# OPTIMAL PIPELINING IN SUPERCOMPUTERS

Steven R. Kunkel and James E. Smith

Department of Electrical and Computer Engineering
University of Wisconsin-Madison
Madison, Wisconsin 53706

**Abstract**

This paper examines the relationship between the degree of central processor pipelining and performance. This relationship is studied in the context of modern supercomputers. Limitations due to instruction dependencies are studied via simulations of the CRAY-1S. Both scalar and vector code are studied. This study shows that instruction dependencies severely limit performance for scalar code as well as overall performance.

The effects of latch overhead are then considered. The primary cause of latch overhead is the difference between maximum and minimum gate propagation delays. This causes both the skewing of data as it passes along the data path, and unintentional clock skewing due to clock fanout logic. Latch overhead is studied analytically in order to lower bound the clock period that may be used in a pipelined system. This analysis also touches on other points related to latch clocking. This analysis shows that for short pipeline segments both the Earle latch and polarity hold latch give the same clock period bound for both single-phase and multi-phase clocks. Overhead due to data skew and unintentional clock skew are each added to the CRAY-1S simulation model. Simulation results with realistic assumptions show that eight to ten gate levels per pipeline segment lead to optimal overall performance. The results also show that for short pipeline segments data skew and clock skew contribute about equally to the degradation in performance.

## 1. Introduction

Pipelining is an essential element of modern supercomputer design. Each new supercomputer generation has used a higher degree of pipelining than its predecessor. Furthermore, pipelining is becoming important in other, lower performance computer systems. Pipelining is a very appealing design technique because it offers a theoretical speedup of $N$ when $N$ pipeline stages are used. There are, however, practical constraints that limit the performance increases that are possible. These are:

(1) **Instruction dependencies** that cause the pipeline to be less than 100 percent utilized;

(2) **Latch overhead** that tends to aggravate the effects of instruction dependencies, as well as placing some fundamental limitations on the clock frequency (and the degree of pipelining) that can be used;

(3) **Control path limitations** that force a minimum amount of logic to be placed between pipeline segments.

In this paper, we study the relationship between the theoretical linear speedup that pipelining offers and the practical limitations. One goal is to determine the ultimate performance improvements that are possible through pipelining. Another goal is to study optimal latching and clocking methods in pipelined computers. In order to increase its usefulness, this study is made in the context of current supercomputer technology, design techniques, architectures, and compilers.

### 1.1. Instruction Dependencies

Instruction dependencies, involving both data and control information, limit performance because they reduce the amount of the potential parallelism that is actually realized. Such dependencies are a very important practical limitation, and are a property of the algorithms, programs, and compilers. Hence, in order to arrive at meaningful results we study them using simulation of a state-of-the-art pipelined computer system, the CRAY-1S. [1]

### 1.2. Latch Overhead

There are three main components to latch overhead.

(1) **Propagation delay** through storage elements can cause extra pipeline latency. Careful latch design can significantly reduce this component of latch overhead, and we discuss such latch designs in Section 3.

(2) **Data skew** is the difference between the maximum and minimum signal propagation times through combinational logic between pipeline stages, and in the latches that separate the stages. This skew occurs even if the clock signal to the latch is perfectly controlled, and it forces constraints on the clock period in order to ensure reliable latching of data.

(3) **Clock skew,** due primarily to differences between maximum and minimum delays in clock fanout logic, causes an unintentional variation in the arrival time of the clock at succeeding latches in a pipeline. This unintentional skew increases the clock period necessary to ensure reliable latching of data.

### 1.3. Control Path Limitations

Control information must pass down the stages of a pipelined computer, just as the data does. It is often more difficult to sub-divide control operations than data operations, however. For example, control logic is required to interlock pipeline stages. These interlocking operations can not be sub-divided while maintaining an execution rate of one instruction per clock period.

Consider the instruction issue logic in the CRAY-1S. At the time an instruction is issued to the execution units, any register that the instruction uses must not be reserved for a result by an earlier instruction. If the operand registers are all available for use, the current instruction reserves its result register as it issues. In order to issue an instruction every clock period, however, checking the register reservations and reserving the result register must both be performed in the same clock period. Because of the indivisibility of this operation, there is a minimum logic delay that limits the clock period that can be used.

Control path limitations are a very important design consideration. The best solution is a clean architecture; those designed by S. Cray are excellent examples. Because a study of control path limitations would involve architectural and detailed logic design alternatives that are beyond the scope of this paper, we will not consider them.

[1]Although the CRAY-1S is older than the CRAY X-MP and CRAY-2, the major differences between a CRAY-1S cpu and the newer models are in the gate and packaging technology, not the cpu architecture and logic design techniques.

## 1.4. Previous Research

The problem of detecting and utilizing independent instructions in a single instruction stream has been studied previously [RIS72, FOS72, TJA70, SHA77, NIC84]. In most of these studies, however, pipelining was not specifically studied, and an infinite hardware machine was assumed so an upper limit on the available parallelism could be determined. Hence, these studies are more theoretical than ours.

In the area of latch timing, Cotten [COT65] developed some basic timing constraints, and looked at maximum clocking rates in 1965. Hallin and Flynn [HAL72] developed timing constraints for the Earle latch (to be described later) without considering clock skew or data skew. Later Fawcett [FAW75] expanded these timing constraints to account for these skews. A summary of Fawcett's work is given later along with additional analysis.

Some of the practical limitations on pipelining caused by control paths are discussed in [AND67]. There is also some interesting discussion about design of highly-overlapped computer systems in [THO70].

## 1.5. Paper Overview

Section 2 describes the simulation method and pertinent assumptions. Section 3 is a simulation study of the importance of instruction dependencies; latch overhead is neglected. Section 4 discusses latch overhead due to propagation delays in latches. Latch designs that minimize latch propagation delays are discussed. Section 5 discusses latch overhead due to data skew; the clock is assumed to be perfectly controlled. Unintentional clock skew is studied in Section 6, and there is a further simulation study with clock skew included. Section 7 contains a summary and conclusions.

## 2. Simulation Method

We use the CRAY-1S, CFT FORTRAN Compiler (version 111g) and the first 14 Lawrence Livermore Loops [MCM72]. The CFT compiler is a mature compiler that accurately represents the state-of-the-art, at least for the specific workload we are using. The Lawrence Livermore Loops are chosen because they are small enough to be simulated for a large number of cases, are representative of a class of real programs, and are widely used for comparing the performance of pipelined computer systems. Furthermore, by using a set of standard benchmarks with a well-documented computer, our experiments are repeatable by other researchers. The simulator used is described in [PAN83].

In the initial part of our study, we neglect latch overhead. We first calculate the total lengths of the various pipelines in the CRAY-1S, measured in gate levels. These base numbers are derived using the fact that there are eight levels of gates between latches in the CRAY-1S. The pipeline lengths in clock periods are given in the CRAY-1S hardware reference manual [CRA79].

