Performance Analysis of Asynchronous Networks

by

Robert Berks

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 1998

0-612-32813-9

Abstract

Asynchronous networks present unique problems in timing analysis. Each component of an asynchronous network may have a delay different from other components, and moreover, the delay within a component may vary from communication to communication. Because the components synchronise with each other locally, the differences in delay can lead to blocking when a component tries to communicate data to another component that is not ready for new data.

We give an analytical technique for finding bounds on the speed of asynchronous networks of components. We give results as closed-form formulae for the worst-case response time and average response time. Response time is the time between a request made to the network and an acknowledging response made by the network. We also give results for the worst-case cycle time and average cycle time. Cycle time is the time between consecutive requests made to the network. In particular, we give bounds for linear arrays of cells and for tree architectures. We give bounds on average cycle time for more general networks.

We use parameters to describe the handshaking behaviour of components, the size of the network, and the delay bounds of each component of the network. These parameters may be instantiated with numeric values to obtain specific results. Closed-form formulae give us insight into design trade-offs and optimisation of asynchronous architectures. We derive formulae by means of proofs, not simulation.

Acknowledgements

This thesis would not have been possible without the patient support of my supervisor Jo Ebergen who has worked above and beyond the call of duty. I also thank the hard work of my thesis committee: John Brzozowski, Prabhakar Ragde, Graham Birtwistle and Bill Cunningham. Ed Carr, Jim Geelen and Erik Demaine helped me greatly with mathematical details. Radu Negulescu has always been there to point me toward interesting areas of research. Mark Greenstreet and Steve Nowick offered enthusiastic advice and suggestions. Ivan Sutherland made suggestions that eventually led to Chapter 6.

I thank the members of the Maveric group: Chuan-Jin (Richard) Shi, Maitham Shams, Igor Benko, Rob Black, John Segers, Piotr Sidorowicz, Kamran Raahemifar, Signe Silver, Jimmy Ning, Hao (Richard) Zhang for their patience and humour.

I am grateful for the financial support of Sun Microsystems, the William Georgetti Scholarship, Ontario Graduate Scholarship, the Institute for Computer Research at the University of Waterloo, and the University of Waterloo itself.

I am grateful to all those above who directly helped me with my work, and moreover, I am grateful for their friendship and support. My stay in beautiful Waterloo could not have been as pleasant without these fine people. Nor could life here have been as pleasant without Thomas, Wolfgang, Rob K., Rob L. Patrick, Mike, Biraj, Peter H., Ben, Keith, Andrew, Julie, Peter V., Lori, Amir, Kamran. Matthew, Steve M., Kurt, Daniel, Toby, Steve H., Michael, Marcus, Jason A., Dan, Hristo, Jonathon, Rob Z., Giovanni, Cam M., Coralee, Tali, Cam S., Jun, Marc, Stuart, Jason S., Dave, Andrea, Shannon, and Franklin. I thank you all.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 The Problem

When we design a circuit there are several criteria that might be important to us. We are often concerned with the speed of a circuit, though we might also be concerned with area, energy consumption, or the cost of circuit fabrication. Most designers have a target speed in mind before they even get to the circuit design. How does the designer know that this performance target is reachable? Perhaps the designer has to implement a few prototypes to get an idea of what speed is attainable. This costs both time and money. Moreover, it may be impossible to create accurate enough prototypes without actually implementing the whole circuit. So, how then do we experiment with various design ideas without having to go to the expense of implementing all of them?

The earlier we can make predictions about the performance of a design, the sooner we can narrow our focus to designs that are worth implementing. Furthermore, if we understand the tradeoffs between the structure of a design and its speed, we will be able to converge on a good design more quickly. In this thesis, we model an asynchronous circuit by means of a network of components that communicate asynchronously. Examples of such networks are linear arrays of cells, and trees of cells. We prove bounds analytically on the delays of these specific networks of components. We can use the high-level timing analysis of a network to predict the timing properties of a circuit implementation of the network. Moreover, we give results as closed-form formulae, and these give us insight into design trade-offs.

Historically, most researchers have focussed their attention on synchronous circuits, but now there is a burgeoning demand for timing analysis of asynchronous circuits, hence this thesis. It is easiest to describe an asynchronous circuit by describing what it is not. A *synchronous* circuit uses the regular pulses of a single global clock to synchronise the communications of components. Each subcircuit in a synchronous circuit must complete its operation before the next clock pulse. Throughput of the overall circuit can be derived from the clock frequency. This typically means that the higher the clock frequency is the faster the circuit is.

Asynchronous circuits have no global clock, and components typically use handshaking to synchronise with each other. Circuit components that wish to engage in a transaction with each other communicate locally to decide when the components are ready for the transaction. Asynchronous circuits have several potential advantages.

- Low power consumption. The commonest circuit technology in use today is Complementary Metal Oxide Silicon (CMOS). Almost all energy consumed by a CMOS circuit occurs when a signal on a wire changes. When a wire is at a high or low level, negligible energy is consumed, and hence, if any part of an asynchronous circuit is idle, negligible energy is consumed in that part. In contrast, a synchronous circuit constantly charges and discharges the clock-wires that are distributed over the entire chip. These clock wires may be switching in parts of the circuit that are idle, thereby wasting energy.

- Modular design. In theory, we can build a circuit component by component and, since timing issues are local in an asynchronous circuit, we can easily to join the components into a composite that will function correctly. Synchronous circuits may require retiming of the entire system before a component may be added, since the clock is distributed globally.

- High speed. A component need not wait for a clock pulse to arrive before processing new data. A circuit may run as fast as it can and, in theory, may maintain average-case performance.

The last claim interests us most. Consider an asynchronous circuit with two components A and B. Suppose component A and component B are linked in series. A transfer of data from A to B can only happen if A has data, and B is empty. In theory, when each component finishes a calculation, it may then proceed to process a new piece of data immediately. In contrast, a component A of a clocked circuit must always wait for the clock pulse before processing new data, even if it has finished processing its current data. This restricts a synchronous circuit to the worst-case speed of the slowest component, as the clock pulse has to arrive after the last possible completion of all components.

The above model of the speed of an asynchronous circuit is too simplistic, however. A component cannot always process new data immediately, because there is no guarantee that new data is available at the component's inputs, i.e., the previous component may not be ready with new data. This phenomenon has been called *starvation* by some [31, 44]. Likewise, a component may not be able to transmit its results, because the receiving component is not yet ready. Despite finishing its calculation, a component may not be able to get rid of its data. This phenomenon is typically called *blocking*. See Figure 1.1. Asynchronous circuits cannot, therefore, achieve average-case speed without considering blocking and starvation.

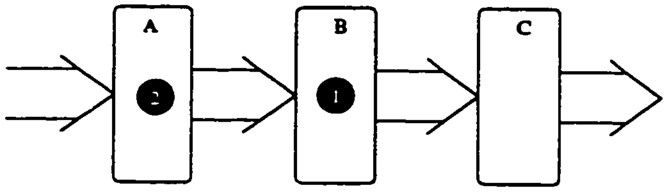


Figure 1.1: An example of an asynchronous circuit where blocking and starvation can occur. Data is represented by the black spots, 1 and 2. Component C starves as it has no data. Component A finishes processing data token 2, but then blocks, because data token 1 still occupies component B.

Any analysis of asynchronous circuits faces further obstacles. The variation in delays of a component are crucial to the performance of an asynchronous system, whereas in a synchronous system we are typically concerned with worst-case delays only. There are many factors that can cause variation in the delay of a circuit. For example,

- Circuit operating temperature can affect the delay of the transistors. Some transistors may switch faster depending on whether an input rises or falls.
- A component's delay may be data-dependent. Some inputs may be easier and quicker to process than others. For example, consider a circuit that multiplies two operands together. If one operand is zero, the computation is trivial, and we may be able to design a circuit that deals with this case more quickly than

the worst-case. There are many examples in the literature of circuits that take advantage of "easy" cases to improve their average speed [21, 23, 28, 30, 34, 40, 50]. These data-dependent designs have varying delay depending on the input.

Readers further interested in asynchronous circuits can pursue introductions to them in the references [9, 15].

Not only is it harder to analyse an asynchronous circuit compared to a synchronous circuit, but asynchronous timing analysis has been less thoroughly researched than synchronous timing analysis. This thesis is a contribution to the field of asynchronous timing analysis.

1.2 An Example Circuit

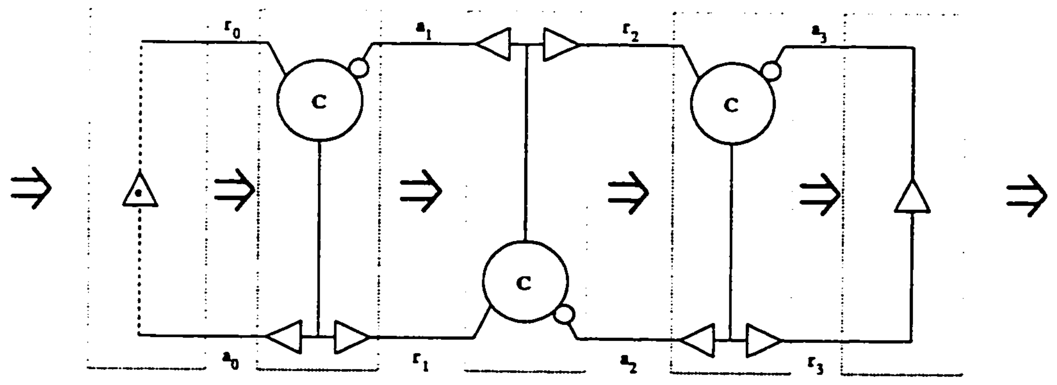


Figure 1.2: The control circuit of a micropipeline with individual cells outlined.

Consider the asynchronous circuit of Figure 1.2. The diagram illustrates the control circuit of a five-stage micropipeline. The detailed operation of the circuit and the behaviour of the C-elements do not concern us at this point. There is a brief description of C-element behaviour in Appendix C. Sutherland describes micropipelines in [47] and we give some further explanation in the next chapter. Data enters from the left and exits to the right and moves from stage to stage as illustrated by the arrows. The control wires are represented by requests r_i and acknowledgments a_i where $0 \leq i \leq 3$. When a stage has data and wishes to pass data to the next stage, it "handshakes" with the next stage. For example, a stage communicates that it is ready to send data onwards by signalling to the next stage on a request wire. When the next stage is ready to receive the data it signals back to the previous stage on an acknowledgment wire. This is a 2-phase handshake. We will describe the circuit more fully in Chapter 2.

Given the possible range of delays of each cell of the micropipeline we might try to determine the throughput of the pipeline. That is, how fast per item can we put a stream of data in one side, process the data in the pipeline, and then retrieve the data from the other? There are many difficulties in determining the throughput in a system such as this. Data must travel through each stage, and each stage may have a different delay from every other stage. Each stage may have delays that vary with each data item. Moreover, if there are many pieces of data in the pipeline, the circuit may block as in Figure 1.1.

As a second problem, we might wish to compare several pipeline designs. How do we compare the speed of these pipelines if they differ in length? Furthermore, the pipelines may differ in the delays of each stage; one pipeline may have a slow first stage, another might have stages with identical speeds, another might have stages with data-dependent delays. To further confound us, the pipelines may not be micropipelines; they may have a different handshaking behaviour. Can we pick the best pipeline to meet our timing needs?

To answer timing questions about the pipelines we create a high level model as

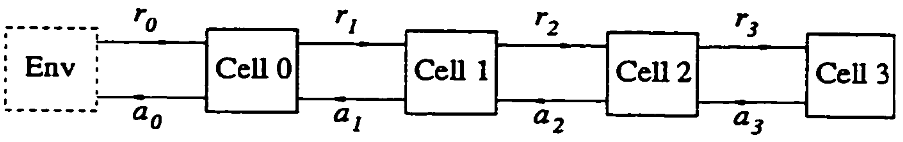


Figure 1.3: The high level model of a pipeline.

in Figure 1.3. The full details of this model are given in Chapter 2. The leftmost cell represents the environment of the pipeline, and controls the input. If we wish we can consider the rightmost cell an output environment, but without loss of generality we consider it integral to the pipeline. The model uses parameters that describe the communication behaviours between the cells, and the component delays. One parameter is the length of the pipeline, L, where L is the number of non-end cells. In Figure 1.3 we have $L = 3$. The models and parameters are chosen for their generality and applicability, allowing a wide range of pipelines to be represented. A well defined model and set of parameters is necessary to derive provable results.

We give two basic measures that describe performance. Response time, RT, is defined as the time between a request from the environment (r_0) and the successive acknowledgment (a_0). We give a closed-form formula for the upper bound of the response time, RT. We also give both lower and upper bounds for the average response time of pipelines, AR. Finally, we give similar measures for the cycle time measured between consecutive requests from the environment.

These formulae allow us to reason about an architecture by studying how each parameter influences a formula. We can also experiment rapidly with different parameter values. For example, we show that average response time AR is bounded as

$$mc_{\Delta_e} - \Delta_e \leq \; AR \; \leq MC_{\delta_e} - \delta_e$$

where the parameters δ_e, and Δ_e are the minimum and maximum delays of the environment respectively. The parameters mc_{Δ_e} and MC_{δ_e} represent the minimum and maximum delay cycles between cells. Figure 1.4 illustrates the possible cycles.

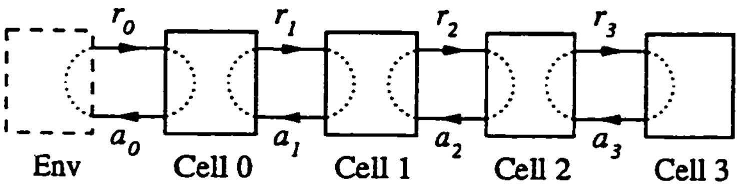


Figure 1.4: Cycles within a pipeline.

The formula above gives us insight into the timing properties of these linear pipelines. The cycles described by mc_{Δ_e} and MC_{δ_e} act as bottlenecks for the average speed of the pipeline. Note the absence of the parameter L in the formula. This indicates that the length of the pipeline is unimportant in determining the bounds on the average response time. Perhaps a designer could take advantage of this and concentrate on the bottlenecks and not the pipeline length. For example, the designer might divide up stages that cause bottlenecks into multiple faster stages. We discuss this issue in detail in Chapter 4.

Lastly, we can instantiate the parameters with values we have derived from the circuit, and thereby calculate bounds on the average response time for a specific circuit. We do just this for the example micropipeline above in Section 4.4. We also calculate the worst-case response time for the same example in Section 3.5.

Our results give bounds, not specific values. This means we need not give specific implementations of cells or compute specific distributions of delays. Different implementations must only satisfy the upper bounds and lower bounds for the delays of the basic cells.

1.3 To Simulate or not to Simulate?

One might ask why one would choose to use a proof-based method rather than use simulation to study timing. Simulation is the workhorse of timing analysis. Simulation involves making a model of the system under study, providing sample inputs, and then simulating the execution of the system. For example, a transistor-level circuit simulator would need to model properties of the transistors, such as the speed at which the output changes when the inputs change. We provide the simulator with a set of transistors, the connections among the transistors, and a set of (possibly random) inputs, and the simulator then uses its model of transistors to calculate the output of the system.

Simulation has the following advantages.

- One can obtain precision at any granularity. One can typically obtain more accurate results by running a longer simulation.
- We have great flexibility in what we can model. One can usually use the same simulator to model a wide range of architectures, a wide range of behaviours, and a wide range of distributions of delays.
- We can model delay distributions and architectures that are impossible to model accurately with mathematical analysis. For example, a commonly used mathematical analysis technique is queueing theory. Queueing theory typically assumes that components with varying delays have delays that are exponentially distributed. Exponential distributions may not accurately reflect the real distribution of the delay.

On the other hand simulation has the following disadvantages.

- Simulation is restricted by the computer time and memory available. We cannot obtain arbitrary accuracy for a simulation, since more complex simulations take more computer memory and more computer time. We never have infinite amounts of either.

- Simulations do not usually give provable results, and we may have to live with confidence intervals. Simulations using Monte-Carlo methods can only gives us an estimate of the likely outcome; they cannot give us a guarantee, since they only use a random subset of the possible inputs. We can only obtain guarantees by performing an exhaustive search of a state space. An exhaustive search is typically computationally intractable.

- Simulation results may not be generalisable. Simulations give us specific results for a specific set of inputs, e.g., for a specific architecture, delay distribution, and behaviour.

- It takes time to set up a simulation to make sure that it is an accurate reflection of the real thing. Obtaining an accurate delay distribution for components, for example, can be difficult.

Therefore, we use an analytic method. The model is chosen so as to balance applicability of results and mathematical tractability. For tractability, we restrict the architectures largely to networks of components that have some regularity to them. For example, Chapters 2, 3 and 4 concentrate on pipelines. Nevertheless even this restricted set of architectures offers a rich area of designs. There are many pipeline designs for various computations where data is flowing in one or both directions. See the references for examples [4, 17, 22, 29, 38, 46, 47]. Chapter 5 investigates tree-based architectures and Chapter 6 examines more general architectures with respect to the bounds on the average cycle time.

1.4 Contributions

The main contributions of this thesis are as follows.

- We give bounds on the worst-case response time RT and worst-case cycle time CT for linear asynchronous pipelines. Cells may have varying, but bounded delays. Each cell may have different delay bounds from other cells, and each cell may have separate delays for communications proceeding through the cell in a forward direction or a reverse direction.

- We give both upper bounds and lower bounds on the average response time AR and the average cycle time AC of linear pipelines.

- We extend the behaviours of simple pipelines to allow different communication behaviours between cells. We can alter the ratio of the number of handshakes, i.e., request/acknowledgment pairs, on one side of a cell with respect to the number of handshakes on the other side. We can also allow for handshakes to be passed conditionally to the next cell in a pipeline. This increased flexibility allows us to apply our results to a larger class of implementations.

- We give upper bounds on the worst-case response time RT and worst-case cycle time CT for tree-based architectures. We give both upper bounds and lower bounds on the average response time AR and the average cycle time AC of tree-based architectures.

- We give results as closed-form formulae. These formulae give insight into the influence of each parameter on the performance of a circuit. We interpret the results, identify bottlenecks, and give advice for pipeline optimisation. Notably:

  - Worst-case response time is linearly dependent on the length of the pipeline for pipelines with multiplication factor $n = 1$. Pipelines with multiplication factor $n = 1$ have cells that have the same number of handshakes on the left as on the right in each repetition cycle.
  - Worst-case response time is dependent only on the first cell for pipelines with multiplication factor $n > 1$, where each cell deeper in the pipeline may be up to n times slower than its predecessor. A similar result holds for trees. Pipelines with multiplication factor $n > 1$ have cells that have n handshakes on the left to every one handshake on the right.
  - Average response time is independent of the length of a pipeline or tree. Asynchronous pipelines may be extremely long, and unlike a synchronous circuit, they cannot suffer from clock-skew problems. Two independent bottlenecks are observed in pipelines with multiplication factor $n = 1$. These bottlenecks constrain respectively the upper and lower bounds on the average response time.
  - We observe the same upper bounds on response time and average response time for pipelines when handshakes are conditionally passed deeper into a pipeline.

