ISSN 1361 - 6161

THE UNIVERSITY of MANCHESTER

Computer Science
University of Manchester

Some Formal Musings on the Performance of Asynchronous Hardware.

C. Tofts

Department of Computer Science
University of Manchester
Technical Report Series
UMCS-96-2-2

Some Formal Musings on the Performance of Asynchronous Hardware.*†

C.Tofts,

Department of Computer Science,
The University,
Manchester,
M13 9PL,
email: cmnt@cs.man.ac.uk

Recent technical reports issued by the Department of Computer Science, Manchester University, are available by anonymous ftp from `ftp.cs.man.ac.uk` in the directory `pub/TR`. The files are stored as PostScript, in compressed form, with the report number as filename. They can also be obtained on WWW via
URL `http://www.cs.man.ac.uk/csonly/cstechrep/index.html`. Alternatively, all reports are available by post from The Computer Library, Department of Computer Science, The University, Oxford Road, Manchester M13 9PL, UK.

*Refereed by: Dr D. Edwards

†Chris Tofts is supported by an EPSRC Advanced Fellowship.

Abstract

We present formal models of the performance of various asynchronous hardware systems. In particular, we examine the behaviour of the systems under deterministic computation cost, and non-deterministic or probabilistic cost. We present proofs of the limiting behaviour of pipelines with variable cost units. Studies of the effect of buffering variable units within constant cost pipelines. The effect of various flushing strategies on the performance of asynchronous pipelines under variable thread length, and preliminary models of the effects of data dependency of processing time on pipelines.
Keywords: Asynchronous Hardware, Performance Analysis, Process Algebra, Formal Methods, Concurrency.
CR categories: B.4.3, B.4.4, F.1.2

1 Introduction

Asynchronous design methods have become popular as a potential route to achieve low power/high performance [Fur93] devices. The analysis of the timing properties of these systems is complicated by two factors. Firstly, as a result of the synchrony the parallelism within the system must be carefully considered when analysing its performance. Secondly, as a result of the self-timing properties, it is possible for the execution time of a computation stage to vary, thus requiring that we take account of such variability.

One popular design style for asynchronous hardware is the use of Sutherland micropipelines [Sut89], breaking the circuit up into processing stages which communicate via a small protocol. In the first part of this paper we concentrate on a timing analysis of this design style. We aim to give a compositional account of the timing properties as this will permit us to address large scale problems with relative ease.

In the second part of the paper we shall examine the performance of asynchronous pipelines when the computation costs are variable. Throughout we shall make use of formal description methods to present the systems understudy, and the results obtained have been derived directly from these formal descriptions.

Throughout we make use of process algebraic notation [Mil90] to formally describe the systems under consideration. In particular we use notations that permit the description of real time properties [MT90, MT91, MT92, MT93] and probabilistic and priority properties [Tof90, Tof94].

2 Deterministic costs

Previous analyses of asynchronous pipelines [Wil90, SS93, Wil94] of the performance of asynchronous pipelines have been holistic approaches to the determination of the circuit performance. In this section we shall explicitly consider the parallelism within a Sutherland micropipeline [Sut89] stage and attempt to derive the performance of the system compositionally. That is to say, we shall derive the performance (time between request and acknowledgement) of each pipeline stage from the timing constraints of its environment. The advantage of this approach is that we shall be able to identify those stage which are starved or stalled. We shall not assume that all of the stages of the pipeline have the same computation time.

Since the performance of an asynchronous system can degrade if its environment makes unreasonable demands upon it [Wil94] we shall approach the calculation of performance from two perspectives. Initially we shall assume that the environment is always waiting for the pipeline and will always immediately launch another processing request on the termination signal of the previous one. Alternatively, there may be some delay between the acknowledgement and the next request, and this can actually improve pipeline performance [Wil94], in which case an alternative view of performance is necessary. We call these two views **forward** and **backward** respectively.

Our approach is to make use of timed extensions of Milner's [Mil80, Mil90] Calculus of Communicating Systems (CCS) called TCCS [MT90] to describe the behaviour of a micropipeline stage and its environment.

2.1 Introduction to TCCS

For a full description of the timed extension to Milner's CCS[Mil80, Mil90] see [MT90, MT91, MT92, MT93]. CCS is based on the notion of interleaving parallelism and handshake communications, with a being an *input* action and $\overline{a}$ an *output*, with τ representing a completed communication between an input and output pair.

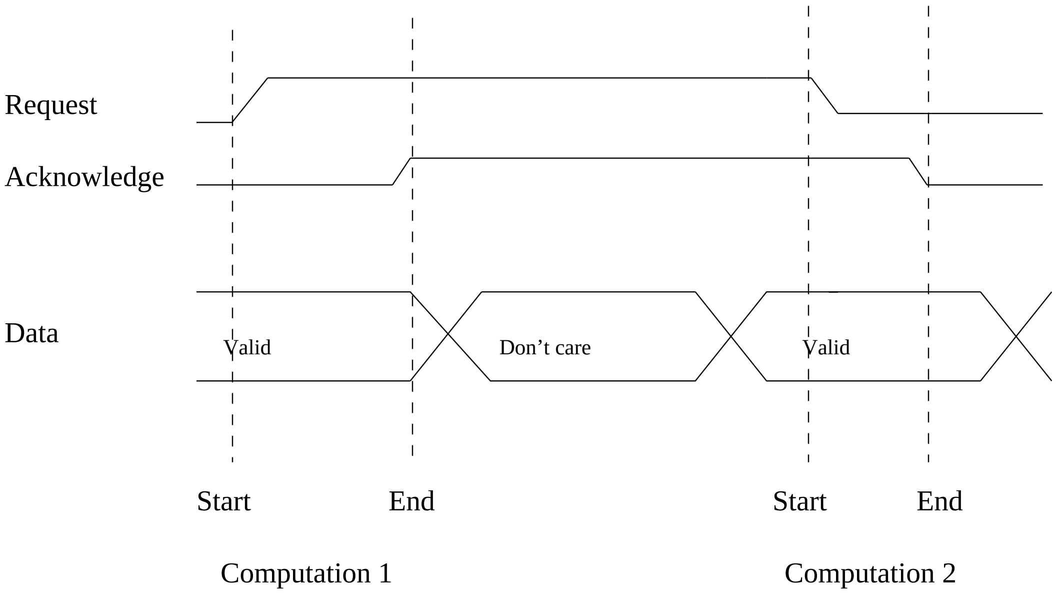

Figure 1: The transitions for the 2 phase Sutherland Asynchronous protocol. After the data becomes valid, the requester can indicate readiness by performing a transition on the request line. The data valid is held valid until an acknowledgement is received by the acknowledge line performing a transition.

In the sequel we use $\tau(a)$ to denote a τ action resulting from the combination of a and $\overline{a}$ to disambiguate its source; in fact the calculus makes no distinction between τ actions. For clarity we include a brief description of the calculus.

Insistent Action Prefix the process $a.P$ *must* perform the action a at this instant and evolve into the process P;

Delaying Action Prefix the process $\underline{a}.P$ will eventually perform the action a and evolve into the process P;

Time Prefix the process $(t)P$ will become the process P after t units of time have elapsed;

Choice $P + Q$ has the behaviours of either P or Q, if *both* can evolve in time then we retain their subsequent computation behaviour, if either has an insistent action only then it will be dropped if the other evolves in time;

Parallel Composition $P|Q$ behaves as the parallel composition of P and Q;

Restriction $P\backslash A$ performs only those actions not in the set A. This is the manner in which two components are forced to communicate on their common communication channels.

Renaming $P[f]$, renames the actions of P with the renaming function f, which has the following property $f(\tau) = \tau$ and $\overline{f(a)} = f(\overline{a})$.

Definition $X \stackrel{def}{=} P$ is the definition of a process variable X

2.2 Sutherland Pipeline Stages

In Figure 1 we present a timing diagram for the 2 phase Sutherland micropipeline protocol. From the diagram it is clear that the timing is dictacted by the control signals on the request an acknowledge lines. Hence, if we model the timing properties of these lines we can derive performance properties without having to explicitly make reference to the data in the system.

The following derived operator of TCCS describes a process that will reach the state P at time t in the future, but is willing to accept an action a within that time period. This commonly occurs when a system is reseting but must accept any further input in that period.

Definition 2.1 *To define the behaviour of a latch we need an auxiliary*[1] *temporal operator:*

$$[a,t]P \stackrel{def}{=} \sum_{s=0}^{s=t}(s)a.(t-s)P$$

Definition 2.2 *Need an auxiliary temporal operator:*

$$[a,t]P \stackrel{def}{=} \sum_{s=0}^{s=t}(s)a(t-s)P$$

We can model a Sutherland computational pipeline stage with the following TCCS processes. A computational pipeline stage consists of two components, a delay (the computation) and a latch (to put the result back).

We can define a time latch process as the following:

$$\begin{aligned} L &\stackrel{def}{=} \underline{rin}.EFL \\ EFL &\stackrel{def}{=} (L_f)FL \\ FL &\stackrel{def}{=} \overline{rout}.\overline{ain}.WL \\ WL &\stackrel{def}{=} \underline{rin}.\underline{aout}.(L_b)(L_f).FL + \underline{aout}.WLA \\ WLA &\stackrel{def}{=} WLA(L_b) \\ WLA(r) &\stackrel{def}{=} [rin,(r)](L_f)FL + (r)L \end{aligned}$$

The computational element can be modelled as a pure delay:

$$Dl \stackrel{def}{=} \underline{rin}.(D)\overline{out}.Dl$$

So a computational pipeline stage will be the following process:

$$Sth \stackrel{def}{=} (Dl|L[out/rin])\backslash\{out\}$$

The process can be pictured as in Figure 2.

To obey the environment rules [Sut89] for a two phase asynchronous system we cannot issue two rin actions unless we have received and ain action between them. We introduce the actions r and a to track the temporal behaviour of the complete system.

$$Env \stackrel{def}{=} r.\overline{rin}.\underline{ain}.a.Env$$

This form of environment drives the pipeline stage as quickly as possible. Finally we have to take account of the behaviour of the subsequent pileline stages in the system. From the point of view of our pipeline stage what it 'sees' is a delay between the issuing of a $rout$ action and the arrival of the subsequent $aout$ action. So we model the remainder of the pipeline as a simple delay:

$$BE \stackrel{def}{=} \underline{rout}.(E)\overline{aout}.BE$$

Define a restriction set:

$$R \stackrel{def}{=} \{rin, ain, rout, aout\}$$

The complete pipeline stage and its respective environments is thus the process:

[1] We take the process $(0)P$ to be identical to the process P.

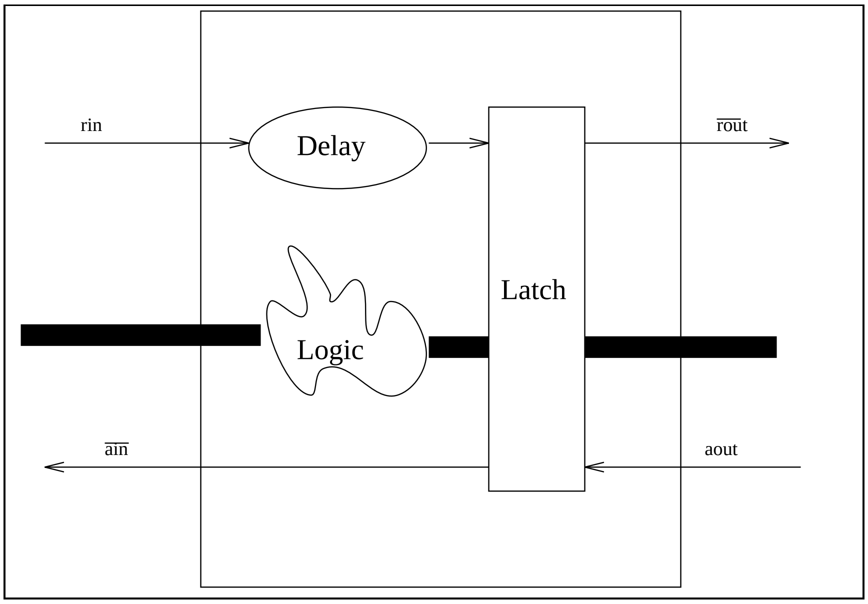

Figure 2: A Sutherland computational micropipline stage.

$$S \overset{def}{=} (Env|Sth|BE)\backslash R$$

Let us examine the evolution of the above process:

$$\begin{array}{lcl}
S & \xrightarrow{r} & (\overline{rin}.\underline{ain}.a.Env|Sth|BE)\backslash R \\
 & \xrightarrow{\tau(rin)} & (\underline{ain}.a.Env|((D)\overline{out}.Dl|L[out/rin])\backslash\{out\}|BE)\backslash R \\
 & \overset{D}{\leadsto} & (\underline{ain}.a.Env|(\overline{out}.Dl|L[out/rin])\backslash\{out\}|BE)\backslash R \\
 & \xrightarrow{\tau(out)} & (\underline{ain}.a.Env|((Dl|(L_f)FL[out/rin])\backslash\{out\}|BE)\backslash R \\
 & \overset{L_f}{\leadsto} & (\underline{ain}.a.Env|((D)\overline{out}.Dl|FL[out/rin])\backslash\{out\}|BE)\backslash R \\
 & \xrightarrow{\tau(rout)} & (\underline{ain}.a.Env|(Dl|\overline{ain}.WL[out/rin])\backslash\{out\}|(E)\overline{aout}.BE)\backslash R \\
 & \xrightarrow{\tau(ain)} & (a.Env|(Dl|WL[out/rin])\backslash\{out\}|(E)\overline{aout}.BE)\backslash R \\
 & \xrightarrow{a} & (Env|(Dl|WL[out/rin])\backslash\{out\}|(E)\overline{aout}.BE)\backslash R
\end{array}$$

So in some senses the system has completed a computation step at this stage, i.e., we have a complete r to a cycle. Note that the system has not returned to its original state. Naming the above state:

$$S_1 \overset{def}{=} (Env|(Dl|WL[out/rin])\backslash\{out\}|(E)BE)\backslash R$$

Examining the evolution of this process we find three different execution paths dependening upon the value of E.

If $E + L_b \leq D$ then we have the following evolution path:

$$
\begin{array}{lcl}
S & \xrightarrow{r} & (\overline{rin}.\underline{ain}.a.Env|(Dl|WL[out/rin])\backslash\{out\}|(E)\overline{aout}.BE)\backslash R \\
 & \xrightarrow{\tau(rin)} & (\underline{ain}.a.Env|((D)\overline{out}.Dl|WL[out/rin])\backslash\{out\}|(E)\overline{aout}.BE)\backslash R \\
 & \stackrel{E}{\leadsto} & (\underline{ain}.a.Env|((D-E)\overline{out}.Dl|WL[out/rin])\backslash\{out\}|\overline{aout}.BE)\backslash R \\
 & \xrightarrow{\tau(aout)} & (\underline{ain}.a.Env|((D-E)\overline{out}.Dl|WLA[out/rin])\backslash\{out\}|BE)\backslash R \\
 & \stackrel{L_b}{\leadsto} & (\underline{ain}.a.Env|((D-E-L_b)\overline{out}.Dl|L[out/rin])\backslash\{out\}|BE)\backslash R \\
 & \stackrel{D-E-L_b}{\leadsto} & (\underline{ain}.a.Env|(\overline{out}.Dl|L[out/rin])\backslash\{out\}|BE)\backslash R \\
 & \xrightarrow{\tau(out)} & (\underline{ain}.a.Env|((Dl|(L_f)FL[out/rin])\backslash\{out\}|BE)\backslash R \\
 & \stackrel{L_f}{\leadsto} & (\underline{ain}.a.Env|((D)\overline{out}.Dl|FL[out/rin])\backslash\{out\}|BE)\backslash R \\
 & \xrightarrow{\tau(rout)} & (\underline{ain}.a.Env|(Dl|\overline{ain}.WL[out/rin])\backslash\{out\}|(E)\overline{aout}.BE)\backslash R \\
 & \xrightarrow{\tau(ain)} & (a.Env|(Dl|WL[out/rin])\backslash\{out\}|(E)\overline{aout}.BE)\backslash R \\
 & \xrightarrow{a} & (Env|(Dl|WL[out/rin])\backslash\{out\}|(E)\overline{aout}.BE)\backslash R
\end{array}
$$

If $D \geq E > D - L_b$ then we obtain the following evolution path, let $D_e = D - E$:

$$
\begin{array}{lcl}
S & \xrightarrow{r} & (\overline{rin}.\underline{ain}.a.Env|(Dl|WL[out/rin])\backslash\{out\}|(E)\overline{aout}.BE)\backslash R \\
 & \xrightarrow{\tau(rin)} & (\underline{ain}.a.Env|((D)\overline{out}.Dl|WL[out/rin])\backslash\{out\}|(E)\overline{aout}.BE)\backslash R \\
 & \stackrel{E}{\leadsto} & (\underline{ain}.a.Env|((D-E)\overline{out}.Dl|WL[out/rin])\backslash\{out\}|\overline{aout}.BE)\backslash R \\
 & \xrightarrow{\tau(aout)} & (\underline{ain}.a.Env|((D-E)\overline{out}.Dl|WLA[out/rin])\backslash\{out\}|BE)\backslash R \\
 & \stackrel{D-E}{\leadsto} & (\underline{ain}.a.Env|(\overline{out}.Dl|([rin,(L_b-D_E))](L_f)FL+ \\
 & & (L_b-D_e))L[out/rin])\backslash\{out\}|BE)\backslash R \\
 & \xrightarrow{\tau(out)} & (\underline{ain}.a.Env|(Dl|(L_b-D_E)(L_f)FL[out/rin])\backslash\{out\}|BE)\backslash R \\
 & \stackrel{L_b-D_e+L_f}{\leadsto} & (\underline{ain}.a.Env|(Dl|FL[out/rin])\backslash\{out\}|BE)\backslash R \\
 & \xrightarrow{\tau(rout)} & (\underline{ain}.a.Env|(Dl|\overline{ain}.WL[out/rin])\backslash\{out\}|(E)\overline{aout}.BE)\backslash R \\
 & \xrightarrow{\tau(ain)} & (a.Env|(Dl|WL[out/rin])\backslash\{out\}|(E)\overline{aout}.BE)\backslash R \\
 & \xrightarrow{a} & (Env|(Dl|WL[out/rin])\backslash\{out\}|(E)\overline{aout}.BE)\backslash R
\end{array}
$$

Finally if $E > D$ then we obtain

$$
\begin{array}{lcl}
S & \xrightarrow{r} & (\overline{rin}.\underline{ain}.a.Env|(Dl|WL[out/rin])\backslash\{out\}|(E)\overline{aout}.BE)\backslash R \\
 & \xrightarrow{\tau(rin)} & (\underline{ain}.a.Env|((D)\overline{out}.Dl|WL[out/rin])\backslash\{out\}|(E)\overline{aout}.BE)\backslash R \\
 & \stackrel{D}{\leadsto} & (\underline{ain}.a.Env|(\overline{out}.Dl|WL[out/rin])\backslash\{out\}|(E-D)\overline{aout}.BE)\backslash R \\
 & \xrightarrow{\tau(out)} & (\underline{ain}.a.Env|(Dl|\underline{aout}.(L_b)(L_f).FL[out/rin])\backslash\{out\}|(E-D)\overline{aout}.BE)\backslash R \\
 & \stackrel{E-D}{\leadsto} & (\underline{ain}.a.Env|(Dl|\underline{aout}.(L_b)(L_f).FL[out/rin])\backslash\{out\}|\overline{aout}.BE)\backslash R \\
 & \xrightarrow{\tau(aout)} & (\underline{ain}.a.Env|(Dl|(L_b)(L_f).FL[out/rin])\backslash\{out\}|BE)\backslash R \\
 & \stackrel{L_b+L_f}{\leadsto} & (\underline{ain}.a.Env|(Dl|FL[out/rin])\backslash\{out\}|BE)\backslash R \\
 & \xrightarrow{\tau(rout)} & (\underline{ain}.a.Env|(Dl|\overline{ain}.WL[out/rin])\backslash\{out\}|(E)\overline{aout}.BE)\backslash R \\
 & \xrightarrow{\tau(ain)} & (a.Env|(Dl|WL[out/rin])\backslash\{out\}|(E)\overline{aout}.BE)\backslash R \\
 & \xrightarrow{a} & (Env|(Dl|WL[out/rin])\backslash\{out\}|(E)\overline{aout}.BE)\backslash R
\end{array}
$$

In summary we can make the following observations of the timing of our system, we take the latency as the time between the first r and a actions and the cycle time as that which the system stabilises in.

