Washington University in St. Louis

Washington University Open Scholarship

All Computer Science and Engineering
Research

Computer Science and Engineering

Report Number: WUCS-93-32

1993-01-01

Clocked and Asynchronous Instruction Pipelines

Mark A. Franklin and Tienyo Pan

Clocked (synchronous) and self-timed (asynchronous) represent the two principal methodologies associated with timing control and synchronization of digital systems. In this paper, clocked and the asynchronous instruction pipelines are modeled and compared. The approach which yields the best performance is dependent on technology parameters, operating range and pipeline algorithm characteristics. Design curves are presented which permit selection of the best approach for a given application and technology environment.

Follow this and additional works at: https://openscholarship.wustl.edu/cse_research

Part of the [Computer Engineering Commons](#), and the [Computer Sciences Commons](#)

Recommended Citation

Franklin, Mark A. and Pan, Tienyo, "Clocked and Asynchronous Instruction Pipelines" Report Number: WUCS-93-32 (1993). *All Computer Science and Engineering Research*.
https://openscholarship.wustl.edu/cse_research/321

Department of Computer Science & Engineering - Washington University in St. Louis
Campus Box 1045 - St. Louis, MO - 63130 - ph: (314) 935-6160.

CLOCKED and ASYNCHRONOUS INSTRUCTION PIPELINES

Mark A. Franklin
Tienyo Pan

WUCS-93-32

June 1993

Clocked (synchronous) and self-timed (asynchronous) represent the two principal methodologies associated with timing control and synchronization of digital systems. In this paper, clocked and the asynchronous instruction pipelines are modeled and compared. The approach which yields the best performance is dependent on technology parameters, operating range and pipeline algorithm characteristics. Design curves are presented which permit selection of the best approach for a given application and technology environment.

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
St. Louis, MO 63130-4899

CLOCKED and ASYNCHRONOUS INSTRUCTION PIPELINES *

Mark A. Franklin and Tienyo Pan

Computer and Communications Research Center
Washington University
St. Louis, Missouri 63130

jbf@wuccrc.wustl.edu
(314) 935-6107

pan@wuccrc.wustl.edu
(314) 935-8562

1. Introduction

Pipelining is a key element in the design of both high performance vector processors and contemporary RISC processors. This paper examines the design of instruction pipelines based on two alternative control and synchronization methodologies: clocked (synchronous) and self-timed (asynchronous). An analysis is presented which yields the relative performance of instruction pipelines described at the micro (logic design) level. Our goal is to develop a set of general performance models which will permit the designer to determine which synchronization methodology yields the best instruction pipeline throughput for a given set of technology, environmental and design operating ranges, and instruction set execution assumptions. The results indicate where efforts should be focussed to obtain improved performance with either methodology, and also indicate how, within different regions of the parameter space, one or the other methodology has performance advantages.

The performance of synchronous pipelines has been considered by a number of investigators. In addition to the general work by Kogge [1], Hennessy and Patterson [2], and Stone [3], there have been a number of studies which have considered the timing constraints, circuit level performance, and stage optimizations associated with pipeline design [4,5,6,7,8]. These studies begin with a given stage and latch design, present a set of timing constraint equations necessary to ensure correct performance, and then develop a set of technology based performance equations for the given pipeline. Since performance will vary with the number of stages used in the pipeline (due to factors such as clock skew), questions of

* This research has been sponsored in part by funding from the NSF under Grant CCR-9021041.

optimal pipeline design (e.g., optimal number of stages) are considered and design strategies are discussed. In the Kunkel and Smith [7] and Dubey and Flynn [8] papers, instruction pipelines are considered and the problems of control and data hazards are included in the analysis.

Performance modeling of asynchronous pipelines has, in general, been less extensive and explicit than the synchronous work cited above. This is probably due, in part, to the fact that asynchronous design techniques are (currently) infrequently used in commercial designs. General work on asynchronous design issues have been considered for some time with some of the early work in this area being incorporated in the design of a set of asynchronous computer design modules [9] which could be easily used in a pipelined fashion [10]. More recently, work which has focussed on the synchronization and pipelining aspects of asynchronous systems may be found in [11] and [12]. In addition, an asynchronous RISC processor design has been implemented at Caltech [13,14] and an alternative design analyzed by Ginosar and Michell [15].

Few quantitative results have been identified which compare the performance of clocked versus asynchronous systems when these differing methodologies have been applied to the implementation of the same function or instruction pipeline. Wann and Franklin [16] consider the two approaches in the context of interconnection network design. They present performance models and associated decision curves which permit the designer to determine which approach is best for a given set of technology parameters. Kung and Gal-Ezer [17] compare asynchronous and clocked implementations of an $N \times N$ array of (systolic/wavefront) processors and conclude that for large array dimensions ($N > 25$) the asynchronous design has higher performance. Meng, Brodersen and Messerschmitt [12] present the design of an asynchronous programmable signal processor and, using simulation techniques, show that for the design and parameters selected, the asynchronous system has about twice the throughput of the equivalent clocked system. Indications are that this performance advantage increases as feature size shrinks and chip size increases. The work presented in this paper continues in this tradition, with the goal here being to formulate a general quantitative framework for comparing the two methodologies in the context of instruction pipelining. To keep the analysis tractable, only systems with a single pipeline are considered.

