Slack Matching

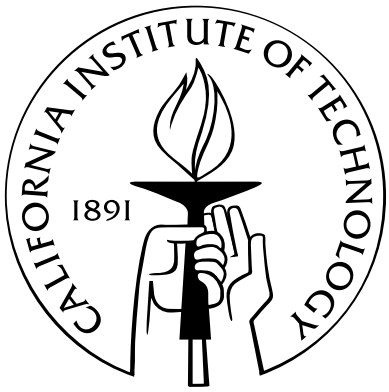
Thesis by

Piyush Prakash

In Partial Fulfillment of the Requirements

for the Degree of

Master of Science



California Institute of Technology

Pasadena, California

2005

(Submitted May, 2005)

Acknowledgements

I wish to thank my advisor, Alain Martin, for his guidance and encouragement. I also thank the members of the Asynchronous VLSI group at Caltech for the stimulating discussions: Mika Nyström, Catherine Wong, Karl Papadantonakis, Wonjin Jang, Jonathan Dama and Paul Pénzes. Last but not least, I wish to thank my parents for their support and encouragement.

The work described in this thesis was sponsored by the Defense Advanced Research Projects Agency and monitored by the Air Force Office of Scientific Research.

Abstract

In this thesis we present a method for slack matching asynchronous circuits, described as a collection of handshaking expansions. We present an execution model for a restricted class of HSE. We define the number of messages that a process contains. The static slack, dynamic slack and dynamic threshold are defined. We state sufficient conditions under which the dynamic slack of a pipeline of half-buffers is the sum of that of the processes comprising the pipeline. The slack matching problem is formulated as that of ensuring that all pipelines and rings in a system can simultaneously contain a number of messages that is no less than the dynamic threshold but no greater than the dynamic slack. We describe an algorithm to formulate the slack matching problem as a mixed integer linear program.

1 Introduction

Asynchronous circuits are designed as communicating modules that use local handshakes instead of a global clock for synchronization. It has been observed that the cycle-time of an asynchronous system can be lower than that of the slowest module in the system running on its own. *Slack matching* is a technique to reduce the system's cycle-time by inserting buffering into communication channels. Slack matching is only guaranteed to be safe on systems that are *slack elastic*, i.e., systems such that an arbitrary amount of buffering can be added along any communication channel without affecting the correctness of the system [5].

Slack matching has been compared to the *retiming* problem in synchronous design. Whilst retiming can be applied to asynchronous systems with good results, the results are inferior to optimal slack matching. Slack matching has been shown to be NP-complete [2] whereas retiming for optimal throughput can be solved in polynomial time [3]. We reduce slack matching to optimizing a given cost function over a set of linear constraints, with some variables restricted to being integers. When the cost function is linear in the number of slack matching buffers added, mixed integer linear programming (MILP) solvers can be used to slack match a system.

Systems are often designed with a target cycle-time, τ_0. We will study the problem of adding buffers to a slack elastic system to reduce the overall cycle-time of the system to τ_0. The processes in the system will be restricted to using each communication channel exactly once per cycle. The systems will be described at the handshaking expansion (HSE) level, with the HSE annotated with delays.

1.1 Prior work

Chapter 2 of Lines[4] studies asynchronous pipelines and introduces the concept of *dynamic slack*; however Lines does not rigorously define dynamic slack. Pénzes[9] presents an execution model for production rule sets. Using this model, the slack matching problem is formulated as an integer linear programming problem. However, no explicit algorithm is given to generate the integer linear program. No comment is made on the size of the integer linear program generated relative to the size of the system being slack matched. Chapter 5 of Wong[10] presents a method for slack matching systems composed of processes with the PCHB reshuffling. If the delays of all processes are identical, the slack matching algorithm presented is similar to that presented here. However, for heterogenous systems Wong[10] determines the dynamic slack of a pipeline of buffers by analyzing all paths in the pipeline. In contrast, we will state conditions under which the dynamic slack of a heterogenous pipeline is the sum of the dynamic slack of the components of the pipeline. This allows us to construct a smaller integer program in polynomial time for the slack matching problem.

1.2 Outline

We first describe an execution model for HSE called *constraint graphs*. We will state restrictions on the class of HSE that can be modeled by constraint graphs. We define the number of messages in rings and pipelines. We study how changing the number of messages in a system affects its constraint graph. The concepts of static slack, dynamic slack and dynamic threshold are defined. We present sufficient conditions under which the dynamic slack of a pipeline of half buffers is the sum of that of the processes comprising the pipeline. The slack matching problem is formulated as that of ensuring that it is possible for all pipelines and rings in a system to simultaneously contain a number of messages that is no less than the dynamic threshold but no greater than the dynamic slack. An algorithm is described for generating a set of linear constraints that need to be satisfied in order to slack match a system. The performance of the algorithm on circuits from the Lutonium[7] and MiniMIPS[6] is discussed.

2 Execution Model—Constraint Graphs

We first describe the constraint graph execution model and then provide an algorithm to generate the constraint graphs of a restricted class of HSE.

2.1 Constraint graphs

Definition 1

A constraint graph is a 4-tuple C=(V, E, k, δ) where

- V is a set of vertices
- E, the edge set, is a set of ordered pairs of vertices
- $k : E \rightarrow \{0, 1\}$ specifies the initial token placement
- $\delta : E \rightarrow [0, \infty)$ is the delay function

Constraint graphs are used to model repetitive systems. The vertices of a constraint graph represent events, and the edges capture the partial ordering between events. An edge (u, v) will have a token when event u has occurred but the subsequent occurrence of event v has not yet occurred. The delay, $\delta(u, v)$, of an edge is the minimum delay between an occurrence of u and the subsequent occurrence of v.

Tokens move through the system per the following rule: when all the fan-in edges of a vertex contain and token, and for each such edge (u, v), the token has been on the edge for at least $\delta(u, v)$ time, a token can be removed from all the fan-in edges and placed on the fan-out edges of the vertex.

2.1.1 Executions of a constraint graph

An *execution* of a constraint graph (V, E, t, δ), is any function, $e : V \times \mathbb{N} \rightarrow [0, \infty)$ such that

$$\forall i \in \mathbb{N}, (u, v) \in E : e(v, i + k(u, v)) - e(u, i) \geq \delta(u, v)$$

$$e(v, 1) = 0 \forall v : \forall (u, v) \in E : k(u, v) = 1$$

$e(u, i)$ denotes the i^{th} occurrence of the event u. Consider the mapping $l_e((u, v), t) : E \times [0, \infty) \rightarrow \{0, 1\}$ to evaluate to 1 when there is a token on edge (u, v) at time t in execution e. $l_e((u, v), t)$ is defined as:

$$l_e((u, v), t) = \exists i : e(v, i + k(u, v)) > t \wedge e(u, i) \leq t$$

A *linear execution* is an execution, where for all u, $e(u, i) = \alpha_u + p_u \cdot i$. α_u and p_u are, respectively, the offset and cycle period of the vertex u. If two vertices u and v appear on the same cycle in the constraint graph, p_u must equal p_v. The cycle time, τ, of a linear execution of the constraint graph (V, E, k, δ) is defined as

$$\tau = \max_{v \in V} \{p_v\}. \tag{1}$$

In the remainder of this thesis, we will only consider systems such that their constraint graphs are strongly connected.

Chapter 2 of Burns[1] defines repetitive Event-Rule (ER) systems. An execution of a constraint graph is analogous to a timing function of a repetitive ER system. Similarly, the fastest execution and fastest linear execution of a constraint graph are analogous to the timing simulation and minimum-period linear timing function of an ER system. Burns[1] shows that there exists a timing simulation of an ER system if for all cycles c in its collapsed constraint graph, either the sum of the occurrence index offsets along the cycle is greater than zero or the total delay along the cycle is zero. Burns shows that there exists a minimum-period linear timing function of a repetitive ER system whenever the timing simulation exists. Let s and s' represent the timing simulation and minimum-period linear timing function of a repetitive ER system, (E,R). It can be shown that there exists a finite B such that $\forall u \in E, i \geq 0, s'(u, i) - s(u, i) \leq B$ when the collapsed constraint graph of the ER system is strongly connected.

Similar arguments can be used to demonstrate the existence of a fastest execution of a constraint graph if for all cycles c in the constraint graph:

$$\sum_{(u,v) \in c} k(u, v) = 0 \Rightarrow \sum_{(u,v) \in c} \delta(u, v) = 0. \tag{2}$$

It can be shown that fastest linear execution exists whenever the fastest execution exists. Let e and e' be

the fastest execution and the fastest linear execution of the constraint graph. Then there must exist a finite B such that $\forall u \in V, i \geq 0, e'(u,i) - e(u,i) \leq B$. Therefore, the cycle period of a vertex u in the fastest linear execution of a system is the mean time between successive occurrences of u in the system's fastest execution.

The fastest execution and fastest linear execution of a constraint graph can be constructed in the same manner as the timing simulation and minimum-period linear timing function of an ER system are constructed.

The arguments in Burns[1] can also be used to show that the cycle time, τ, of the fastest linear execution of a constraint graph is given by

$$\tau = \max_{\text{all cycles c}} \left\{ \frac{\text{delay along cycle } c}{\#\text{ of tokens on cycle } c} \right\}. \tag{3}$$

Only *simple* cycles, i.e., ones that do not have any sub-cycles, need to be considered in the maximization.

2.2 Repetitive straight-line handshaking expansion

In this section, we define a class of HSE that can be modeled with constraint graphs.

An HSE in which each select statement is of the form $[G \rightarrow \mathbf{skip}]$ (a wait), where G is a conjunction of literals, and each repetition statement is of the form $*[\mathbf{true} \rightarrow S]$ (a non-terminating loop) is said to be a *straight-line-handshaking-expansion*(SLHSE) [1].

Constraint graphs will be used to model systems that are described as *repetitive SLHSE* in *standard form.*

An SLHSE that satisfies the following restrictions is said to be a repetitive SLHSE in standard form.

1. Each process can be written as $P \equiv S; *[T]$.
2. S and T are sequences of alternating waits and assignments that begin and end with a wait.
3. Each variable is either only assigned and used in one process or is used to implement a communication channel. A pair of variables (a, b) implement a communication channel if a is assigned only by one process P, b is assigned only by one process Q, and P and Q are the only processes that use the variables a and b. Furthermore, the assignments to a and b obey a four-phase handshake protocol.
4. If a variable appears in a wait statement, then the variable is either a local variable or the input variable of a communication channel such that this process is the one that assigns the corresponding output variable.
5. There are no vacuous assignments or vacuous waits on variables. However all variables are initialized.
6. All assignments occurs at most once in S and exactly once in T.
7. The projection of T onto the statements in S is S.

8. For all assignments a such that a appears in S, if $a; a'$ appears in the projection of T onto $\{a, a'\}$, a' appears in S.

9. For all variables w such that $[f(w)]$ appears in S, all assignments a such that $[f(w)]; a$ appears in the projection of T onto the variables $\{w, a\}$, also appear in S.

Restrictions 1–5 can be satisfied by rewriting an SLHSE. Conditions 6–9 ensure that the system is repetitive and can be modeled by a constraint graph.

Example 1

The process:

$PCHB \equiv init; lo\uparrow; [ri]; ro\downarrow; [\neg li]; lo\downarrow; *[[\neg ri \wedge li]; ro\uparrow; lo\uparrow; [ri]; ro\downarrow; [\neg li]; lo\downarrow]$
$init \equiv lo\downarrow, ro\uparrow, [\neg ri \ \wedge li]$

can be rewritten as a repetitive SLHSE in standard form:

$PCHB \equiv init; [\mathbf{true}]; lo\uparrow; [ri]; ro\downarrow; [\neg li]; lo\downarrow; [\mathbf{true}]; *[[\neg ri \wedge li]; ro\uparrow; [\mathbf{true}]; lo\uparrow; [ri]; ro\downarrow; [\neg li]; lo\downarrow; [\mathbf{true}]]$

2.3 Constructing constraint graphs of an HSE

We now describe how to construct the constraint graph of collections of repetitive SLHSE in standard form.

- **Vertices:** For each variable v, create vertices labeled $v\uparrow$ and $v\downarrow$. These vertices represent v being assigned the values **true** and **false** respectively.

- **Edges:** The edges of the constraint graph capture the partial ordering between assignments in a collection of HSE. Consider an assignment, a, in T, such that at least one assignment precedes a in T. We can rewrite T as $T'; a'; w; a; T''$. w and a' are respectively the *constraining wait* and *constraining assignment* of a. Consider an assignment, a, such that no other assignment precedes a in T. Rewrite T as $w_1; a; T''; a'; w_2$. In this case the constraining assignment of a is a' and w_1 and w_2 are the constraining waits of a. For every assignment a, construct edges (u, a) for all u such that either u is a constraining assignment of a, or u is an assignment that can make a literal in a constraining wait of a evaluate to **true**.

- **Tokens:** An edge (u, v) has a token if and only if the i^{th} occurrence of v precedes the i^{th} occurrence of u in any execution, for all $i > 1, i \in \mathbb{N}$.

- **Delay Function:** A function that, for each edge (u, v), evaluates to the minimum delay between an occurrence of assignment u and the subsequent occurrence of assignment v.

2.3.1 Parallel composition

We can use constraint graphs to model collections of repetitive SLHSE without the restrictions that S and T be sequences. For a program fragment

$$\ldots a_1; (w; a_2; \ldots), (w'; a_2' \ldots); \ldots$$

The constraining waits and assignments of a_2 are w and a_1. Similarly, the constraining waits of a_2' are w' and a_1. For program fragments

$$\ldots; (\ldots; a_1; w), (\ldots; a_1'; w'); a_2 \ldots$$

The constraining waits of a_2 are w and w'. The constraining assignments are a_1 and a_1'. The constraining waits of other assignments can be determined in the usual manner.

Example 2

Figure 1 shows the constraint graph of the HSE

$PCHB \equiv init; \mathbf{[true]}; lo\uparrow; [ri]; ro\downarrow; [\neg li]; lo\downarrow; \mathbf{[true]}; *[[\neg ri \wedge li] ro\uparrow; \mathbf{[true]}; lo\uparrow; [ri]; ro\downarrow; [\neg li]; lo\downarrow; \mathbf{[true]}]$

$init \equiv lo\downarrow, ro\uparrow; \{ri = \mathbf{false} \ \wedge \ li = \mathbf{true}\}$

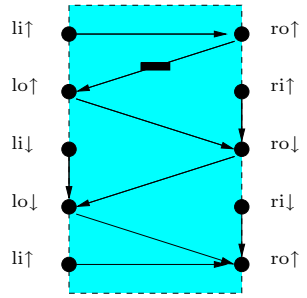


Figure 1: Constraint graph of PCHB

3 Messages

A message is distinct from a token in a constraint graph. A "message" corresponds to a value on a communication channel. We define the number of messages in pipelines and rings and show how the number of messages in rings and pipelines can be changed.