	$E + L_b \leq D$	$D - L_b < E \leq D$	$D < E$
Latency	$D + L_f$	$D + L_f$	$D + L_f$
Cycle time	$D + L_f$	$E + L_b + L_f$	$E + L_b + L_f$

We could have chosen to absorb the time of the delay directly into the latch in the following fashion:

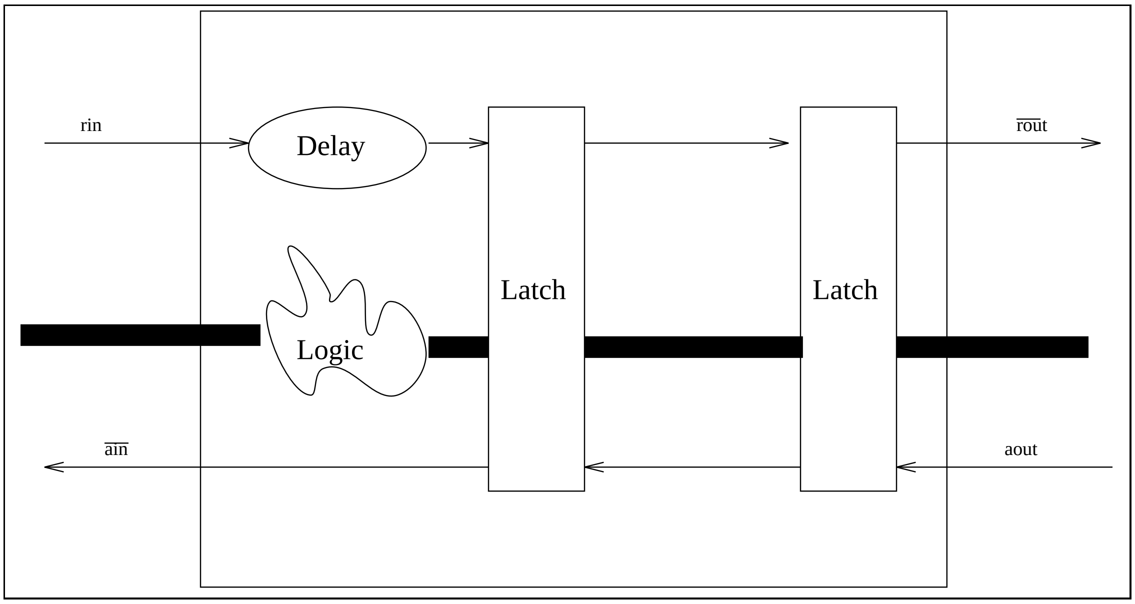

Figure 3: A buffered Sutherland micropipline stage

$$
\begin{aligned}
DL &\stackrel{def}{=} \underline{rin}.DEFL \\
DEFL &\stackrel{def}{=} (D + L_f).DFL \\
DFL &\stackrel{def}{=} \overline{rout}.\overline{ain}.WL \\
DWL &\stackrel{def}{=} \underline{rin}.\underline{aout}.(L_b)(D + L_f).DFL + \underline{aout}.DWLA \\
DWLA &\stackrel{def}{=} [rin, (L_b)](D + L_f)DFL + (L_b)DL
\end{aligned}
$$

but this gives us an **incorrect** behaviour, as the above latch cannot perform computation whilst waiting for an *aout* action whereas the system can. Essentially the computation circuitry behaves as an extra pipeline stage, thus permitting a far greater level of pipeline occupancy than would be suspected [Wil94] (provided that subsequent pipeline stages can execute rapidly enough).

2.2.1 Adding a latch

A possible method of improving performance when a 'fast' stage is followed by a slow one is to introduce some buffering in the form of an extra latch stage between the units. We can model this by inserting the extra latch between our pipeline stage and the subsequent environment. Hence we obtain the following process:

$$
\begin{aligned}
f_1 &\stackrel{def}{=} [cr/rout, ca/aout] \\
f_2 &\stackrel{def}{=} [cr/rin, ca/ain] \\
BPS &\stackrel{def}{=} (Dl|L[out/rin])\backslash\{out\}[f_1]|L[f_2])\backslash\{ca, cr\}
\end{aligned}
$$

which we can picture as Figure 3.

the complete system now becomes the process:

$$B \stackrel{def}{=} (Env|BPS|BE)\backslash R$$

We now consider the evolution of the above system, we assume that $L_f + 2L_b < D$ and $E \leq D + L_B$:

$$
\begin{array}{ll}
 & B \\
\xrightarrow{r} & (\overline{rin}.\underline{ain}.a.Env|(Dl|L[out/rin])\backslash\{out\}[f_1]|L[f_2])\backslash\{ca,cr\}|BE)\backslash R \\
\xrightarrow{\tau(rin)} & (\underline{ain}.a.Env|((D)\overline{out}.Dl|L[out/rin])\backslash\{out\}[f_1]|L[f_2])\backslash\{ca,cr\}|BE)\backslash R \\
\overset{D}{\rightsquigarrow} & (\underline{ain}.a.Env|(\overline{out}.Dl|L[out/rin])\backslash\{out\}[f_1]|L[f_2])\backslash\{ca,cr\}|BE)\backslash R \\
\xrightarrow{\tau(out)} & (\underline{ain}.a.Env|(Dl|(L_f)FL[out/rin])\backslash\{out\}[f_1]|L[f_2])\backslash\{ca,cr\}|BE)\backslash R \\
\overset{L_f}{\rightsquigarrow} & (\underline{ain}.a.Env|(Dl|FL[out/rin])\backslash\{out\}[f_1]|L[f_2])\backslash\{ca,cr\}|BE)\backslash R \\
\xrightarrow{\tau(c_1)} & (\underline{ain}.a.Env|(Dl|FL[out/rin])\backslash\{out\}[f_1]|(L_f)LF[f_2])\backslash\{ca,cr\}|BE)\backslash R \\
\xrightarrow{\tau(ain)} & (a.Env|(Dl|FL[out/rin])\backslash\{out\}[f_1]|(L_f)LF[f_2])\backslash\{ca,cr\}|BE)\backslash R \\
\xrightarrow{a} & (Env|(Dl|FL[out/rin])\backslash\{out\}[f_1]|(L_f)LF[f_2])\backslash\{ca,cr\}|BE)\backslash R \\
\xrightarrow{r} & (\overline{rin}.\underline{ain}.a.Env|(Dl|FL[out/rin])\backslash\{out\}|(L_f)FL[f_2])\backslash\{ca,cr\}|BE)\backslash R \\
\xrightarrow{\tau(rin)} & (\underline{ain}.a.Env|((D)\overline{out}.Dl|FL[out/rin])\backslash\{out\}[f_1]|(L_f)FL[f_2])\backslash\{ca,cr\}|BE)\backslash R \\
\overset{L_f}{\rightsquigarrow} & (\underline{ain}.a.Env|((D-L_f)\overline{out}.Dl|FL[out/rin])\backslash\{out\}[f_1]| \\
 & \overline{rout}.\overline{ain}.WL[f_2])\backslash\{ca,cr\}|BE)\backslash R
\end{array}
$$

$$
\begin{array}{ll}
\xrightarrow{\tau(rout)} & (\underline{ain}.a.Env|((D-L_f)\overline{out}.Dl|FL[out/rin])\backslash\{out\}[f_1]| \\
 & \overline{ain}.WL[f_2])\backslash\{ca,cr\}|(E)\overline{aout}.BE)\backslash R \\
\xrightarrow{\tau(ca)} & (\underline{ain}.a.Env|((D-L_f)\overline{out}.Dl|WLA[out/rin])\backslash\{out\}[f_1]| \\
 & WL[f_2])\backslash\{ca,cr\}|(E)\overline{aout}.BE)\backslash R \\
\overset{L_b}{\rightsquigarrow} & (\underline{ain}.a.Env|((D-L_f-L_b)\overline{out}.Dl|WLA[out/rin])\backslash\{out\}[f_1]| \\
 & WL[f_2])\backslash\{ca,cr\}|(E-L_b)\overline{aout}.BE)\backslash R \\
\overset{L_b}{\rightsquigarrow} & (\underline{ain}.a.Env|((D-L_f-2L_b)\overline{out}.Dl|L[out/rin])\backslash\{out\}[f_1]| \\
 & WL[f_2])\backslash\{ca,cr\}|(E-L_b)\overline{aout}.BE)\backslash R \\
\overset{D-2L_b-L_f}{\rightsquigarrow} & (\underline{ain}.a.Env|(\overline{out}.Dl|L[out/rin])\backslash\{out\}[f_1]|WL[f_2])\backslash\{ca,cr\}|(E-D+L_f-L_b)\overline{aout}.BE)\backslash R \\
\xrightarrow{\tau(out)} & (\underline{ain}.a.Env|(Dl|(L_f)FL[out/rin])\backslash\{out\}[f_1]|WL[f_2])\backslash\{ca,cr\}|(E-D+L_f-L_b)\overline{aout}.BE)\backslash R \\
\overset{(E-D+L_f-L_b)}{\rightsquigarrow} & (\underline{ain}.a.Env|(Dl|(D+L_b-E)FL[out/rin])\backslash\{out\}[f_1]|WL[f_2])\backslash\{ca,cr\}|\overline{aout}.BE)\backslash R \\
\xrightarrow{\tau(aout)} & (\underline{ain}.a.Env|(Dl|(D+L_b-E)FL[out/rin])\backslash\{out\}[f_1]|WLA[f_2])\backslash\{ca,cr\}|BE)\backslash R \\
\overset{D+L_b-E}{\rightsquigarrow} & (\underline{ain}.a.Env|(Dl|FL[out/rin])\backslash\{out\}[f_1]|WLA(E-D)[f_2])\backslash\{ca,cr\}|BE)\backslash R \\
\xrightarrow{\tau(cr)} & (\underline{ain}.a.Env|(Dl|\overline{ain}.WL[out/rin])\backslash\{out\}[f_1]|(E-D)(L_f)FL[f_2])\backslash\{ca,cr\}|BE)\backslash R \\
\xrightarrow{\tau(ain)} & (a.Env|(Dl|WL[out/rin])\backslash\{out\}[f_1]|(E-D)(L_f)FL[f_2])\backslash\{ca,cr\}|BE)\backslash R \\
\xrightarrow{a} & (Env|(Dl|WL[out/rin])\backslash\{out\}[f_1]|(E-D)(L_f)FL[f_2])\backslash\{ca,cr\}|BE)\backslash R
\end{array}
$$

Notice that the cycle time in both cases above is $D + L_f$.

Define $B_1 \stackrel{def}{=} (Env|(Dl|WL[out/rin])\backslash\{out\}[f_1]|(E-D)(L_f)FL[f_2])\backslash\{ca,cr\}|BE)\backslash R$, then examining its evolution:

$$
\begin{array}{cl}
B_1 & \\
\xrightarrow{r} & (\overline{rin}.\underline{ain}.a.Env|(Dl|WL[out/rin])\backslash\{out\}[f_1]|(E-D)(L_f)FL[f_2])\backslash\{ca,cr\}|BE)\backslash R \\
\xrightarrow{\tau(rin)} & (\underline{ain}.a.Env|((D)\overline{out}.Dl|WL[out/rin])\backslash\{out\}[f_1]|(E-D)(L_f)FL[f_2])\backslash\{ca,cr\}|BE)\backslash R \\
\overset{E-D+L_f}{\leadsto} & (\underline{ain}.a.Env|((2D-E-L_f)\overline{out}.Dl|WL[out/rin])\backslash\{out\}[f_1]|FL[f_2])\backslash\{ca,cr\}|BE)\backslash R \\
\xrightarrow{\tau(rout)} & (\underline{ain}.a.Env|((2D-E-L_f)\overline{out}.Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
 & \overline{ain}.WL[f_2])\backslash\{ca,cr\}|(E)\overline{aout}.BE)\backslash R \\
\xrightarrow{\tau(ca)} & (\underline{ain}.a.Env|((2D-E-L_f)\overline{out}.Dl|WLA(L_b)[out/rin])\backslash\{out\}[f_1]| \\
 & WL[f_2])\backslash\{ca,cr\}|(E)\overline{aout}.BE)\backslash R \\
\overset{L_b}{\leadsto} & (\underline{ain}.a.Env|((2D-E-L_f-L_b)\overline{out}.Dl|L[out/rin])\backslash\{out\}[f_1]| \\
 & WL[f_2])\backslash\{ca,cr\}|(E-L_b)\overline{aout}.BE)\backslash R \\
\overset{2D-E-L_f-L_b}{\leadsto} & (\underline{ain}.a.Env|(\overline{out}.Dl|L[out/rin])\backslash\{out\}[f_1]|WL[f_2])\backslash\{ca,cr\}|(2E-2D+L_f)\overline{aout}.BE)\backslash R \\
\xrightarrow{\tau(out)} & (\underline{ain}.a.Env|(Dl|(L_f)FL[out/rin])\backslash\{out\}[f_1]|WL[f_2])\backslash\{ca,cr\}|(2E-2D+L_f)\overline{aout}.BE)\backslash R \\
\overset{L_f}{\leadsto} & (\underline{ain}.a.Env|(Dl|FL[out/rin])\backslash\{out\}[f_1]|WL[f_2])\backslash\{ca,cr\}|(2E-2D)\overline{aout}.BE)\backslash R \\
\xrightarrow{\tau(cr)} & (\underline{ain}.a.Env|(Dl|\overline{ain}.WL[out/rin])\backslash\{out\}[f_1]| \\
 & \underline{aout}.(L_b)(L_f)L[f_2])\backslash\{ca,cr\}|(2E-2D)\overline{aout}.BE)\backslash R \\
\xrightarrow{\tau(ain)} & (a.Env|(Dl|WL[out/rin])\backslash\{out\}[f_1]|\underline{aout}.(L_b)(L_f)L[f_2])\backslash\{ca,cr\}|(2E-2D)\overline{aout}.BE)\backslash R \\
\xrightarrow{a} & (Env|(Dl|WL[out/rin])\backslash\{out\}[f_1]|\underline{aout}.(L_b)(L_f)L[f_2])\backslash\{ca,cr\}|(2E-2D)\overline{aout}.BE)\backslash R
\end{array}
$$

Again the cycle time is $D+L_f$. Call the above process B_2 and continuing:

$$
\begin{array}{cl}
B_2 & \\
\xrightarrow{r} & (\overline{rin}.\underline{ain}.a.Env|(Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
 & \underline{aout}.(L_b+L_f)FL[f_2])\backslash\{ca,cr\}|(2E-2D)\overline{aout}.BE)\backslash R \\
\xrightarrow{\tau(rin)} & (\underline{ain}.a.Env|((D)\overline{out}.Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
 & \underline{aout}.(L_b)(L_f)FL[f_2])\backslash\{ca,cr\}|(2E-2D)\overline{aout}.BE)\backslash R \\
\overset{2(E-D)}{\leadsto} & (\underline{ain}.a.Env|((D-2(E-D))\overline{out}.Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
 & \underline{aout}.(L_b+L_f)FL[f_2])\backslash\{ca,cr\}|\overline{aout}.BE)\backslash R \\
\xrightarrow{\tau(aout)} & (\underline{ain}.a.Env|((D-2(E-D))\overline{out}.Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
 & (L_b+L_f)FL[f_2])\backslash\{ca,cr\}|BE)\backslash R \\
\overset{L_b+L_f}{\leadsto} & (\underline{ain}.a.Env|((D-2(E-D)-L_b-L_f)\overline{out}.Dl| \\
WL[out/rin])\backslash\{out\}[f_1]| & \\
 & FL[f_2])\backslash\{ca,cr\}|BE)\backslash R \\
\xrightarrow{\tau(rout)} & (\underline{ain}.a.Env|((D-2(E-D)-L_b-L_f)\overline{out}.Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
 & \overline{ain}.WL[f_2])\backslash\{ca,cr\}|(E)\overline{aout}.BE)\backslash R \\
\xrightarrow{\tau(ca)} & (\underline{ain}.a.Env|((D-2(E-D)-L_b-L_f)\overline{out}.Dl|WLA(L_B)[out/rin])\backslash\{out\}[f_1]| \\
 & WL[f_2])\backslash\{ca,cr\}|(E)\overline{aout}.BE)\backslash R \\
\overset{L_b}{\leadsto} & (\underline{ain}.a.Env|((D-2(E-D)-2L_b-L_f)\overline{out}.Dl|L[out/rin])\backslash\{out\}[f_1]| \\
 & WL[f_2])\backslash\{ca,cr\}|(E-L_b)\overline{aout}.BE)\backslash R \\
\overset{(D-2(E-D)-2L_b-L_f)}{\leadsto} & (\underline{ain}.a.Env|(\overline{out}.Dl|L[out/rin])\backslash\{out\}[f_1]| \\
 & WL[f_2])\backslash\{ca,cr\}|(3(E-D)+L_b+L_f)\overline{aout}.BE)\backslash R \\
\xrightarrow{\tau(out)} & (\underline{ain}.a.Env|(Dl|((L_f)FL[out/rin])\backslash\{out\}[f_1]| \\
 & WL[f_2])\backslash\{ca,cr\}|(3(E-D)+L_b+L_f)\overline{aout}.BE)\backslash R \\
\overset{L_f}{\leadsto} & (\underline{ain}.a.Env|(Dl|(FL[out/rin])\backslash\{out\}[f_1]| \\
 & WL[f_2])\backslash\{ca,cr\}|(3(E-D)+L_b)\overline{aout}.BE)\backslash R \\
\xrightarrow{\tau(cr)} & (\underline{ain}.a.Env|(Dl|(\overline{ain}.WL[out/rin])\backslash\{out\}[f_1]| \\
 & \underline{aout}.(L_b+L_f)FL[f_2])\backslash\{ca,cr\}|(3(E-D)+L_b)\overline{aout}.BE)\backslash R \\
\xrightarrow{\tau(ain)} & (a.Env|(Dl|(WL[out/rin])\backslash\{out\}[f_1]| \\
 & \underline{aout}.(L_b+L_f)FL[f_2])\backslash\{ca,cr\}|(3(E-D)+L_b)\overline{aout}.BE)\backslash R \\
\xrightarrow{a} & (Env|(Dl|(WL[out/rin])\backslash\{out\}[f_1]| \\
 & \underline{aout}.(L_b+L_f)FL[f_2])\backslash\{ca,cr\}|(3(E-D)+L_b)\overline{aout}.BE)\backslash R
\end{array}
$$

The cycle time is still $D+L_f$. We can see that in the above process we are drifting towards the point at which the backend process cannot have succesfully executed before the delay on the pipeline stage has been exhausted. Let us consider what happens when $3(E-D)+L_b$ is greater than $D+L_f$. Naming the above state B_3:

$$
\begin{array}{cl}
B_3 & \\
\xrightarrow{r} & (\overline{rin}.\underline{ain}.a.Env|(Dl|WL[out/rin])\setminus\{out\}[f_1]| \\
& \underline{aout}.(L_b+L_f)FL[f_2])\setminus\{ca,cr\}|(3(E-D)+2L_b)\overline{aout}.BE)\setminus R \\
\xrightarrow{\tau(rin)} & (\underline{ain}.a.Env|((D)\overline{out}.Dl|WL[out/rin])\setminus\{out\}[f_1]| \\
& \underline{aout}.(L_b+L_f)FL[f_2])\setminus\{ca,cr\}|(3(E-D)+2L_b)\overline{aout}.BE)\setminus R \\
\overset{D}{\leadsto} & (\underline{ain}.a.Env|(\overline{out}.Dl|WL[out/rin])\setminus\{out\}[f_1]| \\
& \underline{aout}.(L_b+L_f)FL[f_2])\setminus\{ca,cr\}|(3(E-D)+2L_b-D)\overline{aout}.BE)\setminus R \\
\xrightarrow{\tau(cr)} & (\underline{ain}.a.Env|(Dl|\underline{aout}.(L_b+L_f)FL[out/rin])\setminus\{out\}[f_1]| \\
& \underline{aout}.(L_b+L_f)FL[f_2])\setminus\{ca,cr\}|(3(E-D)+2L_b-D)\overline{aout}.BE)\setminus R \\
\overset{(3(E-D)+2L_b-D)}{\leadsto} & (\underline{ain}.a.Env|(Dl|\underline{aout}.(L_b+L_f)FL[out/rin])\setminus\{out\}[f_1]| \\
& \underline{aout}.(L_b+L_f)FL[f_2])\setminus\{ca,cr\}|\overline{aout}.BE)\setminus R \\
\xrightarrow{\tau(aout)} & (\underline{ain}.a.Env|(Dl|\underline{aout}.(L_b+L_f)FL[out/rin])\setminus\{out\}[f_1]| \\
& (L_b+L_f)FL[f_2])\setminus\{ca,cr\}|BE)\setminus R \\
\overset{L_b+L_f}{\leadsto} & (\underline{ain}.a.Env|(Dl|\underline{aout}.(L_b+L_f)FL[out/rin])\setminus\{out\}[f_1]| \\
& FL[f_2])\setminus\{ca,cr\}|BE)\setminus R \\
\xrightarrow{\tau(rout)} & (\underline{ain}.a.Env|(Dl|\underline{aout}.(L_b+L_f)FL[out/rin])\setminus\{out\}[f_1]| \\
& \overline{ain}.WL[f_2])\setminus\{ca,cr\}|(E)\overline{aout}.BE)\setminus R \\
\xrightarrow{\tau(ca)} & (\underline{ain}.a.Env|(Dl|(L_b+L_f)FL[out/rin])\setminus\{out\}[f_1]| \\
& WL[f_2])\setminus\{ca,cr\}|(E)\overline{aout}.BE)\setminus R \\
\overset{L_b+L_f}{\leadsto} & (\underline{ain}.a.Env|(Dl|FL[out/rin])\setminus\{out\}[f_1]| \\
& WL[f_2])\setminus\{ca,cr\}|(E-L_b-L_f)\overline{aout}.BE)\setminus R \\
\xrightarrow{\tau(cr)} & (\underline{ain}.a.Env|(Dl|\overline{ain}.WL[out/rin])\setminus\{out\}[f_1]| \\
& \overline{aout}.(L_b+L_f)FL[f_2])\setminus\{ca,cr\}|(E-L_b-L_f)\overline{aout}.BE)\setminus R \\
\xrightarrow{\tau(ain)} & (a.Env|(Dl|WL[out/rin])\setminus\{out\}[f_1]| \\
& \overline{aout}.(L_b+L_f)FL[f_2])\setminus\{ca,cr\}|(E-L_b-L_f)\overline{aout}.BE)\setminus R \\
\xrightarrow{a} & (Env|(Dl|WL[out/rin])\setminus\{out\}[f_1]| \\
& \overline{aout}.(L_b+L_f)FL[f_2])\setminus\{ca,cr\}|(E-L_b-L_f)\overline{aout}.BE)\setminus R
\end{array}
$$