2. Pipelining

An S-stage (segment) pipelined processor offers an S-fold increase in throughput over its non-pipelined counterpart only if every stage is 100% utilized. Instruction pipelines generally do not achieve this full utilization for several reasons:

- **Unbalanced Workload:** Partitioning instruction execution into S stages of equal time duration is difficult. Typically the slowest stage limits pipeline throughput and in clocked systems this leads to underutilization of some stages.
- **Control (Branching) and Data Dependencies:** Control and data dependencies may cause flushing or stalling of a pipeline, thus reducing overall utilization.
- **Synchronization Overhead:** Associated with each stage computation completion and transfer of results to the next stage there is a synchronization overhead. This includes delays attributable to clock skew, latching time, intentional padding (for single phase clocked systems), and completion-detecting and handshaking delay (for asynchronous systems). These delays have the effect of further reducing pipeline utilization and thus overall throughput.

If none of the above overhead constraints were present, then throughput maximization would be achieved simply by maximizing the number of stages. When there are too many stages, however, many of the above problems are exacerbated and pipeline performance suffers. Thus, for each of the synchronization methodologies, technology, and instruction execution parameters there is some optimal number of stages. One problem therefore, is to determine how performance varies with S for each of the design methodologies. Comparison of the two approaches can then be done in a straightforward manner.

In order to develop models which are as general as possible, and yet span the methodologies in a reasonably unbiased manner, certain simplifying assumptions are made. First, we assume that we effectively have an unlimited number of instructions to execute, and thus pipeline fill and flush times arising from finite vector length effects can be ignored. This differs from the vector arithmetic pipeline situation where finite operand vectors are present and pipeline fill and flush times may significantly reduce pipeline utilization.

Second, we assume that instruction execution can be ideally modelled as the evaluation of a combinational logic function whose complexity can be denoted by the number of gate levels required for function implementation. Furthermore, partitioning of this function into any integer number of equal (gate

level) subfunctions is possible. With this assumption, the first constraint above (unbalanced workload) is not present (this is modified somewhat later in the development to reflect variations associated with executing different instruction types).

The second constraint limiting full pipeline utilization concerns the role of hazards. Dubey and Flynn [8] present an approach to extending a clocked performance model to include such hazards. We employ a variant of their approach. The third limit to full utilization relates to synchronization overheads. This is modelled directly in the analysis to follow.

The general argument in favor of using asynchronous methodologies rests on three ideas:

- **Clock Skew:** As clocked systems increase in size, clock skew increases and inevitably limits clock rate. The equivalent delay is not present in asynchronous systems (although other delays are present).
- **Average versus Worst Case Times:** The clock period in a synchronous system must be based on the worst case time for a "computational block". In asynchronous systems, however, average block delays may govern overall throughput rates. Differences between average and worst case timing have three principal origins. First, different instructions may have different average and worst case execution times (due to operand dependence). Second, to ensure correct operation, designs must be based on worst case assumptions concerning fabrication tolerances. Finally, also to ensure correct operation, designs must be based on worst case assumptions concerning environmental operating conditions (e.g., temperature).
- **Design Issues:** Hierarchical, modular design techniques are generally ill suited to handling global design constraints such as clock distribution. Asynchronous techniques permit one to focus on the functional and logical sequencing aspects of design and not on such global issues thus making the design task more manageable.

The models presented in this paper permit one to examine the first two ideas above in a quantitative fashion. The third idea, though intuitively appealing, is difficult to quantify and is not considered here. Most pipelines found in commercial systems are clocked and can be modeled as variants of Figure 1. Each computation block is made up of combinational logic which performs some part of the overall function (instruction) to be achieved. Following each computation block is a latch which, on the occurrence of a clock event, samples the outputs of the block and holds the sample at its own output until the next clock event occurs. A computation block and its latch constitute a stage or a segment of the pipeline.

Figure 2 shows the basic model of an asynchronous pipeline. At a functional level, the computation

Figure 1. Basic clocked pipeline model

Figure 2. Basic asynchronous pipeline model

block is assumed to be the same as in the clocked case*. Now, however, the timing of data transitions between stages is controlled by a handshaking protocol. Blocks are self-timed so that a stage sends a request signal to its successor stage when computation is completed and data has been placed on the data lines. If the successor stage can accept the data, an acknowledge signal is returned, and the job is passed to next stage in the pipe. Otherwise, the job waits in the current stage until the successor stage is empty. Though not shown here, multiple buffers or latches may be present between the computational blocks to help increase average block utilization and overall throughput. Finding the throughput of an asynchronous pipeline is difficult, because the pipeline is elastic in the sense that the time to get through a stage may be variable and dependent not only on the vacancy of succeeding stages, but on the actual data being processed in a given stage. In Section 4 stochastic models are used to find the approximate throughput for the asynchronous case.

To clarify the discussion that follows, the following terms are defined:

* This assumption is made for simplicity of analysis. It will not hold in a number of situations where there may be no advantage, for example, in providing early completion logic when a clocked design is employed.

- **Stage Cycle Time:** The difference between the time that an instruction (generally partially executed) enters a pipeline stage and the time that it leaves the stage.
- **Computation Time:** The difference between the time that an instruction (generally partially executed) enters a computation block and the time that all of its computed outputs are stable.
- **Idle Time:** The time that a (generally) partially executed instruction waits, after computation has been completed, before it moves to the next stage. For a clocked pipeline, an instruction may have to wait until the next clock event. In the asynchronous case, the instruction may have to wait (i.e., is blocked) because its successor stage is still executing.
- **Synchronization time:** Time used to synchronize two adjacent stages is called the synchronization time. For a clocked pipeline, the synchronization time includes clock skew, latching time, and intentional padding delay. For an asynchronous pipeline, it corresponds to the completion-detecting, latching, and handshaking delay.

