Maximum Rate Single-Phase Clocking of a Closed Pipeline including Wave Pipelining, Stoppability, and Startability

Chuan-Hua Chang, Edward S. Davidson, Kareem A. Sakallah

Abstract

Aggressive design using level-sensitive latches and wave pipelining has been proposed to meet the increasing need for higher performance digital systems. The optimal clocking problem for such designs has been formulated using an accurate timing model. However, this problem has been difficult to solve because of its nonconvex solution space. The best algorithms to date employ linear programs to solve an overconstrained case that has a convex solution space, yielding suboptimal solutions to the general problem. A new efficient (cubic complexity) algorithm, Gpipe, exploits the geometric characteristics of the full nonconvex solution space to determine the maximum single-phase clocking rate for a closed pipeline with a specified degree of wave pipelining.

Introducing or increasing wave pipelining by permanently enabling some latches is also investigated. Sufficient conditions have been found to identify which latches can be removed in this fashion so as to guarantee no decrease and permit a possible increase in the clock rate. Although increasing the degree of wave pipelining can result in faster clocking, wave pipelining is often avoided in design due to difficulties in stopping and restarting the pipeline under stall conditions without losing data, or in reduced rate testing of the circuit. To solve this problem, which has not previously been addressed, we present conditions and implementation methods that insure the stoppability and restartability of a wave pipeline.

The authors are with the Advanced Computer Architecture Laboratory, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109-2122.

This work was supported in part by National Science Foundation Grant NSF MIP 9014058 and by a gift from the Hewlett-Packard Corporation.
1 Overview


Level-sensitive latches are used as synchronizers to achieve higher clock rates in synchronous systems. During the active (latch enabled) region of the clock period, level-sensitive latches permit an input signal to propagate through the latch to the output. This property of latches allows “time borrowing” [9] which tends to average the propagation delays of successive stages, resulting in a faster clock. However, timing verification and optimization for latch-controlled systems becomes more complicated when time borrowing is allowed. Various methods have been developed to analyze latch timing behavior and to optimize the clock schedule of synchronous circuits [7][8][9]. However, none of them captures the exact latch timing behavior and guarantees to find the truly optimal clocking rate.

Recent work at the University of Michigan has developed a general timing model which accurately captures the timing behavior of latch-based pipelines and their full nonconvex feasible solution space for single-phase clocking [10][11][12][13]. Because finding an optimum solution in a nonconvex space is a difficult problem in general, previous work has overconstrained the solution space to form a convex region and solved the general synchronous circuit and closed (circular) pipeline maximum rate clocking problems by using a linear programming approach. This timing model has also been successfully applied by Szymanski to solve the problems of timing verification and optimal clocking for general latch-based synchronous circuits by using an iterative relaxation algorithm [14] and a linear programming approach [15], respectively. The optimal clocking problem solved by Szymanski also creates a convex feasible region by overconstraining
the problem. These overconstrained problem formulations can achieve only a limited form of time borrowing since signals experiencing the minimum propagation delay will always arrive at a latch during the passive (latch disabled) region of the clock period. They have a number of attractive properties such as the applicability of linear programming, and ease of stopping and restarting the pipeline. However, the maximum clock rate of the general formulation may not be found as it may be rendered infeasible in the overconstrained problem.

The work reported here is developed from the latch-based timing constraints for a closed pipeline found in [13]. Some of the early results are summarized in [16]. The goal of this work is to solve the nonconvex maximum rate single-phase clocking problem using the general problem formulation. The relationships among the three types of timing constraints are studied. We show how to use these relationships to remove redundant constraints and subsequently speed up the process of finding the maximum rate single-phase clock. This analysis is also useful for dealing with a variety of important synchronous circuit design problems, such as wave pipelining, in a more efficient and rigorous way.

A new program, Gpipe, has been developed using an algorithm that exploits specific geometric relationships among these constraints to find the maximum rate clock. Exploiting such problem-specific knowledge results in a very efficient algorithm for the general problem despite the non-convexity of its feasible solution space. Significantly, its cubic worst-case complexity is less than that of the linear programming solution for the overconstrained problem.

Furthermore, by analyzing the information produced by Gpipe, the effect of “removing” latches on the maximum clock rate of a closed pipeline can be determined. The clock cycle time of the pipeline can sometimes be reduced by simply setting the clock input of some particular latches to the permanently enabled level, thereby letting the signals just flow through these latches unimpeded, and effectively merging several stages into one. Wave pipelining is introduced into each newly merged pipeline stage to maintain the total concurrency of execution.

This result leads to questions of whether redundant latches can easily be identified and removed, and how a pipeline can be stalled and restarted when wave pipelining is introduced in
this way within a pipeline stage. The ability to stall and restart are desired features in most pipeline systems and are essential for testing a system in single-step mode. However, no previous research on wave pipelining has addressed these crucial problems. A key contribution of this paper is that we analyze the problem of stopping and restarting of wave pipelines and provide a solution using redundant latches.

In this paper, section 2 is a brief review and some extension of the work in [13]. Section 3 is a detailed description and complexity analysis of the underlying algorithm of Gpipe. Section 4 provides a theoretical basis for the study of removing latches from a pipeline circuit. Section 5 illustrates the use of the Gpipe program and presents some examples of removing latches. Section 6 addresses the questions of stopping and starting a closed pipeline with a single-phase clock and level sensitive latches.

2 Pipeline and Clocking Model

The pipeline and clocking model is based on the closed pipeline model described in [13]. The pipeline is modeled as a single closed loop with \( N \) stages separated by \( N \) synchronizers, which are chosen to be latches in this paper (Fig. 1). Latches are harder to model, but impose less clocking overhead than flip-flops. Stage 0 may be an actual physical stage, e.g. the closed pipeline may model the closed path between a CPU and memory, or may be used to characterize the input-output constraints of an open pipeline, e.g. a functional unit. Stages and their output latches are numbered consecutively from 0 to \( N-1 \) and are characterized by the following parameters:

- \( S_i \) and \( H_i \): nonnegative setup and hold time of latch \( i \) relative to the latching edge of the clock (see Fig. 2).
- \( \Delta_i \) and \( \delta_i \): maximum and minimum propagation delay from the input of latch \( i-1 \) to the input of latch \( i \). Note that \( 0 \leq \delta_i \leq \Delta_i \). Furthermore, each “latch” defines the end of a physical pipeline stage and is actually a set of \( m \) latches that temporarily store the \( m \) bits of a wave of data. Con-

---

1 A more complex model could describe the correlation of these values as functions of the operating conditions (temperature, power, ...) and manufacturing tolerances. The model above assumes the worst case.
sequently, $\delta_i$ should be viewed as the minimum delay on the minimum path under the fastest operating conditions and manufacturing tolerance. Similarly $\Delta_i$ is the maximum of these.

The single-phase clock waveform seen by every latch is illustrated in Fig. 2. The $t = 0$ origin used for each latch is set in a local time coordinate system relative to its own clock. The clock period, $T_c$, is divided into an active (latch enabled) region of length $T_1$ and a passive (latch disabled) region of length $T_c - T_1$. The onset of the active region is called the enabling edge of the clock; the end of the active region is called the latching edge. When it is necessary to relate events at several latches, a phase shift operator ($\varepsilon$) translates between local time coordinate systems.\(^2\)

Four events measured in latch $i$ local time define a data wavefront arriving at and departing from the input to latch $i$: early arrival $a_i$, late arrival $A_i$, early departure $d_i$, and late departure $D_i$. The previous data wave (Old Data) remains valid until the early arrival of the wave associated with this clock period. This wave becomes valid (New Data) upon its late arrival. The Enabling Edge of the clock enables departure of this new wave into the next stage. The timing constraints for each latch $i = 0, \ldots, N - 1$ are:

\[ t_i = T_c \cdot \frac{i}{N - 1}, \ldots, N - 1 \]

---

\(^2\) A more complex model could incorporate clock skew and even the correlation of clock skews on successive latches. This would add or subtract terms to each $T_c$ and $T_1$ to tighten the constraints below. For simplicity of illustration, we assume zero clock skew.
Pulse Width Constraints: clock pulse limitations posed by latch and clock generation circuits. Constants $w_H$ and $w_L$ are the minimum length requirements for the clock high and low times, respectively.

\[
T_1 \geq w_H \tag{1}
\]

\[
T_c - T_1 \geq w_L \tag{2}
\]

Latching Constraints: early arrival of the new wave must not violate the hold time ($H_i$) for the old data wave and late arrival must satisfy the setup time ($S_i$) for the new wave.

\[
a_i \geq H_i \tag{3}
\]

\[
A_i \leq T_c - S_i \tag{4}
\]

Synchronization Equations: data wave cannot depart a level sensitive latch before the enabling edge of the clock $T_c - T_1$, nor can the early (late) departure occur before the early (late) arrival.

\[
d_i = \max(a_i, T_c - T_1) \tag{5}
\]

\[
D_i = \max(A_i, T_c - T_1) \tag{6}
\]

According to three cases of data wave arrival, we can distinguish three latch operating modes:

i) A latch is synchronizing if $A_i \leq T_c - T_1$, and hence $d_i = D_i = T_c - T_1$, i.e. the early and late wavefronts depart the latch at the same time.
ii) A latch is transparent if \( a_i \geq T_c - T_1 \), and hence \( d_i = a_i \) and \( D_i = A_i \), i.e. the early and late wavefronts pass through the latch unimpeded by the clock at the latch.

iii) Otherwise, \( a_i < T_c - T_1 < A_i \) (see Fig. 2) and the latch is partially-synchronizing, in which case \( d_i = T_c - T_1 \) and \( D_i = A_i \), i.e. the early wavefront is delayed, but the late wavefront passes through the latch unimpeded.

The undefined interval that defines a data wavefront is unaffected by a transparent latch, is reduced somewhat by a partially-synchronizing latch, and is reduced to zero by a synchronizing latch.

**Propagation Equations:** define latch \( i \) arrival in terms of latch \( i-1 \) departure. Since the pipeline is a closed loop, stage subscripts are implicitly interpreted as modulo \( N \). The translation of \( d_{i-1} \) and \( D_{i-1} \) (in latch \( i-1 \) local time) to latch \( i \) local time is accomplished by subtracting a phase shift operator, \( \varepsilon_{i-1,i} \). The phase shift operator is set equal to the time from the beginning of a clock period at latch \( i \), which is associated with the departure of some wave, to the beginning of the clock period at latch \( i+1 \) that is associated with the arrival of that same wave.

\[
\begin{align*}
    a_i &= d_{i-1} + \delta_i - \varepsilon_{i-1,i} \\
    A_i &= D_{i-1} + \Delta_i - \varepsilon_{i-1,i}
\end{align*}
\]  

(7) 

(8)

For a single phase clock, \( \varepsilon_{i-1,i} = (1 + \nu_i)T_c \) where \( \nu_i \) is a nonnegative integer. The parameter \( \nu_i \) is defined to be the degree of wave pipelining in stage \( i \). When \( \nu_i = 0 \), there is no wave pipelining in stage \( i \) and data moves from latch \( i-1 \) to latch \( i \) each \( T_c \). In general there are \( 1 + \nu_i \) waves in stage \( i \) and \( 1 + \nu_i \) clock periods are required for a wave to travel from latch \( i-1 \) to latch \( i \).

