Wave-pipelining: A tutorial and survey of recent research [1]

Wayne Burleson, Maciej Ciesielski
University of Massachusetts at Amherst,

Fabian Klass
SUN Microsystems / Stanford University,

Wentai Liu
North Carolina State University

Abstract:

Wave-pipelining is a method of high-performance circuit design which implements pipelining in logic without the use of intermediate latches or registers. This paper presents a tutorial of the principles of wave-pipelining and a survey of recent wave-pipelined VLSI chips and CAD tools for the synthesis and analysis of wave-pipelined circuits. Wave-pipelining has recently drawn considerable interest from both the academic and industrial communities. [2] The combination of high-performance IC technologies, pipelined architectures, and sophisticated CAD tools has converted wave-pipelining from a theoretical oddity into a realistic, although challenging, VLSI design method. In its crudest form, wave-pipelining is plagued with problems due to process variation, data-dependent delays, and two-sided clock period constraints. However, like clock skew in high-speed synchronous systems, these problems can be mitigated by innovative circuit structures, CAD tools and systematic design. The result is a higher performance circuit in exchange for an often acceptable increase in area.

1 Introduction

Wave-pipelining is an example of one of the many methods currently being used in sophisticated VLSI designs. Recent VLSI chips use pipelining extensively but also encounter

[1] This research was funded in part by the National Science Foundation under grant MIP-9208267 and MIP-9212346

[2] A recent forum on wave-pipelining at the 1994 International Symposium on Circuits and Systems in London drew significant interest from an academic and industrial audience. The authors were members of a panel which discussed recent issues in wave-pipelining, in particular its applicability in practical VLSI.

problems with distributing high frequency clocks. Wave-pipelining provides a method for significantly reducing clock loads and the associated area, power and latency while retaining the external functionality and timing of a synchronous circuit. It is of particular interest today because it involves design and analysis across a variety of levels (process, layout, circuit, logic, timing, architecture) which characterize VLSI design. However it also questions some of the fundamental tenets of simplified VLSI design as popularized in the early '80's.

The idea of wave-pipelining was originally introduced by Cotten [6], who named it *maximum rate pipelining.* Cotten observed that the rate at which logic can propagate through the circuit depends not on the longest path delay but on the difference between the longest and the shortest path delays. As a result, several computation "waves", i.e., logic signals related to different clock cycles, can propagate through the logic simultaneously. One can also view the wave-pipelining as a virtual pipelining, in which each gate serves as a virtual storage element.

In an attempt to understand the wave-pipelining phenomenon and turn it into a useful and reliable computer technology, recent research focused on the following aspects: 1) developing correct timing models and analyzing the problem mathematically [37, 9, 10, 18, 12, 13, 27], 2) developing logic synthesis techniques and CAD tools for wave-pipelined circuits [44, 46, 17, 21, 22, 39], 3) developing new circuit techniques specifically devoted to wave-pipelining [28], and 4) testing the wave-pipelining ideas by building VLSI chips [28, 45, 23]. A comparative study of existing methods in wave-pipelining can be found in [24].

Wave-pipelining has suffered from a number of myths and the problems which need to be solved in a wave-pipelined design are not widely understood. This paper presents a tutorial with examples to demonstrate the type of design problems which arise in wave-pipelining. In section 2 timing constraints are developed which ensure the proper operation of a wave-pipelined circuit. Section 3 discusses the sources of delay variation which affect the timing constraints and presents methods for minimizing their impact. Section 4 reviews CAD tools available for synthesizing wave-pipelined circuits and demonstrates their use with two example circuits. Section 5 reviews a fairly large number of industrial and academic designs which actually use wave-pipelining. Section 6 poses some open research problems in wave-pipelining and related areas.

The impact of the paper is broader than only wave-pipelining as the methods of analysis and synthesis apply to many other aggressive timing and circuit techniques being used in industry today. In the future, we anticipate that this type of VLSI design techniques will be required to maintain the steady increases in performance improvement to which we have become accustomed. VLSI designers, CAD developers and system designers need to be aware of these trends and their future impact on design, manufacture, testing and education in Microelectronics.

2 Clocking of Wave-pipelined Circuits

It can be said that, to a first order, wave pipelining achieves a speed-up of N, where N is the number of computation waves that propagate simultaneously through the logic. In general, the maximum allowed number of waves depends on the delay characteristics of the logic block, and specifically on the longest and shortest path delays through the logic. Additional constraints result from the insertion of clocked storage elements (registers or latches) at both ends of the combinational logic, and from the signal propagation characteristic (stable time) of the internal nodes. All of these constraints, subject to worst case process and environment variations, must be taken into account for the circuit to operate reliably. How these factors constraint the value of N is the subject of this section.

2.1 Definitions

The timing requirements of wave-pipelined circuits will be defined for a single combinational logic block with registers attached to its inputs and outputs. In the case of multi-stage pipelines, the timing constraints must hold for all stages (as well as possible loop constraints). In the sequel, the term *conventional pipelining* will be used to refer to a pipeline where only a single set of data propagates between registers at any given time. The term *constructive clock-skew* will be used to refer to a clock-skew that is intentionally created between two clock signals and that can be adjusted with predictable effects. This is in contrast to an *uncontrolled clock-skew* that exists in the circuit due to delay differences along the clock lines.

Parameters listed below will be used in the derivation of the timing constraints. The parameters are defined under worst-case conditions, including manufacturing tolerances, data-dependent delays, and environmental changes.

- D_{MAX}: Maximum propagation delay in the combinational logic block
- D_{MIN}: Minimum propagation delay in the combinational logic
- T_{CK}: Clock-period
- T_S: Register setup time
- T_H: Register hold time
- D_R: Propagation delay of a register
- Δ: Constructive clock skew between the output and input registers
- Δ_{CK}: Worst-case uncontrolled clock skew at a register.

Our analysis assumes that all the input and output registers have the same setup time T_S, hold time T_H, and propagation delay D_R. For simplicity and without much loss of generality, we shall only consider edge-triggered registers. Timing constraints for other types of clocking, including clocking with transparent latches, can also be derived.

2.2 Circuit and Timing Models

The phenomenon of wave-pipelining can be illustrated using the timing model presented in [13], shown in Fig. 1. A synchronous logic block is clocked by a set of input and output registers, with the constructive clock skew equal to Δ, Fig. 1(a). Associated with each block is a pair of minimum and maximum propagation delays, D_{MIN}, D_{MAX}. These parameters are the delays of the earliest and the latest signals propagating through the logic along its shortest and longest paths, respectively.

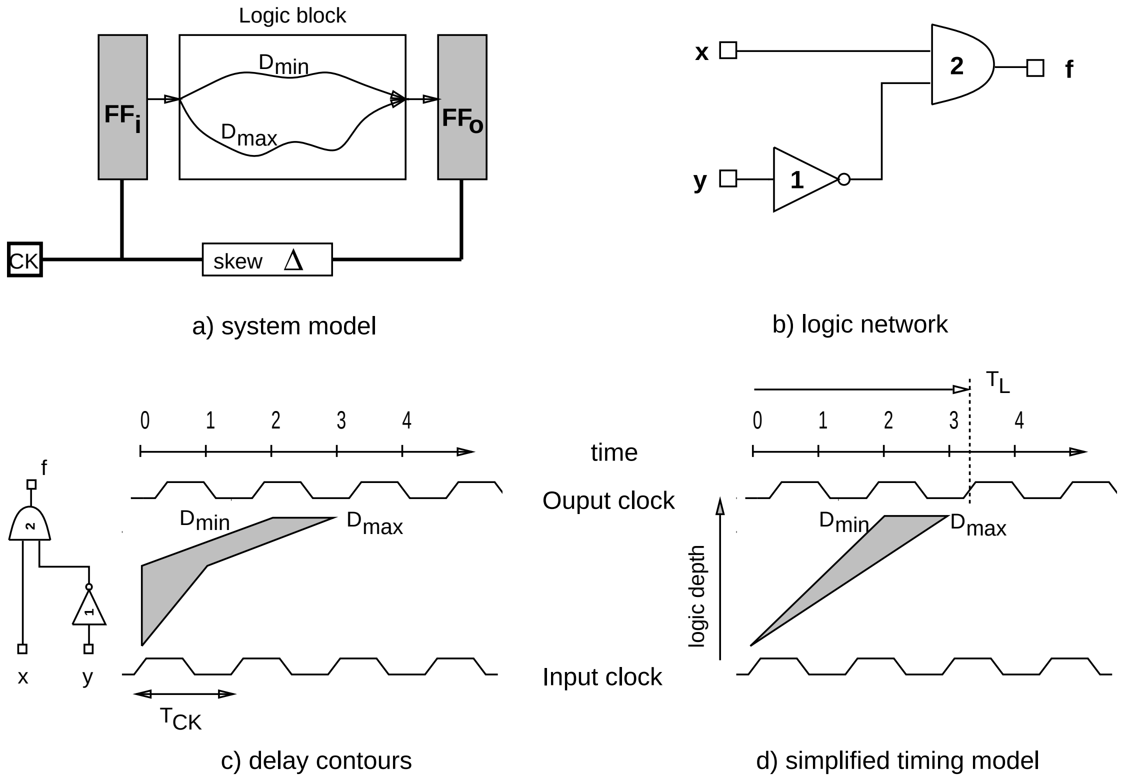


Figure 1: Model of wave-pipelined circuit

The spread of signals traveling along the longest and the shortest paths through the logic block can be visualized as *delay contours*. The shaded area between the contour lines represents an unstable region when signals switch, i.e., when the computation is being performed by the logic block. For a simple logic network in Fig. 1(b) the delay contours are shown in Fig. 1(c). The exact contours depend on the logic and the delay characteristics of its elements. Following [13], we shall simplify the delay contours by linearizing the delays along the logic block to obtain delay "cones", as shown in Fig. 1(d).

