Designing High-Performance Digital Circuits Using Wave Pipelining

Derek Wong, Giovanni De Micheli, and Michael Flynn *
Department of Electrical Engineering
Stanford University
Stanford, California 94305

Abstract

Wave pipelining is a technique for pipelining digital systems that can increase clock frequency in practical circuits without increasing the number of storage elements. In wave pipelining, multiple coherent waves of data are sent through a block of combinational logic by applying new inputs faster than the delay through the logic. Ideally, if all paths from input to output have equal delay, then the circuit's clock frequency is limited by rise/fall times, clock skew, and set-up and hold times of the storage elements. In practice, due to the above limits and variations in fabrication, clock frequency can be increased by a factor of 2 to 3 using the best available design methods.

We present algorithms to automatically equalize delays in combinational logic circuits to achieve wave pipelining. For both normal and wave-pipelined circuits, the algorithms also optimally minimize power under delay constraints.

An analysis of circuit technologies shows that CML and super-buffered ECL are well suited for designing circuits with uniform delay. Regular ECL is not as good, and static CMOS is substantially worse.

1 Introduction

Wave pipelining is a design method that can boost the pipeline rate of a system without using additional registers.

In ordinary pipelined systems, there is one "wave" of data between register stages. When a new set of values is clocked into one set of registers, the values are allowed to propagate to the next set of registers before the first set is clocked again.

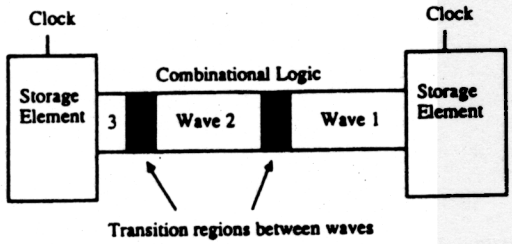


Figure 1: In wave pipelining, multiple coherent waves of data are sent through combinational logic acting as a pipeline.

In contrast, wave pipelining is the use of multiple coherent "waves" of data between storage elements (see Figure 1). This is achieved by clocking the system faster than the propagation delay between registers. In this method, the data values at the first set of registers are changed before the old data values have propagated to the next set of registers. The capacitance in the combinational logic circuit is being used to store values for pipelining.

For example, a fast 64-bit, floating-point multiplier implemented in combinational logic might have a propagation time of 10 nsec. Instead of 100 Mhz, the multiplier could operate using 3 pipeline waves to achieve a clock frequency of 300 Mhz with the same 10 nsec latency.

To operate at the highest possible clock frequency, all path delays from every starting to every ending storage element must be the same. Clock skew, variations in path length, rise/fall times, and the setup time of the storage element limit the maximum pipeline rate.

The possible advantages of wave pipelining are the following:

*This work was supported in part by an NSF Graduate Fellowship and a supplement from Stanford University. This research was also supported by the Center for Integrated Systems at Stanford. The work was performed using equipment provided by NASA under contract NAGW 419. This work was also supported by NSF, DEC, and AT&T under a PYI award.

- wide applicability to all pipelined digital systems
- lower power, area, and delay by using fewer stages of storage elements
- very high rates of pipelining without the added delay of storage elements dominating the pipeline latency.

The following disadvantages exist:

- requires specialized design algorithms to equalize the length of all paths.
- harder to use at the system level. A wave-pipelined chip must be run at the proper clock frequency; it does not work at higher or lower rates. When a system using multiple wave-pipelined chips is assembled, the chips must have matched speeds.
- may need to add delay buffers to lengthen some short paths. This increases the area of some circuits.

2 The Wave Pipelining Project

Previous work in this area includes the following:

- Anderson and Cotten first described the concept as used in the floating point unit of the IBM 360/91 [Anderson67, Cotten69].
- Lin and Xia designed and implemented an experimental computer using wave pipelining in its arithmetic units [Lin88].
- Fawcett described the theory of pipelined system clocking in detail [Fawcett75]. He developed detailed equations for the maximum clock rate of Earle latch systems as a function of many parameters. His experimental work did not include wave pipelining although his clocking theory applies.
- Ekroot developed some theory of wave pipelining [Ekroot87]. Assuming gates and modules with fixed delays, he developed linear programs to determine where to insert delay elements to balance the circuit. Also, he compared the minimum clock period using wave pipelining and regular pipelining.

Our ultimate goal is to build an actual wave pipelining chip in VLSI, identifying and solving the necessary practical problems enroute. We are focusing on the following goals:

- Analyzing technologies and speed limits.
- Designing the algorithms and developing the necessary CAD tools to automatically balance the delays in combinational logic circuits.
- Designing, building, and testing a sample chip design.

The IBM 360/91 and the experimental computer by Lin and Xia were designed using manual design techniques to balance circuits of fixed-delay gates. In contrast, we are developing new algorithms to balance the delays using gates with adjustable delay.

Unlike Ekroot's methods, our methods minimize power consumption and added circuitry. Also, we use gates with adjustable delay vs. power rather than assuming fixed delays.

This paper discusses some of the important issues in wave pipelining. First, we explain the frequency limits of wave pipelining. Next, we introduce new algorithms that balance delays to achieve wave pipelining. As a by-product, these methods can also be used to optimize power vs. delay for both normal and wave-pipelined circuits. Then, various technologies are graded by their suitability for wave pipelining. We show why ECL and CML are better suited for this technique than CMOS. Finally, we state our project status.

3 Increasing the Maximum Pipeline Rate

3.1 The Minimal Clock Period Relation

The maximum pipeline rate is limited by technological parameters. Clocking the circuit at a frequency above the limit would mix the waves of data together.