  Our use of parameterised formulae to compute bounds on pipeline delay is new, but see the related work below.

- We can simplify the bounded delay model to a fixed delay model when computing average-case bounds, thereby making analysis much simpler.
- We consider a range of architectures beyond pipelines and trees by allowing limited connections between components. We calculate bounds on the average cycle time AC of these new architectures.

Note that parameters, such as the multiplication factor n, and abbreviations, such as AC for the average cycle time, are explained as they are encountered in the following chapters. For reference, these symbols are also summarised in Table B.1 of Appendix B.

The contributions in this thesis extend those found in the work of Ebergen and Berks [16]. This thesis is based on the previous work but extends it in the following ways:

- Cell delays have forward and reverse latencies. Previously, each cell only had one parameter.
- Each cell has its own set of delay parameters. Previously, each cell was identical except for the end cell which had its own parameter.
- Lower bounds on average response time are calculated. Previously, only upper bounds were calculated.
- Cycle time measures are calculated. Previously, the only measures were response time and average response time.
- Results are calculated for tree-based architectures. Previously, they were calculated for linear pipelines only.
- Results are simpler than the previous work. The proofs are also shorter and simpler.
- Chapter 6 that discusses joining components is entirely novel.

1.5 Related Work

We break the related work into two broad categories: (i) analytical approaches and (ii) simulation and state-space search.

Both categories typically have several things in common.

- An event-based behavioural model. That is, actions in a circuit consist of atomic actions, called events, that have time stamps associated with them. Thus event e occurs at time $T(e)$. Events themselves have no duration. The time stamps are determined by the delay model.

- A graphical description of behaviour. Graphical descriptions are an intuitive way to describe behaviour. Dependency graphs, as used in this thesis, are directed graphs, where the vertices represent events in the system, and the edges represent dependencies between events. An event can only occur after all events it is dependent on have occurred.

  A similar model, Signal Transition Graphs (STGs), uses a dependency graph to describe the behaviour. STGs also contain a notion of state, represented by a *marking* of the graph. This marking indicates which events can happen next as the behaviour is simulated. A description of the state of a circuit is only necessary if we are simulating the execution of a system, or if we need to observe the state of a system.

- A reliance on regularity and cyclic behaviour to simplify the analysis. We can typically fold an infinite description of behaviour into a concise repetitious behaviour. For example, consider the behaviour of a wire with input a and output b. An infinite representation of the behaviour of the wire in terms of a sequence of symbols might be

$$abababababababa....$$

After an input a the wire eventually responds by producing an output b. The behaviour then repeats. Using a repetition operator $*[\ ]$ and sequence operator ; we can concisely represent the behaviour of the wire as

$$*[\, a\, ;\, b\,]$$

- A method of finding a critical path, and some method of reducing the complexity of finding that path. In many systems there may be more than one sequence of events occurring concurrently. The critical path is the sequence with the longest delay, as this sequence is responsible for the delay of the entire system. We can restrict the problem of finding a critical path sufficiently such that the problem only takes time polynomial in the size of the graph that represents the behaviour. See Section 2.7.2 for a discussion of this and why the general problem is NP-complete. One can reduce the complexity of finding a critical path in several ways. We can
  - keep to behaviours that are repetitive in nature.
  - keep to behaviours that have some regular structure.
  - simplify the delay model.
  - restrict or disallow certain behaviours that involve choice.

In this thesis, we use an event based model, and describe behaviour using a dependency graph. We largely restrict ourselves to cyclic behaviours, though we often give an unfolded, and possibly infinite, representation of a behaviour to help understand where the critical paths are.

To simplify the complexity of finding a critical path we

- specify behaviours that are exclusively repetitive in nature with the exception of the more general behaviours of Chapter 6.
- keep mostly to investigating regular structures such as linear-arrays of cells, and trees of cells.
- use a bounded delay model.
- make all choices deterministic.

The last point deserves further consideration. Some specifications can, either non-deterministically or on the basis of the value of an input, choose an alternate behaviour. These kind of behaviours are sometimes known as *disjunctive* behaviours. The nice property of deterministic cyclic behaviours is that the ordering of events is predictable, and the timing relationships stay the same from one iteration to the next. This will frequently allow us to iterate to a fixed point. Where choice is involved, we have no way of knowing either the event ordering or the timing relationships a priori. Thus, choice makes timing analysis far more difficult.

We deal with conditional behaviour by effectively eliminating the conditional elements. If we are trying to find worst-case timing behaviour we assume that the behaviour that produces the largest delay is *always* chosen. The alternatives that give less than the worst-case delay are eliminated, and hence choice is removed. While this may seem a pessimistic approach, it still gives tight worst-case bounds. Since we do not specify the probability of making the worst choice, this choice may indeed be made every time. We perform a similar analysis for best-case timing behaviour.

In the sections below we restrict ourselves to examining work on timing analysis of asynchronous circuits only.

1.5.1 Analytical Approaches

Williams [49] examines rings and pipelines, where a ring is simply a pipeline with the output looped back to the input. In Williams's model, signals that pass through a cell in one direction may have a different delay from those travelling in the other. Williams describes the behaviour of a cell by a signal transition graph and then, by finding critical paths through the signal transition graph, derives the forward and reverse delays through the cell. Williams then uses the delays of each cell as parameter values of formulae that compute the throughput and latency of a pipeline. The formulae are simple, because a fixed delay model is used, and because all cells are identical. Williams applies the result to an asynchronous divider circuit [50].

Sparsø and Staunstrup [45] build on the work of Williams. They also use a fixed delay model, and describe component behaviours by means of a signal transition graph. As with Williams, they calculate the delay of the critical path within a component cell. Then they calculate the throughput of a network of components, which this time is a set of rings connected together. They derive simple formulae for throughput and instantiate the parameters with technology-derived values.

Greenstreet and Steiglitz[24] describe pipeline behaviour with a graphical model similar to a signal transition graph. Like Williams, they simplify the model considerably by making all cells identical. The authors use a variety of delay models for the delays of each component of each pipeline cell, viz., a fixed-delay model, a bounded-delay model, and an exponential delay model. They use a different analysis technique for each one. When using fixed delays, they use a simple algebraic technique similar to that used by Williams. Secondly, given (upper) bounded delays they compute a simple bound for throughput. Lastly, given delays with exponential distributions, they compute pipeline throughput using a queueing model and

Monte-Carlo simulation. The results focus on giving the optimum number of data tokens in a pipeline given the cell delays, and calculating the utilisation of the cells. The utilisation is the percentage of time a cell spends on average processing data as opposed to being idle.

Pang and Greenstreet extend the results above [24] to arrays of cells [41]. These architectures consist of rectangular arrays of identical cells. The delay distributions are assumed to be exponential. Again queueing theory is used to obtain conservative bounds on the average throughput and utilization given the number of data tokens in the system.

Franklin and Pan [20] compare the throughput of asynchronous pipelines with that of synchronous pipelines using algebraic methods similar to those used by Williams. The large number of parameters values are sometimes simple approximations, and sometimes they are modelled more accurately. For example, the authors use the circuit simulation package Spice to calculate values for the clock skew parameter in the synchronous pipeline analysis. They also have a parameter that describes the ratio of the speed of the fastest calculation of a stage to the slowest calculation of a stage. They guess that this parameter is 0.5. Clock skew is a major factor in their conclusion that asynchronous pipelines are generally faster.

1.5.2 Simulation and State-Space Search

In this section we look at programming solutions to timing analysis rather than mathematical solutions. Most of the methods given below are exhaustive simulations. That is, all possibilities are examined. However, investigating all possibilities takes time even after making many simplifying assumptions to make the problem tractable.

Burns and Martin [11] present an analysis technique largely derived from Steve Burns's PhD thesis [10]. They model delays with an event-rule system, which describes the events in a system and the dependencies among those events. A behaviour described by an event-rule system can also be described by a constraint graph. A constraint graph is much the same as a dependency graph in this thesis. The behaviours allowed are quite flexible and are not restricted to just pipelines, but are nevertheless cyclic. The authors use a fixed delay model and allow a general set of behaviours that is restricted to strongly connected constraint graphs, i.e, there is a path from each event in the system to every other. They present an algorithm for determining the time separation between events in a system. For example, they can compute response time of a system by computing the time separation between a request event and its corresponding acknowledgment.

Hulgaard et al. [25, 26, 27] extend the work of Burns. The fixed-delay model is extended to allow for bounded delays, and two new algorithms are developed to compute time separations of events. One algorithm works on systems without choice, the other works on systems that allow a restricted form of choice. The analysis algorithm processes cyclic constraint graphs by unfolding them into infinite acyclic graphs. Bounds on the average case, much like our own bounds on average cycle time, are calculated by repeated iterations until a fixed point is reached. Walkup and Borriello [48] give an algorithm for computing event separations using a similar model to Hulgaard, but they apply it to acyclic graphs. The authors do not, however, allow disjunctive behaviours.

Nielsen and Kishinevsky [39] give another algorithm-based approach using the model of Burns [10]. Again the model uses a fixed delay. They find the critical cycle in a behaviour described by a Signal Transition Graph. The graphs describe a repetitive behaviour, but are otherwise arbitrary. The critical cycle is simply the longest cycle in the graph. Lee [33] also extends the model of Burns to allow a greater number of behaviours to allow some forms of disjunction though delays are still fixed. Lee's algorithm also finds critical cycles in cyclic behaviours. Chakraborty et al. [12, 13] calculate event separations given a bounded delay model and general behaviours allowing choice. The behaviours are described as a constraint graph much as in Hulgaard et al. [25]. The algorithm nevertheless runs in polynomial time in the number of events as it is an approximation algorithm. The algorithm gives conservative, i.e., not necessarily tight, bounds.

Xie and Beerel [51] calculate average event separations. They use Markov Chains and stationary probabilities to describe a wide range of concurrent systems. They approximate delay distributions of components with discrete sample points. Xie and Beerel describe a timing analysis algorithm and a tool which uses Binary Decision Diagrams (BDDs) to ameliorate the problems with state-space explosion. BDDs make state-space exploration much quicker for typical problems. The key advantage of their method is that the entire state space is searched exhaustively and so results are provable. The state space is, however, restricted by the discretisation of the probability distributions. Lastly, Berkelaar [5] uses normal distributions to approximate the delays of circuit components, and then computes a statistical average delay of a circuit using max and + operators over these distributions. In effect he has a simulator that calculates the approximate worst-case and average delays.

1.5.3 Comparison of Related Work to this Thesis

Our work borrows the delay model, and many other details, directly from Hulgaard [25] and Burns [11]. We prove bounds analytically, not by execution of a program. Furthermore, our use of closed-form formulae for bounds is novel. Greentreet et al. [24, 41] and Williams [49] use formulae for throughput and latency, but not for bounds, and restrict themselves to identical cells. Franklin and Pan [20] do give some formulae for bounds, but our model uses fewer assumptions, fewer parameters, yet it is more flexible, and gives simpler results. We allow for cells that are not identical, and allow for forward and reverse latencies. Most importantly, we do not have to instantiate our parameters with application-dependent values to achieve meaningful insight. Our results concentrate on the control circuitry, not the data. We are more interested in what the bounds are on the best-case and worst-case throughput, not in finding under which circumstances the best case occurs.

Our goal is not circuit synthesis. Rather we wish to gain understanding of the performance of a system from simple, intelligible models. Unlike the analytic approaches of Franklin and Pan, or of Greenstreet and Steiglitz, the formulae derived in this thesis are all based on one coherent methodology. We believe that the results are more elegant than other approaches, and that proving new results is easier.

1.6 Roadmap of this Thesis

Chapters 1 and 2 in this thesis are largely introductory, and Chapter 7 presents conclusions and future possibilities. Chapters 3 through 6 contain the heart of the thesis: the results, the proofs, and the commentary on the results. Because the proofs are one of the main contributions of this thesis, all proofs are contained within the body of the text and are not hidden away in appendices. To make the main chapters easier to digest, each chapter begins with a summary of the results and of the salient points presented within the chapter. A good overview of the thesis can therefore be gained by reading the introductory sections of each chapter.

A more detailed chapter-by-chapter roadmap is:

1. This introduction, including related work.
2. The behavioural model used, the delay model used, and justifications for both.
3. A discussion of worst-case response time RT and worst-case cycle time CT. Results and proofs are given for pipeline architectures.
4. A discussion of average response time AR and average cycle time AC, two performance measures that are related to throughput. Results and proofs are given for pipeline architectures.
5. Tree-based architectures. The same performance measures as in chapters 3 and 4 are applied to tree architectures.
6. A further investigation of average cycle time. Cycle time is generalised to arbitrary behaviours, and the average cycle time is calculated for networks of components.
7. Conclusions.

Chapter 2

The Model

2.1 Motivation

To perform timing analysis of a system we need a model of the system, and this model can be broken into two parts. First we need a model of the system behaviour. Ideally this model is flexible, intuitive, and succinct. Secondly, we need a timing model. Most importantly, this model needs to be mathematically tractable, so that we may use analytic methods on it.

To describe the behaviours in this thesis, we use two forms. We use a regular-expression like description of behaviour when we wish to be concise. This method is summarised in Appendix A. We also give a graphical description of behaviour. While only one representation of behaviour is strictly necessary, the graphical representation is clearly more intuitive.

We wish to have no restriction on whether events are inputs or outputs, or whether they are derived from voltage levels or from voltage transitions. Our model is therefore event based, where each event is a single atomic action. If we wish to distinguish inputs, outputs, or levels, then we can by labelling events as such. The behaviours we describe here are based on voltage transitions but there is no inherent restriction in the model.

We wish our delay model to be as flexible as possible without compromising mathematical tractability. Our delay model uses bounded delays for the delays of each cell. Instead we could have used models which involved probability distributions of delays. There are, however, several disadvantages of using probability distributions.

- Which distributions should we use? Is the distribution appropriate for the pipeline we are modelling? For example, suppose that we model an instruction pipeline. How long does the instruction execute stage take? Not only do we need to know all possible cases of delay in that stage (which presumably depends on the instruction type), but we also need to know the frequency of the type of instruction.

  Is the distribution appropriate for the environment? As another example, in an instruction pipeline, the arrival of new data is dependent on many factors, including the type of caching, the number of levels of caching, the type of memory, and the percentage of cache hits that occur. Without extensive simulation we can't be sure of the distributions of the inputs, and extensive simulation defeats the original purpose, which was to avoid simulation.

- How do we perform the calculations? Is the distribution tractable mathematically or do we have to use simulation? If we choose something tractable (such as uniform distributions or exponential distributions), we run into the problems above of whether the distribution is accurate. If we choose an accurate distribution, we may not have anything tractable mathematically.

- Are the results rigorously provable? If we don't have a mathematical foundation for our calculations, the answer is no. If the inputs to our calculations are convenient approximations to a distribution, can we make any more than approximate conclusions about the results?

The advantage of the bounded delay model is that it is (almost) always applicable, it is simple enough that we can use it easily, and we can derive provable results, because the bounded delay model is mathematically tractable. In any case, there are some circumstances where we prefer to have bounds, e.g., when synchronising a component with (or within) a clocked circuit.

Markov chain analysis is one compromise that can be made [51], but this requires that the processes involved be memoryless. That is, the next state of a system is based on the current state of a system, and not the entire history of states. One has to use mathematically tractable distributions, such as exponential distributions, to ensure that this property holds. These distributions may not be appropriate. It may also be unclear whether this memoryless property accurately models a given system. Simplifications, such as queueing theory, avoid simulation, but are difficult to apply. Some authors [24, 41] make assumptions, for example, that delays in one direction are negligible.

The parameters used in this chapter describe the behaviour and delays of a pipeline. The symbols used are summarised in Table B.1 in Appendix B.

2.2 Pipelines

We describe pipelines with a behavioural model, a delay model, and a set of parameters. One such parameter is the length of the pipeline, L, where L is the number of non-end cells. In Figure 2.1 we have $L = 3$.

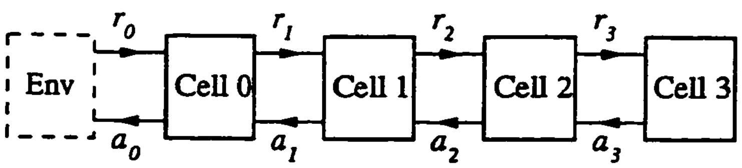


Figure 2.1: A linear array of cells.

We are interested only in the communication behaviour at the interface of each cell as an ordering of events. Each interface between two cells has the same communication behaviour, viz., a repetition of a request r followed by an acknowledgment a. A request followed by an acknowledgment is called a handshake. The end cell handshakes at its only interface, the other cells perform handshakes at both interfaces so that in each cycle one handshake occurs at either side.

Requests and acknowledgments are considered instantaneous events, and delays can be incurred in the cells or the environment of the pipeline. For each cell we assume that the delay between the receipt of the *last* input causing an output and the production of that output is always bounded from above and below. Each cell may have a set of bounds different from other cells, and we also distinguish between whether a signal is transmitted through a cell in a "forward" or "reverse" direction.

The delays of the cells do not have to be fixed, but may vary between a fixed upper and lower bound. Delays may depend on the data received, or they may vary over different instances of the same cell, or they may vary over time.

Section 2.3 discusses the behavioural model. Section 2.4 discusses folded dependency graphs and Section 2.5 introduces some notational conventions. Section 2.6 discusses the delay model. Section 2.7 discusses the computational complexity of the problems in this thesis and relates them to a particular problem in graph theory.

2.3 Communication Behaviours

We give a simple communication behaviour on a cell-by-cell basis, and then combine the communication behaviours of the individual cells to form a pipeline. Consider the cell in Figure 2.2 from a linear array. At interface i, the communication be-

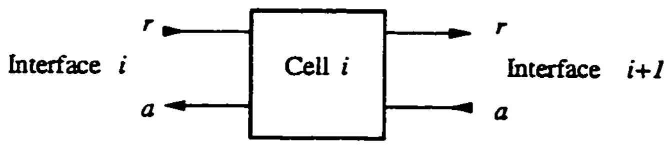


Figure 2.2: One cell.

haviour for the handshakes can be given by $*[r_i?; a_i!]$. That is, there is an input request, $r_i?$, and then there is an output acknowledgement $a_i!$. The notation ? denotes an input, while ! denotes an output. This behaviour then repeats (indicated by $*[..]$). At interface $i+1$ the communication behaviour is given by $*[r_{i+1}!; a_{i+1}?]$.

Cell i synchronizes the handshakes at interface i and $i+1$. In general, the communication behaviour for a cell can take many different forms. We restrict ourselves to behaviours for cells where for every n handshakes on interface i there is one handshake on interface $i+1$. More precisely, the communication behaviours we look at are given by

$$(r_i?;\ *[\ (a_i!;\ r_i?)^n \| (r_{i+1}!;\ a_{i+1}?)\ ])$$

In other words, every occurrence of the segment $(r_{i+1}!;\ a_{i+1}?)$ takes place in parallel with the n handshakes $(a_i!;\ r_i?)$. The parallel bar $\|$ denotes concurrent operation.

