

# Automating Wave-Pipelined Circuit Design

Woo Jin Kim  
Sun Microsystems

Yong-Bin Kim  
Northeastern University

Wave pipelining offers faster clock rates than conventional pipelining; however, wave-pipelined circuit design is time-consuming and requires a high level of expertise. By augmenting a commercial tool with delay-balancing scripts, this method automates the synthesis part of the process with minimal delay deviation.

■ **CONVENTIONAL PIPELINING** partitions the combinational logic into smaller chunks and inserts registers at the boundaries. Wave pipelining takes another approach, relying on the logic path being long enough and the data dispersion reasonably small, so the system can send multiple sets of data (waves) through the logic at a faster clock rate and without latching the data on the way.

Although wave-pipelined designs enjoy several advantages over their conventional counterparts, they require tighter timing constraints because they have no intermediate registers in which to store intermediate data. Wave pipelining is especially vulnerable to delay changes due to variations in process, voltage, and temperature (PVT) and in the operating environment. Whereas conventional pipelining's performance depends only on the maximum-delay path, wave pipelining's performance is also affected by the minimum-delay path. Thus, minimum delay is also a concern for wave-pipelined circuits. Automating wave-pipelined circuit design, especially generating a design netlist, is therefore challenging.

To automate the generation of wave-pipelined design netlists, we use a commercial synthesis tool, the Synopsys Design Compiler, and delay-balancing scripts. Results show less than 20% delay deviation between maximum and minimum delays, which is comparable to results from custom designs.

## Wave pipeline basics

To a first-order approximation, wave pipelines generally achieve a speedup of  $N$ , where  $N$  represents the

number of data waves that propagate simultaneously through the design. Conventional pipelines achieve a similar speedup, with  $N$  the number of pipe stages.

Two factors constrain conventional pipeline design: Worst-case delay between adjoining pipeline stages and the setup and hold times of registers at the design boundary. However, both the worst and best delays constrain wave pipelining because the clocking is a function of the difference between the two. Equation 1 expresses the simple clocking constraint:

$$T_{\text{CLK}} > T_{\text{DELAY}}(\text{max}) - T_{\text{DELAY}}(\text{min}) + T_{\text{SETUP}} + T_{\text{HOLD}} + 2 \times T_{\text{SKEW}} \quad (1)$$

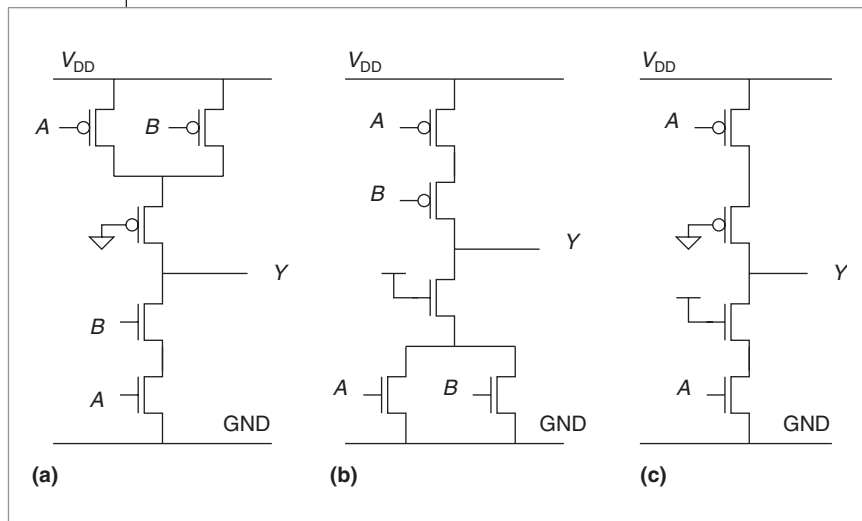
where  $T_{\text{CLK}}$  is the clock cycle,  $T_{\text{DELAY}}(\text{max})$  is the design's maximum propagation delay,  $T_{\text{DELAY}}(\text{min})$  is its minimum propagation delay,  $T_{\text{SETUP}}$  and  $T_{\text{HOLD}}$  are the register's setup and hold times, and  $T_{\text{SKEW}}$  is the clock skew. The sum of the setup and hold times and the clock skew is the clocking overhead.