In Appendix 1, we show that the minimum clock period at which a wave-pipelined circuit can be clocked is bounded by

$$t_{CP} > \Delta t_P + 2 * \Delta C + t_{SH} + t_{RF}$$

where

- t_{CP} = clock period
- t_P = propagation time of the longest path in the combinational logic
- Δt_P = max difference in path length over worst-case design, process, and environment
- ΔC = worst-case clock skew
- t_{SH} = set-up plus hold time for edge-triggered registers.

  (For latches, t_{SH} = length of transparent period plus hold time.)
- t_{RF} = worst-case rise or fall time (10% to 90% voltage swing) at the last logic stage

3.2 Reducing the Minimal Clock Period

A designer would like to minimize t_{CP} by attacking each of its components. We assume here that t_{SH} and t_{RF} are parameters that depend on the technology. The clock skew ΔC can be minimized using standard design techniques.

Therefore, we consider the problem of minimizing t_{CP} by reducing Δt_P. The normal clock period t_{CPN} without using wave pipelining would be bounded by

$t_{CPN} > t_P + t_S$ (plus possibly clock skew depending on the design technique)

where t_S is the set-up time of the storage element.

In most cases, t_{CP} can be made much smaller than this by reducing Δt_P to a small fraction of t_P.

The path variation Δt_P arises from several sources: path differences due to design, process and temperature-induced variations within one chip, and data-dependent delay variations. Process and temperature-induced variations are unavoidable, but their effects are limited within one chip. Data-dependent delays can be limited by selecting the proper technology. Using our algorithms, the worse-case Δt_P can be reduced to 10-20% of t_P in most cases.

For example, assume that in a hypothetical technology ΔC, t_{SH}, and t_{RF} equal to 500ps each (or one gate delay). Then the clock period is bounded by

$t_{CP} > \Delta t_P + 2$ nsec

In this example, Δt_P can be reduced to 1 to 2 nsec for a circuit that has $t_P = 10$ nsec (20 gate delays), leading to a clock period t_{CP} of 3 to 4 nsec.

Thus, one should be able to fit 2 to 3 waves within a typical wave-pipelined circuit.

4 Using Wave Pipelining in a System Design

A combinational circuit like a floating point adder is the ideal type of circuit for wave pipelining. Typically, a system designer may have one of two design goals when using wave pipelining:

1. Build the fastest possible adder that can handle as many waves as possible. The designer will fit the rest of the system to the adder after t_P and t_{CP} are determined.

   In this case, the desired delay t_P is equal to the critical path delay with all gate drives set at maximum. The tuning algorithms minimize Δt_P while keeping the delay equal to t_P.

2. Build an adder that has a delay t_P determined by the design of the rest of the system. For example, this might occur if the clock period is based on cache speed and the semantics of the system design require that the adder be 3 waves deep.

   In this case, the desired delay t_P is determined by external constraints. The objective is to minimize Δt_P at that delay.

Our algorithms are designed to balance the circuits to a user-specified propagation delay called D_{MAX} while minimizing power and added area. If desired, the designer can iterate while varying D_{MAX} to construct a range of solutions with varying t_{CP}, power and area.

5 Algorithms for Designing Wave-Pipelined Circuits

If possible, a wave-pipelined circuit should be designed to have balanced paths under nominal fabrication and temperature conditions. When a technology satisfies some assumptions, this can be achieved by using our balancing algorithms. In section 6, we show that technologies exist that satisfy these assumptions.

In this section, we describe the CAD algorithms and tools for automatically taking a combinational logic circuit and balancing its delays. One of the tools can also be used to set the gate drives in both normal and wave-pipelined circuits to achieve the minimal possible power for a given maximum delay D_{MAX}.

Our methods are designed primarily to work with ECL/CML circuits.

5.1 Problem Formulation

Given a combinational logic circuit without feedback, we wish to insert delay elements and adjust gate drives so that every path from an input to an output takes a certain nominal delay D_{MAX}. The added area and power should be minimized.

The following modeling assumptions are made:

1. Each gate propagates signals one way from inputs to outputs.
2. Gate delay can be adjusted by a parameter (e.g. tail current in ECL).
3. Adjusting the delay of a gate does not affect the delays of gates connected to its inputs

or outputs. (This is a good assumption for ECL/CML but not for MOS.)

4. Path delays can be increased by inserting active delay elements. These elements are buffers with one input and one non-inverting output.

5.2 Rough and Fine Tuning

We propose two ways of attacking the balancing problem: inserting delay elements and adjusting gate drives. We combine these in a two-step process to balance a circuit while minimizing power and added area:

1. *Rough tuning* inserts a minimal number of delay elements so that it is feasible to balance the circuit by just adjusting gate drives. Minimizing the number of inserted elements minimizes the added area.
2. *Fine tuning* adjusts gate drives so that the circuit is nominally balanced to a delay D_{MAX} with minimal power consumption.

Rough tuning can be performed prior to placement and route using a rough estimate of capacitative loads. Using gates that have constant area independent of power, fine tuning is best performed after placement and route because the exact value of each wire capacitance is then known. Following fine tuning, the maximum pipeline rate can be determined.

Designers currently perform rough tuning by hand so that the fine tuning program successfully balances the circuit. We have developed a rough tuning algorithm to automatically insert delay elements where needed. The rough tuning algorithm guarantees that the circuit can be subsequently balanced by fine tuning.

In this paper, we discuss fine tuning in detail. Rough tuning is discussed in another paper [Wong89].