Most pipelines operate with $n = 1$. That is, for each handshake on the left of a cell there is a handshake on the right. Results for response time, RT, and average response time, AR, were given in [6]. The results are repeated and expanded to include cycle time here. Handshaking behaviours when $n > 1$ occur for some implementations of counters. Designs that employ a linear array and a multiplication factor of 2 are, for example, modulo-N counter implementations [3, 18, 42] and up-down N counter implementations [19].

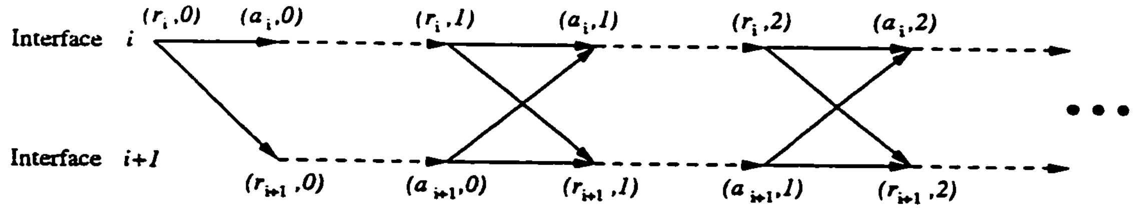


Figure 2.3: Unfolded communication behaviour for a cell.

For $n = 1$ we have a communication behaviour that can be represented by

$$(r_i?;\ *[\ (a_i!;\ r_i?)\|(r_{i+1}!;\ a_{i+1}?)\ ])$$

An unfolded process graph for $n = 1$ for a cell is given in Figure 2.3. Let us denote the k-th occurrence of event a_i by (a_i, k), where $k \geq 0$. A process graph gives a precedence relation for occurrences of events. A precedence between occurrences of events e_0 and e_1 is denoted by $e_0 \rightarrow e_1$. For example, in Figure 2.3 we have the precedences

$$(r_i, 1) \rightarrow (a_i, 1) \quad \text{and} \quad (a_{i+1}, 0) \rightarrow (a_i, 1)$$

These precedences indicate that occurrence j of event a_i can only happen after occurrence j of event r_i and occurrence $j-1$ of event a_{i+1} has happened.

Each precedence arrow has to be caused by either the cell itself or the environment of the cell. Solid arrows indicate precedence relations brought about by the cell. Dashed arrows indicate precedence relations brought about by the environment of the cell.

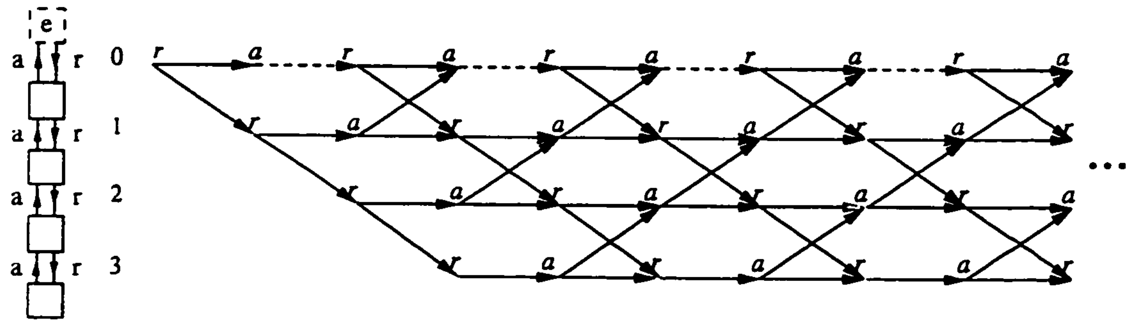


Figure 2.4: Unfolded communication behaviour for a linear array of four cells and multiplication factor $n = 1$.

If we consider the complete network of three non-end cells, one end cell, and the environment of the pipeline, then we can depict its behaviour in a process graph as in Figure 2.4.

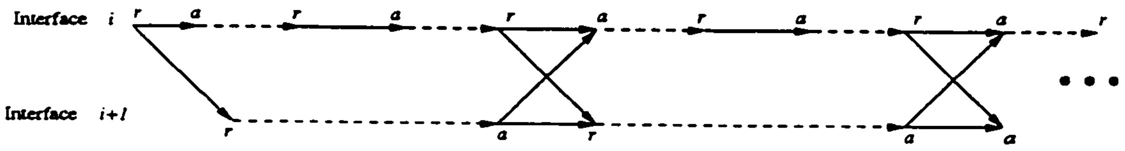


Figure 2.5: Unfolded communication behaviour for a cell with multiplication factor $n = 2$.

For $n = 2$ we have a communication behaviour

$$(r_i?;\ *[\,(a_i!;\ r_i?)^2 \| (r_{i+1}!;\ a_{i+1}?)\,])$$

that can be represented by an unfolded process graph given in Figure 2.5. The corresponding graph for a linear array of four cells is given in Figure 2.6.

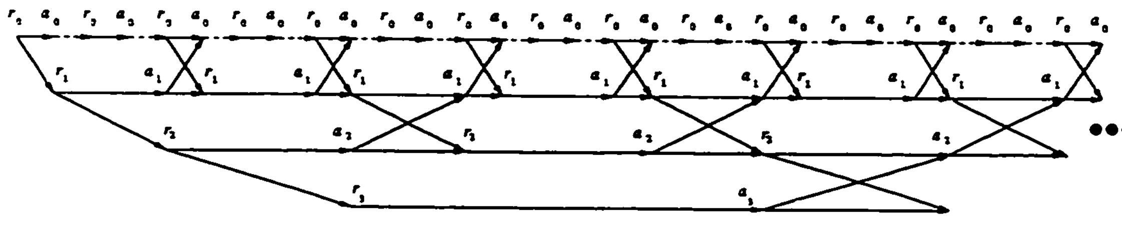


Figure 2.6: Part of an unfolded process graph for a linear array of four cells with multiplication factor $n = 2$.

Our behavioural model is a compromise between applicability and tractability. The behavioural model is tied closely with being able to model micropipelines, as these are the most common pipeline architectures in the asynchronous world, though there is no reason why other pipeline architectures cannot be modelled. There is also nothing in the behavioural descriptions which denies the possibility of other architectures. For example, Chapter 5 gives results for tree architectures. The behavioural model of a pipeline has a regular structure which we exploit in later chapters.

The model is extremely flexible, because we abstract away from the data flow and concentrate on the communication protocols. Several authors [24, 41, 45, 49] calculate the number of "bubbles" needed in pipelines and rings to maximise throughput. These bubbles are the empty spaces in a system. Spaces are necessary in an asynchronous system for data to move around without blocking. We observe that bubbles and data tokens are functionally identical, though they travel in opposite directions in an asynchronous circuit. That is, when a data token displaces a bubble, the data moves to where the bubble was, and the bubble moves to where the data was. Therefore we do not talk of bubbles, data, or any kind of token. All that is important is the timing of the control signals.

Another detail we abstract away from is whether a handshake is *active* or *passive* [2, 35, 36, 43]. A communication is active when it initiates a handshake and passive when it waits for the other side to initiate the handshake. For example, suppose cell i in our pipeline model handshakes with cell $i+1$. Cell i is the active participant, because it initiates the handshake with a request r_i. Cell $i+1$ is the passive participant, because it waits for the request r_i before responding with an acknowledgment a_i. It is irrelevant to our model, however, which side of a handshake is active and which is passive. We are only interested in the events that occur and the dependencies.

One detail we do not deal with in this thesis is four-phase handshaking. Each event is a transition, but we do not distinguish between a rising or falling transition. Four phase handshaking introduces new complexities which we do not have time to deal with here. See [8, 43] for some discussion of these.

2.4 Folded Dependency Graphs

We have been using unfolded dependency graphs to represent behaviour. They are infinite, and contain no cycles. When behaviour follows a regular pattern we can "fold" the dependency graph to give a more concise description of the behaviour. This will become useful in Chapter 6.

As an example of a folded dependency graph consider Figure 2.7 which is the folded representation of the unfolded graph of Figure 2.4. Edges can be labelled with an occurrence index offset. This indicates the difference in occurrence index between the target event of an edge, and the source event of an edge. For example, at level 0, the edge $a_0 \rightarrow r_0$ in the folded dependency graph is labelled with $+1$. That edge represents edge $(a_0, i) \rightarrow (r_0, i+1)$ for some i. The occurrence index offset can be any integer including 0 or a negative number. Edges marked with 0 are usually omitted. Therefore we have an edge $r_0 \rightarrow a_0$ with no occurrence index offset.

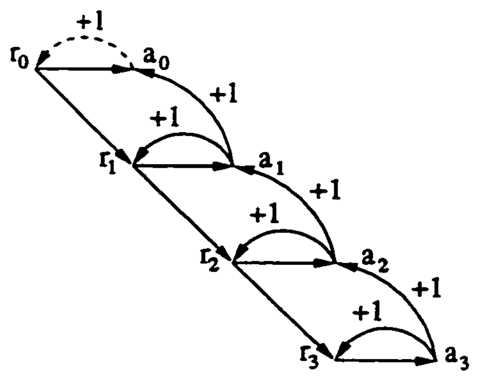


Figure 2.7: Folded dependency graph representing the behaviour of a linear pipeline with multiplication factor $n = 1$ and length $L = 3$.

Note that graphs representing pipelines with $n > 1$ and graphs representing trees (see Chapter 5) cannot easily be represented as folded dependency graphs. Consider a graph such as the pipeline of Figure 2.6. There are many more events at level 0 than there are at level 1, for example. Note that occurrence 0 of a_1 is a direct predecessor of occurrence 2 of a_0. That is, there is an edge $(a_1, 0) \rightarrow (a_0, 2)$. In a folded dependency graph we might have an edge $a_1 \rightarrow^{+2} a_0$ where $+2$ is the occurrence offset. Note, that in Figure 2.6 we also have an edge $(a_1, 1) \rightarrow (a_0, 4)$. Yet our folded dependency graph would give us $(a_1, 1) \rightarrow (a_1, 3)$. The problem is that the offset is an additive offset, rather than a multiplicative one. This problem is not easily solved by the current model of a dependency graph, though allowing multiplicative offsets seems the obvious solution. Folded dependency graphs are a small part of this thesis, but we will be returning to them in Section 2.7.2 and in Chapter 6.

2.5 Notational Conventions

There are certain conventions we adhere to in this thesis. We will use a straight arrow $\rightarrow$ to represent a single edge in the graph. For example, $a \rightarrow b$ represents an edge between event a and its *direct successor* b. We will use a wavy arrow $\leadsto$ to indicate a path of zero or more edges between events. Thus $a \leadsto b$ describes a path in the graph between a and b. Since the path may have zero edges, a and b may be the same event.

We will use Greek letters such as α and β for specific occurrences of events and Roman letters such as a and b for event labels. For example we might have a specific event α such that $\alpha = (a, i)$, for a specific index i. Similarly, $\alpha \rightarrow \beta$ would correspond to a particular edge in an unfolded dependency graph. For example, if $\alpha = (r_0, 0)$ and $\beta = (r_1, 0)$ then $\alpha \rightarrow \beta$ corresponds to the first request/request edge in the graph. In contrast the edge $r_0 \rightarrow r_1$ could correspond to the edge between any occurrence of r_0 and its successor. Also note that, $a \rightarrow b$, without further restrictions on a or b, could correspond to any occurrence of any edge.

2.6 The Delay Model

We need to add a delay model to our behavioural model. As stated in the introduction to this chapter, we will be using bounds on delays, rather than distributions of delays, for their general applicability and their mathematical tractability. The delays in the model have parameters that describe the upper and lower bounds on a delay. These parameters are further subdivided, by allowing different parameters for different cells, and by allowing different parameters for forward or reverse signal flow. Thus for each cell h in a pipeline we have four delay parameters.

- $\delta_{h,f}$ is the lower bound on the delay of information flowing forward through the cell.
- $\Delta_{h,f}$ is the upper bound on the delay of information flowing forward through the cell.
- $\delta_{h,r}$ and $\Delta_{h,r}$ are the corresponding lower and upper bounds on the delay through the cell in the reverse direction.

"Forward" is defined as the direction away from the input environment. In all the pipeline figures given in this thesis where the environment is at the left, this forward direction is left to right. For example, see Figure 2.1. When the pipeline layout is vertical as in Figure 2.4 the forward direction is down. All of these parameters are assumed to have finite values greater than or equal to zero.

As with behaviour we can describe delays in two ways. One is the formal definition, which associates the delay parameters with formal definitions of separations of events. There is also a graphical definition, which associates delays with individual edges in a dependency graph. First we give the formal definition.

Let c be a node in a (unfolded) process graph with c an output of cell h. $T(c)$ is the time assigned to node c and

$$T_{pred}(c) = \max\{T(b) \mid b \text{ is a direct predecessor of c}\}$$

That is, $T_{pred}(c)$ is the latest occurring direct predecessor of c. We stipulate that the time stamp of an event c, i.e., $T(c)$, is calculated from the time stamp of the *last* directly preceding event, i.e., $T_{pred}(c)$. For example, if the precedences $a \rightarrow c$, and $b \rightarrow c$ exist, then the time stamp, $T(c)$ is calculated by adding a delay to the time stamp of the latest occurring of a and b. Thus we know both a and b must have occurred before c can occur. Our delay model prescribes that any timing assignment T of the process graph must satisfy the following constraints.

$$T_{pred}(c) + \delta_{h,f} \leq T(c) \leq T_{pred}(c) + \Delta_{h,f}$$

if node c is a request, and

$$T_{pred}(c) + \delta_{h,r} \leq T(c) \leq T_{pred}(c) + \Delta_{h,r}$$

if node c is an acknowledgment. Thus, event c occurs at least $\delta_{h,f}$ later than $T_{pred}(c)$ and at most $\Delta_{h,f}$ later than $T_{pred}(c)$ if c is a request. A similar reasoning applies when c is an acknowledgment.

The constraints prescribe that "forward" delays through cell h have lower bound $\delta_{h,f}$ and upper bound $\Delta_{h,f}$, that is

$$\delta_{h,f} \leq \text{cell } h \text{ forward delays} \leq \Delta_{h,f}$$

Similarly, "reverse" delays for cell h are bounded by $\delta_{h,r}$ and $\Delta_{h,r}$.

We have only chosen to divide delays into "forward" and "reverse" delays. We could have chosen to divide the parameters into four groups. For example, we could have separate parameters for $r \rightarrow r$ and $a \rightarrow r$ edges. Likewise, we could have separate parameters for $a \rightarrow a$ and $r \rightarrow a$ edges. We have for simplicity used only two sets of parameters, but in most cases two sets of parameters is enough.

Consider the worst-case delay between an input request to an output request; that is, the delay on request/request edge. Call this worst-case delay $D_{r \to r}$. Consider the delay between an input acknowledgment to an output request; that is, the delay on acknowledgment/request edge. Call this delay $D_{a \to r}$. Let $D_{r \to r} = x + z$ and let $D_{r \to r} = y + z$ where z represents the delay of the part of the circuit that is common to both paths. If there is combinational logic between cells, this common delay part will be large, since part of this delay is the delay that matches the combinational logic. Note that this delay is modelled by the triangular buffers in the micropipeline schematics seen in Figure 1.2 and Figure 2.13(i). If z is large relative to x and y, then $D_{r \to r} \approx D_{a \to r}$ and the assumption that we need one set of parameters only for the forward direction is reasonable.

Even if $D_{r \to r}$ is significantly different from $D_{a \to r}$ we can always make conservative approximations. For example, if $D_{r \to r} > D_{a \to r}$ then we use $D_{r \to r}$ as our forward delay. Similarly, for lower bounds we take the smallest of the two possible delays. These approximations mean the bounds we derive may not be tight. While it is likely that reverse paths have similar commonality in delays, we can always make conservative estimates here too.

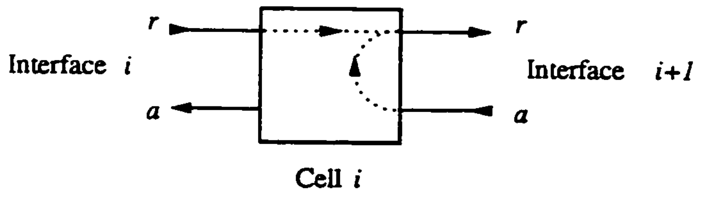


Figure 2.8: Forward delays in a cell.

We must be clear about the delays associated with cell h. Cell 0 controls the "forward" edges $r_0 \to r_1$ and $a_1 \to r_1$. See Figure 2.8. Likewise, cell 0 controls the "reverse" edges $r_0 \rightarrow a_0$ and $a_1 \rightarrow a_0$ See Figure 2.9. Therefore cell i controls edges

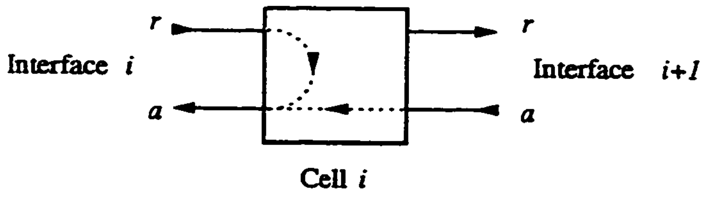


Figure 2.9: Reverse delays in a cell.

$$\left.\begin{array}{l} r_i \rightarrow r_{i+1} \\ a_{i+1} \rightarrow r_{i+1} \end{array}\right\} \text{forward latencies.}$$

$$\left.\begin{array}{l} r_i \rightarrow a_i \\ a_{i+1} \rightarrow a_i \end{array}\right\} \text{reverse latencies.}$$

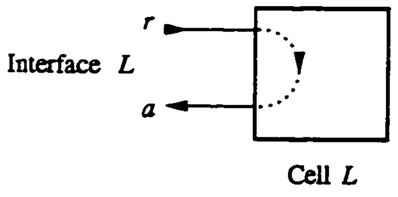


Figure 2.10: The end cell.

The end cell in Figure 2.10 is special in that it has only reverse latencies. The only edge through the end cell is the "reverse" edge $r_L \rightarrow a_L$. Similarly, the environment has only forward latencies associated with it as the environment edges are of the form $a_0 \rightarrow r_0$. The bound for this delay can be thought of as $\delta_{-1,f}$ and $\Delta_{-1,f}$. Instead, we label the lower bound and upper bound on the environment delays δ_e and Δ_e, respectively.

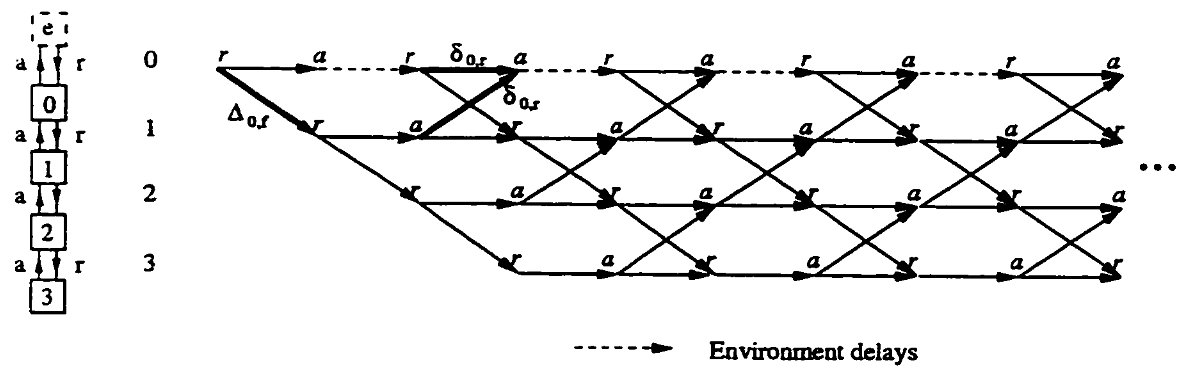


Figure 2.11: Delays applied to the three edges in the graphical model of a pipeline with length $L = 3$ and multiplication factor $n = 1$.