If a pipelined operation requires $n$ clock periods, we estimate that between $8n$ and $8(n-1)+1=8n-7$ gate levels are needed. The minimum number comes about because if one fewer gate level is used, i.e. $8(n-1)$, then one fewer pipeline segment can be used. For example, the floating point adder in the CRAY-1S is six clock periods long and the number of gates per clock period is eight. Therefore the maximum number of gate levels possible is 48 and a reasonable minimum is 41. Because there is an uncertainty in the number of gate levels, it is possible to calculate a range of performance levels. Continuing with the example of the floating point adder, suppose the clock frequency is changed so that there are four gate levels in each segment. Using the maximum number of gate levels, 48, the new adder has 12 segments. But, using the minimum number of gates levels, 41, the adder has only 11 segments. In our study, we simulated both endpoints of the range to yield a maximum and minimum performance.

We simulated pipeline segments varying in length from two gate levels to 16 gate levels at even numbered intervals and 32 gate levels. In all the simulations, we used code exactly as it is scheduled for the CRAY-1S. Because the ratios of the pipeline lengths remain the same, the code schedule is of the same quality in virtually all the cases. The only exceptions occur because of scheduling conflicts involving the result buses that feed into the register files. In these very few cases, some of the code runs slightly faster when a larger number of gate levels is used (giving a slightly longer pipeline) than when a smaller number of gate levels is used.

Seven of the 14 Lawrence Livermore loops are vectorized by the CFT compiler. Because the operations that flow down a vector pipeline are by definition independent, one would expect significantly better performance on vector code than on scalar code. For this reason we give separate performance numbers for the vectorizable loops, the scalar loops, and the combination.

To measure performance, first the computation rate in Millions of Floating Point Operations per Second (MFLOPS) was generated for each loop. These numbers were used to compute the harmonic mean MFLOPS. The harmonic mean was chosen because it is a much more meaningful measure than the more common arithmetic mean [WOR84].

Informally, to compute the harmonic mean one should first compute the time, $T_i$, it takes to execute exactly $F$ ($F$ can be arbitrarily chosen) floating point operations in each of the loops $L_i$. Then the harmonic mean performance for $m$ loops is $\frac{mF}{\sum_{i=1}^{m} T_i}$. If the MFLOPS rate for loop $i$ is $M_i$, it can be shown that the harmonic mean also equals $\frac{m}{\sum_{i=1}^{m} \frac{1}{M_i}}$.

## 3. Instruction Dependencies

An instruction may be dependent on an earlier instruction for either data or control. A data dependency occurs, for example, if one instruction computes a result that is used as an input operand by another. A control dependency occurs in the case of a conditional branch where the execution of an instruction following the branch is dependent on the branch outcome. More complete discussions of the types of instruction dependencies that can occur are given in [KUC78].

Instruction dependencies limit the efficiency of a pipelined system. For example, in a straightforward pipeline design if two consecutive instructions passing down the pipeline are dependent, then the second must wait for the first to complete before the second can begin. Pipelining is wasted because instruction execution is not overlapped and pipeline segments sit idle.

The effects of dependencies can be handled to some extent by the compiler, and/or clever pipeline design. For example, the compiler can schedule the instructions so that consecutive instructions tend to be independent. There are limits, however, to the number of independent instructions that can be found. Conditional branches pose a particular problem because the possibilities for scheduling them are much more limited than for other instructions.

For a high degree of pipelining, the theoretical peak throughput is increased, but the pipeline may be used inefficiently because there are not enough independent instructions to keep it full. Conversely, if there is a low degree of pipelining, it is relatively easy to keep the pipeline full, but its peak throughput is reduced. This tradeoff is of course dependent on the problem, algorithm, program, and even the programming language. Hence, we study instruction dependencies through simulation with realistic computers and programs.

### 3.1. Simulation Results

The results of the first set of simulations which neglect latch overhead are shown in Table 1. The results are all normalized to the performance for the combined loops when eight gate levels are used (the same as in the CRAY-1S). In the tables and graphs that follow there is a minimum and maximum performance given that bounds the range of possible performances. This arises because of uncertainty in the number of pipeline segments as described in section 2. Obviously, the maximum performance is achieved when the minimum number of pipeline segments is used for a given number of gate levels per segment.

The data in Table 1 shows that the performance for scalar and combined code is significantly less than the performance that would be

| gate levels per segment | 7 scalar loops | | 7 vector loops | | 14 combined loops | |
|---|---|---|---|---|---|---|
| | max | min | max | min | max | min |
| 2 | 0.78 | 0.67 | 11.90 | 11.20 | 1.47 | 1.27 |
| 4 | 0.70 | 0.65 | 7.03 | 6.90 | 1.28 | 1.18 |
| 6 | 0.65 | 0.59 | 5.04 | 4.94 | 1.15 | 1.05 |
| 8 | 0.57 | 0.57 | 3.88 | 3.88 | 1.00 | 1.00 |
| 10 | 0.55 | 0.52 | 3.18 | 3.15 | 0.93 | 0.89 |
| 12 | 0.50 | 0.47 | 2.68 | 2.66 | 0.84 | 0.80 |
| 14 | 0.45 | 0.43 | 2.33 | 2.30 | 0.76 | 0.73 |
| 16 | 0.40 | 0.40 | 2.04 | 2.04 | 0.66 | 0.66 |
| 32 | 0.23 | 0.23 | 0.94 | 0.94 | 0.37 | 0.37 |

Table 1. Normalized performance with no latch overhead.

theoretically predicted if there were no data dependencies. For example, the performance with two gate levels should theoretically be four times the performance with eight gate levels. However, the results show a maximum increase in performance of 23% over this interval. This deviation from the theoretical peak becomes larger as the number of gate level becomes smaller. The gain achieved in going from eight to four gates per segments is only about two thirds of that achieved in going from sixteen to eight gates per segment. Performance for the seven loops that vectorize does not deviate as significantly from the theoretical as it does for the scalar loops.

For vector code, the performance increases almost linearly with the increasing clock frequency. When the scalar loops and vector loops are combined the curve more closely resembles that of scalar loops by themselves because of the well-known dominance of the slower code [WOR81].

## 4. Latch Propagation Delay

This section is the first of three that study latch overhead. We separate the three components of latch overhead in order to measure the contribution of each to performance degradation. Our discussion concentrates on latch designs that minimize the effects of latch propagation delay.

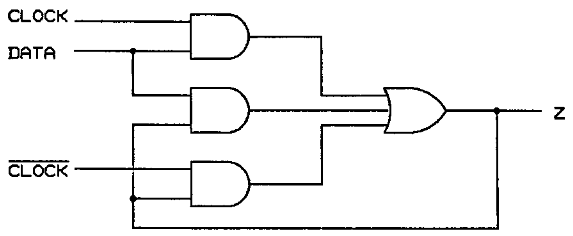
### 4.1. The Earle Latch

Latch propagation delay occurs in gates used to construct latches. A distinguishing characteristic of propagation delay is that it would be present even with skewless gates, i.e. all gates have exactly the same propagation delay. A latch typically has a propagation delay from clock to output of at least two gate delays. While this may be insignificant in some systems, it is a major concern in pipelined systems when the clock period becomes very short. To reduce its significance, J. G. Earle introduced a latch that was used in carry-save adders for the IBM360/91 [EAR65]. The so-called Earle latch can be used for any combinational logic function, however, not just carry-save addition.