5.3 Fine Tuning

The Fine Tuning Algorithm is presented as follows. First, we describe an abstraction of the circuit as a directed acyclic graph. Then we formulate two related problems: Fine Tuning and Power Minimization. We show that a solution to the Fine Tuning problem can be derived from a solution to Power Minimization. Then, we show that the Power Minimization problem can be solved by a linear program. Finally, the effects of uncertain delays and other practical considerations are presented.

5.3.1 Circuit Modeling Using a DAG

We model the circuit using a polar, weighted, directed acyclic graph (DAG). A graph is constructed from a circuit using the following steps:

1. Each node i is in one-to-one correspondence with a gate output i. (Note that in ECL/CML, there are 2 outputs per gate.)
2. A directed arc (i,j) exists iff the gate corresponding to j takes i as an input. We number all the arcs from 1 to NumArcs.
3. Each arc (i,j) indicates gate delay by a tuple weight

   (rising delay R, falling delay F, unate flag U) where

   - R is the delay of j's gate from the tail i to cause a rising transition at j.
   - F is the delay of j's gate from the tail i to cause a falling transition at j.
   - U indicates whether j's gate is positive unate, negative unate, or not unate with respect to the tail input.
4. A source node 0 is connected to all primary input nodes, and a sink node N is connected to all primary output nodes.

An example conversion from a circuit to a corresponding DAG is shown in Figure 2. In the example, 3 global inputs go to outputs 1 and 2 through an OR/NOR gate. Three arcs are drawn from the source node to each node 1 and 2 representing the delays through that OR/NOR gate. Similarly, output 2 goes through an OR gate to output 3. The corresponding arc goes from 2 to 3 and represents the OR gate's delay.

Path Delays

A source-to-sink path is a sequence of nodes and directed arcs leading from the source to the sink. *Path delay* denotes a pair (X, Y) where

- X = delay to rising transition at sink
- Y = delay to falling transition at sink

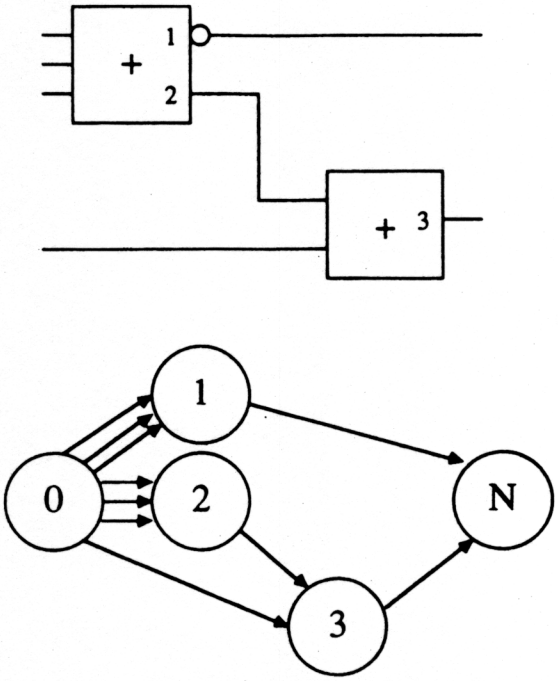
It is the sum of the arc weights using the unateness property. For example, for a path composed of positive unate arcs 1..n, the path delay is

$(\sum_{i=1}^{n} R[i], \sum_{i=1}^{n} F[i])$.

Using a series of negative unate arcs, the path delay equals

$(..R[n-2]+F[n-1]+R[n], ..F[n-2]+R[n-1]+F[n])$

For paths with some non-unate arcs, the idea of a unique path delay cannot be applied. However, max and min delays for a path can be



Each global input or gate output is connected to all of the nodes in its fanout.

Figure 2: Example of converting from a circuit to a DAG for fine-tuning

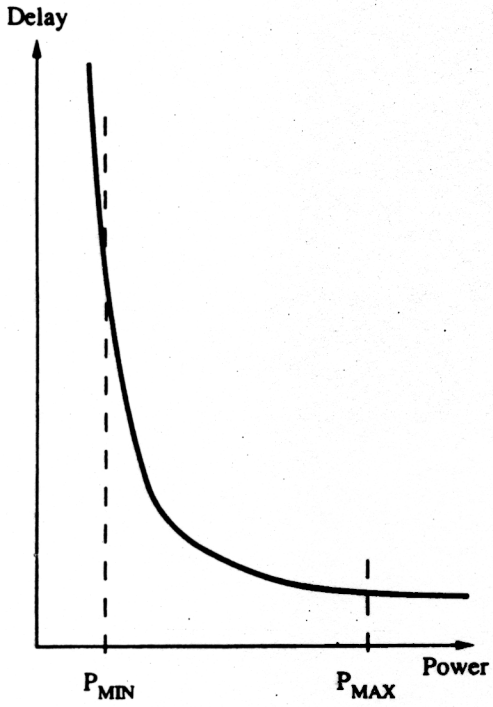


Figure 3: Delay functions should be convex and monotonically decreasing with respect to Power.

defined. For instance, suppose that the head of arc i represents the output of a non-unate gate. The max rising path delay up to arc i is defined recursively as

$d_{MAX-RISE}[i] = max(d_{MAX-RISE}[i-1] + R[i], d_{MAX-FALL}[i-1] + R[i])$

This $d_{MAX-RISE}[i]$ may be used for any max delay computations with arcs beyond arc i.

5.3.2 Problem Formulation for Fine Tuning and Power Minimization

Let us define the following four terms:

A. Defn. Combinational Logic Circuit with Fine Tuning Information

A *Combinational Logic Circuit with Fine Tuning Information* is a circuit represented by a DAG with the following delay model:

1. Drive (power or current) $P[i]$ for each gate i.
2. Load capacitance $L[i]$ for each gate output i.
3. Power limits $P_{MIN}[i]$ and $P_{MAX}[i]$ for each gate i,
4. Two delay functions for each arc (i,j) where node j is an output of gate k:

   $f[i,j,0](power\ \ P[k], load\ \ L[j])$ = time from tail i to a falling transition at head j.