Because wave pipelining requires no additional registers (and thus avoids their inherent delays), latency is lower than in conventional pipelining. A designer can write the wave-pipelined design's latency or, more accurately, the time it takes for the initial data to be latched at the output ( $T_{\text{DESIGN}}$ ), as

$$T_{\text{DESIGN}} = N \times T_{\text{CLK}} + T_{\text{SKEW\_GOOD}} \quad (2)$$

where  $T_{\text{SKEW\_GOOD}}$ , sometimes termed *constructive skew*, is the skew between the input and output register, which actually helps the design. It can be greater than or equal to zero but is less than  $T_{\text{CLK}}$ .

Similar constraints can be derived for internal nodes.<sup>1,2</sup>



**Figure 1. Expanded set of stacked CMOS logic: two-input NAND (a) and two-input NOR (b) gates, and inverter (c).**

### Design considerations

One focus of this research was simplifying the design effort for wave-pipelined circuits. Most (if not all) previous work has treated wave-pipeline design as a customized process. This gives more control to the designer, who can build regularity into the design, making it easier to balance maximum and minimum delays. Although custom processes produce the best design quality, they can be extremely time consuming and require an expert designer with detailed knowledge of the circuit.

Automating parts or all phases of wave-pipeline design is an obvious solution. We looked for areas in which we could use existing commercial tools to perform the bulk of the automated work. Generating the netlist is the most complicated phase in terms of time and expertise. We thus chose to automate this phase using Design Compiler, which contains the most widely used synthesis engine.

Wave pipelining is also more susceptible to process and environmental variations than conventional pipelining. In fact, wave pipelining is as much as six times as sensitive as conventional pipelining. We introduce a simple clocking interface to assure correct functionality in the face of PVT variations.

### Design synthesis

Although it is easy to come up with a workable design that meets our specifications at a high abstraction level, it is difficult at the gate and circuit levels. We want Design Compiler to synthesize the netlist from an abstract level to the gate level. The designer will then

take this netlist and edit it as necessary to achieve the desired specification.

To accomplish this, we first create a library of cells we expect will produce good results. Standard CMOS gates are not very balanced in terms of delay because of their topology, and each additional input aggravates this imbalance. Thus, wave-pipelined designs using CMOS gates usually limit the number of inputs to two.<sup>3</sup> Researchers have presented results using different types of logic cells, including pass gates and stacked structures.<sup>46</sup> Despite some drawbacks, the CMOS logic family best facilitated design.

### Expanded set of stacked CMOS logic

For his wave-pipelined design, Klass used only the stacked NAND gate,<sup>3</sup> restricting it to two inputs to minimize the gate's delay variance. To increase the functions with which Design Compiler could generate the netlist, we expanded the stacking concept to include a two-input NOR gate and inverter (INV), giving the synthesis tool more blocks to play with while keeping the two-input restriction. Figure 1 illustrates the expanded set of stacked CMOS logic.

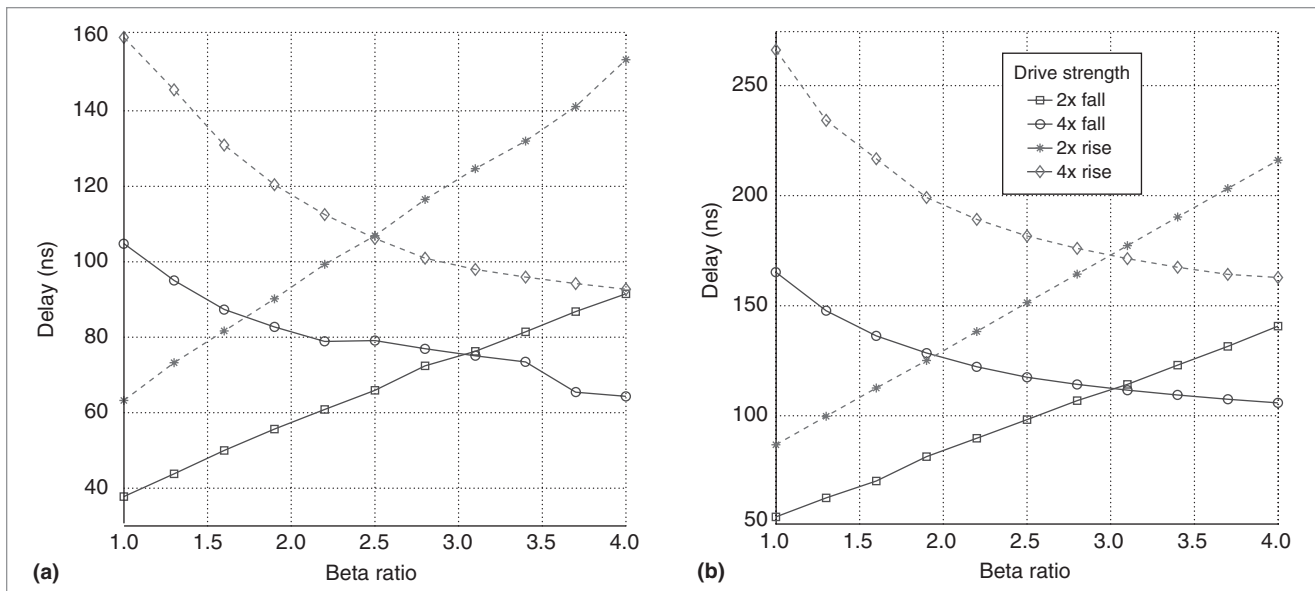
One thing that stands out in Figure 1 is that for all cells, the depth of the transistors is the same for P and N networks. In typical static CMOS circuits, the depths for the P and N networks differ as do the depths of the individual gates. It was primarily for this reason that Klass stacked the NAND2 gate and did not use NOR2. This configuration sets the gate depth at the output pin to two for all three cells. Doing so makes delays in the first-order match for all three of these stacked circuits.