The concurrency, \( K \), of a closed pipeline is the total number of data waves traveling around the loop, where \( K = N + \sum_{i=0}^{N-1} \nu_i \). \( K \) may be thought of as the number of virtual stages in a closed pipeline, where \( N \) is the number of latches, or physical stages.

**2.1 Derivation of Optimal Cycle Time Constraints**

We can now adapt and summarize the derivation in [13] to yield a necessary and sufficient set of constraints, (9)-(13), that defines all feasible \( (T_j, T_c) \) solutions, and hence minimum \( T_c \), for sin-
Single-phase clocking of a given closed pipeline with a specified degree of wave pipelining in each stage, i.e. given \( N, \Delta_i, \delta_i, \nu_i, S_i, H_i, w_H \) and \( w_L \).

**Pulse Width Constraints** (two from (1) and (2)):  
\[
T_1 \geq w_H \\
T_c - T_1 \geq w_L
\]  

(9) \hspace{2cm} (10)

**Long Path Constraints** \((N^2 + 1)\) from recursive substitution of (8) and (6) in (4)):  
\[
T_c \geq \left( \sum_{j=0}^{N-1} \Delta_j \right) / \left( N + \sum_{j=0}^{N-1} \nu_j \right) \\
\left( 1 + l + \sum_{j=0}^{i} \nu_j \right) T_c + T_1 \geq \left[ \left( \sum_{j=0}^{i} \Delta_j \right) + S_i \right]
\]

for \( i = 0, \ldots, N-1 \) and \( l = 0, \ldots, N-1 \) \hspace{2cm} (11) \hspace{2cm} (12)

The above linear Long Path constraints are derived by

1. Starting from a specific latching constraint, (4), calculate the late arrival time, \( A_j \), by recursively applying (8) and (6) resulting in an expression for \( A_j \) as a function of \( A_{j-1} \), then \( A_{j-2} \) etc. Stop this process when \( A_j \) is expressed as a function of \( A_i \). At this point, \( A_j \) has the form \( \max \{ A_i + f, \ldots \} \) where no \( A \) or \( D \) variables appear in \( f \) or among the \( N \) other arguments.\(^3\)

2. Thus \( A_j \geq A_i + f \) and \( A_i \geq \) argument \( j \) where \( j = 1, \ldots, N \). We can use these \( N+1 \) inequalities in place of setting \( A_j \) equal to the \( \max \) function because in minimizing \( T_c \), \( A_j \) will be driven to satisfy one of these inequalities by equality.

3. Long path constraint (11) is equivalent to \( f \leq 0 \). It follows that \( A_i + f \) will not be the argument that defines \( A_j \). Hence we need not use the \( T_c - S_i \geq A_i + f \) constraint.

4. The remaining \( N \) inequalities are replaced by \( T_c - S_i \geq \) argument \( j \) for \( j = 1, \ldots, N \), resulting in long path constraint (12).

**Short Path Constraints** \((N^2)\) from recursive substitution of (7) and (5) in (3)):
for each $i = 0, \ldots, N-1$, there must be at least one $l = 0, \ldots, N-1$ for which this inequality holds.

The above linear Short Path constraints are derived from (3) in a similar fashion by recursively applying (7) and (5) to yield $H_i \leq a_i = \max \{ a_i + f', \ldots \}$. Again the first argument does not define $a_i$. Furthermore, since $\delta_j \leq \Delta_j$, $f' \leq f \leq 0$ and thus $f' \leq 0$ is a redundant constraint. Therefore this inequality reduces to $H_i \leq \max \{ \ldots \}$, i.e. at least one of the following must be satisfied: $H_i \leq \text{argument } j$ for $j = 1, \ldots, N$, resulting in short path constraint (13). Furthermore, minimizing $T_c$ will force $a_i$ to equal the maximum of the $N$ other arguments in the $\max$ expression.

Given the total maximum propagation delay of a closed pipeline circuit and a desired value of $K$, the goal of latch placement and clock design is to satisfy (11) by equality. This may be possible with latches, and occurs when every latch is partially-synchronizing or transparent, i.e. $D_i = A_i$ for all $i$, in the optimal solution.

### 2.2 Long Path Constraints Reduction

Note that some of the $N^2$ constraints in (12) can be discarded. The coefficients of $T_c$ are integers ranging from 1 to $K$, where $K = N + \sum_{j=0}^{N-1} v_j$. For each distinct $T_c$ coefficient we need only use the constraint with the largest right hand side. The number of constraints in this reduced long path set is no more than $K$. More formally, let

$$B_k = \{ (i,l) \mid 1 + l + \sum_{j=i-l}^{i} v_j = k \} \quad l = 0, \ldots, N-1$$

and let

$$R_k = \max_{(i,l) \in B_k} \left\{ \sum_{j=i-l}^{i} \Delta_j + S_i \right\}$$

the reduced long path constraint set is then

\[
\left( l + \sum_{j=i-l}^{i} v_j \right) T_c + T_1 \leq \left( \sum_{j=i-l}^{i} \delta_j \right) - H_i
\]

for each $i = 0, \ldots, N-1$.
2.3 Problem Statement

Now the formal definition of our problem is to

\[
\text{Minimize } T_c;
\]

subject to (9), (10), (11), (13), and (14).

The Pulse Width and Long Path constraints confine the solution space to convex regions as in Fig. 3(a) and Fig. 3(b) respectively. However, the \( N \) Short Path constraint sets confine the solution space to a nonconvex region as in Fig. 3(c).

\[
kT_c + T_1 \geq R_k \quad \text{for each } k = 1, 2, \ldots, K
\]

\[
\text{for which } B_k \neq \emptyset
\]

Fig. 3. The feasible regions of (a) Pulse Width constraints (b) Long Path constraints (c) Short Path constraints

3 The Gpipe Algorithm

Previous work in [12] used a general linear program (LP) solver to find the minimum \( T_c \), for the case that all latching edges occur simultaneously; \( T_1 \) can vary from stage to stage (coincident multiphase clocking). It also produced a closed form solution for the restricted case that (13) must be satisfied for \( l = 0 \). This commonly used restriction overconstrains the problem and leads to suboptimal solutions. The general single-phase clocking problem, however, is not an LP due to the nonconvexity of the feasible region, caused by (13) without restrictions. Nevertheless, by using knowledge of the specific geometric relationships among the constraints above we are able to analyze and solve this problem in cubic polynomial time using the Gpipe algorithm. Gpipe is
the first known unrestricted single-phase clock optimization algorithm for the closed pipeline model.

The nature of the Gpipe algorithm is illustrated by considering the 4-stage closed pipeline with no wave pipelining in any stage, as shown in Table 1. The geometric relationships of the constraints is shown in Fig. 4. The optimum \((T_i, T_c)\) is \((6, 14)\).

<table>
<thead>
<tr>
<th>(i)</th>
<th>(\delta_i)</th>
<th>(\Delta_i)</th>
<th>(v_i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>16.0</td>
<td>18.0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>8.0</td>
<td>10.0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>12.0</td>
<td>14.0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>8.0</td>
<td>10.0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 1: Timing parameters for the example.
\((w_H = w_L = 2, H_i = S_i = 2)\)

Fig. 4. A geometric view of the example constraints.

Because the pulse width constraints (9) and (10) and the \(K+1\) (or fewer) long path constraints (11) and (14) do form a convex set, the minimum \(T_i\) satisfying these constraints is always found on the boundary. Therefore we first compute the boundary contour of the convex region (Fig. 6) without considering constraints (13).

A subset of the \(K+1\) Long Path linear equations forms a nonincreasing broken line contour in the positive \((T_i, T_c)\) quarter plane, i.e. \(T_i\) decreases monotonically to a constant as \(T_c\) increases, since the slopes of the long path linear equations from (14) with \(k = 1, \ldots, K\) are all negative and
monotonically increasing to 0 for (11), namely \(-1, -1/2, \ldots, -1/K, 0\). This long path contour is further confined to a finite region by the two pulse width constraints, with slopes of \(\infty\) and 1, respectively. This finite region is defined to be the “confined long path contour”.

Normally, the two pulse width constraints (9) and (10) intersect outside (below) the long path feasible region and the long path contour is confined between (9) on the left and (10) on the right (Fig. 6 (a)). However, a degenerate situation (Fig. 6 (b)) occurs when (9) and (10) intersect within the long path feasible region or on the boundary of this region. In this case, the “confined long path contour” is defined to be the single point intersection of (9) and (10) and, if any feasible solution exists, the optimum solution is at this point. In either case, \(Gpipe\) seeks the rightmost point of the confined long path contour that satisfies the short path constraints (13).

Consider constraints (13). Although the combination of all \(N\) sets of short path constraints forms a complex nonconvex feasible space, the solution space of each set of constraints with the same value of \(i\) has a very regular shape (The complement of a set’s solution space is a convex hull, Fig. 3(c)). This fact can be used to find the optimal solution by applying each constraint set in turn to the confined long path contour. The short path boundary equations for each set also form a piece-wise decreasing contour and the feasible solution must lie below this contour. Because of this shape, each short path contour is used to cut off portions of the confined long path contour iteratively until the rightmost feasible point of the confined long path contour is found (Fig. 8). The shape of short path contour is similar to the long path contour, however the first line segment of the contour is vertical when \(v_i = 0\), and the last line segment of the contour is finite in length.

Fig. 6. Relationship between long path and pulse width constraints in a closed pipeline.
(because of its nonzero slope it ends at the $T_1$ axis). Fig. 4 shows a confined long path contour, how it is cut by the four short path contours, and the resulting optimal solution point.

Thus by exploiting knowledge of the specific shapes of the long and short path and pulse width constraint boundary contours, $Gpipe$ is able to employ an efficient algebraic method to find the minimum value for $T_c$.

### 3.1 Detailed Procedure for $Gpipe$

1. Input the numerical values for all variables except $T_c$ and $T_1$, compute the coefficients and set up the constraints: 2 pulse width constraints, $N^2 + 1$ long path constraints and reduction to $\leq K$ constraints, and $N^2$ short path constraints.