   $f[i,j,1](power\ \ P[k], load\ \ L[j])$ = time from tail i to a rising transition at head j.

   We assume that these functions are convex and monotonically decreasing with respect to P (Figure 3). Technologies nearly always satisfy these assumptions over reasonable power bounds P_{MIN} and P_{MAX}.
5. A user-specified delay D_{MAX}.
6. $D[i,0]$ = max delay from source to a falling transition at node i.
7. $D[i,1]$ = max delay from source to a rising transition at node i.

B. Defn. Fine Tuning Problem

The following is the *Fine Tuning Problem*:

Given a Combinational Logic Circuit with Fine Tuning Information, find a vector of gate drives

P to

- Make the longest input-to-output path have nominal delay D_{MAX} and
- Minimize δ where the shortest path has nominal delay $D_{MAX} - \delta$.

A secondary goal is to minimize total power consumption $\sum_{i=1}^{n} P[i]$.

□

The parameter δ measures the path length difference due to design.

When gates have different rise and fall delays, a solution with $\delta = 0$ might not even exist. We therefore postulate a restriction of the Fine Tuning Problem, called a Simplified Fine Tuning Problem.

We develop a method for finding a solution to the Simplified Fine Tuning problem. This method also solves the full Fine Tuning Problem with a guaranteed bound on δ. If δ is small with respect to the other components of t_{CP}, then an efficient wave-pipelined implementation can be constructed.

C. Defn. Simplified Fine Tuning Problem

If the gates in a Fine Tuning Problem all have rising delays equal to falling delays, i.e.,

$f[i,j,0](P,L) = f[i,j,1](P,L) \ \forall i, j, P, L$ and

then it is a *Simplified Fine Tuning Problem.*

D. Defn. Power Minimization Problem

The following is the *Power Minimization Problem*:

Given a Combinational Logic Circuit with Fine Tuning Information, find a vector of gate drives P such that all input-to-output paths have AT MOST delay D_{MAX} with minimal total power consumption $\sum_{i=1}^{n} P[i]$.

5.3.3 Equivalence of Fine Tuning And Power Minimization Problems

At this point, we first explain how to solve the Simplified Fine Tuning problem. We address the full Fine Tuning problem later.

Let us begin with this definition:

Defn. S-Fine-Tunable Using Power Minimization

Given a Combinational Logic Circuit with Fine Tuning Information where all rising delays equal falling delays, i.e.,

$f[i,j,0](P,L) = f[i,j,1](P,L) \ \forall i, j, P, L$,

the circuit is *S-Fine-Tunable Using Power Minimization* iff the following property is true:

If $\hat{P}$ is a solution to the Power Minimization problem such that $P_{MIN}[i] \leq \hat{P}[i] \leq P_{MAX}[i]$ are not active constraints, then $\hat{P}$ is also a solution to the Simplified Fine Tuning Problem for the same circuit and D_{MAX} with δ as explained below.

Remarks

An analysis of the bounds on δ assumes that a circuit is rough tuned before fine tuning is applied. Rough tuning is explained in [Wong89].

Given delay elements that can implement any delays greater than zero, our rough tuning algorithm can always balance a circuit to be S-Fine-Tunable Using Power Minimization with $\delta = 0$.

However, if delay elements can only implement delays larger than T_{MIN}, then the circuit might not be perfectly balanced because delay elements with smaller delays are sometimes needed but are not implementable. The rough tuning algorithm guarantees that $\delta \leq nT_{MIN}/2$ where n is the maximum number of delay elements with delay less than T_{MIN} that the algorithm would like to insert on any input-to-output path. In practice, our algorithms may achieve better results, and $nT_{MIN}/2$ is considered small compared to the other parameters contributing to the bound on t_{CP} (for more details on rough tuning, we refer to [Wong89]).

5.3.4 Solution Method

The Power Minimization problem can be efficiently solved as a linear program by approximating the non-linear delay functions $f(P,L)$ by piece-wise linear functions. This is possible because the functions are assumed convex. In practice, the delay functions are roughly hyperbolic and can be well-approximated with a few line segments. Without this simplification, we would have a non-linear program whose solution would be costly for large circuits.

This linear programming formulation is related to a non-linear program used by Marple [Marple86] for optimizing MOS circuit power under timing constraints.

A. Linear Program

The Power Minimization problem can be formulated as the following linear program:

Minimize $P = \sum_{i=1}^{n} P[i]$.

subject to the following constraints:

1. $P_{MIN}[i] \leq P[i] \leq P_{MAX}[i]$ for each gate i
2. $D[0,0] = D[0,1] = 0$

3. $D[N,0] \leq D_{MAX}$

   $D[N,1] \leq D_{MAX}$

4. We build delay constraints from each arc (i,j) depending on the arc's unate flag U. Let x be the gate corresponding to output j.

   If U is positive unate:

   $D[i,1] + f[i,j,1](P[x], L[j]) \leq D[j,1]$

   $D[i,0] + f[i,j,0](P[x], L[j]) \leq D[j,0]$

   If U is negative unate:

   $D[i,1] + f[i,j,0](P[x], L[j]) \leq D[j,0]$

   $D[i,0] + f[i,j,1](P[x], L[j]) \leq D[j,1]$

   If U is not unate:

   $D[i,1] + f[i,j,1](P[x], L[j]) \leq D[j,1]$

   $D[i,1] + f[i,j,0](P[x], L[j]) \leq D[j,0]$

   $D[i,0] + f[i,j,0](P[x], L[j]) \leq D[j,0]$

   $D[i,0] + f[i,j,1](P[x], L[j]) \leq D[j,1]$

   By replacing each non-linear function $f(P,L)$ (where L is a constant) by a piecewise linear function, these constraints can be made linear. One linear constraint is superimposed on each line segment of the piecewise linear function. Since the function is convex, each such constraint is binding only along its line segment. This turns each constraint above into a set of linear constraints, each using P only as a linear variable.