As shown in Table 1, for both clocked and asynchronous designs, a stage cycle time is composed of three non-overlapped components: computation time, idle time, and synchronization time. Certain design techniques can be used to overlap these components thus reducing the stage cycle time. The effect of such techniques on performance are considered later in the paper.

Table 1: Pipeline Stage Cycle and Its Elements

Methodology	Stage Cycle Time		
	Computation Time	Idle Time	Synchronization Time
Clocked	The actual amount of time used for instruction execution	Time used to wait for the next clock pulse	Clock skew and latching and padding delay
Asynchronous	As above	Time used to wait for the availability of the next stage	Completion-detecting, latching, and handshaking delay

If t_s is the stage cycle time for the pipeline, the throughput for a pipeline that is 100% utilized (i.e., one operation is completed by the pipeline per cycle) is $1/t_s$. Since the average number of Operations being completed Per Cycle (OPC) is usually less than one due to control and data hazards, the throughput is given by:

$$\text{Throughput} = \text{OPC} / t_s. \quad (1)$$

Stage cycle times of the clocked and asynchronous methodologies will be modeled in Sections 3 and 4 respectively, and relative performance of the two methodologies will be compared in Section 5. The values used for the OPC is discussed in Section 6 along with the analysis of optimal pipeline depth and resulting pipeline throughput.

3. Clocked Pipeline Model Development

The instruction to be pipelined is assumed to require the equivalent of M gate levels in its nonpipelined version. Its pipelined implementation therefore takes N gates per stage where $N = M / S$. Gates are taken to be identical with maximum and minimum times given by $D_{\max}$ and $D_{\min}$ respectively ⁺. Stage computation time is considered to be randomly distributed with maximum, $t_{\text{comp-max}}$, and minimum, $t_{\text{comp-min}}$, values given by:

$$t_{\text{comp-max}} = N * D_{\max}, \quad t_{\text{comp-min}} = N * D_{\min} \quad (2)$$

For a clocked system, the computation time plus idle time is always equal to or greater than the maximum computation time, $t_{\text{comp-max}}$ ^{*}. Table 2 contains variable definitions which apply to both clocked and asynchronous pipeline designs. Table 3 contains definitions associated with the clocked pipeline model.

Table 2: General Variable Definitions

Variable	Definition	Remark
M	No. gate levels needed to implement the function	
S	No. pipeline stages	
N	No. gate levels per stage	$N = M / S$
$D_{\max}$	Max. prop. delay of a gate	
$D_{\min}$	Min. prop. delay of a gate	
r	The prop. delay ratio	$r = D_{\min} / D_{\max}$
L	No. gate-delays for data latching	
$t_{\text{comp-max}}$	Max. comp. time of a stage	$t_{\text{comp-max}} = N * D_{\max}$
$t_{\text{comp-min}}$	Min. comp. time of a stage	$t_{\text{comp-min}} = N * D_{\min}$

Table 3: Clocked Design Variable Definitions

Variable	Definition	Remark
t_{skew}	Clock skew	
t_{latch}	Latching delay	
k	Clock skew parameter	
p	Fraction of clk. tree levels requiring buffers	

On some high performance vector machines, a single-phase, single-level-latch clocking scheme is used. To avoid race conditions with this type of scheme certain timing constraints on clock cycle time and pulse width must be obeyed. When the constraints cannot be satisfied on certain pipeline stages, an

⁺ During this development this delay difference, at a given worst case temperature, is assumed to arise from design and fabrication variations. The further effects of environmental variations (e.g., temperature and power) are considered later.

^{*} Jouppi [18] has proposed "time borrowing" technique to loosen this constraint. However, for a pipeline with a balanced workload per stage, as assumed here, the technique is not applicable.

intentional (padding) delay is added to the computation blocks. Cotten [4], Fawcett [6], and Kunkel and Smith [7] present constraints and performance analysis for such single-phase clocked pipeline designs. Most contemporary CMOS-based RISC machines, however, use multiple-phase clocking schemes, and in the section to follow a two-phase scheme is modeled and used as a basis for comparison with its asynchronous counterpart. The equivalent analysis for the single-phase case can be found in [19].

Figure 3 shows the non-overlapped two-phase clocking scheme being considered. Proper use of the two-phase clocking and associated two level latching (master and slave) eliminates the race condition difficulties [20] associated with single-phase schemes, however, introduces additional latching delays and potentially additional delays due to clock skew (e.g., inter-phase clock skew).

data from passing through more than one stage during each cycle thus preventing races from occurring.

>From the above description the following constraint follows. During the time period which starts with the rising edge of ϕ_2 and ends with the falling edge of ϕ_1 :

1. Data must be latched by the slave (controlled by ϕ_2),
2. Data must propagate through the computation block (i.e., processed), and
3. Computation block results must be latched by the master (controlled by ϕ_1).