We can apply the delay model to the graphical behavioural model to get a better understanding of the delay model. The behavioural model describes the infinite behaviour of a pipeline. For any given execution of the pipeline behaviour, each edge in the infinite behaviour has a delay associated with it. Refer to Figure 2.11. We have associated delays only with the three highlighted edges. Normally we would associate a delay with every edge. Note that the labelling of the edges here corresponds to delays, not to the occurrence index offsets of folded dependency graphs.

The edge $(r_0, 0) \rightarrow (r_1, 0)$ has been allocated delay $\Delta_{0,f}$. Note that we could have chosen *any* value between the bounds $\delta_{0,f}$ to $\Delta_{0,f}$ for this edge. This is the delay of edge $(r_0, 0) \rightarrow (r_1, 0)$ in this execution, and ensures that event $(r_1, 0)$ occurs $\Delta_{0,f}$ after event $(r_0, 0)$.

Now consider the edges marked with $\delta_{0,r}$. Here we could have chosen any delay within the bounds $\delta_{0,r}$ to $\Delta_{0,r}$ for these edges. The acknowledgment event at the destination of the two edges either occurs at time $T(a_1, 0) + \delta_{0,r}$ or at time $T(r_0, 1) +$

$\delta_{0,r}$. The destination event depends on both predecessor events, so it occurs at time $\max\{T(a_1, 0) + \delta_{0,r}, T(r_0, 1) + \delta_{0,r}\}$.

Note that the choice of delay we make for these edges has no bearing on the delays we choose for any other edge (including later occurrences of the same edges) in the execution. By allowing complete independence of delays between edges, we have a simple model to work with.

2.6.1 Minimum and Maximum Cycle Times

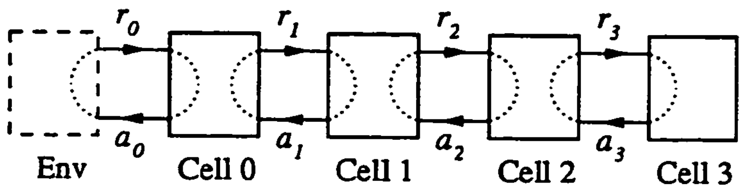


Figure 2.12: Cycle times within a pipeline.

Lastly we define the following expressions representing the maximum cycle time and minimum cycle time respectively between cells. These are derived by calculating the sum of a forward and of a reverse latency. See Figure 2.12. The cycle time between consecutive cells will prove to be important. The maximum cycle time between cells with the environment delay maximised is defined as

$$MC_{\Delta_e} = \max\{\Delta_e + \Delta_{0,r}, \Delta_{0,f} + \Delta_{1,r}, \Delta_{1,f} + \Delta_{2,r}, \ldots, \Delta_{L-1,f} + \Delta_{L,r}\}$$

where we see that all delays are maximised. The minimum cycle time is defined as

$$mc_{\delta_e} = \max\{\delta_e + \delta_{0,r}, \delta_{0,f} + \delta_{1,r}, \delta_{1,f} + \delta_{2,r}, \ldots, \delta_{L-1,f} + \delta_{L,r}\}$$

where we see that all delays are minimised.

We also define the corresponding expressions that will be useful when we calculate response time, which is a measure that excludes the delay of the environment. The maximum cycle time between cells with the environment delay minimised is defined as

$$MC_{\delta_e} = \max\{\delta_e + \Delta_{0,r}, \Delta_{0,f} + \Delta_{1,r}, \Delta_{1,f} + \Delta_{2,r}, \ldots, \Delta_{L-1,f} + \Delta_{L,r}\}$$

Note that the environment delay is a lower bound, while all other values are upper bounds. We choose this representation because when we calculate worst-case response times we are interested in the maximum delay of the pipeline and the minimum delay of the environment. This will become clear in the next chapter.

Lastly we have the corresponding minimum cycle time. The minimum cycle time is defined as

$$mc_{\Delta_e} = \max\{\Delta_e + \delta_{0,r}, \delta_{0,f} + \delta_{1,r}, \delta_{1,f} + \delta_{2,r}, \ldots, \delta_{L-1,f} + \delta_{L,r}\}$$

Note that the environment delay is an upper bound, while all other values are lower bounds. Since $\Delta_e \geq \delta_e$ it is not necessarily true that $mc_{\Delta_e} \leq MC_{\delta_e}$.

2.6.2 A Micropipeline Example

Micropipelines were first described by Sutherland [47] and are a frequently used type of asynchronous circuits. Figure 2.13 shows three representations of a pipeline.

Figure 2.13 (i) gives the schematic of the control circuitry of a micropipeline. Data (not illustrated) travels from left to right and the C-elements regulate the data's progress. Each C-element waits for a transition on both inputs before producing an output transition. See Appendix C for a description of C-element behaviour. The small triangles represent delay buffers, that are associated with data processing delay.

Figure 2.13 (ii) gives a conceptual idea of the control circuitry. The wheels represent the flow of control and turn in the same direction as the signals in the micropipeline. Each spot on a wheel must synchronise with a matching spot on an adjacent wheel at a bar, but may otherwise spin at any speed. The speed would be governed by the delay buffers in the micropipeline. The bars perform the same synchronisation as the C-elements in (i).

Figure 2.13 (iii) converts the micropipeline to our model. In our model the pipeline has three non-end cells and one end-cell. Hence, we model it as a linear pipeline with parameter $L = 3$. Micropipeline stages have one handshake on the right to every handshake on the left and so we use the parameter $n = 1$.

In Figure 2.14 we annotate the micropipeline with delays. We assume that the only delays in the circuit are the delay buffers. That is, we assume that the C-elements and wires have zero-delay. This causes no loss of generality, because we can always model the delays in the C-elements within the delays of the buffers. We can do this by adding the delay of the C-element to the delay of both buffers.

2.6.3 Critical Paths

When a dependency graph and a timing assignment is given, we can define the useful notion of *critical paths*. Informally, given two events α and β, and a critical path $\alpha \leadsto \beta$ between them, then the sum of the delays on this path is equal to $T(\beta) - T(\alpha)$. That is to say, the delay on this path is responsible for the delay between the events.

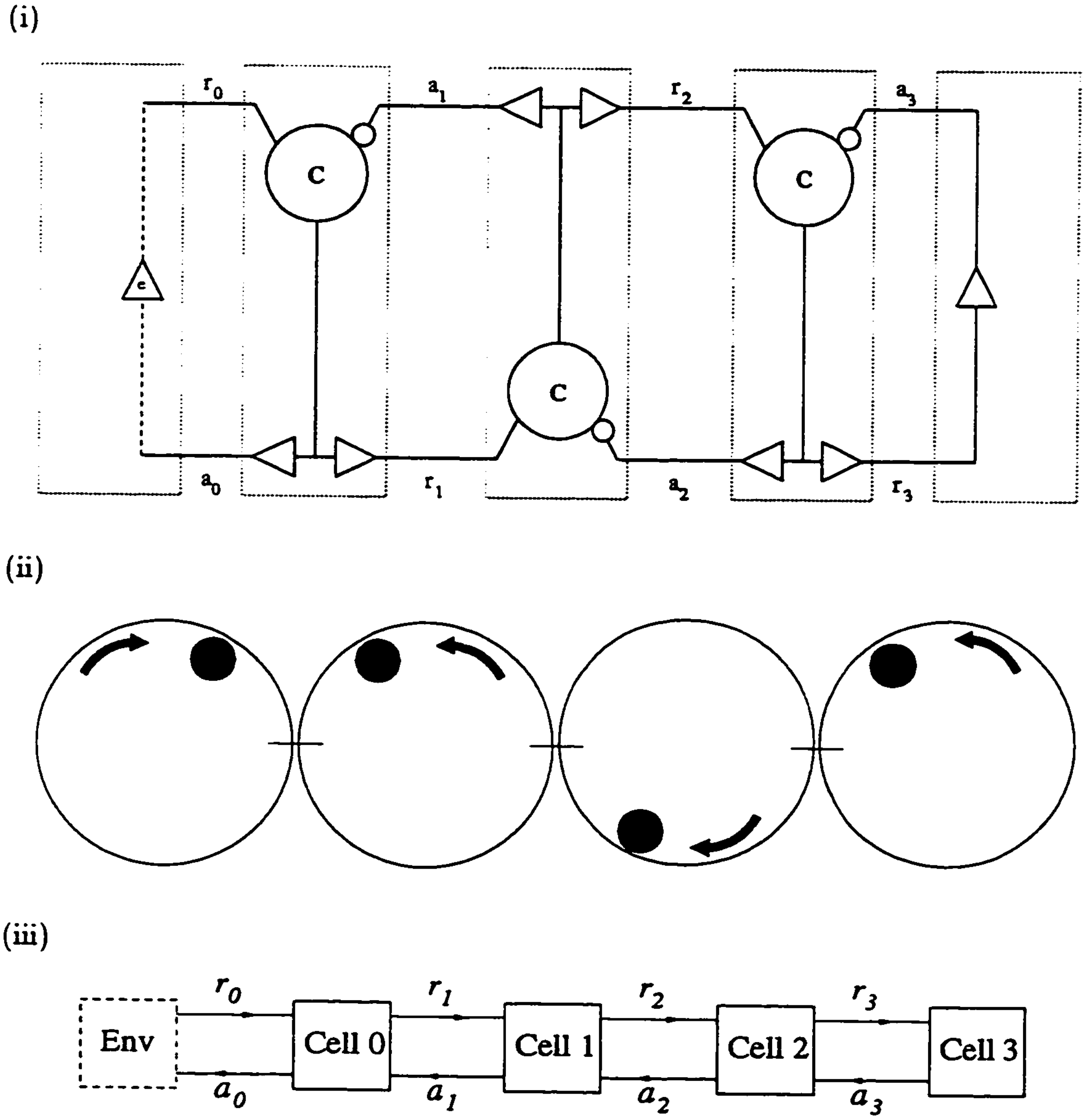


Figure 2.13: Three representations of a pipeline: (i) a micropipeline. The environment is labelled with an e and the cells are outlined, (ii) a simplified representation of the control flow, and (iii) our model where length $L = 3$.

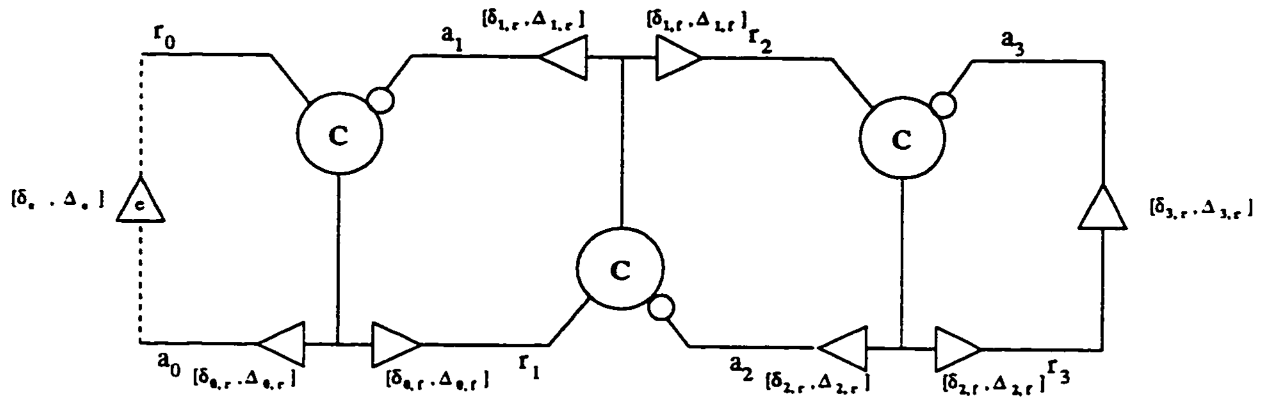

r_0
a_1
$[\delta_{1,r}, \Delta_{1,r}]$
$[\delta_{1,f}, \Delta_{1,f}]$
r_2
a_3
C
C
C
$[\delta_e, \Delta_e]$
e
$[\delta_{3,r}, \Delta_{3,r}]$
a_0
$[\delta_{0,r}, \Delta_{0,r}]$
$[\delta_{0,f}, \Delta_{0,f}]$
r_1
a_2
$[\delta_{2,r}, \Delta_{2,r}]$
$[\delta_{2,f}, \Delta_{2,f}]$
r_3

Figure 2.14: A micropipeline with length $L = 3$. The environment delay is labelled with an e.

We define an edge $\alpha \dashrightarrow \beta$ to be a *critical edge* if one of the following conditions is met,

(i) $\beta = r_0$, and $\delta_e \le T(\beta) - T(\alpha) \le \Delta_e$ where α must be an occurrence of the event a_0.

(ii) $\beta = r_{h+1}$, and $\delta_{h,f} \le T(\beta) - T(\alpha) \le \Delta_{h,f}$ for some event α.

(iii) $\beta = a_h$, and $\delta_{h,r} \le T(\beta) - T(\alpha) \le \Delta_{h,r}$ for some event α.

The above definition covers the three cases: environment delays, forward delays and reverse delays. Note that the lower bound δ_x (where $x \in \{e, (h,f), (h,r)\}$ depending on the type of edge) is automatic by our statement that edge delays must be at least δ_x. The candidate upper bound Δ_x may not always reflect the upper bound on delay between events α and β, because other dependencies may make event β occur later. We call a directed path a *critical path*, if all its edges are critical edges.

We can view the critical edges as pieces of elastic which can "stretch" between lengths δ_x and Δ_x. For each node at least one incoming edge must be a critical edge, because of the delay constraints. If we stretch an edge beyond Δ_x it breaks. On a critical path, we know that no bit of elastic on that path is broken, since, by the choice of the edges on the path, none of the bits of elastic is stretched beyond its upper bound Δ_x.

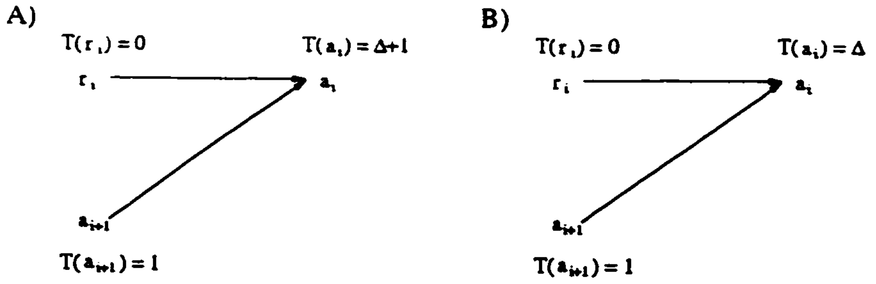


Figure 2.15: Example of critical paths. In **A)** only edge $a_{i+1} \to a_i$ is critical. In **B)** both edges are critical.

Examples of critical edges are given in the process graph fragments of Figure 2.15. The paths that we define as critical depend on the time stamps. The edge delays may influence the time stamps, but do not directly determine whether an edge is critical. In Figure 2.15 some critical edges are illustrated by means of two different timing assignments to the nodes of a process graph. Notice that in (A) only edge $a_{i+1} \to a_i$ is critical, while in (B) both edges are critical. In part (B), note that edge $a_{i+1} \to a_i$ is critical by our definition, because $T(a_i) - T(a_{i+1}) \leq \Delta$.

2.6.4 Special Path Definitions

To simplify notation in later chapters we use the following definitions. These types of paths will crop up when we look at average-case behaviours.

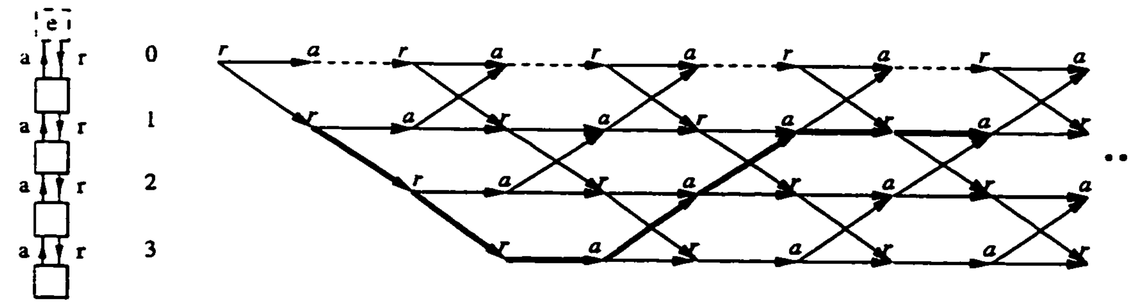


Figure 2.16: An example of a path P in $[1,3]$.

A path $P \in [x,y]$ is a path in a dependency graph that consist of events between level x and level y, where level refers to the depth in the pipeline of the event. For example, a path $P \in [0,L]$ would be an unrestricted set of paths. A path $P \in [1,3]$ would consist of only the events r_1, a_1, r_2, a_2, r_3, and a_3. Figure 2.16 illustrates an example path in $[1,3]$.

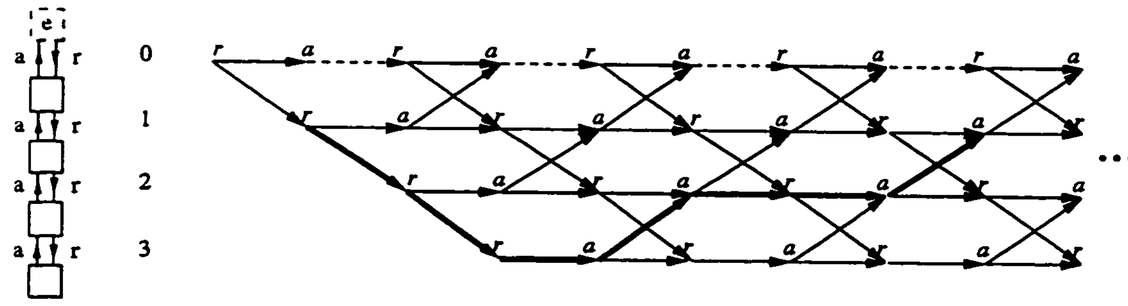


Figure 2.17: An example of a path P in $(1,3]$. Note that only the start and end events are at level 1. All other events are in $[2,3]$.

A path $P \in (x,y]$ is a path $P \in [x+1,y]$, except for the end points of the path, which are events at level x. For example $P \in (1,3]$ might be a path $r_1 \rightarrow r_2 \rightarrow r_3 \rightarrow a_3 \rightarrow a_2 \rightarrow r_2 \rightarrow a_2 \rightarrow a_1$. See Figure 2.17. We will typically be using paths, $P \in (0,L]$, that have both endpoints at level 0, (i.e., at the environment), and no intervening events at level 0. Note, that the definition does not preclude single edge paths at the environment in this case. For example, a path $(r_0, i) \rightarrow (a_0, i)$ is a path in $(0, L]$.