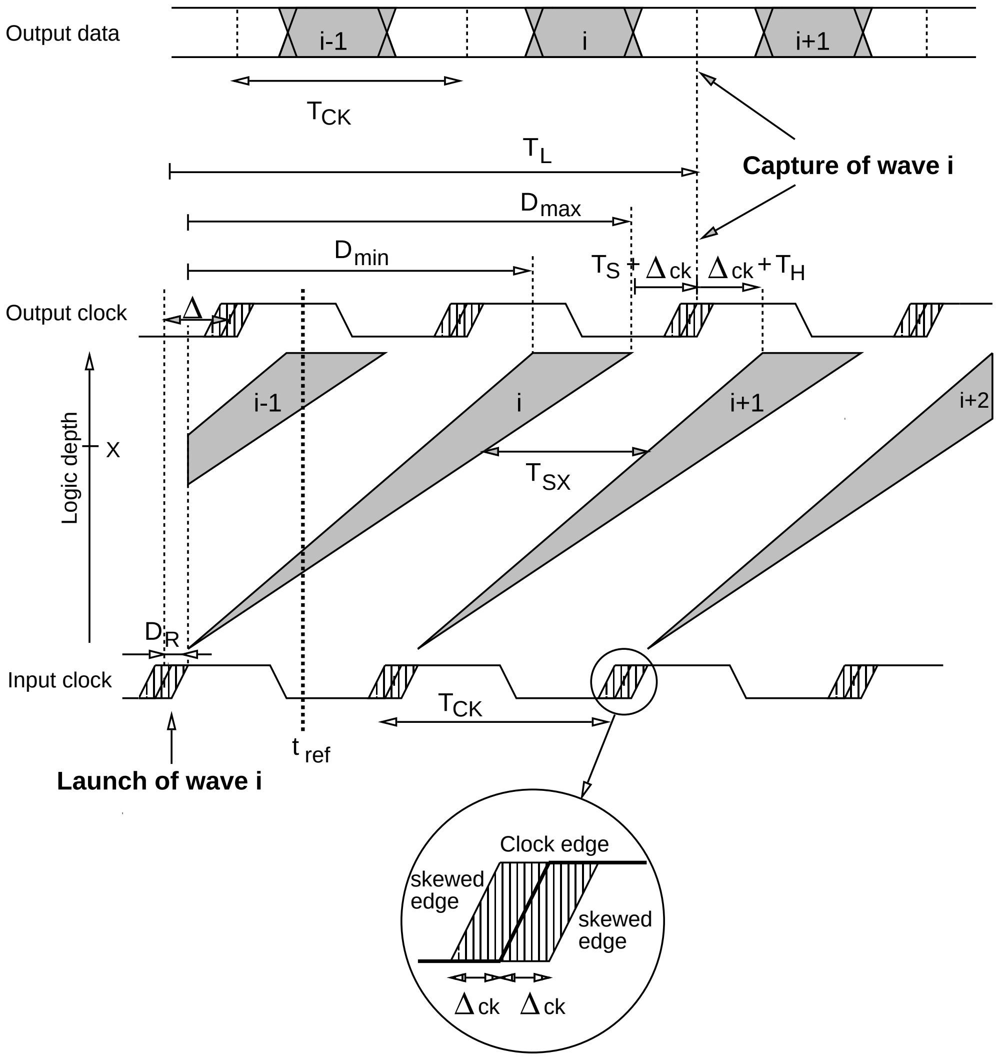


Figure 2: Temporal/spacial diagram for wave-pipelining system

Fig. 2 depicts a combined temporal and spacial diagram for a wave-pipelined circuit. The horizontal axis represents time, while the vertical axis represents the *logic depth* of the circuit. A point on the logic depth axis represents a node (gate) of the circuit.

The bottom part of Fig. 2 shows the clock waveforms at the input and output registers with the simplified delay contours of the computation regions. The upper part shows data waveforms at the outputs of the logic block. Again, the shaded rgions denote the time intervals when computation is being performed and data is unstable, while the space between two adjacent computation regions corresponds to a stable data. The computation regions are labeled with the clock cycle during which the data has been latched at the input register. It can be observed that at any given reference time, t_{ref}, several "waves" of computation can present in the logic simultaneously. In Fig. 2 two waves can be observed at time t_{ref}: wave $(i-1)$ is about to reach the output, while wave i has just started at the input.

For correct clocking, the output data must be sampled at time T_L at which data is stable, i.e., T_L must fall in the nonshaded area. Furthermore, at any internal node of the network, there must be a minimum separation between the arrival time of the latest signal of a given wave, and the earliest signal of the next wave. The temporal separation between the waves at an internal node x is represented in the figure by T_{SX}.

2.3 Timing Constraints

For a wave-pipelined system to operate correctly, the system clocking must be such that the output data is clocked/latched *after* the latest data has arrived at the outputs and *before* the earliest data from the *next* clock cycle arrives at the outputs. We shall first derive the conditions for clocking of the latest and the earliest data propagating in the circuit, referred to as the *register constraints.* We introduce parameter N, which represents the number of clock cycles needed for a signal to propagate through the logic block before being latched by the output register. This parameter serves as an intuitive measure of the *degree of wave-pipelining.* The data should be clocked at time T_L by the rising edge of the output register N clock cycles after it has been clocked by the input register. Due to possible constructive skew Δ between the output and the input registers, this time can be expressed as

$$T_L = N\ T_{CK} + \Delta \tag{1}$$

where $\Delta >= 0$. (In general, Δ can have an arbitrary value; in this case T_L can also be expressed as a sum of an integer number of clock cycles and some delay smaller than T_{CK}).

2.3.1 Register Constraints

1. **Clocking of the latest data**.

   This constraint requires that the latest possible signal arrives early enough to be clocked by the output register during *N-th* clock cycle. Therefore, the lower bound on T_L, which denotes the time at which the output wave is captured, is given by

$$T_L > D_R + D_{MAX} + T_S + \Delta_{CK} \tag{2}$$

2. **Clocking of the earliest data**.

   This condition requires that the arrival of the *next* wave must not interfere with the clocking of the current wave. That is, the earliest possible signal of wave $i+1$ must arrive later than the clocking of the i-th wave at the output register. This condition is similar to the *race-through* constraint in conventional pipelining. Notice that the earliest arrival of wave $i+1$ is given by $T_{CK} + D_R + D_{MIN}$. After the clock pulse has been applied to the output register, additional hold time T_H must be allowed for the data to remain steady. In addition, one must account for an uncontrolled clock-skew Δ_{CK} at the output register. As a result, the T_L is bounded above as follows:

$$T_L < T_{CK} + D_R + D_{MIN} - (\Delta_{CK} + T_H) \tag{3}$$

Combining constraints (2) and (3) gives us the well known condition of Cotten:

$$T_{CK} > (D_{MAX} - D_{MIN}) + T_S + T_H + 2\,\Delta_{CK} \tag{4}$$

It can be concluded that the minimum clock period is limited by the maximum difference in path delay $(D_{MAX} - D_{MIN})$, plus the *clocking overhead* $(T_S + T_H + 2\,\Delta_{CK})$, resulting from the insertion of clocked registers.

2.3.2 Internal Node Constraints

Constraint (4) guarantees that waves do not collide, or overlap, at the output of the logic block. Additional constraints need to be imposed at the internal nodes of the combinational block to prevent wave collision at the individual logic gates. These signal separation constraints can be informally expressed as follows: *The next earliest possible wave should not arrive at a node until the latest possible wave has propagated through.* The signal separation constraints, also called *internal node constraints*, were derived, in slightly different forms, by Ekroot [9], Wong et al. [46], Joy et al. [18], and Gray et al. [13].

Let x be an internal node (output of a gate) of the logic network, a point on the logic depth axis in Fig. 2. To help derive the internal node constraint we define $d_{MAX}(x)$ and $d_{MIN}(x)$ as the longest and the shortest propagation delays from the primary inputs to node x. Then, from Fig. 2, we have the following inequality:

$$T_{CK} + d_{MIN}(x) - d_{MAX}(x) - T_{SX} - \Delta_{CK} \geq 0 \tag{5}$$

where T_{SX} is the minimum time that node x must be stable to correctly propagate a signal through the gate, and Δ_{CK} is the worst case uncontrolled clock skew at the input register. This gives the following internal node constraint that must be satisfied at each node x of the circuit:

$$T_{CK} \geq d_{MAX}(x) - d_{MIN}(x) + T_{SX} + \Delta_{CK} \tag{6}$$

Notice the similarity between constraint (6) and (4). While $(d_{MAX}(x) - d_{MIN}(x))$ is equivalent to $D_{MAX} - D_{MIN}$, the term T_{SX} is equivalent to $T_S + T_H$ (the minimum sampling time for the data); the factor of 2 in the clock-skew term is not required since the signals are not stored (clocked) at the internal nodes.