2. Generate contours (1 long path and $N$ short path contours) as ordered lists of line segments:
   
   i) Sort the linear constraint equations according to the absolute values of their slopes.  
   ii) Clear an ordered list ($L$) which is used to store the line segments of a contour on the fly.  
   iii) Add the first constraint line to $L$.  
   iv) Intersect all remaining lines with the last entry of $L$.  
   v) Mark the line which intersects with the last entry of $L$ at the highest $T_c$ value.  
   vi) Discard all lines between the last entry of $L$ and the marked line.  
   vii) Append the marked line along with the intersection point to $L$.  
   viii) If generating a short path contour, constrain the short path contour to lie above the $T_1$ axis. In Fig. 7, line 3 has the highest intersection with line 1, line 2 is then discarded, and line 4 then has the highest next intersection with line 3.

![Diagram](image)  

**Fig. 7.** Formation of a contour.
3. Generate the confined long path contour by intersecting the pulse width constraints with the long path contour. If (9) intersects the contour left of (10), the resulting contour is between the two intersection points. If, however, (10) intersects the contour left of (9), i.e. (9) and (10) intersect within the long path feasible region, then use the single point where (9) and (10) intersect as the confined long path contour.

4. Reduce the contour from step 3 using the short path contours: 
   i) Take the last entry in the ordered list of the confined long path contour (the segment with the lowest $T_c$ values). 
   ii) Intersect the segment with all the line segments of the short path contour for each stage in turn. 
   iii) If any part of the segment is on the feasible side of all short path contours, the optimal solution is the rightmost point of that part (or the entire feasible portion if the segment has slope 0) and the algorithm halts. If not, then delete this segment entry and repeat i) until the algorithm halts or no entry remains in the confined long path contour. If no entry remains, then there is no feasible solution. Note that a short path contour may intersect a long path contour segment in several different ways (Fig. 8).

5. Output the optimum (minimum $T_c$) solution, the contour information (for graphical analysis), and list each short path constraint with a 'yes' or 'no' status to represent whether it is satisfied by the optimum solution (for later analysis).

![Fig. 8. Relationships between a long path line segment (L) and a short path contour (S) for stage i.](image-url)
3.2 Complexity of Gpipe

In step 1, all the boundary equations are obtained directly from a specified pipeline structure and desired degree of wave pipelining per stage. By computing the constraint coefficients in a reasonable order, we can generate each of the $2N^2 + 3$ original equations in constant time, including discarding redundant long path constraints as they are generated. Step 1 time complexity is thus $O(N^2)$. In step 2, $N+1$ piecewise linear contours have to be found. In the worst case there are $K+1$ line segments for the long path contour and $N$ line segments for each short path contour. Each of these line segments can be determined in $O(N)$ ($O(K)$ for long path) time. So the worst time to find a short path contour is $O(N^2)$ ($O(K^2)$ for long path). Considering all the contours, the time complexity in this step is $O(N^3+K^2)$. Step 3 only involves the global long path contour which contains at most $K+1$ line segments. The worst case complexity of this step is $O(K)$. Step 4 is the main part of this algorithm. There are at most $K+1$ global long path contour line segments and, in the worst case, each one is tested for satisfiability against all $N^2$ short path line segments. This is the case where no feasible solution can be found and no intersections occur between the long path contour and any short path contours. In this worst case situation, the time complexity of this step is $O(KN^2)$. Step 5 is $O(N^2)$. Combining the complexity of all steps, we conclude that the worst case complexity of our algebraic algorithm is $O(N^3+K^2+KN^2)$. For a pipeline without wave pipelining, where $K=N$, the complexity is $O(N^3)$.

3.3 Solution and Waveform Analysis

After obtaining the optimum solution from the Gpipe algorithm, we can further analyze the solution by using the geometric plot (Fig. 4) and the waveform plot (Fig. 5). From the resulting data waveforms, shown for each latch in the example of Table 1, we can see that data wavefronts flow through latch 0 without being delayed by the clock, i.e. the early arrival of the data wave ($a_{0}$) is after the enabling edge of the clock ($T_{c}-T_{1}$). Thus latch 0 is transparent. It can be “removed” without invalidating the current optimal solution by setting its clock constantly high. Furthermore, we see that latch 0 is tight on its setup constraint and latch 3 is tight on its hold constraint; These latches prevent the optimal clock of this example from achieving ($T_{c} = 13$), the
A latch whose late signal is tight on setup or whose early signal is tight on hold is called a critical latch. The examples in section 5 will show that if the lower bound for the clock cycle time (11) is not reached, then removing a critical latch, if it can be removed, may reduce the clock cycle time.

4 Removing Transparent Latches

For given values of $T_c$ and $T_1$ (which may or may not be optimal), when the early arrival time of a wave occurs after (or simultaneously with) the enabling edge of the clock $T_c - T_1$ for some latch, this latch is transparent and $d_i = a_i$. In this section, we consider “removing” a transparent latch under the requirement that the total number of waves in the pipeline, $K$, is preserved when the latch is removed. What will happen to the set of constraints and the current timing parameters $(T_c, T_1)$?

Three possible ways to “remove” a latch from a pipeline circuit are:

(a) Set the latch enable permanently active (the easiest case). In this case, the latch poses no setup and hold constraints. However, the “removed” latch can be reintroduced when needed (e.g. to stall the pipeline).

(b) Replace the latch with some delay element which has a delay equal to the latch delay. This model is mathematically equivalent to (a).

(c) Delete latch $i$ and subtract its delay from the $\delta_i + \delta_{i+1}$ and $\Delta_i + \Delta_{i+1}$ delays of the newly combined stage of the pipeline circuit (the hardest case).

While (a) and (b) will preserve the total delays of the original two stages in the newly combined stage, method (c) will reduce the total delays in this new stage. Method (c) is intuitively more practical but will change the original pipeline constraints in a more complex way than methods (a) and (b). In the following discussion, we focus on “removing” a latch by using method (a); we define “remove latch $r$” to be “set the latch $r$ enable input permanently active.”

Based on the constraints (9) (10) (11) (12) (13) and removal method (a) above, we notice that removing latch $r$ from the closed pipeline circuit will generate new constraints from the original
constraints by simply deleting the long and short path constraints for \( i=r \) and deleting the \( l=i-r-1 \) long and short path constraints for each other stage, \( i \neq r \), in the formulation of the problem. This reformulation is proved by the next two Lemmas.

**Lemma 1:** *Short Path Constraints* Consider removing latch \( r \) from a single-phase clocked closed pipeline which has \( N \) sets of \( N \) short path constraints (13). The new short path constraint sets will be formed by deleting the following short path constraints from the original short path constraint sets:

\[
(l + \sum_{j=i-l}^{i} v_j)T_c + T_1 \leq \left( \sum_{j=i-l}^{i} \delta_j \right) - H_i
\]

for \( i = r, \ l = 0, \ldots, N-1 \) and for \( i \neq r, \ l = i-r-1 \)

**Proof:** See Appendix.

**Lemma 2:** *Long Path Constraints* Consider removing latch \( r \) from a single-phase clocked circular pipeline which has \( N \) sets of \( N \) long path constraints (12) and one \( T_c \) lower bound constraint (11).

(i) The new long path constraint sets will be formed by deleting the following long path constraints from the original long path constraint sets:

\[
(1 + l + \sum_{j=i-l}^{i} v_j)T_c + T_1 \geq \left[ \left( \sum_{j=i-l}^{i} \Delta_j \right) + S_i \right]
\]

for \( i = r, \ l = 0, \ldots, N-1 \) and for \( i \neq r, \ l = i-r-1 \)

(ii) The \( T_c \) lower bound constraint of the new circuit will be the same as (11) of the original circuit.

**Proof:** See Appendix.

In summary, the necessary and sufficient set of constraints in terms of the original circuit parameters for the pipeline with latch \( r \) removed are as follows:

**Pulse Width Constraints** (two from (1) and (2)):
\[ T_1 \geq w_H \quad \text{and} \quad T_c - T_1 \geq w_L \]

**Long Path Constraints** (\((N-1)^2 + 1\) from Lemma 2):

\[
T_c \geq \left( \sum_{j=0}^{N-1} \Delta_j \right) / \left( N + \sum_{j=0}^{N-1} v_j \right)
\]

\[
\left( 1 + l + \sum_{j=l}^{i} v_j \right) T_c + T_1 \geq \left( \sum_{j=l}^{i} \Delta_j \right) + S_i
\]

for \(i \in \{0, \ldots, r-1, r+1, \ldots, N-1\}\)

and \(l \in \{0, \ldots, i-r-2, i-r, \ldots, N-1\}\)

**Short Path Constraints** (\((N-1)^2\) from Lemma 1):

\[
\left( 1 + \sum_{j=l}^{i} v_j \right) T_c + T_1 \leq \left( \sum_{j=l}^{i} \delta_j \right) - H_i
\]

for each \(i \in \{0, \ldots, r-1, r+1, \ldots, N-1\}\), there must be at least one

\(l \in \{0, \ldots, i-r-2, i-r, \ldots, N-1\}\) for which this inequality holds

Using the above two *Lemmas*, we can prove the following theorem.

**Theorem 1:** *In an N stage closed pipeline, if \((T_1, T_c)\) is a feasible single-phase clock solution under which the early data wave arrival time \(a_r\) at stage \(r\) occurs no sooner than the latch enabling edge \(T_c - T_1\), i.e. \(a_r = \max (a_r, T_c - T_1)\), then if we remove latch \(r\) in the circuit by setting its enable permanently active, \((T_1, T_c)\) will still be a feasible single-phase clock solution in the modified circuit.*

**Proof:** See Appendix.

A consequence of Theorem 1 is that when a nonsynchronizing latch is removed, the optimal solution for \(T_c\) will never increase, but may further decrease from its original optimal value. The deletion of some long path constraints which may have been critical in the original pipeline, i.e. in cases where \(A_r = T_c - S_r\), can sometimes allow the pipeline to run faster when latch \(r\) and consequently its setup constraint are removed.
5 Maximum Rate Clocking and the Effect of Removing Latches

In this section, we use the example of section 3.3 to explore the effects of removing latches and introducing wave pipelining.

5.1 Removing a Redundant Latch

The timing parameters for the modified pipeline with latch 0 removed in this fashion are shown in Table 2 without renumbering the remaining latches and the stages. Wave pipelining is introduced \( \nu_i = 1 \) to maintain four data waves in the pipeline with two waves as before between latches 3 and 1.

<table>
<thead>
<tr>
<th>( i )</th>
<th>( \delta_i )</th>
<th>( \Delta_i )</th>
<th>( \nu_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>24.0</td>
<td>28.0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>12.0</td>
<td>14.0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>8.0</td>
<td>10.0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2: Timing parameters for the modified pipeline with redundant latch 0 removed.