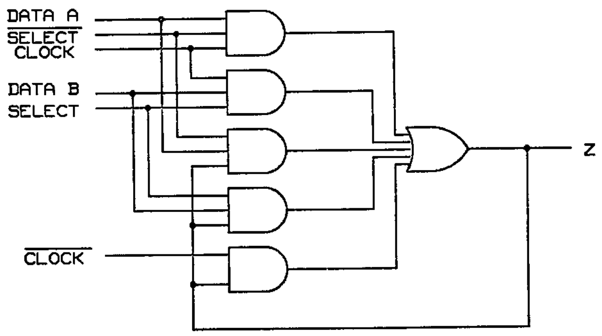
Fig. 1a is a simple Earle latch; the NAND gate equivalent is, of course, more commonly used in practice. This latch performs two levels of useful logic as well as the latching function. There is no added propagation delay beyond that needed to perform the useful function. This enables the overhead due to propagation delay through latches to be essentially eliminated. As an example of the way a combinational function can be built into an Earle latch, Fig. 1b is a 2-to-1 multiplexer/latch.

### 4.2. The Polarity Hold Latch

The polarity hold latch is a simplified form of the Earle latch (see Fig. 2). It does not use the center AND gate that protects against a logic hazard. This logic hazard can be avoided by intentionally skewing the $C$ and $\overline{C}$ signals properly. In practice, this reduction in gate usage is attractive not only because it is cheaper but also because it reduces fan-in at the output gate. Polarity hold latches are used in



a) Basic Earle latch



b) Earle latch with built-in multiplexer

Fig. 1. Earle latches

pipelined CPUs designed at CDC, Cray Research, and the Amdahl Corporation. As we shall see, however, clock skew constraints are tighter for polarity hold latches than for Earle latches in order to avoid logic hazards.

Because Earle and polarity hold latches can eliminate propagation delay, we assume their use throughout the rest of this paper. Other causes of latch overhead are examined in the context of Earle and polarity hold latches.

## 5. Data Skew

Data skew occurs because gates used to implement latches and combinational logic between latches have different propagation delays. In order to isolate the effects of data skew, we assume the logic used for clock fanout has no unintentional skew due to gate delay differences. That is, clock signals reaching the latches are perfectly controlled.

The distinction we are making between intentional skew and unintentional skew should be carefully noted. It is often necessary to intentionally skew clock signals. This is done in the case of multiphase clocks, and in controlling the $C$ and $\overline{C}$ signals to avoid hazards. Unintentional skew is due to imperfections in the clock distribution network and fanout logic. While unintentional clock skew is ignored in this section, it is the primary topic of the next section.

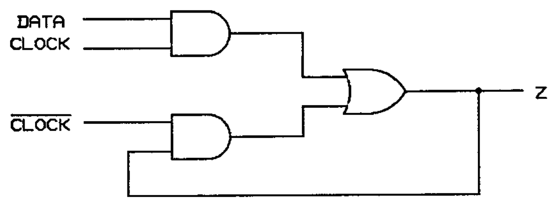


Fig. 2. Polarity hold latch

A basis for the analysis given here for the Earle latch is presented in [FAW75]. As shown in Fig. 1, the Earle Latch actually has two clock signals, one is the complement of the other. Our analysis assumes that the width of the $C_{high}$ pulse is the same as the width of the $\overline{C}_{low}$ pulse.

We now introduce Fawcett's terminology.

| | |
|---|---|
| $t_{\max}$ | - maximum propagation delay time for a logic gate (NAND gate) |
| $t_{\min}$ | - minimum propagation delay time for a logic gate (NAND gate) |
| $C_{high}$ | - the duration of C = 1 (clock high) |
| $C_{low}$ | - the duration of C = 0 (clock low) |
| $P_{\max}$ | - maximum delay time on the maximum delay path from the output of a latch to input of next latch. This does not include any delay for the latch itself. |
| $P_{\min}$ | - minimum delay time on the minimum delay path from the output of a latch to input of next latch. |
| S(X,Y) | - This quantity is the skew between the edges X and Y and is always positive. It is the difference between the arrival of edge X and edge Y. If Y arrives before X, the quantity is zero. |
| AS(X,Y) | - This quantity is the algebraic skew between the edges X and Y and can be positive or negative. It is the difference between the arrival of edge X and edge Y. If Y arrives before X, the quantity is negative. |
| $U_X$ | - This quantity is the uncertainty of X. It represents the quantity such that X lies in the interval $(X - U_X, X + U_X)$. |

## 5.1. Timing Constraints on Earle Latches

Repeating the work of Fawcett, we give four timing constraints on the clock signal. These constraints are discussed only briefly; we regret that space does not permit a more complete explanation of Fawcett's work. The first constraint is on the minimum width of the clock pulse $C_{high}$. The pulse has to be wide enough to ensure that valid data is stored in the latch. That is, it must be wide enough to allow data to propagate from the D input to the latch output, then to feed back around to be latched. This gives

$$C_{high} \geq 3t_{\max} - t_{\min} + S(C_{rise}, \overline{C}_{fall}). \tag{1}$$

The second timing constraint is on the maximum width of the clock pulse $C_{high}$. Obviously, the clock width must be shorter than the minimum propagation delay from the input of one latch to the input of the next, assuming no intentional clock skew. Two other terms must be added to allow for skewed clocks. The precise constraint is

$$\begin{aligned} C_{high} + AS(C_{i-1,rise}, C_{i,rise}) \leq 2t_{\min} + P_{\min} - S(\overline{C}_{fall}, C_{rise}) \\ - \max[0,\, t_{\max} - t_{\min} + AS(C_{rise}, \overline{C}_{fall})]. \end{aligned} \tag{2}$$

Fawcett's third constraint is on the minimum clock period, $C_{high} + C_{low}$. This constraint arises because the latch can not be clocked until the data from the previous latch has arrived. Thus the minimum clock period must be longer than the maximum propagation delay from the input of one latch to the input of the following latch. The complete expression derived by Fawcett is

$$C_{high} + C_{low} + AS(C_{i-1,rise}, C_{i,rise}) \geq 2t_{\max} + P_{\max} + S(C_{rise}, \overline{C}_{fall}) \tag{3}$$

Fawcett combined the three constraints to arrive at an alternative lower bound on the clock period:

$$\begin{aligned} C_{high} + C_{low} \geq\ & P_{\max} - P_{\min} + 5t_{\max} - 3t_{\min} \\ & + 2S(C_{rise}, \overline{C}_{fall}) + S(\overline{C}_{fall}, C_{rise}) \\ & + \max[0,\, t_{\max} - t_{\min} + AS(C_{fall}, \overline{C}_{rise})]. \end{aligned} \tag{4}$$

This concludes our initial summary of Fawcett's work. What follows in this subsection and in succeeding subsections are extensions and further analysis of the timing constraints.