In addition, the maximum clock skew which may develop between the two phases and between adjacent stages (discussed further in the next section) must be accounted for during in this period. Thus (see Figure 3), the first constraint of the two-phase clocking scheme can be written as:

$$t_1 + t_2 + t_3 \geq t_{comp-max} + 2 * t_{latch-max} + t_{skew}. \quad (3)$$

A second constraint concerns the "dead time" time t_4 . To prevent race conditions, ϕ_1 and ϕ_2 must not be high at the same time. Thus, the "dead time" must be long enough to tolerate the maximum clock skew between phases and adjacent stages. That is:

$$t_4 \geq t_{skew}. \quad (4)$$

Adding equations (3) and (4) yields a constraint on the clock cycle time.

$$t_{clk} = t_1 + t_2 + t_3 + t_4 \geq t_{comp-max} + 2 * (t_{latch-max} + t_{skew}). \quad (5A)$$

Since it is desirable to have the clock period to be as short as possible, the inequality in equation (5A) is replaced with an equality. The maximum computation time is given in equation (2). If L is the number of equivalent gate delays associated with each latch in the pipeline, the maximum latch delay is $L * D_{max}$, and equation (5A) can be rewritten as:

$$t_{clk} = N * D_{max} + 2 * (L * D_{max} + t_{skew}). \quad (5B)$$

3.2. Clock Skew

Four factors determine the inter-stage clock skew, $t_{skew,is}$:

- Differences in clock signal line lengths.
- Differences in line parameters (e.g., resistivity, etc.) that determine line time constant.
- Differences in delays through active elements inserted in the lines (e.g., clock drivers).

- Differences in threshold voltages of pipeline stage input latch devices.

Assuming that the number of stages, S , is a power of 2, the first source of skew can be eliminated by using a binary tree distribution (see Figure 4). This assumes a simple linear pipeline layout. The skew associated with the remaining sources can be estimated as being some fraction, f , of the average overall delay through the tree.

Define p as the fraction of levels at which there is a driver (clock buffers) present, and define B_{avg} as the average delay of a driver and the line segment it drives. When a clock signal travels from the root to one of the leaves, it passes through $(1+p*\log_2 S)$ drivers and line segments. The overall delay can be approximated as:

$$t_{tree-avg} = B_{avg} * (1 + p * \log_2 S) \quad (6)$$

The clock skew equals $f * t_{tree-avg}$. If we assume that all elements on the chip including drivers and line segments have the same minimum over maximum delay ratio, the fraction f is equivalent to $(D_{max} - D_{min}) / D_{avg}$. Clock skew can then be written as:

$$t_{skew,ls} = (1 + p * \log_2 S) * B_{avg} * (D_{max} - D_{min}) / D_{avg}. \quad (7A)$$

Defining B_{avg} / D_{avg} (i.e., the average delay of a driver and the line segment it drives in terms of average unit of gate-delays) equal to a clock skew parameter, k , equation (7A) can be rewritten as:

$$t_{skew,ls} = k * (p * \log_2 S + 1) * (D_{max} - D_{min}). \quad (7B)$$

k can be obtained by fitting to Spice simulations programs, or calculated analytically from RC-equivalent

Figure 4. Clock distribution using a binary tree

circuits [21].

The above analysis was developed with reference to the single phase, inter-stage skew. It also applies, however, to both the inter-phase skew, and the combination of inter-stage and inter-phase skews if one assumes:

- the clock trees for both clock phases are of identical design and layout, and
- there is no inter-phase skew present at the root of the two clock trees.

The maximum and minimum delays associated with these various skews are identical to $t_{skew.is}$ developed above and are designated simply as t_{skew} in the remainder of the paper^{**}. Equation (7B) can now be substituted into (5B) and the overall clock time evaluated.

4. Asynchronous Pipeline Model Development

The throughput of an asynchronous pipeline is difficult to calculate directly since one or more buffers may be present between adjacent stages and queueing delays may be introduced when a computation completes and finds the next stage busy, or the interstage buffers full. However, two extreme cases, one optimistic and one pessimistic, may be used to find approximations to the throughput.

In the optimistic case we assume that a buffer with sufficient capacity is present between any two adjacent stages so that there is always an empty buffer available for intermediate results (i.e., there is no idle time present). In addition, it is assumed that data (i.e., a partially executed instruction) is available at the input to each stage whenever the stage becomes free. The stage cycle time at stage i therefore consists of the computation time and synchronization time, and is denoted by the random variable t_i with its expected value being $E(t_i)$. The overall throughput of the pipeline will be bounded from below by the longest of these expected stage cycle times (i.e., the bottleneck stage) and this longest time (under these optimistic assumptions) characterizes the best possible pipeline throughput. Thus, for this optimistic case the overall stage cycle time can be expressed as:

$$t_{s-op} = \text{Max} [E(t_1), E(t_2), \dots E(t_S)]. \quad (8)$$

^{**} This can be shown formally, however, is not presented here due to lack of space.

A second, pessimistic bound on the stage cycle time, t_{s-pe} , can be obtained by assuming that there is a global AND gate present which gathers completion signals from all stages. Assertion of this gate's output indicates that computation over all stages is complete. At this point the output of the AND gate enables each stage to pass its results to the succeeding stage and accept results from a prior stage thus starting the next cycle. Thus, every stage is in lock-step with the stage which has the maximum stage cycle time and the average stage cycle time may be taken to be the mean of the maximum of stage cycle times, over all stages.

$$t_{s-pe} = E [\text{Max} (t_1, t_2, \dots t_S)]. \quad (9)$$

Variable definitions for the asynchronous pipeline model can be found in Table 4.

