carefully considered [6]. A change of a hard IP core might necessitate a complete redesign of the clock network. For soft IP cores, clock tree routing is performed during chip-level integration. Clock distribution and alignment within soft IP cores become the responsibility of the system integrator, but a similar problem exists with soft macros which had been already placed and routed, so that engineers are reluctant to redesign them.

In a competitive commercial environment, IC design is typically optimized for shortest time to market. To shorten design times, physical design is often performed in parallel with logic design, although in theory the former should follow the completion of latter. In such cases, global physical features of the IC, such as the global clock distribution network, may have to be redesigned multiple times, where each change in the logic incurs painful and expensive redo of the global nets.

Clock tuning can be used to eliminate repetitive redesign of the clock network. High-speed systems with multiple boards often require clock tuning after assembly [2]. A tuning circuit can be used statically or dynamically to perform clock alignment according to the uncertainty of the system [1]. Multiple phase-locked loops (PLLs) may be employed to align the clock dynamically [13], but are expensive and difficult to design.

We propose an efficient method for clock alignment in SoC design, using a programmable circuit for static-delay tuning. The main goals of the static delay tuning are to enable quick and easy integration of IP cores into SoC and to ease the design of the clock distribution network of the SoC. In Section II, we demonstrate the problem of IP core integration due to different clock delays. Next, the common solution of signal delay insertion is described and its inefficiency is discussed. The preferred method of clock-delay insertion is presented in Section III, using either global clock redesign or tuning. Sections IV–VII describe variants of clock tuning and analysis of clock skew. Sections VIII–XI present circuits, experimental results, and conclusions.

II. SoC Timing Example

Data sheets and texts on logic design describe timing in terms of setup and hold times for input ports, and contamination and propagation delays for output ports. *Contamination delay* t_{cd} is the period following an input transition during which the output still retains its old value or the earliest time for the output to make its first transition. The contamination delay is closely related to the fastest logic path in the circuit. *Propagation delay* t_{pd} is the latest time for the output transition. This delay is related to the slowest path in the circuit [1], [7], [14]. "Black-box" models provide similar block boundary timing information for hard IP cores [9].

Consider the example IP cores in Figs. 1 and 2. The graphical timing diagrams specify the timing characteristics of the IP cores. The t_{int} parameter describes the internal clock delay in the IP core from its clock input port to all the IP core state elements. This delay resides in the tree structure of the clock dis-

Manuscript received July 27, 2001; revised May 1, 2002.

Y. Elboim was with Oren Semiconductor, Ltd., Yoqneam, Israel. He is now with Intel Corporation, Haifa 31015, Israel (e-mail: yaron.elboim@intel.com).

A. Kolodny is with the Electrical Engineering Department, Technion–Israel Institute of Technology, Haifa 32000, Israel (e-mail: kolodny@ee.technion.ac.il).

R. Ginosar is with the Very Large Scale Integration (VLSI) Systems Research Center, Electrical Engineering Department, Technion–Israel Institute of Technology, Haifa 32000, Israel (e-mail: ran@ee.technion.ac.il).

Digital Object Identifier 10.1109/TVLSI.2003.812371

1063-8210/03$17.00 © 2003 IEEE

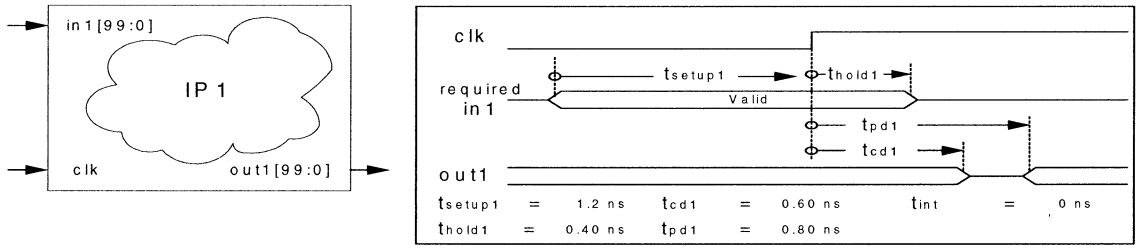


Fig. 1. IP1 and its timing diagram. The clock arrives at the flip-flops immediately ($t_{\mathrm{int1}} = 0$). t_{setup1} and t_{hold1} are required in order to guarantee correct capture of inputs. Module outputs may start changing after t_{cd1} and are guaranteed to settle by t_{pd1}.

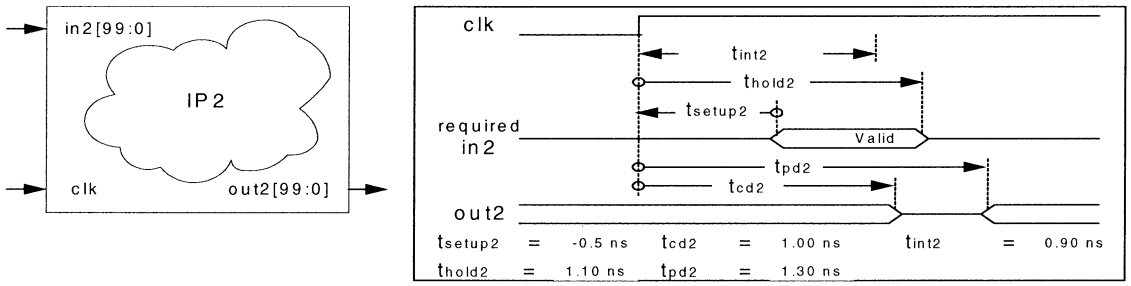


Fig. 2. IP2 and its timing diagram. The clock arrives at the flip-flops t_{int2} after it rises at the clock port of IP2, but all parameters are still measured with respect to the external clock port. Hence t_{setup2} is negative and t_{hold2} is quite long.

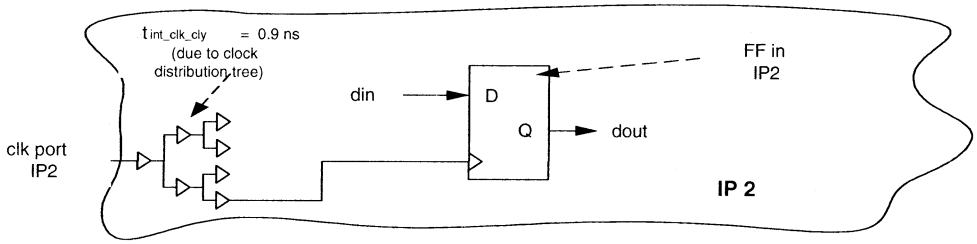


Fig. 3. Internal clock-delay IP2. The deep clock distribution tree causes a large t_{int2}.