Fawcett's work allows for the possibility of multi-phase clocks. We will consider both single and multi-phase clocks. The pipelined computers produced by Cray Research, CDC, and Amdahl all use single phase clocks. We do not consider $C$ and $\overline{C}$ going to a single latch as a multi-phase clock. In a typical supercomputer system, a single clock waveform is generated and distributed to the logic chips (modules in the case of the CRAY-1S). On each chip, there is a "clock shaper" circuit that, among other things, produces both the $C$ and $\overline{C}$ signals.

If we restrict ourselves to an ideal, single phase-clock then $C_{i-1,rise} = C_{i,rise}$. Also, (3) gives a minimum clock period if the clock signals can be controlled so that $\overline{C}_{fall}$ precedes $C_{rise}$, making the last term zero. Thus for an ideal, single phase clock we can derive a lower bound on the clock period using (3).

**Clock Period Bound: Earle Latch, Single Phase Clock**

$$C_{high} + C_{low} \geq 2t_{\max} + P_{\max}, \quad provided\ AS(C_{rise}, \overline{C}_{fall}) \leq 0. \tag{5}$$

For a multi-phase clock, we can derive a lower bound on the clock period using (4).

**Clock Period Bound: Earle Latch, Multi-Phase Clock**

$$C_{high} + C_{low} \geq P_{\max} - P_{\min} + 6t_{\max} - 4t_{min}, \tag{6}$$

$$provided\ -(t_{\max} - t_{\min}) \leq AS(C_{fall}, \overline{C}_{rise}) \leq 0$$

## 5.2. Timing Constraints on Polarity Hold Latches

Because the polarity hold latch is an Earle latch with the hazard gate removed, the clock signals must be intentionally skewed to ensure proper operation of the latch. This additional constraint can be expressed as

$$AS(C_{rise}, \overline{C}_{fall}) \leq -(t_{\max} - t_{\min}). \tag{7}$$

Applying (7) makes some of the terms in (1), (2), and (3) zero, simplifying them to (8), (9), and (10) given below.

$$C_{high} \geq 3t_{\max} - t_{min.} \tag{8}$$

$$C_{high} + AS(C_{i-1,rise}, C_{i,rise}) \leq 2t_{\min} + P_{\min} - S(\overline{C}_{fall}, C_{rise}). \tag{9}$$

$$C_{high} + C_{low} + AS(C_{i-1,rise}, C_{i,rise}) \geq 2t_{\max} + P_{\max} \tag{10}$$

For an ideal, single phase clock $C_{i-1,rise} = C_{i,rise}$, which reduces (10) to the following lower bound on the clock period.

**Clock Period Bound: Polarity Hold Latch, Single Phase Clock**

$$C_{high} + C_{low} \geq 2t_{\max} + P_{\max}, \tag{11}$$

$$provided\ AS(C_{rise}, \overline{C}_{fall}) \leq -(t_{\max} - t_{\min}).$$

For the case in which a multi-phase clock is used, the same optimization technique that Fawcett used can be performed on (8), (9), and (10) yielding,

$$C_{high} + C_{low} \geq P_{\max} - P_{\min} + 5t_{\max} - 3t_{\min} + S(\overline{C}_{fall}, C_{rise}). \tag{12}$$

Considering (7), (12) can be reduced to a lower bound clock period for multi-phase clocks.

**Clock Period Bound: Polarity Hold Latch, Multi-Phase Clock**

$$C_{high} + C_{low} \geq P_{\max} - P_{\min} + 6t_{\max} - 4t_{min,} \tag{13}$$

$$provided\ AS(C_{rise}, \overline{C}_{fall}) \leq -(t_{\max} - t_{\min}).$$

We have shown that the Earle latch and the polarity hold latch have the same lower bound on the clock period for both single and multiple phase clock. The difference is that the constraint on the intentional clock skew is more restrictive for the polarity hold latch if the bound is to be met. This implies that more careful control over the clock signals is required.

## 5.3. Latch Overhead

We are now ready to analyze the latch overhead due to data skew. We emphasize single phase clocks with polarity hold latches because they are commonly used in practice. Some discussion of multi-phase clocks and Earle latches is also included, however.

Let $n$ be the number of gate levels between latches (this does not include the two gate levels in the latch, itself). Considering that useful logic can be performed in the latches, there are a total of $n+2$ useful gate levels in each pipeline segment. We assume an equal number of gate levels on all paths. The generalization to different length paths is straightforward. Equal length paths tend to allow shorter clock periods, however. For example, note the importance of $P_{max} - P_{min}$ in (4).

Let $r$ be the ratio of the minimum gate delay to the maximum. Then $t_{min} = rt_{max}$. For simplicity, we begin by assuming all delays are concentrated in gates, not in wires between gates. In practice, one can add wire delay following a gate to the gate, itself. Later, when we need to intentionally add extra delay to paths, we do consider separate wire delays.

Because delays are assumed to be in the gates, $P_{max} = nt_{max}$, and $P_{min} = nt_{min} = nrt_{max}$.

### 5.3.1. Polarity Hold Latch; Single Phase Clock

For a single phase clock, (11) becomes

$$C_{high} + C_{low} \ge (n+2)t_{max} \qquad (14)$$

For a single phase clock, however, any clock period lower bound is subject to a lower bound on $P_{min}$ that comes from combining (8) and (9) to eliminate $C_{high}$. The resulting inequality can be rearranged to yield $P_{min} \ge 3t_{max} - 3t_{min} + S(\overline{C}_{fall}, C_{rise})$. Further restricting the condition that gives the lower bound clock period as determined in (11) to $AS(\overline{C}_{fall}, C_{rise}) = t_{max} - t_{min}$, yields

$$P_{min} \ge 4(t_{max} - t_{min}). \qquad (15)$$

Substituting $rt_{max}$ for $t_{min}$ and $nrt_{max}$ for $P_{min}$ and rearranging yields

$$n \ge \frac{4}{r} - 4 \qquad (16)$$

That is, for a single phase clock there must be at least $n \ge \frac{4}{r} - 4$ gate levels for proper operation. If the number of gate levels is fewer, then some extra intentional delay pad must be added. This can either be in the form of gates or wires.

### 5.3.2. Delay Padding

If intentional delay is added with gates, then $\frac{4}{r} - 4 - n$ gate delays must be added to the path so that (16) is satisfied. This yields:

$$\begin{aligned} C_{high} + C_{low} &\ge (n+2)t_{max} + (\frac{4}{r} - 4 - n)t_{max} \\ &\ge \frac{4}{r}t_{max} - 2t_{max} \end{aligned} \qquad (17)$$

In the first line of (17), the $(n+2)t_{max}$ term accounts for the delay for performing useful logic; two levels in the latch and $n$ levels between latches. The $(\frac{4}{r} - 4 - n)t_{max}$ term is overhead. Also note that the final form of (17) is independent of $n$. Thus, if gates are used for delay padding, there is no advantage to using fewer than $\frac{4}{r} - 4$ gates between latches.