The longer overall delay in stacked CMOS gates comes from transistor depth. The difference between rise and fall delays, however, is significantly smaller than that of regular CMOS gates. Figure 2 shows the beta versus delay trend of the regular and stacked configurations; these less-variable rise and fall delays make it easier to match (the delay) using the stacked CMOS gates.

### Synthesis

The next phase is design synthesis. Our target designs were adders and multipliers. For adders, we ran 4-, 16-, and 32-bit designs; for multipliers, the designs were 8 and 16 bits.

Table 1 shows the results using the regular CMOS



**Figure 2. Beta versus delay for regular (a) and stacked (b) CMOS.**

library on the designs. Table 2 shows the results using the stacked CMOS library. The tables give final results, including the minimum delay compilation for delay matching.

For larger designs, such as 32-bit adders and 16-bit multipliers, Design Compiler does not effectively add buffers to meet the minimum constraint. For smaller designs, such as 4- and 16-bit adders and 8-bit multipliers, the delay difference drops to below 31% for both regular and stacked CMOS. This difference is not bad compared to earlier work, which reported a 21.6% delay difference for a custom-designed 8-bit adder that researchers had fine-tuned.<sup>4</sup>

#### Balancing algorithm

Because Design Compiler does not adequately balance delay, especially in large designs, we apply a balancing algorithm to the netlist.

Our balancing scheme partitions the netlist into lev-

els but forces the levels to have a gate depth of one. Limiting the number of delay elements per level that are insertable into any path limits the number of overall delay elements. This guarantees that the whole design's gate depth will stay the same—matched to the design's worst path in terms of gate depth—and that the design latency will not change, theoretically, as a consequence of delay element insertion. Adding fewer

**Table 1. Synthesis results using regular CMOS.**

Design	Maximum delay	Minimum delay	Difference		Maximum depth (No. of gates)
	(ns)	(ns)	(ns)	(Percentage)	
ADD4	0.72	0.60	0.12	16.7	10
ADD16	3.36	2.68	0.68	20.0	43
ADD32	6.52	3.02	2.50	53.7	85
MUL8	4.00	2.78	1.22	31.0	48
MUL16	8.55	1.60	6.95	81.0	103

**Table 2. Synthesis results using stacked CMOS.**

Design	Maximum delay	Minimum delay	Difference		Maximum depth (No. of gates)
	(ns)	(ns)	(ns)	(Percentage)	
ADD4	0.84	0.72	0.12	14.3	9
ADD16	3.46	2.06	1.40	40.5	36
ADD32	6.97	4.87	2.10	30.1	65
MUL8	4.07	3.48	0.59	14.0	42
MUL16	9.40	2.18	7.22	76.8	84

Table 3. Timing before and after delay match using regular CMOS gates.

Design	Before balancing			After balancing		
	Maximum delay (ns)	Minimum delay (ns)	Deviation (percentage)	Maximum delay (ns)	Minimum delay (ns)	Deviation (percentage)
ADD32	5.54	0.07	98.9	5.84	4.42	24.3
MUL16	8.55	0.07	99.2	8.78	6.28	28.5

Table 4. Timing before and after delay match using stacked CMOS gates.