In this case the cycle time is $3(E-D)+4L_b+2L_f$ and since $3(E-D)+2L_b > D$ this is $> D+2L_b+2L_f$. Later we shall see that if $E > D+L_b$ this will give a stable cycle, but evidently this can discharge the E before the D so we continue to examine its evolution:

$$
\begin{aligned}
& (Env|(Dl|WL[out/rin])\backslash\{out\}[f_1]|\overline{aout}.(L_b+L_f)FL[f_2])\backslash\{ca,cr\}|(E-L_b-L_f)\overline{aout}.BE)\backslash R \\
\xrightarrow{r}\ & (\overline{rin}.\underline{ain}.a.Env|(Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
& \overline{aout}.(L_b+L_f)FL[f_2])\backslash\{ca,cr\}|(E-L_b-L_f)\overline{aout}.BE)\backslash R \\
\xrightarrow{\tau(rin)}\ & (\underline{ain}.a.Env|((D)\overline{out}.Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
& \overline{aout}.(L_b+L_f)FL[f_2])\backslash\{ca,cr\}|(E-L_b-L_f)\overline{aout}.BE)\backslash R \\
\overset{E-L_b-L_f}{\rightsquigarrow}\ & (\underline{ain}.a.Env|((D-E+L_b+L_f)\overline{out}.Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
& \overline{aout}.(L_b+L_f)FL[f_2])\backslash\{ca,cr\}|\overline{aout}.BE)\backslash R \\
\xrightarrow{\tau(aout)}\ & (\underline{ain}.a.Env|((D-E+L_b+L_f)\overline{out}.Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
& (L_b+L_f)FL[f_2])\backslash\{ca,cr\}|BE)\backslash R \\
\overset{D-E+L_b+L_f}{\rightsquigarrow}\ & (\underline{ain}.a.Env|(\overline{out}.Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
& (E-D)FL[f_2])\backslash\{ca,cr\}|BE)\backslash R \\
\xrightarrow{\tau(out)}\ & (\underline{ain}.a.Env|(\overline{out}.Dl|\underline{aout}.(L_b+L_f)FL[out/rin])\backslash\{out\}[f_1]| \\
& (E-D)FL[f_2])\backslash\{ca,cr\}|BE)\backslash R \\
\overset{E-D}{\rightsquigarrow}\ & (\underline{ain}.a.Env|(\overline{out}.Dl|\underline{aout}.(L_b+L_f)FL[out/rin])\backslash\{out\}[f_1]| \\
& FL[f_2])\backslash\{ca,cr\}|BE)\backslash R \\
\xrightarrow{\tau(rout)}\ & (\underline{ain}.a.Env|(\overline{out}.Dl|\underline{aout}.(L_b+L_f)FL[out/rin])\backslash\{out\}[f_1]| \\
& \overline{ain}.WL[f_2])\backslash\{ca,cr\}|(E)\overline{aout}.BE)\backslash R \\
\xrightarrow{\tau(ca)}\ & (\underline{ain}.a.Env|(\overline{out}.Dl|(L_b+L_f)FL[out/rin])\backslash\{out\}[f_1]| \\
& WL[f_2])\backslash\{ca,cr\}|(E)\overline{aout}.BE)\backslash R \\
\overset{L_b+L_f}{\rightsquigarrow}\ & (\underline{ain}.a.Env|(\overline{out}.Dl|FL[out/rin])\backslash\{out\}[f_1]| \\
& WL[f_2])\backslash\{ca,cr\}|(E-L_b+L_f)\overline{aout}.BE)\backslash R \\
\xrightarrow{\tau(cr)}\ & (\underline{ain}.a.Env|(\overline{out}.Dl|\overline{ain}.WL[out/rin])\backslash\{out\}[f_1]| \\
& \underline{aout}.(L_b+L_f)FL[f_2])\backslash\{ca,cr\}|(E-L_b+L_f)\overline{aout}.BE)\backslash R \\
\xrightarrow{\tau(ain)}\ & (a.Env|(\overline{out}.Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
& \underline{aout}.(L_b+L_f)FL[f_2])\backslash\{ca,cr\}|(E-L_b+L_f)\overline{aout}.BE)\backslash R \\
\xrightarrow{a}\ & (Env|(\overline{out}.Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
& \underline{aout}.(L_b+L_f)FL[f_2])\backslash\{ca,cr\}|(E-L_b+L_f)\overline{aout}.BE)\backslash R
\end{aligned}
$$

So we have identified a stable cycle with cycle time $E+L_f+L_b$. In order that we may undestand how the cycles approach stability let us examine the following state further under the assumption that $D > 3(E-D)+L_b$:

$$
\begin{array}{ll}
 & (Env|(Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
 & \underline{aout}.(L_b+L_f)FL[f_2])\backslash\{ca,cr\}|(3(E-D)+L_b)\overline{aout}.BE)\backslash R \\
\xrightarrow{r} & (\overline{rin}.\underline{ain}.a.Env|(Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
 & \underline{aout}.(L_b+L_f)FL[f_2])\backslash\{ca,cr\}|(3(E-D)+L_b)\overline{aout}.BE)\backslash R \\
\xrightarrow{\tau(rin)} & (\underline{ain}.a.Env|((D)\overline{out}.Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
 & \underline{aout}.(L_b+L_f)FL[f_2])\backslash\{ca,cr\}|(3(E-D)+L_b)\overline{aout}.BE)\backslash R \\
\overset{(3(E-D)+L_b)}{\rightsquigarrow} & (\underline{ain}.a.Env|((D-(3(E-D)-L_b))\overline{out}.Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
 & \underline{aout}.(L_b+L_f)FL[f_2])\backslash\{ca,cr\}|\overline{aout}.BE)\backslash R \\
\xrightarrow{\tau(aout)} & (\underline{ain}.a.Env|((D-(3(E-D)-L_b))\overline{out}.Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
 & (L_b+L_f)FL[f_2])\backslash\{ca,cr\}|BE)\backslash R \\
\overset{L_b+L_f}{\rightsquigarrow} & (\underline{ain}.a.Env|((D-(3(E-D)-2L_b-L_f))\overline{out}.Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
 & FL[f_2])\backslash\{ca,cr\}|BE)\backslash R \\
\xrightarrow{\tau(rout)} & (\underline{ain}.a.Env|((D-(3(E-D)-2L_b-L_f))\overline{out}.Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
 & \overline{ain}.WL[f_2])\backslash\{ca,cr\}|(E)\overline{aout}.BE)\backslash R \\
\xrightarrow{\tau(ca)} & (\underline{ain}.a.Env|((D-(3(E-D)-2L_b-L_f))\overline{out}.Dl|WLA(L_b)[out/rin])\backslash\{out\}[f_1]| \\
 & WL[f_2])\backslash\{ca,cr\}|(E)\overline{aout}.BE)\backslash R \\
\overset{L_b}{\rightsquigarrow} & (\underline{ain}.a.Env|((D-(3(E-D)-3L_b-L_f))\overline{out}.Dl|L[out/rin])\backslash\{out\}[f_1]| \\
 & WL[f_2])\backslash\{ca,cr\}|(E-L_b)\overline{aout}.BE)\backslash R \\
\overset{D-(3(E-D)-3L_b-L_f}{\rightsquigarrow} & (\underline{ain}.a.Env|(\overline{out}.Dl|L[out/rin])\backslash\{out\}[f_1]| \\
 & WL[f_2])\backslash\{ca,cr\}|(4(E-D)+2L_b+L_f)\overline{aout}.BE)\backslash R \\
\xrightarrow{\tau(out)} & (\underline{ain}.a.Env|(Dl|(L_f)FL[out/rin])\backslash\{out\}[f_1]| \\
 & WL[f_2])\backslash\{ca,cr\}|(4(E-D)+2L_b+L_f)\overline{aout}.BE)\backslash R \\
\overset{L_f}{\rightsquigarrow} & (\underline{ain}.a.Env|(Dl|FL[out/rin])\backslash\{out\}[f_1]| \\
 & WL[f_2])\backslash\{ca,cr\}|(4(E-D)+2L_b)\overline{aout}.BE)\backslash R \\
\xrightarrow{\tau(cr)} & (\underline{ain}.a.Env|(Dl|\overline{ain}.WL[out/rin])\backslash\{out\}[f_1]| \\
 & \underline{aout}.(L_b+L_f)FL[f_2])\backslash\{ca,cr\}|(4(E-D)+2L_b)\overline{aout}.BE)\backslash R \\
\xrightarrow{\tau(ain)} & (a.Env|(Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
 & \underline{aout}.(L_b+L_f)FL[f_2])\backslash\{ca,cr\}|(4(E-D)+2L_b)\overline{aout}.BE)\backslash R \\
\xrightarrow{a} & (Env|(Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
 & \underline{aout}.(L_b+L_f)FL[f_2])\backslash\{ca,cr\}|(4(E-D)+2L_b)\overline{aout}.BE)\backslash R
\end{array}
$$

From the above let us consider a generalised state $GS \stackrel{def}{=} (Env|(Dl|WL[out/rin])\backslash\{out\}[f_1]|\underline{aout}.(L_b+L_f)FL[f_2])\backslash\{ca,cr\}|(G)\overline{aout}.$

.

There a several possible evolution paths dependening on the relative values of D and G. If $G > D$ then we get the following path:

$$
\begin{array}{ll}
GS & \\
\xrightarrow{r} & (\overline{rin}.\underline{ain}.a.Env|(Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
 & \underline{aout}.(L_b+L_f)FL[f_2])\backslash\{ca,cr\}|(G)\overline{aout}.BE)\backslash R \\
\xrightarrow{\tau(rin)} & (\underline{ain}.a.Env|((D)\overline{out}.Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
 & \underline{aout}.(L_b+L_f)FL[f_2])\backslash\{ca,cr\}|(G)\overline{aout}.BE)\backslash R \\
\overset{D}{\rightsquigarrow} & (\underline{ain}.a.Env|(\overline{out}.Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
 & \underline{aout}.(L_b+L_f)FL[f_2])\backslash\{ca,cr\}|(G-D)\overline{aout}.BE)\backslash R \\
\xrightarrow{\tau(cr)} & (\underline{ain}.a.Env|(Dl|\underline{aout}.(L_b+L_f)FL[out/rin])\backslash\{out\}[f_1]| \\
 & \underline{aout}.(L_b+L_f)FL[f_2])\backslash\{ca,cr\}|(G-D)\overline{aout}.BE)\backslash R \\
\overset{G-D}{\rightsquigarrow} & (\underline{ain}.a.Env|(Dl|\underline{aout}.(L_b+L_f)FL[out/rin])\backslash\{out\}[f_1]| \\
 & \underline{aout}.(L_b+L_f)FL[f_2])\backslash\{ca,cr\}|\overline{aout}.BE)\backslash R \\
\xrightarrow{\tau(aout)} & (\underline{ain}.a.Env|(Dl|\underline{aout}.(L_b+L_f)FL[out/rin])\backslash\{out\}[f_1]| \\
 & (L_b+L_f)FL[f_2])\backslash\{ca,cr\}|BE)\backslash R \\
\overset{L_b+L_f}{\rightsquigarrow} & (\underline{ain}.a.Env|(Dl|\underline{aout}.(L_b+L_f)FL[out/rin])\backslash\{out\}[f_1]|FL[f_2])\backslash\{ca,cr\}|BE)\backslash R \\
\xrightarrow{\tau(rout)} & (\underline{ain}.a.Env|(Dl|\underline{aout}.(L_b+L_f)FL[out/rin])\backslash\{out\}[f_1]|\overline{ain}.WL[f_2])\backslash\{ca,cr\}|(E)\overline{aout}.BE)\backslash R \\
\xrightarrow{\tau(ca)} & (\underline{ain}.a.Env|(Dl|(L_b+L_f)FL[out/rin])\backslash\{out\}[f_1]| \\
 & WL[f_2])\backslash\{ca,cr\}|(E)\overline{aout}.BE)\backslash R \\
\overset{L_b+L_f}{\rightsquigarrow} & (\underline{ain}.a.Env|(Dl|FL[out/rin])\backslash\{out\}[f_1]| \\
 & WL[f_2])\backslash\{ca,cr\}|(E-L_b-L_f)\overline{aout}.BE)\backslash R \\
\xrightarrow{\tau(cr)} & (\underline{ain}.a.Env|(Dl|\overline{ain}.WL[out/rin])\backslash\{out\}[f_1]| \\
 & WL[f_2])\backslash\{ca,cr\}|(E-L_b-L_f)\overline{aout}.BE)\backslash R \\
\xrightarrow{\tau(ain)} & (a.Env|(Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
 & WL[f_2])\backslash\{ca,cr\}|(E-L_b-L_f)\overline{aout}.BE)\backslash R \\
\xrightarrow{a} & (Env|(Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
 & WL[f_2])\backslash\{ca,cr\}|(E-L_b-L_f)\overline{aout}.BE)\backslash R
\end{array}
$$

A cycle time of $G+2(L_b+L_f)$. The above state we have already seen gives us cycles of time $E+L_b+L_f$.

Now consider the case when $G < D$.

$$
\begin{array}{ll}
GS & \\
\xrightarrow{r} & (\overline{rin}.\underline{ain}.a.Env|(Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
 & \underline{aout}.(L_b+L_f)FL[f_2])\backslash\{ca,cr\}|(G)\overline{aout}.BE)\backslash R \\
\xrightarrow{\tau(rin)} & (\underline{ain}.a.Env|((D)\overline{out}.Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
 & \underline{aout}.(L_b+L_f)FL[f_2])\backslash\{ca,cr\}|(G)\overline{aout}.BE)\backslash R \\
\overset{G}{\rightsquigarrow} & (\underline{ain}.a.Env|((D-G)\overline{out}.Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
 & \underline{aout}.(L_b+L_f)FL[f_2])\backslash\{ca,cr\}|\overline{aout}.BE)\backslash R \\
\xrightarrow{\tau(aout)} & (\underline{ain}.a.Env|((D-G)\overline{out}.Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
 & (L_b+L_f)FL[f_2])\backslash\{ca,cr\}|BE)\backslash R
\end{array}
$$

At this point we have two possiblities either $D-G < L_f+L_b$ or not. Taking the first possiblity:

$$
\begin{aligned}
& (\underline{ain}.a.Env|((D-G)\overline{out}.Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
& (L_b+L_f)FL[f_2])\backslash\{ca,cr\}|BE)\backslash R \\
\overset{D-G}{\rightsquigarrow}\; & (\underline{ain}.a.Env|(\overline{out}.Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
& (L_b+L_f+G-D)FL[f_2])\backslash\{ca,cr\}|BE)\backslash R \\
\overset{\tau(out)}{\rightarrow}\; & (\underline{ain}.a.Env|(Dl|\underline{aout}.(L_b+L_f)FL[out/rin])\backslash\{out\}[f_1]| \\
& (L_b+L_f+G-D)FL[f_2])\backslash\{ca,cr\}|BE)\backslash R \\
\overset{L_b+L_f+G-D}{\rightsquigarrow}\; & (\underline{ain}.a.Env|(Dl|\underline{aout}.(L_b+L_f)FL[out/rin])\backslash\{out\}[f_1]| \\
& FL[f_2])\backslash\{ca,cr\}|BE)\backslash R \\
\overset{\tau(rout)}{\rightarrow}\; & (\underline{ain}.a.Env|(Dl|\underline{aout}.(L_b+L_f)FL[out/rin])\backslash\{out\}[f_1]| \\
& \overline{ain}.WL[f_2])\backslash\{ca,cr\}|(E)\overline{aout}.BE)\backslash R \\
\overset{\tau(ca)}{\rightarrow}\; & (\underline{ain}.a.Env|(Dl|(L_b+L_f)FL[out/rin])\backslash\{out\}[f_1]| \\
& WL[f_2])\backslash\{ca,cr\}|(E)\overline{aout}.BE)\backslash R \\
\overset{L_b+L_f}{\rightsquigarrow}\; & (\underline{ain}.a.Env|(Dl|FL[out/rin])\backslash\{out\}[f_1]| \\
& WL[f_2])\backslash\{ca,cr\}|(E-L_b+L_f)\overline{aout}.BE)\backslash R \\
\overset{\tau(cr)}{\rightarrow}\; & (\underline{ain}.a.Env|(Dl|\overline{ain}.WL[out/rin])\backslash\{out\}[f_1]| \\
& \underline{aout}.(L_b+L_f)FL[f_2])\backslash\{ca,cr\}|(E-L_b+L_f)\overline{aout}.BE)\backslash R \\
\overset{\tau(ain)}{\rightarrow}\; & (a.Env|(Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
& \underline{aout}.(L_b+L_f)FL[f_2])\backslash\{ca,cr\}|(E-L_b+L_f)\overline{aout}.BE)\backslash R \\
\overset{a}{\rightarrow}\; & (Env|(Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
& \underline{aout}.(L_b+L_f)FL[f_2])\backslash\{ca,cr\}|(E-L_b+L_f)\overline{aout}.BE)\backslash R
\end{aligned}
$$

Which has a cycle time of$G+2(L_b+L_f)$ and is the stable cycling at $E+L_b+L_f$. In the second case $L_b+L_f < D-G$.

$$
\begin{aligned}
& (\underline{ain}.a.Env|((D-G)\overline{out}.Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
& (L_b+L_f)FL[f_2])\backslash\{ca,cr\}|BE)\backslash R \\
\overset{L_b+L_f}{\rightsquigarrow}\; & (\underline{ain}.a.Env|((D-G-L_b-L_f)\overline{out}.Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
& FL[f_2])\backslash\{ca,cr\}|BE)\backslash R \\
\overset{\tau(rout)}{\rightarrow}\; & (\underline{ain}.a.Env|((D-G-L_b-L_f)\overline{out}.Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
& \overline{ain}.WL[f_2])\backslash\{ca,cr\}|(E)\overline{aout}.BE)\backslash R \\
\overset{\tau(ca)}{\rightarrow}\; & (\underline{ain}.a.Env|((D-G-L_b-L_f)\overline{out}.Dl|WLA(L_b)[out/rin])\backslash\{out\}[f_1]| \\
& WL[f_2])\backslash\{ca,cr\}|(E)\overline{aout}.BE)\backslash R
\end{aligned}
$$

Again we have two more cases two consider either $D-G > 2L_b+L_f$ in which case the system will continue to evolve as:

$$
\begin{aligned}
& (\underline{ain}.a.Env|((D-G-L_b-L_f)\overline{out}.Dl|WLA(L_b)[out/rin])\backslash\{out\}[f_1]| \\
& WL[f_2])\backslash\{ca,cr\}|(E)\overline{aout}.BE)\backslash R \\
\overset{L_b}{\rightsquigarrow}\; & (\underline{ain}.a.Env|((D-G-2L_b-L_f)\overline{out}.Dl|L[out/rin])\backslash\{out\}[f_1]| \\
& WL[f_2])\backslash\{ca,cr\}|(E-L_b)\overline{aout}.BE)\backslash R \\
\overset{D-G-2L_b-L_f}{\rightsquigarrow}\; & (\underline{ain}.a.Env|(\overline{out}.Dl|L[out/rin])\backslash\{out\}[f_1]| \\
& WL[f_2])\backslash\{ca,cr\}|(E+G-D+L_b+L_f)\overline{aout}.BE)\backslash R \\
\overset{\tau(out)}{\rightarrow}\; & (\underline{ain}.a.Env|(Dl|(L_f)FL[out/rin])\backslash\{out\}[f_1]| \\
& WL[f_2])\backslash\{ca,cr\}|(E+G-D+L_b+L_f)\overline{aout}.BE)\backslash R \\
\overset{L_f}{\rightsquigarrow}\; & (\underline{ain}.a.Env|(Dl|FL[out/rin])\backslash\{out\}[f_1]| \\
& WL[f_2])\backslash\{ca,cr\}|(E+G-D+L_b)\overline{aout}.BE)\backslash R \\
\overset{\tau(cr)}{\rightarrow}\; & (\underline{ain}.a.Env|(Dl|\overline{ain}.WL[out/rin])\backslash\{out\}[f_1]| \\
& \underline{aout}.(L_b+L_f)FL[f_2])\backslash\{ca,cr\}|(E+G-D+L_b)\overline{aout}.BE)\backslash R \\
\overset{\tau(ain)}{\rightarrow}\; & (a.Env|(Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
& \underline{aout}.(L_b+L_f)FL[f_2])\backslash\{ca,cr\}|(E+G-D+L_b)\overline{aout}.BE)\backslash R \\
\overset{a}{\rightarrow}\; & (Env|(Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
& \underline{aout}.(L_b+L_f)FL[f_2])\backslash\{ca,cr\}|(E+G-D+L_b)\overline{aout}.BE)\backslash R
\end{aligned}
$$

Which has a cycle time of $D+L_f$ and a new 'G' of $E+G-D+L_b$. Finally if $L_b+L_f < D-G < 2L_b+l_f$ then we have the following evolution:

$$
\begin{array}{ll}
 & (\underline{ain}.a.Env|((D-G-L_b-L_f)\overline{out}.Dl|WLA(L_b)[out/rin])\backslash\{out\}[f_1]| \\
 & WL[f_2])\backslash\{ca,cr\}|(E)\overline{aout}.BE)\backslash R \\
\stackrel{D-G-L_b-L_f}{\leadsto} & (\underline{ain}.a.Env|(\overline{out}.Dl|WLA(D-G-L_b-L_f)[out/rin])\backslash\{out\}[f_1]| \\
 & WL[f_2])\backslash\{ca,cr\}|(E-D+G+L_b+L_f)\overline{aout}.BE)\backslash R \\
\stackrel{\tau(out)}{\rightarrow} & (\underline{ain}.a.Env|(Dl|(L_b-D+G+L_b+L_f)(L_f)FL[out/rin])\backslash\{out\}[f_1]| \\
 & WL[f_2])\backslash\{ca,cr\}|(E-D+G+L_b+L_f)\overline{aout}.BE)\backslash R \\
\stackrel{2(L_b+L_f)+G-D}{\leadsto} & (\underline{ain}.a.Env|(Dl|FL[out/rin])\backslash\{out\}[f_1]| \\
 & WL[f_2])\backslash\{ca,cr\}|(E-L_b-L_f)\overline{aout}.BE)\backslash R \\
\stackrel{\tau(cr)}{\rightarrow} & (\underline{ain}.a.Env|(Dl|\overline{ain}.WL[out/rin])\backslash\{out\}[f_1]| \\
 & \underline{aout}.(L_b+L_f)WL[f_2])\backslash\{ca,cr\}|(E-L_b-L_f)\overline{aout}.BE)\backslash R \\
\stackrel{\tau(ain)}{\rightarrow} & (a.Env|(Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
 & \underline{aout}.(L_b+L_f)WL[f_2])\backslash\{ca,cr\}|(E-L_b-L_f)\overline{aout}.BE)\backslash R \\
\stackrel{a}{\rightarrow} & (Env|(Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
 & \underline{aout}.(L_b+L_f)WL[f_2])\backslash\{ca,cr\}|(E-L_b-L_f)\overline{aout}.BE)\backslash R
\end{array}
$$