We now consider using wire delays for padding. First consider that without padding $P_{min}$ is $nrt_{max}$. $P_{min}$ must be at least $4(t_{max} - rt_{max})$, however. Hence, an additional pad of $4(t_{max} - rt_{max}) - nrt_{max}$ must be added. This yields

$$\begin{aligned} C_{high} + C_{low} &\ge (n+2)t_{max} + 4(t_{max} - rt_{max}) - nrt_{max} \\ &\ge (n+2)t_{max} + (4 - (n+4)r)t_{max} \end{aligned} \qquad (18)$$

In (18) the wire delay overhead is $(4 - (n+4)r)t_{max}$. We now compare this overhead with that in (17). The gate pad overhead is easily shown to be $\frac{1}{r}$ times the wire pad overhead. By definition $\frac{1}{r} \ge 1$. Hence, a wire delay pad potentially leads to a shorter clock period than a gate pad.

### 5.3.3. Polarity Hold Latch; Multi-phase Clock

For a multi-phase clock, (13) easily reduces to (18). For multi-phase clocks, (18) holds for all values of $n$. When $n \le \frac{4}{r} - 4$ the "overhead" term in (18) is positive, and the minimum clock period is the same for both a single and multi-phase clock. Hence, for very short clock periods, there is no performance advantage to using a multi-phase clock.

On the other hand, when $n > \frac{4}{r} - 4$ the "overhead" term in (18) is negative, and the multiphase clock period can be less than the single phase clock period. Note that for the case of a multi-phased clock there is no limit on $P_{min}$ because the phase of the clocks can be shifted using (2) and (9) to accommodate any value of $P_{min}$.

### 5.3.4. Earle Latches

A similar line of reasoning to the above can be used to calculate a minimum for $P_{min}$ in the Earle latch using (1) and (2). This also yields (14) through (18), provided $-(t_{max} - t_{min}) \le AS(C_{fall}, \overline{C}_{rise}) \le 0$ which is slightly more restricting than the minimum condition for (3).

## 5.4. Simulation Results

We now use simulation to evaluate the performance of a single phase clock and polarity hold latches under some realistic assumptions. A survey of the currently available, high performance TTL and ECL parts [MOT82, FAI84] indicates values of $r$ in the range of .3 to .4. It is quite possible that $r$ could be increased by screening of the chips, however. Also wire delays added to gate outputs tend to increase the effective value of $r$. A survey of ECL gate arrays, [FAI85, APP85] indicates larger values of $r$, up to .6. Hence a realistic value for our simulations is $r$ =.5.

The simulation results with data skew overhead added are given in Table 2. The performance results are again normalized with respect the combined loop performance with eight useful gate levels. These results show the best scalar and combined performance occurs at about six useful gates levels per pipeline segment. A substantial performance improvement can be achieved by increasing the clock frequency on vectors. No peak appears as the performance seems to continue to increase as the clock frequency increases.

## 6. Unintentional Clock Skew

The results in the previous section are based on an assumption of a perfectly controlled clock. When designing real processors, however, there is always some uncertainty in the clock. To model this, it is necessary to add uncertainty terms to the clocking constraints.

As done by Fawcett, we assume there can be unintentional skew between the $C$ and $\overline{C}$ signals (denoted as $U_{C,\overline{C}}$ ) and between clock signals reaching different latches ($U_{C_{i-1},C_i}$). Expressions with

| useful gates levels per segment | 7 scalar loops | | 7 vector loops | | 14 combined loops | |
|---|---|---|---|---|---|---|
| | max | min | max | min | max | min |
| 2 | 0.39 | 0.34 | 5.95 | 5.60 | 0.74 | 0.64 |
| 4 | 0.56 | 0.52 | 5.62 | 5.52 | 1.02 | 0.95 |
| 6 | 0.65 | 0.59 | 5.04 | 4.94 | 1.15 | 1.05 |
| 8 | 0.57 | 0.57 | 3.88 | 3.88 | 1.00 | 1.00 |
| 10 | 0.55 | 0.52 | 3.18 | 3.15 | 0.93 | 0.89 |
| 12 | 0.50 | 0.47 | 2.68 | 2.66 | 0.84 | 0.80 |
| 14 | 0.45 | 0.43 | 2.33 | 2.30 | 0.76 | 0.73 |
| 16 | 0.40 | 0.40 | 2.04 | 2.04 | 0.66 | 0.66 |
| 32 | 0.23 | 0.23 | 0.94 | 0.94 | 0.37 | 0.37 |

Table 2. Normalized performance with data skew overhead.

uncertainty terms added are derived in a manner very similar to (1) through (18). For brevity, we give only the most significant ones. A complete set of expressions is given in the appendix.

### 6.1. Basic Clocking Constraints

The uncertainties not only affect the clock inequalities, but they affect the intentional clock skew constraints that give minimum clock periods. The four lower bound expressions follow.

**Clock Period Bound: Earle Latch, Single Phase Clock**

$$C_{high} + C_{low} \geq 2t_{\max} + P_{\max} + U_{C_{i-1},C_i} \qquad (5^*)$$

$$\text{provided } AS(C_{rise}, \bar{C}_{fall}) \leq -U_{C,\bar{C}} \ .$$

**Clock Period Bound: Earle Latch, Multi-Phase Clock**

$$C_{high} + C_{low} \geq P_{\max} - P_{\min} + 6t_{\max} - 4t_{\min} + 2U_{C_{i-1},C_i} + 2U_{C,\bar{C}} \ , \qquad (6^*)$$
$$\text{provided } -(t_{\max} - t_{\min}) - U_{C,\bar{C}} \leq AS(C_{rise}, \bar{C}_{fall}) \leq -U_{C,\bar{C}}$$

**Clock Period Bound: Polarity Hold Latch, Single Phase Clock**

$$C_{high} + C_{low} \geq 2t_{\max} + P_{\max} + U_{C_{i-1},C_i} \ , \qquad (11^*)$$

$$\text{provided } AS(C_{rise}, \bar{C}_{fall}) \leq -(t_{\max} - t_{\min}) - U_{C,\bar{C}}$$

**Clock Period Bound: Polarity Hold Latch, Multi-Phase Clock**

$$C_{high} + C_{low} \geq P_{\max} - P_{\min} + 6t_{\max} - 4t_{\min} + 2U_{C_{i-1},C_i} + 2U_{C,\bar{C}} \ , \qquad (13^*)$$
$$\text{provided } AS(C_{rise}, \bar{C}_{fall}) = -(t_{\max} - t_{\min}) - U_{C,\bar{C}}$$

As before, the Earle latch and the polarity hold latch have the same lower bound on the clock period for both single and multiple phase clock. The only difference is in the required constraint on the intentional clock skew.

#### 6.1.1. Latch Overhead due to Unintentional Clock Skew

Once again, we assume $n$ gates between latches, and a ratio $t_{\min}$ to $t_{\max}$ of $r$. This gives $P_{\max} = nt_{\max}$, and $P_{\min} = nt_{\min} = nrt_{\max}$. We begin with a single phase clock and polarity hold latch.