\( (w_H = w_L = 1.0, \ H_i = S_i = 2) \)

The constraints and the solution for this modified pipeline are illustrated in Fig. 9 and Fig. 10. We find that the minimum clock period is reduced from 14 to 13. This solution achieves the lower bound on \( T_c \) imposed by (11), and confirms our conjecture that removing redundant latches may let the pipeline run faster. In Fig. 5, latch 0 suffers a tight setup constraint which limits the clock speed for the entire pipeline. By setting the latch 0 enable input constantly high, this setup constraint is deleted and the Long Path contour is relaxed (lower); hence the clock speed is increased. The ghost image in Fig. 8 shows the waveform as it passes through the “removed” latch 0. It is clear that the waveform would violate the setup requirement if latch 0 were not removed. Notice that since the late signals arrive later and are not synchronized by most of the latches in Fig. 8 (vs. Fig. 5), the “changing” portions of the signals become larger in Fig. 8.

One effect of introducing wave pipelining is that the size of the feasible region for \( (T_i, T_c) \) is reduced from an infinite region to a finite region. When there is no wave pipelining in any pipeline stage, the leftmost segment of every short path contour is a vertical ray extending to infinite \( T_c \). If
the $T_i$ values of these vertical lines all exceed $w_{li}$, then there is an infinite feasible region and $T_c$ can be made arbitrarily long, e.g. for testing purposes or pipeline stalls. When $\nu_i \geq 1$ for stage $i$ of the pipeline, the leftmost portion of short path contour $i$ will be a line segment with a finite slope which forms an upper bound for the $(T_i, T_c)$ feasible region. Transparent latches or some other buffering devices must be used to avoid losing data if and when it is necessary to operate the clock at an infeasibly slow frequency.

### 5.2 Removing Synchronizing Latches

Table 3 shows the short path constraints for the pipeline of Table 1. Note that for each $i$ at least one is satisfied under the optimal clock schedule.

<table>
<thead>
<tr>
<th>Latch ($i$)</th>
<th>$l$</th>
<th>Short Path Constraint</th>
<th>Satisfied by $T_i=6, T_c=14$?</th>
<th>Latch ($i$)</th>
<th>$l$</th>
<th>Short Path Constraint</th>
<th>Satisfied by $T_i=6, T_c=14$?</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>$T_i + 0T_c \leq 14$</td>
<td>yes</td>
<td>2</td>
<td>0</td>
<td>$T_i + 0T_c \leq 10$</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>$T_i + 1T_c \leq 22$</td>
<td>yes</td>
<td>1</td>
<td>1</td>
<td>$T_i + 1T_c \leq 18$</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>$T_i + 2T_c \leq 34$</td>
<td>yes</td>
<td>2</td>
<td>2</td>
<td>$T_i + 2T_c \leq 34$</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>$T_i + 3T_c \leq 42$</td>
<td>no</td>
<td>3</td>
<td>3</td>
<td>$T_i + 3T_c \leq 42$</td>
<td>no</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>$T_i + 0T_c \leq 6$</td>
<td>yes</td>
<td>0</td>
<td>0</td>
<td>$T_i + 0T_c \leq 6$</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>$T_i + 1T_c \leq 22$</td>
<td>yes</td>
<td>1</td>
<td>1</td>
<td>$T_i + 1T_c \leq 18$</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>$T_i + 2T_c \leq 30$</td>
<td>no</td>
<td>2</td>
<td>2</td>
<td>$T_i + 2T_c \leq 26$</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>$T_i + 3T_c \leq 42$</td>
<td>no</td>
<td>3</td>
<td>3</td>
<td>$T_i + 3T_c \leq 42$</td>
<td>no</td>
</tr>
</tbody>
</table>

Table 3: Short path constraint satisfaction.
From the analysis in [13], we know that the short path constraint with \( l = 0 \) for stage \( i \) applies when the early data wavefront departs from the previous latch \((i-1)\) at the enabling edge of its clock, \( i.e. \) the early arrival data wavefront \((a_{i-1})\) is blocked by latch \(i-1\) until its clock goes high, and latch \(i-1\) is a synchronizing latch. The \( l = 0 \) constraint insures that there will be no hold time violation for latch \( i \) in this case, \( i.e. \) \( a_i \geq H_i \). However, this constraint need not be satisfied since \( d_{i-1} \) and hence \( a_i \) will occur later if latch \(i-1\) is nonsynchronizing. In general, consider a synchronizing latch, latch \( k \), where all of the latches \( k+1, k+2, \ldots, i-1 \) are nonsynchronizing, the \( l = i - k - 1 \) short path constraint for stage \( i \) insures that \( a_i \geq H_i \). Thus from Table 3 we conclude that \( a_i \) is synchronized by latch 2 (since only \( l = 0 \) is satisfied for \( i = 3 \)). If more than one short path constraint is satisfied for stage \( i \), more than one latch is capable of being the synchronizing latch for \( a_i \). Thus \( a_i \) may be synchronized by latch 1 or 3, \( a_j \) by latch 0 or 3, and \( a_j \) by latch 1, 2, or 3.

We say that latch \( i \) is \textit{supported} by latch \( k \) under some \((T_1,T_c)\) if the \( l = i - k - 1 \) short path constraint is satisfied under \((T_1,T_c)\), \( i.e. \) latch \( k \) could be the synchronizing latch for \( a_i \). A \textit{support graph} illustrates the support relationships between the latches and is derived from the satisfiability conditions of the short path constraints for a particular \((T_1,T_c)\). A \textit{support graph} \( G(V,E) \) has vertices \( V = \{ \text{ latch } \} \) and directed edges \( E = \{ (k,i) \text{ supported by } k \} \) (see Fig. 11).

![Fig. 11. Support graph from Table 3.](image)

![Fig. 12. The revised support graph for removing latch 0, 1, 2, or 3, respectively.](image)

We claim that the optimum \((T_1,T_c)\) solution remains feasible when any set of up to \( N-1 \) latches is removed from a closed pipeline provided that each remaining latch is supported (\( i.e. \) there is an inbound edge to each remaining vertex in the support graph from another remaining vertex). Note that the degree of wave pipelining will increase by 1 for each latch removed within a stage of the
resulting pipeline. After such removal, there may be a new single-phase clock solution with a lower value of $T_c$. This follows from the fact that the setup and hold times need not be satisfied for the removed latches and the constraints are thereby relaxed. Furthermore, for the remaining latches the late arrival time may be earlier, relaxing their setup (long path) constraints. The early arrival time may also be earlier, reducing the slack in the hold (short path) constraints. However, the fact that each remaining latch is still supported guarantees that these tighter short path constraints can be satisfied.

When a latch is removed that makes some remaining latch *unsupported*, the original optimum $(T_1, T_c)$ solution becomes infeasible. In this case, there may be no feasible solution, or there may be solutions with larger or even smaller values of $T_c$.

Removing a latch corresponds to removing the corresponding vertex and all the edges associated with the vertex in the support graph. (See Lemma 1). Fig. 12 shows the modified support graphs for removing each latch.

From Fig. 12 (c) we see that latch 3 is not supported when latch 2 is removed; however, all remaining latches are supported in the other graphs. $(T_1, T_c) = (6, 14)$ thus remains feasible when latch 0, 1, or 3 is removed. Removing latch 0 was described above. Table 4 and Table 5 define the pipelines and Fig. 13, Fig. 14, Fig. 15 and Fig. 16 illustrate the feasible regions and the optimal clock schedule for removing latch 1 and removing latch 3. As for removing latch 2, the modified pipeline will either have no feasible solutions or have a feasible region excluding the clock schedule of the original pipeline. In this example, it has no feasible solution, as shown in Table 4 and Fig. 17.

<table>
<thead>
<tr>
<th>$i$</th>
<th>$\delta_i$</th>
<th>$\Delta_i$</th>
<th>$\nu_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>16.0</td>
<td>18.0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>20.0</td>
<td>24.0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>8.0</td>
<td>10.0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4: Timing parameters with latch 1 removed. ($w_H = w_L = 1.0$, $H_i = S_i = 2$)

<table>
<thead>
<tr>
<th>$i$</th>
<th>$\delta_i$</th>
<th>$\Delta_i$</th>
<th>$\nu_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>24.0</td>
<td>28.0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>8.0</td>
<td>10.0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>12.0</td>
<td>14.0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 5: Timing parameters with latch 3 removed. ($w_H = w_L = 1.0$, $H_i = S_i = 2$)
The support graph in Fig. 11 indicates that if we remove both latch 2 and 3, leaving only latches 0 and 1 in the pipeline (or vice versa), the pipeline will still operate correctly on the schedule. The optimal solution of the pipeline with latch 2 and 3 (or latch 0 and 1) remaining actually improves to (6, 14). Although (6, 14) is no longer feasible, the other cases of closed pipelines with two latches remaining have the optimum schedule (5, 13) for latch 0.
and 2 (or latch 3 and 1) and (6, 13) for latch 1 and 2 (or latch 3 and 0) remaining. Since there are no self cycles in Fig. 11, (6, 14) is not a feasible schedule with only one remaining latch in the pipeline. The pipeline, however, can still operate correctly with only one remaining latch and the optimal schedule improves to (3, 13). When there is only one latch in a closed pipeline, it does not matter which latch remains.

6 Stopping and Starting of A Single-phase Closed Pipeline

When wave pipelining is used in any stage, the short path contour of that stage has no segment with infinite slope. Thus the feasible region is finite and the clock cannot be slowed down arbitrarily. This has been a practical barrier to the use of wave pipelines in circuits where single-step or slow-clock testing is needed, or where some pipeline stalls are implemented by temporarily disabling latches. Many designers avoid wave pipelining for this reason.4

Previous research on wave pipelining [1][2][6][18][19] has focused primarily on the theory of wave pipelining and various implementation issues without addressing the practical problems of stoppability and startability. Solutions to this problem must be found and accepted before designers will use of this aggressive design technique in their systems.

The stopping and starting of wave pipelines is considered in this section. We describe one simple and direct implementation method in which each physical stage, i (which ends with one nonremoved latch and begins just after the previous nonremoved latch), includes \( i \) “removed” latches within it. It may then be possible to stop the clock, i.e. drop the latch enable inputs, and catch one wave of data in each latch. Similarly it may be possible to restart the clock and resume computation from the saved state. A formal characterization of stoppability and startability indicates where the removed latches may be placed and how to sequence the shut down and restart of the clock.5

---

4 A false, but often cited, reason for avoiding wave pipelining is that minimum delays cannot be specified accurately. Although the performance advantage of wave pipelining increases with accuracy, accurate specification is not required for wave pipelining, only some lower bound on what the minimum delay is.

5 Use of Load control inputs on latch-based register modules allows the latch Enable inputs to be connected to a continuously running clock, but the discussion is simplified by assuming that the clock itself is stopped and restarted.
6.1 Stopping a Wave Pipeline with “Removed” Latches

6.1.1 Problem Definition:

To stop the pipeline, we choose one latching edge of the single-phase clock as the stopping edge, defined as $t = 0$ in global time and disable all further clock pulses for all nonremoved latches (Fig. 18). We say that each nonremoved latch, $i$ has a stopping time of $\eta_i = 0$. These non-removed latches could be of any type: transparent, partially synchronizing or synchronizing. Then each “removed” latch, $j$, is to be assigned a stopping time of $\eta_j$ relative to $t = 0$. We wish to find a set of stopping windows $[L_j, U_j]$ such that if each $\eta_j$ is chosen anywhere in its stopping window, the $\nu_j$ added waves in each stage are reliably stored in the removed latches when the pipeline is stopped. Once the valid stopping windows are known, it is a simple matter to select the $\eta_j$ values for economic clocking logic, e.g. by minimizing the number of distinct $\eta_j$.