Table 4: Asynchronous Design Variable Definitions

Variable	Definition
t_i	Stage cycle time at stage i (a random variable)
$t_{s-dr-op}$	Optimistic mean of the stage cycle time for double-rail approach
$t_{s-dr-pe}$	Pessimistic mean of the stage cycle time for double-rail approach
$t_{s-bd-op}$	Optimistic mean of the stage cycle time for bounded-delay approach
$t_{s-bd-pe}$	Pessimistic mean of the stage cycle time for bounded-delay approach
W	Completion time parameter
H	No. gate-delays for four-phase handshaking
h	No. gate-delays for two-phase handshaking
e	Operating-environment parameter

4.1. Double-Rail Approach

A *pure* asynchronous design must be able to detect the computation completion for each stage. This can be done using multi-level logic (where a logic state is assigned to the completion event), or by use of double-rail encoding (where the complementary computation logic is implemented and a difference between the two is detected) [11,22]. The double-rail approach is considered here while a *pseudo*-asynchronous approach (the bounded delay approach) is considered in a later section. In double-rail designs (see Figure 5) the completion signal of a computation block is formed using complementary output signals obtained from standard and complementary implementations of the logic. These two signals are fed into a completion detector whose output is asserted when the computation block output and its complement output are either a (1, 0) or (0, 1). Assertion of the completion detector output indicates the

Figure 5: Double-Rail Approach

computation has completed and the output of the computation block is ready and stable. The (1, 1) condition is forbidden while the (0, 0) condition is generally taken as a completion signal "reset" (i.e., the completion signal set to 0). This serves to separate two successive computation block completions.

The principal drawback associated with double-rail encoding is the increase (approximate doubling) in the logic needed to generate both the standard output and its complement. The advantage of double-rail encoding is the speed increase associated with eliminating the bound tied to $t_{comp-max}$ (equation 5A).

This advantage derives from the following sources:

- **Algorithm/Instruction Variation:** Double-rail approach can take advantage of computation time differences which are related to input operands that may cause certain function evaluations to proceed faster than others. In instruction pipelines, a large variance in stage completion times can be expected due to the differing characteristics of various instructions. A completion time parameter W ($0 \leq W \leq 1$) is introduced to reflect this possibility of early completion. The minimum computation time, thus, can be written as $W * N * D_{min}$.
- **Delay and Environmental Variation:** The delay of a gate has been taken to be randomly distributed between D_{max} and D_{min} . This delay difference is due to fabrication variations under worst case environmental operating conditions. For clocked systems, these values must guide design so that the clock cycle time is long enough for correct operation. However, system environments are not usually operating under worst-case conditions. An environmental parameter e ($0 \leq e \leq 1$) is introduced to reflect the actual (normal) operating environment. The maximum and minimum values of a gate delay under these typical conditions can be written as $e * D_{max}$ and $e * D_{min}$ respectively. Since the double-rail approach is not limited by worst case delay, its computation time is closer to these propagation delays, and these delays are likely to be lower than the worst case.

Define H as the number of gate-delays for synchronization that includes completion detection and handshake. Considering the above variations, we can write maximum and minimum values of a stage cycle time, t_i , as:

$$t_{i-max} = t_{comp-max} + t_{sync-max} = N * e * D_{max} + H * e * D_{max}, \quad (10)$$

$$t_{i-min} = t_{comp-min} + t_{sync-min} = N * W * e * D_{min} + H * e * D_{min}. \quad (11)$$

The distribution of t_i appears too complex to be modeled since it depends on all of delay variation described above. For simplicity of analysis, we assume that all t_i 's ($i = 1, 2, \dots, S$) are identical, independent, and uniformly distributed between the maximum and minimum values formulated as equations (10) and (11). Using equations (8) and (9), the optimistic and pessimistic solutions for the average stage cycle time of the double-rail approach can be written as:

$$t_{s-dr-op} = [(N + H) * e * D_{max} + (N * W + H) * e * D_{min}] / 2, \quad (12)$$

$$t_{s-dr-pe} = [(N + H) * S * e * D_{max} + (N * W + H) * e * D_{min}] / (S + 1). \quad (13)$$

The S terms in equation (13) result from obtaining the *maximum* of the expected values of a set of S identical, independently distributed uniform random variables.

The stage cycle time, t_{s-dr} , is between $t_{s-dr-op}$ and $t_{s-dr-pe}$. If there is no buffer physically implemented in the asynchronous pipeline, t_{s-dr} will be close to the pessimistic approximation. On the other hand, our numerical experiments have shown that if there are buffers between all adjacent stages with capacity greater than 5, t_{s-dr} is close to the optimistic bound. Over the parameter values of interest as shown in Section 5, the optimistic and pessimistic bounds are typically within 20% of each other, and the average of the two is used as an approximation to t_{s-dr} . Note that due to this averaging process, results based on this development imply an asynchronous design with a few buffers between stages.

4.2. Synchronization Overhead for Asynchronous Pipelines

Synchronization overhead varies with the handshaking protocol selected. Four-phase and two-phase handshaking protocols are shown in Figures 6 and 7 respectively. With four-phase handshaking every control signal returns to zero at the end of each cycle, thus it is a natural approach in conjunction with double-rail encoding where the completion signal also returns to zero (reset) in every cycle. However, the four-phase handshaking has a relative high overhead since both rising and falling edges of the request and acknowledge signals exist in a cycle.

Figure 6: Four-Phase Handshaking Protocol

Figure 7: Two-Phase (Transition) Handshaking Protocol

For two-phase handshaking protocol (also called transition signaling), both rising and falling transitions are used as control signals. Therefore, in one cycle, the number of transitions is only half of what the four-phase protocol requires. A pseudo-asynchronous design, which works well with the two-phase handshake, is discussed in the following section.

4.3. Bounded-Delay Approach

An approach to developing a pseudo-asynchronous pipeline which has been presented in the literature is the bounded-delay (micropipeline) approach [23]. With this approach a delay element is placed in parallel with the computation logic (see Figure 8). The delay time is fixed to the maximum possible

Figure 8: Bounded-Delay Approach

delay associated with the computation logic. Thus, when a signal exits the delay element, completion of the computation is guaranteed. Since the delay element (i.e., the control path) is isolated from the computation logic (i.e., the data path), the control path may execute transition signaling (two-phase handshaking) while the data path remains level-sensitive. The main advantage of this approach over the double-rail approach is that it takes less logic and has lower synchronization overhead. However, since the delay is fixed at the maximum (worst case) computation delay, the bounded-delay approach cannot take advantage of algorithm/instruction and design/fabrication variations. However, assuming the delay element experiences the same environmental conditions as the functional logic, the method can take advantage of environmental variations.