For a single phase clock,

$$C_{high} + C_{low} \geq (n+2)t_{\max} + U_{C_{i-1},C_i} \qquad (14^*)$$

When $AS(C_{fall}, \bar{C}_{rise}) = -(t_{\max} - t_{\min}) - U_{C,\bar{C}}$ , a lower bound on $P_{\min}$ is

$$P_{\min} \geq 4(t_{\max} - t_{\min}) + U_{C_{i-1},C_i} + 2U_{C,\bar{C}} \ . \qquad (15^*)$$

Substituting $rt_{\max}$ for $t_{\min}$ and $nrt_{\max}$ for $P_{\min}$ and rearranging yields

$$n \geq \frac{4}{r} - 4 + \frac{U_{C_{i-1},C_i} + 2U_{C,\bar{C}}}{rt_{\max}} \qquad (16^*)$$

Equation (16*) bounds the number of gate levels necessary for proper operation with a single phase clock. As before, if the number of gate levels is fewer, then some extra intentional delay pad in the form of either gates or wires must be added.

If intentional delay is added with gates, then

$$C_{high} + C_{low} \geq \frac{4}{r}t_{\max} - 2t_{\max} + \frac{(r+1)}{r}U_{C_{i-1},C_i} + \frac{2U_{C,\bar{C}}}{r} \qquad (17^*)$$

Equation (17*) is independent of $n$. Thus, if gates are used for delay padding, there is no advantage to using fewer than indicated by (16*).

If wire delays are used for padding, then

$$C_{high} + C_{low} \geq \ (n+2)t_{\max} + (4 - (n+4)r)t_{\max} + 2U_{C_{i-1},C_i} + 2U_{C,\bar{C}} \qquad (18^*)$$

As we did earlier, it can be shown that a wire delay pad leads to a shorter clock period than a gate pad.

To determine the overhead due to unintentional clock skew, we compare the above results with those in the previous section. When $n \geq \frac{4}{r} - 4 + \frac{U_{C_{i-1},C_i} + 2U_{C,\bar{C}}}{rt_{\max}}$, (14*) indicates that unintentional clock skew adds additional overhead of $U_{C_{i-1},C_i}$. When $n < \frac{4}{r} - 4 + \frac{U_{C_{i-1},C_i} + 2U_{C,\bar{C}}}{rt_{\max}}$, (18*) indicates that unintentional clock skew adds an additional overhead of $2U_{C_{i-1},C_i} + 2U_{C,\bar{C}}$.

For a multi-phase clock, the clock period bound is the same as in (18*). This means that the minimum clock period is the same for both a single and multi-phase clock when

$$n \leq \frac{4}{r} - 4 + \frac{U_{C_{i-1},C_i} + 2U_{C,\bar{C}}}{rt_{\max}}.$$

Otherwise, the multi-phase clock may be faster.

For the Earle latch, the same results as above can be derived. As has typically been the case, however, the intentional clock skew constraints are reduced. In particular, minimum clock periods are achieved when $-(t_{\max} - t_{\min}) - U_{C,\bar{C}} \leq AS(C_{fall}, \bar{C}_{rise}) \leq -U_{C,\bar{C}}$.

### 6.2. Simulation Results

We continue our simulations to determine the effect of unintentional clock skew under realistic conditions. To do this, we must estimate values of unintentional skews. We first observe that $U_{C,\bar{C}}$ is relatively easy to control, because both signals feeding a particular latch are formed on the same printed circuit module (CRAY-1S) or on the same chip (CYBER205) from a single master clock signal.

If one gate is used to generate $\bar{C}$, then

$$U_{C,\bar{C}} = \frac{1}{2}(t_{\max} - t_{\min}).$$

When $r = .5$, $U_{C,\bar{C}} = .25t_{\max}$.

If two logic levels are used to fanout the clock, then

$$U_{C_{i-1},C_i} = 2t_{\max} - 2t_{\min}.$$

When $r = .5$, the uncertainty becomes $U_{C_{i-1},C_i} = t_{\max}$. If the fanout logic is extended to four levels, the uncertainty doubles and $U_{C_{i-1},C_i} = 2t_{\max}$.

The performance when clock uncertainty is added to the latch overhead is shown in Tables 3 and 4. Comparing these results with those obtained without unintentional clock skew shows that peak performance moves to a larger number of gate levels per segment. Also vectors alone now have a peak; they do not continue to increase as the number of gate levels decreases. As expected, for a larger uncertainty, a larger number of gate levels per segment is needed to achieve peak performance. This can be seen by comparing the two-level fanout results to the four-level fanout results. For the two-level clock fanout, scalar code has the best performance at eight to ten gate levels per segment while vectors peak at four gate levels per segment. Scalars dominate when the two are combined to give a peak at eight to ten gate

| useful gate levels per segment | 7 scalar loops | | 7 vector loops | | 14 combined loops | |
|---|---|---|---|---|---|---|
| | max | min | max | min | max | min |
| 2 | 0.29 | 0.25 | 4.35 | 4.09 | 0.54 | 0.46 |
| 4 | 0.45 | 0.41 | 4.45 | 4.37 | 0.81 | 0.75 |
| 6 | 0.55 | 0.49 | 4.22 | 4.14 | 0.97 | 0.88 |
| 8 | 0.57 | 0.57 | 3.88 | 3.88 | 1.00 | 1.00 |
| 10 | 0.59 | 0.56 | 3.43 | 3.40 | 1.01 | 0.96 |
| 12 | 0.55 | 0.51 | 2.93 | 2.92 | 0.92 | 0.87 |
| 14 | 0.50 | 0.48 | 2.58 | 2.55 | 0.84 | 0.81 |
| 16 | 0.44 | 0.44 | 2.28 | 2.28 | 0.74 | 0.74 |
| 32 | 0.27 | 0.27 | 1.08 | 1.08 | 0.43 | 0.43 |

Table 3. Normalized performance with data skew and unintentional clock skew; two-level clock fanout.

| useful gate levels per segment | 7 scalar loops | | 7 vector loops | | 14 combined loops | |
|---|---|---|---|---|---|---|
| | max | min | max | min | max | min |
| 2 | 0.27 | 0.23 | 4.03 | 3.79 | 0.50 | 0.43 |
| 4 | 0.43 | 0.39 | 4.25 | 4.18 | 0.78 | 0.72 |
| 6 | 0.53 | 0.48 | 4.14 | 4.06 | 0.95 | 0.86 |
| 8 | 0.57 | 0.57 | 3.88 | 3.88 | 1.00 | 1.00 |
| 10 | 0.63 | 0.60 | 3.65 | 3.62 | 1.07 | 1.02 |
| 12 | 0.62 | 0.58 | 3.30 | 3.28 | 1.04 | 0.98 |
| 14 | 0.57 | 0.54 | 2.93 | 2.90 | 0.95 | 0.92 |
| 16 | 0.50 | 0.50 | 2.61 | 2.61 | 0.85 | 0.85 |
| 32 | 0.31 | 0.31 | 1.27 | 1.27 | 0.50 | 0.50 |

Table 4. Performance with data skew and unintentional clock skew; four-level clock fanout.

levels per segment. The same is true for a four-level clock fanout which peaks at ten gate levels per segment for both scalars and scalars combined with vectors.