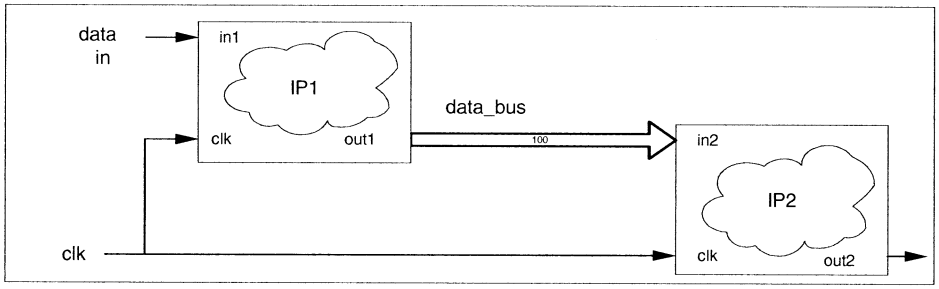


Fig. 4. Connecting IP1 and IP2 to single SoC.

tribution network inside the IP core itself (Fig. 3). Typically, the internal clock delay shown in Figs. 2 and 3 is not described by IP core timing specifications. Note that all timing parameters are measured with respect to the external clock port of the IP core, rather than the clock pin of flip-flops. In IP1, the internal clock delay is assumed zero for simplicity. In IP2, the internal clock delay is 0.9 ns. The negative setup time in Fig. 2 results from this large internal clock delay (deep clock tree, as presented in Fig. 3). A nonnegligible internal clock delay is typical in deep submicron processes [10], [11]. Note, that if there were sufficient logic delay between the inputs and the flip-flops in IP2, the required setup time would not be negative, but this is not a common practice in design of IP cores.

In Fig. 4, we illustrate how the two IP cores may be connected in a single SoC. The data_bus (100 bits) connects out1 of IP1 to in2 of IP2. The clk wire feeds both IP cores. For simplicity, we assume no delays at the system level, on the external clock distribution network and on data_bus. The combined SoC timing diagram is shown in Fig. 5. By convention, IP2 needs to sample data value input A rather than B. The diagram reveals a hold time violation for in2. The delay on out1 is insufficient for the needed hold time constraint on in2 input

$$T_{\mathrm{cd1}} > T_{\mathrm{hold2}}. \tag{1}$$

From Fig. 5 it can easily be seen that IP1 is the faster IP core. Its signals are available before IP2 is ready to receive them. The

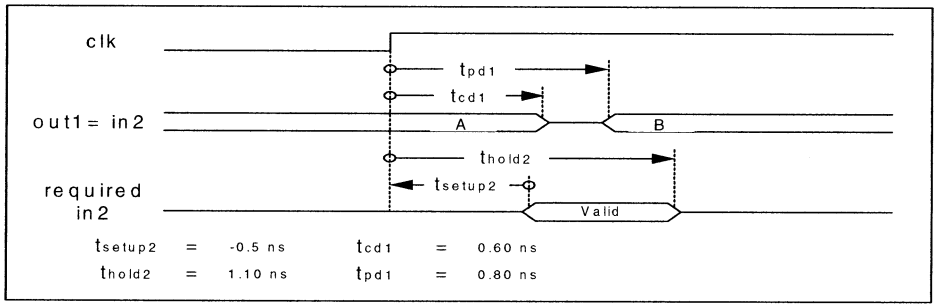


Fig. 5. Combined timing diagram of IP1 and IP2. Value A should be captured, but a hold time violation is caused because the outputs of IP1 change too early.

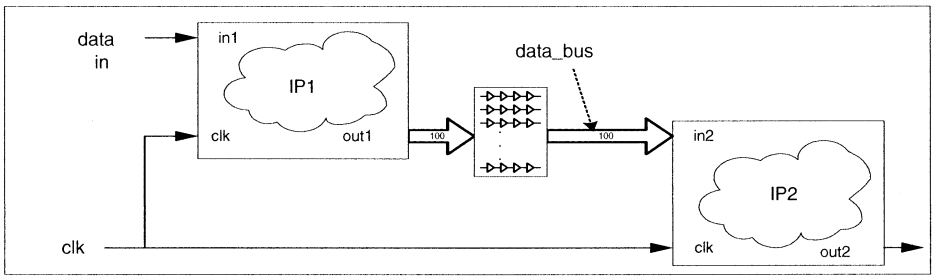


Fig. 6. Solution for IP1 and IP2 timing connectivity problem by data delay insertion method.

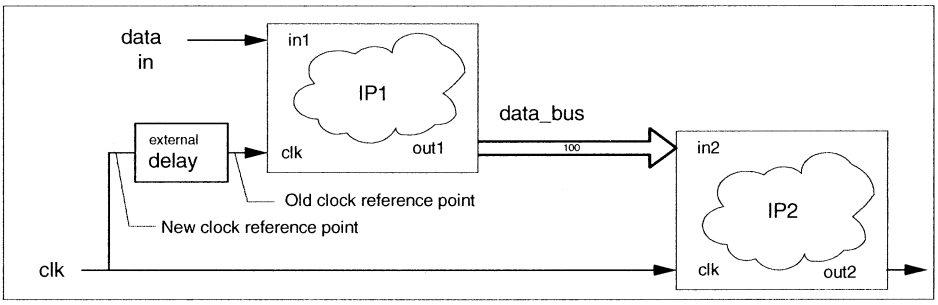


Fig. 7. Solution for IP1 and IP2 timing connectivity problem by clock-delay insertion method.

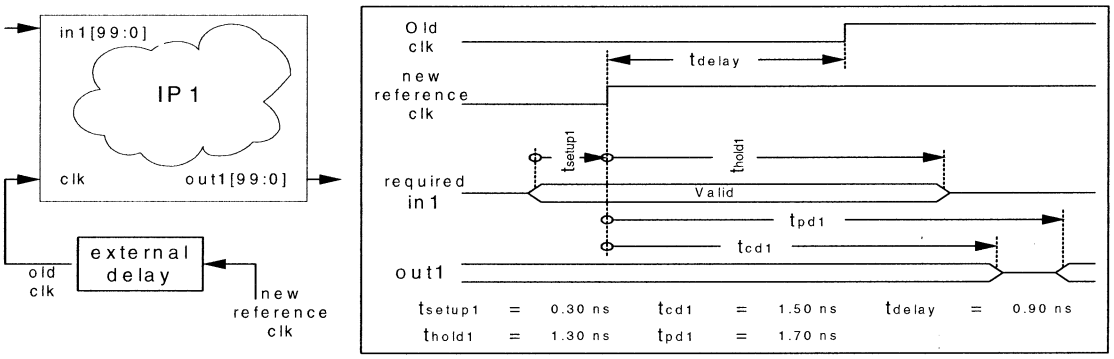


Fig. 8. New timing diagram for IP1. The hold time violation between out1 of IP1 and in2 of IP2 is removed.

delay of the clock to IP2 flip-flops cannot be sped up since the clock tree in IP2 cannot be redesigned. Common solutions to this situation are based on delaying these early signals of IP1. This is known as *data delay insertion*. Synthesis tools typically use the data delay insertion method. Delays are inserted on the interconnecting bus in order to remove any setup or hold time violations. In the example, a 0.5-ns delay would be added to each of the 100 lines of data bus, resulting in e.g., 400 added inverters, as demonstrated in Fig. 6. Obviously, the data delay insertion method may be expensive in terms of area and power.