So after a cycle of length $G+2(L_b+L_f)$ we reach the stable state.
Let us consider the case when $E > D + L_B$, and starting from the first state where this will have any influence:

$$
\begin{array}{ll}
 & (Env|(Dl|FL[out/rin])\backslash\{out\}[f_1]| \\
 & (L_f)FL[f_2])\backslash\{ca,cr\}|BE)\backslash R \\
\stackrel{r}{\rightarrow} & (\overline{rin}.\underline{ain}.a.Env|(Dl|FL[out/rin])\backslash\{out\}| \\
 & (L_f)FL[f_2])\backslash\{ca,cr\}|BE)\backslash R \\
\stackrel{\tau(rin)}{\rightarrow} & (\underline{ain}.a.Env|((D)\overline{out}.Dl|FL[out/rin])\backslash\{out\}[f_1]| \\
 & (L_f)FL[f_2])\backslash\{ca,cr\}|BE)\backslash R \\
\stackrel{L_f}{\leadsto} & (\underline{ain}.a.Env|((D-L_f)\overline{out}.Dl|FL[out/rin])\backslash\{out\}[f_1]| \\
 & \overline{rout}.\overline{ain}.WL[f_2])\backslash\{ca,cr\}|BE)\backslash R \\
\stackrel{\tau(rout)}{\rightarrow} & (\underline{ain}.a.Env|((D-L_f)\overline{out}.Dl|FL[out/rin])\backslash\{out\}[f_1]| \\
 & \overline{ain}.WL[f_2])\backslash\{ca,cr\}|(E)\overline{aout}.BE)\backslash R \\
\stackrel{\tau(ca)}{\rightarrow} & (\underline{ain}.a.Env|((D-L_f)\overline{out}.Dl|WLA[out/rin])\backslash\{out\}[f_1]| \\
 & WL[f_2])\backslash\{ca,cr\}|(E)\overline{aout}.BE)\backslash R \\
\stackrel{L_b}{\leadsto} & (\underline{ain}.a.Env|((D-L_f-L_b)\overline{out}.Dl|WLA[out/rin])\backslash\{out\}[f_1]| \\
 & WL[f_2])\backslash\{ca,cr\}|(E-L_b)\overline{aout}.BE)\backslash R \\
\stackrel{L_b}{\leadsto} & (\underline{ain}.a.Env|((D-L_f-2L_b)\overline{out}.Dl|L[out/rin])\backslash\{out\}[f_1]| \\
 & WL[f_2])\backslash\{ca,cr\}|(E-2L_b)\overline{aout}.BE)\backslash R \\
\stackrel{D-2L_b-L_f}{\leadsto} & (\underline{ain}.a.Env|(\overline{out}.Dl|L[out/rin])\backslash\{out\}[f_1]| \\
 & WL[f_2])\backslash\{ca,cr\}|(E-D+L_f)\overline{aout}.BE)\backslash R \\
\stackrel{\tau(out)}{\rightarrow} & (\underline{ain}.a.Env|(Dl|(L_f)FL[out/rin])\backslash\{out\}[f_1]| \\
 & WL[f_2])\backslash\{ca,cr\}|(E-D+L_f)\overline{aout}.BE)\backslash R \\
\stackrel{L_f}{\leadsto} & (\underline{ain}.a.Env|(Dl|FL[out/rin])\backslash\{out\}[f_1]| \\
 & WL[f_2])\backslash\{ca,cr\}|(E-D)\overline{aout}.BE)\backslash R \\
\stackrel{\tau(cr)}{\rightarrow} & (\underline{ain}.a.Env|(Dl|\overline{ain}.WL[out/rin])\backslash\{out\}[f_1]| \\
 & \underline{aout}.(L_f+L_b)FL[f_2])\backslash\{ca,cr\}|(E-D)\overline{aout}.BE)\backslash R \\
\stackrel{\tau(ain)}{\rightarrow} & (a.Env|(Dl|\overline{ain}.WL[out/rin])\backslash\{out\}[f_1]| \\
 & \underline{aout}.(L_f+L_b)FL[f_2])\backslash\{ca,cr\}|(E-D)\overline{aout}.BE)\backslash R \\
\stackrel{a}{\rightarrow} & (Env|(Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
 & \underline{aout}.(L_f+L_b)FL[f_2])\backslash\{ca,cr\}|(E-D)\overline{aout}.BE)\backslash R
\end{array}
$$

Which is an instance of our generalised state and can be analysed in the above manner.
Let us assume that $E >> D$ then continuing from B_1' we get:

$$
\begin{array}{ll}
B_1' & \\
\stackrel{r}{\rightarrow} & (\overline{rin}.\underline{ain}.a.Env|(Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
 & \underline{aout}.(L_f+L_b)FL[f_2])\backslash\{ca,cr\}|(E-D-L_b)\overline{aout}.BE)\backslash R \\
\stackrel{\tau(rin)}{\rightarrow} & (\underline{ain}.a.Env|((D)\overline{out}.Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
 & \underline{aout}.(L_f+L_b)FL[f_2])\backslash\{ca,cr\}|(E-D-L_b)\overline{aout}.BE)\backslash R \\
\stackrel{D}{\leadsto} & (\underline{ain}.a.Env|(\overline{out}.Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
 & \underline{aout}.(L_f+L_b)FL[f_2])\backslash\{ca,cr\}|(E-2D-L_b)\overline{aout}.BE)\backslash R \\
\stackrel{\tau(out)}{\rightarrow} & (\underline{ain}.a.Env|(Dl|\underline{aout}.(L_f+L_b)FL[out/rin])\backslash\{out\}[f_1]| \\
 & \underline{aout}.(L_f+L_b)FL[f_2])\backslash\{ca,cr\}|(E-2D-L_b)\overline{aout}.BE)\backslash R \\
\stackrel{E-2D-L_b}{\leadsto} & (\underline{ain}.a.Env|(Dl|\underline{aout}.(L_f+L_b)FL[out/rin])\backslash\{out\}[f_1]| \\
 & \underline{aout}.(L_f+L_b)FL[f_2])\backslash\{ca,cr\}|\overline{aout}.BE)\backslash R \\
\stackrel{\tau(aout)}{\rightarrow} & (\underline{ain}.a.Env|(Dl|\underline{aout}.(L_f+L_b)FL[out/rin])\backslash\{out\}[f_1]| \\
 & (L_f+L_b)FL[f_2])\backslash\{ca,cr\}|BE)\backslash R \\
\stackrel{L_f+L_b}{\leadsto} & (\underline{ain}.a.Env|(Dl|\underline{aout}.(L_f+L_b)FL[out/rin])\backslash\{out\}[f_1]| \\
 & FL[f_2])\backslash\{ca,cr\}|BE)\backslash R \\
\stackrel{\tau(rout)}{\rightarrow} & (\underline{ain}.a.Env|(Dl|\underline{aout}.(L_f+L_b)FL[out/rin])\backslash\{out\}[f_1]| \\
 & \overline{ain}.WL[f_2])\backslash\{ca,cr\}|(E)\overline{aout}.BE)\backslash R \\
\stackrel{\tau(ca)}{\rightarrow} & (\underline{ain}.a.Env|(Dl|(L_f+L_b)FL[out/rin])\backslash\{out\}[f_1]| \\
 & WL[f_2])\backslash\{ca,cr\}|(E)\overline{aout}.BE)\backslash R \\
\stackrel{L_f+L_b}{\rightarrow} & (\underline{ain}.a.Env|(Dl|FL[out/rin])\backslash\{out\}[f_1]| \\
 & WL[f_2])\backslash\{ca,cr\}|(E-L_f-L_b)\overline{aout}.BE)\backslash R \\
\stackrel{\tau(cr)}{\rightarrow} & (\underline{ain}.a.Env|(Dl|\overline{ain}.WL[out/rin])\backslash\{out\}[f_1]| \\
 & \underline{aout}.(L_f+L_b)FL[f_2])\backslash\{ca,cr\}|(E-L_f-L_b)\overline{aout}.BE)\backslash R \\
\stackrel{\tau(ain)}{\rightarrow} & (a.Env|(Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
 & \underline{aout}.(L_f+L_b)FL[f_2])\backslash\{ca,cr\}|(E-L_f-L_b)\overline{aout}.BE)\backslash R \\
\stackrel{a}{\rightarrow} & (Env|(Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
 & \underline{aout}.(L_f+L_b)FL[f_2])\backslash\{ca,cr\}|(E-L_f-L_b)\overline{aout}.BE)\backslash R
\end{array}
$$

Call the above state B_2' and continuing:

$$
\begin{array}{ll}
B_2' & \\
\xrightarrow{r} & (\overline{rin}.\underline{ain}.a.Env|(Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
 & \underline{aout}.(L_f+L_b)FL[f_2])\backslash\{ca,cr\}|(E-L_f-L_b)\overline{aout}.BE)\backslash R \\
\xrightarrow{\tau(rin)} & (\underline{ain}.a.Env|((D).\overline{out}.Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
 & \underline{aout}.(L_f+L_b)FL[f_2])\backslash\{ca,cr\}|(E-L_f-L_b)\overline{aout}.BE)\backslash R \\
\stackrel{D}{\leadsto} & (\underline{ain}.a.Env|(\overline{out}.Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
 & \underline{aout}.(L_f+L_b)FL[f_2])\backslash\{ca,cr\}|(E-D-L_f-L_b)\overline{aout}.BE)\backslash R \\
\xrightarrow{\tau(out)} & (\underline{ain}.a.Env|(Dl|\underline{aout}(L_f+L_b)FL[out/rin])\backslash\{out\}[f_1]| \\
 & \underline{aout}.(L_f+L_b)FL[f_2])\backslash\{ca,cr\}|(E-D-L_f-L_b)\overline{aout}.BE)\backslash R \\
\stackrel{E-D-L_f-L_b}{\leadsto} & (\underline{ain}.a.Env|(Dl|\underline{aout}(L_f+L_b)FL[out/rin])\backslash\{out\}[f_1]| \\
 & \underline{aout}.(L_f+L_b)FL[f_2])\backslash\{ca,cr\}|\overline{aout}.BE)\backslash R \\
\xrightarrow{\tau(aout)} & (\underline{ain}.a.Env|(Dl|\underline{aout}.(L_f+L_b)FL[out/rin])\backslash\{out\}[f_1]| \\
 & (L_f+L_b)FL[f_2])\backslash\{ca,cr\}|BE)\backslash R \\
\stackrel{L_f+L_b}{\leadsto} & (\underline{ain}.a.Env|(Dl|\underline{aout}.(L_f+L_b)FL[out/rin])\backslash\{out\}[f_1]| \\
 & FL[f_2])\backslash\{ca,cr\}|BE)\backslash R \\
\xrightarrow{\tau(rout)} & (\underline{ain}.a.Env|(Dl|\underline{aout}.(L_f+L_b)FL[out/rin])\backslash\{out\}[f_1]| \\
 & \overline{ain}.WL[f_2])\backslash\{ca,cr\}|(E).\overline{aout}.BE)\backslash R \\
\xrightarrow{\tau(ca)} & (\underline{ain}.a.Env|(Dl|(L_f+L_b)FL[out/rin])\backslash\{out\}[f_1]| \\
 & WL[f_2])\backslash\{ca,cr\}|(E).\overline{aout}.BE)\backslash R \\
\stackrel{L_f+L_b}{\leadsto} & (\underline{ain}.a.Env|(Dl|FL[out/rin])\backslash\{out\}[f_1]| \\
 & WL[f_2])\backslash\{ca,cr\}|(E-L_f-L_b).\overline{aout}.BE)\backslash R \\
\xrightarrow{\tau(rout)} & (\underline{ain}.a.Env|(Dl|\overline{ain}.WL[out/rin])\backslash\{out\}[f_1]| \\
 & \underline{aout}.(L_f+L_b).FL[f_2])\backslash\{ca,cr\}|(E-L_f-L_b).\overline{aout}.BE)\backslash R \\
\xrightarrow{\tau(ain)} & (a.Env|(Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
 & \underline{aout}.(L_f+L_b).FL[f_2])\backslash\{ca,cr\}|(E-L_f-L_b).\overline{aout}.BE)\backslash R \\
\xrightarrow{a} & (Env|(Dl|WL[out/rin])\backslash\{out\}[f_1]| \\
 & \underline{aout}.(L_f+L_b).FL[f_2])\backslash\{ca,cr\}|(E-L_f-L_b).\overline{aout}.BE)\backslash R
\end{array}
$$

Which is the same process we started with and gives a cycle time of $E + L_f + L_b$.

So in summary, we have three possibilities if $E > 2D + L_f$ then the system will have the following cycle times:

	$E + L_b \leq D$	$D - L_b < E \leq D$	$D < E$
Latency	$D + L_f$	$D + L_f$	$D + L_f$
Cycle time	$D + L_f$	$E + L_b + L_f$	$E + L_b + L_f$

If $D < E < D + l_f$ then we have an initial two cycles of length $D + L_f$. Subsequently we calculate the cycle times on the basis of the amount of displacement (or precomputation) in the environment, with respect to the start time of the request, represented by the variable G and its cahnge over the computation. Whilst $G < D + L_b$ the system continues to cycle at $D + L_f$, eventually cycling at $E + L_f + L_b$.

2.3 Forks in the Pipeline

We consider the effect of forking the flow of control into more than one path. For the moment we consider that the pipelines do not resynchronise at a latter point. The fork of control is modelled by the following process:

$$
\begin{array}{lcl}
Fo & \stackrel{def}{=} & \underline{rin}.F1 \\
F1 & \stackrel{def}{=} & \overline{rup}.\overline{rlow}.FB \\
FB & \stackrel{def}{=} & \underline{aup}.\underline{alow}.FA + \underline{alow}.\underline{aup}.FA \\
FA & \stackrel{def}{=} & \overline{ain}.Fo
\end{array}
$$

In the above we do not worry about the order of initialisation since both processes will be available. We can use the same environment process to drive this system, but need two different back end processes.

$$
\begin{array}{lcl}
BU & \stackrel{def}{=} & \underline{rup}(EU).\overline{aup}.BU \\
BL & \stackrel{def}{=} & \underline{rlow}(EL).\overline{alow}.BL
\end{array}
$$

Without loss of generality we can assume that $EU \geq EL$. Notice that the abstraction we use here is the same one that describes the behaviour of our pipline stages. Hence, we can consistently use forks as pipeline stages. The complete system will be the following process:

$$\begin{array}{lcl} R_1 & \stackrel{def}{=} & \{rup, aup, rlow, alow, rin, ain\} \\ FS & \stackrel{def}{=} & (Env|Fo|BL|BU)\backslash R_1 \end{array}$$

So deriving the behaviour of the above system we see that:

$$\begin{array}{lcl} FS & \xrightarrow{r} & (\overline{rin}.\underline{ain}.a.Env|Fo|BL|BU)\backslash R_1 \\ & \xrightarrow{\tau(rin)} & (\underline{ain}.a.Env|F1|BL|BU)\backslash R_1 \\ & \xrightarrow{\tau(rup)} & (\underline{ain}.a.Env|\overline{rlow}.FB|BL|(EU)\overline{aup}.BU)\backslash R_1 \\ & \xrightarrow{\tau(rlow)} & (\underline{ain}.a.Env|FB|(EL)\overline{alow}.BL|(EU)\overline{aup}.BU)\backslash R_1 \\ & \stackrel{EL}{\leadsto} & (\underline{ain}.a.Env|FB|\overline{alow}.BL|(EU-EL)\overline{aup}.BU)\backslash R_1 \\ & \xrightarrow{\tau(alow)} & (\underline{ain}.a.Env|\underline{aup}.FA|BL|(EU-EL)\overline{aup}.BU)\backslash R_1 \\ & \stackrel{EU-EL}{\leadsto} & (\underline{ain}.a.Env|\underline{aup}.FA|BL|\overline{aup}.BU)\backslash R_1 \\ & \xrightarrow{\tau(aup)} & (\underline{ain}.a.Env|FA|BL|BU)\backslash R_1 \\ & \xrightarrow{\tau(ain)} & (a.Env|Fo|BL|BU)\backslash R_1 \\ & \xrightarrow{a} & (Env|Fo|BL|BU)\backslash R_1 \end{array}$$

So unsurprisingly the cycle time is given by the greater of the times taken to execute by either of the pipeline stages.

2.4 Stages with back delays

A more realistic description would be one where the return acknowledge signal from a latch is also subject to a delay. We can model such a pipeline stage by the following process:

$$\begin{array}{lcl} DF & \stackrel{def}{=} & \underline{rin}.(D_f)\overline{out}.Dl \\ DB & \stackrel{def}{=} & \underline{in}.(D_b)\overline{ain}.DB \end{array}$$

our pipeline stage is now described by the following process:

$$BSt \stackrel{def}{=} (DF|DB|L[out/rin, in/ain])\backslash\{out, in\}$$

We can try and understand how this pipeline stage will perform by placing it within the usual context:

$$BS \stackrel{def}{=} (Env|BSt|BE)\backslash R$$

Examing the evolution of the above process we see that assume that $(E > D_b)$:

$$\begin{array}{lcl} BS & \xrightarrow{r} & (\overline{rin}.\underline{ain}.a.Env|BSt|BE)\backslash R \\ & \xrightarrow{\tau(rin)} & (\underline{ain}.a.Env|((D_f)\overline{out}.DF|DB|L[out/rin, in/ain])\backslash\{out, in\}|BE)\backslash R \\ & \stackrel{D_f}{\leadsto} & (\underline{ain}.a.Env|(\overline{out}.DF|DB|L[out/rin, in/ain])\backslash\{out, in\}|BE)\backslash R \\ & \xrightarrow{\tau(out)} & (\underline{ain}.a.Env|(DF|DB|(L_f)FL[out/rin, in/ain])\backslash\{out, in\}|BE)\backslash R \\ & \stackrel{L_f}{\leadsto} & (\underline{ain}.a.Env|(DF|DB|FL[out/rin, in/ain])\backslash\{out, in\}|BE)\backslash R \\ & \xrightarrow{\tau(rout)} & (\underline{ain}.a.Env|(DF|DB|\overline{ain}.WL[out/rin, in/ain])\backslash\{out, in\}|(E)\overline{aout}.BE)\backslash R \\ & \xrightarrow{\tau(in)} & (\underline{ain}.a.Env|(DF|(D_b)\overline{ain}.DB|WL[out/rin, in/ain])\backslash\{out, in\}|(E)\overline{aout}.BE)\backslash R \\ & \stackrel{D_b}{\leadsto} & (\underline{ain}.a.Env|(DF|\overline{ain}.DB|WL[out/rin, in/ain])\backslash\{out, in\}|(E-D_B)\overline{aout}.BE)\backslash R \\ & \xrightarrow{\tau(ain)} & (a.Env|(DF|DB|WL[out/rin, in/ain])\backslash\{out, in\}|(E-D_B)\overline{aout}.BE)\backslash R \\ & \xrightarrow{a} & (Env|(DF|DB|WL[out/rin, in/ain])\backslash\{out, in\}|(E-D_B)\overline{aout}.BE)\backslash R \end{array}$$

Now we get the three same evolutions as before, depending on whether $E - D_B - L_b < D_f$ in which case the cycle time will be $D_f + D_b + L_f$ otherwise the cycle time will be $E + L_f + L_b$. So the temporal behaviour of the above is the same as if the delay of the computation was $D_f + D_b$ so we can treat the circuit as simply having a delay on forward computation equal to the sum of the delays in both directions.

2.5 Variable Environments

So far we have assumed that the environment in which a pipeline stage exists is constant. However, since we wish to add stages to pipelines we note that our single stages actually have variable response times.In the previous systems it was always the case that the sequence of response times was non-decreasing.

Consider an environment where we have a nondecreasing sequence of times $E_1, E_2, \ldots, E_i, E_{i+1}, \ldots$ where after some i, for $j \geq i.E_j = E_i$. We can model such an environment by the processes (with $1 < k < i$):

$$\begin{array}{lcl}
BE_1 & \stackrel{def}{=} & \underline{rout}.(E_1)\overline{aout}.BE_2 \\
BE_k & \stackrel{def}{=} & \underline{rout}.(E_k)\overline{aout}.BE_{k+1} \\
BE_i & \stackrel{def}{=} & \underline{rout}.(E_i)\overline{aout}.BE_i
\end{array}$$

Forming our system as before, we have the following process:

$$V \stackrel{def}{=} (Env|Sth|BE_1)\backslash R$$

Examining the evolution of the above process we discover that:

$$\begin{array}{lll}
V & \xrightarrow{r} & (\overline{rin}.\underline{ain}.a.Env|Sth|BE_1)\backslash R \\
 & \xrightarrow{\tau(rin)} & (\underline{ain}.a.Env|((D)\overline{out}.Dl|L[out/rin])\backslash\{out\}|BE_1)\backslash R \\
 & \overset{D}{\rightsquigarrow} & (\underline{ain}.a.Env|(\overline{out}.Dl|L[out/rin])\backslash\{out\}|BE_1)\backslash R \\
 & \xrightarrow{\tau(out)} & (\underline{ain}.a.Env|((Dl|(L_f)FL[out/rin])\backslash\{out\}|BE_1)\backslash R \\
 & \overset{L_f}{\rightsquigarrow} & (\underline{ain}.a.Env|((D)\overline{out}.Dl|FL[out/rin])\backslash\{out\}|BE_1)\backslash R \\
 & \xrightarrow{\tau(rout)} & (\underline{ain}.a.Env|(Dl|\overline{ain}.WL[out/rin])\backslash\{out\}|(E_1)\overline{aout}.BE_2)\backslash R \\
 & \xrightarrow{\tau(ain)} & (a.Env|(Dl|WL[out/rin])\backslash\{out\}|(E_1)\overline{aout}.BE_2)\backslash R \\
 & \xrightarrow{a} & (Env|(Dl|WL[out/rin])\backslash\{out\}|(E_1)\overline{aout}.BE_2)\backslash R
\end{array}$$

Assuming $E_1 - L_b < D$ we have the following evolution:

$$\begin{array}{ll}
\xrightarrow{r} & (\overline{rin}.\underline{ain}.a.Env|(Dl|WL[out/rin])\backslash\{out\}|(E_1)\overline{aout}.BE_2)\backslash R \\
\xrightarrow{\tau(rin)} & (\underline{ain}.a.Env|((D)\overline{out}.Dl|WL[out/rin])\backslash\{out\}|(E_1)\overline{aout}.BE_2)\backslash R \\
\overset{E_1}{\rightsquigarrow} & (\underline{ain}.a.Env|((D-E)\overline{out}.Dl|WL[out/rin])\backslash\{out\}|\overline{aout}.BE_2)\backslash R \\
\xrightarrow{\tau(aout)} & (\underline{ain}.a.Env|((D-E)\overline{out}.Dl|WLA[out/rin])\backslash\{out\}|BE_2)\backslash R \\
\overset{L_b}{\rightsquigarrow} & (\underline{ain}.a.Env|((D-E-L_b)\overline{out}.Dl|L[out/rin])\backslash\{out\}|BE_2)\backslash R \\
\overset{D-E-L_b}{\rightsquigarrow} & (\underline{ain}.a.Env|(\overline{out}.Dl|L[out/rin])\backslash\{out\}|BE_2)\backslash R \\
\xrightarrow{\tau(out)} & (\underline{ain}.a.Env|((Dl|(L_f)FL[out/rin])\backslash\{out\}|BE_2)\backslash R \\
\overset{L_f}{\rightsquigarrow} & (\underline{ain}.a.Env|((D)\overline{out}.Dl|FL[out/rin])\backslash\{out\}|BE_2)\backslash R \\
\xrightarrow{\tau(rout)} & (\underline{ain}.a.Env|(Dl|\overline{ain}.WL[out/rin])\backslash\{out\}|(E_2)\overline{aout}.BE_3)\backslash R \\
\xrightarrow{\tau(ain)} & (a.Env|(Dl|WL[out/rin])\backslash\{out\}|(E_2)\overline{aout}.BE_3)\backslash R \\
\xrightarrow{a} & (Env|(Dl|WL[out/rin])\backslash\{out\}|(E_2)\overline{aout}.BE_3)\backslash R
\end{array}$$

Assume that $E_j < D - L_b$ for $1 \leq j < k$, obviously if the fixed environment time is less then $D - L_b$ this k does not exist and the system continues to behave as above. As we have already seen in Section 1, this will lead to a cycle time of $E_k + L_b + L_f$, stabilising at $E_i + L_b + L_f$.