2.4 Intervals of Valid Clocking

Based on the conditions presented in the previous section, a linear program (LP) can be readily formulated to find a minimum value of clock period for a given value of N (number of waves). The LP approach minimizes T_{CK} subject to both register constraints (2, 3) and the internal node constraints (6) for all nodes in the circuit, if needed. This approach, taken by Fishburn [10] and Joy et al. [18], seems to suggest that the circuit will operate correctly at any value of clock cycle above the computed minimum, that is, that the region of valid clock period is only bounded below. It has been shown, however, that this is not necessarily the case. In particular, it has been independently demonstrated by Lam et. al, [27] and by Gray at al.[13] that the feasibility region of the valid clock period T_{CK} is not continuous but is composed of a finite set of disjoint regions. Furthermore, it has also been shown that the degree of wave-pipelining, N, does not change monotonically with the increase in clock frequency.

To see why this is the case we shall examine again register constraints (2) and (3) and derive a new constraint by expressing the output latching time, T_L, as a function of T_{CK}, see eq. (1). To simplify subsequent analyses, we will assume that these constraint prevail over the internal node constraints, which seems to be the case for most practical circuits. (If this assumption is not satisfied, the internal node constraints must be included.) With this assumption, the constraints (2) and (3) can be combined to form a two-sided constraint on the clock period. Recalling that T_L, the latching time of the data at the output register can be expresses as $T_L = N\ T_{CK} + \Delta$, we obtain the following inequality (refer to Fig. 2):

$$D_R + D_{MAX} + T_S + \Delta_{CK} < N\ T_{CK} + \Delta < T_{CK} + D_R + D_{MIN} - (\Delta_{CK} + T_H) \qquad (7)$$

To simplify the interpretation of the above relation let us introduce two new parameters, T_{MAX} and T_{MIN}.

$$T_{MAX} = D_R + D_{MAX} + T_S + \Delta_{CK} - \Delta \qquad (8)$$

will represent the maximum delay through the logic, including clocking overhead and clock skews, while

$$T_{MIN} = D_R + D_{MIN} - \Delta_{CK} - T_H - \Delta \qquad (9)$$

will represent minimum delay through the logic, including clocking overhead and skews. With this, eq. (7) can be expressed as follows:

$$\frac{T_{MAX}}{N} < T_{CK} < \frac{T_{MIN}}{N-1} \qquad (10)$$

Analyzing the above result reveals that the feasibility region of clock period T_{CK} may not be continuous. For $N = 0$ (for which this equation was not really derived) there is no pipelining or useful clocking to talk about; the data is latched by the input and output registers during the same clock period, so the circuit is combinational (equivalent to having $T_{CK} = \infty$). For $N = 1$ the equation confirms the classical result that the clock period is only bounded below by T_{MAX}, as in the conventional pipelining. For larger values of N, however, it shows that the period is also bounded above by $\frac{T_{MIN}}{N-1}$. In addition, if $T_{MAX} > T_{MIN}$, then any two successive intervals $[\frac{T_{MAX}}{N+1}, \frac{T_{MIN}}{N}]$ and $[\frac{T_{MAX}}{N}, \frac{T_{MIN}}{N-1}]$ are disjoint. For a given value of N the size of the interval which represents valid clock period grows with the increase of T_{MIN} and the decrease of T_{MAX}.

2.5 Timed Boolean Function Approach

Notice that clock interval defined by eq. (10) itself may not correspond to a valid clocking if the internal node constraints are not subsumed by it. An alternative approach to clocking analysis of wave-pipelined systems was proposed by Lam et al. [27], whereby the internal node constraints are taken into account implicitly and are represented in the same fashion as the register constraints.

This approach is based on a *Timed Boolean Function* (TBF) representation of the circuit. A Timed Boolean Function is a Boolean function of Boolean variables with an added time argument. We shall employ the following notation to represent the dependence of TB variables and TB functions on time. Parenthesis (), as in $x(t)$ or $f(t)$, will refer to a *continuous*, or real time. Square brackets [], as in $x[n]$, will denote a *discrete* time, i.e, time measured in terms of clock cycles. All Boolean manipulation methods that apply to Boolean functions also apply to TBF's, provided that the variables on which a TBF depends all have the same time argument. For example, $x[t] + \overline{x[t]} = 1$, or $x[t]\ \overline{x[t]} = 0$, etc., but $x[t-1] + \overline{x[t]} \neq 1$, etc.

Consider a TBF for output y_j of the circuit in terms of its input variables $x(t)$

$$y_j(t) = f(..., x_i(t - d_{ij}), ...) \tag{11}$$

where d_{ij} is a logic delay from input x_i to output y_j. The value of this function at time $t = kT_{CK}$, normalized w.r.t., and sampled with the clock period T_{CK} is

$$y_j(kT_{CK}) = f(..., x_i(k - \lceil \frac{d_{ij}}{T_{CK}} \rceil), ...) \tag{12}$$

which can be equivalently expressed as

$$y_j(kT_{CK}) = f(..., x_i[k - N_{ij}], ...) \tag{13}$$

where $N_{ij} = \lceil \frac{d_{ij}}{T_{CK}} \rceil$ is the number of clock cycles needed to propagate a signal from input i to ouput j (and hence the number of waves in the circuit). Now the condition for valid computation that satisfies internal node constraints can be expressed as follows [27]: if and only if all variables x_i in the support of y_j, for all j, have the same time argument $[k - N]$, i.e., if $f(kT_{CK}) = f[k - N]$. In this case the latency of the circuit, measured in the number of clock cycles is the same as the degree of wave-pipelining, N. Combining the above constraint with equation 10, results in the following theorem for valid clock period [27]: *The clock period T_{CK} in a circuit is valid if it satisfies equation 10 for some positive integer N, and the computed function at $t = kT_{CK}$ is $f[k-N]$.*

In summary, the TBF approach provides a convenient way to express the condition for safe signal separation both at the internal nodes of the circuit and at the output register. The internal node constraints are simply accounted for by adding the requirement that all variables in the support of the logic function have the same discrete argument $[k-N]$. Lam et. al. [27] describe a procedure to compute all valid clock intervals for a circuit with given delay parameters and tolerances. The reader is referred to their paper for details.

3 Sources of Delay Variations

As discussed in Section 2, other speed-limiting factors in wave-pipelining, in addition to the path-delay difference, are the uncontrolled clock-skew, the sampling time of registers (i.e., setup time plus hold time), and the worst-case transition time (rise or fall) at the logic outputs. In order to achieve the highest clock-rate, all these factors must be minimized. While this minimization process has been a major challenge in the design of conventional high-speed pipelined systems as well, the *equalization of path-delays* comes as a new challenge for the design of wave-pipelined systems. In general, path-delay equalization is achieved in two ways. One way consists of increasing short paths by inserting delay elements (i.e., non-inverting buffers). This technique has been called *rough tuning* since the adjustment resolution is determined by the delay elements. The second way consists of increasing short paths by adjusting gate-delays. Since this method allows for fine delay adjustments, it has been called *fine tuning.* In general, the two techniques are combined for optimality [46]. While in theory the path-delay equalization problem has been solved, in practice the real challenge is to accomplish it in the presence of a variety of static and dynamic delay tolerances, some of which are listed below.

1. *Gate-delay data-dependence:* Gate-delay independence on the input pattern, i.e., constant delay, is not satisfied in general. It depends on the particular technology and the structure of logic gates. Some input patterns may cause significant delay variations.

2. *Coupling capacitance effects:* The effective capacitance seen by a gate depends on the capacitive coupling between adjacent wires. This may introduce significant changes in the gate-delay, especially in advanced multi-level-metal processes.

3. *Power-supply induced noise:* Power supply noise is attributed mainly to IR drops, capacitive coupling between the interconnection wires and the power supply lines, and the inductance of bonding wires, package trace, and pins. Noise in the power-supply voltage can cause accumulative delay dispersions as waves propagate through several logic layers.

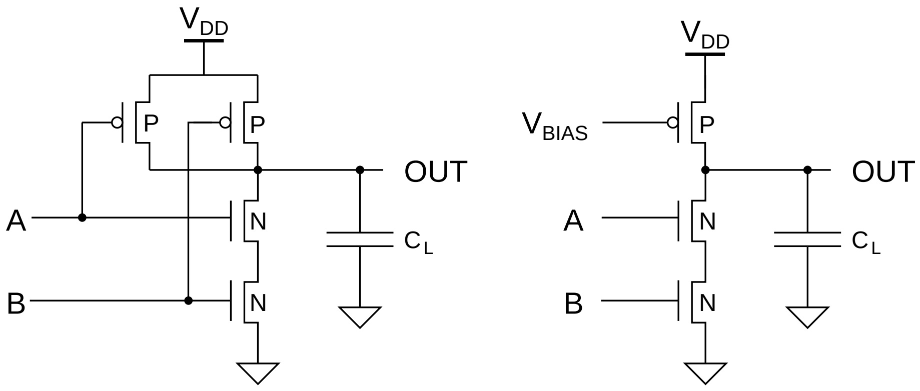
4. *Process parameter variations:* Variations of process parameters during manufacture may result in substantial gate-delay variations. Equivalent circuits from different

fabrication runs may have different propagation delays. This delay dispersion must be accounted for to establish the minimum separation between waves.

5. *Temperature-induced delay changes:* Some process parameters, i.e., carrier surface mobility of MOS transistors, are highly thermally sensitive. Changes in the operational temperature result in changes in the gate-delay.