The preferred solution as shown in Fig. 7, is to align the clock and to turn the SoC into a zero clock skew, or clock aligned system. This method is referred to as the *clock-delay insertion* method. As described above, the internal clock delay of IP2 is 0.9 ns. An external delay of 0.9 ns is prepended to the clock signal of IP1, affecting a new timing relationship between the

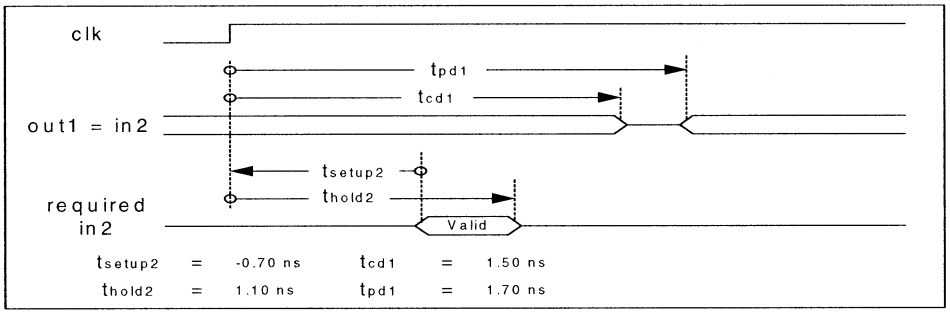


Fig. 9. New combined timing diagram of IP1 and IP2 (cf. Fig. 5.).

IP cores. Now the clock reference point is not the clock port of IP1, but rather the clock signal preceding the additional clock delay. The new timing diagram of IP1 with the new clock reference point is demonstrated in Fig. 8. The new combined timing diagram of IP1 and IP2 is presented in Fig. 9.

In typical SoC designs, the clock delays are added manually between the clock distribution network and the clock port of each IP core. This paper proposes a programmable method for inserting clock delays.

III. Clock-Delay Insertion Methodology

A. Delay Insertion Algorithm

A typical clock distribution network is shown schematically in Fig. 10(a). The network consists of a balanced clock tree where the delay from the root to each leaf is the same. Thus, the clock inputs of all IP cores receive the same clock phase. The approach presented in Fig. 10(a) is easy to design and implement, but as shown in Section II, it may require data delay insertion for correct operation of the complete SoC. Alternatively, the clock-delay insertion method enables a different total clock delay to each IP core, as demonstrated in Fig. 10(b). These delays compensate for the different internal clock delays of the various IP cores. The complete SoC is thus clock aligned with zero skew among all state elements in all IP cores.

The clock insertion method is based on the following algorithm:

$D := \max\{d_i\}$
for each IP core $i = 0 \ldots N$
Add clock delay $\Delta_i = D - d_i$;

where d_i is the internal clock delay of IP core i. Optionally, $D' = D + \Psi$ may be employed instead of D, with some $\Psi > 0$. The added delay Ψ leaves margin for future changes, in case the largest internal clock delay exceeds D.

Fig. 10(b) demonstrates the result of applying this algorithm to the network of Fig. 10(a), given the set of IP cores of the SoC. Note that knowing the internal clock delays (t_{int}) of all IP cores is essential for the clock-delay insertion algorithm. For soft IP cores, the designer can compute these delays. If the internal clock delay is not supplied for hard IP cores, the clock-delay insertion method may be inapplicable. Unfortunately, the internal clock-delay information is not a standard part of IP core delivery (the "black box" representation includes no such data) [9]. Designers who wish to apply the algorithm should require their hard IP core providers to supply the internal clock-delay information with the IP core delivery [6]. In some cases, it may be possible to estimate that internal clock delay from the provided I/O timing specifications.

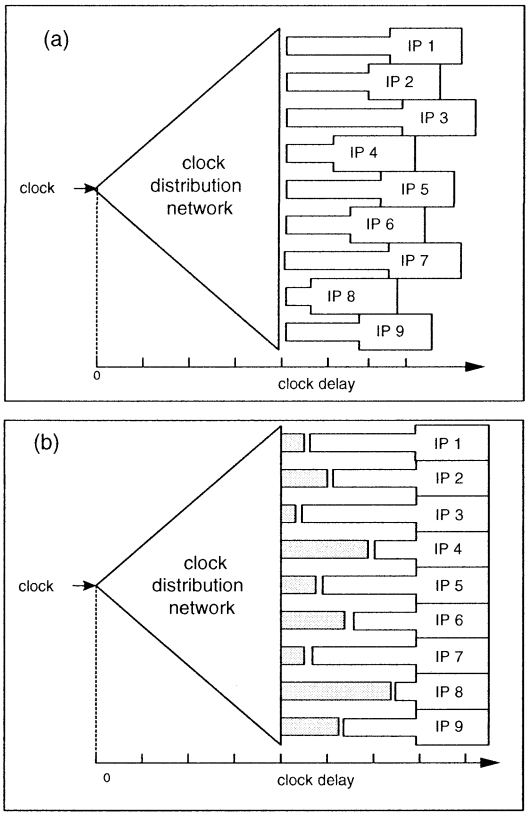


Fig. 10. (a) An aligned clock distribution network driving IP cores having different internal clock delays; the SoC is not clock aligned. (b) Clock-delay insertion compensates for the different internal clock delays, leading to a clock aligned SoC.

Some circuits employ nonzero clock skew intentionally [22]. Such "useful skew" methods can also be applied to SoCs. Our clock-tuning algorithm can be modified to generate useful skew by means of clock-delay insertion. That method and its application are not discussed in this paper.

B. Global Clock Redesign

The implementation of clock distribution networks is not straightforward. Ideally, when designing a global clock distribution network, changes in one IP core should not affect other parts of the system. In practice, however, changing an IP core

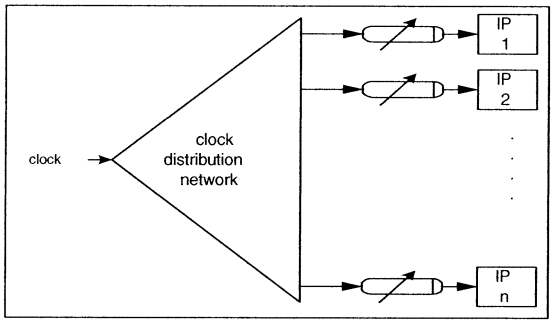


Fig. 11. SoC with static clock-tuning unit.

might change its layout, wire capacitances, resistances, etc. Such changes may affect the entire clock distribution network and may require its redesign. We call this phenomenon "back propagation." Some typical clock distribution architectures, such as delay-matched serpentines, and symmetric trees [3], [12], complicate the application of clock insertion. Global redesign of such clock networks incurs significant changes of the network at the cost of a heavy engineering effort even for small changes.

The clock-delay insertion method involves the following stages.