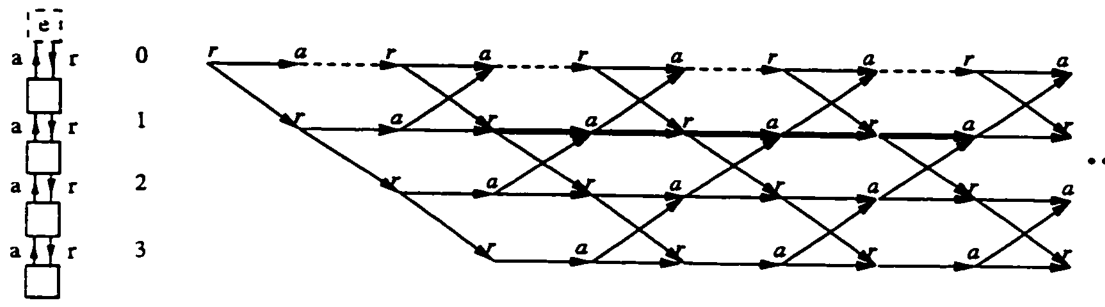


Figure 2.18: A dependency graph, with an example of a path P in $[1, 1]$ highlighted.

Finally, we call a path a *level x path*, if it consists of events only at level x. This is represented in the graphs by a horizontal set of edges. A path $P \in [x, x]$ would also describe a level x path. Figure 2.18 illustrates a path at level 1.

2.7 Related Problems in Graph Theory

Now that we have introduced our graphical model we can relate the problems in this thesis to graph theory. First, however, we discuss the computational complexity of the problem, if we were to compute our results by means of a program.

2.7.1 Min-Max Constraints

Our work is a restriction of a more general problem, the problem of finding event separations in the presence of "Min-Max" constraints. When an event α is dependent on *all* predecessor events, we find the time stamp of the last occurring predecessor using the max operator and calculate the time of event α from there. We call the formula used to calculate the time stamp of an event a "max constraint" if it uses only the max operator. We can see such constraints in Section 2.6. Some concurrent systems such as those that have events that depend on only one of a set of predecessor events may require the use of the min operator. These behaviours crop up when we deal with choice in a behaviour. When an event α is dependent on *one* of a set of predecessor events, we find the earliest time of the predecessors using the min operator and calculate the time of event α from there. We call the formula used to calculate the time stamp of an event a "min constraint" if it uses only the min operator. Finally, we call a formula that calculates the time stamp of an event a "min-max constraint" if it uses both min operators and max operators.

McMillan and Dill [37] show that finding event separations when we have "min-max constraints" and bounded component delays is NP-complete for acyclic systems. As we saw in Section 1.5.2, most authors restrict the problem so that it is not NP-complete. They examine either a cyclic system, avoid disjunction, or use fixed delays. The concurrent systems in this thesis require only the use of the max operator, and thus we stick to the simpler model of "max-algebras".

2.7.2 The Minimal Cost-to-Time Ratio Cycle Problem

We relate the problems in this thesis to the Minimal Cost-to-Time Ratio Cycle Problem as described in Dantzig [14] and in Lawler [32]. Here the problem is described as a directed cyclic graph, which is like a folded dependency graph. Each edge is weighted by two values, a "profit" value, and a "cost" value[1]. The problem is to find the cycle that maximises the profit, while minimising the cost.

[1]The terms used here are mine.

For example, consider a transport ship that may travel to any port it chooses [14, 32]. Each port is a vertex in the graph. Each arc is a voyage between ports and the "cost" is the number of days that the voyage between ports takes. The "profit" weighting is the amount of money that the ship collects by travelling between a pair of ports. The journey that maximises the ship's profits per day is therefore the cycle through the graph that maximises the ratio of profits/cost, or in other words minimises the ratio of cost/profits. This problem is a fixed delay problem with max constraints. Lawler [32] gives an algorithm that solves this problem in $O(|V|^3 \log |V|)$ time, where $|V|$ is the number of vertices in the graph.

Williams [49] recasts this problem to finding the critical cycle in a folded dependency graph. The "profit" value is now the (fixed) delay of an edge. The "cost" value is now the change in occurrence indices around the cycle. The problem is nevertheless the same as the Minimal Cost-to-Time Ratio Cycle problem. We wish to find the cycle with the maximum delay divided by the number of occurrence indices the cycle encompasses. For example, perhaps we have a cycle in a graph that is weighted with 10 units of delay between events (e, i) and $(e, i+1)$, and another cycle that is weighted with 18 units of delay between events (e, i) and $(e, i+2)$. The ratio of (delay/number of event occurrences) is 10 for the first cycle, and 9 for the second cycle, so the first cycle is the critical one.

Solving the Minimal Cost-to-Time Ratio Cycle problem is addressed by other authors, each giving a polynomial algorithm. Burns [10] gives an algorithm that is $O(|V||E|)$ given certain restrictions on the occurrence indices. Nielsen and Kishinevsky [39] give an algorithm that computes the problem in $O(b^2|E|)$ for some small value b and an initial state marking. So, in general, finding the critical cycle is a problem polynomial in the size of a graph. In contrast, our solution gives analytical solutions for specific examples.

Chapter 3

Calculating Worst-Case Response Times

3.1 Response Times

In the previous chapter we described our model, and in particular, how that model applied to pipelines. Now we are ready to start answering questions about the speed of the pipeline. In this chapter we introduce two of our measures, Response Time, RT, and Cycle Time, CT, and give closed-form formulae for the upper bounds of both quantities.

Both response time and cycle time measure the worst-case time separation between two events given the worst possible combination of delay assignments to the elements in a pipeline. In the case of response time, we measure the delay between a request from the environment of the pipeline to the succeeding acknowledgment from the pipeline. See the illustration in Figure 3.1. In the case of cycle time, we measure the delay between a request from the environment and the next request from the environment. See the illustration in Figure 3.2.

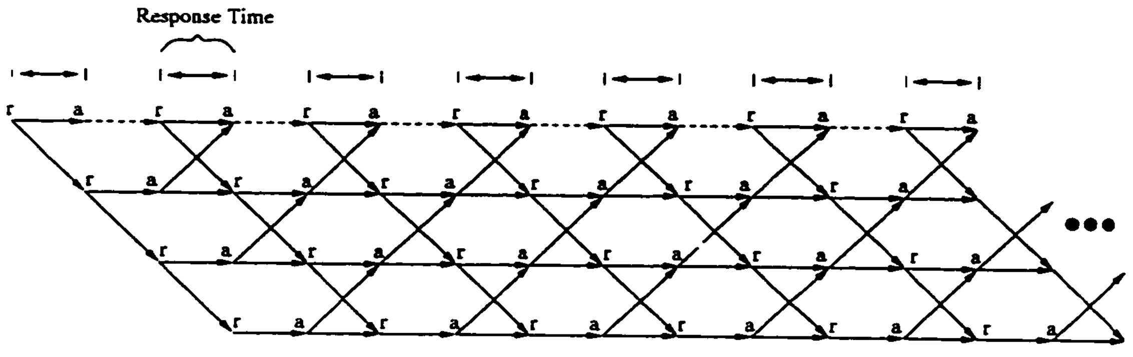


Figure 3.1: Response times are the time separations between a request and its corresponding acknowledgment. Given a graph of the behaviour of a pipeline with length $L = 3$ and multiplication factor $n = 1$, the relevant separations are illustrated. We have to select the occurrence of the request/acknowledgment pair carefully to find the worst-case response time.

Because we are calculating a worst-case bound, we can reduce the complexity of the problem significantly. There are infinitely many possible choices of arbitrary delay assignments for a process graph, as the delay ranges are continuous. Even given a particular delay assignment we still have to find the critical path from the infinite number of paths through the graph. Instead of considering an infinite number of cases, we show that we need to examine only one timing assignment and L paths, where L is the length of the pipeline, by judiciously choosing delay assignments and paths. This is one of the key strengths of using bounds – a reduction of the complexity of calculating worst-case response times to manageable proportions.

The measures in this chapter are useful if we wish to know the worst possible delay of a pipeline for one communication. This may be vital information for synchronizing the pipeline with other systems that represent the environment of

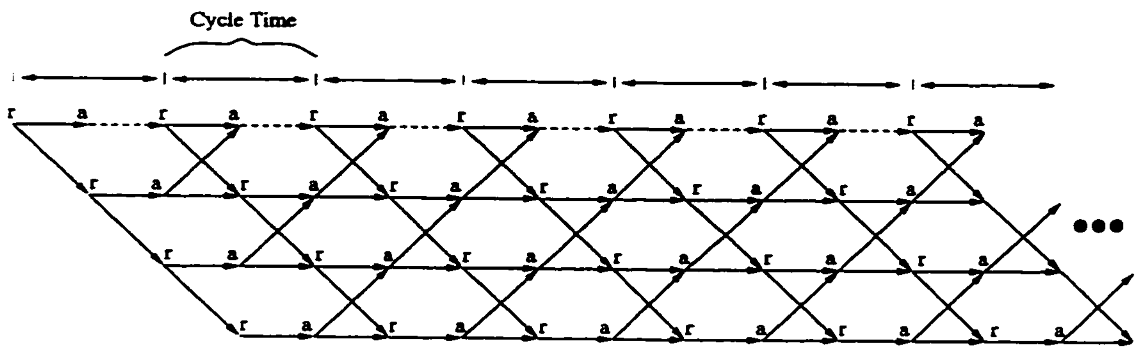


Figure 3.2: Cycle times are the time separations between successive requests. Given a graph of the behaviour of a pipeline with length $L = 3$ and multiplication factor $n = 1$, the relevant separations are illustrated.

the pipeline. For example, we need guarantees of the worst-case speed of an asynchronous circuit if we are to interface it with a clocked circuit. As another example, consider a real-time system. Real-time systems also need to give delay guarantees.

This chapter proves the following.

- Response time for pipelines, RT, is bounded as follows

$$\begin{aligned} RT \;\leq\; & \max(\{\Delta_{k,r} + \Delta_{k-1,f} + \sum_{h=0}^{k-1}(\Delta_{h,r} - \delta_{h,r}) \\ & -(\delta_{0,r} + \delta_e)n^k + \delta_{0,r} \mid 1 \leq k \leq L\} \cup \{\Delta_{0,r}\}) \end{aligned}$$

  Response time depends on the longest of $L + 1$ possible critical paths, where each path corresponds to the maximum depth the path reaches in the pipeline. The formula above maximises over the delays on each of the $L + 1$ paths. Note that the response time is strongly dependent on the variation in reverse latencies, viz., the summation term. If there is some variation in the reverse latencies, then this summation term gets bigger and bigger with k, and hence is maximised at L.

If we can rearrange the pipeline we can minimise the bound by careful placement of cells. When the multiplication factor is $n = 1$, we place the slow cells at the front. Thus when $k = 1$ we maximise $\Delta_{k,r} + \Delta_{k-1,f}$ and when $k = L$ we maximise the summation term. When $n > 1$ we can hide slower cells deeper in the pipeline because the exponential term $(\delta_{0,r} + \delta_e)n^k$ takes over. We then optimise cell 0 for speed, and we have the freedom to optimise cells deeper in the pipeline for other measures such as power consumption or area. See Section 3.3 for discussion and caveats.

- An upper bound for the cycle time of pipelines, CT_{ub}, is

$$CT_{ub} = RT_{ub} + \Delta_e$$

where RT_{ub} is any upper bound on the response time RT. In particular, we can add Δ_e to the bound for response time given above to obtain

$$\begin{aligned} CT \leq\ & \Delta_e + \max(\{\Delta_{k,r} + \Delta_{k-1,f} + \sum_{h=0}^{k-1}(\Delta_{h,r} - \delta_{h,r}) \\ & -(\delta_{0,r} + \delta_e)n^k + \delta_{0,r} \mid 1 \leq k \leq L\} \cup \{\Delta_{0,r}\}) \end{aligned}$$

Thus cycle time has the same bound as that for RT with the addition of one environment delay Δ_e.

- In general, any upper bound on cycle time, CT_{ub}, is related to an upper bound on response time, RT_{ub}, for *any* system with a handshaking environment as

$$CT_{ub} = RT_{ub} + \Delta_e$$

- We can calculate cycle time between consecutive requests or consecutive acknowledgments. We choose to calculate cycle time between consecutive requests, since it serves as an upper bound for cycle time calculated between consecutive acknowledgements. Surprisingly, the bounds calculated are not necessarily the same in each case. See Section 3.4 and, in particular, the discussion of Lemma 3.6 as to why this is so.

- Behaviour in which handshakes are conditionally passed deeper into the pipeline have the same worst-case bounds for RT and CT as those calculated for non-conditional behaviour. We define conditional pipeline behaviour in the same way we define normal pipeline behaviour except that each cell has a boolean guard. When the guard is true, a normal handshake communication takes place with the cell's right hand neighbour. When the guard is false, the normal handshake communication with the cell's right hand neighbour is omitted. See Section 3.6. Conditional behaviour can be used to model some implementations of stacks, for example. As with previous calculations, we carefully select a set of possible critical paths to prove our results. Since we are looking for worst-case bounds, we can choose a set of paths that is equivalent to the set of paths chosen for non-conditional behaviour.

3.2 A Timing Property

Before we formulate and prove some properties about response times, we define the concept of *response depth* of an occurrence of an event. For the definition of response depth one can use Figure 3.3 as an illustration. Let the following path exist in B, where B is the process graph that describes the pipeline behaviour.

$$(a_{i+k}, j_{i+k}) \rightarrow (a_{i+k-1}, j_{i+k-1}) \rightarrow ...(a_{i+1}, j_{i+1}) \rightarrow (a_i, j_i)$$

with $k \geq 0$ and occurrences $j_{i+k} < j_{i+k-1} < j_{i+k-2} < ... < j_{i+1} < j_i$. That is, the acknowledgment a_i depends on a_{i+k} via a sequence of acknowledgments, one for each interface. Let furthermore T be a timing assignment for behaviour B. We

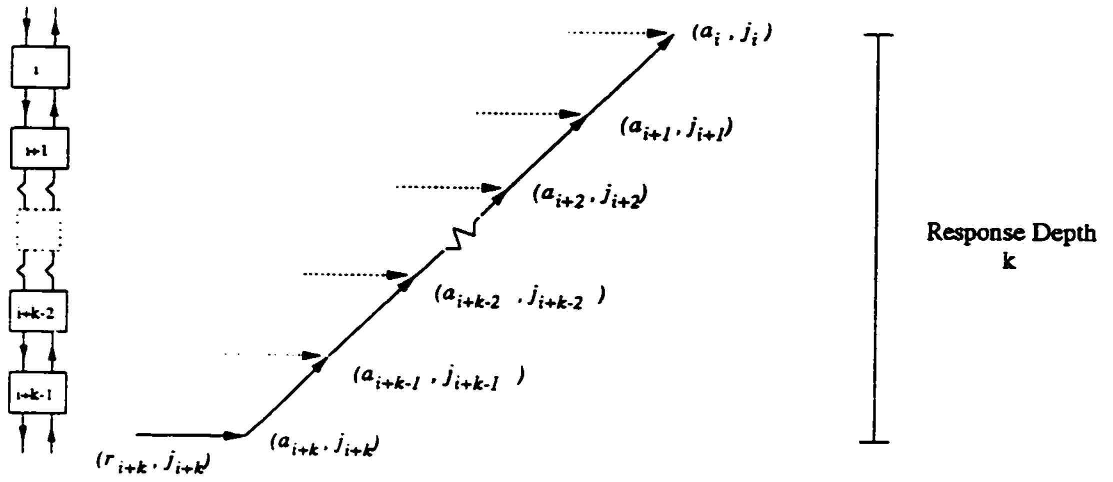


Figure 3.3: Event (a_i, j_i) has response depth k. That is, there is a critical path $(a_{i+k}, j_{i+k}) \rightsquigarrow (a_i, j_i)$ consisting only of acknowledgment/acknowledgment events. All solid edges $(a_{x+1}, j_{x+1}) \rightarrow (a_x, j_x)$ have delay of at most $\Delta_{x,r}$. Either there is no edge $(a_{i+k+1}, j_{i+k+1}) \rightarrow (a_{i+k}, j_{i+k})$ or that edge is not critical.

say that (a_i, j_i) has *response depth k* for behaviour B and timing assignment T iff the path $(a_{i+k}, j_{i+k}) \rightarrow (a_{i+k-1}, j_{i+k-1}) \rightarrow ...(a_{i+1}, j_{i+1}) \rightarrow (a_i, j_i)$ is a critical path, and the edge $(a_{i+k+1}, j_{i+k+1}) \rightarrow (a_{i+k}, j_{i+k})$ does not exist, or, if it does exist, it is not critical. Let us call this critical path the "response path" for acknowledgment (a_i, j_i). For example, if we have a response path with $i = 0$, and $k = L$, then we have a critical path of acknowledgments from the end-cell to the environment, with response depth L.

The following auxiliary lemma will be useful when we actually calculate the worst-case response time.

Lemma 3.1 *If the response depth of* (a_i, j_i) *is* $k, i+1 \leq i+k \leq L$, *then event* $(a_i, j_i - n^k)$ *exists, and*

$$T(a_i, j_i) - T(a_i, j_i - n^k) \leq \Delta_{i+k,r} + \Delta_{i+k-1,f} + \sum_{h=0}^{k-1}(\Delta_{i+h,r} - \delta_{i+h,r})$$

□

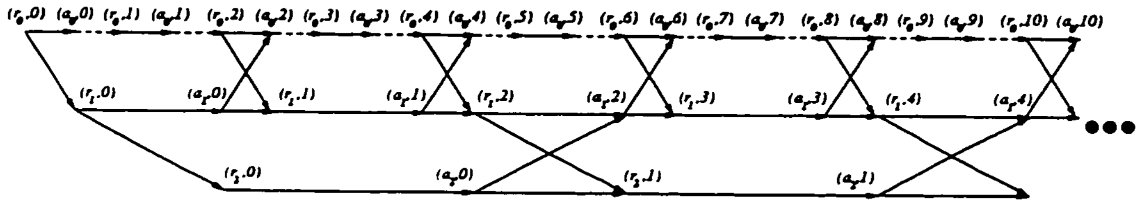

$(r_0,0)$ $(a_0,0)$ $(r_0,1)$ $(a_0,1)$ $(r_0,2)$ $(a_0,2)$ $(r_0,3)$ $(a_0,3)$ $(r_0,4)$ $(a_0,4)$ $(r_0,5)$ $(a_0,5)$ $(r_0,6)$ $(a_0,6)$ $(r_0,7)$ $(a_0,7)$ $(r_0,8)$ $(a_0,8)$ $(r_0,9)$ $(a_0,9)$ $(r_0,10)$ $(a_0,10)$
$(r_1,0)$ $(a_1,0)$ $(r_1,1)$ $(a_1,1)$ $(r_1,2)$ $(a_1,2)$ $(r_1,3)$ $(a_1,3)$ $(r_1,4)$ $(a_1,4)$
$(r_2,0)$ $(a_2,0)$ $(r_2,1)$ $(a_2,1)$
●●●

Figure 3.4: Part of an unfolded process graph for a linear array of three cells, i.e., length $L = 2$, with multiplication factor $n = 2$. The vertices are labelled with occurrence indices.