3.1 Data Dependencies

One of the major obstacles initially found in static CMOS is the strong dependence of the gate-delay on the input data. For example, consider the static CMOS 2-input NAND gate shown in Fig. 3(a). Depending on whether one or two of the parallel PMOS devices are switching on, the delay of the gate varies by a factor of about two. For gates with several inputs, this factor is even larger. Clearly, this feature is undesired for wave-pipelining: logic gates with constant delay is a requirement for the equalization of path-delays. To overcome this problem, the use of biased-CMOS gates has been proposed, also known as pseudo-NMOS gates, where parallel devices are replaced by a single device whose gate is connected to a bias voltage [29]. Fig. 3(b) shows a biased-CMOS 2-input NAND gate. While this approach reduces the delay dependence problem and makes CMOS better suited for wave-pipelining, it is achieved at the costly expense of increased power dissipation: biased-CMOS gates dissipate static power in one of the two logic states of the output.



(a) Static CMOS NAND (b) Biased CMOS NAND

Figure 3: CMOS Gates

3.2 Process and Environmental Delay Variations

Changes in temperature, supply levels, and process parameters can have a substantial effect on the delay of a CMOS circuit. How such changes affect the operation of wave-pipelining is discussed below. Although the following analysis is for temperature, it applies as well to process and power supply changes.

As shown in Section 2.4, the clock-period T_{CK} and the number of waves N in a wave-pipelined circuit are related by equation (10). This inequality can be rearranged as

$$T_{MAX} < N\, T_{CK} < T_{MIN} + T_{CK} \tag{14}$$

where the values of T_{MAX} and T_{MIN} are specified at nominal temperature.

We assume that the temperature across the die area is uniform and that every path-delay in the circuit is affected in the same proportion by changes in temperature. If, for a temperature above the nominal, T_{MAX} and T_{MIN} are increased by a factor $\beta_S > 1$, constraint (14) becomes:

$$\beta_S\, T_{MAX} < N\, T_{CK} < \beta_S\, T_{MIN} + T_{CK} \tag{15}$$

In the same manner, if, for a temperature below the nominal, T_{MAX} and T_{MIN} are reduced by a factor $\beta_F < 1$, constraint (14) is given by:

$$\beta_F\, T_{MAX} < N\, T_{CK} < \beta_F\, T_{MIN} + T_{CK} \tag{16}$$

In order to find a valid clock-period that satisfies constraints (14), (15), and (16), it is required that:

$$\beta_S\, T_{MAX} < N \times T_{CK} < \beta_F\, T_{MIN} + T_{CK} \tag{17}$$

which transforms to:

$$T_{CK} > \beta_S\, T_{MAX} - \beta_F\, T_{MIN} \tag{18}$$

The interpretation of Eq. (18) is as follows. The clock period is limited by the difference between T_{MAX} at maximum temperature ($\beta_S T_{MAX}$) and T_{MIN} at minimum temperature ($\beta_F T_{MIN}$). This is the minimum clock period that assures correct operation in the entire range between the two temperatures (Notice that the use of a fixed zero constructive clock skew still results in invalid intervals for T_{CK}).

The conclusion from this analysis is that in addition to the limit imposed by the maximum difference $T_{MAX} - T_{MIN}$, process- and environment-induced delay changes impose tighter constraints on the minimum clock-period for wave-pipelining, or conversely, on the maximum number of waves. Notice that even in the ideal case of $T_{MAX} = T_{MIN}$ (i.e., under the perfect delay equalization and zero clocking overhead assumptions), the clock-period is still limited by $T_{CK} > (\beta_S - \beta_F)\, T_{MAX}$.

While in conventional pipelining the minimum clock-period can be specified based upon the worst-case delay accounting for temperature, voltage, and process, in wave-pipelining the best-case delay must also be considered. The consequence is a more substantial performance degradation for wave-pipelining. Although for nominal conditions, a larger number of waves could be achieved, for typical operational conditions the maximum number of waves might be rather limited, regardless of the logic depth of pipeline. Means to overcome this limitation are worthy of further study.

4 CAD Tools for Wave-pipelining

Equations 4 and 10 derived in Section 2 suggest that the minimum value of clock period as well as the range of valid clock interval can be adjusted by modifying certain parameters of the system. The following parameters can be controlled by the designer: maximum and minimum delays, D_{MAX} and D_{MIN} of the combinational logic block, and the constructive clock skew, Δ, between the output and input registers. Two techniques are generally adopted to control these parameters. One, referred to as a *logic balancing*, attempts to minimize the difference $D_{MAX} - D_{MIN}$; it involves logic restructuring, delay buffer insertion along selected logic paths, and selected device sizing. The other technique, which which can be termed as *clock buffer insertion*, aims at adjusting the value of constructive clock skew Δ by inserting precisely controlled delays along the clock paths. Furthermore, for wave-pipelined systems with multiple pipeline stages and multiphase clocking, clock period can be minimized by adjusting the duration of clock phases controlling individual pipeline stages.

This section briefly reviews some of the most popular techniques and practical CAD tools for the analysis, optimization, and synthesis of wave-pipelined systems.

4.1 Timing analysis and optimization

In addition to the theoretical work on modeling and analysis of wave-pipelining presented in Section 2, a number of timing analysis and verification tools for synchronous pipelined and wave-pipelined systems have been developed. A notable example of such a tool is a pipeline scheduler, called $pipeT_C$, developed by Sakallah et al [37]. Their work provides a unified formalism for describing the timing in pipelines, accounts for multiphase synchronous clocking, handles both short and long path delays, and accommodates (to a certain extent) the effect of wave-pipelining. The tool generates correct minimum cycle-time clock schedules and signal waveforms from a multiphase pipeline specification. However, wave-pipelining is considered here only implicitly. The designer is required to provide the parameter N, the degree of wave-pipelining. There is no analysis of upper bound on the clock cycle as a function of intrinsic parameters of a pipeline stage. Finally, the results do not take clock skew into account.

A number of timing optimization tools were developed to minimize clock period by adjusting constructive clock skews. These techniques are based on the LP approach proposed by Fishburn [10] and Joy et al. [17], mentioned in Section 2.4. Joy et al. [17] developed an optimization technique and a layout synthesis tool for static CMOS in a standard cell layout environment. This technique uses timing constraints and the optimum clock skew value to guide the placement of cells and clock distribution trees, including the insertion of delay buffers along clock lines. At the same time it attempts to select proper routing layers to balance path delays to maximize the degree of wave-pipelining. It must be recognized, however, that the controllability of parameters such as the sheet resistance of VLSI conductors is very low, their adjustment imprecise, and the uncertainty of the final timing too high to be of practical value. Other problems include extreme sensitivity to wire delay modeling, and reliable implementation of constructive skew in the form of buffers with precisely controlled delays.

4.2 Synthesis tools

Most of the work in synthesis explores the idea of minimizing the difference in logic path delays by means of logic balancing. The theory presented in section 2, and in particular equations 4 and 10, clearly demonstrate that the valid clock period is determined by a single interval, $[\frac{D_{MAX}}{N}, \infty]$, only when $D_{MAX} = D_{MIN}$, i.e., when all paths in the circuit are perfectly balanced (Assuming no constructive clock skew). In this case there are no "forbidden" frequencies in the operation of the circuit and the circuit computes function $f[k-N]$, where N is the desired depth, or the degree of wave-pipelining.

The techniques that attempt to equalize logic paths by minimizing $(D_{MIN} - D_{MAX})$ are based on one or more of the following approaches: logic restructuring, insertion of delay buffers or latches, device (buffer and gate) sizing, and controlled placement and routing of both the circuit components and clock distribution lines/trees.

The most popular techniques for logic balancing are based on logic restructuring and delay buffer insertion along logic paths. Wong et al. [44, 46] developed a method and a CAD tool to minimize the variance of path delays in ECL logic circuits. Their method is based on inserting the delay buffers followed by device fine-tuning. The delay of each gate is tuned by controlling the tail current, a feature unique to ECL circuits. This method was tested on circuits with regular structures, such as adders and multipliers. A factor of 2.5 increase in throughput (number of waves) has been achieved at a cost of 10-50% increase in area [45]. Shenoy et al. [39] expressed the logic balancing problem as an optimization procedure with additional short path constraints. This technique was implemented as an experimental CAD tool that interfaces with the SIS system [40].

Both of these approaches aim at obtaining the maximum rate pipelining while using a minimum number of delay buffers. However, for multiple-fanout gates the delay buffers are inserted individually for each fanout. As a result a separate step is typically required to merge the common buffers for area recovery. For the balancing method developed by [44] this is a simple task because the delay of ECL gates does not depend on the fanout. In the case of CMOS circuits, the fanout load significantly affects the delay and its impact on timing cannot be ignored. For this reason, Kim et al. [22] developed a delay buffer insertion method using chains of buffers, thus eliminating the need for merging common buffers. The problem of delay buffer insertion on a multiple fanout net is simplified by assuming that only one size of delay buffer is to be used. This method permits a quick estimation of the number of buffers needed to balance a circuit and can therefore be used in heuristics for logic restructuring. The algorithm is implemented in the SIS logic synthesis environment [40]. The remainder of this section briefly describes the logic balancing method for CMOS circuits according to this scheme. The reader is referred to [21, 22] for details.

4.2.1 Logic balancing

The goal of logic balancing is to equalize signal arrival times at the inputs of each gate of the circuit. It is accomplished by restructuring the logic and balancing the logic expression tree representing a given function. Since an ideal balancing is rarely possible, logic restructuring aims at minimizing the maximum difference between the arrival times at the gate inputs. To facilitate delay estimation during logic restructuring the circuit is first decomposed into two-input gates and inverters. It is then followed by *node collapsing*, and recursive *decomposition*.