Design	Before balancing			After balancing		
	Maximum delay (ns)	Minimum delay (ns)	Deviation (percentage)	Maximum delay (ns)	Minimum delay (ns)	Deviation (percentage)
ADD32	4.31	0.11	97.5	5.07	4.37	13.8
MUL16	9.39	0.11	98.8	10.40	8.76	15.8

delay components reduces the likelihood of increased delay difference arising from the components' inherent delay variance.

For this process to work, the library cells must be good in the sense that the delay of the different functional cells is close and the drive-strength-to-delay mapping is similar across cells. This similarity is necessary because cell selection will depend on the loading at the output pins.

Once generated, the netlist enters the Cell Mapping I (MapI) phase. (If the netlist was generated with appropriately selected cells, only this phase should be necessary.) MapI partitions the netlist into levels, leaving only one gate per level. If it finds paths that go through levels with no gates, it inserts default buffers for those paths on those levels. Working from the output, MapI updates the drives of each inserted buffer, depending on the loading. It then calculates delays for all paths, also keeping information on all gate delays. After completing the calculation, MapI checks path and gate delay numbers against other gates on the same level. If the results are satisfactory, MapI is complete.

MapI lets the designer quickly balance the design's delay if all the checks return an OK. If the checks throw up a flag, however, control switches to Cell Mapping II (MapII), which goes over the problematic gates and resizes them as necessary. It then performs damage control, updating the drives belonging to gates that send the signal into the updated problematic gate. MapII calculates delay and applies checks. If problem spots still exist, MapII repeats until the checks are clean or until it reaches the maximum iteration number.

## Results

We ran the two larger, poorly balanced designs—ADD32 and MUL16—using Design Compiler to balance the data flow. Tables 3 and 4 show the two designs' timing after we applied MapI.

Although somewhat rough, the numbers from MapI were satisfactory. Timing deviation was less than 20%, a loose reference point as much of the earlier work on wave pipelining borders was around that number. This is a huge improvement over the Design Compiler alone.

As expected, timing deviation results from the stacked CMOS gates were 5% to 10% better than from regular CMOS gates; however, total delay was worse for stacked CMOS.

## Self-delayed checking scheme

Designers initially believed that as long as a design satisfied wave-pipeline constraints, its clock period could have any value.<sup>2,7</sup> However, Lam<sup>8</sup> and Gray<sup>9</sup> showed that the clock period's valid region is discontinuous. (See the "Related Work on Process and Environmental Variations" sidebar for other research in this area.)

For simplicity, assume that the output register constraint is the determining constraint and that  $T_{\text{SKEW\_GOOD}} = 0$ . We can derive the following equation, applicable for  $N = 1$ :

$$\frac{\text{Max\_Time}}{N} < T_{\text{CLK}} < \frac{\text{Min\_Time}}{N-1}$$

If  $N = 1$ , the design reflects a traditional pipeline scheme in which clock period  $T_{\text{CLK}}$  depends on Max\_Time and is unbound above that time. When  $N = 2$ , the design is bound-

ed below by  $\text{Min\_Delay}/(N-1)$  and above by  $\text{Max\_Delay}/N$ .

Thus, unless a circuit is perfectly balanced (that is,  $\text{Max\_Time} = \text{Min\_Time}$ ), the valid clock range will be discontinuous.

We propose a simple mechanism that accounts for possible effects of process and environmental variations and guarantees that the chip will work even when the wave-pipelined circuit slows down or speeds up, and the design no longer aligns with the clock.

The basic idea of this self-delayed checking scheme is to skew the output register clocks and work the wave-pipelined circuit every other clock if the deviation between maximum and minimum delays becomes larger than the clock period.

#### Path mirroring

For path mirroring to work, the designer must know the worst and best timing paths in advance. The difference between the two, plus some overhead, will basically determine the wave-pipelined design's operating frequency.

The timing analysis performed on the circuit during the delay-balancing process should provide the worst and best timing paths. After singling out the paths, the designer should duplicate each one. The pin connections of these paths are the controlling pins. These controlling pins—for the gates in the duplicate paths—are connected in the same way as the original paths, except for the leftover input pins. These pins are connected to the appropriate power rail— $V_{DD}$  or  $V_{SS}$ —to make them uncontrolling and thus reduce the extra wiring, as Figure 3 illustrates.

To be effective, these duplicated paths must experience the same process and environmental variations as the original paths. Thus, the original and duplicate paths must be near each other; grouping the like cells of both paths and placing them together during layout makes this collocation possible.

Because the cells will sit next to each other, any variations would theoretically impact them in the same way. Designers can place and route these paths before placing any other circuit component to ensure that both paths have a mirror topology.