In the next section, first we derive the equations for computing $[L_j, U_j]$ using the latch signal arrival time and the latch parameters. Then, we discuss the shut down sequence for several extreme latch placement schemes. Last, we compute the length of the stopping window and give necessary and sufficient conditions for pipeline stoppability.

![Fig. 18. The pipeline stopping windows.](image)
6.1.2 Analysis of Stopping Windows:

Suppose we have a closed pipeline with a valid clock schedule \((T_1, T_c)\) where each physical stage, \(i\), contains \(\nu_i\) “removed” latches. In particular, there exists a group of \(k\) “removed” transparent latches \(i-k, i-k+1, \ldots, i-1\) between two nonremoved latches \(i-k-1\) and \(i\), and \(\nu_i = k\). The propagation relationship between latch \(i-k-1\) and \(i\) can be characterized by the following early and late arrival equations (in latch \(i\) local time):

\[
d_{i-k-1} + \left( \sum_{j=i-k}^{i} \delta_j \right) - (k+1) T_c = a_i \tag{15}
\]

\[
D_{i-k-1} + \left( \sum_{j=i-k}^{i} \Delta_j \right) - (k+1) T_c = A_i \tag{16}
\]

Under \((T_1, T_c)\), there are \(k+1\) waves between latch \(i-k-1\) and \(i\). When stopping the clock, let wave 0 be the wave caught by latch \(i\), then let latch \(i-1\) catch wave 1, ..., latch \(i-r\) catch wave \(r\), ..., and latch \(i-k\) catch wave \(k\).

To calculate the stopping window for latch \(i-r\) \((1 \leq r \leq k)\), we need to know (in global time): i) the latest arrival time, \(t_{lat(i-r)}\), of wave \(r\) at latch \(i-r\), and ii) the earliest arrival time, \(t_{ea(i-r)}\), of wave \(r+1\) at latch \(i-r\) at which time the valid wave \(r\) changes to invalid at latch \(i-r\). Latch \(i-r\) can catch wave \(r\) successfully if the clock is stopped (dropped) anywhere between setup time after \(t_{lat(i-r)}\) and between hold time before \(t_{ea(i-r)}\). Thus the stopping window for latch \(i-r\) is:

\[
[L_{i-r}, U_{i-r}] = [t_{la(i-r)} + S_{i-r}, t_{ea(i-r)} - H_{i-r}] \tag{17}
\]

By using (8) and a local-global time change, \(t_{la(i-r)}\) can be calculated from \(D_{i-k-1}\) as follows:

\[
t_{la(i-r)} = D_{i-k-1} + \left( \sum_{j=i-k}^{i-r} \Delta_j \right) - (k-r+2) T_c \tag{18}
\]

Substituting (16) into (18), we get

\[
t_{la(i-r)} = A_i + (r-1) T_c - \sum_{j=i-r+1}^{i} \Delta_j \tag{19}
\]

Similarly, \(t_{ea(i-r)}\) can be calculated from (7), \(d_{i-k-1}\) and a local-global time transfer as follows:
\[ t_{ea(i-r)} = d_{i-k-1} + \left( \sum_{j = i-k}^{i-r} \delta_j \right) - (k-r+1) T_c \]  

(20)

Substituting (15) into (20), we get

\[ t_{ea(i-r)} = a_i + rT_c - \sum_{j = i-r+1}^{i} \delta_j \]  

(21)

Consider the examples of removed latches in section 5. The stopping windows for removed latch 0 in Fig. 10, removed latch 1 in Fig. 14 and removed latch 3 in Fig. 16 are [1,8], [-4,1] and [-6,0], respectively. All latches of the example with latch 3 “removed” can be stopped at the same time, \( t = 0 \), even though this example shows a speedup over the clock rate of the original pipeline in section 3.3.

If the removed latches are nearly equally spaced between two nonremoved latches then the \( \eta_j \) will all be nearly 0. However, if one nonremoved latch and a series of \( \nu_i \) removed latches are nearly adjacent to one another, they will span approximately \( \nu_i T_c \) in stopping time (Fig. 19). Case (a) may be preferred over case (b) when removed latches cannot be equally spaced since in case (a) the nonremoved latches may be stopped within one clock period of a decision to stop, and the removed latches are stopped later, while in case (b) after decision to stop, the removed latches may be stopped successively over the next several clock periods while the clock keeps running and the nonremoved latches are stopped last. The total stopping time is approximately \( \nu_i T_c \) in both cases. The reason for stopping the pipeline is associated with retaining the observable state in some nonremoved latch, then case (a) provides more immediate control.

From (17), (19) and (21), we calculate the length of the stopping window to be

\[
W_{len(i-r)} = U_{i-r} - L_{i-r} \\
= t_{ea(i-r)} - t_{la(i-r)} - S_{i-r} - H_{i-r} \\
= a_i - A_i + T_c + \sum_{j = i-r+1}^{i} (\Delta_j - \delta_j) - S_{i-r} - H_{i-r} \\
\]  

(22)

We say that latch \( i-r \) is **stoppable** if \( W_{len(i-r)} \geq 0 \); otherwise, latch \( i-r \) is **not stoppable**. The pipeline is **stoppable** if and only if every removed latch is stoppable. From the following analysis, we
see that a given pipeline is stoppable as long as a reasonable restriction on the $S_{i,r}$ and $H_{i,r}$ is satisfied for the removed latches.

Recall that latch $i$ is not removed. So latch $i$ satisfies equations (3) and (4). Combining these equations, we get

$$a_i - A_i + T_c - S_i - H_i \geq 0$$  \hspace{1cm} (23)

By subtracting and adding $S_i + H_i$ in (22) and rearranging, we get

![Diagram](image-url)
\[ W_{\text{len}(i-r)} = (a_i - A_i + T_c - S_i - H_i) + \sum_{j = i-r+1}^{i} (\Delta_j - \delta_j) + (S_i + H_i - S_{i-r} - H_{i-r}) \] \hspace{1cm} (24)

Note that (23) insures that the first term is nonnegative. The middle term is nonnegative since \( \Delta_j \geq \delta_j \). Normally, \( H_{i-r} \equiv H_i \), \( S_{i-r} \equiv S_i \), and the pipeline is stoppable.\(^6\) If the last term in (24) is negative for some \( i-r \), then \( W_{\text{len}(i-r)} \) must be fully evaluated to see whether the pipeline is stoppable.

6.2 Restarting a Wave Pipeline with “Removed” Latches

6.2.1 Problem Definition:

Consider the pipeline described in section 6.1 and the latch starting mechanism shown in Fig. 20. Define \( t = 0 \) (global) as the first time after starting when the clocks on all nonremoved latches drop. Each nonremoved latch, \( i \), has a starting time, \( \eta_i \), measured relative to \( t=0 \), where \(-T_c < \eta_i \leq -w_H\). Thus, at a nonremoved latch, \( i \), the first clock pulse after starting will rise at \( max \{ \eta_i, -T_1 \} \) and will drop at \( t=0 \). The normal steady-state clock will appear on the latch EN line from \( t=0 \) onward. Similarly, each “removed” latch, \( j \), has a time, \( \eta_j \), measured relative to \( t=0 \), where its clock (EN) is raised and remains high until the pipeline is stopped.

We wish to find a starting window \([ L_i, U_i ]\) for the \( \eta_i \) of each nonremoved latch, \( i \), and \([ L_j, U_j ]\) for the \( \eta_j \) of each “removed” latch, \( j \), such that if each \( \eta \) is chosen anywhere in its starting window the restart is valid, i.e. the pipeline eventually resumes its steady-state operating mode without any setup or hold time violations. As with stopping, we desire to find an assignment with as few different \( \eta \) values as possible.

In the next section, we discuss several ways of computing the starting window \([ L_j, U_j ]\). These methods are chosen according to the types of the latches and the way that we want the pipeline to be restarted. If we want the pipeline to be stable immediately upon restart, the calculation is dependent on the steady-state arrival and departure times of the pipeline. However, the resulting

\(^6\) Since the stable wave time, \( t_{\text{stat}(i-r)}^{\text{stat}(i-r)} \), is nondecreasing in \( r \), the stable wave time is no less than \( H_i + S_i \) and nondecreasing in \( r \). Furthermore, placement of removed latches earlier in the physical stage will increase \( W_{\text{len}} \) by the increase in \( \sum (\Delta - \delta) \). Even if the pipeline is nonstoppable, such retiming could make it stoppable.
number of different $\eta$ values may be large. If we want to reduce this number, the starting windows need to be extended in both directions to produce more overlap among the windows. However, when extended windows are used for starting, although the pipeline will operate correctly but steady-state arrival and departure operation may only be reached after many clock cycles.

In section 6.2.3 and 6.2.4, we derive the algorithms to extend the left side of the window and the right side of the window, respectively. In section 6.2.6, we use an extreme case to study the time required after restart for a pipeline to resume steady-state operation.

In the following discussion, the steady-state arrival and departure times for each nonremoved latch of the pipeline is denoted as $(a_i, A_i)$ and $(d_i, D_i)$, respectively, in latch $i$ local time.

### 6.2.2 Analysis of Starting Windows:

As in section 6.1.2, we assume a group of $k = \nu_i$ “removed” latches $i-k, i-k+1, ..., i-1$ between two nonremoved latches $i-k-1$ and $i$. The propagation relationships between latch $i-k-1$ and $i$ are characterized by (15) and (16). After stopping, $k+2$ successive waves of data are stored in latches $i$ through $i-k-1$, inclusive, and we assume that the pipeline has been stopped long enough for each wave to have arrived at the next latch before restarting. When the clock is raised for the first time at a latch, the arrived wave will depart immediately.

The pipeline is said to enter “steady-state operation” immediately upon restarting if for each nonremoved latch, $i$, we choose the $\eta_j$ and $\eta_{i-k-1}$ such that the first waves that depart latches $i-1$, $i-2$, ..., $i-k-1$ arrive at latch $i$ in the intervals $[a_i, A_i]$, $[a_i + T_c, A_i + T_c]$, $[a_i + kT_c, A_i + kT_c]$, respectively, in global time relative to $t = 0$. Although this starting strategy is sufficient, it is not
necessary for a legal startup as discussed below. With this in mind, we can now discuss starting strategies for two cases.