Node collapsing, or resubstitution, is a standard algebraic transformation technique which combines several nodes of a logic network into a single node. The collapsing facilitates the subsequent transformations, such as recursive decomposition, which will finally transform the function into a network with a desired timing property. In the case of logic balancing, collapsing enables decomposition of the network leading to the generation of a more balanced circuit. This is illustrated in Fig. 4(a) and (b) where a node with delay 3 is collapsed into its fanin nodes, creating a single node with 4 inputs. The subsequent decomposition transformation decomposes this node in such a way as to create a more balanced structure.

The *decomposition* of the collapsed nodes is accomplished using the *kernel division* technique employed in MIS [8]. The logic expression of node n slated for kernel-based decomposition can be represented as $n = p \cdot q + r$; divisor p is a multiple-cube expression called the *kernel*, quotient q is a single cube (*co-kernel*), and r is the *remainder*.

The goal is to find a decomposition that minimizes the difference between latest arrival times at the inputs to the collapsed node. This is accomplished by computing all kernels of the expression n and selecting the one which leads to the most balanced structure. Using each kernel as a divisor, its respective quotient q and remainder r are then calculated. By estimating the delay of the kernel, co-kernel, and remainder, the kernel which gives the best balancing of the expression is selected. The kernel-based decomposition is applied recursively to nodes p, q, and r until no further improvement is possible. This step is

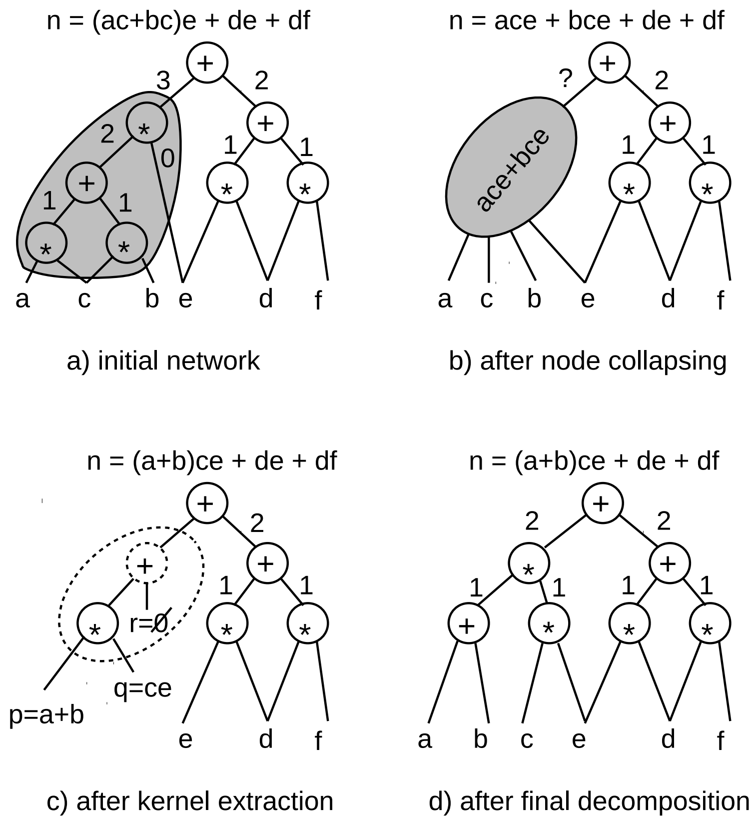


Figure 4: Node collapsing and decomposition

then followed by a decomposition of the remaining collapsed nodes into two-input gates to achieve further balancing and facilitate a reliable delay computation. Fig. 4(c) and (d) shows the result of such a decomposition. The details of the procedure can be found in [21].

4.2.2 Delay buffer insertion

Once the circuit is restructured, additional balancing can be achieved by means of delay buffer insertion. Delay buffers are inserted in appropriate places so as to further minimize the difference in signal arrival times at the gate inputs.

The underlying requirement imposed on buffer insertion is that it should not affect the latency of the circuit. For this reason the buffers are inserted only along fast paths, without affecting the slow ones. The amount of delay to be inserted at an input to a gate is chosen so that the arrival time at that input matches the arrival time of the slowest input, but does not exceed it. As a result, the latest arrival time at the output of the gate never increases as a result of buffer insertion. This approach allows one to localize the buffer insertion problem to that of inserting a single chain of buffers at the output of each gate. The desired delays at the fanouts of the gate are then obtained by providing taps at different stages of the buffer chain. Similar methods for multiple sized buffers have also been developed.

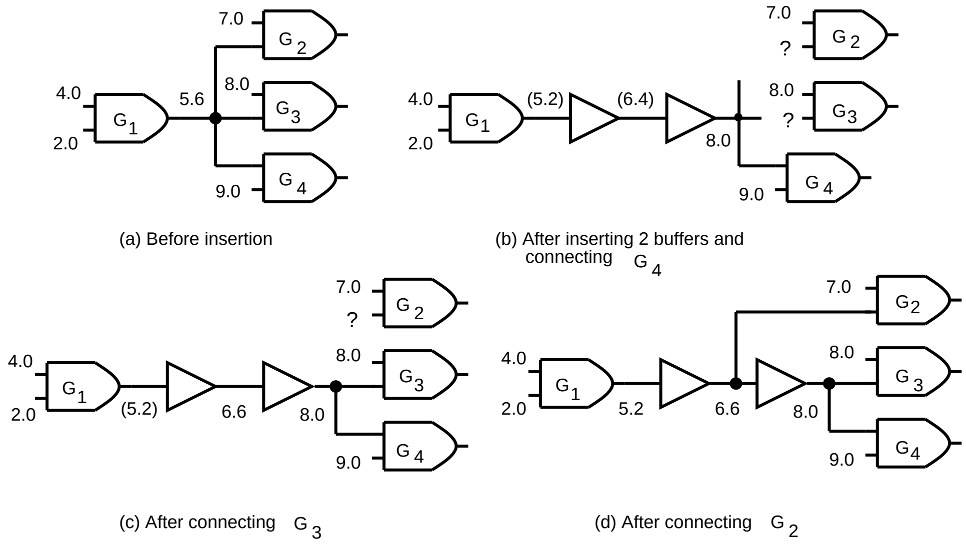


Figure 5: Example of delay buffer insertion

Fig. 5 illustrates the process of buffer chain construction and fanout distribution for the three gates connected to G_1. For simplicity it is assumed that each buffer contributes 1 unit delay, and the load at each fanout (input to a gate or a buffer) contributes 0.2 unit delay. The numbers given at the inputs to the gates represent the latest signal arrival times. A chain of 2 buffers is created to achieve the terminal delay at G_4 equal to 8.0. The input to G_3 is then tapped off the end of the chain to match the arrival time (8.0) of its other input, while the input to G_2 is tapped of the first stage of the chain, which provides the delay of 6.6. Notice that the arrival time at the end of the chain (8.0) is independent of how the inputs to G_2 and G_3 are connected to the chain. The details of the procedure, including the proper ordering of the gates for buffer chain construction, can be found in [22].

5 Synthesis Example

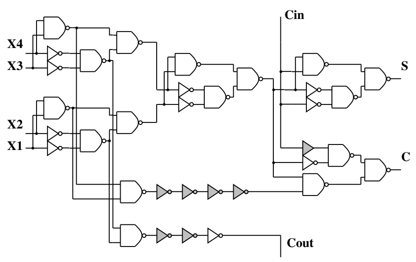
This section illustrates the use of CAD tools for wave-pipelining with two examples. The first example is a simple (4,2) compressor circuit (counter), whose synthesized design is compared with a manual design. The second example, with a much more complex design, shows some of the limitations of current algorithms for path balancing at the logic and circuit level.

5.1 (4,2) Compressor

Fig. 6(a) shows a manual design of a (4,2) compressor circuit, used as a basic multiplier cell in [25]. Fig.6(b) is a synthesized version of the same function, obtained with logic balancing tools described in Section 4.2.

A fair comparison of the two circuits is difficult to make since they were designed with different goals, they used different processing technology and performance was measured differently. Circuit (a) is part of a 16×16 multiplier; it was designed, balanced and tunned manually in order to obtain the fastest running circuit; it was implemented for a commercial (HP) $1\mu m$ CMOS technology; its path delays were measured by exhaustive HSPICE simulation, taking into consideration input transition delays, transistor input and wiring capacitances extracted from layout, and the process variations. The maximum and minimum reported delays were 1.46 and 1.19 ns, respectively, resulting in a maximum delay variation of less than 20%. Circuit (b) was automatically synthesized to allow for 3 waves; it was targeted for a generic 2μ MOSIS cell library; and its delays were computed using a simple unit fanout delay model. The circuit has delay of $2.18ns$ and a latency of $6.6ns$ (3 waves). Notice its balanced logic structure and the inserted buffers.

A simple-minded comparison of circuit areas, based on gate count, and of latencies, based on unit fanout delay, shows that synthesis tools can be used effectively to produce designs comparable with the manually tuned circuits.