## Related Work on Process and Environmental Variations

Researchers have suggested several techniques to reduce the performance impact of process and environmental variations.

Researchers described an adaptive clock generator that adjusts the clock rate using a ring oscillator.<sup>1</sup> In this design, delay variations guarantee the correct clocking of  $N$  waves at all times. Another method is Nowka's adaptive power supply.<sup>2</sup> In this technique, the delay chain, constructed with the slowest anticipated CMOS process, indicates voltage changes. It compares input and output phases and generates a signal when the difference exceeds a certain threshold.

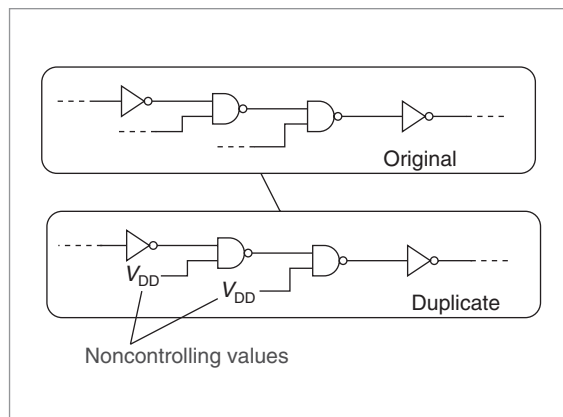
Biased logic gates can also help compensate for delay variation because of process variations.<sup>2,3</sup> Designers add series of extra biasing transistors to the existing transistors in both the NMOS and PMOS networks, setting the bias voltage to compensate for the delay variations. The driver-current starving method is similar to biased logic but applied on a more macro scale. It's similar in that biased voltage is applied to the design; however, it biases only the output driver.

Another approach uses a special output driver with controlled loading. The bias voltage of the shunted transistors controls the driving gate's effective loading—the receiver gate loading plus any additional loading that contributes to the driver output loading—so the delay can be adjusted.

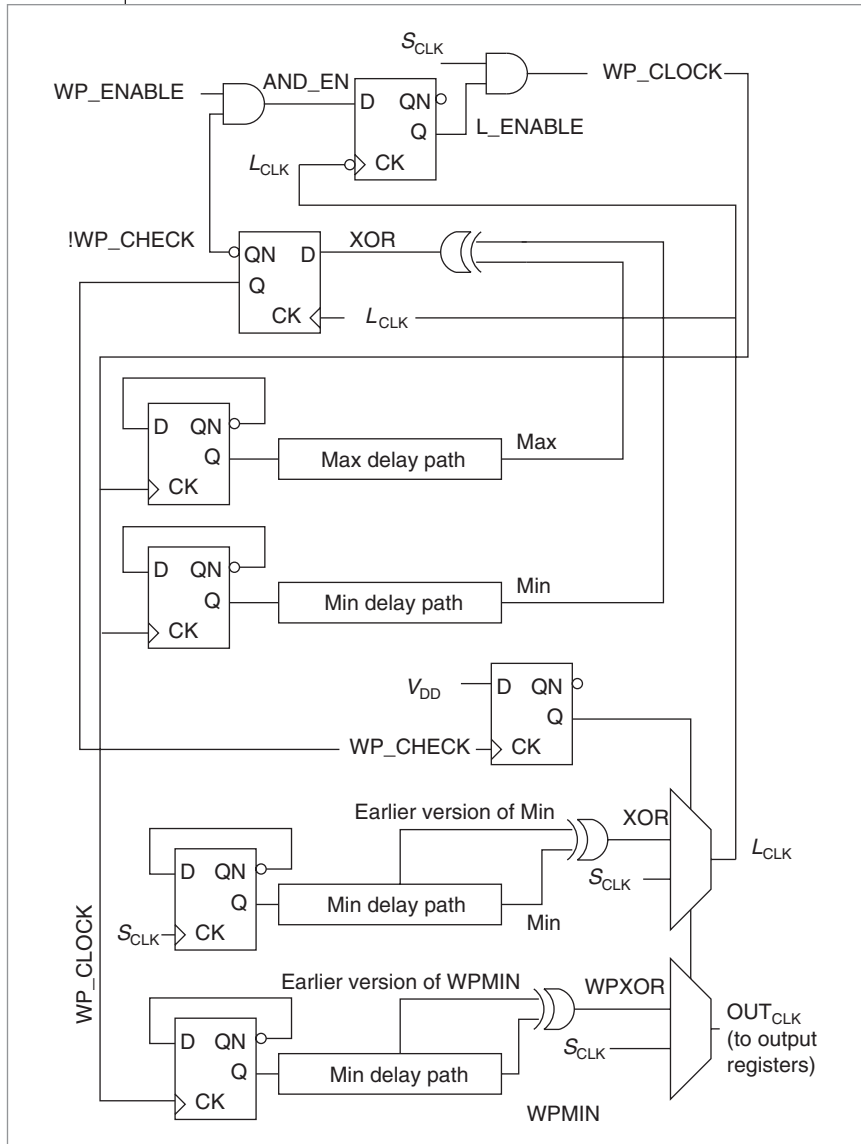
All these methods require some form of manual intervention by the designer, such as controlling the outside power supply to speed up or slow down the circuitry. Our research seeks to automate this design aspect.