**Case 1:** If no transparent or removed latches exist, the easiest way to restart the pipeline is to set all \( \eta_i = -T_1 \). Thus the first departing wave will arrive at latch \( i+1 \) in the \([-T_1 + \delta_{i+1}, -T_1 + \Delta_{i+1}]\) interval. This restart is valid since in this case \( d_i = T_c - T_1 \) and \( T_c - T_1 \leq D_{i+1} \) in the steady-state for all latches. Combining them with (7) and (8), we can obtain:

\[
a_{i+1} = -T_1 + \delta_{i+1} \leq -T_1 + \Delta_{i+1} \leq A_{i+1}.
\]

**Case 2:** If at least one transparent or removed latch exists, two starting strategies are described below:

2.A To enter steady-state operation immediately, place \( \eta \) within the steady-state departure interval.
- For a transparent nonremoved latch, set \( \eta_i \) in the window \( [L_i, U_i] = [d_i - T_c, D_i - T_c] \).
- For a partially-synchronizing latch, set \( \eta_i \) in \( [L_i, U_i] = [-T_1, D_i - T_c] \) since \( d_i = T_c - T_1 \).
- For a synchronizing latch, set \( \eta_i \) in \( [L_i, U_i] = [-T_1, -T_1] \) since \( d_i = D_i = T_c - T_1 \).
- For a removed latch, \( i-r \), the wave that departs at \( t = \eta_{i-r} \), has early and late arrival times at latch \( i \) within \( [a_i + (r-1) T_c, A_i + (r-1) T_c] \) if and only if \( \eta_{i-r} \) is in the window

\[
[L_{i-r}, U_{i-r}] = \left[ a_i + (r-1) T_c - \sum_{j = i-r+1}^{i} \delta_j A_i + (r-1) T_c - \sum_{j = i-r+1}^{i} \Delta_j \right]
\]  

(25)

This starting assignment (Fig. 21) is similar to the stopping assignment in section 6.1.2 (Fig. 19).

2.B To have as few different \( \eta \) values as possible (without requiring immediate steady-state operation), we attempt to extend the starting windows \( [L_i, U_i] \) from Case 2.A to obtain more overlap:

i) One possibility is to let waves depart earlier than the steady-state \( d_i \), i.e. to decrease \( L_i \) further (Fig. 22). To do this we find the minimum \( L_i \), i.e. the maximum reduction of \( L_i \) (called the maximum early extension), without violating the hold time requirements for latches in the pipeline.
ii) Another possibility is to let waves depart later than the steady-state $D_i$, i.e. to increase $U_i$ further (Fig. 22). To do this we find the maximum $U_i$, i.e. the maximum increase of $U_i$ (called the maximum late extension), without violating the setup constraints.

This approach has a drawback that approach i) does not have. If we let the late departure of a wave be even later than in steady-state, then because of the transparent characteristic of a latch, this wave may cause a series of latches that are synchronizing in steady state to become partially-synchronizing, at least temporarily. In this case, the following latches in the loop may not resume steady state until $cN$ clocks later, where $N$ is the number of latches in the pipeline and $c$ may be $\gg 1$.

Fig. 21. Two starting cases (with $k = 2$): (a) 2 removed latches placed close to latch $i$ (b) 2 removed latches placed close to latch $i-3$. 
In the remaining sections we consider both early and late extensions of the starting window, show a reduction in the number of distinct $\eta$ by example, and describe how steady-state operation is eventually reached. The following theorem is the justification for the extension calculation algorithm in section 6.2.3:

**Theorem 2:** For an $N$-stage closed pipeline, with $\sum_{j=0}^{N-1} \delta_{j,j+1} < \sum_{j=0}^{N-1} \Delta_{j,j+1}$, there must exist at least one nontransparent latch in any feasible clocking solution.

**Proof:** See Appendix.

This theorem allows the MEE algorithm described in the next section to begin with any nontransparent latch and proceed backwards around the closed pipeline by induction.

![Fig. 22. Early and Late extensions of a legal starting window.](image-url)
6.2.3 Algorithm to find the earliest possible starting point for all latches in the pipeline

Let $MEE_i$ be the maximum early extension for latch $i$. Let $L^e_i$ be the earliest starting time for latch $i$, i.e. $L^e_i = L_i - MEE_i$ for removed latches and $L^e_i = \max (L_i - MEE_i, -T_c)$ for non-removed latches.

Algorithm to determine MEE: (Fig. 23)

1. Choose any nontransparent latch, say latch $i$. From latch $i$ we can compute $MEE_{i-1}$ because a data wave departing from latch $i-1$ has to satisfy the hold time constraint of latch $i$. Also because latch $i$ is nontransparent, a data wave that arrives earlier than its steady state early arrival time will be delayed at latch $i$ by its clock. It follows that an early arrival at latch $i$ cannot cause hold time violations for the latches succeeding latch $i$. Thus:

$$MEE_{i-1} = a_i - H_i \quad \text{and} \quad L^e_{i-1} = L_{i-1} - MEE_{i-1}$$

It follows that for latch $i$ nontransparent, $MEE_{i-1} \leq T_c - T_1$ and $-T_1 \leq L_{i-1}$. Therefore $L^e_{i-1} \geq -T_c$.

2. Next, for $r=1$ to N-1, compute $MEE_{i-r-1}$ from the status of latch $i-r$ and $MEE_{i-r}$:

   (a) If latch $i-r$ is nontransparent, $MEE_{i-r-1} = a_{i-r} - H_{i-r}$

   (b) If latch $i-r$ is a nonremoved transparent latch:

      i) If $L^e_{i-r} > -T_1$,

      $$MEE_{i-r-1} = \min (MEE_{i-r}, a_{i-r} - H_{i-r})$$

      $$= MEE_{i-r} \quad H_{i-r} < T_c - T_{\text{normal}}$$

      ii) If $L^e_{i-r} \leq -T_1$, $MEE_{i-r-1} = a_{i-r} - H_{i-r}$.

   (c) If latch $i-r$ is a removed transparent latch, $MEE_{i-r-1} = MEE_{i-r}$.

Explanation for Step 2:

- Step 2(a) has the same explanation as Step 1.
- Step 2(b):
i) Because \( \eta_{i-r} \geq L_{i-r}^e > -T_1 \), \( T_c + L_{i-r}^e \) is the earliest legal departure time for waves leaving latch \( i-r \) without causing a hold time violation at latch \( i-r+1 \). Therefore the data wave leaving from latch \( i-r-1 \) has to arrive at latch \( i-r \) no earlier than \( T_c + L_{i-r}^e \). Otherwise, since \( L_{i-r}^e > -T_1 \), a data wave arriving before \( T_c + L_{i-r}^e \) would depart before \( T_c + L_{i-r}^e \) and would therefore violate the hold time of some subsequent latch. Thus in the formula for \( MEE_{i-r-1} \), the \( a_{i-r} - H_{i-r} \) term insures that no hold violation occurs at latch \( i-r \) and the \( MEE_{i-r} \) term insures that no hold violation occurs at any subsequent latch (after latch \( i-r \)) for the wave departing latch \( i-r-1 \).

ii) Because \( L_{i-r}^e \leq -T_1 \), a data wave departing latch \( i-r-1 \) and arriving at latch \( i-r \) earlier than \( T_c + L_{i-r}^e \) will still depart latch \( i-r \) at \( T_c - T_1 \). Since \( T_c - T_1 \geq T_c + L_{i-r}^e \), this wave will not violate the hold time of any subsequent latch (after latch \( i-r \)). The only limiting factor in computing \( MEE_{i-r-1} \) is latch \( i-r \) itself. So \( MEE_{i-r-1} = a_{i-r} - H_{i-r} \).

- Step 2(c): Since latch \( i-r \) is “removed” by setting its enable permanently high at \( \eta_{i-r} \), there is no hold time consideration for latch \( i-r \). The only limiting factor on \( MEE_{i-r} \) is to insure no hold violations occur at subsequent latches.

### 6.2.4 Algorithm to find the latest possible starting point for all latches in the pipeline

Let \( MLE_i \) be the maximum late extension for latch \( i \). Let \( U_i^e \) be the latest starting time for latch \( i \), i.e. \( U_i^e = U_i + MLE_i \) for removed latches and \( U_i^e = \min (U_i + MLE_i, -w_H) \) for non-removed latches.

**Algorithm to determine MLE:** (Fig. 24)

1. Search through all latches, except “removed” transparent latches, which have no setup constraints, to find the latch \( i \) which has the minimum value of \( T_c - S_i - A_i \). In steady-state operation, latch \( i \) will be the latch with the least slack for setup among all latches in the pipeline. From this starting latch we can compute \( MLE_{i-1} \) for latch \( i-1 \) and proceed backwards to...
Latch \( i \) is nontransparent

\[ MEE_{i-1} = a_i - H_i \]

Step 1

Latch \( i-1 \) nonremoved trans. or removed

Step 2

(a) Synch

(b) Partially synch

(c) Latch \( i-1 \) Removed

\[ MEE_{i-2} = a_{i-1} - H_{i-1} \]

Case (a)

\[ MEE_{i-1} = \min \left( \frac{H_{i-1}}{H_i}, \frac{a_i}{H_i} \right) \]

Case (b)

\[ L_{i-1}^e \leq T_1 \]

\[ L_{i-1}^e > T_1 \]

Case (c)

\[ MEE_{i-2} = \min( MEE_{i-1}, a_{i-1} - H_{i-1} ) \]

Fig. 23. Timing diagrams for MEE_{i-1} and MEE_{i-2}. 
compute \( MLE \) for all other latches until obtaining \( MLE_i \) for latch \( i \). When computing \( MLE_{i-1} \), we are concerned only about satisfying the setup time at latch \( i \). The fact that latch \( i \) has minimum setup slack guarantees that latches succeeding latch \( i \) do not influence the computation of \( MLE_{i-1} \):

\[
MLE_{i-1} = T_c - S_i - A_i
\]

2. Next, for \( r=1 \) to \( N-1 \), compute \( MLE_{i-r-1} \) from the status of latch \( i-r \) and \( MLE_{i-r} \):

(a) If latch \( i-r \) is a nonremoved transparent or partially-synchronizing latch,

\[
MLE_{i-r-1} = \min \left( MLE_{i-r}, T_c - S_{i-r} - A_{i-r} \right)
\]

(b) If latch \( i-r \) is a removed transparent latch, \( MLE_{i-r-1} = MLE_{i-r} \).

(c) If latch \( i-r \) is synchronizing,

\[
MLE_{i-r-1} = \min \left( T_c - T_1 - A_{i-r} + MLE_{i-r}, T_c - S_{i-r} - A_{i-r} \right)
\]

**Explanation for Step 2:**

- Step 2(a) and 2(c): A wave departing latch \( i-r-1 \) later than its steady state late departure time will arrive latch \( i-r \) later than its steady state late arrival time, but the \( T_c - S_{i-r} - A_{i-r} \) term in the formula of both cases insures that no setup violation occurs for latch \( i-r \).