1) Adding delay elements to the relevant paths according to the algorithm.
2) Delay extraction of the clock distribution network inside each block, as well as that of the global clock distribution network.
3) Timing verification of all clock delays and design iteration if required.

The process is repeated whenever any of the system parts is changed. Therefore, proposed changes to the system are not easily accepted. This global clock redesign is far from the desirable "plug-and-play" concept of true modular design [5]. We propose an alternative implementation, which eliminates the need for repetitive global clock redesign.

C. Clock Tuning

We propose a novel and efficient implementation of clock-delay insertion. The implementation is based on programmable clock-delay lines [1]. The delay units are inserted at the clock input of each IP core (Fig. 11). Delay values are computed at the very last stage of the design, once the rest of the SoC design is finalized. The delay units are programmed by hardwiring their control bits.

The most important advantage of the programmable clock delay is the elimination of repeated clock network redesign every time any IP core is changed. Another benefit of this method is the ability to employ an unbalanced global clock distribution network, as explained in the next section.

IV. Unbalanced Global Clock Distribution Network

As described in Section III, the global clock distribution network of Fig. 10(b) is balanced. This balance is typically achieved at a high cost in terms of design time and effort, as well as chip area and power. In many cases, the clock distribution network must be predesigned before the other parts of the SoC (because it demands placement and routing resources, which might not be available at a later phase of the design, and due to time-to-market considerations). These complex demands are major obstacles to modular design and are also heavy time and effort consumers.

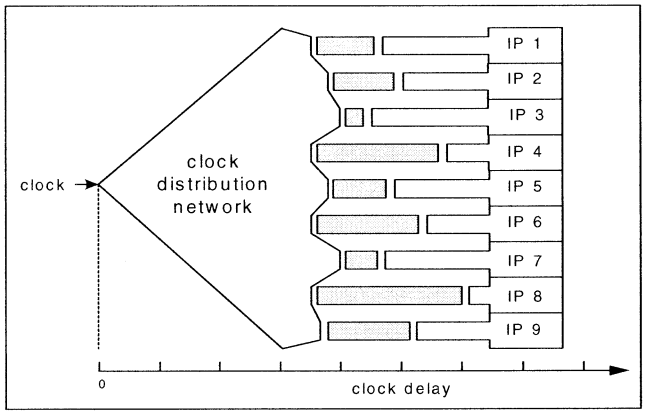


Fig. 12. An unbalanced clock distribution network. Clock-delay insertion compensates for the unbalanced clock distribution network as well as the different internal clock delays. The result is a clock aligned SoC. Gray rectangles represent inserted clock delays (either programmable or regular).

Unbalanced clock distribution networks may save a lot of time and effort in modular design. The clock skew of the unbalanced network is compensated for by the same inserted clock delays that also compensate for different internal clock delays inside the IP cores, as in Fig. 12. Notice again that the clock-tuning process is carried out only once at the end of the design process.

V. Hierarchical Clock Tuning

In hierarchical SoC design, some IP cores may be nested inside others. Higher-level IP cores supply the clock signal to the lower-level IP cores. This concept is useful for large modular design with many IP cores and modules, as well as for IP core vendors who integrate and resell multiple small IP cores in one bigger core.

Hierarchical clock tuning is carried out bottom up. First, the clock is tuned separately for each of the lower IP cores, using the shortest possible delays. As the algorithm ascends the hierarchy, the different branches of the tree are aligned with each other, until a fully aligned SoC is achieved. Fig. 13 exemplifies the method for a three levels hierarchy. The algorithm traverses the hierarchy recursively, as follows:

```
1. Generate a tree of clock distribution
 networks according to the SoC hierar-
 chical structure
2. Call H-tune(root).
/* The recursive routine H-tune is invoked
 for each "cell."
* It assigns tuning delays to the sub-
 cells and returns the cumulative clock
 delay
```

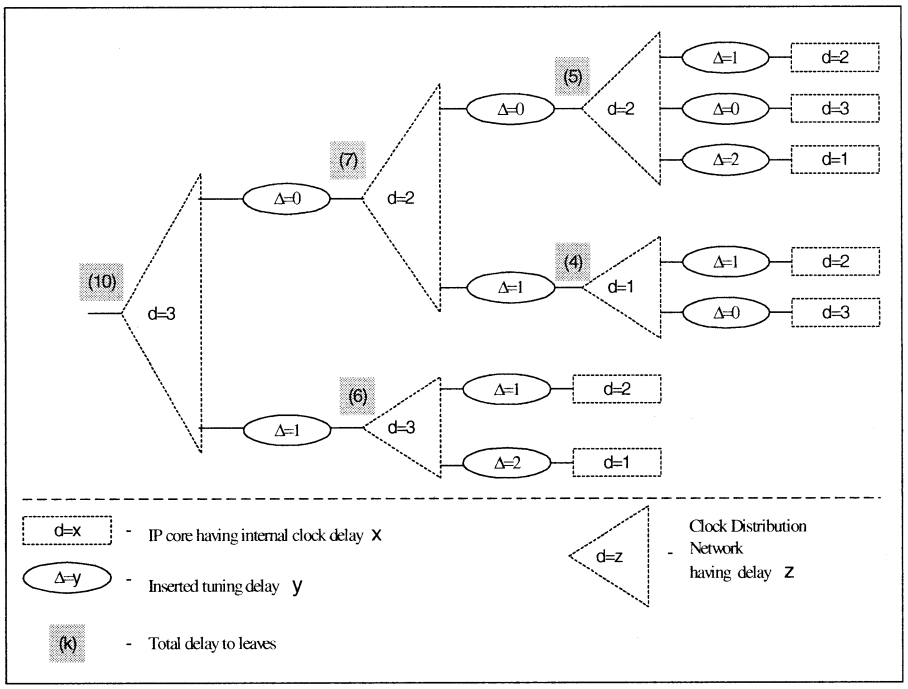


Fig. 13. Numeric example of hierarchical clock tuning. Numbers in parentheses represent cumulative clock delay from the leaves. Each subtree is independently clock aligned.

```
* from the cell's clock input pin to the
 FF's at the lowest level of the hier-
 archy.
*/
 H-tune(Cell) {
  S = set of sub-cells contained in Cell;
  Foreach (sub-cell jεS) {
   if (sub-cell j is a leaf IP core)
    d_j = sub-cell j's internal clock delay;
   else
    d_j = H-tune(sub-cell); /* proceed recur-
 sively */
  }
  D = max{d_j};
  Foreach (sub-cell jεS)
   add tuning delay Δj = D − dj; /* Tuning
 at the clock entry to sub-cell */
  distribution_delay = Get_distribution_delay(cell) /*
 Delay of clock distribution network at
 this level */
  Return (D + distribution_delay);
 }
```

Assuming that most data communication is local, the static and random skews introduced by the tuning circuits in Fig. 13 are minimal. This issue is discussed further in Section VII.