## References

1. W. Burleson, F. Klass, and W. Liu, *Wave Pipelining: A Tutorial and Survey of Recent Research*, tech. report, Computer Systems Laboratory, Stanford Univ., 1998.
2. K. Nowka, *High-Performance CMOS System Design Using Wave Pipelining*, PhD dissertation, Dept. of Electrical Eng., Stanford Univ., 1995.
3. D. Fan et al., "A CMOS Parallel Adder Using Wave Pipelining," *Proc. MIT/Brown Conf. Advanced Research in VLSI and Parallel Systems*, MIT Press, 1992, pp. 147-164.



**Figure 3. Duplication of maximum and minimum paths.**



**Figure 4. Localized clock generation circuitry for wave-pipelined design.**

Depending on the length of the routing segments, coupling could affect delay, having more of an impact on the worst delay path. Maintaining the minimum spacing between the mirrored segments and using grid-like power rails (which can act as a shielding structure) would address this problem.

**Clocking interface**

Next the designer needs to create a circuit interface with the system clock and the rest of the chip. This interface is necessary for determining whether the wave-pipelined circuit can operate at the system frequency or whether the clock needs tweaking to accommodate the design.

First, the designer must determine if the path delay variation has changed so that the clocking at the output registers does not latch unstable data. Figure 4 shows implementation that prevents this latching of unstable data. In the figure,  $S_{CLK}$  is the system clock,  $L_{CLK}$  is the skewed version of the system clock, and  $WP\_CLOCK$  is the locally generated clock for the wave-pipelined design.

The Min and Max paths are duplicates of Min\_Delay and Max\_Delay paths. This circuit XORs the path outputs to create the XOR signal, which is latched onto the rising edge of the system clock to generate  $WP\_CHECK$ . The initial values are either  $V_{DD}$  or  $V_{SS}$ ; these values must be the same for registers on a paired set of paths (original and duplicate).

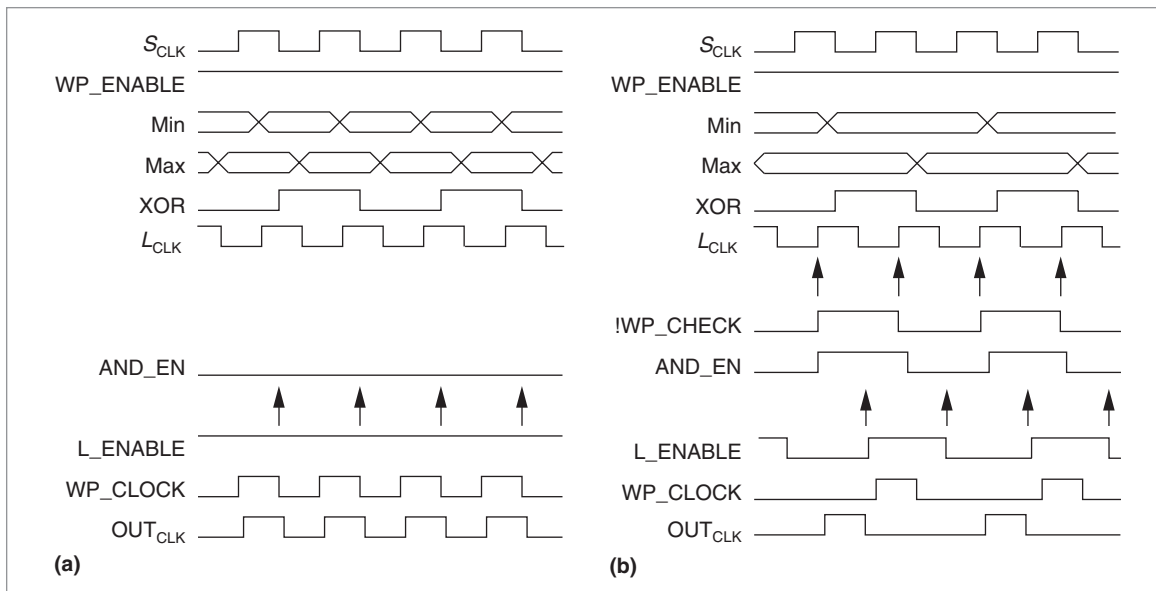
$WP\_CHECK$  indicates whether the design accommodates the original system clock. If the design is still workable with the system clock,  $WP\_CHECK$  remains low. If not, the signal starts toggling at every system clock cycle.