So by placing a stage in front we get a sequence of times of the following form:

Cycle	Duration
1	$D + L_f$
2	$D + L_f$
$\vdots$	$\vdots$
$k+1$	$E_k + L_b + L_f$
$\vdots$	$\vdots$
i	$E_{i-1} + L_b + L_f$

Obviously if $E_i < D - L_b$ then we see cycles of constant duration $D + L_f$.

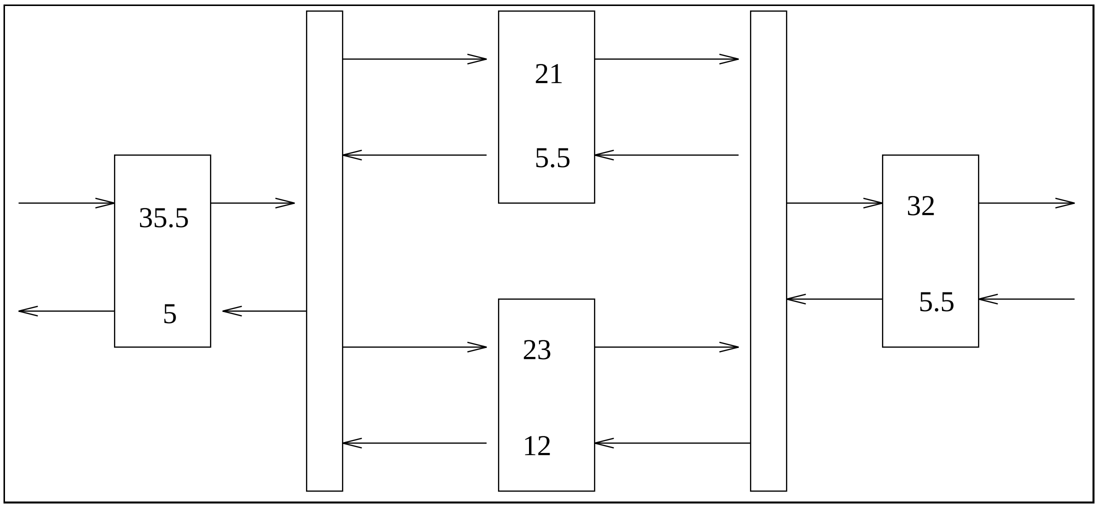

Figure 4: The Amulet I Instruction Execution pipeline with stage forward and back times.

2.6 The Amulet Pipeline

The Amulet I pipeline can be pictured as in Figure 4.

In order to estimate its performance we model it as described by Figure 5.

In the Amulet [Fur93] we can take the times L_b and L_f to be 6.

So assuming that the final environment has a response time of better than $37.5 - 6 = 31.5$. Then at the fork we see pipeline stages behaving with times $37.5 + 6 + 12 = 55.5$ and 55.5 for the upper and lower stages respectively, so the fork will respond in 55.5 and the whole system in 67.5.

The time to first response will be $35.5 + 5 + 6 = 46.5$ and the second response will be at 53 (equal to max of 32.5 and 41 plus 12), the third response will be after 67.5 the stable cycle time.

So assuming we have to load and execute 5 instructions per fork the total time will be $46.5 + 53 + 5 * 67.5 = 437$, which gives an average instruction time of 86.4. Or the initial latency is 167 and the throughput rate is 67.5.

Clearly the performance of the pipeline is dominated by the last stage. The optimal performance (assuming that the first stage is unchanged) is 46.5 and can be achieved by making the total time of the second pipeline stages less than 40.5 and that of the third less than 34.5.

2.7 Calculating times forward

So far we have assumed that the pipeline stages will dominate the performance of the system. A possible other case it that the two are in close balance, that is the initial system will provide information to the pipeline at some fixed (maximal) rate, the question is then can the pipeline cope with this information flux rate.

We need a new kind of environment to drive the pipeline stage. This will provide data at a fixed rate, but will obey the data bundling requirements if the pipeline cannot execute:

$$
\begin{array}{rcl}
Cl & \stackrel{def}{=} & \underline{sc}.(C)\overline{fc}.Cl \\
EP & \stackrel{def}{=} & \overline{rin}.\overline{sc}.EPW \\
EPW & \stackrel{def}{=} & (\underline{ain}.EPC + \underline{fc}.EPA) \\
EPC & \stackrel{def}{=} & \underline{fc}.EP \\
EPA & \stackrel{def}{=} & \underline{ain}.EP \\
EPD & \stackrel{def}{=} & (EP|Cl)\backslash\{sc, fc\}
\end{array}
$$

An interesting observation is that if this process reaches the state EPA then the front end of the pipeline has become stalled. That is the pipeline *cannot* process information as rapidly as the environment wishes to produce it.

We also need a new notion of back end environment,

$$NBE \stackrel{def}{=} \underline{rout}.r.\overline{aout}.NBE$$

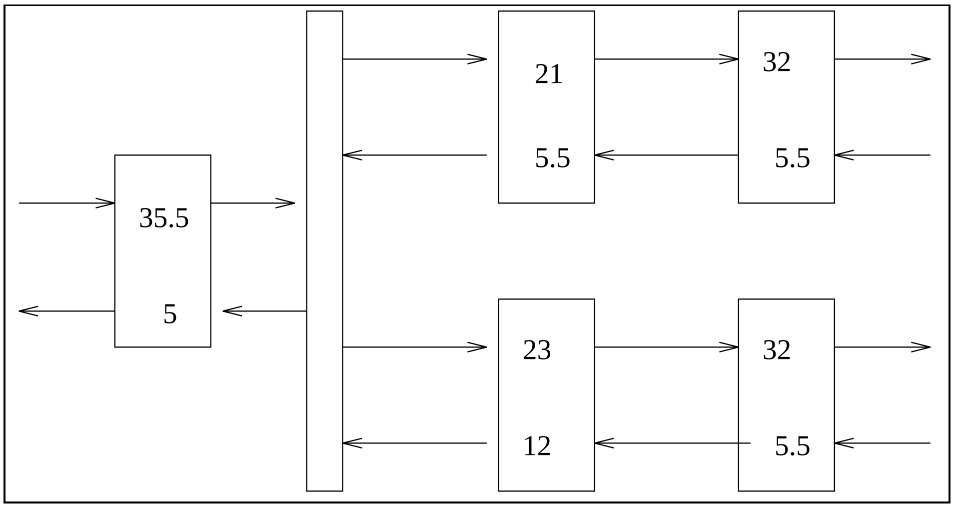

Figure 5: The Amulet I Instruction Execution pipeline model.

We examine the duration between the r actions as the drive rate for the next pipline stage on. The complete system is the process:

$$F \stackrel{def}{=} (EPD|Sth|NBE)\backslash R$$

Let us examine its evolution assuming $D + L_f + L_b < C$.

$$\begin{array}{lll}
F & \xrightarrow{\tau(rin)} & ((\overline{sc}.EPW|Cl)\backslash\{sc,fc\}|((D)\overline{out}.Dl|L[out/rin])\backslash\{out\}|NBE)\backslash R \\
 & \xrightarrow{\tau(sc)} & ((EPW|(C)\overline{fc}.Cl)\backslash\{sc,fc\}|((D)\overline{out}.Dl|L[out/rin])\backslash\{out\}|NBE)\backslash R \\
 & \stackrel{D}{\rightsquigarrow} & ((EPW|(C-D)\overline{fc}.Cl)\backslash\{sc,fc\}|(\overline{out}.Dl|L[out/rin])\backslash\{out\}|NBE)\backslash R \\
 & \xrightarrow{\tau(out)} & ((EPW|(C-D)\overline{fc}.Cl)\backslash\{sc,fc\}|(Dl|(L_f)FL[out/rin])\backslash\{out\}|NBE)\backslash R \\
 & \stackrel{L_f}{\rightsquigarrow} & ((EPW|(C-D-L_f)\overline{fc}.Cl)\backslash\{sc,fc\}|(Dl|FL[out/rin])\backslash\{out\}|NBE)\backslash R \\
 & \xrightarrow{\tau(rout)} & ((EPW|(C-D-L_f)\overline{fc}.Cl)\backslash\{sc,fc\}| \\
 & & (Dl|\overline{ain}.WL[out/rin])\backslash\{out\}|r.\overline{aout}.NBE)\backslash R \\
 & \xrightarrow{r} & ((EPW|(C-D-L_f)\overline{fc}.Cl)\backslash\{sc,fc\}| \\
 & & (Dl|\overline{ain}.WL[out/rin])\backslash\{out\}|\overline{aout}.NBE)\backslash R \\
 & \xrightarrow{\tau(ain)} & ((EPC|(C-D-L_f)\overline{fc}.Cl)\backslash\{sc,fc\}| \\
 & & (Dl|WL[out/rin])\backslash\{out\}|\overline{aout}.NBE)\backslash R \\
 & \xrightarrow{\tau(aout)} & ((EPC|(C-D-L_f)\overline{fc}.Cl)\backslash\{sc,fc\}| \\
 & & (Dl|WLA(L_b)[out/rin])\backslash\{out\}|NBE)\backslash R \\
 & \stackrel{L_b}{\rightsquigarrow} & ((EPC|(C-D-L_f+L_b)\overline{fc}.Cl)\backslash\{sc,fc\}| \\
 & & (Dl|L[out/rin])\backslash\{out\}|NBE)\backslash R \\
 & \stackrel{C-D-L_f-L_f}{\rightsquigarrow} & ((EPC|\overline{fc}.Cl)\backslash\{sc,fc\}|(Dl|L[out/rin])\backslash\{out\}|NBE)\backslash R \\
 & \xrightarrow{\tau(fc)} & ((EP|Cl)\backslash\{sc,fc\}|(Dl|L[out/rin])\backslash\{out\}|NBE)\backslash R
\end{array}$$

if $D + L_f < C < D + L_f + L_b$ then we see the following pattern:

$$
\begin{array}{rl}
 & ((EPC|(C-D-L_f)\overline{fc}.Cl)\backslash\{sc,fc\}|(Dl|WLA(L_b)[out/rin])\backslash\{out\}|NBE)\backslash R \\
\stackrel{C-D-L_f}{\leadsto} & ((EPC|(\overline{fc}.Cl)\backslash\{sc,fc\}|(Dl|WLA(L_b+L_f+D-C)[out/rin])\backslash\{out\}|NBE)\backslash R \\
\stackrel{\tau(fc)}{\longrightarrow} & ((EPC|Cl)\backslash\{sc,fc\}|(Dl|WLA(L_b+L_f+D-C)[out/rin])\backslash\{out\}|NBE)\backslash R \\
\stackrel{\tau(rin)}{\longrightarrow} & ((\overline{sc}.EPW|Cl)\backslash\{sc,fc\}| \\
 & ((D)\overline{out}.Dl|WLA(L_b+L_f+D-C)[out/rin])\backslash\{out\}|NBE)\backslash R \\
\stackrel{\tau(sc)}{\longrightarrow} & ((EPW|(C)\overline{fc}.Cl)\backslash\{sc,fc\}| \\
 & ((D)\overline{out}.Dl|WLA(L_b+L_f+D-C)[out/rin])\backslash\{out\}|NBE)\backslash R \\
\stackrel{L_b+L_f+D-C}{\leadsto} & ((EPW|(2C-L_b-L_f-D)\overline{fc}.Cl)\backslash\{sc,fc\}| \\
 & ((C-L_b-L_f)\overline{out}.Dl|L[out/rin])\backslash\{out\}|NBE)\backslash R \\
\stackrel{C-L_b-L_f}{\leadsto} & ((EPW|(C-D)\overline{fc}.Cl)\backslash\{sc,fc\}| \\
 & (\overline{out}.Dl|L[out/rin])\backslash\{out\}|NBE)\backslash R \\
\stackrel{\tau(out)}{\longrightarrow} & ((EPW|(C-D)\overline{fc}.Cl)\backslash\{sc,fc\}|(Dl|(L_f)FL[out/rin])\backslash\{out\}|NBE)\backslash R \\
\stackrel{L_f}{\leadsto} & ((EPW|(C-D-L_f)\overline{fc}.Cl)\backslash\{sc,fc\}|(Dl|FL[out/rin])\backslash\{out\}|NBE)\backslash R \\
\stackrel{\tau(rout)}{\longrightarrow} & ((EPW|(C-D-L_f)\overline{fc}.Cl)\backslash\{sc,fc\}|(Dl|\overline{ain}.WL[out/rin])\backslash\{out\}|r.\overline{aout}.NBE)\backslash R \\
\stackrel{r}{\longrightarrow} & ((EPW|(C-D-L_f)\overline{fc}.Cl)\backslash\{sc,fc\}|(Dl|\overline{ain}.WL[out/rin])\backslash\{out\}|\overline{aout}.NBE)\backslash R \\
\stackrel{\tau(ain)}{\longrightarrow} & ((EPC|(C-D-L_f)\overline{fc}.Cl)\backslash\{sc,fc\}|(Dl|\overline{ain}.WL[out/rin])\backslash\{out\}|\overline{aout}.NBE)\backslash R \\
\stackrel{\tau(aout)}{\longrightarrow} & ((EPC|(C-D-L_f)\overline{fc}.Cl)\backslash\{sc,fc\}|(Dl|WLA(L_b)[out/rin])\backslash\{out\}|NBE)\backslash R
\end{array}
$$

Which is the state we started in. So again we have stable cycles of length C. Notice we never observed the environment in state EPA so this pipeline stage did not stall.

Now consider the case when $C < D + L_f$, for brevity we assume $D < C$.

$$
\begin{array}{rrl}
F & \stackrel{\tau(rin)}{\longrightarrow} & ((\overline{sc}.EPW|Cl)\backslash\{sc,fc\}|((D)\overline{out}.Dl|L[out/rin])\backslash\{out\}|NBE)\backslash R \\
 & \stackrel{\tau(sc)}{\longrightarrow} & ((EPW|(C)\overline{fc}.Cl)\backslash\{sc,fc\}|((D)\overline{out}.Dl|L[out/rin])\backslash\{out\}|NBE)\backslash R \\
 & \stackrel{C}{\leadsto} & ((EPW|\overline{fc}.Cl)\backslash\{sc,fc\}|((D-C)\overline{out}.Dl|L[out/rin])\backslash\{out\}|NBE)\backslash R \\
 & \stackrel{\tau(fc)}{\longrightarrow} & ((EPA|Cl)\backslash\{sc,fc\}|((D-C)\overline{out}.Dl|L[out/rin])\backslash\{out\}|NBE)\backslash R \\
 & \stackrel{D-C}{\leadsto} & ((EPA|Cl)\backslash\{sc,fc\}|(\overline{out}.Dl|L[out/rin])\backslash\{out\}|NBE)\backslash R \\
 & \stackrel{\tau(out)}{\longrightarrow} & ((EPA|Cl)\backslash\{sc,fc\}|(Dl|(L_f)FL[out/rin])\backslash\{out\}|NBE)\backslash R \\
 & \stackrel{L_F}{\leadsto} & ((EPA|Cl)\backslash\{sc,fc\}|(Dl|FL[out/rin])\backslash\{out\}|NBE)\backslash R \\
 & \stackrel{\tau(rout)}{\longrightarrow} & ((EPA|Cl)\backslash\{sc,fc\}|(Dl|\overline{ain}.WL[out/rin])\backslash\{out\}|r.\overline{aout}.NBE)\backslash R \\
 & \stackrel{\tau(ain)}{\longrightarrow} & ((EP|Cl)\backslash\{sc,fc\}|(Dl|\overline{ain}.WL[out/rin])\backslash\{out\}|r.\overline{aout}.NBE)\backslash R \\
 & \stackrel{r}{\longrightarrow} & ((EP|Cl)\backslash\{sc,fc\}|(Dl|WL[out/rin])\backslash\{out\}|\overline{aout}.NBE)\backslash R \\
 & \stackrel{\tau(aout)}{\longrightarrow} & ((EP|Cl)\backslash\{sc,fc\}|(Dl|WLA(L_b)[out/rin])\backslash\{out\}|NBE)\backslash R
\end{array}
$$

$$
\begin{array}{rl}
 & ((EP|Cl)\backslash\{sc,fc\}|(Dl|WLA(L_b)[out/rin])\backslash\{out\}|NBE)\backslash R \\
\stackrel{\tau(rin)}{\longrightarrow} & ((\overline{sc}.EPW|Cl)\backslash\{sc,fc\}|((D)\overline{out}.Dl|WLA(L_b)[out/rin])\backslash\{out\}|NBE)\backslash R \\
\stackrel{\tau(sc)}{\longrightarrow} & ((EPW|(C)\overline{fc}.Cl)\backslash\{sc,fc\}|((D)\overline{out}.Dl|WLA(L_b)[out/rin])\backslash\{out\}|NBE)\backslash R \\
\stackrel{L_b}{\leadsto} & ((EPW|(C-L_b)\overline{fc}.Cl)\backslash\{sc,fc\}|((D-L_b)\overline{out}.Dl|L[out/rin])\backslash\{out\}|NBE)\backslash R \\
\stackrel{C-L_b}{\leadsto} & ((EPW|\overline{fc}.Cl)\backslash\{sc,fc\}|((D-C)\overline{out}.Dl|L[out/rin])\backslash\{out\}|NBE)\backslash R
\end{array}
$$

Which is a state we have seen before, so we can calculate the effective cycle time.