(a) Manually balanced design

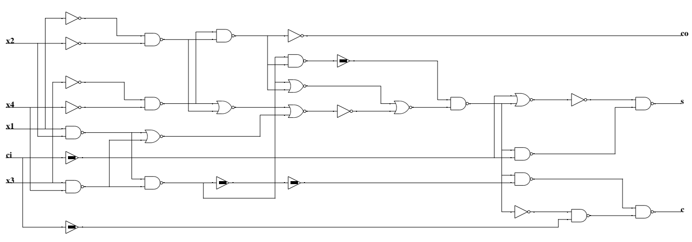
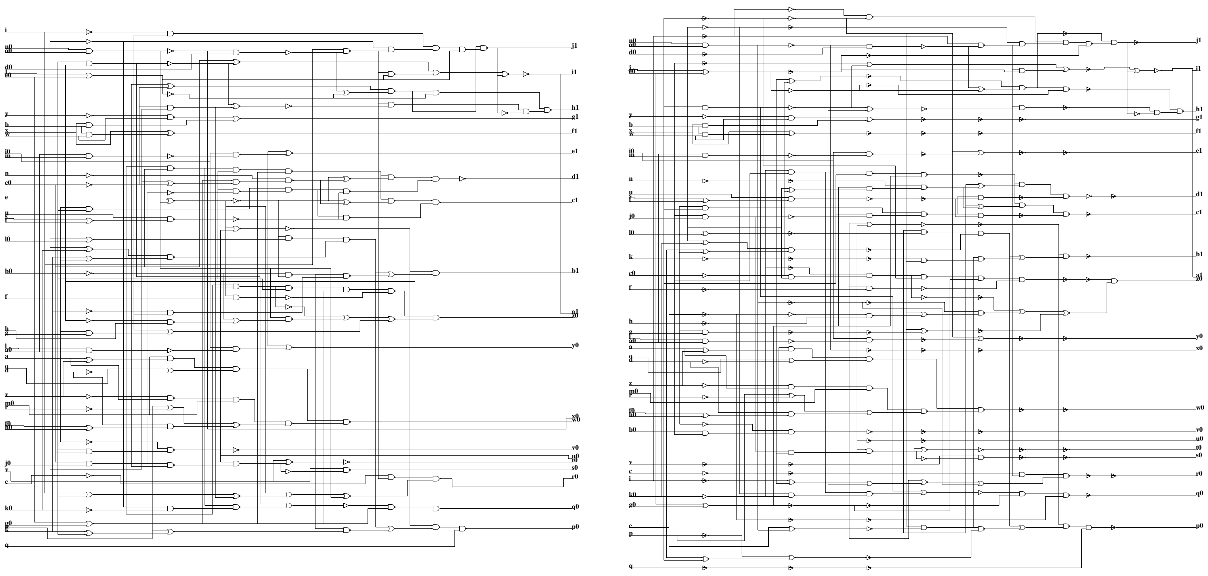

(b) Synthesized design

Figure 6: (4,2) compressor circuit

5.2 Random logic example

To validate the logic synthesis approach described in the previous sections a number of combinational circuits were synthesized and tested. These included both regular arithmetic circuits and a set of random logic circuits from the MCNC benchmark set. The random logic circuits are characterized by a very irregular structure, and as such are not well suited for logic balancing. The below example demonstrates that even for those circuits logic restructuring followed by buffer insertion can result in a significant improvement in circuit performance by means of wave-pipelining.

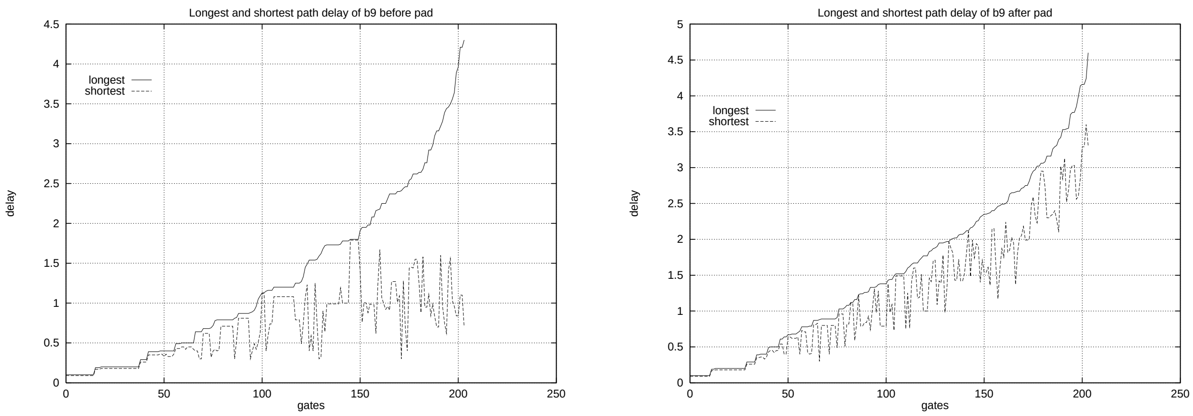
Figure 7 shows a schematic of circuit *b9* from the MCNC benchmark set after (a) restructuring and (b) buffer insertion (padding).



(a) *b9* after restructuring (b) *b9* after restructuring and padding

Figure 7: Schematic of circuit *b9* after restructuring and padding

Fig.8 shows the longest and shortest path delays seen at the inputs of each gate of circuit *b9*. As predicted, the path delay difference between the longest and shortest path a gate tends to increase for the gates placed closer to the output. Note how the short path delays are adjusted by delay buffer insertion to meet the balancing constraint.



(a) before padding (b) after padding

Figure 8: Longest and shortest path delays at the gate inputs of circuit *b9*

The circuit was automatically synthesized using procedures *balance* and *pad*, described in Section 4.2, integrated with the SIS program. Using basic 2-input gates and inverters, and two types of delay buffers from the MSU standard cell library, version 2.2, gate delay parameters were constructed for the SIS library and the MOSIS 2μ p-well process. The circuit was synthesized to allow for 3 waves, and TimberWolf [38] was used to generate the physical layout. The simulated value of the minimum clock period of the circuit is $2.78ns$ with latency $7.37ns$. CAzM [7] was used for the circuit simulation.

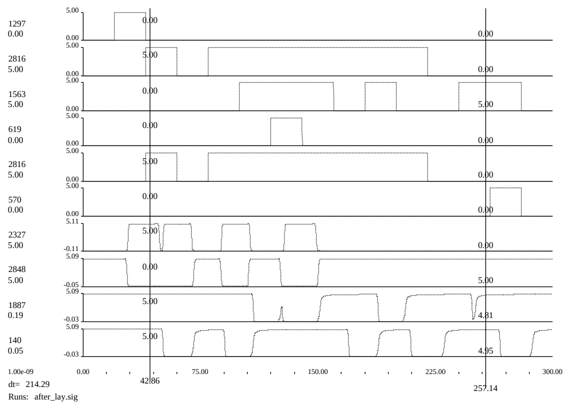
Table 1 compares the result of CAzM simulation for the synthesized circuit and the circuit extracted from the final layout. It shows the longest/shortest arrival times measured in ns at the selected outputs before and after the layout.

output	before layout	after layout
h1	NA/6.20	NA/8.88
i1	7.52/4.60	10.79/5.73
q0	7.36/5.86	11.39/8.47
d1	7.13/5.47	10.87/6.29

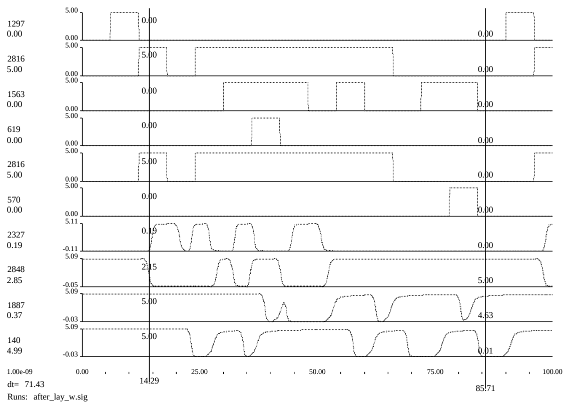
NA : no sensitization vector was found for the longest path to h1.

Table 1: CAzM results: delays of the synthesized circuit *b9* before and after layout

One can observe as much as $4ns$ increase in long path delay and $1ns$ increase in short path delay in the laid out circuit. This increase can be attributed to the wiring delay introduced by routing. The final circuit has latency of $11.39ns$ and the degree of wave-pipelining equal to 2, even though the circuit was synthesized to incorporate 3 waves. Again, this is due to unaccounted for physical effects of placement and routing. Considering the fact that the layout synthesis was performed without any consideration for timing optimization, it is believed that obtaining the physical circuit satisfying the target constraint is possible by using more sophisticated timing-driven layout tools. Fig.9 shows the plot of CAzM simulation result, with inputs appled every (a) $20ns$ and (b) every 6ns. Notice how the two waveforms, scaled accordingly to account for different clock period, match closely.



(a) Operating at 20ns clock cycle



(b) Operating at 6ns clock cycle

Figure 9: CAzM simulation results for circuit *b9* for 2 different values of clock period

6 Real Designs with Wave-pipelining

Many attempts have been made at using wave-pipelining in digital systems since Cotten proposed the wavepipelined technique in 1969 [6]. Prior to 1990, many designs of wave-pipelining systems have been done with bipolar technology, including a floating point unit [1], an experimental computer [36], and a population counter [45]. In general, CMOS has been viewed unfavorably for wave-pipelining systems due to the data-dependent delay from serial and parallel transistor connections in static CMOS gates. However, recently several university research groups have shown the feasibility of CMOS implementations of wave-pipelining. In addition, industry has begun to apply wave-pipelining in RAM designs using both BiCMOS and CMOS technology. The activities relating to wave-pipelining span theoretical formulation, CAD tool research, and design projects that include dedicated processors and static/dynamic RAMs. In the following sections, we will focus on the design activities using wave-pipelining. Tables 2 and 3 list the design activities in both academia and industry.