VI. Combined Dynamic and Static Clock Tuning

Typical clock distribution networks employ either delay-locked loops (DLLs) or phased-locked loops (PLLs). The former aligns the internal clock with the external one, while the latter can also provide frequency synthesis, clock conditioning, duty cycle correction and phase shifting. In either case, the clock distribution network can still benefit substantially from delay insertion.

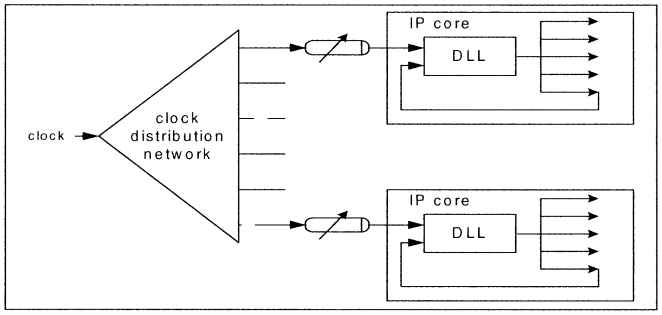


Fig. 14. Combined dynamic and static clock tuning employing DLLs at the IP cores. The DLLs align all clocks inside the IP cores with those entering the IP cores, and static tuning by means of programmable inserted delays compensate for any lack of balance in the clock distribution network.

Consider Fig. 14. Each IP core employs a DLL to align each internal clock with the clock entering the IP core. It is assumed that the delays to all memory elements inside the IP core are equal. Any imbalance on the global clock distribution network, however, is not accounted for, and different IP cores may be misaligned with each other. The inserted delays can compensate for an unbalanced network, as in Section IV.

In Fig. 15, a DLL is employed for the global clock distribution network as well as for each of the IP cores. This scheme is designed to align all flip-flops with the external clock. As above, it may be more effective to apply inserted delays to compensate for an unbalanced clock distribution network than to invest the resources required to balance it.

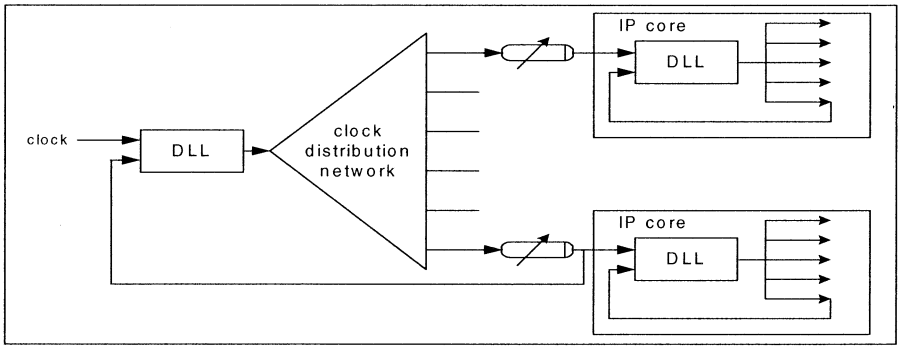


Fig. 15. Combined dynamic and static clock tuning employing two levels of DLLs. The DLL on the left aligns the clocks at the input of the IP cores with the external clock. The remaining DLLs align each IP core. Any lack of balance in the global clock distribution network is compensated for statically by the programmable delays. The clock signal arrives at the flip-flops two whole clock cycles after entering the SoC.

VII. Incremental Skew Analysis

Inserting delays into the clock distribution network incurs a certain increase in skew. In the following, we analyze the incremental skew added to the clock by the clock-tuning circuits. Clock skew can be categorized into three basic types: deterministic, static, and random [18]. Deterministic skew results from delay mismatch among the various branches of the distribution network, e.g., a different number of gates or a different wire length. Static skew results from process, voltage, and temperature (PVT) variations. Random skew results from various types of dynamic noise. Clearly, deterministic skew can be extracted at the design time while static and random skews can only be statistically characterized.

The clock distribution network in our SoC consists of three levels (Fig. 16): a global distribution network, programmable clock-delay units, and the IP core internal clock distribution network. While each level makes its own contribution to the total clock skew, we are interested here only in that of the programmable delay unit.

As explained above, the clock-delay unit attached to IP core i (which has an internal clock delay d_i) is programmed to the value $(D - d_i)$, where D is the largest internal clock delay over all IP cores in the subtree. For the sake of implementation simplicity we opt to implement the programmable clock delay as a series of identical digital buffers (pairs of balanced inverters), and thus, the nominal delay of a single buffer, d_r, defines the delay resolution. Each clock-delay unit is programmed to engage k_i of its buffers, creating a (quantized) delay of $k_i \cdot d_r$. Thus, we may be unable to generate exactly the desired delay value. However, this mismatch is bounded; given that programmable clock-delay unit i engages k_i delay stages (buffers), the mismatch, as demonstrated in Fig. 17

$$|k_i \cdot d_r - (D - d_i)| \leq \frac{d_r}{2}. \tag{2}$$

Therefore, the maximal deterministic skew between any two IP cores is d_r.

All PVT-related static skew variations (from the mean value of deterministic d_r) can be lumped within $\pm\delta_s$. We also assume that all clock-delay buffers on the SoC track PVT variations in the same direction (ignoring in-die variations as a worst case). Therefore, the worst scenario between any two IP cores is

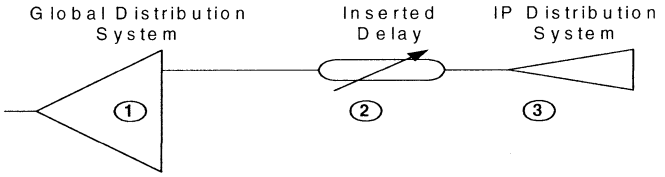


Fig. 16. Basic SoC clock distribution network.

$$\begin{aligned} k_i \cdot d_r &\rightarrow k_i \cdot (d_r - \delta_s) \\ k_j \cdot d_r &\rightarrow k_j \cdot (d_r + \delta_s) \\ \mathrm{Max}T_{\mathrm{skew}}(i,j) &= \delta_s \cdot (k_i + k_j) = 2 \cdot k_{\max} \cdot \delta_s. \end{aligned} \tag{3}$$

The added static skew due to such PVT variations is $\delta_s \cdot (k_i + k_j)$. Thus, the maximal static skew is $2 \cdot k_{\max} \cdot \delta_s$.

In the case of random skew, noise-related delay variations do not all happen in the same direction. Hence, assuming that the dynamic delay variation of a single clock-delay buffer is $\pm\delta_d$ (around the mean value d_r) the combined contribution of $2k_{\max}$ independent random variables is $\sqrt{2k_{\max}}\delta_d$. Adding all three skew sources together we get

$$\text{total additional skew due to clock-delay insertion} \leq d_r + 2k_{\max}\delta_s + \sqrt{2k_{\max}}\delta_d. \tag{4}$$

In conclusion, we observe that to minimize skew, d_r should be designed as small as possible and the delay lines should be kept as short as possible. In particular, it is advisable to minimize delay chains at the lower levels of the clock distribution hierarchy, in order to minimize skew among nearby components where intercommunication is more likely than among more distant parts.