5.3.5 Properties of the Algorithm When Delays are Uncertain

Due to process variations and temperature-induced effects within one chip and data-dependent delays, a gate's speed is not perfectly controllable relative to other gates within a chip.

Proposition 1. Balancing Using Uncertain Delays

Suppose each arc i has a delay $d[i]$ bounded by

$(1-\alpha)d_{MAX}[i] \leq d[i] \leq d_{MAX}[i]$ for some α, $0 \leq \alpha \leq 1$.

The parameter α measures the speed variations that do not track within a chip.

If Power Minimization is performed using the $d_{MAX}[i]$'s as the delays and the result satisfies the definition of S-Fine-Tunable Using Power Minimization with a bound δ, then a similar bound δ_2 can be defined to include the effects of uncertain delays:

$\delta_2 \leq \alpha(D_{MAX} - \delta) + \delta$

5.3.6 Solving the Full Fine Tuning Problem

When rising delays do not equal falling delays, the Power Minimization method is a reasonable heuristic for a difficult problem. Since each gate has only one power parameter which controls a rising and falling delay per output, solutions with nearly balanced delays may not even exist in some cases.

The following defines the use of Power Minimization to solve the full Fine Tuning Problem:

Defn. F-Fine-Tunable Using Power Minimization

Given a Combinational Logic Circuit with Fine Tuning Information, define

- $d_{MAX}[i]$ = max(rising delay, falling delay) of arc i
- $d_{MIN}[i]$ = min(rising delay, falling delay) of arc i

Suppose each arc i has delay $d_{MIN}[i]$ bounded by

$(1-\alpha)d_{MAX}[i] \leq d_{MIN}[i] \leq d_{MAX}[i]$ for some α, $0 \leq \alpha \leq 1$.

Suppose that Power Minimization is performed on the circuit using both the rising and falling delays.

The circuit is *F-Fine-Tunable Using Power Minimization* iff the following property is true:

If $\hat{P}$ is a solution to the Power Minimization problem such that $P_{MIN}[i] \leq \hat{P}[i] \leq P_{MAX}[i]$ are not active constraints, then $\hat{P}$ also solves the Fine Tuning Problem with
$\delta \leq \alpha(D_{MAX} - \delta') + \delta'$
where δ' is explained below.

Remarks

- Any circuit that has been Rough Tuned using the algorithm described in [Wong89] is F-Fine-Tunable Using Power Minimization.

  Let T_{MIN} be the larger of the minimum rising delay and falling delay for a delay element.

  If $T_{MIN} = 0$, the circuit can always be balanced with $\delta' = 0$.

  If $T_{MIN} > 0$, the best bound that our algorithm can guarantee is $\delta' = nT_{MIN}/2$ where n is the maximum number of delay elements with delay less than T_{MIN} that the Rough Tuning algorithm desires to insert on any input-to-output path [Wong89].

- If negative unate logic is used, then the rise/fall differences will tend to cancel out, so the actual δ can be much better than the definition suggests.

- Since ECL emitter-followers have more discrepancy between t_{RISE} and t_{FALL} at low power levels, it may be desirable to set P_{MIN} relatively high for emitter-followers.

5.3.7 Practical Considerations

Space only permits brief, informal discussions of the topics below.

ECL/CML gates are actually tunable in small discrete steps rather than continuously because resistors have integral dimensions in VLSI. However, the rounding error should be small compared to other components of Δt_P such as process variations. After rounding, the max and min paths can be computed again to get a revised Δt_P.

In practice, even when the Power Minimization solution sets some $P[i]$'s at P_{MAX} or P_{MIN}, this method is still a very good heuristic for balancing the circuit within the previously-stated bounds on δ. In fact, the rough tuning algorithm is designed to never create circuits where Power Minimization would fail to meet the bounds on δ. Hence, our combined tuning procedure always balances a circuit.

5.3.8 Implementation

The above algorithm has been implemented. A fine-tuning program reads a circuit and gate library description and produces a linear program. This is then fed into a standard linear program solver called MINOS [Murtagh87] that uses the Simplex method to solve for the vector P. The fine-tuning program then produces a modified circuit file with all the tail currents set.

A run on a test circuit of 75 gates took 4.8 CPU minutes to solve on a uVAX 3200. In the solution, the max delays of all the primary outputs were set equal to D_{MAX}. The variation δ in nominal path length was not zero because the circuit was not rough tuned and had different rise/fall delays.

6 Which Technology is Best for Wave Pipelining?

Some technologies are better than others for wave pipelining. Ideally, a technology should possess the following properties:

1. Many points of finely-controlled speed adjustment which have predictable effects
2. The same gate delay whether the output is rising or falling

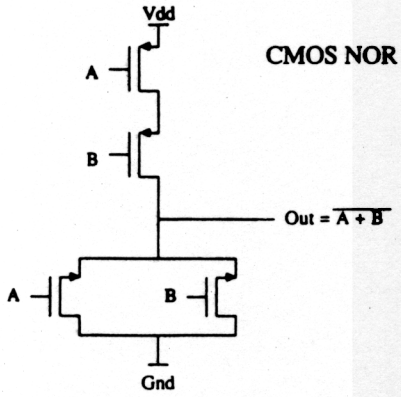


Figure 4: Static CMOS gates have undesirable pattern-dependent delays.