Suppose that the delay element consists of a series of inverters. The number of gate-levels in the delay element must be at least equal to $N * D_{\max} / D_{\min}$. Hence, the minimum delay of the delay element is still greater than the maximum computation time. If h is defined as the number of gate-delays for two-phase handshaking protocol, the minimum and maximum values of the stage cycle time can be written as:

$$t_{i-\max} = t_{\text{delay-element-max}} + t_{\text{sync-max}} = N * e * D_{\max}^2 / D_{\min} + h * e * D_{\max}, \quad (14)$$

$$t_{i-\min} = t_{\text{delay-element-min}} + t_{\text{sync-min}} = N * e * D_{\max} + h * e * D_{\min}. \quad (15)$$

Again, for simplicity of analysis, we assume that all stage cycle times, t_i 's, are identical, independent, and uniformly distributed between the maximum and minimum values. The optimistic and pessimistic

solutions for the average stage cycle time of the bounded-delay approach, thus, can be written as:

$$t_{s-bd-op} = [N * e * (D_{\max}^2 / D_{\min} + D_{\max}) + h * e * (D_{\max} + D_{\min})] / 2, \quad (16)$$

$$t_{s-bd-pe} = [N * e * (S * D_{\max}^2 / D_{\min} + D_{\max}) + h * e * (S * D_{\max} + D_{\min})] / (S + 1). \quad (17)$$

The stage cycle time, t_{s-bd} , is now equal to $(t_{s-bd-op} + t_{s-bd-pe}) / 2$.

5. Clocked and Asynchronous Pipeline Comparison

Stage cycle times yielded by the three synchronization approaches (one clocked and two asynchronous) have been modeled in previous sections. Given that $r * D_{\max} = D_{\min}$ ($0 \leq r \leq 1$), and normalizing the clock time and stage time equations by $D_{\max}$, one obtains:

- Single-Phase Clocked Pipeline (from equations 5B and 7B):

$$t_{clk} = N + 2 * (L + t_{skew})$$

where $t_{skew} = k * (p * \log_2 S + 1) * (1 - r).$ (18)

- Double-Rail Asynchronous Pipeline (from equations 12 and 13):

$$t_{s-dr} = (t_{s-dr-op} + t_{s-dr-pe}) / 2,$$

where $t_{s-dr-op} = e * [(N + H) + (N * W + H) * r] / 2,$

and $t_{s-dr-pe} = e * [(N + H) * S + (N * W + H) * r] / (S + 1).$ (19)

- Bounded-Delay Asynchronous Pipeline (from equations 16 and 17):

$$t_{s-bd} = (t_{s-bd-op} + t_{s-bd-pe}) / 2,$$

where $t_{s-bd-op} = e * [N * (1/r + 1) + h * (1 + r)] / 2,$

and $t_{s-bd-pe} = e * [N * (S/r + 1) + h * (S + r)] / (S + 1).$ (20)

5.1. Parameter Selection

To evaluate and compare the design methodologies contained in the above equations a host of parameters must be chosen which reflect implementation technology, operating range, and pipeline instruction characteristics. Although readers will no doubt differ with some of the values chosen, and the values themselves change with technology and assumptions, Table 5 below is an estimate of typical

parameters associated with current technology.

Table 5: Working Set of the Parameters

Name	M	L	p	k	r	e	W	H	h
Value	128	2	0.5	4.7	0.9	0.8	0.5	10	4

The instruction pipeline is considered to be implemented in a manner equivalent to a combinational logic function having 128 gate-levels ($M = 128$) with the levels being evenly partitioned into an S -stage pipeline. The design assumes use of a Earle latch that takes two gate delays ($L = 2$), and latching is assumed (initially) not to overlap with computation [1]. The clock skew parameter is derived by assuming that a driver is inserted every two levels of the clock tree ($p = 0.5$). A least square fit of equation (7B) with the results of a Spice tree model of the clock tree circuit yields a value of $k = 4.7$. A 0.5μ technology process is assumed with a 3.3 V supply and tree width, $L_{tree} = 1$ cm. [21].

Fabrication derived variation in the design (i.e., the difference between D_{min} and D_{max}) is taken to be 10% ($r = 0.9$). The environmental parameter, e , is assumed to be 0.8, which means a gate delay under the typical environment is 80% of that under the worst-case environment that is tolerated by the clocked design. The simulation results of [24] indicate this assumption is within a reasonable range. The completion time parameter, W , which reflects algorithm/instruction variation is given as 0.5. That means the effective computation time of the simplest operation takes (on average) 50% of the worst-case computation time. Synchronization overhead of the double-rail approach, H , is taken as 10 gate delays. This includes two gate delays for completion detection and eight gate delays for control signals traveling through a data latch and two Muller-C elements twice in each cycle (i.e., a full-handshake) [11,21]. Since bounded-delay approach uses two-phase handshaking protocol and does not require completion detection, its synchronization delay, h , is estimated as 4 gate delays.