This theorem says that if acknowledgment (a_i, j_i) has a response depth of $k, i+1 \leq i+k \leq L$, then the duration of the n^k handshake cycles at interface i that end in (a_i, j_i) is at most $\Delta_{i+k,r} + \Delta_{i+k-1,f} + \sum_{h=0}^{k-1}(\Delta_{i+h,r} - \delta_{i+h,r})$. The factor n^k in the above equation will prove useful, because we give a linear bound to the duration between two events that have an exponential number of events between them if $n > 1$. Note also that the only term that depends on the length of the pipeline, i.e., the summation term, only refers to reverse latencies. We will return to this later.

Here is a brief example of the main theorem. Suppose the multiplication factor is $n = 2$. the response depth is $k = 2$, and the pipeline length is $L = 2$, and that we are looking at event a_0 so $i = 0$. Refer to Figure 3.4. Let us choose $j_0 = 10$. Therefore we wish to show that

$$\begin{aligned}
T(a_0, 10) - T(a_0, 10 - 2^2) &\leq \Delta_{i+k,r} + \Delta_{i+k-1,f} + \sum_{h=0}^{k-1}(\Delta_{i+h,r} - \delta_{i+h,r}) \\
T(a_0, 10) - T(a_0. 10 - 4) &\leq \Delta_{0+2,r} + \Delta_{0+2-1,f} + \sum_{h=0}^{2-1}(\Delta_{0+h,r} - \delta_{0+h,r}) \\
T(a_0, 10) - T(a_0, 6) &\leq \Delta_{2,r} + \Delta_{1,f} + \sum_{h=0}^{1}(\Delta_{h,r} - \delta_{h,r}) \\
T(a_0, 10) - T(a_0, 6) &\leq \Delta_{2,r} + \Delta_{1,f} + (\Delta_{1,r} - \delta_{1,r}) + (\Delta_{0,r} - \delta_{0,r})
\end{aligned}$$

Since the response depth $k = 2$, we know that $(a_0, 10)$ is dependent on an acknowledgment 2 levels deeper in the pipeline, viz., $(a_2, 1)$. So, tracing further back, $(a_0, 10)$ must be dependent on $(r_2, 1)$. In turn $(r_2, 1)$ is dependent on both $(r_1, 2)$ and $(a_2, 0)$. Suppose for the moment that the critical edge is $(a_2, 0) \rightarrow (r_2, 1)$. The path from $(a_2, 0)$ to $(a_0, 10)$ consists of the following four edges: $(a_2, 0) \rightarrow (r_2, 1)$, $(r_2, 1) \rightarrow (a_2, 1)$, $(a_2, 1) \rightarrow (a_1, 4)$, and $(a_1, 4) \rightarrow (a_0, 10)$. Maximising this path means assigning the following delays respectively for each edge: $\Delta_{1,f}$,$\Delta_{2,r}$,$\Delta_{1,r}$, and $\Delta_{0,r}$. Notice that edges going up have a reverse delay while edges going down have a forward delay. The horizontal edges alternate between forward and reverse delays. The path from $(a_2, 0)$ to $(a_0, 6)$ consists of the following two edges: $(a_2, 0) \rightarrow (a_1, 2)$, and $(a_1, 2) \rightarrow (a_0, 6)$. Minimising this path means assigning the following delays respectively for each edge: $\delta_{1,r}$, and $\delta_{0,r}$. Then,

$$\begin{aligned}
T(a_0, 10) - T(a_0, 10 - 2^2) \leq\ & (T(a_2, 0) + \Delta_{1,f} + \Delta_{2,r} + \Delta_{1,r} + \Delta_{0,r}) \\
& -(T(a_2, 0) + \delta_{1,r} + \delta_{0,r})
\end{aligned}$$

which is equivalent to

$$T(a_0, 10) - T(a_0, 6) \leq \Delta_{2,r} + \Delta_{1,f} + (\Delta_{1,r} - \delta_{1,r}) + (\Delta_{0,r} - \delta_{0,r})$$

If the critical edge is $(r_1, 2) \rightarrow (r_2, 1)$ the same result can be derived.

Before we prove the main theorem, we give another lemma. The lemma says that if an acknowledgment to a request to cell k depends only on cell k itself and not on cells $k+1, \ldots L$, then the only delay incurred is the delay of cell k.

Lemma 3.2 *For any $k, 0 \leq k \leq L$, if the response depth of (a_k, j_k) is 0, then*

$$T(a_k, j_k) - T(r_k, j_k) \leq \Delta_{k,r}$$

Proof. In any behaviour B of a pipeline with the given restrictions, there are at most two direct dependencies that end in (a_k, j_k), viz.

$$(r_k, j_k) \rightarrow (a_k, j_k) \text{ and possibly } (a_{k+1}, j_{k+1}) \rightarrow (a_k, j_k)$$

If the response depth of (a_k, j_k) is 0, then either the dependency $(a_{k+1}, j_{k+1}) \rightarrow (a_k, j_k)$ does not exist or, if it does exist,

$$T(a_k, j_k) - T(a_{k+1}, j_{k+1}) > \Delta_{k,r}$$

This is equivalent to saying that the edge $(a_{k+1}, j_{k+1}) \rightarrow (a_k, j_k)$, if it exists, is not a critical edge. Since each node must have at least one incoming critical edge, the edge $(r_k, j_k) \rightarrow (a_k, j_k)$ is a critical edge. Furthermore, we observe that the delay through any edge of the form $(r_k, j_k) \rightarrow (a_k, j_k)$ is not an environment delay. Hence, by our definition of the timing model, the delay through edge $(r_k, j_k) \rightarrow (a_k, j_k)$

must be bounded above by $\Delta_{k,r}$ since it is a "reverse" delay.
□

We can now prove the main lemma.

Proof of Lemma 3.1. By induction on k.
Basis. Assume that (a_i, j_i) has a response depth of $k = 1$. Let $(a_{i+1}, j_{i+1}) \rightarrow (a_i, j_i)$; then (a_{i+1}, j_{i+1}) has response depth 0. We have a (portion of a) dependency graph as in Figure 3.5 which we can derive from the behavioural model.

First we deal with the statement that event $(a_s, j_s - n^k)$ exists. Let us look at the behaviour of a cell at level i.

$$(r_i?;\ *[\ (a_i!;\ r_i?)^n \| (r_{i+1}!;\ a_{i+1}?)\ ])$$

If (a_i, j_i) exists and has a response depth greater than 0 then it must depend on an acknowledgment from a deeper level. Therefore, from the behaviour given above, there must be at least n handshakes at level i before the first synchronisation with an acknowledgment from level $i + 1$. Hence, (a_i, j_i) must be at least the $n + 1$st occurrence of a_i. Therefore $(a_i, j_i - n)$ exists. This is illustrated in Figure 3.5.

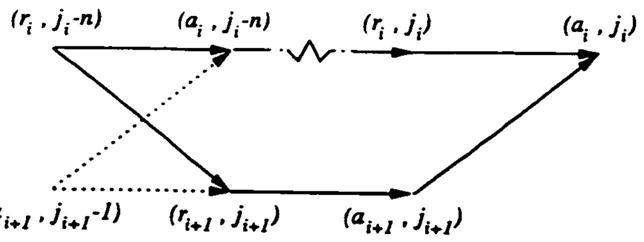


Figure 3.5: Example of paths used in the base case. The jagged line consists of $2n - 1$ edges.

We now move onto the central claim. From the definition of $T_{pred}()$ and the dependency graph in Figure 3.5, we have $T_{pred}(r_{i+1}, j_{i+1}) = T_{pred}(a_i, j_i - n)$. In the proof below we take $T_{pred} = T_{pred}(a_i, j_i - n)$. We wish to derive an upper bound for $T(a_i, j_i) - T(a_i, j_i - n)$. We observe

$T(a_i, j_i) - T(a_i, j_i - n)$

$=$ { Break path into 4 edges. }

$(T(a_i, j_i) - T(a_{i+1}, j_{i+1})) + (T(a_{i+1}, j_{i+1}) - T(r_{i+1}, j_{i+1})) + (T(r_{i+1}, j_{i+1}) - T_{pred}) + (T_{pred} - T(a_i, j_i - n))$

$\leq$ { (a_i, j_i) has positive response depth so timing dependency exists. Also, ack/ack pair is a "reverse" delay. }

$\Delta_{i,r} + (T(a_{i+1}, j_{i+1}) - T(r_{i+1}, j_{i+1})) + (T(r_{i+1}, j_{i+1}) - T_{pred}) + (T_{pred} - T(a_i, j_i - n))$

$\leq$ { (a_{i+1}, j_{i+1}) has resp. depth 0, Lemma 3.2 }

$\Delta_{i,r} + \Delta_{i+1,r} + (T(r_{i+1}, j_{i+1}) - T_{pred}) + (T_{pred} - T(a_i, j_i - n))$

$=$ { $T_{pred} = T_{pred}(a_i, j_i - n) = T_{pred}(r_{i+1}, j_{i+1})$ }

$\Delta_{i,r} + \Delta_{i+1,r} + (T(r_{i+1}, j_{i+1}) - T_{pred}(r_{i+1}, j_{i+1})) + (T_{pred} - T(a_i, j_i - n))$

$\leq$ { From definition of timing assignments and because edges ending in request are forward delays }

$\Delta_{i,r} + \Delta_{i+1,r} + \Delta_{i,f} + (T_{pred} - T(a_i, j_i - n))$

$\leq$ { Minimum edge delay is $\delta_{i,r}$ }

$\Delta_{i,r} + \Delta_{i+1,r} + \Delta_{i,f} + (-\delta_{i,r})$

$=$ { calc. }

$\Delta_{i+1,r} + \Delta_{i,f} + (\Delta_{i,r} - \delta_{i,r})$

$=$ { calc., $k = 1$ }

$\Delta_{i+k,r} + \Delta_{i+k-1,f} + \sum_{h=0}^{k-1}(\Delta_{i+h,r} - \delta_{i+h,r})$

Step. Assume the theorem holds for $k > 0$ and that (a_i, j_i) has response depth $k+1$. Then there is a node (a_{i+1}, j_{i+1}) such that dependency $(a_{i+1}, j_{i+1}) \rightarrow (a_i, j_i)$ exists and (a_{i+1}, j_{i+1}) has response depth k. From the induction hypothesis it then follows that node $(a_{i+1}, j_{i+1} - n^k)$ exists. Given that node $(a_{i+1}, j_{i+1} - n^k)$ and dependency $(a_{i+1}, j_{i+1}) \rightarrow (a_i, j_i)$ exist, we show that node $(a_i, j_i - n^{k+1})$ and dependency $(a_{i+1}, j_{i+1} - n^k) \rightarrow (a_i, j_i - n^{k+1})$ also exist and that we can depict the dependencies only for levels i and $i+1$ as in Figure 3.6.

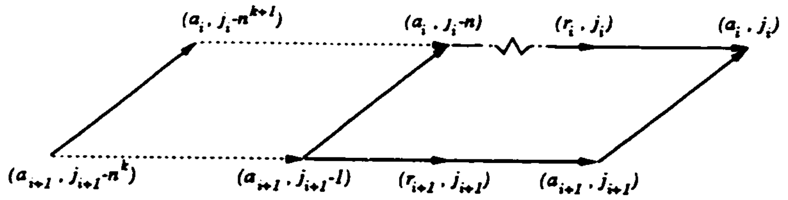

$(a_i, j_i\text{-}n^{k+1})$
$(a_i, j_i\text{-}n)$
(r_i, j_i)
(a_i, j_i)
$(a_{i+1}, j_{i+1}\text{-}n^k)$
$(a_{i+1}, j_{i+1}\text{-}1)$
(r_{i+1}, j_{i+1})
(a_{i+1}, j_{i+1})

Figure 3.6: A part of the behaviour graph at levels i and $i+1$. The jagged line consists of $2n-1$ edges. Dotted lines represent paths.

Let us look at the behaviour of a cell at level i.

$(r_i?;\ *[\,(a_i!;\ r_i?)^n \| (r_{i+1}!;\ a_{i+1}?)\,])$

From this behaviour it follows that between any two successive edges $a_{i+1} \rightarrow a_i$ there are n edges $a_i \rightarrow r_i$. It follows that dependency $(a_{i+1}, j_{i+1} - 1) \rightarrow (a_i, j_i - n)$

exists. Repeating this step n^k times yields the dependency graph above with direct dependency $(a_{i+1}, j_{i+1} - n^k) \to (a_i, j_i - n^{k+1})$.

Note that the above proof also gives us the existence of $(a_i, j_i - n^{k+1})$.

Now we observe

$T(a_i, j_i) - T(a_i, j_i - n^{k+1})$

$=$ { Break path into three: edge, path, edge. }

$((T(a_i, j_i)) - T(a_{i+1}, j_{i+1})) + (T(a_{i+1}, j_{i+1}) - T(a_{i+1}, j_{i+1} - n^k)) + (T(a_{i+1}, j_{i+1} - n^k) - T(a_i, j_i - n^{k+1}))$

$\leq$ { (a_{i+1}, j_{i+1}) has resp. depth k, induction hyp. }

$((T(a_i, j_i)) - T(a_{i+1}, j_{i+1})) + \Delta_{i+1+k,r} + \Delta_{i+1+k-1,f} + \sum_{h=0}^{k-1}(\Delta_{i+1+h,r} - \delta_{i+1+h,r}) + (T(a_{i+1}, j_{i+1} - n^k) - T(a_i, j_i - n^{k+1}))$

$\leq$ { (a_i, j_i) has positive response depth, and ack/ack pair is a reverse latency }

$\Delta_{i,r} + \Delta_{i+k+1,r} + \Delta_{i+k,f} + \sum_{h=1}^{k}(\Delta_{i+h,r} - \delta_{i+h,r}) + (T(a_{i+1}, j_{i+1} - n^k) - T(a_i, j_i - n^{k+1}))$

$\leq$ { $(a_{i+1}, j_{i+1} - n^k) \to (a_i, j_i - n^{k+1})$ exist, minimum edge delay is $\delta_{i,r}$ }

$\Delta_{i,r} + \Delta_{i+k+1,r} + \Delta_{i+k,f} + \sum_{h=1}^{k}(\Delta_{i+h,r} - \delta_{i+h,r}) + (-\delta_{i,r})$

$\leq$ { calc. }

$\Delta_{i+k+1,r} + \Delta_{i+k,f} + \sum_{h=0}^{k}(\Delta_{i+h,r} - \delta_{i+h,r})$

$\square$

3.3 Worst-Case Response Time

The worst-case response time is useful if we are comparing an asynchronous circuit against a clocked circuit. It is also useful should we wish to interface an asynchronous circuit with a synchronous one. In this case we have to show that the asynchronous circuit will be ready with a result within the time allowed, even if the worst case occurs.

With the preceding theorem, we have now done most of the work to find an upper bound for worst-case response time. We divide the calculation into L cases depending on the depth from which the critical path of acknowledgments originates.

We have the following theorem.

Theorem 3.3 *The worst-case response time RT for any pipeline (as defined in Section 2.3 operating under a delay model of Section 2.6) is bounded from above as*

$$\begin{aligned} RT \ \leq \ & \max(\{\Delta_{k,r} + \Delta_{k-1,f} + \sum_{h=0}^{k-1}(\Delta_{h,r} - \delta_{h,r}) \\ & -(\delta_{0,r} + \delta_e)n^k + \delta_{0,r} \mid 1 \leq k \leq L\} \cup \{\Delta_{0,r}\}) \end{aligned}$$

Proof. We have to find an upper bound for $T(a_0, j) - T(r_0, j)$ over all $j \geq 0$ and all valid delay distributions. Each acknowledgment has some response depth, say k, where $0 \leq k \leq L$. For a response depth $k = 0$, we have from Lemma 3.2, $T(a_0, j) - T(r_0, j) \leq \Delta_{0,r}$. For a response depth of $k > 0$, we use the result of the previous Section. We observe

$$T(a_0, j) - T(r_0, j)$$

$$= \quad \{ \text{ calc. } \}$$

$$(T(a_0, j) - T(a_0, j - n^k)) + (T(a_0, j - n^k) - (T(r_0, j))$$

$$\leq \quad \{\ (a_0, j) \text{ has response depth } k, \text{ Lemma 3.1, } i = 0\ \}$$

$$\Delta_{k,r} + \Delta_{k-1,f} + \textstyle\sum_{h=0}^{k-1}(\Delta_{h,r} - \delta_{h,r}) + (T(a_0, j - n^k) - (T(r_0, j))$$

Notice that there are n^k handshakes at level 0 between occurrences (a_0, j) and $(a_0, j - n^k)$. The handshakes at level 0 experience delays through the environment and through cell 0. Assuming that the environment's minimum delay is δ_e for edge $a_0 \to r_0$ and the minimum delay of cell 0 for a reverse latency edge $r_0 \to a_0$ is $\delta_{0,r}$, the minimum duration of one cycle through environment and cell is $\delta_{0,r} + \delta_e$. Therefore the minimum duration from $T(a_0, j - n^k)$ to $T(r_0, j)$ is $(\delta_{0,r} + \delta_e)n^k - \delta_{0,r}$. This then leads to the inequality

$$T(a_0, j) - T(r_0, j) \leq \Delta_{k,r} + \Delta_{k-1,f} + \sum_{h=0}^{k-1}(\Delta_{h,r} - \delta_{h,r}) - (\delta_{0,r} + \delta_e)n^k + \delta_{0,r}$$

Maximising these upper bounds over all $k, 0 \leq k \leq L$ gives the desired result.
□

3.3.1 Simplifying the Bound

The bound above cannot be simplified further without further assumptions about the variables.

Let us look at the $n = 1$ case.

$$\begin{aligned} RT \;&\leq\; \max(\{\Delta_{k,r} + \Delta_{k-1,f} + \sum_{h=0}^{k-1}(\Delta_{h,r} - \delta_{h,r}) \\ &\qquad -(\delta_{0,r} + \delta_e)\cdot 1 + \delta_{0,r} \mid 1 \leq k \leq L\} \cup \{\Delta_{0,r}\}) \\ &=\; \max(\{\Delta_{k,r} + \Delta_{k-1,f} + \sum_{h=0}^{k-1}(\Delta_{h,r} - \delta_{h,r}) - \delta_e \mid 1 \leq k \leq L\} \cup \{\Delta_{0,r}\}) \end{aligned}$$

We maximise over $L+1$ possibilities. One possibility corresponds to a path through cell 0 and gives us the delay $\Delta_{0,r}$. The other L cases correspond to a response depth of k, where $1 \leq k \leq L$. Each case consists of the delays $\Delta_{k,r} + \Delta_{k-1,f}$, a summation term $\sum_{h=0}^{k-1}(\Delta_{h,r} - \delta_{h,r})$, and an environment delay term $-\delta$. Notice that the summation term depends only on the reverse latencies. As k gets larger the summation term can only get bigger, since $\Delta_{h,r} \geq dhr$ for all h. The bound always depends on the cycle time between cells k and $k-1$. Consequently, the bound becomes smaller if we put the adjacent cells with the slowest cycles, i.e., $\Delta_{k,r} + \Delta_{k-1,f}$, near the entrance of the pipeline, as this is where the summation term is minimised.