If the first non-vacuous action in S that a process π performs on the variables implementing a channel is a wait, then the channel is said to be a *passive* channel of the process. Otherwise the channel is said to be an *active* channel of the process.

A *buffer* is a process with exactly two channels, one of which is passive and the other active. A *pipeline* is a sequence of one or more processes such that for all but the last process in the sequence, exactly one passive channel of a process is an active channel of the next process in the sequence. One active channel of the first process in the sequence is said to be the active channel of the pipeline; one passive channel of the last process in the sequence is said to be the passive channel of the pipeline. For processes that have more than one active channel and/or more than one passive channel, any passive channel such that it is neither the passive channel of the the pipeline nor the active channel of the next process in the sequence is connected to a source/sink. Similarly any active channel of a process that is neither the active channel of the pipeline nor a passive channel of the previous process in the sequence is connected to a source/sink.

A *ring* of buffers is a pipeline such that the active channel of the pipeline is also the passive channel of the pipeline.

In the proceeding sections, we will refer to one of the channels as the L channel of a pipeline, and the other channel as the R channel of the pipeline. We will adopt the convention that the L channel of a pipeline is implemented using the variables li and lo; the R channel is implemented using the variables ri and ro. The variables li and ri are only assigned to by the environment; the variables lo and ro are only assigned to by the pipeline being considered. When multiple pipelines are being consider, the variables li, lo, ri, ro will be subscripted with the name of the pipeline that the variables correspond to.

In order to determine the number of messages in a pipeline, it will be necessary to keep a count of the number of times certain variables in the system are assigned to. To keep track of the number of times a variable, v, has been assigned to we introduce a ghost variable $\mathbf{c}v$ that is initialized to zero. Each assignment $v := a$ in the system is replaced by the pair of assignments $\langle v := a; \mathbf{c}v := \mathbf{c}v + 1\rangle$. We use the notation $\langle S\rangle$ to indicate that S is an atomic action. In the rest of this thesis, the variables $\mathbf{c}ri$ and $\mathbf{c}lo$ are the ghost variables that count the number of assignments to ri and lo, respectively.

3.1 Pipelines

3.1.1 Initial messages

Definition 2

The number of initial messages in π, init_msg(π), is given by init_msg(π) $= \frac{\mathbf{c}ri - \mathbf{c}lo}{2}$ when no more assignments to ri can occur in the system:

$$Init_L; I_L; lo' := lo; [lo \neq lo']; li := \neg li \parallel \pi \parallel$$
$$Init_R; \mathbf{c}ri := 0; I_R; *[ro' := ro; [ro \neq ro']; \langle ri := \neg ri; \mathbf{c}ri := \mathbf{c}ri + 1\rangle]$$

where:

- $Init_L$ initializes the L channel of π.

- $Init_R$ initializes the R channel.
- I_R is the assignment $\langle ri := \neg ri; \mathbf{c}ri := \mathbf{c}ri + 1\rangle$ if the R channel of π is passive, **skip** otherwise.
- I_L is the assignment $li := \neg li$ if the L channel of π is passive, **skip** otherwise.

3.1.2 Messages during the execution

Definition 3

The number of messages in a pipeline, π, in the system

$$Init_L; I_L; *[lo' := lo; [lo \neq lo']; li := \neg li] \parallel \pi \parallel$$
$$Init_R; \mathbf{c}ri := 0; I_R; *[ro' := ro; [ro \neq ro']; \langle ri := \neg ri; \mathbf{c}ri := \mathbf{c}ri + 1\rangle]$$

is given by $m = \frac{\mathbf{c}lo - \mathbf{c}ri}{2} + \text{init\_msg}(\pi)$.

Let $\pi_A \cdot \pi_B$ denote the pipeline such that the R channel of pipeline π_A is the L channel of π_B.

Lemma 1

The number of messages in the pipeline $\pi \equiv \pi_A \cdot \pi_B$ is the sum of the number of messages in π_A and π_B.

LemmaProof 1

Since the R channel of π_A is the L channel of π_B if the number of messages in π_A has been decreased by n, the number of messages in π_B must have been increased by n at any point during the execution of π. Since each communication channel is used on every cycle, the number of messages that can be removed from a pipeline is the sum of the number of initial messages in the pipeline and the number of messages inserted into the pipeline. Thus, $\text{init\_msg}(\pi) = \text{init\_msg}(\pi_A) + \text{init\_msg}(\pi_B)$. The number of messages in π is given by $\text{init\_msg}(\pi) + \frac{\mathbf{c}lo_\pi - \mathbf{c}ri_\pi}{2}$. The number of messages in π_A is $\text{init\_msg}(\pi_A) + \frac{\mathbf{c}lo_{\pi_A} - \mathbf{c}ri_{\pi_A}}{2}$. The number of messages in π_B is $\text{init\_msg}(\pi_B) + \frac{\mathbf{c}lo_{\pi_B} - \mathbf{c}ri_{\pi_B}}{2}$. Since $\mathbf{c}lo_{\pi_B} = \mathbf{c}ri_{\pi_A}$, the number of messages in π can be rewritten as

$$\text{init\_msg}(\pi_A) + \text{init\_msg}(\pi_B) + \frac{\mathbf{c}lo_\pi - \mathbf{c}ri_{\pi_A} + \mathbf{c}lo_{\pi_B} - \mathbf{c}ri_\pi}{2}.$$

Since lo_{π_A} is another name for lo_π and ri_{π_B} is another name for ri_π, the number of messages in π is equal to the sum of the number of messages in π_A and π_B. □

3.1.3 Changing the number of messages in a pipeline

The number of messages in a pipeline π is increased by n when the program in definition 3 is executed in such a manner that $\frac{\mathbf{c}lo - \mathbf{c}ri}{2} = n$.

3.2 Rings

Consider a pipeline π such that the initial values of the variables lo and ri are equal, and the values of the variables li and ro are equal. A ring can be formed by merging the variables li and ro into one variable, and merging lo and ri into another variable. The ring thus formed is said to be the ring formed by *connecting* the R channel of π to its L channel. Recall that either the L channel is active and the R channel is passive or vice versa.

Consider a ring r and some channel on r implemented by the variable pair a, b. The channel must be an active channel of some process p_1 and the passive channel of some process p_2. Let a be the variable that is assigned to by p_1, and b be the variable assigned to by p_2. In p_1, replace all assignments to a by assignments to the variable a_1; replace waits on b with waits on b_1. Similarly, in p_2, replace all assignments to b with assignments to b_2 and waits on a with waits on a_1. The resulting pipeline is said to be the pipeline formed by *disconnecting* the ring r at the channel (a, b). The channel (a_1, b_1) is the active channel of the pipeline, and (a_2, b_2) the passive channel of the pipeline.

3.2.1 Initial messages

The number of initial messages on a ring, r, is the number of initial message in a pipeline formed by disconnecting r at some channel.

Lemma 2

All pipelines, π, formed by disconnecting a ring, r, have the same number of initial messages.

LemmaProof 2

Consider pipelines π_1 and π_2 obtained by disconnecting the same ring at two different channels. There exist pipelines π_A and π_B such that $\pi_1 \equiv \pi_A \cdot \pi_B$, and $\pi_2 \equiv \pi_B \cdot \pi_A$.

From lemma 1, $\text{init\_msg}(\pi_1)$ must be $\text{init\_msg}(\pi_A) + \text{init\_msg}(\pi_B)$. Similarly, $\text{init\_msg}(\pi_2)$ must be $\text{init\_msg}(\pi_B) + \text{init\_msg}(\pi_A)$. □

Definition 4

For any ring, r, $\text{init\_msg}(r) = \text{init\_msg}(\pi_r)$ where π_r is any pipeline formed by disconnecting r.

3.2.2 Messages during the execution

Since a process, p, uses each communication channel exactly once per cycle, the number of messages on a ring is constant.

3.2.3 Changing the number of messages on a ring

In this section, we describe a method to change the number of messages on a ring. We also discuss how the placement of tokens in the constraint graph of a ring is affected when the number of messages on the ring is changed.

The number of messages on a ring can be changed by disconnecting the ring, changing the number of messages in the pipeline formed, and then connecting the active channel of the resulting pipeline to its passive channel to form a ring.

Lemma 3

The number of messages on a ring can only be changed by integral amounts.

LemmaProof 3

In order to be able to compose π_r with itself to form a ring, li and ro must be initialized to the same values. Similarly, lo and ri must be initialized to the same values.

In order to be able to connect the passive channel of π_r to the active channel of π_r to form a ring after inserting/removing messages, li must have the same value as ro and lo must have the same value as ri. Thus either both in and out must be even, or both must be odd. In either case, the number of messages inserted is given by $\frac{\mathbf{c}lo-\mathbf{c}ri}{2}$. Since $\mathbf{c}lo - \mathbf{c}ri$ is even, the number of messages inserted into the ring must be integral. □

Next, we classify all the cycles in a ring. We will use this classification of cycles to study how the number of tokens on a cycle is affected by changing the number of messages on a ring.

Given a ring, r, let π_r be a pipeline formed by disconnecting the ring r at some channel. The paths in the constraint graph of π_r between its input variables and output variables can be grouped into three categories:

1. σ paths—paths from an assignment of li to an assignment of lo and paths from an assignment of ri to an assignment of ro.
2. ρ paths—paths from an assignment of li to an assignment of ro.
3. λ paths—paths from an assignment of ri to an assignment of lo.

Figure 2 labels the paths in the constraint graph of π_r between assignments to input variables of π_r and assignments to output variables of π_r. Each label consists of two parts, the first specifies which of the three groups the path belongs to and the second is an index that along with the groups specifies the source and sink of the path.

According to this classification, any cycle in the constraint graph of a ring, r, is either a cycle in the

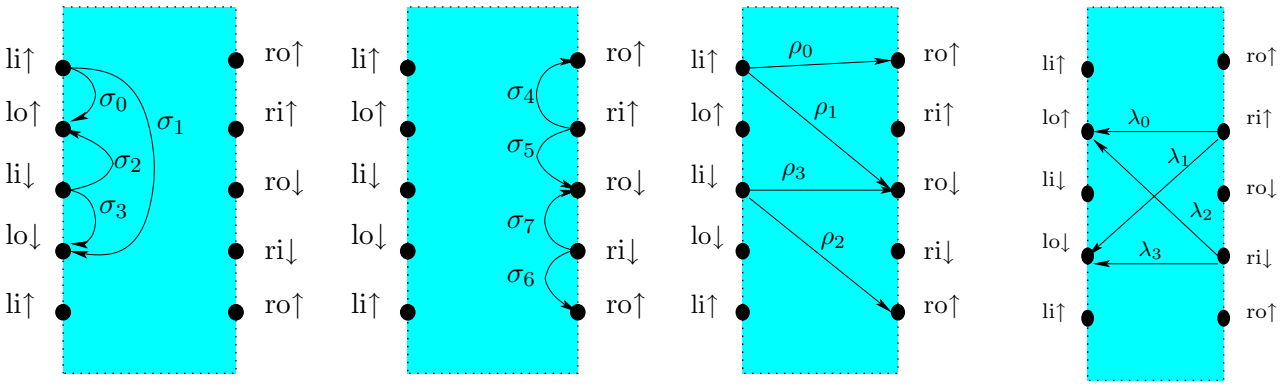


Figure 2: Classification of paths in a pipeline

constraint graph of π_r, or can be expressed as a composition of paths in π_r that matches the regular expression

$$C = F_2|F_1|Loc|B_1|B_2 \tag{4}$$

$$F_2 = \rho_1\rho_2 \tag{5}$$

$$F_1 = \rho_0|\rho_1\sigma_2\sigma_4|\rho_1\sigma_3\sigma_6|\rho_2\sigma_0\sigma_5|\rho_2\sigma_1\sigma_7|\rho_3 \tag{6}$$

$$B_2 = \lambda_1\lambda_2 \tag{7}$$

$$B_1 = \lambda_0|\lambda_1\sigma_6\sigma_0|\lambda_1\sigma_7\sigma_2|\lambda_2\sigma_4\sigma_1|\lambda_2\sigma_5\sigma_3|\lambda_3 \tag{8}$$

$$Loc = \sigma_0\sigma_4|\sigma_1\sigma_6|\sigma_2\sigma_5|\sigma_3\sigma_7|\sigma_0\sigma_5\sigma_3\sigma_6|\sigma_1\sigma_7\sigma_2\sigma_4|\rho_1\sigma_2\lambda_1\sigma_6|\rho_1\sigma_3\lambda_2\sigma_4|\rho_2\sigma_0\lambda_1\sigma_7|\rho_2\sigma_1\lambda_2\sigma_5 \tag{9}$$

Consider a ring r, and pipeline π_r formed by disconnecting the ring at some point. If a cycle in the constraint graph of r consists of a path in π_r that matches one of the regular expressions (5)–(8), then the cycle will consist of paths matching the same regular expression in any pipeline formed by disconnecting r.

Let $c(v)$ denote the number of times assignment v occurs when the number of messages in π_r increases by n. After increasing the number of messages in π_r by n, the pipeline's active channel can be connected with its passive channel to form a ring. Thus the sequencing of the handshake implies that after the number of messages in π_r has increased by n:

$$c(li\uparrow) - c(ro\uparrow) = c(li\downarrow) - c(ro\downarrow) = c(lo\uparrow) - c(ri\uparrow) = c(lo\downarrow) - c(ri\downarrow) = n \tag{10}$$

$$e = c(li\uparrow) - c(li\downarrow) \in \{-1, 0, 1\} \tag{11}$$

$$a = c(li\uparrow) - c(lo\uparrow) \in \{-1, 0, 1\} \tag{12}$$

$$b = c(li\uparrow) - c(lo\downarrow) \in \{-1, 0, 1\} \tag{13}$$

	$lo\uparrow$	$lo\downarrow$	$ro\uparrow$	$ro\downarrow$
$li\uparrow$	a	b	n	$n+e$
$li\downarrow$	$a-e$	$b-e$	$n-e$	n
$ri\uparrow$	$-n$	$b-a-n$	$-a$	$-a+e$
$ri\downarrow$	$a-b-n$	$-n$	$-b$	$-b+e$

Table 1: Change in number of tokens on path when the number of messages is increased by n

Table 1 shows the change in the number of tokens on the various classes of paths the number of messages in π_r has increased by n. The table is easily derived from (10)–(13).