3. No variation in gate delay depending on input pattern
4. No variation in gate delay depending on previous input patterns
5. Zero contribution to Δt_P due to process and temperature-induced variations within a chip
6. The standard goals of high noise immunity, low power, high density, and high speed.

Let us examine CMOS, ECL, CML, and superbuffered ECL with respect to these goals.

6.1 CMOS

Static CMOS (see Figure 4) is difficult to use because the gate delay depends on the input pattern. Any parallel transistor arrangement will have variable delay due to the varying amount of resistance. A series transistor arrangement also has variable delay because the capacitance that must be charged depends on which transistor in the series is the last to begin conducting.

Dynamic MOS techniques such as domino are even more difficult to use with the wave pipelining method because the gates must be precharged before each wave arrives.

6.2 ECL

A basic ECL gate is shown in Figure 5. The delay is controlled by adjusting the "tail current" by sizing the resistors proportionately. A gate can have multiple, independently-adjustable emitter-followers for different fanouts.

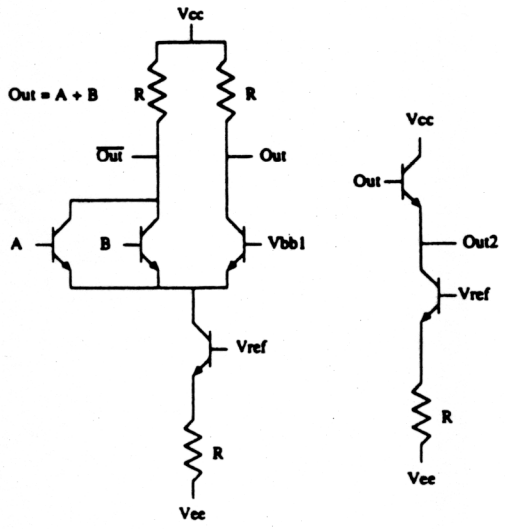


Figure 5: Standard ECL logic gate with emitter-follower

The logic section of the gate has fairly balanced rise/fall delay, but the emitter-follower section has rising delays which are much shorter than falling delays. This can be partially overcome by using a modified emitter-follower which slows down the rising delay (see Figure 6). Higher tail currents also reduce the difference in delays.

An example of a stacked ECL structure is shown in Figure 7. Stacked structures are less useful for wave pipelining because their speed depends on the input patterns. The intermediate node can be charged to different voltages just prior to switching, thus causing variation in delay. For instance, suppose the voltage were higher on the reference side for both the upper and lower levels. Now suppose that the A input of the upper level rose momentarily. This would charge the intermediate node X to a higher voltage. This would be additional charge to be drained when the lower level switches, thus slowing down the gate.

6.3 CML

One way to eliminate the unbalanced rise/fall delay in ECL is not to use emitter-followers. Instead, the output of one ECL logic stage is directly connected to the next in a logic family called CML. In non-wave-pipelined designs, CML is slower than ECL because both rising

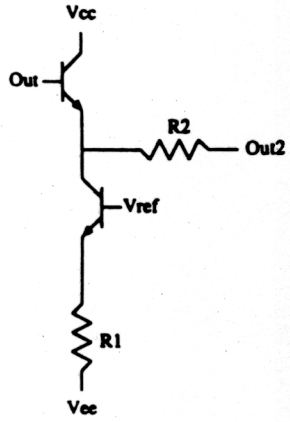


Figure 6: Modified emitter-follower with improved balance between rise and fall delays

and falling transitions are through limited current devices. In contrast, ECL has a fast rising delay that can be used effectively with inverting logic. However, a faster rising delay is not advantageous for wave pipelining.

Since there are no emitter-followers, one cannot independently adjust each fanout except by using additional gates as buffers. A buffer gate costs 3 transistors and 3 resistors compared to an emitter-follower of 2 transistors and 1 resistor. In addition, no wired-OR is possible.

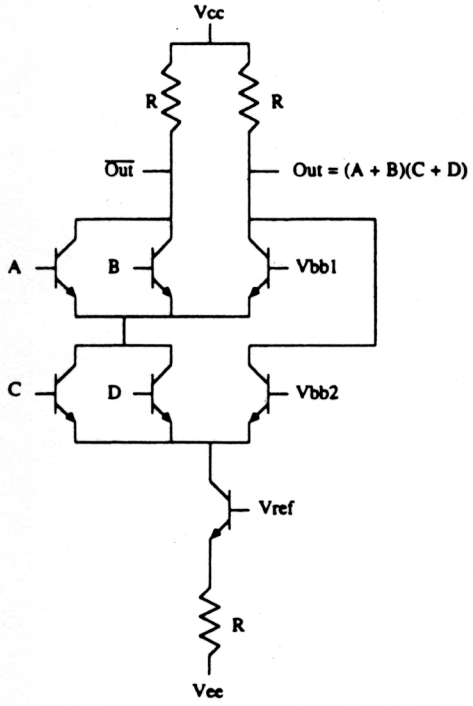
6.4 Super-buffered ECL

There are high-performance ECL buffers which have balanced delay. By using dynamic techniques, such circuits have fast transitions in both directions, i.e., faster than the static current source can achieve. A circuit presented by Coy et al [Coy88] is repeated in Figure 8 in a modified form.

The advantage of super-buffered ECL is that it is very fast and has balanced delay with less power consumption (since the static current sources can be smaller). Unfortunately, each buffer costs 4 transistors, 2 resistors, and 1 capacitor compared to an emitter-follower of 2 transistors and 1 resistor.