The relative performance of the asynchronous and clocked approaches can be obtained by taking the ratios of t_{clk} / t_{s-dr} and t_{clk} / t_{s-bd} respectively. The relative throughputs are plotted for the selected parameter values as functions of pipeline depth, S , in Figure 9. The following observations can be made:

1. For short pipelines (small S), the double-rail approach yields the highest performance among the three due to the advantage of variable computation time.
2. For longer pipeline depths, the bounded-delay yields the highest performance of the two asynchronous designs because of its relatively low handshaking overhead.
3. Both asynchronous designs yield better performance than the clocked design due to multiple factors. The asynchronous designs can take advantage of variable computation time and, for longer pipelines, clock skew overhead has a greater effect than handshaking delays in the asynchronous cases. For the selected parameters, with 8 stages, the asynchronous designs are better by about 25%. The bounded delay approach has about a 50% performance advantage (over the clocked system) with 16 stages.

5.2. Parameter Sensitivity

This section explores the sensitivity of the results to certain parameter variations. In Figure 10, the clock skew parameter k is taken as 8.9 (versus 4.7 in Figure 9). This corresponds to the Spice results for the case $L_{tree} = 2\text{ cm}$ (versus 1 cm in Figure 9) [21]. Clearly the increased clock skew results in further relative performance advantage associated with the asynchronous approach.

The instruction completion time parameter W is highly dependent on the architecture and instruction set considered. The value of $W = 0.5$ corresponds roughly to what would be expected with a five stage DLX pipeline executing an aggregate DLX instruction mix [2]. Figure 11 shows the impact of W on relative throughput for the double-rail case. As expected with lower values of W , asynchronous design

Figure 9 ($k = 4.7$)

Figure 10 ($k = 8.9$)

performance improves since there is greater opportunity for this approach to take advantage of the lower minimum execution times. Note also that the relative advantage of the asynchronous design decreases with longer pipelines. This is due to the higher probability that with a longer pipe, some stage will require a maximum stage time thus blocking earlier stages in the pipe which may have shorter execution times.

Synchronization overhead of the double-rail design, H , is given as ten gate delays in the working set. Sensitivity of results to H is shown in Figure 12. We can, for example, expect smaller H values when techniques are used which overlap synchronization with computation time. In the extreme case of complete overlap ($H = 0$) about a factor of 2 improvement is predicted over the clocked case (for $S = 8$).

Variations in the environmental parameter e are explored in Figure 13. The case $e = .8$ is our nominal value and indicates that typical operating conditions result in delays which are 20% better than worst case. Note that if worst case conditions exist (e.g., $e = 1.0$), then in the range of 4 to 8 stages, there is little difference between performance of the double-rail and clocked systems. However, if there is a large difference between worst case and typical operating conditions (e.g., $e = .5$) then the asynchronous design is superior.

Figure 11 (double-rail)

Figure 12 (double-rail)

Figure 13 (double-rail)

Figure 14 (double-rail)
(zero synchronization delays)

Finally, it is worthwhile to explore the impact of aggressive design techniques on the relative performance of these methodologies. Figure 14 presents the relative performance when there is no synchronization overhead with either of the design methodologies. For the asynchronous cases, this is equivalent to setting $H = 0$. For the clocked case, this is equivalent to setting both the clock skew and the latching delay to zero. The results indicate that the asynchronous designs (with a few buffers between stages) have better performance than clocked designs. This performance accrues from the ability of this approach to take advantage of average versus worst cases delays.

6. Optimal Pipeline Design

The number of stages, S , is inversely proportional to the number of gate levels per stage, N . Thus, although increasing the depth of a pipeline decreases computation time per stage, it increases clock skew and, for asynchronous systems, increases the probability that later stages will be blocked by a long computation in an earlier stage. Additionally, with more stages the effects of control and data hazards reduces performance. The result of these effects is that there exists an optimal number of stages which maximizes throughput.

The optimal pipeline length may be found by maximizing the throughput as indicated in equation 1. Stage cycle times can be found in equations (18) to (20), (multiplied by $D_{\max}$ taken as 0.35 ns for 0.5 μ technology and 3.3 V supply). Unfortunately, there is little experimental data available on the OPC of instruction pipelines as a function of pipeline depth, and there are no simple and comprehensive theoretical models which can be used. Given this situation, an approximate function of OPC (Figure 14) is developed based on following notions:

- OPC of a 1-stage processor of the DLX [2] type is 94% *.
- Simulation results indicate that the OPC of a 5-stage DLX processor (under the same assumptions as above) is 73%.
- OPC will decrease with an increasing number of stages due to the difficulties associated with control and data hazards. That is, while hazards can be effectively controlled when the number of stages are small, this becomes increasingly difficult as S increases.
- Due to the above we take OPC to be zero when $S \geq 32$.
- We assume that OPC decreases in a log-linear fashion between the three points $S = 1, 5$, and 32.

The resulting throughput of the three approaches is shown in Figure 16 which presents the variation in throughput as a function of the number of stages for the parameters of Table 5. The resulting curves show that the optimum number of stages is about 8 which roughly corresponds to the number being used

Figure 15: Instruction Pipeline OPC

Figure 16: Instruction Pipeline Performance

* This is under the assumption that there are no cache misses, 5% of the instructions are floating point, each non-floating point instruction takes 1 clock cycle, and each floating point instruction takes an average of 2.3 clock cycles.

in the instruction pipelines of contemporary RISC processors (e.g., MIPS R4000) [25]. The clock rate associated with this single pipeline design is about 115 Mhz.