In many implementations the variation among cell delays is only in the forward latencies. The reverse latencies may be fixed, i.e., $\Delta_{h,r} = \delta_{h,r}$ for $0 \leq h \leq L$. Consider a pipeline with combinational logic (for example, arithmetic logic) between the latches. There may very well be a variation in delay in the forward path, since the delay of the combinational logic may have variations and the delay of the forward path matches this delay. The reverse path simply signals an acknowledgement and is not matched with any computations. Thus variation in the delay of the reverse path are less likely. With this assumption the summation term disappears and the result is independent of the length of the pipeline. Thus,

$$\begin{aligned}
RT &\leq \max(\{\Delta_{k,r} + \Delta_{k-1,f} + \sum_{h=0}^{k-1}(0) - \delta_e \mid 1 \leq k \leq L\} \cup \{\Delta_{0,r}\}) \\
&= \max(\{\Delta_{k,r} + \Delta_{k-1,f} - \delta_e \mid 1 \leq k \leq L\} \cup \{\Delta_{0,r}\}) \\
&= \max(\{\Delta_{k,r} + \Delta_{k-1,f} \mid 1 \leq k \leq L\} \cup \{\Delta_{0,r} + \delta_e\}) - \delta_e \\
&= MC_{\delta_e} - \delta_e
\end{aligned}$$

In other words, in a linear pipeline with no variation in reverse latencies per cell, the worst-case response time is at most the maximum cycle time between adjacent cells minus the minimum environment delay.

Now, we can look at the $n > 1$ case. To simplify the formula we restrict the possible delay ranges of the maximum forward and reverse latencies of each cell. We assume that $\Delta_{h,r} \leq \Delta_{0,r} \cdot n^h$ and $\Delta_{h-1,f} \leq \delta_e \cdot n^h$ where $h \geq 1$. Each cell has maximum delays that are exponential in the depth of the cell in the pipeline. That is, cell $i+1$ can be up to n times slower than cell i and the bound still holds. In most designs this will be a reasonable assumption. It is hard to imagine the delay of cells getting more than exponentially bigger the further one gets into the pipeline.

To simplify the formula further, we again assume that there is no variation in reverse latencies, i.e., $\Delta_{h,r} = \delta_{h,r}$ for all $0 \leq h \leq L$.

$$\begin{aligned}
RT &\leq \max(\{\Delta_{k,r} + \Delta_{k-1,f} + \sum_{h=0}^{k-1}(0) - (\delta_{0,r} + \delta_e)n^k + \delta_{0,r} \mid 1 \leq k \leq L\} \cup \{\Delta_{0,r}\}) \\
&\quad \{\text{Using } \Delta_{h,r} \leq \Delta_{0,r} \cdot n^h\} \\
&\leq \max(\{\Delta_{0,r} \cdot n^k + \Delta_{k-1,f} - (\delta_{0,r} + \delta_e)n^k + \delta_{0,r} \mid 1 \leq k \leq L\} \cup \{\Delta_{0,r}\}) \\
&\quad \{\text{Using } \delta_{0,r} = \Delta_{0,r}\} \\
&= \max(\{\Delta_{0,r} \cdot n^k + \Delta_{k-1,f} - (\Delta_{0,r} + \delta_e)n^k + \Delta_{0,r} \mid 1 \leq k \leq L\} \cup \{\Delta_{0,r}\}) \\
&= \max(\{\Delta_{k-1,f} - \delta_e \cdot n^k + \Delta_{0,r} \mid 1 \leq k \leq L\} \cup \{\Delta_{0,r}\}) \\
&\quad \{\text{Using } \Delta_{h-1,f} \leq \delta_e \cdot n^h\} \\
&\leq \max(\{0 + \Delta_{0,r} \mid 1 \leq k \leq L\} \cup \{\Delta_{0,r}\}) \\
&= \Delta_{0,r}
\end{aligned}$$

This is a tight bound, because $\Delta_{0,r}$ is always a possible worst-case response time when the response depth is zero. That is, cell 0 alone is responsible for the delay.

Note that we can simplify the formula in different ways by making different assumptions. The assumptions $\Delta_{h,r} \leq \Delta_{0,r} \cdot n^h$, $\Delta_{h-1,f} \leq \delta_e \cdot n^h$, and $\Delta_{h,r} = \delta_{h,r}$ seem reasonable and give a tight bound. With these assumptions one can say that if the reverse latencies have no variance (i.e., $\Delta_{h,r} = \delta_{h,r}$) the delay in the first cell is the only one that matters. The rest of the pipeline can be extremely slow and may be optimised for power consumption or area.

3.3.2 Tightness of the Bound

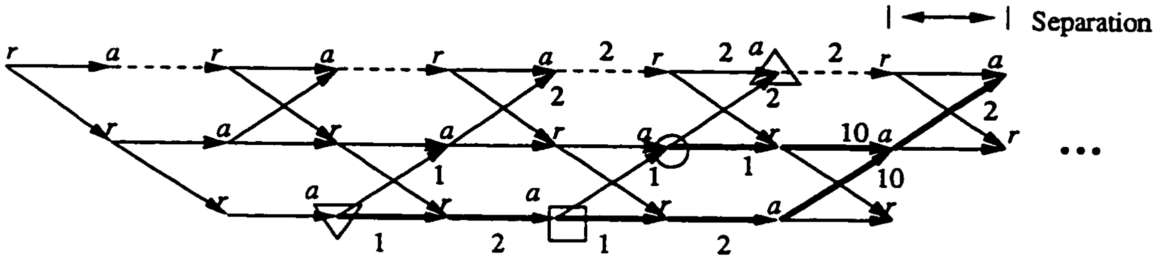


Figure 3.7: Dependency graph of a pipeline with multiplication factor $n = 1$ and length $L = 2$. Maximised paths are in bold. All other edges are minimised. Four events of interest are labelled by a $\triangle$, $\square$, $\triangledown$, and $\bigcirc$ respectively. The request/acknowledgment pair that we use to calculate the response-time separation is also marked.

The bound given in Theorem 3.3 may not necessarily be tight. For example, the proof of Lemma 3.1 relies on the property that certain edges have their minimum delay $\delta_{h,r}$. If these edges are critical edges then the bound is tight. In particular cases, however, these edges cannot be critical. As an example, consider the graph in Figure 3.7 and with bounds described by Table 3.1. Our formula for worst-case response time can be determined from calculations with reference to points $\bigcirc$ and

h	$\delta_{h,f}$	$\Delta_{h,f}$	$\delta_{h,r}$	$\Delta_{h,r}$
e	2	2	$--$	$--$
0	1	1	2	2
1	1	1	1	10
2	$--$	$--$	2	2

Table 3.1: Delays for a pipeline of length $L = 2$.

□. For each $1 \leq k \leq L$ we calculate

$$\Delta_{k,r} + \Delta_{k-1,f} + \sum_{h=0}^{k-1}(\Delta_{h,r} - \delta_{h,r}) - (\delta_{0,r} + \delta_e) \cdot n^k + \delta_{0,r}$$

We have $n = 1$. Therefore we can simplify the formula to

$$\Delta_{k,r} + \Delta_{k-1,f} + \sum_{h=0}^{k-1}(\Delta_{h,r} - \delta_{h,r}) - \delta_e$$

At point ○ we have $k = 1$.

$$\begin{aligned}\Delta_{k,r} + \Delta_{k-1,f} + \sum_{h=0}^{k-1}(\Delta_{h,r} - \delta_{h,r}) - \delta_e &= \Delta_{1,r} + \Delta_{0,f} + (\Delta_{0,r} - \delta_{0,r}) - \delta_e \\ &= 10 + 1 + (2 - 2) - 2 \\ &= 9\end{aligned}$$

At point □ we have $k = 2$.

$$\begin{aligned}\Delta_{k,r} + \Delta_{k-1,f} + \sum_{h=0}^{k-1}(\Delta_{h,r} - \delta_{h,r}) - \delta_e &= \Delta_{2,r} + \Delta_{1,f} + (\Delta_{1,r} - \delta_{1,r}) \\ &\quad +(\Delta_{0,r} - \delta_{0,r}) - \delta_e \\ &= 2 + 1 + (10 - 1) + (2 - 2) - 2 \\ &= 3 + (9) - +(0) - 2 \\ &= 10\end{aligned}$$

So, according to our formula, the bound is

$$\begin{aligned} RT &\leq \max(\{\Delta_{k,r} + \Delta_{k-1,f} + \sum_{h=0}^{k-1}(\Delta_{h,r} - \delta_{h,r}) \\ &\quad -(\delta_{0,r} + \delta_e)n^k + \delta_{0,r} \mid 1 \leq k \leq L\} \cup \{\Delta_{0,r}\}) \\ &= \max(\{9, 10\} \cup \{2\}) \\ &= 10 \end{aligned}$$

The formula for worst-case response time gives a tight bound in this example if the path between □ and △ is a critical path and the delays of the edges on that path are minimised. Here, however, no matter how we choose the delays on the path $\nabla \leadsto \Box$, the path $\nabla \leadsto \Box \leadsto \triangle$ is not critical. In the diagram, we see that there is an alternative path $\nabla \leadsto \triangle$ consisting of the path $a_2 \leadsto a_1 \leadsto a_0 \leadsto r_0 \leadsto a_0$. The edges on this path are minimised in the diagram. The delay on this path (i.e., $1+2+2+2 = 7$) is as short as it can be, yet it is the critical path $\nabla \leadsto \triangle$. Because of the choice of delays, the paths $\bigcirc \leadsto \triangle$ and $\Box \leadsto \triangle$ cannot be critical paths as assumed.

This contradiction does not prevent our formula from giving a bound, it is just that in some circumstances, such as this cooked up example, the bound may not be tight.

3.4 Worst-case Cycle Time

We can also calculate bounds on worst-case cycle time of a pipeline. Response time was the separation between a request and the corresponding acknowledgment. Cycle time is the time between consecutive requests, i.e., the separation

$$T(r_0, i+1) - T(r_0, i)$$

Obviously the delay of the environment $(T(r_0, i+1) - T(a_0, i))$ is the crucial difference between response time and cycle time.

Theorem 3.4 *The worst-case cycle time CT for any pipeline (as defined in Section 2.3 operating under a delay model of Section 2.6) is bounded from above as*

$$\begin{aligned} CT &\leq \Delta_e + \max(\{\Delta_{k,r} + \Delta_{k-1,f} + \sum_{h=0}^{k-1}(\Delta_{h,r} - \delta_{h,r}) \\ &\quad -(\delta_{0,r} + \delta_e)n^k + \delta_{0,r} \mid 1 \leq k \leq L\} \cup \{\Delta_{0,r}\}) \end{aligned}$$

Proof. This upper bound is the same as the bound for RT plus one maximum environment delay Δ_e. The cycle time CT between $(r_0, i) \rightarrow (r_0, i+1)$ is calculated by maximising the delay on $(r_0, i) \rightarrow (a_0, i)$. The delay on edge $(a_0, i) \rightarrow (r_0, i+1)$ is independent of other constraints and so has maximum delay Δ_e. See Figure 3.2. □

Given that an architecture has a handshaking behaviour at the environment, viz, the environment alternates between making requests and receiving acknowledgments, and given that we know the upper bound for the worst-case response time for that architecture we can make the following generalisation.

Lemma 3.5 *Given an upper bound for the response time, RT_{ub} for any architecture where the environment has a handshaking behaviour, an upper bound on the cycle-time between requests is*

$$CT_{ub} \leq RT_{ub} + \Delta_e$$

where Δ_e is the maximum delay of the environment.

Proof. We wish to find the maximum separation $T(r_0, i+1) - T(r_0, i)$ for some i. We note that due to the handshaking behaviour of the environment, $(r_0, i+1)$ has only one predecessor event, i.e. (a_0, i). This is the environment edge $(a_0, i) \rightarrow (r_0, i+1)$ and has maximum delay Δ_e. The separation $T(a_0, i) - T(r_0, i) \leq RT_{ub}$. The result follows immediately.

□

Note that we have chosen to define cycle time to be the delay between consecutive requests. We could also define cycle time to be the delay between consecutive acknowledgments. The worst case cycle time may be different in this case. This is formalised in the following lemma.

Lemma 3.6 *The cycle time CT defined between $(r_0, i) \rightarrow (r_0, i+1)$ is an upper bound for the cycle time, CT, defined between $(a_0, i) \rightarrow (a_0, i+1)$.*

Proof. Let us label the cycle time between $(a_0, i) \rightarrow (a_0, i+1)$ as CT_a. We can calculate an upper bound for CT_a in the same way as we did above. The edge $(a_0, i) \rightarrow (r_0, i+1)$ has maximum delay Δ_e. The edge $(r_0, i+1) \rightarrow (a_0, i+1)$ can be bounded by RT_{ub}. Summing these two delays gives us the same formula as for CT between $(r_0. i)$ and $(r_0, i+1)$. This is enough to show that the maximum cycle time between acknowledgments is no greater than the maximum cycle time between requests.

□

Note, the cycle time between acknowledgments might be less than between requests, because the calculation of the separation between $(r_0, i+1)$ and $(a_0, i+1)$ (i.e., RT_{ub}) relies on the delay of the previous edge. In other words we must minimise the delay on $(a_0, i-1) \rightarrow (r_0, i)$ to maximise the separation between (r_0, i) and

(a_0, i). If we minimise the delay on $(a_0, i-1) \to (r_0, i)$ this is obviously going to affect the delay $(a_0, i-1) \to (a_0, i)$ (i.e., CT_a).

3.5 An Example Micropipeline

Here we give an example of calculating the response and cycle times of a micropipeline. We refer back to the example given in Section 2.6.2 where we have a pipeline with $L = 3$. We instantiate the delay elements as in Table 3.2 and as in Figure 3.8.

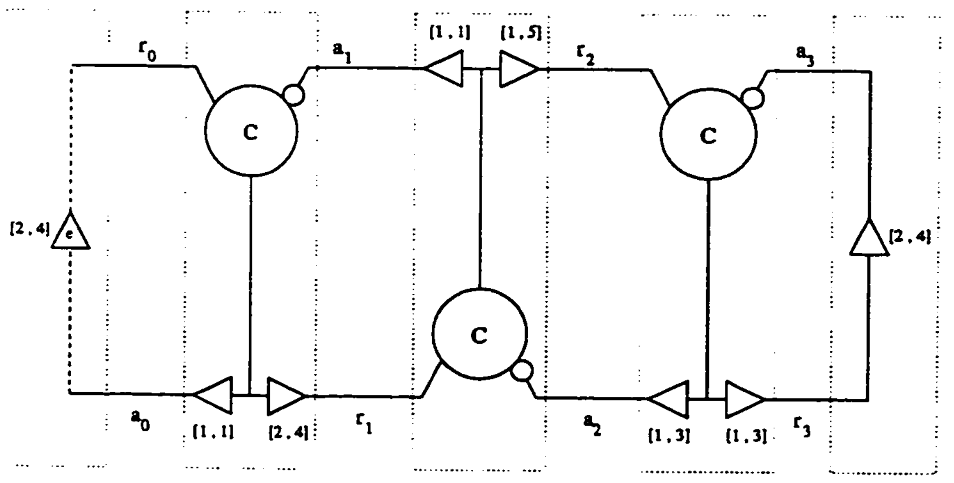


Figure 3.8: An example micropipeline with length $L = 3$. The environment delay is labelled with an e. The stages of our pipeline model are labelled in outline.

Suppose we wish to find the bounds on the response time, RT. For $n = 1$ we have the formula.

$$RT \leq \max(\{\Delta_{k,r} + \Delta_{k-1,f} + \sum_{h=0}^{k-1}(\Delta_{h,r} - \delta_{h,r}) - \delta_e \mid 1 \leq k \leq L\} \cup \{\Delta_{0,r}\})$$

h	$\delta_{h,f}$	$\Delta_{h,f}$	$\delta_{h,r}$	$\Delta_{h,r}$
e	2	4	--	--
0	2	4	1	1
1	1	5	1	1
2	1	3	1	3
3	--	--	2	4

Table 3.2: Delays for a pipeline of length $L = 3$.

We have to calculate the maximised term of the formula for all values of $1 \leq k \leq L$. For $k = 1$ we have

$$\Delta_{1,r} + \Delta_{0,f} + \sum_{h=0}^{1-1}(\Delta_{h,r} - \delta_{h,r}) - \delta_e \quad = \quad 3$$

For $k = 2$ we have

$$\Delta_{2,r} + \Delta_{1,f} + \sum_{h=0}^{2-1}(\Delta_{h,r} - \delta_{h,r}) - \delta_e \quad = \quad 6$$

For $k = 3$ we have

$$\Delta_{3,r} + \Delta_{2,f} + \sum_{h=0}^{3-1}(\Delta_{h,r} - \delta_{h,r}) - \delta_e \quad = \quad 7$$

Combining these results we have,

$$\begin{aligned} RT &\leq \max(\{\Delta_{k,r} + \Delta_{k-1,f} + \sum_{h=0}^{k-1}(\Delta_{h,r} - \delta_{h,r}) - \delta_e \mid 1 \leq k \leq L\} \cup \{\Delta_{0,r}\}) \\ &= \max(\{3, 6, 7\} \cup \{1\}) \\ &= 7 \end{aligned}$$

Suppose we wish to find the bounds on the cycle time, CT. We have the formula.

$$\begin{aligned} CT &\leq \Delta_e + \max(\{\Delta_{k,r} + \Delta_{k-1,f} + \sum_{h=0}^{k-1}(\Delta_{h,r} - \delta_{h,r}) - \delta_e \mid 1 \leq k \leq L\} \\ &\quad \cup\{\Delta_{0,r}\}) \\ &= 4 + 7 \\ &= 11 \end{aligned}$$

3.6 Conditional Behaviour

We can make a simple extension to the behaviour given in Chapter 2 by allowing conditional behaviour. Suppose that we have a behaviour of the form

$$(r_i?; *[\textbf{if } B_i \textbf{ then } (a_i!;\ r_i?)^n \,\|(r_{i+1}!;\ a_{i+1}?) \textbf{ else } (a_i!;\ r_i?)^n \textbf{ fi }])$$

The guard B_i may depend on data values that have been communicated. If $B_i =$ **false**, no handshake takes place with the right-hand neighbour.

This thesis strives to avoid probabilistic models; therefore we have chosen a method that avoids specifying the probability that a guard is true. All we assume is that each guard has the possibility of being true at some point, i.e., that $P(\{B_i = \textbf{true}\}) > 0\}$. Note that if any guard B_i is false all the time, we effectively have a shorter pipeline and we can calculate results using $L = \min\{i \mid P(\{B_i = \textbf{true}\}) = 0\}$. In any case, this situation will not happen in a useful design.