6.1 Wave-pipelined Designs by Academia

The main hurdle for designing a wave-pipelined system is the two-sided constraints on the timing. The problem becomes even more complicated if a system has multiple wave-pipelines with feedback [9, 13]. Thus, designers must address the various sources of delay imbalance in the design at the gate level, the circuit level, and at the detailed layout level. The delay imbalance must also be minimized for environmental variations such as voltage and temperature fluctuations. Also, care must be given in the design of an on-chip test structure so that high speed testing can be done without the requirement for high speed input/output. This unique feature is employed by most of the university prototypes presented in this section.

Researchers have devised various circuits to overcome the imbalances in successful implementations of wave-pipelined systems, Examples include wave-domino gates, biased-CMOS gates, static CMOS gates, and multiplexer-based gates.

In order to demonstrate the high speed capabilities of wave-pipelining techniques using a conservative CMOS process, most university prototypes share some common aspects in architecture, circuit and layout level. First, they are large circuits, where the effect of circuit design as well as the properties of the interconnect affect the circuit performance. Second, they can be implemented by regular structures, therefore, wave-pipelining techniques can be more easily applied. Third, a major portion of a prototype, such as the carry tree, usually has a circuit structure where each cell has regular fan-out. Therefore, the cell output capacitances are mostly dominated by the length of the interconnection wires and next input gates, whose lengthes can be predicted due to the circuit regularity. Lastly, due to lack of commercial tools that are directly applicable to designs using wave-pipelining, each group has more or less developed in-house design analysis and optimization tools which enable VLSI design using wave-pipelining.

Wong et al. at Stanford University have designed a 63 bit population counter in ECL/CML technology [45]. The counter achieves 2.5 waves and is the first bipolar LSI chip that uses wave-pipelining.

Liu et al. at North Carolina State University have reported an architecture, circuit, layout, and testing techniques for achieving high speedup in a CMOS parallel adder using wave-pipelining [29]. The adder achieves nine waves at 250 MHz and is implemented by biased-CMOS gates with 2 μm CMOS technology. The design depends very much upon the CAD tools at circuit level in order to carefully achieve delay balance.

Klass et al. at Stanford University have designed and tested a 16x16 CMOS wave-pipelined multiplier. The multiplier is implemented by *static gate* with 1.0 μm CMOS technology. It achieves 3.7 waves at the clock rates between 330 and 350 MHz despite the substantial effects of data-dependent delay [25]. Circuit level tools are also important for this design.

Using 2 μm CMOS domino gates, Lien and Burleson at the University of Massachusetts, have designed a 4x4 Wallace tree multiplier that achieves two waves in the multiplier circuit [28].

Nowka and Flynn of Stanford University have designed a CMOS vector unit which includes a vector register file, an adder, and a multiplier [34]. The vector unit is implemented with 1 μm CMOS technology and simulated at 300 MHz. In this design, an adaptive supply method is used to counteract the effects of process variation.

6.2 Wave-pipelined Designs by Industry

The application of wave-pipelining in industry has been almost exclusively in the area of fast RAMs and fast switches (Table 3. High speed S/DRAM is necessary to boost the overall performance of microprocessor systems. Due to the nature of cache systems, pipelining is particularly appropriate for reducing the access and cycle time for S/DRAM. A shorter cycle time is usually achieved by pipelining the RAM by the insertion of latches in appropriate places. However, the insertion of latches in a RAM is prohibited due to the costs of clock distribution, and the extreme area overhead. Since S/DRAM already use a variety of exotic timing methods, wave-pipelining is an efficient way to reduce access/cycle time without additional registers. Along with the reduced access time, wave-pipelining provides the additional advantages of reduced latency and power dissipation. Several successful RAM designs using wave-pipelining have been reported [4, 26, 2, 32, 47, 16].

It is fair to say that the design of S/DRAM using wave-pipelining has become a trend in industry. Chappel et. al. designed a "bubble pipelined" RAM chip with 2ns access time. Wave-pipelining is used in the cache RAM access in the HP Snake workstation. NEC has designed a 220MHz, 16Mb BiCMOS wave-pipelined SRAM. Hitachi has designed a

Table 2: Wave-pipelined Designs by Academia

Institution	*Status*	*Design*
Chiao-Tung U.	simulated	CMOS wave-pipelined multiplier [19]
Delft U.	fabricated/tested	16x16 multiplier (CMOS - 3 waves) [25]
NC State U.	fabricated/tested	16-bit adder (2 μm CMOS and 9 waves) [29],
NC State U.	fabricated/tested	digital sampler (1.2 μm CMOS and 25ps resolution) [14]
NC State U.	fabricated/tested	multiplier (1.2 μm CMOS and 4 waves) [33]
NC State U.	fabricated/tested	data recovery (1.2 μm CMOS) [20]
NC State U.	fabricated/tested	pattedern generator (1.2 mum CMOS) [31]
SUNY (Buffalo)	simulated	image processor (1.2 mum CMOS) [42]
Stanford U.	fabricated/tested	63-bit population counter (ECL with 2.3 waves) [45]
Stanford U.	fabricated/tested	16x16 multiplier (1 μm CMOS - 3.5 waves) [25]
Stanford U.	fabricated	vector unit (1 μm CMOS - 300 MHz) [34]
TI-India	simulated	8-bit multiplier (CMOS) [11]
U. of Colorado	designed	Optical computer with wave-pipelining [35]
U. of Mass.	fabricated/tested	multiplier and parity generator (2 μm CMOS) [28]

300MHz, 4Mb wave-pipelined CMOS SRAM. Moreover Hitachi proposed a concept of dual sensing latch circuit in order to achieve a shorter cycle time at 2.6ns. Hyundai has designed a 150MHz, 8-banks, 256Mb synchronous DRAM. All of these designs sustain from three to four waves.

Table 3: Wave-pipelined Designs by Industry

Campany	*Design*
DEC	L3 Cache Memory for AXP 21164 [2]
GTE	SONET-OC12 switch [5]. (CMOS - 4 waves)
HP	Cache RAM in the HP Snake Workstation [26] (CMOS - 1.16 waves)
IBM	"Bubbled pipelining" in RAM chip [4] (CMOS - 4 waves)
IBM	Crosspoint switch [41] (ECL- 2 waves)
NEC	SRAM chip [32] (BiCMOS - 2 waves)
Hitachi	SRAM (CMOS - 330MHz and 2.6ns) [16, 43]
Hyundai	DRAM (CMOS - 150MHz, 256Mb) [47]

7 Open Problems

Despite recent research in wave-pipelining and other aggressive timing approaches, a number of open problems still exist, in particular related to manufacturability. As with the research already surveyed in this paper, solutions to these problems may have broader application than just wave-pipelining.

1. **Testing:** Delay testing for both short and long paths, especially considering false paths, is currently a hot topic in the test community. Wave-pipelining presents additional challenges due to the difficulty of observing internal points in what looks like a combinational circuit but behaves as a sequential circuit. Testing for the interaction of long and short paths and the sequential false paths which may arise all warrant further study.

2. **Low-power:** Is wave-pipelining a low-power technique? Despite reduced clock loading, it appears not, since at low power supply levels, delay variations tend to worsen. However, alternative low power methods, such as dynamic power supply adjustment and power down techniques could favor wave-pipelining.

3. **Higher-level synthesis:** High level synthesis tools are only recently able to exploit internally pipelined modules. Wave-pipelined modules can present an additional degree of freedom to such tools, however the methods for modelling and verifying the timing properties unique to wave-pipelining for use at higher levels still need to be developed.

4. **Dynamic power supply and clock tuning:** One approach to process and environmental variation is the dynamic tuning of both power supplies and clock rates. This has been explored in [34] but is still far from being a widely used technique.

5. **Physical design issues:** Deep sub-micron technologies will continue to complicate abstractions of physical design. As interconnect delays continue to dominate gate delays, most timing optimization, whether it be longest path, shortest path or balancing, will have to be done mostly at the physical level.

6. **Parameter variation:** Advanced technologies tend to prioritize density and speed at the cost of wider parameter variation. This will make clock distribution and design centering more of a challenge and could severely limit a designer's choice of timing schemes. One solution is the use of asynchronous circuit techniques. However, as the ever-increasing density and speed surpass the capabilities of designers to exploit their use, it may be that a truly "advanced" technology of the future will prioritize parameter variation, much like analog processes of today.

8 Conclusion

In this tutorial and research survey we attempted to present a cogent and readable treatment of wave-pipelining for VLSI circuit designers, CAD tool developers, and system architects. We presented the theory of wave-pipelining which required a new approach to dealing with two-sided timing constraints. We emphasized the effect of process and environmental variations on wave-pipelining behavior. We then demonstrated algorithms for synthesizing wave-pipelined circuits at the logic and circuit levels. Results showed that synthesized circuits can approximate the performance of manual designs. We then addressed the question of how practical wave-pipelining is in manufacturable circuits. We concluded that while wave-pipelining does require a new way of thinking about timing constraints, it is no more complex than many of the other advanced circuit techniques currently being used in state-of-the-art integrated circuits. While preparing this paper, the list of recent commercial applications of wave-pipelining, most notably in RAMs, has continued to grow and we anticipate further applications in deeply pipelined DSP and communications architectures. Finally, we presented some open research problems related to wave-pipelining, most of which are applicable to any aggressively timed VLSI design in an advanced CMOS technology.