VIII. Clock-Delay Tuning Circuits

A. Circuit Parameters

Three parameters should be controlled for the programmable clock-delay unit: minimum delay ($d_{\min}$), maximum delay ($d_{\max}$), and delay resolution (d_r). They should be chosen carefully in order to enable maximal future flexibility and correct operation of the circuit. While typically we optimize

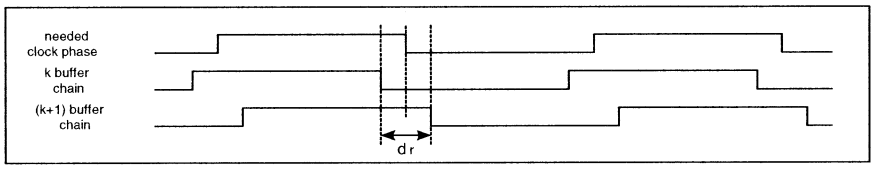


Fig. 17. Delay gap between two successive buffers in a buffer chain.

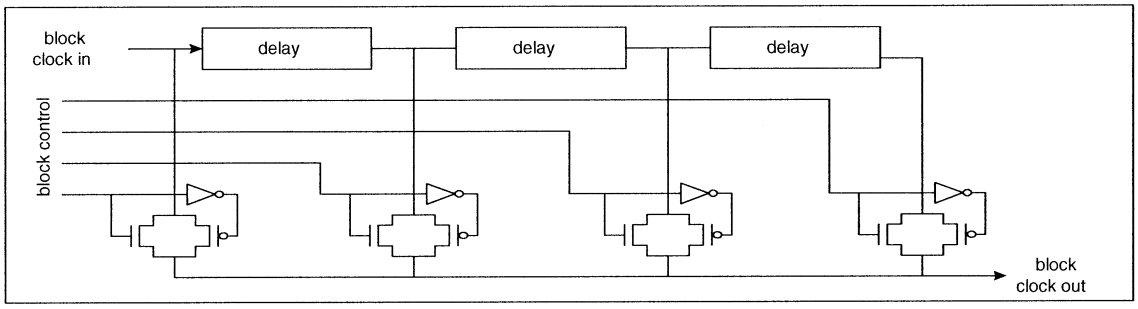


Fig. 18. Tapped delay line block, providing 0, 1, 2, or 3 minimum delays. The delays may consist of a pair of inverters. Multiple blocks can be concatenated to obtain longer delays.

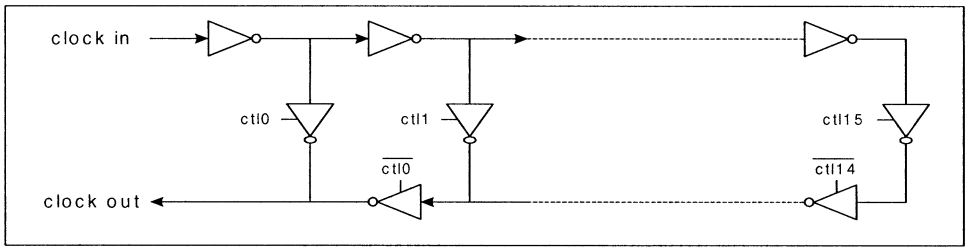


Fig. 19. Mirror delay line.

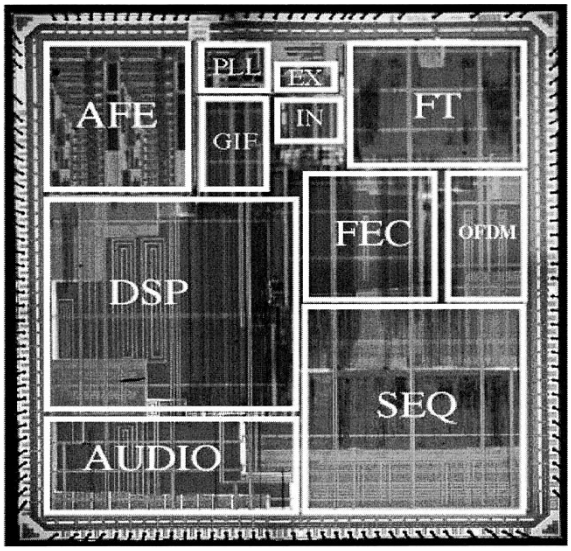


Fig. 20. Chip micrograph.

for the smallest minimum delay, a certain margin may provide flexibility for accommodating future SoC changes. Since accurate internal clock delays of all IP cores are unknown at the beginning of the design process, a certain margin should be planned into $d_{\max}$.

As discussed above, d_r affects the clock skew of the circuit. High resolution calls for many stages in the unit decoder and might be an overkill for the circuit, while low resolution might cause clock skew and degrade performance.

TABLE I
SOC PARAMETERS

Device Count	12M
Die Size	6.3x7.1mm^2
Frequency	200MHz
Supply Voltage	1.8/3.3V
Power Dissipation	<1W
Metal Layers	6
Minimum Feature Size	0.18μm
Package	128 pin QFP

B. Delay Circuit

Two delay circuits are considered, a tapped delay line and a mirror delay line. The former (Fig. 18) comprises two blocks in series, where the first contains three buffers and can be programmed for a delay of 0, 1, 2, or 3 minimum delays, and the second block comprises three stages of four delay buffers each, providing delays of 0, 4, 8, or 12 minimum delays. The two blocks can thus be programmed for zero through fifteen buffer delays. Note that even with zero delay buffers the total delay is nonzero due to the taps.

The mirror delay line (Fig. 19) employs three-state inverters and eliminates the taps [20]. While the circuit is simpler, the delay resolution is slightly larger than in the former case since

TABLE II
UNITS PROGRAMMING IN THE SOC

Block	Internal Clock Delay	Added Programmable Delay (ns)	Total Clock Delay (ns)
Audio	0.95	0.95	1.9
DSP	1.1	0.75	1.85
GIF	0.7	1.25	1.95
FEC	1.05	0.85	1.9
FT	0.6	1.35	1.95
OFDM	0.85	1.1	1.95
EX	0.4	1.55	1.95
IN	0.8	1.1	1.9
SEQ	0.7	1.2	1.9

the propagation delay of a three-state inverter is somewhat longer than that of a simple inverter.

IX. EXPERIMENTAL RESULTS

The SoC that incorporates the programmable clock-delay circuits is a multistandard demodulator and decoder for terrestrial and cable DTV and analog TV reception (Fig. 20 and Table I). It is designed to support 2 K/8 K-OFDM, 4–256 QAM, and QPSK in full compliance with DVB-T, DVB-C, and DVB-S digital television standards. Its basic function is to recover the digital data encoded into the broadcast signal, which includes video and audio program information and auxiliary data. The device outputs the demodulated data as a standard MPEG-2 transport stream in either parallel or serial format.