7. Conclusion

In this paper, performance models for clocked and asynchronous pipelines are developed. The design approach which yields the best performance is then evaluated based on a set of technology parameters, operating range, and instruction set execution characteristics. Two asynchronous approaches, double-rail and bounded-delay, are modeled and compared with their clocked counterpart under a common set of assumptions. Developing such models and comparisons is important in gaining an understanding of the limiting factors and underlying constraints which determine the performance of the two methodologies. This paper has aimed at gaining such an understanding.

For the clocked approach, clock skew is one barrier that limits pipeline depth and chip size. In addition, the clocked approach cannot take advantage of any of the variations in pipeline stage time due to different instructions, or average versus worst case logic delays. The results indicate that, with the parameters selected, and about an 8 stage instruction pipe, on the order of a 20 or 30 percent performance improvement can be expected using an asynchronous design. Such a design would require two or three buffers between stages. Increased performance can be obtained if aggressive design techniques are employed which overlap handshaking with computation. When compared with zero skew and latch overhead clocked systems; these asynchronous systems still appear to outperform their clocked counterparts.

Therefore, although clocked design is most frequently used, asynchronous design should be considered when the clock skew is large, the variance of computation time is large, or the system is designed to tolerate highly variable operating environments.

REFERENCES

1. P.M. Kogge, *The Architecture of Pipelined Computers*, Hemisphere Publishing Corporation, New York, NY (1981).
2. J.L. Hennessy and D.A. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publishers, Palo Alto, CA (1990).
3. H.S. Stone, *High-Performance Computer Architecture*, Addison-Wesley, Reading, MA (1990).
4. L.W. Cotten, "Circuit Implementation of High-Speed Pipeline Systems," *Proc. AFIPS - Fall Joint Computer Conference*, pp. 489-504 (1965).
5. T.G. Hallin and M.J. Flynn, "Pipelining of Arithmetic Functions," *IEEE Trans. Comput.* C-21,8 pp. 880-886 (August 1972).
6. Fawcett, "Maximal Clocking Rates for Pipelined Digital Systems," *M.S. Thesis, Dept. of Elect. Engr., Univ. of Ill. Urbana-Champaign*, (1975).
7. S.R. Kunkel and J.E. Smith, "Optimal Pipelining in Supercomputers," *13th Inter. Symp. Comput. Arch.*, pp. 404-411 (June 1986).
8. P.K. Dubey and M.J. Flynn, "Optimal Pipelining," *J. of Parallel and Distributed Computing* 8 pp. 10-19 (1990).
9. W.A. Clark, "Macromodular Computer Systems," *Proc. of the Spring Joint Comput. Conf., AFIP*, (1967).
10. E. Sadeh and M.A. Franklin, "Monte Carlo Solution of Partial Differential Equations by Special Purpose Digital Computer," *IEEE Trans. Computers* C-23 pp. 389-397 (April 1974).
11. T.H. Meng, *Synchronization Design for Digital Systems*, Kluwer Academic Publishers, Norwell, MA (1991).
12. T.H. Meng, R.W. Bordersen, and D.G. Messerschmitt, "Asynchronous Design for Programmable Digital Signal Processors," *IEEE Trans. Signal Processing* 39,4 pp. 939-952 (April 1991).
13. A.J. Martin, S.M. Burns, T.K. Lee, D. Borkovic, and P.J. Hazewindus, "The Design of an Asynchronous Microprocessor," *Proc. Decennial Caltech Conf. on VLSI (MIT Press)*, pp. 20-22 (March 1989).
14. A.J. Martin, S.M. Burns, T.K. Lee, D. Borkovic, and P.J. Hazewindus, "The First Asynchronous Microprocessor: The Test Results," *Comput. Arch. News* 17,4 pp. 95-110 (June 1989).
15. R. Ginosar and N. Michell, "On the Potential of Asynchronous Pipelined Processors," *Comput. Arch. News* 18,4 pp. 27-34 (December 1990).

16. D.F. Wann and M.A. Franklin, "Asynchronous and Clocked Control Structures for VLSI Based Interconnection Networks," *IEEE Trans. Comput.* **C-32,3** pp. 284-293 (March 1983).
17. S.Y. Kung and R.J. Gal-Ezer, "Synchronous Versus Asynchronous Computation in Very Large Scale Integrated (VLSI) Array Processors," *Proc. SPIE Int. Soc. Opt. Eng.* **341** pp. 53-65 (1982).
18. N.P. Jouppi, "Timing Verification and Performance Improvement of MOS VLSI Designs," *Doctoral Dissertation*, Dept. EE and CS, Stanford Univ., (October 1984).
19. M.A. Franklin and T. Pan, "Performance Comparison of Clocked and Asynchronous Pipelines," *Tech. Rep. WUCCRC-92-9*, Washington Univ., (October 1992).
20. M. Shoji, *CMOS Digital Circuit Technology*, Prentice Hall, Englewood Cliffs, NJ (1988).
21. T. Pan and M.A. Franklin, "Synchronization Delay in Pipelined Systems," *Tech. Rep. WUCCRC-92-8*, Washington Univ., (October 1992).
22. C. Mead and L. Conway, *Chap. 7, Introduction to VLSI Systems*, Addison-Wesley, Reading, MA (1980).
23. I.E. Sutherland, "Micropipelines," *Commun. ACM* **32,6** pp. 720-738 (June 1989).
24. J.D. Garside, "A CMOS VLSI Implementation of an Asynchronous ALU," *IFIP Working Conf. on Asynchronous Design Methodologies*, (31 March - 2 April 1993).
25. S. Mirapuri, M. Woodacre, and N. Vasseghi, "The Mips R4000 Processor," *IEEE Micro* **12,2** pp. 10-22 (April 1992).