In Step 2(a) due to the transparent nature of the steady-state late arrival of a data wave at latch \( i-r \), the \( MLE_{i-r} \) term in the formula is required to ensure that no setup violation occurs for any latch subsequent to latch \( i-r \).

In Step 2(c) a data wave arrival may be delayed by \( T_c - T_1 - A_{i-r} \) beyond the steady state late arrival time, \( A_{i-r} \), without affecting the departure from latch \( i-r \) at all. However, the departure from latch \( i-r \) may be delayed by up to \( MLE_{i-r} \) without causing hold violations at any succeeding latches (after latch \( i-r \)). Consequently, the \( T_c - T_1 - A_{i-r} + MLE_{i-r} \) term in the formula insures that no hold violation occurs for any latch subsequent to latch \( i-r \).

- Step 2(b) has the same explanation as Step 2(c) in section 6.2.3, except that the hold time constraint is replaced with the setup time constraint.
6.2.5 Examples

The starting windows of all latches in the removed-latch examples in section 5 are illustrated in Table 7. The original pipeline under the optimal solution can be started by using a single starting signal at $\eta = -6$, even though latch 0 is not synchronizing, provided that an early extension of the latch 0 starting window is used. Otherwise $\eta_0 \in [-4, -2]$ and at least two different $\eta$ values are required.

Fig. 24. Timing diagrams for $\text{MLE}_{i-1}$ and $\text{MLE}_{i-2}$.
All latches of the latch 0 removed example can be started by using a single starting signal at
\( \eta_i \in [-6, -5] \) after extending the window boundary; however, at least two different \( \eta \) values are
required if the steady-state departure interval is not extended. One interesting situation in this
example is that if we choose \( \eta = -6 + \zeta \in (-6, -5) \) as the starting point for all latches, the
steady state of the pipeline will be shifted by \( \zeta \). This reaffirms that there are multiple steady-state
operation modes due to the range of possible solutions for \( T_1 \) under the optimum \( T_c \) value, as
seen in Fig. 9.

For the other examples, latch 1 or 3 removed, at least 2 values of \( \eta \) are required even with
extensions. When latch 3 is removed and \( T_c \) is reduced to 13, latch 3 must be restarted earlier than
\(-T_1\). Consequently, removing latch 0 is a better \( T_c = 13 \) design than removing latch 3.

<table>
<thead>
<tr>
<th>Example</th>
<th>( L (i) )</th>
<th>SW (2.A)</th>
<th>Fewest ( \eta )</th>
<th>Fewest ( \eta )</th>
<th>MEE (i)</th>
<th>MLE (i)</th>
<th>SW (2.B)</th>
<th>Fewest ( \eta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>No latch removed</td>
<td>0</td>
<td>[-4,-2]</td>
<td>2</td>
<td></td>
<td>2</td>
<td>4</td>
<td>[-6,-1]</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>[-6,-6]</td>
<td></td>
<td></td>
<td>4</td>
<td>4</td>
<td>[-6,-2]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>[-6,-6]</td>
<td></td>
<td></td>
<td>0</td>
<td>4</td>
<td>[-6,-2]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>[-6,-6]</td>
<td></td>
<td></td>
<td>8</td>
<td>0</td>
<td>[-14,-6]</td>
<td></td>
</tr>
<tr>
<td>Latch 0 removed</td>
<td>0</td>
<td>[-3,-1]</td>
<td>2</td>
<td></td>
<td>3</td>
<td>1</td>
<td>[-6,0]</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>[-6,-4]</td>
<td></td>
<td></td>
<td>4</td>
<td>1</td>
<td>[-10,-3]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>[-6,-3]</td>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
<td>[-6,-2]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>[-6,-6]</td>
<td></td>
<td></td>
<td>3</td>
<td>1</td>
<td>[-9,-5]</td>
<td></td>
</tr>
<tr>
<td>Latch 1 removed</td>
<td>0</td>
<td>[-4,-2]</td>
<td>2</td>
<td></td>
<td>0</td>
<td>4</td>
<td>[-4,-1]</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>[-10,-6]</td>
<td></td>
<td></td>
<td>0</td>
<td>4</td>
<td>[-10,-2]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>[-6,-6]</td>
<td></td>
<td></td>
<td>0</td>
<td>4</td>
<td>[-6,-2]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>[-6,-6]</td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>[-6,-6]</td>
<td></td>
</tr>
<tr>
<td>Latch 3 removed</td>
<td>0</td>
<td>[-6,-3]</td>
<td>2</td>
<td></td>
<td>0</td>
<td>1</td>
<td>[-6,-2]</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>[-6,-6]</td>
<td></td>
<td></td>
<td>4</td>
<td>1</td>
<td>[-10,-5]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>[-6,-5]</td>
<td></td>
<td></td>
<td>3</td>
<td>1</td>
<td>[-9,-4]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>[-11,-8]</td>
<td></td>
<td></td>
<td>3</td>
<td>1</td>
<td>[-14,-7]</td>
<td></td>
</tr>
</tbody>
</table>

Table 7: Starting Windows for previous examples.
6.2.6 Stability analysis of a started pipeline

When choosing an $\eta$ value from each extended starting window, we desire to choose as few different $\eta$ values as possible. Suppose that the intersection of all the extended windows is non-empty. Then a single value of $\eta$ can be chosen for starting all the latches in the pipeline. In this section we consider how soon after starting the pipeline is guaranteed to resume steady-state operation when a single value of $\eta$ is used.

Define:

- $K$: the total number of waves in the pipeline.
- $\sum \Delta$: the total long path delay of a given pipeline.
- $(\sum \Delta)/K$: the absolute lower bound on the clock cycle time of a feasible solution from (11).

**Theorem 3:** Consider a pipeline with a clock cycle time $T_c \geq (\sum \Delta)/K$. Suppose further that a single value of $\eta$, $\eta \geq -T_1$, can be used to start all latches of the pipeline. If this value is used, the pipeline will resume steady-state operation within

$$\max \left\{ K \left[ \frac{T_1 - S}{KT_c - \sum \Delta} \right] + K - 1, K \left[ \frac{T_1 + \eta}{KT_c - \sum \Delta} \right] \right\}$$

clock cycles after starting. The minimum setup time over all nonremoved latches is $S$.

**Proof:** See Appendix.

**Corollary (to Theorem 3):** If the conditions of Theorem 3 apply, except that several distinct $\eta$ values, $\eta_1, \eta_2, \ldots, \eta_q$, are used, then the pipeline will resume steady-state operation within

$$\max \left\{ K \left[ \frac{T_1 - S}{KT_c - \sum \Delta} \right] + K - 1, K \left[ \frac{T_1 + \max (\eta_i)}{KT_c - \sum \Delta} \right] \right\}$$

clock cycles after starting.

The proof of this Corollary follows from the fact that the analysis in the proof of Theorem 3 can be carried out independently for each distinct value of $\eta$. 
7 Conclusion

We have presented a new algorithm, Gpipe, that uses the geometric relationships among the timing constraints of a pipeline to solve the maximum rate unrestricted single-phase pipeline clocking problem. The complexity of Gpipe is \( O(N^3 + K^2 + KN^2) \) for an \( N \) stage, \( K - N \) degree wave-pipelined closed pipeline.

We have investigated the impact of removing latches on the minimum clock cycle time of a closed pipeline. It is interesting that the maximum clock speed of a pipeline can be increased in some cases by removing latches. After removing a latch, a pipeline without wave pipelining will become wave pipelined and the feasible \((T_1, T_c)\) region becomes finite. If this region becomes empty, there is no feasible single-phase clocking solution. We have, however, described a simple criterion for deciding which latches can be removed without making a previously optimum \((T_1, T_c)\) solution infeasible. By removing only such latches, the optimum solution can only improve.

We have analyzed the stopping and starting conditions of a single-phase closed pipeline and deduced ranges of stopping and starting points that can be used to stop and restart a single-phase closed pipeline under the proposed strategies without losing any data when wave pipelining is used.

These results encourage single-phase clocked, latch-based pipeline design. They insure optimum feasible clocking, and provide guidance for latch placement. The effects and feasibility of applying wave pipelining can also readily be determined.

References


Appendix

Proof (Lemma 1): For notational simplicity in the proof, we do not renumber the stages in the circuit. Thus the new N-1 stage closed pipeline has latches numbered from 0 to N-1, excluding r. Since latch r has been removed by setting its clock permanently active, it has no hold time constraints. Thus the short path constraints for \( i = r \) are simply deleted.

Let \( \hat{\delta}_i, \hat{v}_i \) be the minimum stage delay and the degree of wave pipelining of the modified pipeline respectively. The relationships in stage parameters are as follows:

\[
\hat{\delta}_i = \delta_i \quad i = 0, \ldots, r-1, r+1, \ldots, N-1 \tag{28}
\]

\[
\hat{\delta}_{r+1} = \delta_r + \delta_{r+1} \tag{29}
\]

since the old stages \( r \) and \( r+1 \) now form a new stage, labeled \( r+1 \), and

\[
\hat{v}_i = v_i \quad i = 0, \ldots, r-1, r+1, \ldots, N-1 \tag{30}
\]

\[
\hat{v}_{r+1} = v_r + v_{r+1} + 1 \tag{31}
\]

since the total number of waves, \( K = N + \sum_{0}^{N-1} v_j \), in the pipeline must remain unchanged. Note that \( \hat{v}_{r+1} \) is the total degree of wave pipelining in the new combined stage which preserves the total waves since \( K = (N-1) + \sum_{0}^{r-1} \hat{v}_i + \sum_{r+1}^{N-1} \hat{v}_i \).

To derive the short path constraint sets of the modified pipeline, we first list equations (5) and (7) with the parameters of the new pipeline, namely

\[
d_{i} = \max (a_{i}, T_c - T_l) \quad \text{for} \quad i = 0, \ldots, r-1, r+1, \ldots, N-1 \tag{32}
\]

\[
a_{i} = d_{i-1} + \hat{\delta}_i - (1 + \hat{v}_i) T_c \quad \text{for} \quad i = 0, \ldots, r-1, r+2, \ldots, N-1 \tag{32}
\]

\[
a_{r+1} = d_{r-1} + \hat{\delta}_{r+1} - (1 + \hat{v}_{r+1}) T_c
\]

Following steps similar to the derivation summary in section 2.1, we begin with equation (32) above for some \( a_j, i \neq r \), and recursively substitute for the \( d \) variable and then the \( a \) variable on the right hand side until \( a_i \) appears again on the right hand side. Then we obtain the short path constraints for this new N-1 stage pipeline as

\[
\left( \hat{l} + \sum_{j=i-1}^{i} \hat{v}_j \right) T_c + T_l \leq \left( \sum_{j=i-1}^{i} \hat{\delta}_j \right) - H_i \quad \text{for} \quad \hat{l} = 0, \ldots, i-r-1
\]
To compare (33) with (13), we rewrite (33) with the old stage parameters by using (28), (29), (30) and (31). We then see that the new constraints for latch with are the same as the old constraints for the same latch with and the new constraints with are the same as the old constraints with .