Table II describes the final programming of the clock-delay units in the ten IP cores of the SoC. The programmable clock-delay units were placed in each one of the modules marked in Fig. 20.

As explained in Section IV, an important advantage of the programmable clock-delay circuits is the ability to use an unbalanced global clock distribution network. The programmable clock-delay units compensate for the unbalanced distribution network and enable easy clock balancing at the IP level. Fig. 21 schematically shows the layout of the unbalanced clock tree of the chip of Fig. 20.

Productivity of the proposed clock-tuning method was proven very high. Weeks of iterative clock distribution network design were reduced to several days in which the complete network was designed, tuned and tested. The implementation demanded several design flow changes with standard computer-aided design (CAD) tools (such as synthesis, scan generation, and static timing analysis).

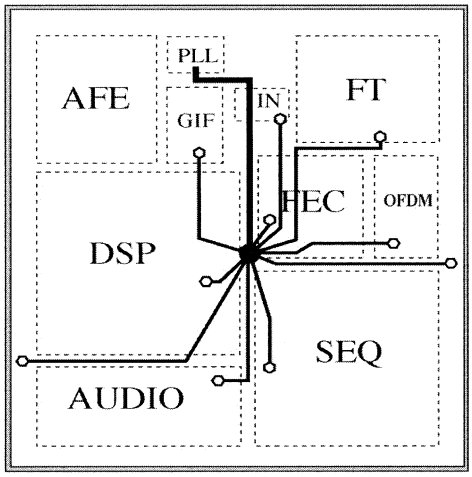


Fig. 21. Global clock distribution network. The PLL sends the clock signal to the center of the chip. All IP cores receive the clock signal from the buffer in the center. The connections are unbalanced and no delay matching effort had to be invested. The delays are balanced merely by means of the programmable delay units.

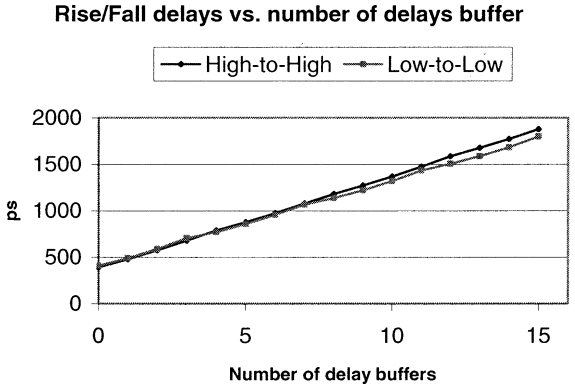


Fig. 22. Rise/fall delay from simulations of the programmable delay unit.

X. PERFORMANCE ANALYSIS

A. Delays of the Programmable Tapped Delay Line

We have simulated the tapped delay line under PVT conditions of typical *n* and *p* transistors, $T = 100°\text{C}$ and $V_{dd} = 1.8\,\text{V}$. As can be deduced from Table III and Fig. 22, $d_r = 94.3$ ps, and the average inverter delay is 47 ps. Note that our buffers are not perfectly symmetric, resulting in a small mismatch between the high and low delays. As clearly shown by Table III, the deterministic skew is only 100 ps.

B. Corner Variation Analysis

To investigate the effect of the choice of circuit and the number of buffers on skew, we have performed PVT corner simulations of four different delay circuits—a programmable tapped delay line at maximum delay of 15 buffers, a simple delay line without taps having 22 buffers (Fig. 23), a delay line with longer channel transistors (0.36 μm) and only eight

TABLE III
DELAYS FOR EACH STATE IN SIMULATIONS OF THE PROGRAMMABLE DELAY LINE

Number of engaged buffers	High-to-High delay [ns]	Low-to-Low delay [ns]
0	0.391	0.411
1	0.480	0.492
2	0.575	0.590
3	0.681	0.707
4	0.788	0.771
5	0.877	0.857
6	0.973	0.957
7	1.079	1.068
8	1.181	1.136
9	1.272	1.222
10	1.368	1.319
11	1.473	1.432
12	1.587	1.507
13	1.677	1.586
14	1.773	1.683
15	1.879	1.799

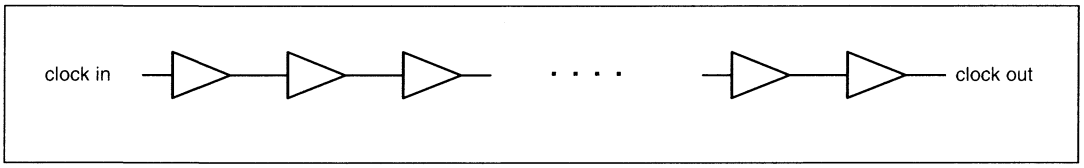


Fig. 23. Alternative design – simple delay line of 22 buffers.

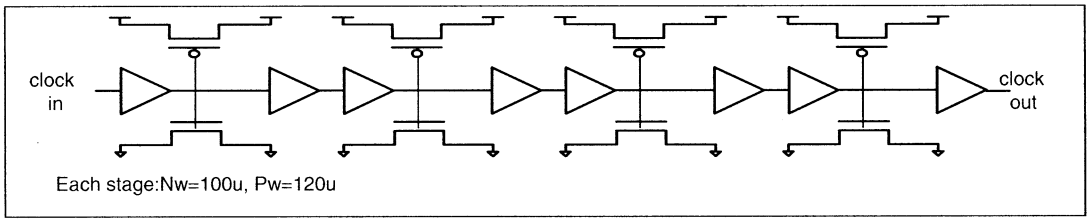


Fig. 24. Alternative design – capacitive load delay line.

TABLE IV
MINIMUM/MAXIMUM DELAYS AND DELAY SPAN AS SIMULATED FOR VARIOUS DELAYSTRUCTURES

	Programmable Delay Line (15 buffers)	Simple Delay Line (22 buffers)	Long Channel Transistors Delay Line (8 buffers)	Capacitive Load Delay Line (8 buffers)
Minimum Delay (ns)	1.25	1.28	1.37	1.35
Maximum Delay (ns)	2.70	2.80	2.93	2.81
Delay Span (ns)	1.45	1.52	1.56	1.46

delay stages, and a capacitive load delay line of eight buffers (Fig. 24). All were designed to achieve similar delay under nominal conditions. The results are presented in Table IV.

As can be seen in Table IV, the proposed programmable delay line outperforms all other designs as it slightly less sensitive to process corner variations. The simple delay line has higher delay span due to increased number of stages. The other two designs, although having significantly fewer stages, exhibit higher delay span due to the longer channel and the higher capacitive load.

XI. CONCLUSION

Two methods for timing integration of IP cores in SoC were discussed: data delay insertion and clock-delay insertion. Data delay insertion was shown inefficient in terms of power and area. Clock-delay insertion requires frequent redesign of the clock distribution network every time any of the IP cores is changed. Such redesign varies in different application scenarios; greater flexibility is possible in SOFT IP cores as compared to hard IP cores. In any case, this clock network redesign carries a high price in design time and engineering resources.