2.7.1 Prebuffering

The dual of the extra latch when calculating the cycle time with respect to the forward drive, is to take an extra latch in front of the computational pipeline stage. In this case we end up with the following process:

$$
\begin{array}{rcl}
f_1 & \stackrel{def}{=} & [cr/rout, ca/aout] \\
f_2 & \stackrel{def}{=} & [cr/rin, ca/ain] \\
FBP & \stackrel{def}{=} & (L[f_1]|(Dl|L[out/rin])\backslash\{out\}[f_2])\backslash\{ca,cr\}
\end{array}
$$

the complete system is now:

$$FB \stackrel{def}{=} (EPD|FBP|NBE)\backslash R$$

Let us examine its behaviour in the usual fashion, assume $C > L_f$ and $C < L_f + D$, since we are interested in prolonging the avoidance of stalls, we shall follow the evolutionary path that avoids them for as long as possible.

$$\begin{array}{lll}
FB & \xrightarrow{\tau(rin)} & ((\overline{sc}.EPW|Cl)\backslash\{sc,fc\}|((L_f)FL[f_1]| \\
 & & (Dl|L[out/rin])\backslash\{out\}[f_2])\backslash\{ca,cr\}|NBE)\backslash R \\
 & \xrightarrow{\tau(sc)} & ((EPW|(C)\overline{fc}.Cl)\backslash\{sc,fc\}|((L_f)FL[f_1]| \\
 & & (Dl|L[out/rin])\backslash\{out\}[f_2])\backslash\{ca,cr\}|NBE)\backslash R \\
 & \stackrel{L_f}{\leadsto} & ((EPW|(C)\overline{fc}.Cl)\backslash\{sc,fc\}|(FL[f_1]| \\
 & & (Dl|L[out/rin])\backslash\{out\}[f_2])\backslash\{ca,cr\}|NBE)\backslash R \\
 & \xrightarrow{\tau(cr)} & ((EPW|(C-L_f)\overline{fc}.Cl)\backslash\{sc,fc\}|(\overline{ain}.WL[f_1]| \\
 & & ((D)\overline{out}.Dl|L[out/rin])\backslash\{out\}[f_2])\backslash\{ca,cr\}|NBE)\backslash R \\
 & \xrightarrow{\tau(ain)} & ((EPC|(C-L_f)\overline{fc}.Cl)\backslash\{sc,fc\}|(WL[f_1]| \\
 & & ((D)\overline{out}.Dl|L[out/rin])\backslash\{out\}[f_2])\backslash\{ca,cr\}|NBE)\backslash R \\
 & \stackrel{C-L_f}{\leadsto} & ((EPC|\overline{fc}.Cl)\backslash\{sc,fc\}|(WL[f_1]| \\
 & & ((D-C+L_f)\overline{out}.Dl|L[out/rin])\backslash\{out\}[f_2])\backslash\{ca,cr\}|NBE)\backslash R \\
 & \xrightarrow{\tau(fc)} & ((EP|Cl)\backslash\{sc,fc\}|(WL[f_1]| \\
 & & ((D-C+L_f)\overline{out}.Dl|L[out/rin])\backslash\{out\}[f_2])\backslash\{ca,cr\}|NBE)\backslash R \\
 & \xrightarrow{\tau(rin)} & ((\overline{sc}.EPW|Cl)\backslash\{sc,fc\}|(\underline{aout}.(L_f+L_b)FL[f_1]| \\
 & & ((D-C+L_f)\overline{out}.Dl|L[out/rin])\backslash\{out\}[f_2])\backslash\{ca,cr\}|NBE)\backslash R \\
 & \xrightarrow{\tau(sc)} & ((EPW|(C)\overline{fc}.Cl)\backslash\{sc,fc\}|(\underline{aout}.(L_f+L_b)FL[f_1]| \\
 & & ((D-C+L_f)\overline{out}.Dl|L[out/rin])\backslash\{out\}[f_2])\backslash\{ca,cr\}|NBE)\backslash R \\
 & \stackrel{D-C+L_f}{\leadsto} & ((EPW|(2C-D-L_f)\overline{fc}.Cl)\backslash\{sc,fc\}|(\underline{aout}.(L_f+L_b)FL[f_1]| \\
 & & (\overline{out}.Dl|L[out/rin])\backslash\{out\}[f_2])\backslash\{ca,cr\}|NBE)\backslash R \\
 & \xrightarrow{\tau(out)} & ((EPW|(2C-D-L_f)\overline{fc}.Cl)\backslash\{sc,fc\}|(\underline{aout}.(L_f+L_b)FL[f_1]| \\
 & & (Dl|(L_f)FL[out/rin])\backslash\{out\}[f_2])\backslash\{ca,cr\}|NBE)\backslash R \\
 & \stackrel{L_f}{\leadsto} & ((EPW|(2C-D-2L_f)\overline{fc}.Cl)\backslash\{sc,fc\}|(\underline{aout}.(L_f+L_b)FL[f_1]| \\
 & & (Dl|FL[out/rin])\backslash\{out\}[f_2])\backslash\{ca,cr\}|NBE)\backslash R \\
 & \xrightarrow{\tau(rout)} & ((EPW|(2C-D-2L_f)\overline{fc}.Cl)\backslash\{sc,fc\}|(\underline{aout}.(L_f+L_b)FL[f_1]| \\
 & & (Dl|\overline{ain}.WL[out/rin])\backslash\{out\}[f_2])\backslash\{ca,cr\}|r.\overline{aout}.NBE)\backslash R \\
 & \xrightarrow{r} & ((EPW|(2C-D-2L_f)\overline{fc}.Cl)\backslash\{sc,fc\}|(\underline{aout}.(L_f+L_b)FL[f_1]| \\
 & & (Dl|\overline{ain}.WL[out/rin])\backslash\{out\}[f_2])\backslash\{ca,cr\}|\overline{aout}.NBE)\backslash R \\
 & \xrightarrow{\tau(ca)} & ((EPW|(2C-D-2L_f)\overline{fc}.Cl)\backslash\{sc,fc\}|((L_f+L_b)FL[f_1]| \\
 & & (Dl|WL[out/rin])\backslash\{out\}[f_2])\backslash\{ca,cr\}|\overline{aout}.NBE)\backslash R \\
 & \xrightarrow{\tau(aout)} & ((EPW|(2C-D-2L_f)\overline{fc}.Cl)\backslash\{sc,fc\}|((L_f+L_b)FL[f_1]| \\
 & & (Dl|WLA(L_b)[out/rin])\backslash\{out\}[f_2])\backslash\{ca,cr\}|NBE)\backslash R \\
 & \stackrel{L_b}{\leadsto} & ((EPW|(2C-D-2L_f-L_b)\overline{fc}.Cl)\backslash\{sc,fc\}|((L_f)FL[f_1]| \\
 & & (Dl|L[out/rin])\backslash\{out\}[f_2])\backslash\{ca,cr\}|NBE)\backslash R \\
 & \stackrel{L_f}{\leadsto} & ((EPW|(2C-D-3L_f-L_b)\overline{fc}.Cl)\backslash\{sc,fc\}|(FL[f_1]| \\
 & & (Dl|L[out/rin])\backslash\{out\}[f_2])\backslash\{ca,cr\}|NBE)\backslash R \\
 & \xrightarrow{\tau(cr)} & ((EPW|(2C-D-3L_f-L_b)\overline{fc}.Cl)\backslash\{sc,fc\}|(\overline{ain}.WL[f_1]| \\
 & & ((D)\overline{out}.Dl|L[out/rin])\backslash\{out\}[f_2])\backslash\{ca,cr\}|NBE)\backslash R \\
 & \xrightarrow{\tau(ain)} & ((EPC|(2C-D-3L_f-L_b)\overline{fc}.Cl)\backslash\{sc,fc\}|(WL[f_1]| \\
 & & ((D)\overline{out}.Dl|L[out/rin])\backslash\{out\}[f_2])\backslash\{ca,cr\}|NBE)\backslash R
\end{array}$$

We can see that we have a general state above, so consider the following state, and assume that we are not going to stall:

$$
\begin{array}{ll}
 & ((EPC|(G)\overline{fc}.Cl)\backslash\{sc,fc\}|(WL[f_1]| \\
 & ((D)\overline{out}.Dl|L[out/rin])\backslash\{out\}[f_2])\backslash\{ca,cr\}|NBE)\backslash R \\
\overset{G}{\rightsquigarrow} & ((EPC|\overline{fc}.Cl)\backslash\{sc,fc\}|(WL[f_1]| \\
 & ((D-G)\overline{out}.Dl|L[out/rin])\backslash\{out\}[f_2])\backslash\{ca,cr\}|NBE)\backslash R \\
\xrightarrow{\tau(fc)} & ((EP|Cl)\backslash\{sc,fc\}|(WL[f_1]| \\
 & ((D-G)\overline{out}.Dl|L[out/rin])\backslash\{out\}[f_2])\backslash\{ca,cr\}|NBE)\backslash R \\
\xrightarrow{\tau(rin)} & ((\overline{sc}.EPW|Cl)\backslash\{sc,fc\}|(\underline{aout}.(L_f+L_b)FL[f_1]| \\
 & ((D-G)\overline{out}.Dl|L[out/rin])\backslash\{out\}[f_2])\backslash\{ca,cr\}|NBE)\backslash R \\
\xrightarrow{\tau(sc)} & ((EPW|(C)\overline{fc}.Cl)\backslash\{sc,fc\}|(\underline{aout}.(L_f+L_b)FL[f_1]| \\
 & ((D-G)\overline{out}.Dl|L[out/rin])\backslash\{out\}[f_2])\backslash\{ca,cr\}|NBE)\backslash R \\
\overset{D-G}{\rightsquigarrow} & ((EPW|(C+G-D)\overline{fc}.Cl)\backslash\{sc,fc\}|(\underline{aout}.(L_f+L_b)FL[f_1]| \\
 & (\overline{out}.Dl|L[out/rin])\backslash\{out\}[f_2])\backslash\{ca,cr\}|NBE)\backslash R \\
\xrightarrow{\tau(out)} & ((EPW|(C+G-D)\overline{fc}.Cl)\backslash\{sc,fc\}|(\underline{aout}.(L_f+L_b)FL[f_1]| \\
 & (Dl|(L_f)FL[out/rin])\backslash\{out\}[f_2])\backslash\{ca,cr\}|NBE)\backslash R \\
\overset{L_f}{\rightsquigarrow} & ((EPW|(C+G-D-L_f)\overline{fc}.Cl)\backslash\{sc,fc\}|(\underline{aout}.(L_f+L_b)FL[f_1]| \\
 & (Dl|FL[out/rin])\backslash\{out\}[f_2])\backslash\{ca,cr\}|NBE)\backslash R \\
\xrightarrow{\tau(rout)} & ((EPW|(C+G-D-L_f)\overline{fc}.Cl)\backslash\{sc,fc\}|(\underline{aout}.(L_f+L_b)FL[f_1]| \\
 & (Dl|\overline{ain}.WL[out/rin])\backslash\{out\}[f_2])\backslash\{ca,cr\}|r.\overline{aout}.NBE)\backslash R \\
\xrightarrow{r} & ((EPW|(C+G-D-L_f)\overline{fc}.Cl)\backslash\{sc,fc\}|(\underline{aout}.(L_f+L_b)FL[f_1]| \\
 & (Dl|\overline{ain}.WL[out/rin])\backslash\{out\}[f_2])\backslash\{ca,cr\}|\overline{aout}.NBE)\backslash R \\
\xrightarrow{\tau(ca)} & ((EPW|(C+G-D-L_f)\overline{fc}.Cl)\backslash\{sc,fc\}|((L_f+L_b)FL[f_1]| \\
 & (Dl|WL[out/rin])\backslash\{out\}[f_2])\backslash\{ca,cr\}|\overline{aout}.NBE)\backslash R \\
\xrightarrow{\tau(aout)} & ((EPW|(C+G-D-L_f)\overline{fc}.Cl)\backslash\{sc,fc\}|((L_f+L_b)FL[f_1]| \\
 & (Dl|WLA(L_b)[out/rin])\backslash\{out\}[f_2])\backslash\{ca,cr\}|NBE)\backslash R \\
\overset{L_f+L_b}{\rightsquigarrow} & ((EPW|(C+G-D-2L_f-L_b)\overline{fc}.Cl)\backslash\{sc,fc\}|(FL[f_1]| \\
 & (Dl|L[out/rin])\backslash\{out\}[f_2])\backslash\{ca,cr\}|NBE)\backslash R \\
\xrightarrow{\tau(cr)} & ((EPW|(C+G-D-2L_f-L_b)\overline{fc}.Cl)\backslash\{sc,fc\}|(\overline{ain}.WL[f_1]| \\
 & ((D)\overline{out}.Dl|L[out/rin])\backslash\{out\}[f_2])\backslash\{ca,cr\}|NBE)\backslash R \\
\xrightarrow{\tau(ain)} & ((EPC|(C+G-D-2L_f-L_b)\overline{fc}.Cl)\backslash\{sc,fc\}|(WL[f_1]| \\
 & ((D)\overline{out}.Dl|L[out/rin])\backslash\{out\}[f_2])\backslash\{ca,cr\}|NBE)\backslash R
\end{array}
$$

So if $C+G-D-2L_f-L_b > 0$ the system will not stall, otherwise it will.

As an example consider $L_f = L_b = 6$ and $C = 60$ and $D = 65$. So to avoid stall on 2nd cycle require $2C - D - 3L_f - L_b > 0$ that is $120 - 65 - 18 - 6 = 31$, to avoid stall on next cycle $31 + 60 - 65 - 18 = 8 > 0$, and for the next cycle $8 + 60 - 65 - 18 = -15 < 0$, so with these times stalls after 3 cycles. Consider the above but with $D = 61$ the initial G value is $120 - 61 - 18 - 6 = 35$, next $35 + 60 - 61 - 18 = 16$, next $16 + 60 - 61 - 18 = -3$, so still stalls after 3 cycles.

So unless both C and D are large with respect to the latch times and the difference between them is small there is little benefit from extra buffering.

2.7.2 Handling Forks

In the forward direction we can exploit a simpler method for handling forks, simply replace the forked pipeline section with the slower of the two pipelines assuming the current value of the clocking process.

2.8 Conclusions

We have presented a method for analysing the performance of asynchronous pipelines from the perspective of making as many inputs as possible, and from that of attempting to make inputs at a constant rate. Our main observation is that the pipeline can work at *full* capacity (no stalls) if the downstream pipe elements are sufficiently rapid. In the case of interruptible pipelines (as in execution units) latency can be helped by buffering between fast and slow stages, but in the long term this cannot improve pipeline performance. A further observation is the importance of the parallelism within a Sutherland [Sut89] pipeline stage. Ignoring this parallelism leads to an underestimate of both the occupancy and performance of the pipe system.

Since our approach is compositional it is straightforward to use it for calculating the performance of large systems. In particular we can identify when the system causes stalls. In the case of the backward calculation a stall occurs when the system forces the environment to delay providing data.

In this work we have examined the consequences of introducing forks into the computation pipeline. In the full Amulet processor [Fur93] there are many forms of composition between the units, including some which are in competition. In the longer term it would be interesting to characterise the performance of these more general forms of composition of asynchronous processors. Of great importance in asynchronous systems is the exploitation of average case [Sut89, Fur93] performance, in further work [Tof95] we examine the effect on asynchronous pipeline performance of variability in component execution speed.

3 The effect of variability

A major benefit claimed for asynchronous systems is that they permit designers to concentrate on the average rather than the worst case performance of their circuits [Fur93].

Most analyses [Wil90, SS93, Wil94] of the performance of asynchronous pipelines have concentrated on their deterministic time behaviour. It is assumed that each processing stage takes some fixed time to perform its function, much as in a synchronous circuit. Recent simulation studies [DIY91, KB95] have examined the effects of variability on the performance of pipelines and concluded that whilst the performance of the pipestages will be that of the average case, the performance of the pipeline will not be.

In this Section we aim to obtain a simple analytic account of the performance of an asynchronous pipeline with variable pipe stages. We justify our results by making use of formal process algebraic descriptions of the system in our derivation of its behaviour.

3.1 Introduction to WSCCS

In the remainder of this document we shall use the language WSCCS to describe the behaviours of the systems under consideration. A full introduction to this calculus can be found in [Tof90, Tof94]. The informal interpretation of the WSCCS operators is as follows:

- $\mathbf{0}$ a process which cannot proceed;
- X the process bound to the variable X;
- $a.E$ a process which can perform the action a whereby becoming the process described by E;
- $\sum\{w_i E_i | i \in I\}$ the *weighted* choice between the processes E_i, the weight of the outcome E_i being determined by w_i. We think in terms of repeated experiments on this process and we expect to see over a large number of experiments the process E_i being chosen with a relative frequency of $\frac{w_i}{\Sigma_{i \in I} w_i}$.
- $E \times F$ the synchronous parallel composition of the two processes E and F. At each step each process must perform an action, the composition performing the composition (in Act) of the individual actions;
- $E\lceil A$ represents a process where we only permit actions in the set A. This operator is used to enforce communication and bound the scope of actions;
- $\Theta(E)$ represents taking the prioritised parts of the process E only.
- $E[S]$ represents the process E relabelled by the function S;
- $\mu_i \tilde{x} \tilde{E}$ represents the solution x_i taken from solutions to the mutually recursive equations $\tilde{x} = \tilde{E}$.

The actions a are drawn from an arbitrary free group over a name set. The set of weights $\mathcal{W}$, denoted by w_i, are of the form[2] $n\omega^k$ with n from the positive natural numbers and the ω^k (with $k \geq 0$) a set of infinite objects, with the following multiplication and addition rules (assuming $k \geq k'$):

$$n\omega^k + m\omega^{k'} = n\omega^k = m\omega^{k'} + n\omega^k \qquad n\omega^k + m\omega^k = (n+m)\omega^k = n\omega^k + m\omega^k$$
$$n\omega^k * m\omega^{k'} = (nm)\omega^{k+k'} = m\omega^{k'} * n\omega^k$$

We consider the objects n used as weights to be abbreviations for $n\omega^0$ and ω^k to be an abbreviation for $1\omega^k$.

The analytic methods and tools exploited to obtain performance measures for the systems are described in [Tof95]. The syntax of presentation for some of the examples in the sequel will be that of the tool set described in this paper.

[2] Here n is the relative frequency with which this choice should be taken and k is the priority level of this choice.

3.2 Performance Analysis

We can simplify our analysis of the behaviour of asynchronous buffers, by concentrating on the control in the system, rather than the data values computed upon. In the asynchronous calculus CCS[3] [Mil80, Mil90] a buffer element is described by the process:

$$Buff \stackrel{def}{=} in.\overline{out}.Buff$$

and an n element buffer by the process:

$$B^{(n)} \stackrel{def}{=} (Buff[in_2/out] | Buff[in_2/in, in_3/out] | \ldots | Buff[in_n/in]) \backslash \{in_2, \ldots, in_n\}$$

We can give an analogous definition of a buffer element in SCCS[4] [Mil83], as the following process:

$$Buff \stackrel{def}{=} \$in : \$\overline{out} : Buff$$

however unlike the previous asynchronous definition this description insists that the in and $\overline{out}$ actions[5] are separated in time. That is to say the two events cannot occur simultaneously. Possible a better realisation of a buffer in SCCS would be the following process:

$$\begin{array}{l} Buff \stackrel{def}{=} \$in : FBuff \\ Fbuff \stackrel{def}{=} \$(in\#\overline{out} : FBuff + \overline{out} : Buff) \end{array}$$

Whilst this gives a reasonable encoding of a buffer element, it still does not encompass the fact that these circuits will proceed, perform input and output actions, as soon as their duals become available. In the calculus WSCCS [Tof90, Tof94] we can describe the progress requirements by exploiting the priority constructs:

$$\begin{array}{l} Buff \stackrel{def}{=} \omega.in : FBuff + 1.\surd : Buff \\ Fbuff \stackrel{def}{=} \omega^2.in\#\overline{out} : FBuff + \omega.\overline{out} : Buff + 1.\surd : FBuff \end{array}$$

in this case we insist that the process can perform either an input or an output, or both actions, as soon as the environment will permit it.

Finally we can extend our buffer element above to give durations when the element will processing either quickly or slowly. Initially we assume only 2 different speeds at which the element can function, but in general many could be considered. In this case we assume that elements process quickly with probability p and slowly with probability $(1-p)$, independently of the data and each other.

$$\begin{array}{lcl} Bu & \stackrel{def}{=} & (1-p)\omega.in\#slow : Wbu + p\omega.in\#fast : Fbu + 1.\surd : Bu \\ Wbu & \stackrel{def}{=} & \surd : Fbu \\ Fbu & \stackrel{def}{=} & (1-p)\omega^2.\overline{out}\#in\#slow : Wbu + \omega^2.\overline{out}\#in\#fast : Fbu \\ & & +\omega^1.\overline{out} : Bu + 1.\surd : Fbu \end{array}$$

We can compose buffer elements of the above form together to form a pipeline:

$$Pipe^{(n)} \stackrel{def}{=} \Theta((Bu[in_2/out] \times Bu[in_2/in, in_3/out] \times \ldots \times Bu[in_n/in]) \lceil \{in, out\})$$

In the sequel we shall concentrate on the time taken between in actions by the pipeline. In particular we shall be able to tell when it is going slow by the presence of extra ticks from the buffer element definition above. In which case we can exploit the $fast$ and $slow$ actions to analytically compute the duration of the system.

3.2.1 Upper bound

We can simplify the system by insisting that a buffer element both has new work to do and somewhere to place its result before moving to the next processing task. Clearly this results in elements stalling when otherwise they could continue to compute. As a result we shall obtain an **upper** bound on the time the pipeline will take between performing in actions. If we concentrate on the behaviour when the pipeline elements are full[6], we obtain the following version of a processing element:

[3] For brevity we omit an introduction to the notation of the process algebras used in this paper. We recommend that any reader makes us of [Mil90] for the basic calculi and [Tof94] for the extended probabilistic form.

[4] $\$P \stackrel{def}{=} P + \surd : \$P$ is the asynchronisation operator of SCCS

[5] The action $in\#\overline{out}$ denotes the *simultaneous* performance on an in and an $\overline{out}$ action

[6] As we are *not* using latches this does not cause the same problems as the analysis of Williams [Wil94] would suggest

$$\begin{array}{rcl} Ubuff & \stackrel{def}{=} & (1-p)\omega^2.\overline{out}\#in\#slow : \surd : Ubuff + \omega^2.\overline{out}\#in\#fast : Ubuff \\ & & +1.\surd : Ubuff \end{array}$$

Note, that in the above we have simply prevented it from performing unaccompanied *in* actions, and have added the extra tick caused by a slow action directly into that choice. For simplicity we have not reduced the priority level on the other actions. So forming the above elements into a pipeline we obtain the following:

$$UB^{(n)} \stackrel{def}{=} \Theta((Ubuff[in_2/out] \times Ubuff[in_2/in, in_3/out] \times \ldots \times Ubuff[in_n/in])\lceil\{in, out\})$$

We can demonstrate that the above process is equivalent to the following:

$$\begin{array}{rcl} UB^{(n)} & \stackrel{def}{=} & p^n.in\#fast^n\#\overline{out} : UB^{(n)} \\ & & + \sum_{i=1}^{i=n} p^{(n-i)} q^i \left(\begin{array}{c} i \\ n \end{array} \right) in\#fast^{(n-i)}\#slow^i\#\overline{out} : \surd : UB^{(n)} \end{array}$$

In the above we can clearly see that the pipeline *only* computes quickly when *all* of its elements do so (evidenced by the extra tick on those branches with one or more element processing slowly).

So exploiting the above we see that the probability of the upper bound time being fast $P^u(fast) = p^n$ and similarly the probability of the system being slow $P^u(slow) = 1 - p^n$. We can evaluate the mean upper bound performance as $\mu^u = p^n fast + (1-p^n)slow$ and clearly as $n \rightsquigarrow \infty$ then $\mu^u \rightsquigarrow slow$.

3.2.2 Exact Solution for 3 Pipe Stages

We present the exact solution for a system consisting of 3 pipestages as an illustration of the full behaviour of the system to justify our later abstraction to obtain a lower bound on the system cycle time.