Lemma 4

Changing the number of messages in a ring by $n \in \mathbb{Z}$ changes the number of tokens on any cycle in a ring's constraint graph by an amount independent of the channel at which the ring was disconnected to insert the messages.

LemmaProof 4

Changing the number of messages on a ring, by an integral amount only changes the number of messages on cycles matching F_1, F_2, B_2, B_1.

If a cycle matches one of F_1, F_2, B_2, B_1 when a ring is disconnected at a particular channel, it must match the same expression, no matter where the ring is disconnected.

Inspection of table 1 and (4) reveals that inserting a message into a ring, changes the number of tokens on all cycles matching one of the expressions F_1, F_2, B_2, B_1, by the same amount. □

4 Static slack

In this section the static slack of a buffer is defined. For processes with multiple passive channels and/or active channels, the static slack of the process between a specified pair of passive and active channels is defined as for a buffer, with all other channels being connected to sources and sinks.

4.1 Pipelines

Static slack measures the maximum number of messages a buffer, or pipeline of buffers can contain.

Definition 5 Static Slack: Pipeline

The static slack of a pipeline π, is $s = (\mathbf{c}lo - \mathbf{c}ri)/2 + \text{init\_msg}(\pi)$ when no more assignments can be performed on li in the following program:

$$Init_L; I_L; *[lo' := lo; [lo \neq lo']; li := \neg li] \parallel \pi \parallel Init_R; I_R; ro' := ro; [ro \neq ro']; \langle ri := \neg ri; \mathbf{c}ri := 1\rangle$$

4.2 Rings

Definition 6 Static Slack: Ring

The static slack of a ring of buffers is the maximum number of messages that can be on the ring.

Consider a pipeline, π, of process such that its passive channel can be connect to its active channel to form a ring. If a ring of n instances of π has static slack s_n, then π must have static slack $\frac{s_n}{n}$. Since the static slack need not be integral, the static slack must be measured by considering a sequence of rings composed of π.

Definition 7 Static Slack : Pipeline when part of a ring

The static slack, s, of a pipeline, π, when part of a ring can be measured as

$$s = \lim_{n \to \infty} \frac{s_n}{n},$$

where s_n is the static slack of a ring of n instances of π.

The definition of static slack of a ring is a special case of the definition of the dynamic slack of a ring. The proof that this limit exists will follow from the proof that the corresponding limit in the definition of the dynamic slack exists.

5 Dynamic slack and Dynamic Threshold

The cycle time of a ring of buffers varies with the number of messages on the ring. Inserting a message into the ring adds tokens to some cycles and removes tokens from others. This imposes a two sided constraint on the number of messages needed for a ring to maintain a specified cycle time, τ. The upper bound will be referred to as the *dynamic slack* of the ring at τ, and the lower bound as the *dynamic threshold* at τ.

In the interest of clarity, the dynamic slack and dynamic threshold are defined for a pipeline of buffers. For pipelines of processes with multiple passive channels and/or multiple active channels, dynamic slack and dynamic threshold between the L and R channels of the pipeline are defined as for a pipeline of buffers, with all other channels being connected to infinitely fast sources or sinks.

5.1 Dynamic Slack

We will define the dynamic slack of a ring, the dynamic slack of a pipeline, when part of a ring and the dynamic slack of a pipeline when part of a larger pipeline. We will show that the dynamic slack of a pipeline when part of a ring equals the dynamic slack of a pipeline when part of a larger pipeline.

5.1.1 Ring

Definition 8 Dynamic Slack: Ring

The dynamic slack $ds(r, \tau)$, of a ring of buffers, r, at cycle time, τ, is defined as the maximum number of messages that r can contain such that its cycle time is no greater than τ.

The dynamic slack of a pipeline, π, when part of a ring is defined only for pipelines that are initialized in such a manner that their passive and active channels can be connected to form a ring. If a ring, r_n, composed of n instances of π has dynamic slack $ds(r_n, \tau)$, then π is said to have dynamic slack at least $\frac{ds(r_n,\tau)}{n}$. The ratio of the dynamic slack of a ring to the number of processes on the ring need not be constant. Therefore, the dynamic slack is measured by considering a sequence of rings composes of n instances of π.

A ring composed of n instances of π may not be able to operate at the cycle time τ, for all $n \in \mathbb{N}$. We consider rings composed of $L(i)$ instances of π where $L(i)$ is the i^{th} smallest $n \in \mathbb{N}$ such that a ring of n instances of π can operate at cycle time τ.

Definition 9 Dynamic Slack: Pipeline when part of a ring

The dynamic slack, $ds_{ring}(\pi, \tau)$, of a pipeline, π, when part of a ring is defined as

$$ds_{ring}(\pi, \tau) \stackrel{\text{def}}{=} \lim_{i \to \infty} \frac{ds(r_{L(i)}, \tau)}{L(i)},$$

where $r_{L(i)}$ is a ring of $L(i)$ instances of π.

Note that $ds_{ring}(\pi, \infty)$ is the static slack of the pipeline when part of a ring.

We proceed to show that this limit exists. We first prove a lemma that relates the dynamic slack of a ring composed of n instances of a pipeline and the dynamic slack of rings composed $i \cdot n, i \in \mathbb{N}$ instances of the pipeline. This lemma is used to prove that the limit exists.

Lemma 5

Let r_n be a ring of n instances of a pipeline, π that has cycle time at most τ. There exists $\kappa_n \in \mathbb{R}$ such that

$$ds(r_n, \tau) = \text{init\_msg}(r_n) + \lfloor \kappa_n \rfloor \text{ and}$$
$$ds(r_{i \cdot n}, \tau) = i \cdot \text{init\_msg}(r_n) + \lfloor i \cdot \kappa_n \rfloor$$

for all $i \in \mathbb{N}$ where $r_{i \cdot n}$ is a ring of $i \cdot n$ instances of π.

LemmaProof 5

Consider any simple cycle, j, in the constraint graph of r_n. Let Δ_j be the delay along this cycle, k_j the number of tokens initially present on the cycle, and α_j the change in number of tokens on this cycle when a message is added to the ring.

Since the ring can operate at the target cycle time, all cycles in the constraint graph of r_n must satisfy (3) when there are initially $ds(r_n, \tau)$ messages on the ring. Thus, all cycles, j, satisfy (14)–(16) where $m = ds(r_n, \tau) - \text{init\_msg}(r_n)$.

$$\frac{k_j\tau - \Delta_j}{\alpha_j\tau} \geq -m \qquad \text{if } \alpha_j > 0. \tag{14}$$

$$k_j\tau - \Delta_j \geq 0 \qquad \text{if } \alpha_j = 0. \tag{15}$$

$$\frac{k_j\tau - \Delta_j}{\alpha_j\tau} \leq -m \qquad \text{if } \alpha_j < 0. \tag{16}$$

Let $C(r_n)$ be the set of cycles in the constraint graph of r_n. The dynamic slack of r_n is given by

$$ds(r_n, \tau) = \text{init\_msg}(r_n) + \left\lfloor \frac{\Delta_c - k_c\tau}{\alpha_c \cdot \tau} \right\rfloor, \tag{17}$$

where c is a cycle in the constraint graph of r_n such that

$$\frac{\Delta_c - k_c\tau}{\alpha_c \cdot \tau} = \min_{j \in C(r_n)} \frac{\Delta_j - k_j\tau}{\alpha_j \cdot \tau}.$$

We now express the dynamic slack of the ring $r_{i \cdot n}$ in a similar manner. Observe that $r_{i \cdot n}$ has i initial messages for each initial message in ring r_n. Thus, $ds(r_{i \cdot n}, \tau)$ is given by

$$ds(r_{i \cdot n}, \tau) = \text{init\_msg}(r_{i \cdot n}) + \left\lfloor \frac{\Delta_{c'} - k_{c'}\tau}{\alpha_{c'} \cdot \tau} \right\rfloor \tag{18}$$

$$= i \cdot \text{init\_msg}(r_n) + \left\lfloor \frac{\Delta_{c'} - k_{c'}\tau}{\alpha_{c'} \cdot \tau} \right\rfloor, \tag{19}$$

where c' is a cycle such that

$$\frac{\Delta_{c'} - k_{c'}\tau}{\alpha_{c'} \cdot \tau} = \min_{j \in C(r_{i \cdot n})} \frac{\Delta_j - k_j\tau}{\alpha_j \cdot \tau}.$$

Any cycle in the constraint graph of $r_{i \cdot n}$ can be written as a composition of cycles in r_n. For each cycle j in r_n, let v_j be a non-negative integer that denotes the number of times that cycle j appears when the c' is expressed as a composition of cycles in r_n.

$$\Delta_{c'} = \sum_j v_j \cdot \Delta_j \tag{20}$$

$$k_{c'} = \sum_j v_j \cdot k_j \tag{21}$$

$$i \cdot \alpha_{c'} = \sum_j v_j \cdot \alpha_j \tag{22}$$

The cycle c' must have $\alpha < 0$. Furthermore, from the arguments in section 2.1.1, only simple cycles need

to be considered. Table 1 and (4) show that simple cycles have $|\alpha| \leq 2$. If $\alpha_{c'} = -1$, then the cycle $c^\dagger$ that traverses each edge in cycle c' twice has $\alpha_{c^\dagger} = -2$ and

$$\frac{\Delta_{c'} - k_{c'}\tau}{\alpha_{c'} \cdot \tau} = \frac{\Delta_{c^\dagger} - k_{c^\dagger}\tau}{\alpha_{c^\dagger} \cdot \tau}.$$

Thus we can restrict our attention to the case where $\alpha_{c'} = -2$.

From (20)–(22), and the definition of c', we have that

$$\frac{\Delta_{c'} - k_{c'}\tau}{\alpha_{c'} \cdot \tau} = \min_{\sum_j v_j \cdot \alpha_j = -2i} \frac{i \sum_j v_j \left(\Delta_j - k_j\tau\right)}{\tau \sum_j v_j \cdot \alpha_j} \tag{23}$$

$$= \min_{\sum_j v_j \cdot \alpha_j = -2i} \frac{i \sum_j v_j \left(k_j\tau - \Delta_j\right)}{-\tau \sum_j v_j \cdot \alpha_j}. \tag{24}$$

Note that i, τ and v_j are non-negative. From (15) we note that when (24) is minimized $v_j = 0 \; \forall j : \alpha_j = 0$. Thus,

$$\frac{\Delta_{c'} - k_{c'}\tau}{\alpha_{c'} \cdot \tau} = \min_{\sum_j v_j \cdot \alpha_j = -2i} \frac{\sum_{j:\alpha_j \neq 0} v_j \alpha_j \left(\frac{k_j\tau - \Delta_j}{\alpha_j}\right)}{2\tau}. \tag{25}$$

(25) is minimized when $v_j = 0 \; \forall j \neq c$, $v_c = \frac{2i}{|\alpha_c|}$.

Therefore,

$$ds(r_{i \cdot n}, \tau) = i \cdot \text{init\_msg}(r_n) + \left\lfloor \frac{2i\left(\Delta_c - k_c\tau\right)}{-2\left|\alpha_c\right| \cdot \tau} \right\rfloor \tag{26}$$

$$= i \cdot \text{init\_msg}(r_n) + \left\lfloor \frac{i\left(\Delta_c - k_c\tau\right)}{\alpha_c \cdot \tau} \right\rfloor. \tag{27}$$

This proves the lemma for $\kappa_n = \frac{\Delta_c - k_c\tau}{\alpha_c \cdot \tau}$ □

Theorem 1

Consider a pipeline π for which a ring of h instances of π operate at cycle time at most τ. There exists $\kappa_h \in \mathbb{R}$ such that if $ds(r_h, \tau) = \text{init\_msg}(r_h) + \lfloor \kappa_h \rfloor$, $ds_{ring}(\pi, \tau) = \text{init\_msg}(\pi) + \frac{\kappa_h}{h}$.

Proof 1

We use the notation r_n to denote a ring consisting of n instances of a pipeline π.

Recall that

$$ds_{ring}(\pi, \tau) \stackrel{\text{def}}{=} \lim_{i \to \infty} \frac{ds(r_{L(i)}, \tau)}{L(i)}$$

In order to prove the theorem, we need to show that for any $\epsilon > 0$, there exists I such that for all $i \geq I$,

$$\left| \frac{ds(r_{L(i)}, \tau)}{L(i)} - \text{init\_msg}(\pi) + \frac{\kappa_h}{h} \right| \leq \epsilon$$

In order to prove this, we bound the difference between $ds(r_n, \tau)$ and $(\text{init\_msg}(\pi) + \frac{\kappa_h}{h})$.
For $n \in [j \cdot h, (j+1) \cdot h]$, from lemma 5

$$ds(r_{j \cdot n \cdot h}, \tau) \leq j \cdot h \cdot (ds(r_n, \tau) + 1) \tag{28}$$

Using lemma 5, substitute for the left hand side of the inequality

$$j \cdot n \cdot \text{init\_msg}(r_h) + \lfloor j \cdot n \cdot \kappa_h \rfloor \leq j \cdot h \cdot (ds(r_n, \tau) + 1) \tag{29}$$

Rewriting

$$n \left(j \cdot \text{init\_msg}(r_h) + \lfloor j \cdot \kappa_h \rfloor \right) \leq j \cdot h \cdot (ds(r_n, \tau) + 1) \tag{30}$$

$$j \cdot \text{init\_msg}(r_h) + \lfloor j \cdot \kappa_h \rfloor \leq ds(r_n, \tau) + 1 \tag{31}$$

From lemma 5,

$$(j+1) \cdot h \cdot ds(r_n, \tau) \leq ds(r_{(j+1) \cdot n \cdot h}, \tau) \tag{32}$$

Applying the lemma to the right hand side and rewriting,

$$(j+1) \cdot h \cdot ds(r_n, \tau) \leq (j+1) \cdot n \cdot \text{init\_msg}(r_h) + (j+1) \cdot n \cdot \kappa_h \tag{33}$$

$$\leq n \left((j+1) \cdot \text{init\_msg}(r_h) + (j+1) \cdot \kappa_h \right) \tag{34}$$

$$ds(r_n, \tau) \leq (j+1) \cdot \text{init\_msg}(r_h) + (j+1) \cdot \kappa_h \tag{35}$$

Recall that $\text{init\_msg}(r_h) = h \cdot \text{init\_msg}(\pi)$. Let $X = \text{init\_msg}(r_h)$. Thus,

$$\frac{ds(r_n, \tau)}{n} - \frac{X + \kappa_h}{h} \in \left[\frac{j \cdot (X + \kappa_h) - 2}{(j+1) \cdot h} - \frac{X + \kappa_h}{h}, \frac{(j+1) \cdot (X + \kappa_h)}{j \cdot h} - \frac{X + \kappa_h}{h} \right] \tag{36}$$

$$\in \left[\frac{-\text{init\_msg}(r_h) - \kappa_h - 2}{(j+1) \cdot h}, \frac{\text{init\_msg}(r_h) + \kappa_h}{j \cdot h} \right] \tag{37}$$

Thus, for $L(i) \in [j \cdot h, (j+1) \cdot h]$,

$$\left| \frac{ds(r_{L(i)}, \tau)}{L(i)} - \frac{\text{init\_msg}(r_h) + \kappa_h}{h} \right| \leq \frac{2 + \text{init\_msg}(r_h) + \kappa_h}{j \cdot h} \tag{38}$$

Recall that $L(i)$ is increasing in i. Thus, for all $i > I$ such that $L(I) > \frac{2+\text{init\_msg}(r_h)+\kappa_h}{\epsilon} + h$, (39) holds.

$$\left| \frac{ds(r_{L(i)}, \tau)}{L(i)} - \frac{\text{init\_msg}(r_h) + \kappa_h}{h} \right| \leq \epsilon \tag{39}$$

This proves the theorem. □

5.1.2 Pipelines

In this section we study the dynamic slack of a pipeline, π. The dynamic slack is the maximum number of messages an instance of π can contain whilst maintaining a cycle time of τ.