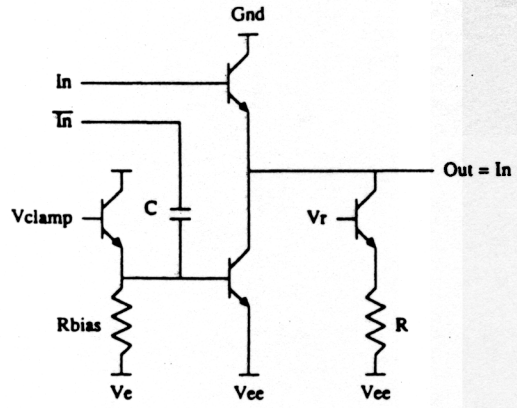
6.5 General Remarks on the Current-Steering Technologies

In ECL, CML, and super-buffered ECL, the use of stacked structures increases logic density but hurts substantially the possibilities for wave pipelining. One should consider using a



Logic Section of Stacked Structure

Figure 7: Stacked ECL logic structures have pattern-dependent delays due to charging of the intermediate node.



The capacitor C boosts the base voltage of the pull-down transistor to provide a dynamic pull-down current on Out.

Figure 8: Super-buffer for ECL has balanced rise/fall delays.

reduced supply voltage to conserve power if stacked structures are not used.

The effect of process and temperature-induced variations within one chip on Δt_P should be similar for all the current-steering technologies.

Careful attention should be paid to the power and ground networks during design. Voltage drops should be small; otherwise, the result would be equivalent to shifting the reference voltages. This would cause a difference in gate speed depending on whether the input is rising or falling.

6.6 Conclusions and Tradeoffs

Static and dynamic CMOS are not well suited for wave pipelining. We are not using these technologies in the Wave Pipelining Project.

Bipolar ECL, CML, and super-buffered ECL offer a range of tradeoffs between area, absolute speed, wave-pipelining potential, power, and number of independent adjustments:

- ECL has medium area density but has unbalanced rise/fall delay and fairly high power consumption.
- CML has high area density and balanced delays with medium power consumption. However, it offers fewer points of adjustment than ECL unless additional buffer gates are used.
- Super-buffered ECL is very fast and has balanced delay, medium power consumption, and the same number of adjustments as ECL. However, buffers consume more area than emitter-followers.

To help decide among the various current-switching technologies, one needs to consider the fanout in the circuit. For high fanout circuits, the need to adjust each fanout independently would favor ECL since it has small buffers.

7 Summary and Future Directions

Wave pipelining can potentially increase a system's clock frequency by 2 to 3 times without using additional pipeline registers. To maximize clock rate, we must minimize the variation in path length.

Some technologies are better than others for wave pipelining. Static CMOS has substantial pattern-dependent delay variations which reduces its usefulness. CML and super-buffered ECL have good delay properties. ECL has the advantage of small, individually-adjustable

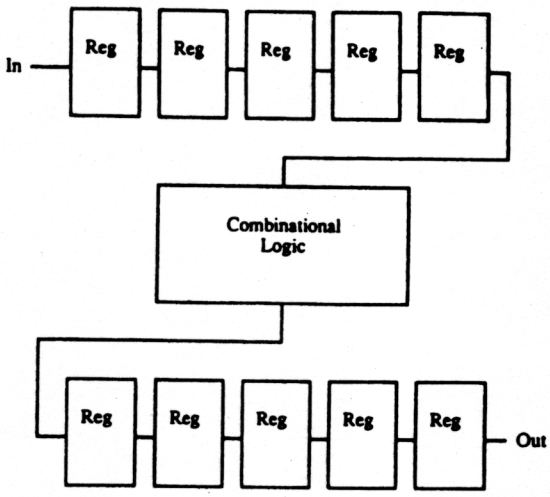


Figure 9: Test Structure for Demonstrating Wave Pipelining

emitter-followers with the disadvantage of unequal rise/fall delays.

We have developed new algorithms to automatically balance delays in combinational logic circuits. An optimal fine-tuning algorithm has been developed, implemented, and tested on small circuits. It uses a linear program to minimize power and balance the delays. The rough tuning algorithm is now being implemented.

The more complex case where the rising delay does not equal falling delay is a difficult problem for both rough and fine tuning. Well-balanced solutions might not even exist. The best that our methods can do is to guarantee a reasonable bound on the difference δ in propagation time under nominal conditions. In practice, δ is usually small compared to t_P, so an efficient wave-pipelined implementation can still be designed.

Next, we plan to design a test chip of the type shown in Figure 9. The chip is clocked very fast in order to develop pipeline waves in the combinational logic. We have selected CML as the technology for implementing our sample chip which will be a 32-bit multiplier with possibly some additional datapath logic.

8 Appendix 1: Deriving the Minimum Clock Period Relation

The maximum pipeline rate is affected by technological parameters. Here, we derive relations for minimal clock period as a function of variations in path length, clock skew, rise/fall times, and the setup and hold time of the storage elements. These relations are extensions upon those stated in [Cotten69], [Fawcett75], [Ekroot87], and [Kogge81] in order to include clock skew and rise/fall times.

8.1 Definitions

- t_{SH} = set-up plus hold time for edge-triggered registers.
  (For latches, t_{SH} = length of transparent period plus hold time.)
- t_P = propagation time of the longest path in the combinational logic
- t_E = worst-case shortest path length
- Δt_P = max difference in path length over worst-case design, process, and environment $= t_P - t_E$
- The clock transition to start wave 1 is at time 0.
  Wave 1 is captured at the ending storage element by another clock transition at time t_C.
- ΔC_E = worst-case early clock skew
- ΔC_L = worst-case late clock skew
- ΔC = worst-case clock skew = $\Delta C_E + \Delta C_L$
- t_{CP} = clock period
- t_{RF} = worst-case rise or fall time (10% to 90% voltage swing) at the last logic stage

8.2 Analysis with zero rise/fall time

For simplicity, we begin with the case which assumes zero rise/fall times.