## 7. Summary and Conclusions

Our first set of simulations with no latch overhead show that data dependencies alone place rather severe limits on pipeline performance. Performance always improves as pipeline segments are shortened, however.

When data skew is considered, our analysis shows that multiphase and single phase clocks have the same lower bound clock periods when few gate levels are used. For longer pipeline segments, however, multiphase clock periods can be shorter than single phase clock periods. Earle and polarity hold latches give the same lower bound clock periods, but constraints on clock skew to achieve these clock periods are tighter with polarity hold latches.

When pipeline segments are extremely short, it becomes necessary to pad with intentional delay. For this purpose, we have shown that wire pads lead to better performance than gate pads.

Our simulation results with realistic data skews added to our simulation model show that overall performance peaks at about six gate levels per pipeline segment. When unintentional clock skew is added to the model, overall performance peaks at eight to ten gate levels per segment.

To get an idea of the relative importance of the primary causes of latch overhead, we have combined the data in Tables 1, 2, and 3 into graphs; one each for scalar, vector, and combined performance. Originally, each of the tables was normalized with respect to a different data point. Consequently, before combining the results from the different tables, we have first re-normalized all the data with respect to eight gate level performance for the combined loops with all skews included (this value comes closest to a real pipelined computer).

These graphs are Figs. 3, 4, and 5. They clearly show the points of optimal performance we pointed out earlier. They also show that for relatively long pipeline segments, adding data skew does not affect performance. For short pipeline segments, however, its effects become increasingly significant. For short pipeline segments, clock skew adds about the same amount of additional degradation as data skew. For longer segments, there is still some small degradation due to clock skew.

## 8. Acknowledgement

This is material based upon work supported by the National Science Foundation under Grant ECS-8207277.

## 9. References

[AND67] D. W. Anderson, F. J. Sparacio, and R. M. Tomasulo, "The System/360 Model 91: Machine Philosophy and Instruction Handling," *IBM Journal*, Vol. 11, No. 34, pp. 8-24, January 1967.

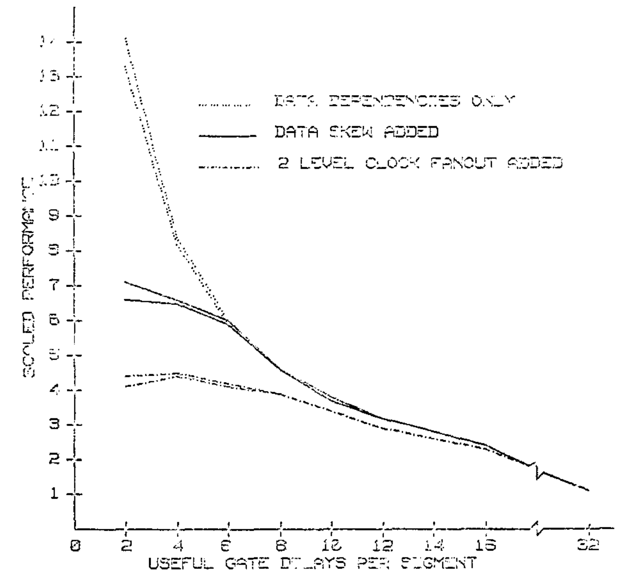


Fig. 4. Performance for seven vector loops

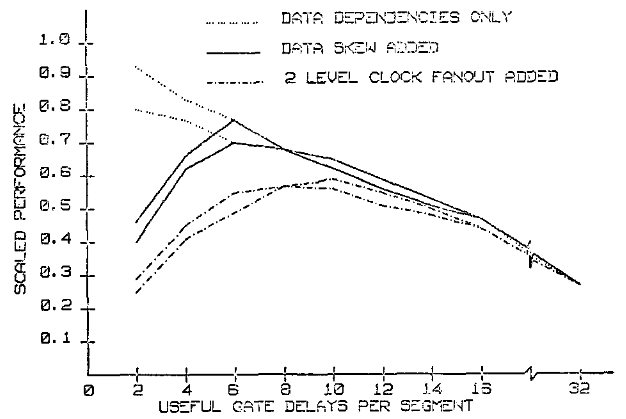


Fig. 3. Performance for seven scalar loops

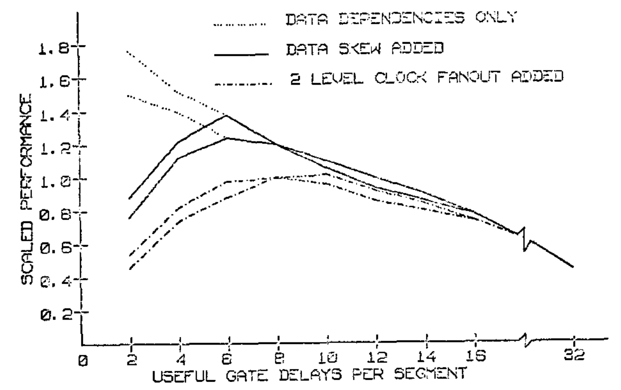


Fig. 5. Performance for fourteen combined loops

[APP85] Applied Micro Circuits Corp., *Q1500 Series Design Guide*, 1985.

[COT65] L. W. Cotten, "Circuit Implementation of High-Speed Pipeline Systems", *AFIPS Fall Joint Computer Conference*, pp. 489-504, 1965.

[CRA79] *CRAY-1 Computer Systems, Hardware Reference Manual*, Cray Research, Inc., Chippewa Falls, WI, 1979.

[EAR65] J. G. Earle, "Latched Carry-Save Adder", *IBM Technical Disclosure Bull.*, Vol. 7, pp. 909-910, March 1965.

[FAI84] Fairchild, *Fast Fairchild Advanced Schottky TTL*, 1984.

[FAI85] Fairchild, *FGE Series Design Manual*, Vol. II, 1985.

[FAW75] B. K. Fawcett, "Maximal Clocking Rates for Pipelined Digital Systems", M.S. Thesis, Dept. Elec. Eng., University of Illinois at Urbana-Champaign, 1975

[FLY66] M. J. Flynn, "Very High-Speed Computing Systems," *Proceedings of the IEEE*, Vol. 54, No. 12, pp. 1901-1909, December 1966.

[FOS72] C.C. Foster and E. M. Riseman, "Percolation of Code to Enhance Parallel Dispatching and Execution", *IEEE Trans. Computers*, Vol. C-21, No. 12, pp. 1411-1415, December 1972.

[HAL72] T.G. Hallin and M. J. Flynn, "Pipelining of Arithmetic Functions", *IEEE Trans. Computers*, Vol. C-21, No. 8, pp. 880-886, August 1972.

[KUC78] D. J. Kuck, *The Structure of Computers and Computations, vol. 1*, John Wiley and Sons, New York, 1978.