Clock tuning, employing programmable clock-delay units, alleviates the need for global clock redesign. In addition, it can deal with unbalanced clock distribution networks [21], by providing programmable delay compensation. With this methodology, geared particularly toward SoC design, electrical parameter extraction needs to be carried out only once. Programming the inserted delays and simulating the circuit for clock timing verification is the only required effort. If any of the clock delays need to be changed, the clock tuning is reprogrammed and timing verification is repeated. Changes are possible at any stage of the design, and a quick verification is always possible. The complete design flow becomes more flexible and the burden of timing verification is reduced.

These advantages do not come without a price; a fault in the programmable clock-delay unit may leave it stuck at a certain delay value. This fault may lead to complete circuit failure. Short of such complete functional failures, which are easily testable, a timing fault may not be functionally testable but rather require testing at full high frequency to be detected. Similar limitations exist in traditional clock distribution systems in case of a delay-fault in a clock buffer, but the added circuitry in the tuning units might increase the likelihood of such a problem.

In summary, a clock distribution strategy for integrating IP cores in SoCs has been proposed, analyzed, and demonstrated in a commercial chip. It improves ease of IP cores reuse by enabling simple clock tuning. Using clock-tuning circuits with programming options enables easy integration of many IP cores into a complete SoC, and eliminates design iterations in the engineering flow. Thus, design effort is reduced, design modularity is improved, and "last minute changes" are enabled.

REFERENCES

[1] W. J. Dally and J. W. Poulton, *Digital Systems Engineering*. Cambridge, U.K.: Cambridge Univ. Press, 1998.

[2] K. D. Wagner, "Clock system design," *IEEE Des. Test Comput.*, pp. 9–27, Oct. 1988.

[3] P. J. Restle and A. Deutsch, "Designing the best clock distribution network," in *Proc. 1998 Symp. Very Large Scale Integration (VLSI) Circuits*, 1998, pp. 1–5.

[4] R. Ginosar and R. Kol, "*Adaptive synchronization*," in *Proc. 2000 Workshop Asynchronous Interfaces (AINT)*, Delft, The Netherlands, July 2000.

[5] C. K. Lennard and E. Granata, "The meta-methods: Managing design risk during IP selection and integration," in *Proc. IP 1999 Europe*, pp. 285–299.

[6] M. Keating and P. Bricaud, *Reuse Methodology Manual for System-on-a-Chip Designs*. Norwell, MA: Kluwer, 1999.

[7] J. M. Rabaey, *Digital Integrated Circuits a Design Perspective*. Englewood, NJ: Prentice-Hall, 1996, Prentice-Hall Electronics and VLSI series.

[8] D. G. Messerschmitt, "Synchronization in digital system design," *IEEE J. Selected Areas Commun.*, vol. 8, pp. 1404–1419, Oct. 1990.

[9] VSI Alliance Architecture Document. VSI Alliance, Los Gatos, CA. [Online]. Available: http://www.vsi.org/library/specs/summary.htm

[10] D. Sylvester and K. Keutzer, "Rethinking deep-submicron circuit design," *IEEE Computer*, pp. 25–33, Nov. 1999.

[11] R. Ho, K. W. Mai, and M. Horowitz, "The future of wires," *Proc. IEEE*, vol. 89, pp. 490–504, Apr. 2001.

[12] D. W. Baily and B. J. Benschneider, "Clocking design and analysis for a 600-MHz alpha, microprocessor," *IEEE J. Solid-State Circuits*, vol. 33, pp. 1627–1633, Nov. 1998.

[13] H. Mizuno and K. Ishibashi, "A noise-immune GHz-clock distribution scheme using synchronous oscillators," in *Proc. IEEE Int. Solid-State Circuits Conf.*, 1988, pp. 404–405.

[14] S. A. Ward and R. H. Halsted, *Computation Structures*. Cambridge, MA: MIT Press, 1990.

[15] E. G. Friedman, *Clock Distribution Networks in VLSI Circuits and Systems*. Piscataway, NJ: IEEE Press, 1995.

[16] S. Rusu and G. Singer, "The first IA-64 microprocessor," *IEEE J. Solid-State Circuits*, vol. 35, pp. 1539–1544, Nov. 2000.

[17] S. Tam, S. Rusu, U. N. Desai, R. Kim, J. Zhang, and I. Young, "Clock generation and distribution for the first IA-64 microprocessor," *IEEE J. Solid-State Circuits*, vol. 35, pp. 1545–1552, Nov. 2000.

[18] C. -S. Li, K. N. Sivarajan, and D. Messerschmitt, "Statistical analysis of timing rules for high speed synchronous VLSI systems," *IEEE Trans. VLSI Syst.*, vol. 7, pp. 477–482, Dec. 1999.

[19] V. Gutnik and A. P. Chandrakasan, "Active GHz clock network using distributed PLL's," *IEEE J. Solid-State Circuits*, vol. 35, pp. 1553–1560, Nov. 2000.

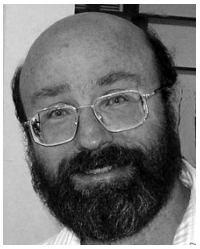
[20] T. Saeki, "A 2.5-ns clock access, 250 MHz, 256-Mb SDRAM with synchronous mirror delay," *IEEE J. Solid-State Circuits*, vol. 30, pp. 1656–1668, Nov. 1996.

[21] P. J. Restle *et al.*, "A clock distribution network for microprocessors," *IEEE J. Solid-State Circuits*, vol. 36, pp. 792–797, May 2001.

[22] I. S. Kourtev and E. G. Friedman, *Timing Optimization through Clock Skew Scheduling*. Norwell, MA: Kluwer, 2000.

Yaron Elboim received the B.Sc. degree in electrical engineering from the Technion—Israel Institute of Technology, Haifa, in 1998.

He was with Oren Semiconductor, Yoqneam, Israel, where he performed logic design and system integration of IP core-based SoC. Since 2001, he has been with Intel Corporation, Israel, where he works on communication products.



Avinoam Kolodny (M'81) received the D.Sc. degree in electrical engineering from the Technion—Israel Institute of Technology, Haifa, in 1980.

He has worked on silicon technology development and design automation at Intel Corporation in Israel and in California. His research interests include VLSI design and CAD.

Ran Ginosar (S'79–M'82) received the B.Sc. degree (*summa cum laude*) in electrical and computer engineering from the Technion—Israel Institute of Technology, Haifa, in 1978, and the Ph.D. degree in electrical engineering and computer science from Princeton University, Princeton, NJ, in 1982.

After working with AT&T Bell Laboratories for one year, he joined the Technion faculty in 1983. He was a Visiting Associate Professor with the University of Utah, Salt Lake City, from 1989 to 1990 and a Visiting Faculty Member with the Strategic CAD Laboratory, Intel Corporation, from 1997 to 1999. He is currently the Head of the VLSI Systems Research Center at the Technion. His research interests include asynchronous systems and electronic imaging.