When  $WP\_CHECK$  toggles, the skew generator (which generates  $L_{CLK}$  and  $OUT_{CLK}$  at the bottom of Figure 4) comes into play. This circuit applies the constructive-skew concept to generate a skewed clock for the output registers so that they properly latch the data. The circuit uses  $WP\_CHECK$  to select between the skewed clock and the system clock.

In the figure, we used extra Min components to more simply show the skewed-

clock's generation. In reality, we can use existing Min components to generate clocks and reduce redundancy.

Once the output registers are properly skewed, the design should reset  $WP\_CHECK$  if the delay difference between maximum and minimum delay paths falls within the allocated budget of the clocking constraint. However, if delay changes due to process or environmental variations cause a larger-than-budgeted delay difference,  $WP\_CHECK$  toggles again. This signifies that the clock period is not large enough for the design to function correctly. In this situation, the circuitry slows the clock frequency by two cycles through a  $WP\_CHECK$ -controlled  $L\_ENABLE$  signal, which should allow for proper data latching.



**Figure 5. Timing waveform when wave-pipelined design is fast enough (a) or too slow (b).**

Figure 5a shows the timing of the self-checking scheme for the delay clock when the wave-pipelined design is fast enough. Figure 5b shows how the clock slows by two cycles when the design is too slow because of increased delay from process and environmental variations.

Figure 6 shows the HSpice results when the difference between the maximum and minimum delay paths has grown too large for the system clock to handle. As mentioned, the proposed circuitry recognizes that proper latching requires a slower clock frequency, resulting in WP\_CLOCK triggering every other system clock cycle.

**THE NUMBERS** resulting from our tests look promising, indicating that automation of the synthesis phase is feasible for wave-pipelined designs. The next step in design automation should be to automate the layout process, emphasizing place-and-route for delay balancing. ■