1. The earliest that the signal wave could arrive at the ending storage elements is
   $t_P - \Delta t_P - \Delta C_E$
2. The latest that the signal wave could arrive is
   $t_P + \Delta C_L$
3. The earliest that the clock pulse could arrive at the ending storage elements is
   $t_C - \Delta C_E$
4. The latest that the clock pulse could arrive is
   $t_C + \Delta C_L$
5. Since the latest possible arriving signal of a wave must be captured by the earliest pos-

sible clock pulse
$t_C - \Delta C_E > t_P + \Delta C_L + t_{SH}$

6. Since the earliest arriving signal must be captured by the latest clock pulse, i.e., the earliest signal of the next clock period must not interfere
$t_P - \Delta t_P - \Delta C_E + t_{CP} > t_C + \Delta C_L$
7. Using some algebra, we get the following relation for clock period:
$t_{CP} > \Delta t_P + 2 * \Delta C + t_{SH}$

8.3 Analysis including last-stage rise/fall delay

Now we include a non-zero rise/fall time at the last stage of combinational logic before the ending storage element.

1. Inequality 8.2.6 is then changed:
The earliest signal must be captured by the latest clock pulse, i.e., the earliest signal of the next clock period must not interfere
$t_P - \Delta t_P - \Delta C_E + t_{CP} - t_{RF} > t_C + \Delta C_L$
2. This changes the inequality for t_{CP} to
$t_{CP} > \Delta t_P + 2 * \Delta C + t_{SH} + t_{RF}$

8.4 Other Bounds on t_{CP}

In addition to inequality 8.3.2, another bound can be derived to ensure that waves are not mixed together at signal points within the logic.

1. **Definitions**
   - t_X is the propagation time of the longest path from the source to a signal X which is not an input of an ending storage element.
   - t_{RF} is the worst-case rise/fall time at X.
   - t_{MS} is the minimum time that X must be stable for the next stage of logic to operate correctly.
   - Δt_X is the worst-case variation in path length from the source to X and is always less than or equal to Δt_P.
2. The latest that a wave could arrive at X is
$t_X + \Delta C_L$
3. The earliest that the next wave could begin to arrive at X is
$t_X - \Delta t_X - \Delta C_E - t_{RF} + t_{CP}$
4. To avoid interference, the next wave must arrive at least t_{MS} later
$t_X + \Delta C_L + t_{MS} < t_X - \Delta t_X - \Delta C_E - t_{RF} + t_{CP}$
5. This can be simplified to
$t_{CP} > \Delta t_X + \Delta C + t_{MS} + t_{RF}$
6. The power bounds $P_{MIN}[i]$ and $P_{MAX}[i]$ can generally be set so that t_{RF} is small enough to make this bound less stringent than inequality 8.3.2. In any case, the goal of minimizing Δt_P includes minimizing Δt_X for all signals X.

9 Acknowledgements

Prof. Mark Horowitz was very helpful in pointing out pitfalls and in particular analyzing technology considerations. He thought of the modified ECL emitter-follower and suggested the use of super-buffered ECL. Michael Saunders and Walter Murray were generous in their advice about practical considerations of solving optimization problems. Timothy Pinkston reviewed drafts of this paper.

10 Bibliography

Anderson67 S. Anderson, J. Earle, R. Goldschmidt, and D. Powers. "The IBM System/360 Model 91 Floating Point Execution Unit." Jan. 1967, IBM Journal of Research and Development, pp. 34-53.

Cotten69 L. Cotten. "Maximum Rate Pipelined Systems." 1969 AFIPS Proceedings of Spring Joint Computer Conference, pp. 581-586.

Coy88 B. Coy, A. Mai, and R. Yuen. "A 13,000 Gate 3 Layer Metal Bipolar Gate Array." 1988 Custom Integrated Circuits Conference, pp. 20.1.1 - 20.1.3.

Ekroot87 B. Ekroot. Optimization of Pipelined Processors by Insertion of Combinational Logic Delay. Sept. 1987, Ph.D. Dissertation, Electrical Engineering, Stanford University, Stanford, CA.

Fawcett75 B. Fawcett. Maximal Clocking Rates for Pipelined Digital Systems. Dec. 1975, Report R-706 from Coordinated Science Laboratory, University of Illinois, Urbana, IL.

Kogge81 P. Kogge. The Architecture of Pipelined Computers. 1981, McGraw-Hill, New York, NY.

Lin88 Q. Lin and P. Xia. "The Design and Implementation of a Very Fast Experimental Pipelining Computer." 1988, Journal of Computer Science and Technology (Beijing), Vol. 3, No. 1, pp. 1-6.

Marple86 D. Marple. Performance Optimization of Digital VLSI Circuits. Sept. 1986, Ph.D. Dissertation, Electrical Engineering, Stanford University, Stanford, CA.

Murtagh87 B. Murtagh and M. Saunders. MINOS 5.1 USER'S GUIDE. Jan. 1987, Technical Report SOL 83-20R from the Systems Optimization Laboratory, Operations Research, Stanford University, Stanford, CA.

Wong89 D. Wong, G. De Micheli, and M. Flynn. "Inserting Active Delay Elements to Achieve Wave Pipelining." 1989, Technical Report CSL-TR-89, Electrical Engineering, Stanford University, Stanford, CA.