[MCM72] F. H. McMahon, "FORTRAN CPU Performance Analysis", Lawrence Livermore Laboratories, 1972.

[MOT82] Motorola, Inc., *MECL Device Data*, 1982.

[NIC84] A. Nicolau, and J. A. Fischer, "Measuring the Parallelism Available for Very Long Instruction Word Architectures," *IEEE*Trans. Vol. C-33, No. 11, November 1984, pp. 968-976.

[PAN83] N. Pang and J. E. Smith, "CRAY-1 Simulation Tools", Tech. Report ECE-83-11, University of Wisconsin-Madison, Dec. 1983.

[RIS72] E. M. Riseman and C. C. Foster, "The inhibition of Potential Parallelism by Conditional Jumps", *IEEE Trans. Computers*, Vol. C-21, No. 12, pp. 1405-1411, December 1972.

[SHA77] H. D. Shapiro, "A Comparison of Various Methods for Detecting and Utilizing Parallelism in a Single Instruction Stream", *1977 International Conference on Parallel Processing* , pp. 67-76, Aug. 1977.

[THO70] J. E. Thornton, *Design of a Computer - The Control Data 6600*, Scott, Foresman and Co., Glenview, IL, 1970.

[TJA70] G. S. Tjaden and M. J. Flynn, "Detection and Parallel Execution of Independent Instructions", *IEEE Trans. Computers*, Vol. C-19, No. 10, pp. 889-895, October 1970.

[WOR81] J. Worlton, "The Philosophy Behind the Machines," *Computer World*, Nov. 9. 1981.

[WOR84] J. Worlton, "Understanding Supercomputer Benchmarks", *Datamation*, Vol. 30, No. 9, pp. 121-130, September 1, 1984.

## Appendix

Complete set of equations with unintentional clock skew.

$$C_{high} \geq 3t_{\max} - t_{\min} + S(C_{rise} + U_{C,\overline{C}}\ , \overline{C}_{fall}) \tag{1*}$$

$$\begin{aligned} C_{high} + AS(C_{i-1,rise}, C_{i,rise}) \leq\ & 2t_{\min} + P_{\min} \\ & - S(\overline{C}_{fall} + U_{C,\overline{C}}\ , C_{rise}) - U_{C_{i-1},C_i} \\ & - \max[0,\ t_{\max} - t_{\min} + U_{C,\overline{C}} \\ & \quad + AS(C_{rise}, \overline{C}_{fall})\ ] \end{aligned} \tag{2*}$$

$$\begin{aligned} C_{high} + C_{low} + AS(C_{i-1,rise}, C_{i,rise}) \geq\ & 2t_{\max} + P_{\max} + U_{C_{i-1},C_i} \\ & + S(C_{rise} + U_{C,\overline{C}}\ , \overline{C}_{fall}) \end{aligned} \tag{3*}$$

$$\begin{aligned} C_{high} + C_{low} \geq\ & P_{\max} - P_{\min} + 5t_{\max} - 3t_{\min} + 2U_{C_{i-1},C_i} \\ & + 2S(C_{rise} + U_{C,\overline{C}}\ , \overline{C}_{fall}) + S(\overline{C}_{fall} + U_{C,\overline{C}}\ , C_{rise}) \\ & + \max[0,\ t_{\max} - t_{\min} + AS(C_{rise}, \overline{C}_{fall}) + U_{C,\overline{C}}\ ] \end{aligned} \tag{4*}$$

$$C_{high} + C_{low} \geq 2t_{\max} + P_{\max} + U_{C_{i-1},C_i} \tag{5*}$$

$$\textit{provided}\ \ AS(C_{rise}, \overline{C}_{fall}) \leq -U_{C,\overline{C}}$$

$$\begin{aligned} & C_{high} + C_{low} \geq P_{\max} - P_{\min} + 6t_{\max} - 4t_{\min} + 2U_{C_{i-1},C_i} + 2U_{C,\overline{C}}\ , \\ & \textit{provided}\ \ -(t_{\max} - t_{\min}) - U_{C,\overline{C}} \leq AS(C_{rise}, \overline{C}_{fall}) \leq -U_{C,\overline{C}} \end{aligned} \tag{6*}$$

$$AS(\overline{C}_{rise}, C_{fall}) \geq t_{\max} - t_{\min} + U_{C,\overline{C}} \tag{7*}$$

$$C_{high} \geq 3t_{\max} - t_{\min} \tag{8*}$$

$$\begin{aligned} C_{high} + AS(C_{i-1,rise}, C_{i,rise}) \leq\ & 2t_{\min} + P_{\min} - U_{C_{i-1},C_i} \\ & - S(\overline{C}_{fall} + U_{C,\overline{C}}\ , C_{rise}) \end{aligned} \tag{9*}$$

$$C_{high} + C_{low} + AS(C_{i-1,rise}, C_{i,rise}) \geq 2t_{\max} + P_{\max} + U_{C_{i-1},C_i} \tag{10*}$$

$$C_{high} + C_{low} \geq 2t_{\max} + P_{\max} + U_{C_{i-1},C_i}\ , \tag{11*}$$

$$\textit{provided}\ \ AS(\overline{C}_{rise}, C_{fall}) \geq t_{\max} - t_{\min} + U_{C,\overline{C}}$$

$$\begin{aligned} C_{high} + C_{low} \geq\ & P_{\max} - P_{\min} + 5t_{\max} - 3t_{\min} \\ & + S(\overline{C}_{fall} + U_{C,\overline{C}}\ , C_{rise}) + 2U_{C_{i-1},C_i} \end{aligned} \tag{12*}$$

$$\begin{aligned} & C_{high} + C_{low} \geq P_{\max} - P_{\min} + 6t_{\max} - 4t_{\min} + 2U_{C_{i-1},C_i} + 2U_{C,\overline{C}}, \\ & \textit{provided}\ \ AS(C_{rise}, \overline{C}_{fall}) = -(t_{\max} - t_{\min}) - U_{C,\overline{C}} \end{aligned} \tag{13*}$$

$$C_{high} + C_{low} \geq (n+2)t_{\max} + U_{C_{i-1},C_i} \tag{14*}$$

$$P_{\min} \geq 4(t_{\max} - t_{\min}) + U_{C_{i-1},C_i} + 2U_{C,\overline{C}} \tag{15*}$$

$$n \geq \frac{4}{r} - 4 + \frac{U_{C_{i-1},C_i} + 2U_{C,\overline{C}}}{rt_{\max}} \tag{16*}$$

$$C_{high} + C_{low} \geq \frac{4}{r}t_{\max} - 2t_{\max} + \frac{(r+1)}{r}U_{C_{i-1},C_i} + \frac{2U_{C,\overline{C}}}{r} \tag{17*}$$

$$\begin{aligned} C_{high} + C_{low} \geq\ & (n+2)t_{\max} - (nr + 4r - 4)t_{\max} \\ & + 2U_{C_{i-1},C_i} + 2U_{C,\overline{C}} \end{aligned} \tag{18*}$$