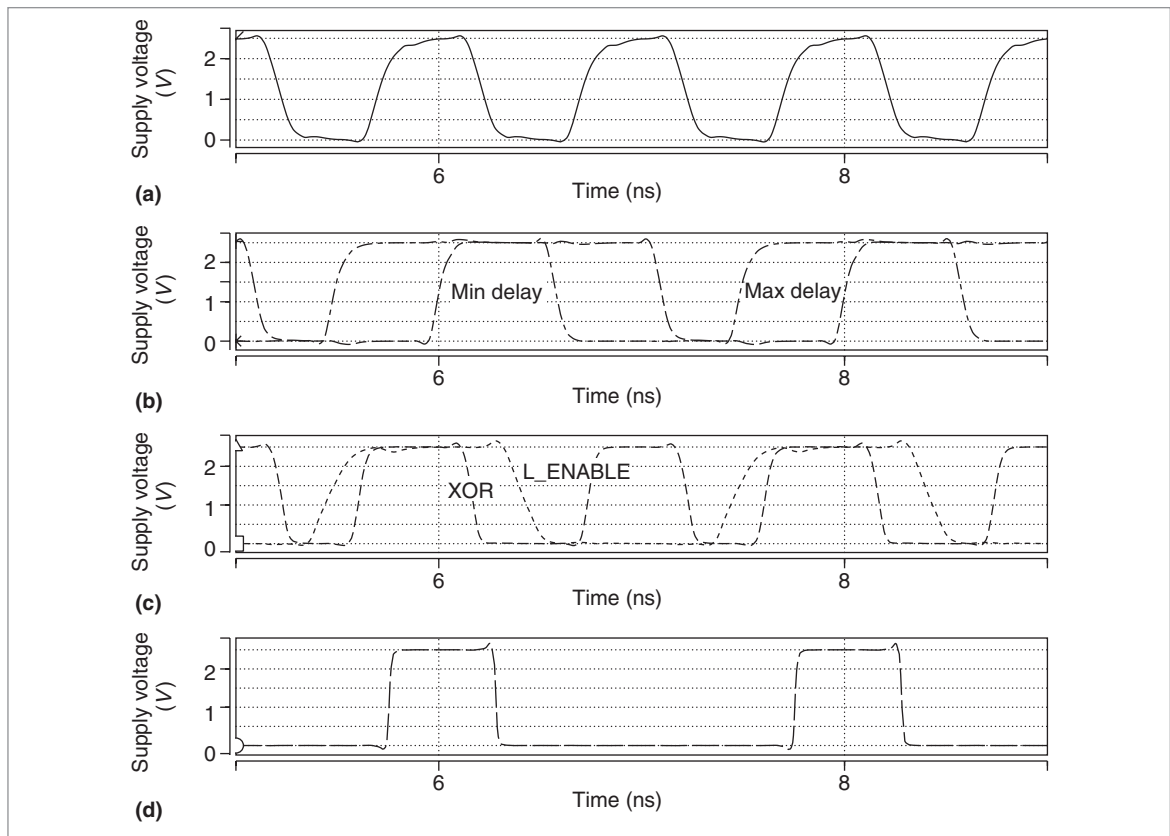
## References

1. D. Wong, G. De Micheli, and M. Flynn, "Designing High-Performance Digital Circuits Using Wave Pipelining: Algorithms and Practical Experiences," *IEEE Trans. Computer-Aided Design Integrated Circuits and Systems*, vol. 12, no. 1, Jan. 1993, pp. 25-46.
2. D.A. Joy and M.J. Ciesielski, "Clock Period Minimization with Wave Pipelining," *IEEE Trans. Computer-Aided Design Integrated Circuits and Systems*, vol. 12, no. 4, Apr. 1993, pp. 461-472.

3. F. Klass, *Wave Pipelining: Theoretical and Practical Issues in CMOS*, PhD dissertation, Dept. of Electrical Eng., Delft Univ., 1995.
4. R. Parthasarathy and R. Sridhar, "Double Pass-Transistor Logic for High Performance Wave Pipeline Circuits," *Proc. 11th Int'l Conf. VLSI Design*, IEEE CS Press, 1998, pp. 495-500.
5. D. Ghosh and S.K. Nandy, "NPCPL: Normal Process Complementary Pass Transistor Logic for Low Latency, High Throughput Application," *Proc. VLSI Design*, IEEE CS Press, 1993, pp. 341-346.
6. J.P. Fishburn, "Clock Skew Optimization," *IEEE Trans. Computers*, vol. 29, no. 5, May 1980, pp. 945-951.
7. W. Lam, R. Brayton, and A. Sangiovanni-Vincentelli, "Valid Clocking in Wave-Pipelined Circuits," *Proc. IEEE Int'l Conf. Computer-Aided Design (ICCAD 92)*, IEEE Press, 1992, pp. 518-525.
8. C.T. Gray, T. Hughes, and R. Cavin III, "Timing Constraints for Wave-Pipelined Systems," *IEEE Trans. Computer-Aided Design*, vol. 13, no. 8, Aug. 1994, pp. 987-1004.



**Woo Jin Kim** is a member of the technical staff at Sun Microsystems, where he is involved in UltraSparc CPU chip design. His research interests include high-speed design, chip integration, and computer-aided design flows. Woo has a BS and an MS in electrical engineering from



**Figure 6. Simulation result for large delay mismatch: system clock (a), maximum and minimum delay (b), XOR of Max and Min/L\_ENABLE (c), and WP\_CLOCK (d).**

Yonsei University, Seoul, South Korea, and a PhD in electrical and computer engineering from Northeastern University, Boston. He is a member of the IEEE.

and a PhD in computer engineering from Colorado State University.



**Yong-Bin Kim** is the Zraket Endowed Professor in the Department of Electrical and Computer Engineering at Northeastern University. His research interests include high-speed low-power VLSI circuit design and methodology. Kim has a BS in electrical engineering from Sogang University, Seoul, South Korea; an MS in computer engineering from the New Jersey Institute of Technology;

Direct questions and comments about this article to Yong-Bin Kim at the Electrical and Computer Eng. Dept., Northeastern Univ., 327 Dana Research Center, 360 Huntington Avenue, Boston, MA 02115; ybk@ece.neu.edu.

**For further information on this or any other computing topic, visit our Digital Library at <http://computer.org/publications/dlib>.**