The full system with 3 pipestages is given by the following process definitions:

$$\begin{array}{rcl} B1 & \stackrel{def}{=} & \omega.in\#s1 : B1a + \omega.in\#f1 : B1b + 1.\surd : B1 \\ B1a & \stackrel{def}{=} & 1.\surd : B1b \\ B1b & \stackrel{def}{=} & \omega^2.in\#\overline{in2}\#s1 : B1a + \omega^2.in\#\overline{in2}\#f1 : B1b + \omega.\overline{in2} : B1 + 1.\surd : B1b \\ B2 & \stackrel{def}{=} & \omega.in2\#s2 : B2a + \omega.in2\#f2 : B2b + 1.\surd : B2 \\ B2a & \stackrel{def}{=} & 1.\surd : B2b \\ B2b & \stackrel{def}{=} & \omega^2.in2\#\overline{in3}\#s2 : B1a + \omega^2.in2\#\overline{in3}\#f2 : B1b + \omega.\overline{in3} : B2 + 1.\surd : B2b \\ B3 & \stackrel{def}{=} & \omega.in3\#s3 : B3a + \omega.in3\#f3 : B3b + 1.\surd : B3 \\ B3a & \stackrel{def}{=} & 1.\surd : B3b \\ B3b & \stackrel{def}{=} & \omega^2.in3\#s3 : B1a + \omega^2.in3\#f3 : B3b + \omega.\surd : B3 + 1.\surd : B3 \end{array}$$

In the above process defining each of the buffer stages we have added actions $f1$, $s1, \ldots$ to denote to performance of a fast or slow behaviour. We are interested in the time taken between input actions to the stage and hence ignore the final output, assuming the environment will always be capable of taking it.

Composing the above processes together to form our pipeline we obtain the following process:

$$Sys \stackrel{def}{=} \Theta((B1b \times B2b \times B3b)\lceil\{in, s1, f1, s2, f2, s3, f3\})$$

Minimizing the transition graph for this system we obtain the graph in Figure 6.

Using the graph of Figure 6 we can observe when we have slow or fast times between the performance of an input. The important observation about the above graph is that if the first stage is **slow** and the second and third are **fast**, then the system can 'store' the time advantage. Assuming that the pipestages perform a fast or slow action with probability $p1$, $q1, \ldots$ respectively we can obtain a transition graph in terms of the time taken between input states and the probability of transition as given in Figure 7.

To evaluate the performance of the system in Figure 7 we need to know the probability of being in state 1 or state 2. This amounts to solving the following equations for $P1$ and $P2$ the probability of being in state 1 or 2 respectively:

$$\begin{array}{rcl} P1 & = & p1p2.P2 + p1p2p3.P1 + (1 - p2p3).P1 + q2.P2 \\ P2 & = & q1p2.P2 + q1p2p3.P1 \end{array}$$

We obtain the result:

$$\begin{array}{rcl} P1 & = & \frac{(1-q1p2)}{(1-q1p2q3)} \\ P2 & = & \frac{q1p2p3}{(1-q1p2q3)} \end{array}$$

To obtain the probability of a fast time between inputs (note that the probability of a slow time must by one minus this value). We multiply the probability of being in a state by the probability of performing a fast action.

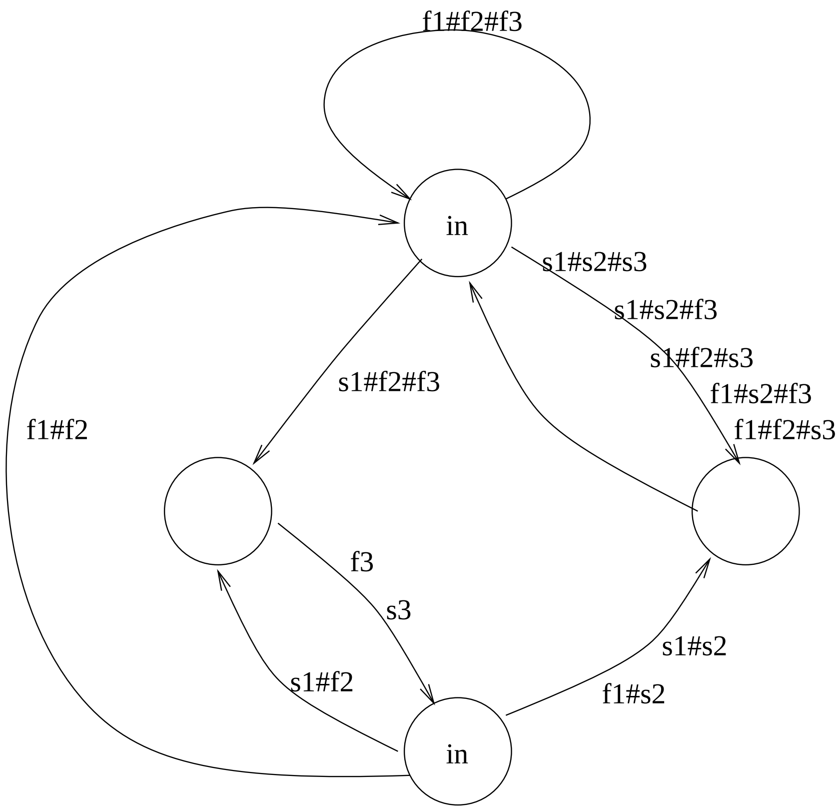

Figure 6: The minimised transition graph for the 3 stage pipeline. For brevity multiple transitions between the same states have their transitions listed alongside the arc. States which perform input actions are labelled in.

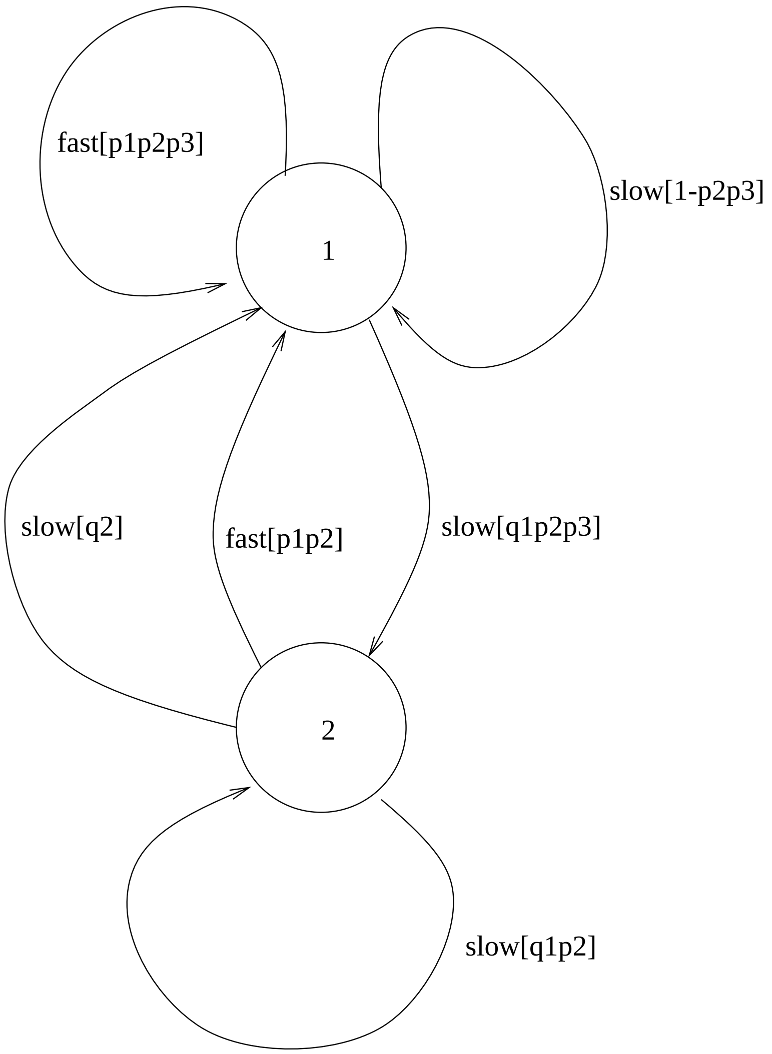

Figure 7: The abstract transition graph between the input states of the 3 stage pipeline. Transitions a labelled with the time between successive inputs and the probability of this transition occuring.

$$
\begin{aligned}
P(fast) &= \frac{(1-q1p2)}{(1-q1p2q3)}p1p2p3 + \frac{q1p2p3}{(1-q1p2q3)}p1p2 \\
&= \frac{p1p2}{(1-q1p2q3)}(p3 - q1p2p3 + q1p2p3) \\
&= \frac{p1p2p3}{(1-q1p2q3)}
\end{aligned}
$$

Let $fast$ and $slow$ represent the time taken in a fast and slow performance respectively then the mean performance of the system is given by:

$$M = \frac{p1p2p3}{(1-q1p2q3)}fast + (1 - \frac{p1p2p3}{(1-q1p2q3)})slow$$

Substuting $pi = \frac{1}{2}$ for all $i \in \{1, 2, 3\}$ and taking $fast = 2$ and $slow = 4$ then the mean time between inputs is $\frac{26}{7}$ or 3.714.

Interpreting the above system. If two of the later pipestages can **both** perform a their work quickly, then the **second** of them can gain time on the system, this time gain being subsequently exploitable to hide a slow time requirement. However, this time advantage is **only** mantained if the stage in front of it continues to perform in a fast time, otherwise the advantage is lost and with it the ability to hide the slow performance of the last stage. In our abstraction to obtain a lower bound, we shall assume that the time advantage can be maintained despite the lack of work for the last stage. We can exploit the technique of the next section to use the results of our analysis of the 3 stage pipeline to give a compositional account of the time taken for a pipeline of odd length. Unfortunately this will generate an upper bound as it only takes account of one half of the locations where a time advantage can occur.

3.2.3 Lower Bound

Consider a two stage system, with the first stage having $P_1(fast) = p$ as the probability of it taking a fast amount of time to perform its calculation, taking $q = 1 - p = P_1(slow)$ Similarly the second stage having $P_2(fast) = p_e$ with $q_e = 1 - p_e$. Further assume that if the second stage can go fast, then it immediately starts work on the next item, that is it can now run ahead of the first stage by the difference between a slow and a fast computation. Clearly this approach will lead to a lower bound, as the next item of work for the second stage will not be ready from the first stage. The second stage cannot gain more than this advantage, since it has to wait for results. If the second stage is advanced and runs slow, and the first stage runs fast, then the result from the first stage can be moved immediately into the second and the system will run fast. Similarly if the first stage runs slow with the second stage advanced then it can stay advanced.

We can describe such a system by the following process:

$$
\begin{aligned}
&P_1 \stackrel{def}{=} pp_e.fast : P_1 + qp_e.slow : P_2 + q_e.slow : P_1 \\
&P_2 \stackrel{def}{=} pp_e.fast : P_2 + pq_e.fast : P_1 + q.slow : P_2
\end{aligned}
$$

In state P_1 above if the second stage can go fast, and the first cannot then it runs ahead of the first stage. If it is running ahead (state P_2) then a fast first stage and a slow second stage results in a fast behaviour, and the stages resynchronising. Similarly a fast first and second stage give a fast behaviour, allowing the second stage to continue running ahead. If the first stage is slow then the second stays ahead of it, but cannot gain more time on the first stage. Since the second stage actually has to wait for work from the first this will give a *lower* bound. The assumption that the second stage can retain its advantage is generous, as it normally would have to wait unless the first stage was fast. The second stage certainly cannot gain more than the gap between a fast and a slow action on the first so we do not need to consider it running any further ahead of the first stage. Essentially we have permitted a higher probability of staying in the advanced state than the system permits, and since this state has the highest possible probability of performing a fast cycle (only requiring the first stage being fast) this must give a lower bound on the systems performance time.

To calculate the probability of observing a fast action from the above system we need to know the probabilities $P(P_1)$ and $P(P_2)$ of being in state P_1 or P_2 independently of time. Since the environment has *no* control over fast or slow actions, this is simply the stable distribution of the above system. In other words:

$$
\begin{aligned}
&qp_eP(P_1) = pq_eP(P_2) \\
&P(P_1) + P(P_2) = 1
\end{aligned}
$$

Solving the above equations we obtain:

$$
\begin{aligned}
&P(P_1) = \frac{pq_e}{pq_e + qp_e} \\
&P(P_2) = \frac{qp_e}{pq_e + qp_e}
\end{aligned}
$$

and hence for our two stage system:

$$
\begin{aligned}
P(fast) &= \frac{pp_epq_e + pqp_e}{pq_e + qp_e} \\
&= \frac{p_e(p^2q_e + pq)}{pq_e + qp_e}
\end{aligned}
$$

Hence, we can construct an upper bound for the probability of a fast behaviour ($P_n(fast)$) from a pipeline of depth n by exploiting the above result iteratively. We start by setting $p_e = p$ and calculating $P_2(fast)$, by setting $p_e = P_n(fast)$ we can calculate $P_{n+1}(fast)$ using the above formula.

Notice that in our formula:

$$P_{n+1}(fast) = \frac{P_n(fast)(p^2(1-P_n(fast))+pq)}{p(1-P_n(fast))+qP_n(fast)}$$

so if we can show that:

$$\frac{p^2(1-P_n(fast))+pq}{p(1-P_n(fast))+qP_n(fast)} < 1$$

then the probability of seeing a fast action from an arbitrarily deep pipe tends to zero. The above inequality holds if

$$\begin{gathered}
p^2(1-P_n(fast)) + p(1-p) < p(1-P_n(fast)) + (1-p)P_n(fast) \\
p^2 - p^2P_n(fast) + p - p^2 < p - pP_n(fast) + P_n(fast) - pP_n(fast) \\
p - p^2P_n(fast) < p - 2pP_n(fast) + P_n(fast) \\
0 < p^2P_n(fast) - 2pP_n(fast) + P_n(fast) \\
0 < p^2 - 2p + 1 \\
0 < (1-p)^2
\end{gathered}$$

So unless $p = 1$ the probability of observing a fast behaviour is *strictly* decreasing with increasing pipeline depth. Hence in the limit the pipeline will be slow.

3.2.4 Comparison

As a comparison we present upper and lower bounds for the performance of a pipeline with: 3, 4, or 5 stages; $fast = 2$, $slow = 4$ and $p = \frac{1}{2}$ we obtain exact solutions to the performance of the pipeline from the process definitions using the tools in [Tof95]. The figures for the bounds are exact but presented as decimals for ease of comparison.

Depth	3	4	5
Mean	3	3	3
Lower	3.39	3.48	3.55
Exact	3.71	3.81	3.87
Upper	3.75	3.875	3.90625

3.3 Conclusions

Superficially it would appear that an asynchronous pipeline is unlikely to deliver average case behaviour and actually delivers worst case. Whilst this may seem to diminish the case for asynchronous systems it should be noted that a synchronous pipeline would have to execute at this (worst case) rate in all cases. Hence, the asynchronous design is at least doing no worse than a clocked design. Our results concur with simular observations in simulo by [KB95].

The processing time of most components is given by a function of the data presented to it. Our analysis has assumed random distributions of processing time independent of the data. If care is taken to ensure that the pipeline has continuous runs of problems taking similar times then the performance of the system can be, to a large extent, restored. Interestingly we have a dual of the normal methods of load balancing in a parallel pipeline. In the case of an asynchronous pipeline the balancing must be achieved dynamically by a suitable ordering of the presentation of problems to the system, if that is possible. The upper bound on performance time (or lower bound on system performance) is relatively easy to calculate, and gives a good approximation in most cases. An anlysis of this form can be extended to take account of the effects of data dependency on pipestage performance, that is a hard problem for one stage is liable to be difficult for another. In a similar fashion upper bounds for pipestages with differing performance parameters are relatively straightforward. Lower bounds are harder to calculate, especially if greater degrees of variability are permitted. As in this case the relative amount of advancement in the system will need to be taken into account. However, even with many different levels of time cost the probability of improving on the worst case behaviour will always be dependent on many pipestages being able to simultaneously perform rapidly and thus our limits will still hold.

One important observation is the closeness (at least for the exact solutions we present) of the upper bound solution to the true solution. Since the upper bound is relatively straightforward to calculate, and does not result in excessive state explosion, it should be possible to exploit this method for gaining performance estimates in more complex scenarios. In particular this method can be applied when the stage delays are data dependent.

4 Variable stages within constant stages

One of the advantages stated for asynchronous logic is that designs can take advantage of expected performance, rather than have to take account of worst-case performance. In a multistage pipeline one of the standard design rules is to match the amount of time that each stage is going to take. With an asynchronous pipeline the stages can take variable amounts of time. We examine some simple examples to identify the effect on performance of the pipeline of this variability in performance.

We use the process algebra WSCCS to describe the pipeline stages. One basic observation is that if a pipestage takes expectedly much longer than all the others then we can expect that time to dominate the execution of the pipe. Similarly if a variable stage is expectedly much faster than other stages then we can expect their time performance to dominate that of the system.

4.1 basic case

In this example we have a three stage buffer with the first and last stages have fixed execution times of 10, and the middle stage having an equal probability of its execution taking either 6,11 or 16 units of time. The mean execution time of this middle pipestage is 11 and we could thus expect to see a pipeline execution time of 11.

The first stage of the pipeline can be described by the following process:

```
bs P1 1.t:P1a1
bs P1a1 1.t:P1a2
bs P1a2 1.t:P1a3
bs P1a3 1.t:P1a4
bs P1a4 1.t:P1a5
bs P1a5 1.t:P1a6
bs P1a6 1.t:P1a7
bs P1a7 1.t:P1a8
bs P1a8 1.t:P1a9
bs P1a9 1@1.in2^-1:P1 + 1.t:P1a9
```

it should be noted that the above pipestage takes 10 units of time and then passes on work to the next pipestage as soon as it can accept it. Thus the priority construct at the last stage ensures that the work proceeds along the pipeline as quickly as possible.

The second variable stage of the pipeline. In this case we shall take either 6, 11 or 16 units of time to perform the necessary work.

```
bs P2 1.in2:P2a1+ 1.t:P2
bs P2a1 1.t:P2a2
bs P2a2 1.t:P2a3
bs P2a3 1.t:P2a4
bs P2a4 2.t:P2a5 + 1.t:P2aout
```

The last state above describes the probability of producing output after 6 ticks, this is 1/3 so that the three times have equal probability.

```
bs P2a5 1.t:P2a6
bs P2a6 1.t:P2a7
bs P2a7 1.t:P2a8
bs P2a8 1.t:P2a9
bs P2a9 1.t:P2a10 + 1.t:P2aout
```

At this point we need a 1/2 choice to give the correct probability of the work taking 11 units of time. (Note $2/3 \times 1/2 = 1/3$).

```
bs P2a10 1.t:P2a11
bs P2a11 1.t:P2a12
bs P2a12 1.t:P2a13
bs P2a13 1.t:P2a14
bs P2a14 1.t:P2aout
bs P2aout 1@1.in3^-1:P2 + 1.t:P2aout
```

Again, the output should be performed as soon as the next stage of the pipeline will accept the work.

The final stage of the pipeline

```
bs P3 1.in3:P3a1+ 1.t:P3
bs P3a1 1.t:P3a2
bs P3a2 1.t:P3a3
bs P3a3 1.t:P3a4
bs P3a4 1.t:P3a5
bs P3a5 1.t:P3a6
bs P3a6 1.t:P3a7
bs P3a7 1.t:P3a8
bs P3a8 1.t:P3a9
bs P3a9 1.done:P3
```

This produces an action `done` which represents the completion of a unit of work by the system.

We can form the complete system with the following two definitions:

```
basi Allow done
btr Sys P1|P2|P3/Allow
```

Solving the above system for its stable distribution we find an effective cycle time of 12.33. This is considerably greater than 11 an must result from the variability of the pipeline stages. In this case we can see the following effects.

If the middle stage performs a 'quick' evaluation then the next stage in the pipeline is unavailable to take the work from it, so it has to wait until a further 4 units of time has gone by, so no time is gained on the short cycles. When the middle stage takes 11 units of time the both the first and last stages have to wait for it. Similarly if it takes 16 units of time. From the above observation we can see that the mean execution time of the pipeline stages should be $\frac{10+11+16}{3} = 12.33$, in agreement with the above calculation.

4.2 Adding a buffer

The above observation would indicate that if we add a buffer between the middle and the third stages then the system could execute faster, as the middle stage would be able to pass on the completed work to the buffer when it was the result of a fast calculation, and then continue executing.

We can define a fast buffer, so that if the data can be passed directly on if the next stage is currently unoccupied:

```
bs IB2 1.ini3#in3^-1:IB2 +1.ini3:IB2a + 1.t:IB2
bs IB2a 1@1.in3^-1:IB2 + 1.t:IB2a

btr Sys1 P1|P2[ini3/in3]|IB2|P3/Allow
```

This still gives an effective cycle time of 12.33, that is the same time as that observed when the extra buffer stage is missing.

The reason for this is whilst the middle stage can pass on the results of a fast computation to the buffer, there is **no** work for it available on the first stage, so it has to wait.

Similarly, if we place a single buffer between the first pipe stage and the rest of the system we observe the same effective cycle time.

4.3 Double buffering

As we have observed that adding a single buffer on either side of a variable pipestage gives no advantage, we consider the effect of adding a buffer on both sides of the middle stage. This effect can be produced by the following process:

```
btr Sys1 P1[ini2/in2]|IB2[ini2/ini3,in2/in3]|P2[ini3/in3]|IB2|P3/Allow
```

Which has an effective cycle time of 11.25 or only 0.25 worse than could be reasonably expected from the mean execution time of the middle pipe stage. When two buffer stages are used each side of the variable stage we observe the cycle time dropping to 11.08,

4.4 Load stalls

Often the first stage of a pipeline will have a variable performance cause by variation in the time needed to get values from memory. This can be represented by using the following description of the first state of the pipeline:

```
bs P1 1.t:P1a1 + 1.t:P1
```

When we give the first stage a geometrically distributed ($p = 1/2$) probability of requiring an extra time unit to fetch data, we find that the base cycle time (no buffers) is 12.833, which is maintained if a buffer is added between stages 2 and 3. If a buffer is added between stages 1 and 2 then the expected cycle time drops to 12.383. If buffers are placed either side of stage 2 then the cycle time drops to 11.64. With double buffering either side of the variable stage then the time falls to 11.38.