Definition 10 Dynamic Slack: Pipeline

Consider a sequence of pipelines, $\{\pi_n\}$, such that π_n is the pipeline consisting of n instances of π.

The dynamic slack of π is

$$ds_{pip}(\pi, \tau) \stackrel{\text{def}}{=} \lim_{n \to \infty} \frac{m_n}{n},$$

where m_n is the maximum number of messages in π_n during the steady state of any fastest linear execution of the system

$$L \parallel \pi_n \parallel R$$

where L and R are sources/sinks that have delay 0 between their input and output variables. It is required that there be a delay of τ between two successive actions on communication variables assigned to by R.

In this section, when we refer to an execution of a pipeline, if the environment of the pipeline is unspecified, the environment of the pipeline is assumed to be that described in definition 10.

We will prove a lemma that relates the dynamic slack of a ring of buffers to the maximum number of messages in the steady state of a fastest linear execution of a pipeline obtained by disconnecting the ring. This lemma will be used to prove that the limit in definition 10 exists, and is equal to the dynamic slack of the pipeline when part of a ring.

Lemma 6

Consider a ring r_n, that consists of n instances of a pipeline, π. If r_n can operate at cycle time τ or lower, then any fastest linear execution of π_{2n} contains no more than $ds(r_{2n}, \tau) + 1$ messages in the steady state.

LemmaProof 6

Using the notation from section 3.2.3, the cycle in the constraint graph of the ring r_{2n} that limits the maximum number of messages on the ring is a cycle that is either a λ_0 or λ_3 path in π_{2n}. Label this path p. Since r_{2n} has cycle time at most τ, the ratio of the delay along the cycle to the number of tokens on the cycle is less then τ.

Consider a pipeline π_{2n}, and its fastest linear execution. In the stead state, the number of tokens on p varies by at most one. The path p needs to have at least $\frac{\delta(p)}{\tau}$ tokens at during the steady state of a fastest linear execution of π_{2n}, where $\delta(p)$ is the delay along p. Thus, during a fastest linear execution π_{2n} must have at least $\frac{\delta(p)}{\tau} - 1$ tokens on p in the steady state. This corresponds to there being at most $ds(r_{2n}, \tau) + 1$ messages in π_{2n}. □

Theorem 2

If there exists h such that a ring composed of h instances of π has an execution with cycle time at most τ, then $ds_{pip}(\pi, \tau) = ds_{ring}(\pi, \tau)$.

Proof 2

The proof of this theorem is structured in the same fashion as that of theorem 1. We will bound the difference between m_n and $ds_{ring}(\pi, \tau)$ for all $n > h$.

Given a linear execution of ring $r_{2j \cdot h}$ with $ds(r_{2j \cdot h}, \tau)$ messages such that the cycle period of any vertex is exactly τ, an execution of pipeline $\pi_n : n \geq 2j \cdot h$ can be constructed by setting the state of $k'th$ process on the pipeline to be the state of the process $(k \mod 2j \cdot h)^{th}$ on the ring. During this execution there are at least $ds(r_{2j \cdot h}, \tau)$ messages in π_n.

From theorem 1, $2j \cdot h \cdot ds_{ring}(\pi, \tau) - 1 \leq ds(r_{2j \cdot h}, \tau) \leq 2j \cdot h \cdot ds_{ring}(\pi, \tau)$ Thus

$$m_n \geq 2j \cdot h \cdot ds_{ring}(\pi, \tau) - 1 \tag{40}$$

For all pipelines π_n such that $n \leq 2\,(j+1) \cdot h$. m_n must satisfy (41). Applying lemma 6, we obtain (42).

$$m_n < m_{2(j+1) \cdot h} \tag{41}$$

$$< 2\,(j+1) \cdot h \cdot ds_{ring}(\pi, \tau) + 1 \tag{42}$$

Thus, for any pipeline π_n such that $n \in [2j \cdot h, 2\,(j+1)\,h]$,

$$\frac{m_n}{n} - ds_{ring}(\pi, \tau) \in \left[\frac{2j \cdot h \cdot ds_{ring}(\pi, \tau) - 1}{2(j+1)h} - ds_{ring}(\pi, \tau), \frac{2\,(j+1) \cdot h \cdot ds_{ring}(\pi, \tau) + 1}{2j \cdot h} - ds_{ring}(\pi, \tau) \right] \tag{43}$$

$$\in \left[\frac{-2h \cdot ds_{ring}(\pi, \tau) - 1}{2(j+1)h}, \frac{2h \cdot ds_{ring}(\pi, \tau) + 1}{2j \cdot h} \right], \text{, and} \tag{44}$$

$$\left| \frac{m_n}{n} - ds_{ring}(\pi, \tau) \right| \leq \frac{ds_{ring}(\pi, \tau) + \frac{1}{2h}}{j}. \tag{45}$$

Note that for any $\epsilon > 0$, there exists $N = \frac{ds_{ring}(\pi,\tau)+\frac{1}{2h}}{\epsilon}$ such that for all $n > 2(N+1)h$,

$$\left|\frac{m_n}{n} - ds_{ring}(\pi,\tau)\right| < \epsilon. \tag{46}$$

Thus,

$$ds_{pip}(\pi,\tau) \stackrel{\text{def}}{=} \lim_{n\to\infty} \frac{m_n}{n} = ds_{ring}(\pi,\tau). \tag{47}$$

□

5.2 Dynamic Threshold

We will define the dynamic threshold of a ring, the dynamic threshold of a pipeline, when part of a ring and the dynamic threshold of a pipeline when part of a larger pipeline. We will show that the dynamic threshold of a pipeline when part of a ring equals the dynamic threshold of a pipeline when part of a larger pipeline. The definitions presented in this section are very similar to the definitions of dynamic slack presented in section 5.1. Like dynamic slack, dynamic threshold is defined as a limit. The proofs that these limits exist closely mirrors the corresponding proofs for dynamic slack. In this section we will simply state the theorems. The proofs of these theorems are provided in appendix A.

5.2.1 Ring

Definition 11 Dynamic Threshold: Ring
The dynamic threshold $dt(r,\tau)$, of a ring of buffers, r, at cycle time,τ, is defined as the minimum number of messages that r must contain such that its cycle time is no greater than τ.

Next we define the dynamic threshold of a pipeline, π when part of a ring. The dynamic threshold of a pipeline, when part of a ring is only defined when the input and output channels of the pipeline are initialized in such a manner that the input channel can be be connected to the output channel to form a ring.

If a ring, r_n, composed of n instances of π has dynamic threshold $dt(r_n,\tau)$, then π must have dynamic threshold $\frac{dt(r_n,\tau)}{n}$. The dynamic threshold need not be rational, therefore, the dynamic threshold is measured by considering a sequence of rings composed of n instances of π.

A ring composed of n instances of π may not be able to operate at the target cycle time for all $n \in \mathbb{N}$. Thus, we consider rings composed of $L(i)$ instances of π where $L(i)$ is the i^{th} smallest $n \in \mathbb{N}$ such that a ring of n instances of π can operate at the target cycle time.

Definition 12 Dynamic Threshold: Pipeline when part of a ring

The dynamic threshold, $dt_{ring}(\pi, \tau)$, of a pipeline, π, when part of a ring is defined as

$$dt_{ring}(\pi, \tau) \stackrel{\text{def}}{=} \lim_{i \to \infty} \frac{dt(r_{L(i)}, \tau)}{L(i)}$$

where $r_{L(i)}$ is a ring of $L(i)$ instances of π.

We now proceed to show that this limit exists. This proof is very similar to the corresponding proof for dynamic slack. We first state a lemma that relates the dynamic threshold of a ring composed of n instances of π to that of a ring composed of $i \cdot n, i \in \mathbb{N}$ instances of π. This lemma can be used to prove that the limit exists.

Lemma 7

Let r_n be a ring of n instances of a pipeline, π that has cycle time at most τ. There exists $\kappa_n \in \mathbb{R}$ such that

$$dt(r_n, \tau) = \text{init\_msg}(r_n) + \lceil \kappa_n \rceil \text{ and}$$
$$dt(r_{i \cdot n}, \tau) = i \cdot \text{init\_msg}(r_n) + \lceil i \cdot \kappa_n \rceil$$

for all $i \in \mathbb{N}$ where $r_{i \cdot n}$ is a ring of $i \cdot n$ instances of π.

Theorem 3

Consider a pipeline π for which there exists h such that a ring of h instances of π has cycle time at most τ. There exists $\kappa_h \in \mathbb{R}$ such that if $\frac{dt(r_h, \tau)}{h} = \text{init\_msg}(r_h) + \lceil \kappa_h \rceil$, $dt_{ring}(\pi, \tau) = \text{init\_msg}(\pi) + \frac{\kappa_h}{h}$.

5.2.2 Pipelines

In this section we study the dynamic threshold of a pipeline, π. The dynamic threshold is the minimum number of messages an instance of π, can contain, whilst maintaining a cycle time of τ.

Definition 13 Dynamic Threshold: Pipeline

Consider a sequence of pipelines, $\{\pi_n\}$, such that π_n is the pipeline consisting of n instances of π. Let each instance of π contain the minimum number of messages such that π can be composed with itself.

The dynamic threshold of π is

$$dt_{pip}(\pi, \tau) \stackrel{\text{def}}{=} \lim_{n \to \infty} \frac{m_n}{n}$$

where m_n is the minimum number of messages in π_n in the steady state of any fastest linear execution of the system:

$$L \parallel \pi_n \parallel R$$

where L is source that has delay 0 between its input and output variables. However, there is a delay of τ between two successive actions on communication variables assigned by L. R is an infinitely fast sink.

In this section, when we refer to the execution of a pipeline, if the environment of the pipeline is unspecified, the environment of the pipeline is assumed to be that described in definition 13.

We will state a lemma that relates the dynamic threshold of a ring of buffers to the minimum number of messages in the steady state of a pipeline obtained by disconnecting the ring. This lemma can be used to prove that the limit in definition 13 exists, and is equal to the dynamic threshold of the pipeline when part of a ring. Since the proofs are very similar to the corresponding proofs for the dynamic slack, we list the proofs in appendix A

Lemma 8

Consider a ring r_n, that consists of n instances of a pipeline, π. Let π_n be the pipeline that consists of n instances of π. If r_n can operate at cycle time τ or lower, then in any fastest linear execution of π_{2n}, π_{2n} contains no less than $dt(r_{2n}, \tau) - 1$ messages.

Theorem 4

If there exists h such that a ring composed of h instances of π has an execution with cycle time at most τ, then $dt_{pip}(\pi, \tau) = dt_{ring}(\pi, \tau)$.

6 Dynamic Slack and Dynamic Threshold of Half-Buffers

Lines [4] presents all the possible reshufflings of an $*[L; R]$ buffer that do not require data to be stored on state variables within the buffer. Of these reshufflings, PCHB, WCHB, B1,B4, and B5 have static slack $\frac{1}{2}$. Sufficient conditions are presented under which the dynamic slack (and dynamic threshold) of a pipeline composed of processes with any of the first four reshufflings is the sum of that of the processes in the pipeline.

6.1 Buffers

First, systems composed of buffers are considered. We will define bounds on the delays of the various paths between the input and output variables of a buffer. We will further need to make some assumptions about the delays of adjacent buffers in a pipeline. This, along with assumptions on the intersections of paths, will be used to identify the cycles that limit the minimum and maximum number of messages that a pipeline can contain at a specified cycle time τ.

All the paths between input and output variables of a process are classified as in section 3.2.3. We will use the notation $\lambda_0(p_0)$ to denote a λ_0 path of the process p_0.

Path	Delay	Tokens	Path	Delay	Tokens
λ_0	$\leq b_i + \frac{\tau}{2}$	1	σ_0	$= x_i$	1
λ_1	$= b_i$	0	σ_1	$\leq u_i + \frac{\tau}{2}$	1
λ_2	$= b_i$	1	σ_2	$\leq v_i + \frac{\tau}{2}$	1
λ_3	$\leq b_i + \frac{\tau}{2}$	1	σ_3	$= r_i$	0
ρ_0	$= f_i$	0	σ_4	$\leq w_i + \frac{\tau}{2}$	0
ρ_1	$\leq f_i + \frac{\tau}{2}$	1	σ_5	$= y_i$	0
ρ_2	$\leq f_i + \frac{\tau}{2}$	0	σ_6	$= s_i$	0
ρ_3	$\leq f_i + k_i\tau$	k_i	σ_7	$\leq z_i + \frac{\tau}{2}$	1

Table 2: Bounds on delays in process p_i

$j \in \{i\}$	$f_i + b_j \leq \frac{\tau}{2}$			
$j \in \{i, i+1\}$	$w_i + x_j \leq \frac{\tau}{2}$	$w_i + u_j \leq \frac{\tau}{2}$	$z_i + v_j \leq \frac{\tau}{2}$	$z_i + r_j \leq \frac{\tau}{2}$
$j \in \{i, i+1\}$	$w_i + v_j \leq \frac{\tau}{2}$	$s_i + r_j \leq \frac{\tau}{2}$	$y_i + x_j \leq \frac{\tau}{2}$	$z_i + u_j \leq \frac{\tau}{2}$
$j \in \{i, i+1, i+2\}$	$s_i + u_j \leq \frac{\tau}{2}$	$s_i + x_j \leq \frac{\tau}{2}$	$y_i + v_j \leq \frac{\tau}{2}$	$y_i + r_j \leq \frac{\tau}{2}$

Table 3: Constraints on delays of connected processes

In order to simplify the set of possible cycles in a constraint graph, we need to make assumption 1. Note that the constraint graph of a half buffer with reshuffling $B5$ does not satisfy the assumption. However, half buffers implementing one of the other four reshufflings do satisfy the assumption.