References

[1] S. Anderson, J. Earle, R. Goldschmidt, and D. Powers. The IBM system/360 model 91 floating point execution unit. *IBM Journal of Research and Development*, pages 34–53, January 1967.

[2] P. Bannon and A. Jan. Today's microprocessor aim for flexibility. *EE Times*, pages 68–70, 1994.

[3] W. Burleson, E. Tan, C. Lee. A 150 Mhz Wave-pipelined Adaptive Digital Filter in 2μ CMOS. *VLSI Signal Processing Workshop*, 1994.

[4] T. I. Chappell, B. A. Chappell, S. E. Schuster, J. W. Allan, S. P. Klepner, R. V. Joshi, and R. L. Franch. A 2-ns cycle, 3.8 ns access 512-kb CMOS ECL SRAM with a fully pipelined architecture. *IEEE Journal of Solid State Circuits*, pages 1577–1585, November 1991.

[5] M. Cooperman and P. Andrade. CMOS gigabit-per-second switching. *IEEE Journal of Solid-State Circuits*, June 1993.

[6] L. W. Cotten. Maximum-rate pipeline systems. In *1969 AFIPS Proc. Spring Joint Computer Conference*, pages 581–586, 1969.

[7] W. M. Jr. Coughran, E. Grosse, and D. J Rose. CAzM: A Circuit Analyzer with Macromodeling. *IEEE Transactions on Electron Devices*, ED-30(9):1207–1213, April 1983.

[8] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang. MIS: A multiple-level logic optimization system. *IEEE Transactions on Computer-Aided Design*, CAD-6(6):1062–1081, Nov. 1987.

[9] B. Ekroot. *Optimization of Pipelined Processors by Insertion of Combinational Logic Delay.* Ph.D. Thesis, Stanford University, 1987.

[10] J. Fishburn. Clock Skew Optimization. In *IEEE Trans. on Computers*, 1990.

[11] D. Ghosh and S. Nandy. Design and realization of high-performance wave-pipelined 8x8 b multiplier in CMOS technology. *IEEE Trans. on VLSI*, pages 37–48, 1995.

[12] C.T. Gray, T. Hughes, S. Arora, W. Liu, and R. Cavin III. Theoretical and Practical Issues in CMOS Wave Pipelining. In *Proc. of VLSI'91*, 1991.

[13] C.T. Gray, W. Liu, and R. Cavin III. Timing Constraints for Wave-Pipelined Systems In *IEEE Transactions on Computer-Aided Design*, Vol.13, No.8, pp. 987-1004, Aug., 1994

[14] C. Gray, W. Liu, W. van Noije, T. Hughes, and R. Cavin. A sampling technique and its CMOS implementation with 1 Gb/s bandwidth and 25 ps resolution. *IEEE Journal of Solid State Circuits*, pages 340–349, March 1994.

[15] S. Gupta. *Private Communication.* Nov 1994.

[16] K. Ishibashi et. al. A 300mhz 4-mb wave-pipelined CMOS SRAM using a multi-phase PLL. In *ISSCC95*, pages 308 – 309, 1995.

[17] D.A. Joy and M.J. Ciesielski. Placement for Clock Period Minimization with Multiple Wave Propagation. In *Proc. 28th Design Automation COnference*, 1991.

[18] D.A. Joy and M.J. Ciesielski. Clock Period Minimization with Wave Pipelining In *IEEE Transactions on Computer-Aided Design*, April, 1993.

[19] S. T. Ju and C. W. Jen. A high speed multiplier design using wave pipelining technique. In *IEEE APCCAS*, pages 502–506, Australia, 1992.

[20] J. Kang, W. Liu, and R. Cavin. A monolithic 625mb/s data recovery circuit in 1.2 μm CMOS. In *Custom Integrated Circuit Conference*, pages 625–628, 1993.

[21] T. S. Kim, W. Burleson, and M. Ciesielski. Logic Restructuring for Wave-pipelined Circuits. In *Proceedings of the International Workshop on Logic Synthesis*, 1993.

[22] T-S. Kim, W. Burleson, M. Ciesielski, "Delay buffer insertion for Wave-pipelined Circuits," Notes of *IFIP International Workshop on Logic and Architecture Synthesis*, Grenoble, France, Dec. 1993.

[23] F. Klass, M.J. Flynn, and A.J. van de Goor. Pushing the Limits of CMOS Technology: A Wave-Pipelined Multiplier Hot Chips, 1992.

[24] F. Klass and M.J. Flynn. *Comparative Studies of Pipelined Circuits.* Technical Report CSL-TR-93-579, Stanford University, July 1993.

[25] F. Klass, M. Flynn, and A. van de Goor. Fast multiplication in vlsi using wave pipelining techniques. *Journal of VLSI Signal Processing*, 7:233–248, 1994.

[26] Charlie Kohlhardt. PA-RISC processor for "Snake" workstations. In *Hot Chips Symposium*, pages 1.20–1.31, 1991.

[27] W. K.C. Lam, R. K. Brayton, and A. Sangiovanni-Vincentelli. Valid Clocking in Wavepipelined Circuits. In *Proceedings of the International Conference on Computer-Aided Design*, 1992.

[28] W. Lien and W. Burleson. Wave-domino logic: Theory and applications. *IEEE Transactions on Circuits and Systems - II Analog and Digital Signal Processing*, 42(2):78–91, 1995.

[29] W. Liu, C. Gray, D. Fan, T. Hughes, W. Farlow, and R. Cavin. A 250-mhz wave pipelined adder in 2-μm CMOS. *IEEE Journal of Solid State Circuits*, pages 1117–1128, Sept. 1994.

[30] B. Lockyear and C. Ebeling. The practical application of retiming to the design of high-performance systems. In *ICCAD93*, pages 288 – 295, 1993.

[31] G. Moyer, M. Clements, W. Liu, T. Schaffer, and R. Cavin. A technique for high-speed, fine-resolution pattern generation and its CMOS implementation. In *16th Conference on Advanced Research in VLSI*, pages 131–145, 1995.

[32] K. Nakamura, S. Kuhara, and et. al. A 220-mhz pipelined 16-mb BiCMOS SRAM with PLL proportional self-timing generator. *IEEE Journal of Solid State Circuits*, pages 1317–1322, November 1994.

[33] V. Nguyen, W. Liu, C. T. Gray, and R. K. Cavin. A CMOS multiplier using wave pipelining. In *Custom Integrated Circuits Conference*, pages 12.3.1–12.3.4, San Diego, CA, May 1993.

[34] K. Nowka and M. Flynn. System design using wave-pipelining: A CMOS VLSI vector unit. In *ISCAS95*, pages 2301 – 2304, 1995.

[35] J. Pratt and V. Heuring. Delay synchronization in time-of-flight optical systems. *Applied Optics*, 31(14):2430 – 2437, 1992.

[36] L. Qi and X. Peisu. The design and implementation of a very fast experimental pipelining computer. *Journal of Computer Science and Technology*, 3(1):1–6, 1988.

[37] K.A. Sakallah, T.N. Mudge, T.M. Burks, and E.S. Davidson. Synchronisation of Pipelines. In *IEEE Transactions on Computer-Aided Design*, 1993.

[38] C. Sechen and A. L. Sangiovanni-Vincentelli. Timber Wolf: A Placement System for Integrated Circuits. *Journal of Solid-State Circuits*, SC-20(4):510–522, April 1985.

[39] N. V. Shenoy, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Minimum Padding to Satisfy Short Path Constraints. In *Proceedings of the International Workshop on Logic Synthesis*, 1993.

[40] E. M. Sentovich, K. J. Singh, C. Moon, H. Savoy, and R. K. Brayton. Sequential Circuit Design Using Synthesis and Optimization. In *Proceedings of the International Conference on Computer Design*, 1992.

[41] H. Shin, J. Warnock, M. Immediato, K. Chin, C.-T. Chuang, M. Cribb, D. Heidel, Y.-C. Sun, N. Mazzeo, and S. Brodskyi. A 5Gb/s 16x16 Si-bipolar crosspoint switch. In *IEEE International Solid State Circuits Conference*, pages 228–229, February 1992.

[42] R. Krishnamurthy, R. Sridhar. A CMOS Wave-pipelined Image Processor for Real-time Morphology. *IEEE Int'l Conf. on Computer Design*, pages 638–643, Oct. 1995.

[43] S. Tachibana and H. Higuchi et. al. A 2.6ns wave-pipelined CMOS SRAM with dual-sensing-latch circuits. *IEEE Journal of Solid State Circuits*, pages 487–490, April 1995.

[44] D. Wong, G. De Micheli, and M Flynn. Inserting active delay elements to achieve wave pipelining. In *Proceedings of the International Conference on Computer-Aided Design*, pages 270–273, November 1989.

[45] D. Wong, G. De Micheli, M. Flynn, and R. Huston. A Bipolar Population Counter Using Wave Pipelining to Achieve 2.5 $\times$ Normal Clock Frequency. In *IEEE Journal of Solid State Circuits*, Vol. 27, May 1992.

[46] D. Wong, G. De Micheli, and M Flynn. Designing High Performance Digital Circuits Using Wave Pipelining: Algorithms and Practical Experiences. In *IEEE Trans. on Computer-Aided Design*, Vol. 12, January 1993.

[47] H. Yoo et. al. A 150mhz 8-banks 256m synchronous DRAM with wave pipelining methods. In *ISSCC95*, pages 250 – 251, 1995.