4.5 Conclusions

It is clear that the expected cycle time of a sequence of variable pipeline stages is **NOT** given by the maximum of the expected execution times of the individual pipestages. For reasonable distributions when the execution time of buffers is small compared to that for pipestages the addition of buffers on **both** sides of variable stages will reduce the expected cycle time close to that of the maximum of the expected times. Increasing the depth of the buffers beyond 1 does not give much extra gain, and is unlikely to be of benefit when the time of execution of the buffers is taken into account.

Similar analyses have been undertaken with the middle stage having a geometrically distributed variation in the time of execution. The results are structurally those obtained above. If all the other pipeline stages have fixed execution times then buffers have to added to either side of the variable stage in order to obtain any advantage from short cycles.

5 Flushing

In any execution pipeline system of a processor a branch point in a program will cause all of the currently executing instructions to become invalid and require that the pipeline be flushed. In a synchronous system there is an obvious point at which this can be achieved but in an asynchronous system it is less clear when the flush can be executed. One method adopted is that of colour [Fur93].

In this paper we compare the effectiveness of 4 different types of flushing:

1. Colour (AMULET),
2. Global (Synchronous?),
3. Forward,
4. Backward.

In each case we present the description of a 3 stage pipeline with the appropriate mechanism. Where possible we shall derive a symbolic representation of the performance of the system.

In the sequel the parameter k will express the sequentiality level of a program, and the parameter m will express the distribution of execution time for a single pipestage.

5.1 Colour

In this case the flush control is performed by the association of a colour bit with each instruction, these result of an instruction is taken as valid if it is decorated with the same colour bit as the current execution.

We can describe the system by the following collection of processes.

This first stage generates instructions that are labelled as black or white. When it receives a flush instruction it then changes the colour of the instructions it is producing.

```
bs Srw 1@1.wout0^-1:Srw + 1.t:Srw + 1.flush:Srb
bs Srb 1@1.bout0^-1:Srb + 1.t:Srb + 1.flush:Srw
```

This is the first pipe stage, it inputs an instruction with its associated colour, and then spends some time processing it.

```
bs P0 1.wout0:P0w + 1.bout0:P0b + 1.t:P0
bs P0w m.t:Pwout + n.t:P0w
bs P0b m.t:Pbout + n.t:P0b
bs Pwout 1@1.wout1^-1:P0 + 1.t:Pwout
bs Pbout 1@1.bout1^-1:P0 + 1.t:Pbout
```

The last two states cause the stage to output its current value as soon as the next pipestage can take the result.

The next stage of the pipe is identical up to the renaming of the input and output actions:

```
bs P1 1.wout1:P1w + 1.bout1:P1b + 1.t:P1
bs P1w m.t:P1wout + n.t:P1w
bs P1b m.t:P1bout + n.t:P1b
bs P1wout 1@1.wout2^-1:P1 + 1.t:P1wout
bs P1bout 1@1.bout2^-1:P1 + 1.t:P1bout
```

Similarly for the final stage of the pipe:

```
bs P2 1.wout2:P2w + 1.bout2:P2b + 1.t:P2
bs P2w m.t:P2wout + n.t:P2w
bs P2b m.t:P2bout + n.t:P2b
bs P2wout 1@2.wout^-1:P2 + 1.t:P2wout
bs P2bout 1@2.bout^-1:P2 + 1.t:P2bout
```

Finally we need an absorber for the work. This process checks that items are of the correct colour to be considered as work. We can only issue a flush when work is presented. After a flush is issued we must see at least one item of completed work before we can issue another.

```
bs Abs l.wout#done:Abs + k.wout#done#flush^-1:Absw +
       l.bout#done:Abs + k.bout#done#flush^-1:Absb + 1.t:Abs
bs Absw 1.wout:Absw + l.bout#done:Abs +k.bout#done#flush^-1:Absb +1.t:Absw
bs Absb 1.bout:Absb + l.wout#done:Abs +k.wout#done#flush^-1:Absw +1.t:Absb
```

Note in the above that items labelled with the incorrect colour are simply ignored. Also after a flush the colour of what items are considered to be work is changed.

We can put the complete pipeline system together, retaining an action that permits us to observe the correct completion of an items progress down the pipeline:

```
basi Allow done
btr Srw|P0|P1|P2|Abs/Allow
```

One observation we can make on this system is that it is deadlock and livelock free and therefore at this that aspect of its behaviour is correct.

The performance of the above for a 1 deep pipeline is:

$$\frac{m}{1+2m}$$

which is independent of k.

Solving the above when the pipeline is two stages deep we obtain the following formula for the systems performance:

$$\frac{2m(2-m)}{5+9k+5m+2km-4m^2-4km^2}$$

5.2 Global

In this case the absorbtion process sends a flush instruction to all of the proceeding pipestages simultaneously. Obviously this should generate the most efficient system, against which all the other could be compared.

```
bs P0 1.t:P0i +1.t:P0+1.flush:P0
bs P0i 1-m.t:P0i+m.t:P0o+1.flush:P0
bs P0o 1@1.in1^-1:P0+1.t:P0o+1.flush:P0
```

note that it always responds to a flush instruction. If we wish to associate a time cost with flushing a pipeline stage it could be added by placing a new state in to represent the cost instead of the P0 after the flushes.

The next stage of the pipeline accepts data and flushes from the preceding stage and passes them on to the next pipeline stage.

```
bs P1 1.in1:P1i +1.t:P1+1.flush:P1
bs P1i 1-m.t:P1i+m.t:P1o+1.flush:P1
bs P1o 1@1.in2^-1:P1+1.t:P1o+1.flush:P1
```

The final pipeline stage is the same as the second up to action renaming:

```
bs P2 1.in2:P2i +1.t:P2+1.flush:P2
bs P2i 1-m.t:P2i+m.t:P2o+1.flush:P2
bs P2o 1@1.in3^-1:P2+1.t:P2o+1.flush:P2
```

We need a final process that observes the completion of work and generates the flushes for the system:

```
bs Abs k.in3#done:Abs + 1-k.in3#done:Absf + 1.t:Abs
bs Absf 3@1.flush2^-3:Abs
```

In this case after a flush is issued we are guaranteed that the next item from the pipe must be a correct piece of work. Notice that we give the flush a high enough priority to override the continued transfer of data.

Finally we put the complete system together:

```
basi Allow done
btr Sys P0|P1|P2|Abs/Allow
```

For a single stage pipeline:

$$\frac{1-m}{6-2k-5m+2km}$$

For a 2 deep pipeline we obtain the following performance:

$$\frac{(m-1)(2+m-m^2)}{16-8k-4m+3km-14m^2+7km^2+6m^3-3km^3}$$

5.3 Forward

In this system the absorber generates a flush signal which is then passed along the pipeline system.

The first stage of the pipeline, creates input for itself

```
bs P0 1.t:P0i +1.t:P0+1.flush0:P0f
bs P0i 1-m.t:P0i+m.t:P0o+1.flush0:P0f
bs P0o 1@1.in1^-1:P0+1.t:P0o+1.flush0:P0f
```

note that it always responds to a flush instruction. After receiving the flush it is passed to the next pipeline stage.

```
bs P0f 1.flush1^-1:P0
```

if we wish to associate a time cost with flushing a pipeline stage it could be added at this point. Note we would have to give this a higher priority than passing on data has from the next stage to override that behaviour.

The next stage of the pipeline accepts data and flushes from the preceding stage and passes them on to the next pipeline stage.

```
bs P1 1.in1:P1i +1.t:P1+1.flush1:P1f
bs P1i 1-m.t:P1i+m.t:P1o+1.flush1:P1f
bs P1o 1@1.in2^-1:P1+1.t:P1o+1.flush1:P1f
bs P1f 1.flush2^-1:P1
```

The final pipeline stage is the same as the second up to action renaming:

```
bs P2 1.in2:P2i +1.t:P2+1.flush2:P2f
bs P2i 1-m.t:P2i+m.t:P2o+1.flush2:P2f
bs P2o 1@1.in3^-1:P2+1.t:P2o+1.flush2:P2f
bs P2f 1.flush3^-1:P2
```

We need a final process that observes the completion of work and generates the flushes for the system:

```
bs Abs k.in3#done:Abs + 1-k.in3#done:Absf + 1.t:Abs
```

When a flush is generated the absorber must wait until the flush signal from the last pipestage arrives which represents the correct flushing of the entire pipeline:

```
bs Absf 1.flush0^-1:Absf1
bs Absf1 1.flush3:Abs+1.t:Absf1
```

Notice when a flush is generated it must be given a higher priority than the highest possible for the transfer of information within the underlying pipe system (in this case 2, since stage 3 has just output).

Finally we put the complete system together:

```
basi Allow done
btr Sys P0|P1|P2|Abs/Allow
```

we can show that this system is deadlock and livelock free.

For a single stage pipeline:

$$\frac{1-m}{6-2k-5m+2km}$$

For a 2 deep pipeline this gives us a performance of:

$$\frac{(m-1)(2+m-m^2)}{16-8k-4m+3km-14m^2+7km^2+6m^3-3km^3}$$

5.4 Backward

In this system a flush is generated at the end of the pipeline and then propagates back to the head of the pipeline clearing out pipestages behind it.

```
bs P0 1.t:P0i +1.t:P0+1.flush0:P0
bs P0i 1-m.t:P0i+m.t:P0o+1.flush0:P0
bs P0o 1@1.in1^-1:P0+1.t:P0o+1.flush0:P0
```

note that it always responds to a flush instruction. If we wish to associate a time cost with flushing a pipeline stage it could be added by placing a new state in to represent the cost instead of the P0 after the flushes.

The next stage of the pipeline accepts data and flushes from the preceding stage and passes them on to the next pipeline stage.

```
bs P1 1.in1:P1i +1.t:P1+1.flush1:P1f
bs P1i 1-m.t:P1i+m.t:P1o+1.flush1:P1f
bs P1o 1@1.in2^-1:P1+1.t:P1o+1.flush1:P1f
bs P1f 1.flush0^-1:P1
```

If we put a cost on flushing we would need to add a priority to the flush action in `P1f` above.

The final pipeline stage is the same as the second up to action renaming:

```
bs P2 1.in2:P2i +1.t:P2+1.flush2:P2f
bs P2i 1-m.t:P2i+m.t:P2o+1.flush2:P2f
bs P2o 1@1.in3^-1:P2+1.t:P2o+1.flush2:P2f
bs P2f 1.flush1^-1:P2
```

We need a final process that observes the completion of work and generates the flushes for the system:

```
bs Abs k.in3#done:Abs + 1-k.in3#done:Absf + 1.t:Abs
bs Absf 1.flush2^-1:Abs
```

in this case after a flush is issued we are guaranteed that the next item from the pipe must be a correct piece of work.

Finally we put the complete system together:

```
basi Allow done
btr Sys P0|P1|P2|Abs/Allow
```

For a single stage pipeline:

$$\frac{1-m}{6-2k-5m+2km}$$

For a 2 deep pipeline we obtain the following solution:

$$\frac{(1-m)(2+m-m^2)}{18-10k-5m+4km-16m^2+9km^2+7m^3-4km^3}$$

5.5 Numerical Comparison

When there are greater then 2 components in the pipeline it is infeasible to solve the resulting systems exactly. However, it is possible to derive *exact* numerical solutions to the execution rates of the various systems. We assume that on average 4 instructions can be executed before a flush occurs, and compare the systems when the total time each object spends being worker on is 60 and 120. That is in a 3 unit pipeline each stage expectedly takes 20 times units, 4 deep taking 15, and 5 deep 12.

For the colour system:

	3 Stages	4 Stages	5 Stages
60	50.42	46.11	44.43
120	98.29	89.84	88.34

For global flushing:

	3 Stages	4 Stages	5 Stages
60	43.67	39.0	35.9
120	84.02	75.0	68.71

For forward flushing:

	3 Stages	4 Stages	5 Stages
60	43.67	39.03	35.90
120	84.41	75.09	68.71

Note that as before the forward and global flushing systems are identical.
For backward flushing:

	3 Stages	4 Stages	5 Stages
60	44.00	39.60	36.70
120	98.29	75.60	68.45

5.6 Flushing delays

There are two places where a delay in flushing can occur; either before or after the flushing signal is passed on. Obviously this makes no difference to the global flushing system. Equally the colour system does not exploit flushing signals. For comparison we present examples when the delay occurs before the flush signal can be passed on for a 3 deep pipeline system:

Basic time	Global	Forward	Backward	Colour
60	48.59	52.80	58.16	50.42
120	94.64	103.64	113.89	94.40

Since we cannot expect systems where the flush delay is incurred after the flush signal is passed on to perform more quickly then the global flush system we leave these unexplored for the moment.

As it is know that the mean performance of the pipeline stages is a poor predictor for the performance of the pipeline when the pipestage have a variable delay we include the figures for comparison. We expect the pipeline to take 37.40 time units per cycle. Which gives us a better figure for comparison when flushing is involved.

6 Data dependency

In the analyses undertaken so far we have assumed that any variation in the time cost of a processing unit was independent of the data it was processing. Clearly this assumption is unrealistic. One problem with the analysis of data dependent delays is that it is fundamentally related to the underlying construction of the system, and hence hard to perform as a generic study. As an example from which others could be derived we present a performance analysis of a stage pipeline using the upper bound technique of Section 3

In the syntax of our tool [Tof95] the pipeline can be described as follows:

```
bs Baf3  1@2.inf2^-1#ins#slow:Bas3 + 1@2.inf2^-1#inf#fast:Baf3 + 1.t:Baf3
bs Bas3  1@2.ins2^-1#ins#slow:Bas3 + 1@2.ins2^-1#inf#fast:Baf3 + 1.t:Bas3


bs Bbf3  1@2.inf3^-1#ins2#slow:Bbs3 + 1@2.inf3^-1#inf2#fast:Bbf3 + 1.t:Bbf3
bs Bbs3  1@2.ins3^-1#ins2#slow:Bbs3 + 1@2.ins3^-1#inf2#fast:Bbf3 + 1.t:Bbs3

bs Bcf3  1@2.ins3#slow:Bcs3 + 1@2.inf3#fast:Bcf3 + 1.t:Bcf3
bs Bcs3  1@2.ins3#slow:Bcs3 + 1@2.inf3#fast:Bcf3 + 1.t:Bcs3


basi P inf,ins,fast,slow

btr Sys Baf3|Bbf3|Bcf3/P
```

In the above we have two states to describe each stage, one being when it was processing a fast datum and the other for a slow datum. We are interested in the rate at which instructions can be taken, so we assume that the downstream processing can never block. We initialise the stages all to fast, as this gives an equivalent performance to assuming that the later stages are empty after flushing after a branch.

From the cost model of 3 we can build a filter to report whether the observed processing will be fast or slow (denoted f or s), f will only be possible if all stages can process fast.

```
bs Filter 1.fast^-3#f:Filter + 1.fast^-2#slow^-1#s:Filter +
          1.fast^-1#slow^-2#s:Filter + 1.slow^-3#s:Filter

basi P1 inf,ins,s,f

btr AS Sys|Filter/P1
```

At this point we have a process which will report the performance of the pipeline when presented with a sequence of instructions.

```
bs I1 1.ins^-1:I2
bs I2 1.inf^-1:I3
bs I3 1.inf^-1:I4
bs I4 1.ins^-1:I5
bs I5 1.inf^-1:I6
bs I6 1.inf^-1:I6

basi P2 s,f

btr S1 I1|AS/P2

bs B1 1.inf^-1:B2
bs B2 1.inf^-1:B3
bs B3 1.inf^-1:B4
bs B4 1.ins^-1:B5
bs B5 1.ins^-1:B6
bs B6 1.inf^-1:B6

btr S2 B1|AS/P2
```

The systems S1 and S2 will output a sequence of actions recording the cost of two instruction sequences $sffsf$ and $fffss$. Note, the loop at the end allows us to see the final instructions clearing the pipeline. Since there is a latency of 2 in the processing we need to observe 7 actions to compute the cost of processing the 5 instructions.

The resulting machines for S1 and S2 are given by:

```
bs S1      1.s^1:S3072
bs S3072   1.s^1:S5120
bs S5120   1.s^1:S7168
bs S7168   1.s^1:S9216
bs S9216   1.s^1:S11264
bs S11264  1.s^1:S13312
bs S13312  1.f^1:S15360
bs S15360  1.f^1:S15360

bs S2      1.f^1:S31744
bs S31744  1.f^1:S33792
bs S33792  1.f^1:S35840
bs S35840  1.s^1:S37888
bs S37888  1.s^1:S39936
bs S39936  1.s^1:S41984
bs S41984  1.s^1:S44032
bs S44032  1.f^1:S46080
bs S46080  1.f^1:S46080
```

Notice that in the case of S1 the cost to completion is 6s + 1f and in the case of S2 the cost is 4s+3f. So if the instruction sequences could be reordered, which is often the case for compiled code, then a considerable saving could be made.

6.1 Conclusions

The analysis of data-dependent systems requires specific knowledge of the data patterns impinging on any particular architecture and the response of its components. The above analysis indicates that it is possible to effectively formally model the effect of data, but that such models a liable to be highly specific. Work is needed on abstracting the nature of time/space patterns in the data presented to processors and other pipeline structures in order to make such analyses more general.

Using the above pipeline model or one with explicit flushing, but a more complex model of the presented data it would be possible to analyse many of the effects of data variability, but we leave that to a later document.

7 Summary of Results

For ease of reference we represent a summary of the main results contained in this presentation.

- The parallelism between the computation components and the latching components of a Sutherland pipeline can be exploited to improve performance.
- Extra buffering of stages can give a few extra cycles with the shortest possible cycle time, but the system will eventually take the limiting cycle time.
- Simple forks in the pipeline can be modelled effectively.
- The variability of the environment can be calculated with.
- Despite the difficulty in formally justifying the above results we obtain relativel simple conditional formulae for the systems performance.
- Whilst stages can be designed to work optimally for the 'average' case. A systems performance will still tend to be dominated by the worst case performance, and therefore it cannot be neglected.
- Asynchronous pipeline flushing techniques can achieve performance similar to that achievable in a synchronous setting.
- Attempting to hide the variability of a single pipestage with a single buffer in the upstream or downstream position will have no performance benefits. The stage must be pre and postbuffered.
- Performance of asynchronous pipelines with variable components may be enhanced by manipulating the order in which operands and data are presented to them.

8 Conclusions and further work

We have presented a wide range of analyses of inline pipe asynchronous pipeline arrangements and demonstrated that these can be successfully studied in an abstract formal setting.

However, we have presented no analysis of competition for resources or components. In the design of real processors, such as Amulet [Fur93], particular subcomponents are often used by more than one part of the processor, to save on area. An analysis of the effects of such contention would be highly complex. In the main such effects are usually studied by simulation, so it would be an interesting challenge to obtain a formal description of them.

9 Acknowledgements

I would like to thank Nigel Paver for many patient discussions of the temporal behaviour of Latches and the Amulet pipeline in general. Doug Edwards for coping with some of the early proof attempts. Graham Birtwistle and Faren Moller for many useful discussions on the problems of representing asynchronous hardware, and Mel for making it worthwhile.

References

[DIY91] Y. Deguchi, N. Ishiura and S. Yajima, Probabilistic CTSS, analysis of timing error probability in asynchronous logic circuits, 28th ACM/IEEE Design Automation Conference, pp 650-655, 1991.

[Fur93] S.B. Furber, Breaking step - the return of asynchronous logic, IEE review, July 159-162, 1993.

[KB95] D.A. Kearney and N.W. Bergmann, Performance evaluation of asynchronous logic pipelines with data dependant delays, 2nd Working Conference om Asynchronous Design Methodologies, 1995.

[Mil80] R. Milner, Calculus of Communicating System, LNCS92, 1980.

[Mil83] R. Milner, Calculi for Synchrony and Asynchrony, Theoretical Computer Science 25(3), pp 267-310, 1983.

[Mil90] R. Milner, Communication and Concurrency, Prentice Hall, 1990.

[MT90] Moller, F. and C. Tofts, *A Temporal Calculus of Communicating Systems*, Proceedings of CONCUR'90, Lecture Notes in Computer Science 458, Amsterdam, August 1990.

[MT91] Moller, F. and C. Tofts, *Relating Processes with Respect to Speed*, Proceedings of CONCUR'91, Lecture Notes in Computer Science 527, (Theories of Concurrency: Unification and Extension), Amsterdam, August 1991.

[MT92] Moller, F. and C. Tofts, *Behavioural Abstraction in TCCS* Proceedings of ICALP'92, Lecture Notes in Computer Science, Vienna, July 1992.

[MT93] Moller, F. and C. Tofts, *Reasoning about Resource Bounded Computations in Real Timed Process Calculi* Proceedings of SofSem, 1993.

[RH80] C.V. Ramamoorthy and G.S. Ho, Performance evaluation of asynchronous concurrent systems using Petri nets, IEEE Trans. Software Engineering, SE 6(5):440-449, 1980

[SS93] J. Sparso and J. Staunstrup, Design and performance analysis of delay insensitive multi-ring structures, IEEE 26th Hawaii Int. Conference on System Science, pp349-358, 1993

[Sut89] I. Sutherland, Micropipelines, CACM 32(6):720-736, 1989.

[Tof90] C. Tofts, A Synchronous Calculus of Relative Frequency, CONCUR '90, Springer Verlag, LNCS 458.

[Tof94] C. Tofts, Processes with Probabilities, Priorities and Time, FACS 6(5): 536-564, 1994.

[Tof95] C. Tofts, Analytic and locally approximate solutions to properties of probabilistic processes, Department of Computer Science report, University of Manchester, UMCS-95-3-1, 1995, also LNCS 1019.

[Tof95a] C. Tofts, A Compositional Analsysis of the Performance of Asynchronous Micropipelines proceedings ACM TAU'95, 1995.

[Tof95b] C. Tofts, The illusion of average case performance in asynchronous pipelines, submitted IEEE CAD, 1995.

[Wil90] T. E. Williams, Latency and throughput tradeoffs in self-timed asynchronous pipelines and rings, Stanford Technical Report CSL-TR-90-431, 1990.

[Wil94] T. E. Williams, Performance of iterative computation in self-timed rings, Journal of VLSI Signal Processing, 1994.