Assumption 1

For any process, p

1. all ρ_1 and ρ_2 paths intersect all λ_i paths
2. all ρ_0 paths intersect all λ_i paths such that $i \neq 1$
3. all ρ_3 paths intersect all λ_i paths such that $i \neq 2$
4. all ρ_0 paths intersect all σ_2 paths
5. all ρ_3 paths intersect all σ_1 paths

Table 2 defines bounds on the delays between input and output variables of a process p_i, and lists the number of tokens on such paths when there are no messages in the process. The bounds on the delays along the paths depend on the specified time, τ.

Assumption 2

The delay on the paths between input and output variables of a process must satisfy the constraints in table 2. Furthermore, the constraints in table 3 must be satisfied for all processes.

We will consider systems composed of processes that have one of the four remaining reshufflings

$$PCHB \equiv [\mathbf{true}]; lo\uparrow; [ri]; ro\downarrow; [\neg li]; lo\downarrow; [\mathbf{true}];$$
$$*[[\neg ri \wedge li] ro\uparrow; [\mathbf{true}]; lo\uparrow; [ri]; ro\downarrow; [\neg li]; lo\downarrow; [\mathbf{true}]]$$
$$WCHB \equiv [\mathbf{true}]; lo\uparrow; [ri \wedge \neg li]; ro\downarrow; [\mathbf{true}]; lo\downarrow; [\mathbf{true}];$$
$$*[[\neg ri \wedge li]; ro\uparrow; [\mathbf{true}]; lo\uparrow; [ri \wedge \neg li]; ro\downarrow; [\mathbf{true}]; lo\downarrow; [\mathbf{true}]]$$
$$B1 \equiv [\mathbf{true}]; lo\uparrow; [ri \wedge \neg li]; lo\downarrow; [\mathbf{true}]; ro\downarrow; [\mathbf{true}];$$
$$*[[\neg ri \wedge li]; ro\uparrow; [\mathbf{true}]; lo\uparrow; [ri \wedge \neg li]; lo\downarrow; ro\downarrow; [\mathbf{true}]]$$
$$B4 \equiv [\mathbf{true}]; lo\uparrow; ([ri]; ro\downarrow), ([ri \wedge \neg li]; lo\downarrow); [\mathbf{true}];$$
$$*[[\neg ri \wedge li]; ro\uparrow; [\mathbf{true}]; lo\uparrow; ([ri]; ro\downarrow), ([ri \wedge \neg li]; lo\downarrow); [\mathbf{true}]]$$

All process are initialized with the program:

$$init \equiv lo\downarrow, ro\uparrow, [\neg ri \wedge \ \ li]$$

Figure 3 shows the constraint graphs for such buffers.

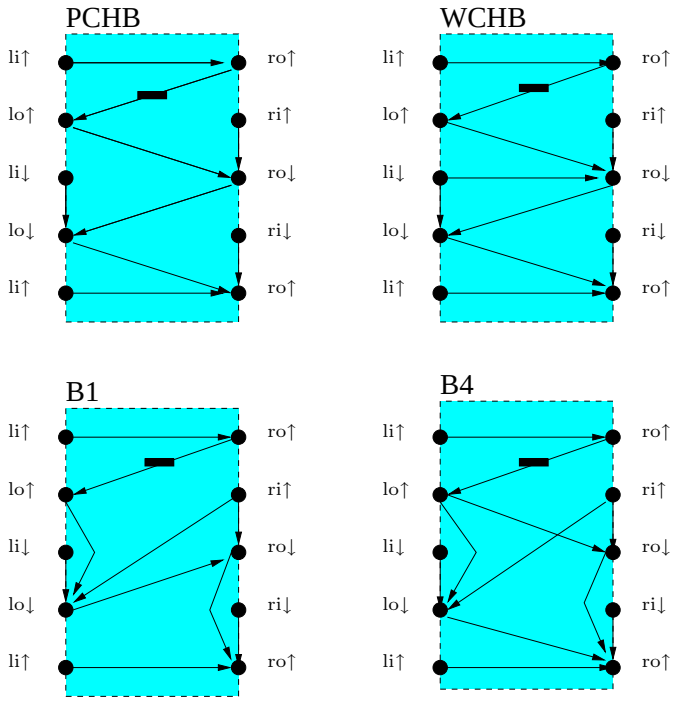


Figure 3: Constraint graphs of half buffers

In addition to assumptions 1 and 2, it is assumed that the ratio of the delay to the number of tokens on cycles that traverse exactly one process is at most τ. If this were not the case, the dynamic slack and dynamic threshold for any system composed of such processes would be undefined. Adding buffers to the system does not change the number of tokens on such cycles.

The L channel of a buffer is referred to as its input channel, and the R channel its output channel. We will refer to cycles in a pipeline of processes matching regular expression (9) as *local cycles*.

Lemma 9

If assumptions 1 and 2 hold, all simple local cycles traverse no more than 3 processes. Furthermore, no simple local cycle constrains the cycle time to be greater than τ.

LemmaProof 9

Consider a local cycle traversing greater than 3 processes. There must be at least one process, p_0, such that the cycle traverses no $\lambda_i(p_0) : i \in \{0,1,2,3\}$ and no $\rho_i(p_0) : i \in \{0,1,2,3\}$ paths, but traverse at least one $\sigma_j(p_0) : j \in \{4,5,6,7\}$ path.

Let p_1 and p_2 be the processes such that the input channel of p_2 is the output channel of p_1 and the input channel of p_1 is the output channel of p_0.

If the local cycle traverse greater than 3 processes, it must traverse p_0, p_1 and p_2. Furthermore, the cycle must contain at least one λ_i and ρ_i edge from both p_1 and p_2.

The only pairs of paths, $\rho_i(p), \lambda_j(p) : i,j \in \{0,1,2,3\}$, such that the paths do not intersect are $\rho_0(p), \lambda_1(p)$ and $\rho_3(p), \lambda_2(p)$. Thus the local cycle must have one of the following sets of paths

1. $\rho_0(p_1), \rho_0(p_2), \lambda_1(p_2), \sigma_7(p_1), \sigma_2(p_2), \lambda_1(p_1)$
2. $\rho_0(p_1), \rho_3(p_2), \lambda_2(p_2), \sigma_7(p_1), \sigma_1(p_2), \lambda_1(p_1)$
3. $\rho_3(p_1), \sigma_2(p_2), \sigma_4(p_1), \rho_0(p_2), \lambda_1(p_2), \lambda_2(p_1)$
4. $\rho_3(p_1), \sigma_1(p_2), \sigma_4(p_1), \rho_3(p_2), \lambda_2(p_2), \lambda_2(p_1)$

All four sets contain paths that intersect, hence the cycle cannot be simple.

If a simple local cycle traverses three processes p_0, p_1 and p_2,the cycle can contain only on $\lambda_i(p_1)$ and one $\rho_j(p_1)$ path, where $i,j \in \{0,1,2,3\}$.

Table 4 lists all simple cycles traversing 2 processes. Table 5 lists all simple cycles traversing 3 processes. Table 3 shows that these processes cannot constrain the cycle time to be greater than τ. Changing the number of messages on a ring does not change the number of tokens on local cycles, thus these cycles can never constrain a ring's cycle time to be greater than τ. □

A cycle that matches one of the regular expressions (5)–(8) is referred as a *global cycle*. A simple global cycle can traverse a ring at most twice. Thus, we need only consider simple global cycles in even length rings that traverse a ring once. Global cycles matching one of the regular expressions (5) or (6) are referred to as *forward cycles*. Global cycles matching one of the regular expressions (7) or (8) are referred to as *backward* cycles. A *critical cycle* of a system is a cycle such that ratio of the delay along the cycle to the number of tokens on the cycle equals the system's cycle time.

Edges	Delay	Tokens
$\sigma_0(p_1)\sigma_4(p_0)$	$\leq x_1 + w_0 + \frac{\tau}{2}$	1
$\sigma_1(p_1)\sigma_6(p_0)$	$\leq u_1 + s_0 + \frac{\tau}{2}$	1
$\sigma_2(p_1)\sigma_5(p_0)$	$\leq v_1 + y_0 + \frac{\tau}{2}$	1
$\sigma_3(p_1)\sigma_7(p_0)$	$\leq r_1 + z_0 + \frac{\tau}{2}$	1
$\sigma_0(p_1)\sigma_5(p_0)\sigma_3(p_1)\sigma_6(p_0)$	$= x_1 + y_0 + r_1 + s_0$	1
$\sigma_1(p_1)\sigma_4(p_0)\sigma_2(p_1)\sigma_7(p_0)$	$\leq u_1 + v_1 + w_0 + z_0 + 2\tau$	3

Table 4: Local cycles traversing at most 2 processes

Edges	Delay	Tokens
$\rho_0(p_1)\sigma_0(p_2)\lambda_1(p_1)\sigma_6(p_0)$	$\leq f_1 + x_2 + b_1 + s_0$	1
$\rho_0(p_1)\sigma_1(p_2)\sigma_7(p_1)\sigma_2(p_2)\lambda_1(p_1)\sigma_6(p_0)$	$\leq f_1 + u_2 + z_1 + v_2 + b_1 + s_0 + \frac{3\tau}{2}$	3
$\rho_3(p_1)\sigma_3(p_2)\lambda_2(p_1)\sigma_5(p_0)$	$\leq f_1 + r_2 + b_1 + y_0 + k_1\tau$	$1 + k_1$
$\rho_3(p_1)\sigma_2(p_2)\sigma_4(p_1)\sigma_1(p_2)\lambda_2(p_1)\sigma_5(p_0)$	$\leq f_1 + v_2 + w_1 + u_2 + b_1 + y_0 + k_1\tau + \frac{3\tau}{2}$	$3 + k_1$

Table 5: Local cycles traversing 3 processes

Lemma 10

If a forward cycle constrains the cycle time of a ring of buffers satisfying assumptions 1 and 2 to be greater than τ, the forward cycle ρ_0^* is a critical cycle.

LemmaProof 10

A forward cycle contains a ρ_i path of each process on the ring. If the cycle contains more than one ρ_i path, it must also contain a λ_i edge. However, any cycle containing two $\rho_i(p)$ paths and a $\lambda_i(p)$ path will contain at least 1 pair of intersecting paths thus the cycle cannot be simple. Thus we need to only consider forward cycles with no λ_i paths.

Consider all possible paths from the $li \uparrow$ input of process p_i to the $ro \uparrow$ output of process p_i. Since a simple forward cycle cannot traverse any $\lambda_j(p)$ edges, any such paths falls in one of three categories, where p_{i-1} is the process connected to the input channel of p_i.

1. $\rho_0(p_i)$
2. $\sigma_0(p_i)\sigma_5(p_{i-1})\rho_0(p_i)$
3. $\sigma_1(p_i)\sigma_7(p_{i-1})\rho_0(p_i)$

Using the bounds from table 2, we see that if the cycle that constrains the cycle time of a ring of buffers to be τ has paths of the type 2 and 3, a cycle containing no paths of these two types also determines the cycle time of the systems.

Table 6 is used to perform similar analysis to show that forward cycles containing any $\sigma_i : i \in \{0, 1, \ldots, 7\}$ edges impose no tighter a constraint on the cycle time than a path that consists of only ρ_i edges.

$li\uparrow -ro\uparrow$ path	Delay	tokens	$li\uparrow -ro\downarrow$ path	Delay	tokens
$\rho_0(p_i)$	$= f_i$	0	$\rho_1(p_i)$	$\leq f_i + \frac{\tau}{2}$	1
$\sigma_0(p_i)\sigma_5(p_{i-1})\rho_2(p_i)$	$\leq f_i + \tau$	1	$\sigma_0(p_i)\sigma_5(p_{i-1})\rho_3(p_i)$	$\leq f_i + k_i\tau + \frac{\tau}{2}$	$1 + k_i$
$\sigma_1(p_i)\sigma_7(p_{i-1})\rho_2(p_i)$	$\leq f_i + 2\tau$	2	$\sigma_1(p_i)\sigma_7(p_{i-1})\rho_3(p_i)$	$\leq f_i + (k_i + 1)\,\tau + \frac{\tau}{2}$	$2 + k_i$
$li\downarrow -ro\uparrow$ path	delay	tokens	$li\downarrow -ro\downarrow$ path	delay	tokens
$\rho_2(p_i)$	$\leq f_i + \frac{\tau}{2}$	0	$\rho_3(p_i)$	$\leq f_i + k_i\tau$	k_i
$\sigma_2(p_i)\sigma_4(p_{i-1})\rho_0(p_i)$	$\leq f_i + \frac{3\tau}{2}$	1	$\sigma_2(p_i)\sigma_4(p_{i-1})\rho_1(p_i)$	$\leq f_i + 2\tau$	2
$\sigma_3(p_i)\sigma_6(p_{i-1})\rho_0(p_i)$	$\leq f_i + \frac{\tau}{2}$	0	$\sigma_3(p_i)\sigma_6(p_{i-1})\rho_1(p_i)$	$\leq f_i + \tau$	1

Table 6: Possible forward paths across 1 process, p_i

$\rho_0(p_i \cdot p_{i+1})$ paths	delay	tokens	$\rho_1(p_i \cdot p_{i+1})$ paths	delay	tokens
$\rho_0(p_i)\rho_0(p_{i+1})$	$= f_i + f_{i+1}$	0	$\rho_0(p_i)\rho_1(p_{i+1})$	$\leq f_i + f_{i+1} + \frac{\tau}{2}$	1
$\rho_1(p_i)\rho_2(p_{i+1})$	$\leq f_i + f_{i+1} + \tau$	1	$\rho_1(p_i)\rho_3(p_{i+1})$	$\leq f_i + f_{i+1} + \frac{(2k_{i+1}+1)\tau}{2}$	$1+ k_{i+1}$
$\rho_2(p_i \cdot p_{i+1})$ paths	delay	tokens	$\rho_3(p_i \cdot p_{i+1})$ paths	delay	tokens
$\rho_2(p_i)\rho_0(p_{i+1})$	$\leq f_i + f_{i+1} + \frac{\tau}{2}$	0	$\rho_2(p_i)\rho_1(p_{i+1})$	$\leq f_i + f_{i+1} + \tau$	1
$\rho_3(p_i)\rho_2(p_{i+1})$	$\leq f_i + f_{i+1} + \frac{(2k_i+1)\tau}{2}$	k_1	$\rho_3(p_i)\rho_3(p_{i+1})$	$\leq f_i + f_{i+1} + (k_i + k_{i+1})\,\tau$	$k_i + k_{i+1}$

Table 7: Possible critical ρ_i paths in pipeline of 2 processes, p_i and p_{i+1}.