The new short path constraints for each latch , except for for which there are no short path constraints, are thus the same as the original short path constraints excluding the one with .

\[ \text{Proof (Lemma 2):} \]

(i) Let \( \hat{\Delta}_i \) be the maximum stage delay of the modified pipeline

\[ \hat{\Delta}_i = \Delta_i \quad i = 0, \ldots, r - 1, r + 2, \ldots, N - 1 \]

\[ \hat{\Delta}_{r+1} = \Delta_r + \Delta_{r+1} \]

and \( \hat{\nu}_i \) be the same as that defined in Lemma 1. The proof is then similar to Lemma 1.

(ii) Follows immediately from the fact that \( \sum_{j \neq r}^{N-1} \hat{\Delta}_j = \sum_{j = 0}^{N-1} \Delta_j \) and

\[ (N - 1) + \sum_{j = 0}^{N-1} \hat{\nu}_j = N + \sum_{j = 0}^{N-1} \nu_j \]

\[ \square \]

\[ \square \]

\[ \square \]

Proof (Theorem 1): The pulse width constraints are unchanged and, from Lemma 2, the long path constraints are relaxed (some are deleted and the others are unchanged). All of the pulse width and long path constraints of the original pipeline were satisfied by \((T_1, T_c)\), whereas only the remaining constraints need be satisfied in the modified pipeline. Thus \((T_1, T_c)\) satisfies the pulse width and long path constraints of the modified pipeline.

It remains to be shown that \((T_1, T_c)\) satisfies the new short path constraints. Since, from Lemma 1, the \( i = r \) constraints have been removed, they are no longer of concern. Since \((T_1, T_c)\) was a feasible solution for the original pipeline, it must satisfy at least one of the \( l \) constraints for each \( i \neq r \). Consider some \( i \neq r \). The short path constraint set for \( i \) insures that \( a_i \geq H_i \).
Now for the given \((T_1, T_c)\) solution, we can compute \(a_i\) in stage \(i\) using (7) and (5) recursively, from index \(i\) backwards to index \(r\). Let \(i - M \equiv r + 1\).

\[
a_i = d_{i-1} + \delta_i - (1 + v_i) T_c
\]

\[
= \max \{ a_{i-1} + \delta_i - (1 + v_i) T_c, \delta_i - v_i T_c - T_1 \}
\]

\[
= \max \{ d_{i-2} + \sum_{j=i-1}^{i} \delta_j - \left( 2 + \sum_{j=i-1}^{i} v_j \right) T_c, F1 \}
\]

\[
= \ldots
\]

\[
= \max \{ \max (a_r, T_c - T_1) + \sum_{j=r+1}^{i} \delta_j - \left( i - r + \sum_{j=r+1}^{i} v_j \right) T_c, FM, \ldots, F1 \}
\]

where \(Fm \equiv \sum_{j=i-m+1}^{i} \delta_j - \left( m - 1 + \sum_{j=i-m+1}^{i} v_j \right) T_c - T_1\).

From the theorem statement, we know that \(\max (a_r, T_c - T_1) = a_r\). Since this \(\max\) function can be resolved, and \(a_r\) can be substituted for it, we eliminate one term from the final \(\max\) expression for \(a_i\), namely \(a_i = F(M + 1)\) which cannot be the solution for \(a_i\). After this, we continue to use (7) and (5) recursively until \(a_i\) appears on the right hand side. The final expression is:

\[
a_i = \max \{ a_i + \left( \sum_{j=0}^{N-1} \delta_j \right) - \left( N + \sum_{j=0}^{N-1} v_j \right) T_c, FN, \ldots, F(M + 2), FM, \ldots, F1 \}
\]

The first term in the \(\max\) expression will never exceed \(a_i\) due to constraint (11), and thus will never be used to define the value of \(a_i\). Thus \(a_i\) will equal the largest of the remaining \(N-1\) terms and that term must satisfy the latch \(i\) hold time (3). Thus at least one of the following must be satisfied by the given \((T_1, T_c)\) solution:

\[
FN \geq H_i \quad \ldots \quad F(M + 2) \geq H_i \quad FM \geq H_i \quad \ldots \quad F1 \geq H_i \quad (34)
\]

But (34) is exactly the new short path constraint set for latch \(i\) from Lemma 1. Thus the short path constraints remain satisfied by \((T_1, T_c)\) when latch \(r\) is removed. \(\Box\)

**Proof (Theorem 2):** (By contradiction) We assume that all latches in the closed pipeline are transparent. The departure times seen by all latches are thus

\[
d_i = a_i \quad D_i = A_i \quad i = 0 \ldots N - 1 \quad (35)
\]

According to (7) and (8),

\[
a_i = d_{i-1} + \delta_i - \epsilon_{i-1,i} \quad (36)
\]

\[
A_i = D_{i-1} + \Delta_i - \epsilon_{i-1,i} \quad (37)
\]

Substituting (35) into (36) recursively, we get
\[ a_i = d_{i-1} + \delta_i - \varepsilon_{i-1,i} \]
\[ = a_{i-1} + \delta_i - \varepsilon_{i-1,i} \]
\[ = d_{i-2} + \delta_{i-1} - \varepsilon_{i-2,i-1} + \delta_i - \varepsilon_{i-1,i} \]
\[ = \ldots \]
\[ = a_i + \sum_{j=0}^{N-1} \delta_j - \sum_{j=0}^{N-1} \varepsilon_{j,j+1} \]
\[ \Rightarrow \sum_{j=0}^{N-1} \delta_j = \sum_{j=0}^{N-1} \varepsilon_{j,j+1} \quad (38) \]
\[ \text{and similarly substituting (35) into (37) recursively, we have} \]
\[ \sum_{j=0}^{N-1} \Delta_j = \sum_{j=0}^{N-1} \varepsilon_{j,j+1} \quad (39) \]

Combining (38) and (39),
\[ \sum_{j=0}^{N-1} \Delta_j = \sum_{j=0}^{N-1} \delta_j \quad (40) \]

(40) contradicts the assumption in the theorem statement.

**Proof (Theorem 3):** If \( \sum \Delta = KT_c \) the pipeline enters steady state immediately for any choice of \( \eta \) in the starting window. This follows from the fact that the first wave departures are legal and the same sequence of \( K \) departures repeats every \( KT_c \) cycles at each latch. The theorem is trivially satisfied in this case. For the remainder of this proof, we assume \( T_c > (\sum \Delta) / K \).

Case 1: Latch \( i \) is a synchronizing latch under steady-state operation. If the clock starting time \( \eta \leq -T_1 \), the wave departing from latch \( i \) will enter its steady-state immediately. If the clock starting time \( \eta \) is greater than \( -T_1 \), the wave departing from latch \( i \) may cause a series of synchronizing latches to become non-synchronizing and the pipeline will not be operating in its steady-state until after they become synchronizing again. In the worst-case situation, all latches succeeding latch \( i \) in the pipeline will let the late arrival of the wave departing from latch \( i \) pass through undelayed by the clock.

Consider what happens when this wave has traveled completely around the pipeline and arrives back at latch \( i \). In this worst case, the late departure of this wave from each latch in this first tour is outside of its steady-state departure window (this could happen, for example, if \( \forall j \neq i, \Delta_j = (1 + v_j) T_c \)). At this point, because \( KT_c > \sum \Delta \), the wave has traveled through the total long path delay \( (\sum \Delta) \) without its late departure being delayed by any latch. Thus this wave has a late arrival time at latch \( i \) of \( \eta + \sum \Delta \). The clock pulse associated with its next departure from latch \( i \)
rises at time $KT_c - T_1$. If $\eta + \sum \Delta \leq KT_c - T_1$ then latch $i$ resumes synchronizing operation and the pipeline is in steady state. If, however, $\eta + \sum \Delta > KT_c - T_1$ then latch $i$ is still nonsynchronizing. This process may continue for several tours of the pipeline. However, there is some number of tours, $X$, at which time latch $i$ must have resumed synchronizing operation, namely when

$$\eta + X\sum \Delta \leq XKT_c - T_1$$

since $\sum \Delta < KT_c$ by assumption. The smallest value of $X$ that guarantees this synchronization is thus

$$X = \left\lceil \frac{\eta + T_1}{KT_c - \sum \Delta} \right\rceil$$

(41)

At this point, latch $i$ is guaranteed to have resumed synchronous operation.

Case 2: Latch $i$ is a partially-synchronizing latch or a transparent latch that is not removed. If the clock starting time $\eta$ is later than the latch $i$ steady-state late data arrival time, the first wave departing latch $i$ may cause the same non-steady-state pipeline operation described in case 1. For the wave departing from this latch, the minimum number of tours, $X$, that guarantees the steady-state operation of latch $i$, similar to Case 1, is $X = \left\lceil \frac{\eta + T_c - A_i}{KT_c - \sum \Delta} \right\rceil$, where $A_i$ is the latest steady-state arrival time for latch $i$. However, we can cover both case 1 and 2 by using (41) to bound the time for reaching steady-state operation of the pipeline, since $T_1 \geq T_c - A_i$, i.e.

$$\left\lceil \frac{\eta + T_1}{KT_c - \sum \Delta} \right\rceil \geq \left\lceil \frac{\eta + T_c - A_i}{KT_c - \sum \Delta} \right\rceil.$$

Case 3: Latch $i$ is removed. If the data wave departing at $\eta$ has a late arrival at the first nonremoved latch $j$ that is no later than the late steady-state departure time of that latch, the data wave will enter steady-state operation. If this is not the case, the wave may experience non-synchronizing operation at succeeding synchronizing latches, similar to that described in Case 1. In the worst case, the data wave may travel $K-1$ clock cycles to encounter the first nonremoved latch $j$ and arrive at $(K - 1) T_c + \zeta$ which is later than the steady-state late departure time of latch $j$. Notice that $\zeta \leq T_c - S_j$ for valid starting. To stabilize the wave leaving latch $j$, as in Case 1, it needs

$$K \left\lceil \frac{\zeta + T_1 - T_c}{KT_c - \sum \Delta} \right\rceil$$

clock cycles in the worst case. This value plus the $K-1$ clock cycles spent before the wave reaches latch $j$ is the total number of clock cycles needed to stabilize latch $i$. The maximum value of $\zeta$ is equal to $T_c - S$, where $S = min(S_j)$. Applying the maximum value of $\zeta$ to the formula, the bound becomes $K \left\lceil \frac{T_1 - S}{KT_c - \sum \Delta} \right\rceil + K - 1$.

The maximum of the bounds in the above three cases determines the upper bound for the theorem.