Base case: Consider a pair of connected buffers p_i and p_{i+1}, where the output channel of p_i is connected to the input channel of p_{i+1}. Denote this pipeline of processes $p_i \cdot p_{i+1}$. Let $\rho_i(p_i \cdot p_{i+1})$ denote the class ρ_i paths in this pipeline.

Table 7 lists the paths ρ_i paths in a pipeline $p_i \cdot p_{i+1}$. The table shows the number of tokens on the paths when there are no messages in the ring. Note that if the number of tokens on a particular $\rho_i(p_i \cdot p_{i+1})$ path changes, it must change on all such paths since the paths share the same start and end points.

Inspection of table 7, shows that the claimed path is indeed critical in rings of 2 and 4 buffers.

Inductive Step: Assume that for all integers $n > 1$, the claim holds for rings of length at most $2n$. Consider a ring of length $2n + 2$. This ring can be expressed as a composition of two pipelines π_A and π_B, of even length such that the critical forward cycle of the ring can be expressed as a composition of a $\rho_0(\pi_A)$ and a $\rho_0(\pi_B)$ path or that of a $\rho_3(\pi_A)$ and $\rho_3(\pi_B)$ path.

Note that the maximum delay on a $\rho_3(\pi)$ path is $\sum f_i + \tau \sum k_i$. This path has $\sum k_i$ tokens on it initially. There is a composition of $\rho_0(\pi_A)$ and a $\rho_0(\pi_B)$ path that has delay exactly $\sum f_i$, with 0 tokens initially. Both cycle have the number of tokens changed by the same amount when a message is added to the ring. Thus the $\rho_3(\pi_A \cdot \pi_B)$ path is no more critical than the $\rho_0(\pi_A \cdot \pi_B)$ cycle. By assumption, the critical $\rho_0(\pi_A)$ and $\rho_0(\pi_B)$ paths must be ρ_0^*. □

Lemma 11

If a backward cycle constrains the cycle time of a ring to be greater than τ, then $(\lambda_1\lambda_2)^*$ is a critical backward cycle.

$ri\uparrow -lo\uparrow$ paths	delay	tokens	$ri\uparrow -lo\downarrow$ paths	delay	tokens
$\lambda_0(p_i)$	$\leq b_i + \frac{\tau}{2}$	1	$\lambda_1(p_i)$	$= b_i$	0
$\sigma_4(p_i)\sigma_1\lambda_2(p_i)$	$\leq b_i + \frac{3\tau}{2}$	2	$\sigma_4(p_i)\sigma_1\lambda_3(p_i)$	$\leq b_i + 2\tau$	2
$\sigma_5(p_i)\sigma_3\lambda_2(p_i)$	$\leq b_i + \frac{\tau}{2}$	1	$\sigma_5(p_i)\sigma_3\lambda_3(p_i)$	$\leq b_i + \tau$	1
$ri\downarrow -lo\uparrow$ paths	delay	tokens	$ri\downarrow -lo\downarrow$ paths	delay	tokens
$\lambda_2(p_i)$	$= b_i + \frac{\tau}{2}$	1	$\lambda_3(p_i)$	$\leq b_i + \frac{\tau}{2}$	1
$\sigma_6(p_i)\sigma_0(p_{i+1})\lambda_0(p_i)$	$\leq b_i + \tau$	2	$\sigma_6(p_i)\sigma_0(p_{i+1})\lambda_1(p_i)$	$\leq b_i + \frac{\tau}{2}$	1
$\sigma_7(p_i)\sigma_2(p_{i+1})\lambda_0(p_i)$	$\leq b_i + 2\tau$	3	$\sigma_7(p_i)\sigma_2(p_{i+1})\lambda_1(p_i)$	$\leq b_i + \frac{3\tau}{2}$	2

Table 8: Possible backward paths across process p_i

$\lambda_0(p_i \cdot p_{i+1})$ paths	delay	tokens	$\lambda_1(p_i \cdot p_{i+1})$ paths	delay	tokens
$\lambda_0(p_{i+1})\lambda_0(p_i)$	$\leq b_i + b_{i+1} + \tau$	2	$\lambda_0(p_{i+1})\lambda_1(p_i)$	$\leq b_i + b_{i+1} + \frac{\tau}{2}$	1
$\lambda_1(p_{i+1})\lambda_2(p_i)$	$= b_i + b_{i+1}$	1	$\lambda_1(p_{i+1})\lambda_3(p_i)$	$\leq b_i + b_{i+1} + \frac{\tau}{2}$	1
$\lambda_2(p_i \cdot p_{i+1})$ paths	delay	tokens	$\lambda_3(p_i \cdot p_{i+1})$ paths	delay	tokens
$\lambda_2(p_{i+1})\lambda_0(p_i)$	$\leq b_i + b_{i+1} + \frac{\tau}{2}$	2	$\lambda_2(p_{i+1})\lambda_1(p_i)$	$= b_i + b_{i+1}$	1
$\lambda_3(p_{i+1})\lambda_2(p_i)$	$\leq b_i + b_{i+1} + \frac{\tau}{2}$	2	$\lambda_3(p_{i+1})\lambda_3(p_i)$	$\leq b_i + b_{i+1} + \tau$	2

Table 9: Possible critical λ_* paths in pipeline of 2 processes p_i and p_{i+1}

LemmaProof 11

Any simple backward cycle must contain at least one λ_i path of each process on the ring. If a backward cycle contains a ρ_i path of a process, it must contain 2 λ_i paths of that process. However, for any process a set of 2 $\lambda_i(p)$ paths and 1 $\rho_i(p)$ path, will contain at least one pair of intersecting paths, thus such a cycle would not be simple.

Table 8 shows that for any backward path containing $\sigma_i(p)$ edges imposes no tighter a constraint on the cycle time than a path consisting solely of λ_i edges. Table 8 is derived from table 2.

Base case: Table 9 lists all the paths from input variables of the output channel to a output variables of input channels in a pipeline of two buffers. The table shows the number of tokens on these paths when no messages are present on the buffers. Note that if the number of tokens on a particular λ_i path changes, then the number of tokens on all λ_i paths must change.

Inspection of the table reveals that the claim does indeed hold for rings of length 2 and 4.

Inductive step: Assume that for all integers $n > 1$, the claim holds for rings of length at most $2n$. Consider the critical backward cycle of ring of length $2n + 2$. The ring of length $2n + 2$ can be expressed as a composition of 2 pipelines, π_A and π_B of even length with the property that the critical cycle consists of either a λ_0 path in both pipelines or a λ_3 path in both pipelines.

Since the claim is assumed to hold for the shorter pipelines, $(\lambda_1\lambda_2)^*$ and $(\lambda_2\lambda_1)^*$ must be critical λ_0 and λ_3 paths respectively. Thus the claimed path must be a critical path of the ring of length $2n + 2$. □

Theorem 5

The dynamic slack (or dynamic threshold) of a pipeline of processes satisfying assumptions 1 and 2 is the sum of the dynamic slack (or dynamic threshold) of the processes on the pipeline.

Proof 5

The theorem follows directly from lemmas 10 and 11. For any given type of buffer, clearly from lemmas 10 and 11, the dynamic slack and dynamic threshold are $\frac{\tau-2b}{2\tau}$ and $\frac{f}{\tau}$ respectively.

For a heterogenous ring, r, we have the constraints that

$$ds(r,\tau) = \frac{\sum_i \tau - 2b_i}{2\tau} \tag{48}$$

$$= \sum_i \frac{\tau - 2b_i}{2\tau} \tag{49}$$

$$dt(r,\tau) = \frac{\sum_i f_i}{\tau} \tag{50}$$

$$= \sum_i \frac{f_i}{\tau} \tag{51}$$

Furthermore, lemma 9 guarantees that no local cycles force the cycle time of the system to be greater than τ. □

6.2 Processes with more than two channels

In this section we will consider the dynamic slack and dynamic threshold of pipelines containing processes with more than two channels. We will state a set of conditions sufficient to show that if the dynamic ranges of all the pipelines in a system intersect, the system can operate at the target cycle time. In the proceeding section, we will adopt the convention that a channel l_j is implemented by variables li_j and lo_j, where li_j is an input variable to the process and lo_j an output variable.

Assumption 3

It is possible to partition the set of channels in any process P into two sets L and R such that the projection of P onto any pair of channels $l \in L$ and $r \in R$ results in a process implementing one of the reshufflings section 6.1. For a system S, any pipeline in the system satisfies conditions of section 6.1.

Let the channels in L be referred to as the input channels of P, and those in R the output channels of P. We will use the notation ${}_iP_j$ to refer to the buffer obtained by projecting P onto the variables implementing channels l_i and r_j.

Path	Delay	# Tokens
$ri_j \uparrow - ro_k \uparrow$	$\leq \delta(ri_j \uparrow, ro_j \uparrow)$	0
$ri_j \uparrow - ro_k \downarrow$	$\leq \delta(ri_j \uparrow, ro_j \downarrow) + \tau$	1
$ri_j \downarrow - ro_k \uparrow$	$\leq \delta(ri_j \downarrow, ro_j \uparrow) + \tau$	1
$ri_j \downarrow - ro_k \downarrow$	$\leq \delta(ri_j \downarrow, ro_j \downarrow)$	1

(a)

Path	Delay	# Tokens
$li_j \uparrow - lo_k \uparrow$	$\leq \delta(li_k \uparrow, lo_k \uparrow)$	1
$li_j \uparrow - lo_k \downarrow$	$\leq \delta(li_k \uparrow, lo_k \downarrow)$	1
$li_j \downarrow - lo_k \uparrow$	$\leq \delta(li_k \downarrow, lo_k \uparrow)$	1
$li_j \downarrow - lo_k \downarrow$	$\leq \delta(li_k \downarrow, lo_k \downarrow)$	0

(b)

Path	Delay
$ri_j \uparrow - ro_k \uparrow$	$\leq f_{iP_k} + b_{lP_j}$
$ri_j \uparrow - ro_k \downarrow$	$\leq f_{iP_k} + b_{lP_j} + \frac{\tau}{2}$
$ri_j \downarrow - ro_k \uparrow$	$\leq f_{iP_k} + b_{lP_j} + \frac{\tau}{2}$
$ri_j \downarrow - ro_k \downarrow$	$\leq f_{iP_k} + b_{lP_j}$

(c)

Path	Delay
$li_j \uparrow - lo_k \uparrow$	$\leq f_{jP_i} + b_{kP_l}$
$li_j \uparrow - lo_k \downarrow$	$\leq f_{jP_i} + b_{kP_l} + \frac{\tau}{2}$
$li_j \downarrow - lo_k \uparrow$	$\leq f_{jP_i} + b_{kP_l} + \frac{\tau}{2}$
$li_j \downarrow - lo_k \downarrow$	$\leq f_{jP_i} + b_{kP_l}$

(d)

Table 10: Constraints on delays of paths in processes with multiple inputs and outputs

Assumption 4

For a process P with multiple input or output channels:

- for any pair of output channels r_j and r_k of P the constraints in table 10(a) are satisfied.
- for any pair of input channels l_j and l_k of P the constraints in table 10(b) are satisfied.
- for any pair of input channels l_i, l_l and any pair of output channels r_j, r_k, the constraints in table 10(c) are satisfied.
- for any pair of input channels l_j, l_k and any pair of output channels r_i, r_l, the constraints in table 10(d) are satisfied.
- for any pair of input channels l_i, l_l and any pair of output channels r_j, r_k, $f_{iP_j} + b_{kP_l} \leq \tau$.

When these constraints hold, it can be shown, by considering all possibilities, that if all pipelines can simultaneously contain a number of messages greater than their dynamic threshold and less than their dynamic slack, then no local cycles constrain the cycle time to be greater than τ. Similarly, an exhaustive case analysis can be used to show that if the global cycles stated in lemmas 10 and 11 do not constrain the cycle time to be greater than τ, no global cycle constrains the cycle time to be greater than τ.

7 Algorithm for Slack Matching

Slack matching is an optimization performed adding buffers to a system in order to reduce the system's cycle time. In this section we formulate the slack matching problem as a mixed integer linear program (MILP).

We will consider systems that consist of processes that can be represented as constraint graphs if all the communication channels are modeled as channels carrying no data. It is assumed that the system being slack matched is closed. If the system is not closed, sources are connected to the system's inputs, and sinks to the outputs. It is assumed that the environment's delays are such that the environment does not constrain the cycle time of the system to be greater than the target cycle time, τ_0. We will first state constraints that can be used to determine whether a system is slack matched. From these constraints, we can derive a system of linear equations that must be satisfied in order to slack match a system.

A system is represented by a directed graph, $G = (V, E)$, with each vertex representing a process, and each edge a communication channel. We adopt the convention that an edge (u, v) represents a channel such that it is an active channel of process u and a passive channel of process v. For every pair of edges, (a, b) and (b, c) let $ds({}_ab_c, \tau)$ and $dt({}_ab_c, \tau)$ denote the dynamic slack and dynamic threshold of the process b between channel connected to a and that connected c at cycle time τ. Let $M({}_ab_c)$ denote the number of messages process b contains between channels connected to a and c. The cycle time of a system will be considered to be that of the slowest assignment in the system. Let $S \subset V$ denote the set of sources in the system. It is assumed that each source has exactly one output. Similarly, let $T \subset V$ denote the set of sinks in the system. It is assumed that each sink has exactly one input.

A *path* in a graph is a sequence of edges, $\{e_i\}$, such that $src(e_{i+1}) = sink(e_i)$. Let $|p|$ denote the length of a path, p. For any path, p, $src(p) = src(e_0)$ and $sink(p) = sink(e_{|p|-1})$. A *cycle* in a graph is a path p, such that $src(p) = sink(p)$.

We will refer to the closed interval, $[dt({}_ab_c, \tau), ds({}_ab_c, \tau)]$ as the *dynamic range* of process b between channels a and c.

7.1 Necessary and sufficient conditions for slack matching

We present necessary and sufficient conditions for the cycle time of a system to be τ_0, under assumptions 5 and 6. Section 6 presents sufficient conditions on systems of half-buffers for which assumptions 5 and 6 are satisfied. Under these assumptions, a system is slack matched when each of its pipelines and rings can simultaneously contain a number of messages within their dynamic range.

Assumption 5

The dynamic slack and dynamic threshold of a pipeline composed of multiple processes are the sum of the dynamic slacks and dynamic thresholds of the processes.

Assumption 6

If the number of messages in a pipeline, or ring, is within the dynamic range of the ring, at the target cycle time, τ_0, then the pipeline, or ring, has an execution with cycle time τ_0.

A process graph is said to be *slack matched* when the following system of equations can be satisfied:

$$S_{abab} = 0 \qquad \forall (a,b) \in E \tag{52}$$

$$S_{abbc} = S_{abab} + dr_{abc} - M({}_a b_c) + S_{bcbc} \qquad \forall (a,b),(b,c) \in E \tag{53}$$

$$S_{abcd} = S_{abab} + dr_{abc} - M({}_a b_c) + S_{bccd} \qquad \forall (a,b),(b,c),(c,d) \in E \tag{54}$$

$$S_{abef} = S_{abab} + dr_{abc} - M({}_a b_c) + S_{bcef} \qquad \forall (a,b),(b,c),(d,e),(e,f) \in E : d \in reachable(c) \tag{55}$$

$$S_{uuuu} = -dr_{uuu} + M({}_u u_u) \qquad \forall (u,u) \in E \tag{56}$$

$$S_{bcab} = -dr_{abc} + M({}_a b_c) \qquad \forall (a,b),(b,c) \in E : a \in reachable(c) \tag{57}$$

$$dr_{abc} \in [dt({}_a b_c, \tau_0), ds({}_a b_c, \tau_0)] \qquad \forall (a,b),(b,c) \in E \tag{58}$$

$$K_a - K_d = S_{abcd} \qquad \forall a \in S, d \in T, (a,b),(c,d) \in E \tag{59}$$

Note that the variables dr_{abc} denote a value in the dynamic range of process b between the channel (a,b) and the channel (b,c). The variables K_a represent the number of messages removed from source a, and K_d the number of messages inserted in to sink d.

For any path in the process graph between processes a and f such that the first edge of the path is (a,b) and the last is (e,f), the sum of S_{abef} and the number of initial messages in the corresponding pipeline lies within the dynamic range of the path. If there are multiple paths in the process graph between a process a and f such that the first edge of the path is (a,b) and the last is (e,f), the variables S_{abef} capture requirement that the dynamic range of the corresponding pipelines be such that there exists a number of messages that can be added to all these a–f pipelines so that the number of messages in the pipeline is within its dynamic range. This requirement arises from the fact that if a message is inserted into one of these pipelines, a message is inserted into all of the pipelines.

Lemma 12

Satisfying the system of equations (52)–(59) is equivalent to satisfying

$$S_{abab} = 0 \qquad \forall (a,b) \in E \tag{60}$$

$$S_{uvwx} = \sum_{\substack{(e_i(p),e_{i+1}(p)) \\ =((a,b),(b,c))}} dr_{abc} - M({}_a b_c) \qquad \forall p : e_0 = (u,v) \wedge e_{|p|-1} = (v,w) \tag{61}$$

$$S_{uuuu} = -dr_{uuu} + M({}_u u_u) \qquad \forall (u,u) \in E \tag{62}$$

$$S_{vwuv} = -dr_{uvw} + M({}_u v_w) \qquad \forall (u,v),(v,w) \in E : u \in reachable(w) \tag{63}$$

$$dr_{abc} \in [dt({}_a b_c, \tau_0), ds({}_a b_c, \tau_0)] \qquad \forall (a,b),(b,c) \in E \tag{64}$$

$$K_a - K_d = S_{abcd} \qquad \forall a \in S, d \in T, (a,b),(c,d) \in E \tag{65}$$

LemmaProof 12

Consider any path p in G such that $src(p) = u$ and $sink(p) = v$. If the path consists of one edge, (60)–(61) has the constraint for this path. Assume that for all paths of length n, satisfying (52)–(55) is equivalent to satisfying (60)–(61). Consider a path p of length $n+1$. Let $e_0(p) = (u, v)$, $e_1(p) = (v, s)$, and $e_n(p) = (w, x)$. Let p' denote the path from v to w such that $e_0(p') = (v, s)$ and $e_{n-1}(p') = (w, x)$. (61) can be rewritten as

$$S_{uvwx} = \sum_{\substack{(e_i(p),e_{i+1}(p)) \\ =((a,b),(b,c))}} dr_{abc} - M({}_a b_c) \tag{66}$$

$$= dr_{uvw} - M({}_u v_w) + \sum_{\substack{(e_i(p'),e_{i+1}(p')) \\ =((a,b),(b,c))}} dr_{abc} - M({}_a b_c) \tag{67}$$

$$= S_{uvuv} + dr_{uvs} - M({}_u v_s) + S_{vswx} \tag{68}$$

All other constraints are identical in the two sets of equations. □

Lemma 13

If a system operates at the target throughput, its process graph is slack matched.

LemmaProof 13

If a system operates at the target cycle time, there must exist an execution such that, simultaneously

- the number of messages in any pipeline in the system is within the pipeline's dynamic range.
- the number of messages on any cycle in the system is within the cycle's dynamic range.

Since each communication channel is used once per cycle, the number of messages on a ring is constant. Furthermore, if there exist two pipelines, π_1 and π_2, between a pair of processes a and b, the difference between the number of messages on the two pipelines is constant. (60)–(65) capture these conditions. □

Lemma 14

If a process graph is slack matched, the corresponding system operates at the target throughput.

LemmaProof 14

If a process graph is slack matched, then (52)–(59) are satisfied. Thus, there exists an execution such that the number of messages in any pipeline (or ring) of the system lies within the dynamic range of the pipeline (or ring). By assumption 6, such an execution has cycle time at most τ_0. □

7.2 MILP for slack matching

Slack matching is performed by adding buffers along communication channels in such a manner that the constraints (52)–(58) on the resulting system are satisfied.

Replacing constraints of the form $S_{abab} = 0$ by ones of the form (69) and (70) allows the variables S_{abab} to take on a value in the dynamic range of a pipeline of N_{ab} buffers. Slack matching a system now reduces to determining values of the variables N_{ab} such that the system of equations (53)–(59) and (69)–(70) is satisfied.

$$S_{ab} \in [N_{ab} \cdot dt(b, \tau_0), N_{ab} \cdot ds(b, \tau_0)] \qquad \forall (a, b) \in E \tag{69}$$

$$N_{ab} \in \mathbb{N} \qquad \forall (a, b) \in E \tag{70}$$

Since there may be multiple solutions, any cost function linear in N_{ab} may be used to drive the optimization.

7.3 Generating the MILP

The set of constraints for slack matching a system can be computed in $O(m^2 n^2)$ time where $m = |E|$ and $n = |V|$.

Constraints of the form (69),(70), (56) and (58) can be generated in $O(m)$ time by looping over the edges set.

It takes $O(mn + n^2)$ time to generate an $n \times n$ matrix R, such that

$$R_{ij} = \begin{cases} 0, & i \in reachable(j) \\ 1, & i \notin reachable(j) \end{cases} \tag{71}$$

This is done by running n breadth first searches, one rooted at each vertex. In $O(n)$ time, an array can be constructed such that indicates whether a vertex is a source or a sink. There are $O(m^2 n^2)$ 4-tuple (a, b, e, f) satisfying the condition $(a, b), (b, c), (d, e), (e, f) \in E : d \in reachable(c)$. Given matrix R, constraints of the form (53)–(55),(57) and (59) can be generated by simply looping over all such 4-tuples.

Due to cycle time constraints on the internal cycle, the number of channels a process has is usually bounded, and significantly smaller than n. In this case, the constraint can be generated in $O(k^4 n^2)$ time.

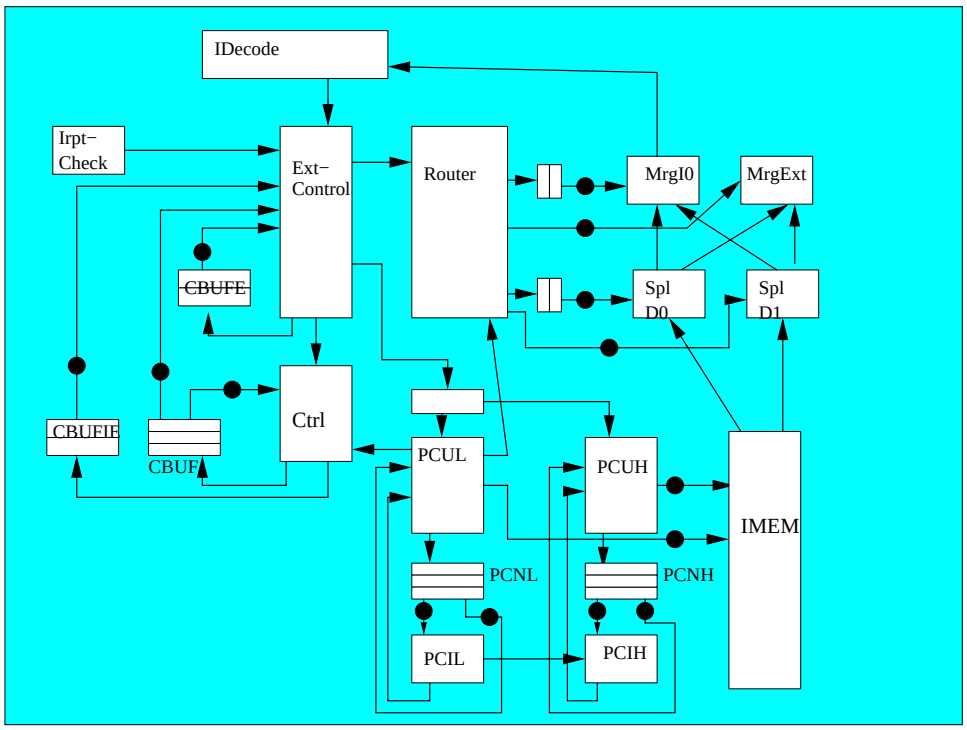


Figure 4: Lutonium fetch loop

8 Case Studies

The algorithm from section 7 was implemented in Modula-3[8] and used with `glpsol`, a freely available MILP solver. Two large examples were studied, the fetch loop of the Lutonium[7], an asynchronous 8051 microcontroller and a control loop in the fetch unit of the MiniMIPS microprocessor [6].

8.1 Example I: Lutonium Fetch Loop

This algorithm was used to slack-match the fetch loop of the Lutonium micro-controller. Whilst the instruction memory is not implemented as a pipeline of half buffers, it can be modeled as one. The memory is modeled as a stage whose dynamic threshold equals its dynamic slack. The dynamic threshold is determined by the forward latency of the memory.

Figure 4 shows the fetch loop of the Lutonium micro controller.

Table 11 shows the buffers needed to slack match the system. The table also lists the results of slack matching when performed by hand on the system. Observe, that there are fewer buffers on the byte channel in the *pc* increment loop when slack matching is performed using this algorithm.

Channel	Slack matching buffers (hand)	Slack matching buffers (MILP)
ExtControl - CBUFE	1	1
ExtControl - Router	1	1
ExtControl - IntCtrl	1	1
CBUF - IntCtrl	1	1
Router - SplD1	2	2
Router - MrgI0	1	1
Router - MrgExt	3	3
PCNH - PCIH	1	0
PCNH - PCUH	1	1
PCNL - PCUL	1	1
PCIL - PCUL	1	1

Table 11: Slack Matching buffers for Lutonium fetch

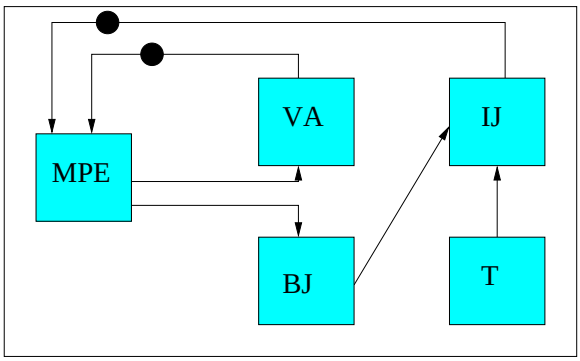


Figure 5: Control Loop in the MiniMIPS fetch

8.2 Example II: Control Loop of MiniMIPS

Figure 5 shows a loop in the fetch of the MiniMIPS. Table 12 shows the buffers required to slack match this loop. It also shows the results when slack matching was performed by hand. Note that extra buffers included when slack matching was performed by hand may be needed because a ring composed of a mixture of half buffers and full buffers was not included when generating the MILP.

Channel	Slack matching buffers (hand)	Slack matching buffers (MILP)
VA-MPE	4	1
IJ-MPE	4	1

Table 12: Slack Matching buffers for the control loop in the MiniMIPS fetch

9 Conclusion and future work

Dynamic slack and dynamic threshold of a buffer have been defined. An algorithm has been presented to slack match systems composed of processes that can be represented as collections on repetitive SLHSE in standard form, given certain assumptions about the dynamic slack and dynamic threshold of these processes. Sufficient conditions have been presented that guarantee that the dynamic slack and threshold of systems of composed of half buffers satisfy these assumptions. This algorithm has been tested on circuits from the Lutonium[7] and the MiniMIPS [6].

For the examples studied so far, solving the MILP has not proved unreasonable. As larger systems are considered, solving an MILP may take excessively large amounts of time.

Similar arguments to those in section 6 can be used to provide a class of full buffers such that systems composed of these buffers satisfy assumptions 5 and 6.

Since most real systems do not use each channel on every cycle, extending the theory to systems with conditional communications would be interesting. It would be interesting to study systems composed of buffers with different static slack, in particular the case of systems containing both half buffers and full buffers.

References

[1] S.M. Burns. *Performance Analysis and Optimization of Asynchronous Circuits*. PhD thesis, California Institute of Technology, 1990.

[2] S. Kim and P. Beerel. Pipeline optimization for asynchronous circuits:complexity analysis and an efficient optimal algorithm. In *Proc. International Conference on Computer-Aided Design*, 2000.

[3] C. Leiserson, F. Rose, and J. Saxe. Optimizing synchronous circuitry by retiming. In *Third Caltech Conference On VLSI*, March 1993.

[4] A.M. Lines. Pipelined asynchronous circuits. Master's thesis, California Institute of Technology, 1995.

[5] R. Manohar and A.J. Martin. Slack elasticity in concurrent computing. In J. Jeuring, editor, *Proc. 4th International Conference on the Mathematics of Program Construction*, Lecture Notes in Computer Science 1422, pages 272–285. Springer Verlag, 1998.

[6] A.J. Martin, A. Lines, R. Manohar, M. Nyström, P. Pénzes, R. Southworth, U. Cummings, and T.K. Lee. The design of an asynchronous MIPS R3000 microprocessor. In *Proc. 17th Conference on Advanced Research in VLSI*, 1997.

[7] A.J. Martin, M. Nyström, K. Papadantonakis, P.I. Pénzes, P. Prakash, C.G. Wong, J. Chang, K.S. Ko, B. Lee, E. Ou, J. Pugh, E. Talvala, J.T. Tong, and A Tura. The Lutonium: A sub-nanojoule asynchronous 8051 microcontroller. In *Proc. 9th IEEE Intl Symposium on Advanced Research in Asynchronous Circuits and Systems*, May 2003.

[8] G. Nelson. *Systems programming with Modula-3.* Prentice Hall, 1991.

[9] P. Pénzes. Pipeline composition for asynchronous circuits. unpublished, September 1999.

[10] C.G. Wong. *High-Level Synthesis and Rapid Prototyping of Asynchronous VLSI Systems.* PhD thesis, California Institute of Technology, 2004.

A Dynamic Threshold

We list the proofs that were omitted in section 5.2.

A.1 Proof of lemma 7

Consider any simple cycle, j in the constraint graph of r_n. Let Δ_j be the delay along this cycle, k_j, the number of tokens initially present on the cycle and α_j the change in number of tokens on this cycle when a message is added to the ring.

Since the ring can operate at the target cycle time, for all cycles in the constraint graph of r_n, (3) must be satisfied when there are $ds(r_n, \tau)$ messages on the ring. Thus, for $m = ds(r_n, \tau) - \text{init\_msg}(r_n)$ all cycles, j, satisfy (14)–(16).

Let $C(r_n)$ be the set of cycles in the constraint graph of r_n. The dynamic threshold of the ring is determined by

$$dt(r_n, \tau) = \text{init\_msg}(r_n) + \left\lceil \frac{\Delta_c - k_c\tau}{\alpha_c \cdot \tau} \right\rceil \tag{72}$$

where c is a cycle such that

$$\frac{\Delta_c - k_c\tau}{\alpha_c \cdot \tau} = \max_{j \in C(r_n)} \frac{\Delta_j - k_j\tau}{\alpha_j \cdot \tau}.$$

We now express the dynamic threshold of the $r_{i \cdot n}$ in a similar manner. Observe that ring $r_{i \cdot n}$ has i initial messages for each initial message in ring r_n. Thus $ds(r_{i \cdot n}, \tau)$ is given by

$$dt(r_{i \cdot n}, \tau) = \text{init\_msg}(r_{i \cdot n}) + \left\lceil \frac{\Delta_{c'} - k_{c'}\tau}{\alpha_{c'} \cdot \tau} \right\rceil \tag{73}$$

$$= i \cdot \text{init\_msg}(r_n) + \left\lceil \frac{\Delta_{c'} - k_{c'}\tau}{\alpha_{c'} \cdot \tau} \right\rceil \tag{74}$$

where c' is the cycle such that

$$\frac{\Delta_{c'} - k_{c'}\tau}{\alpha_{c'} \cdot \tau} = \max_{j \in C(r_{i \cdot n})} \frac{\Delta_j - k_j\tau}{\alpha_j \cdot \tau}.$$

Recall that any cycle in the constraint graph of $r_{i \cdot n}$ can be written as a composition of cycles in r_n. Thus (20) - (22) hold.

The cycle c' must have $\alpha > 0$. Furthermore, from section 2.1.1, only simple cycles need to be considered. Table 1 and (4) show that simple cycles have $|\alpha| \leq 2$. If $\alpha_{c'} = 1$, then the cycle $c^\dagger$ that traverse each edge in cycle c' twice has $\alpha_{c^\dagger} = 2$ and

$$\frac{\Delta_{c'} - k_{c'}\tau}{\alpha_{c'} \cdot \tau} = \frac{\Delta_{c^\dagger} - k_{c^\dagger}\tau}{\alpha_{c^\dagger} \cdot \tau}.$$

Thus we can restrict our attention to the case where $\alpha_{c'} = 2$.

From (20)–(22), we have that

$$\frac{\Delta_{c'} - k_{c'}\tau}{\alpha_{c'} \cdot \tau} = \max_{\sum_j v_j \cdot \alpha_j = 2i} \frac{i \sum_j v_j \left(\Delta_j - k_j \tau\right)}{\tau \sum_j v_j \cdot \alpha_j} \tag{75}$$

Note that i, τ and v_j are non-negative.

From (15) we note that when (75) is maximized, $v_j = 0 \ \forall j : \alpha_j = 0$.

$$\frac{\Delta_{c'} - k_{c'}\tau}{\alpha_{c'} \cdot \tau} = \max_{\sum_j v_j \cdot \alpha_j = 2i} \frac{\sum_{j:\alpha_j \neq 0} v_j \alpha_j \left(\frac{\Delta_j - k_j \tau}{\alpha_j}\right)}{2\tau} \tag{76}$$

Note that (76) is maximized when $v_j = 0 \ \forall j \neq c$, $v_c = \frac{2i}{|\alpha_c|}$.

Thus,

$$dt(r_{i \cdot n}, \tau) = i \cdot \text{init\_msg}(r_n) + \left\lceil \frac{2i\left(\Delta_c - k_c\tau\right)}{2\left|\alpha_c\right| \cdot \tau} \right\rceil \tag{77}$$

$$= i \cdot init\_msg(r_n) + \left\lceil \frac{i\left(\Delta_c - k_c\tau\right)}{\alpha_c \cdot \tau} \right\rceil \tag{78}$$

This proves the lemma for $\kappa_n = \frac{\Delta_c - k_c\tau}{\alpha_c \cdot \tau}$

A.2 Proof of theorem 3

We use the notation r_n to denote a ring consisting of n instances of a pipeline π.

Recall that

$$dt_{ring}(\pi, \tau) \stackrel{\text{def}}{=} \lim_{i \to \infty} \frac{dt(r_{L(i)}, \tau)}{L(i)}$$

In order to prove the theorem, we need to show that for any $\epsilon > 0$, there exists I such that for all $i \geq I$,

$$\left| \frac{dt(r_{L(i)}, \tau)}{L(i)} - \text{init\_msg}(\pi) + \frac{\kappa_h}{h} \right| \leq \epsilon$$

In order to prove this, we bound the difference between $dt(r_n, \tau)$ and $(\text{init\_msg}(\pi) + \frac{\kappa_h}{h})$.

For $n \in [j \cdot h, (j+1) \cdot h]$, from lemma 7

$$dt(r_{(j+1) \cdot n \cdot h}, \tau) \geq (j+1) \cdot h \cdot (dt(r_n, \tau) - 1) \tag{79}$$

Using lemma 7, substitute for the left hand side of the inequality

$$(j+1)\cdot n\cdot \text{init\_msg}(r_h) + \lceil (j+1)\cdot n\cdot \kappa_h \rceil \geq (j+1)\cdot h\cdot (dt(r_n,\tau)-1) \tag{80}$$

Rewriting

$$n\left((j+1)\cdot \text{init\_msg}(r_h) + \lceil (j+1)\cdot \kappa_h \rceil\right) \geq (j+1)\cdot h\cdot (dt(r_n,\tau)-1) \tag{81}$$

$$(j+1)\cdot \text{init\_msg}(r_h) + \lceil (j+1)\cdot \kappa_h \rceil \geq dt(r_n,\tau)-1 \tag{82}$$

From lemma 7,

$$j\cdot h\cdot dt(r_n,\tau) \geq dt(r_{j\cdot n\cdot h},\tau) \tag{83}$$

Applying the lemma to the right hand side and rewriting,

$$j\cdot h\cdot dt(r_n,\tau) \geq j\cdot n\cdot \text{init\_msg}(r_h) + j\cdot n\cdot \kappa_h \tag{84}$$

$$\geq n\left(j\cdot \text{init\_msg}(r_h) + j\cdot \kappa_h\right) \tag{85}$$

$$dt(r_n,\tau) \geq j\cdot \text{init\_msg}(r_h) + j\cdot \kappa_h \tag{86}$$

Recall that $\text{init\_msg}(r_h) = h\cdot \text{init\_msg}(\pi)$. Let $X = \text{init\_msg}(r_h)$. Thus,

$$\frac{dt(r_n,\tau)}{n} - \frac{X+\kappa_h}{h} \in \left[\frac{j\cdot(X+\kappa_h)}{(j+1)\cdot h} - \frac{X+\kappa_h}{h}, \frac{(j+1)\cdot(X+\kappa_h)+2}{j\cdot h} - \frac{X+\kappa_h}{h}\right] \tag{87}$$

$$\in \left[\frac{-\text{init\_msg}(r_h)-\kappa_h}{(j+1)\cdot h}, \frac{\text{init\_msg}(r_h)+\kappa_h+2}{j\cdot h}\right] \tag{88}$$

Thus, for $L(i) \in [j\cdot h, (j+1)\cdot h]$,

$$\left|\frac{dt(r_{L(i)},\tau)}{L(i)} - \frac{\text{init\_msg}(r_h)+\kappa_h}{h}\right| \leq \frac{2+\text{init\_msg}(r_h)+\kappa_h}{j\cdot h} \tag{89}$$

Recall that $L(i)$ is increasing in i. Thus for all $i > I$ such that $L(I) > \frac{2+\text{init\_msg}(r_h)+\kappa_h}{\epsilon} + h$, (90) holds.

$$\left|\frac{dt(r_{L(i)},\tau)}{L(i)} - \frac{\text{init\_msg}(r_h)+\kappa_h}{h}\right| \leq \epsilon \tag{90}$$

A.3 Proof of lemma 8

Using the notation from section 3.2.3, the cycle in the constraint graph of the ring r_{2n} that limits the minimum number of messages on the ring is a cycle that is either a ρ_0 or ρ_3 path in π_{2n}. Label this path p. Since r_{2n} has cycle time at most τ, the ratio of the delay along the cycle to the number of tokens on the

cycle is less then τ.

Consider a pipeline π_{2n}, and its fastest linear execution. During the steady state, the number of tokens on any path varies by at most one. The path p needs to have at least $\frac{\delta(p)}{\tau}$ tokens at some point during any fastest linear execution, where $\delta(p)$ is the delay along p. Thus during any fastest linear execution, π_{2n} must have at least $\frac{\delta(p)}{\tau} - 1$ tokens. This corresponds to there being at least $dt(r_{2n}, \tau) - 1$ messages in π_{2n} at any point during the steady state of a fastest linear execution.

A.4 Proof of theorem 4

The proof of this theorem is structured in the same fashion as that of theorem 3. We will bound the difference between m_n and $dt_{ring}(\pi, \tau)$ for all $n > h$.

Given a linear execution of $r_{2\cdot(j+1)\cdot h}$ with $dt(r_{2\cdot(j+1)\cdot h}, \tau)$ messages in the ring such that the cycle period of any vertex is exactly τ, an execution of $\pi_{2\cdot(j+i)\cdot h}$ can be constructed by mapping the state of the k^{th} process on the ring to the k^{th} process on the pipeline. Such an execution of $\pi_{2\cdot(j+i)\cdot h}$ will have cycle time at most τ. At any point during this execution, there are at least $dt(r_{2\cdot(j+1)\cdot h}, \tau)$ messages in the pipeline. Thus for any $n \leq 2\,(j+1)\cdot h$, (91) holds. From theorem 3, $dt(r_{2(j+1)\cdot h}, \tau) \leq (2j+1)\cdot h \cdot dt_{ring}(\pi, \tau)$. Thus,(91) simplifies to (92).

(92).

$$m_n < dt(r_{2(j+1)\cdot h}, \tau) \tag{91}$$

$$< (2j+1)\cdot h \cdot dt_{ring}(\pi, \tau) \tag{92}$$

Consider $p_n : n \geq 2j \cdot h$. $m_n \geq m_{2j\cdot h}$ From lemma 8

$$m_n \geq 2j \cdot h \cdot dt_{ring}(\pi, \tau) - 1 \tag{93}$$

Thus for any pipeline p_n, $n \in [2j \cdot h, 2\,(j+1)\,h]$,

$$\frac{m_n}{n} - dt_{ring}(\pi, \tau) \in \left[\frac{2j \cdot h \cdot dt_{ring}(\pi, \tau) - 1}{2(j+1)h} - dt_{ring}(\pi, \tau), \frac{2\,(j+1)\cdot h \cdot dt_{ring}(\pi, \tau)}{2j \cdot h} - dt_{ring}(\pi, \tau)\right] \tag{94}$$

$$\in \left[\frac{-2h \cdot dt_{ring}(\pi, \tau) - 1}{2(j+1)h}, \frac{2h \cdot dt_{ring}(\pi, \tau)}{2j \cdot h}\right] \tag{95}$$

$$\left|\frac{m_n}{n} - dt_{ring}(\pi, \tau)\right| \leq \frac{dt_{ring}(\pi, \tau) + \frac{1}{2h}}{j} \tag{96}$$

Note that for any $\epsilon > 0$, there exists $N = \frac{dt_{ring}(\pi,\tau)+\frac{1}{2h}}{\epsilon}$ such that for all $n > 2(N+1)h$,

$$\left|\frac{m_n}{n} - dt_{ring}(\pi, \tau)\right| < \epsilon \tag{97}$$

Thus

$$dt_{pip}(\pi,\tau) \stackrel{\text{def}}{=} \lim_{n\to\infty} \frac{m_n}{n} = dt_{ring}(\pi,\tau) \tag{98}$$