Conditional behaviours could be used to model an up-down counter. We can give linear-pipeline implementations of up-down counters [19] with the following behaviour. When incrementing the counter, carries may be propagated through the counter. When decrementing the counter, carries may have to be borrowed. The environment of the pipeline determines the sequence of increments and decrements that are performed, and hence determines how deep handshaking propagates into the pipeline. A different sequence of operations may produce a different set of handshake depths. We can model the depth to which handshaking propagates by the guards B_i.

We have the following theorem.

Theorem 3.7 *Given the behavioural model of Section 3.6, and given that the probability that a guard B_i is true is non-zero, upper bounds for RT and CT can be given by the upper bounds for non-conditional behaviour as given in Theorems 3.3 and 3.4.*

Proof. To prove the main theorem above we again prove our auxiliary lemma, i.e., we reprove Lemma 3.1. for the conditional behaviour. If we can prove our auxiliary lemma, then Theorems 3.3 and 3.4 that prove the upper bounds for *RT* and *CT* will also hold. Hence, if Theorems 3.3 and 3.4 hold for the conditional behaviour, then Theorem 3.7 above holds.

Lemma 3.8 *Given the behavioural model of Section 3.6, if the response depth of (a_i, j_i) is $k, i+1 \leq i+k \leq L$, then event $(a_i, j_i - n^k)$ exists, and*

$$T(a_i, j_i) - T(a_i, j_i - n^k) \leq \Delta_{i+k,r} + \Delta_{i+k-1,f} + \sum_{h=0}^{k-1}(\Delta_{i+h,r} - \delta_{i+h,r})$$

□

The proof is almost identical to the proof for Lemma 3.1. The only difference is the addition of a "slack" term that deals with the possibility that some guards may be false.

Proof of Lemma 3.8. By induction on k.

We note that the conditional behaviour strips edges from the behavioural graph when guards are false. This messes up the regular numbering of occurrence indices that we have in the normal behaviour. See Figures 3.9 and 3.10 for simple examples.

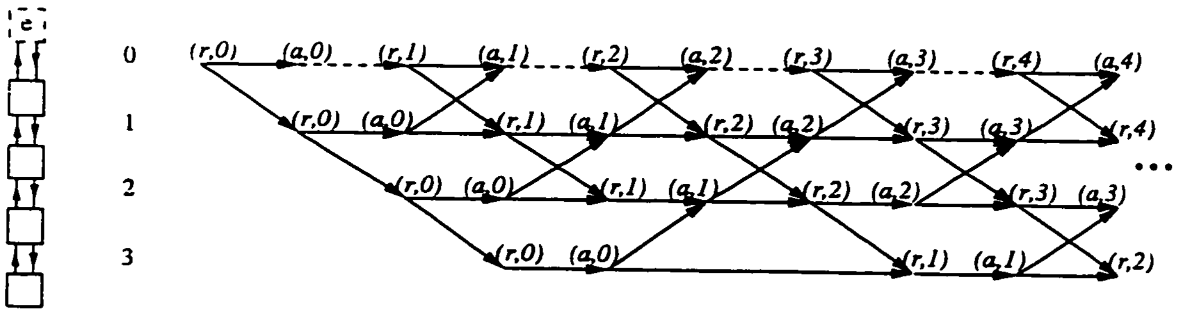


Figure 3.9: Conditional behaviour. Guard B_3 is false once.

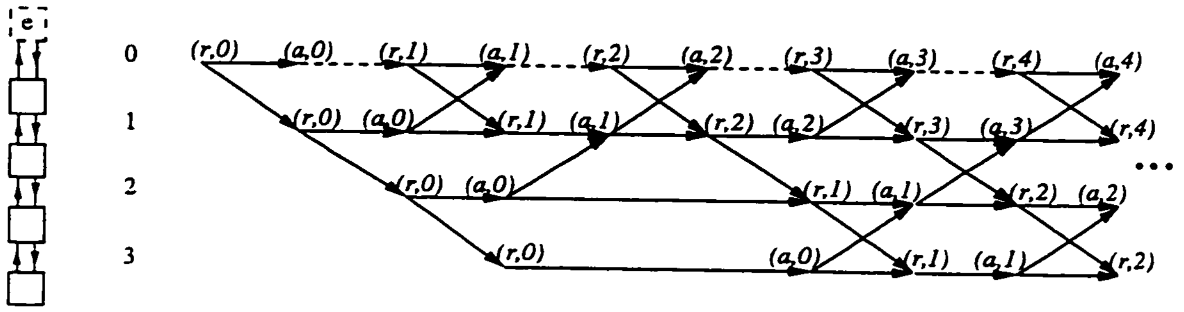


Figure 3.10: Conditional behaviour. Guard B_2 is false once.

Basis. Assume that (a_i, j_i) has a response depth of $k = 1$. Then edge $(a_{i+1}, j_{i+1}) \rightarrow (a_i, j_i)$ exists and (a_{i+1}, j_{i+1}) has response depth 0. See Figure 3.11. Let $x \geq n$ be some value that allows for the possibility of extra slack caused by guards being false. How big x is depends on the values of the guards. For example, look at level 1 and 2 of Figure 3.10. The first acknowledgment/acknowledgment edge is $(a_2, 0) \rightarrow (a_1, 1)$ and the second is $(a_2, 1) \rightarrow (a_1, 3)$. Note the changes in occurrence indices.

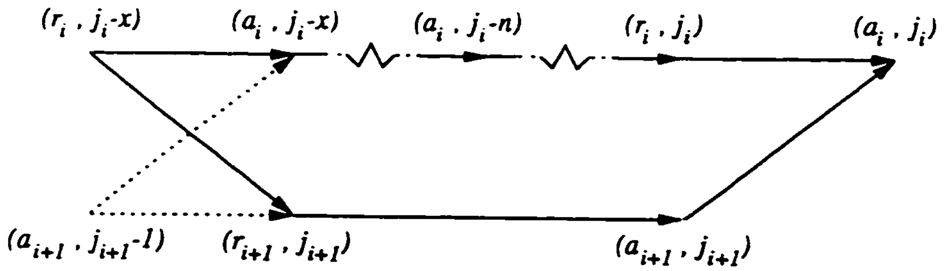


Figure 3.11: Example of paths used in the base case where $x \geq n$. The jagged lines are paths, possibly of zero length,

First we deal with the statement that event $(a_s, j_s - n)$ exists. Let us look at the behaviour of a cell at level i.

$$(r_i?; *[\textbf{if } B_i \textbf{ then } (a_i!;\ r_i?)^n \,\|(r_{i+1}!;\ a_{i+1}?) \textbf{ else } (a_i!;\ r_i?)^n \textbf{ fi }])$$

The reasoning proceeds as before. If (a_i, j_i) exists and has a response depth greater than 0 then it must depend on an acknowledgment from a deeper level. Therefore, from the behaviour given above, there must be at least n handshakes at level i before the first synchronisation with an acknowledgment from level $i+1$. Hence, (a_i, j_i) must be at least the $n+1$st occurrence of a_i. Therefore $(a_i, j_i - n)$ exists. This is illustrated in Figure 3.11.

We wish to derive an upper bound for $T(a_i, j_i) - T(a_i, j_i - n)$. We note that the critical paths between $(a_i, j_i - x)$ and (a_i, j_i), $x \geq n$, are identical to the critical paths between $(a_i, j_i - n)$ and (a_i, j_i) given in the original proof of Lemma 3.1.

Using the same proof we can show that

$$T(a_i, j_i) - T(a_i, j_i - x) \le \Delta_{i+k,r} + \Delta_{i+k-1,f} + \sum_{h=0}^{k-1}(\Delta_{i+h,r} - \delta_{i+h,r})$$

where $k = 1$. Since $x \ge n$ we trivially have that $T(a_i, j_i - x) \le T(a_i, j_i - n)$. Then

$$T(a_i, j_i) - T(a_i, j_i - n) \le \Delta_{i+k,r} + \Delta_{i+k-1,f} + \sum_{h=0}^{k-1}(\Delta_{i+h,r} - \delta_{i+h,r})$$

and so the base case is done.

Step. Assume the theorem holds for $k > 0$ and that (a_i, j_i) has response depth $k+1$. Then there is a node (a_{i+1}, j_{i+1}) such that dependency $(a_{i+1}, j_{i+1}) \to (a_i, j_i)$ exists and (a_{i+1}, j_{i+1}) has response depth k. From the induction hypothesis it then follows that node $(a_{i+1}, j_{i+1} - n^k)$ exists. Given that node $(a_{i+1}, j_{i+1} - n^k)$ and dependency $(a_{i+1}, j_{i+1}) \to (a_i, j_i)$ exist, we show that node $(a_i, j_i - x)$ and dependency $(a_{i+1}, j_{i+1} - n^k) \to (a_i, j_i - x)$ also exist where $x \ge n^{k+1}$. We can depict the dependencies, only for levels i and $i+1$, as in Figure 3.12.

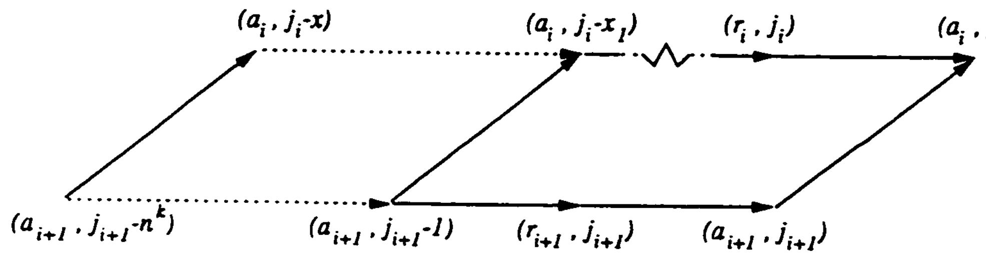


Figure 3.12: A part of the behaviour graph at levels i and $i+1$. The jagged line consists of at least $2n-1$ edges. Dotted lines represent paths. $x_1 \ge n$ and $x \ge n^k$.

Let us look at the behaviour of a cell at level i.

$(r_i?; *[\textbf{if } B_i \textbf{ then } (a_i!;\ r_i?)^n \,\|(r_{i+1}!;\ a_{i+1}?) \textbf{ else } (a_i!;\ r_i?)^n \textbf{ fi }])$

From this behaviour it follows that between any two successive edges $a_{i+1} \rightarrow a_i$ there are *at least* n edges $a_i \rightarrow r_i$. It follows that dependency $(a_{i+1}, j_{i+1}-1) \rightarrow (a_i, j_i - x_1)$ exists where $x_1 \geq n$. Repeating this step n^k times yields the dependency graph above with direct dependency $(a_{i+1}, j_{i+1} - n^k) \rightarrow (a_i, j_i - x)$ where $x = \sum_{l=1}^{n^k} x_l$ and $x_l \geq n$ for all $1 \leq l \leq n^k$. In other words, $x \geq n^{k+1}$. Again, how big x is depends on the values of the guards.

We now follow the same proof steps given in the original proof of Lemma 3.1 in Section 3.2 to calculate the separation between $T(a_i, j_i - x)$ and $T(a_i, j_i)$. The original proof calculated the separation between $T(a_i, j_i - n)$ and $T(a_i, j_i)$ but we can use the same steps to calculate the separation $T(a_i, j_i - x)$ and $T(a_i, j_i)$ here. Notice the similarity between Figure 3.12 and Figure 3.6. Using the previous method we have

$$T(a_i, j_i) - T(a_i, j_i - x) \leq \Delta_{i+k+1,r} + \Delta_{i+k,f} + \sum_{h=0}^{k} (\Delta_{i+h,r} - \delta_{i+h,r})$$

Since $x \geq n^{k+1}$ we trivially have that $T(a_i, j_i - n^{k+1}) \geq T(a_i, j_i - x)$. Then

$$T(a_i, j_i) - T(a_i, j_i - n^{k+1}) \leq \Delta_{i+k+1,r} + \Delta_{i+k,f} + \sum_{h=0}^{k} (\Delta_{i+h,r} - \delta_{i+h,r})$$

and so we are done.

□

Lastly we conclude our main proof for conditional behaviours.

Proof of Theorem 3.7. Theorems 3.3 and 3.4 hold for the conditional behaviour due to Lemma 3.8 given above. Theorem 3.7 therefore holds.

□

We will give an example of conditional behaviour in the stack example of the next chapter.

3.7 Summary

We have introduced two delay measures in this chapter, i.e., Response Time RT and Cycle Time CT. Worst-case response time is the worst-case delay between a request and a corresponding acknowledgment at the environment. Worst-case cycle time is the worst-case delay between consecutive requests. This chapter has proven the following results.

- Response time for pipelines, RT, is bounded as follows

$$\begin{aligned} RT &\leq \max(\{\Delta_{k,r} + \Delta_{k-1,f} + \sum_{h=0}^{k-1}(\Delta_{h,r} - \delta_{h,r}) \\ &\quad -(\delta_{0,r} + \delta_e)n^k + \delta_{0,r} \mid 1 \leq k \leq L\} \cup \{\Delta_{0,r}\}) \end{aligned}$$

  That is, response time is strongly dependent on the variation in the reverse delays, i.e, the summation term. If there is no variation in the reverse delays the term $\Delta_{k,r} + \Delta_{k-1,f}$ becomes a bottleneck. This value is the cycle time between consecutive cells.

- An upper bound for the cycle time of pipelines, CT_{ub}, is

$$CT_{ub} = RT_{ub} + \Delta_e$$

  In general, this formula holds for any behaviour with a handshaking environment. Therefore, if we can calculate an upper bound on the worst-case response time, we can always calculate an upper bound on the worst-case cycle time. Conversely, if we can calculate an upper bound on the worst-case cycle time, we can always calculate an upper bound on the worst-case response time.

- The bounds for worst-case response time and worst-case cycle time calculated for non-conditional behaviour are also bounds for the conditional behaviour described in Section 3.6.

Chapter 4

Bounds on Average Response Times

4.1 Average-Case Measures

In a previous work [16] "amortized" response time was described as an upper bound on the average response time. That is, over many handshake cycles at level 0 we calculate the maximum possible delay caused by the pipeline and divide this by the number of cycles. Over a large enough period of time the average response time will be no worse than the upper bound we calculate in this fashion.

In this chapter we also find a lower bound on the average response time by a similar technique. Averaged over many request/acknowledgment cycles we calculate the *minimum* possible delay caused by the pipeline. In contrast to these average-case measures, the response time measures of the previous chapter were calculated as the separation between a single request and its successive acknowledgment. Average response time, AR, is a measure related to the average throughput of a pipeline and may be more important than the worst-case response time. The change of terminology from amortized response time to average response time has been made for two reasons. Firstly, the word amortized may imply an upper bound only. Here we calculate lower bounds also. Secondly, the value of AR is some unknown average value. We only calculate bounds on AR. While the bounds on AR are calculated by amortized analysis, AR itself is never explicitly calculated and we do not wish to imply any method of calculation.

We can also compute bounds on the average cycle time AC. Average cycle time is calculated in the same way as average response time except that we do not subtract the environment delays. Unlike worst-case cycle time, CT, it is immaterial whether we measure the time between requests or between acknowledgments, since we are averaging over a potentially infinite number of cycles.

Obviously, an upper bound for the worst-case response time RT is also an upper bound for the average response time AR. An upper bound for the worst-case is clearly an upper bound for the average. It may be less clear why upper bounds for AR and RT could be different. The reason for this possible difference is as follows. The worst-case response time is measured over one particular request-to-acknowledgment edge. It may happen that we can only obtain a worst-case response time for that particular request-to-acknowledgment pair, if the delays of certain parts of the pipeline have been *minimised* previous to that request-to-acknowledgment edge. When we calculate the bounds for the average response time AR, any minimisation of delays somewhere is going to be reflected in the bound on the average, since we calculate the bound on AR over a potentially infinite period of time.

The bounds for average response time AR and average cycle time AC are easier to compute than the response-time bounds of the previous chapter. When calculating response time, RT, we have to find the time separation between two events by tracing the difference in lengths of two paths. On one of the two paths we take the maximum delays, and on the other we take the minimum delays. When computing bounds on the average, we only need to trace one path, and this path is maximised to obtain an upper bound, and minimised to obtain a lower bound. Since only one set of delays is used for each problem, i.e., minimal delays for computing the lower bound and maximal delays for computing the upper bound, it turns out we may simplify our model to a fixed delay model.

Not only are the results of this chapter simpler to compute, but the results are also simpler to express. The main results of this chapter are:

- The average response time, AR, for pipelines with multiplication factor, $n = 1$ is bounded as follows.

  $$mc_{\Delta_e} - \Delta_e \leq \; AR \; \leq MC_{\delta_e} - \delta_e$$

  where mc_{Δ_e} is the "minimum cycle time" of the pipeline with a slow environment given by

  $$mc_{\Delta_e} = \max\{\Delta_e + \delta_{0,r}, \delta_{0,f} + \delta_{1,r}, \delta_{1,f} + \delta_{2,r}, \ldots, \delta_{L-1,f} + \delta_{L,r}\}$$

  and where MC_{δ_e} is the "maximum cycle time" of the pipeline with a fast environment given by

  $$MC_{\delta_e} = \max\{\delta_e + \Delta_{0,r}, \Delta_{0,f} + \Delta_{1,r}, \Delta_{1,f} + \Delta_{2,r}, \ldots, \Delta_{L-1,f} + \Delta_{L,r}\}$$

  This bound is tight.

  The results tell us that the average response time of the pipeline is constrained by two independent bottlenecks that depend on the cycle time between neighbouring cells. Note that the bounds are independent of the length of the pipeline, and hence enormously deep pipelines do not affect our bounds except that we take the maximum over larger sets.

- The average cycle time AC for pipelines with multiplication factor $n = 1$ is bounded as follows.

  $$mc_{\delta_e} \leq AC \leq MC_{\Delta_e}$$

  where mc_{δ_e} is the "minimum cycle time" of the pipeline with a fast environment given by

  $$mc_{\delta_e} = \max\{\delta_e + \delta_{0,r}, \delta_{0,f} + \delta_{1,r}, \delta_{1,f} + \delta_{2,r}, \ldots, \delta_{L-1,f} + \delta_{L,r}\}$$

  and where MC_{Δ_e} is the "maximum cycle time" of the pipeline with a slow environment given by

  $$MC_{\Delta_e} = \max\{\Delta_e + \Delta_{0,r}, \Delta_{0,f} + \Delta_{1,r}, \Delta_{1,f} + \Delta_{2,r}, \ldots, \Delta_{L-1,f} + \Delta_{L,r}\}$$

  This bound is tight.

  Notice that this is a similar result to that given for average response time AR. Again, the bounds are independent of the depth of the pipeline.

- The average response time AR for pipelines with multiplication factor $n > 1$ is bounded as follows.

  $$\delta_{0,r} \leq AR \leq \max\{\Delta_{0,r}, \Delta_{0,r} + \Delta_{0,f} - \delta_e\}$$

  provided that $\Delta_{i,f} \leq \Delta_{0,f} \cdot n^i$ and $\Delta_{i,r} \leq \Delta_{0,r} \cdot n^i$ for $0 \leq i \leq L$. That is, a cell may be exponentially slower than cell 0 depending on its depth in the pipeline. The lower bound still applies if the conditions on the delays $\Delta_{i,f}$ and $\Delta_{i,r}$ do not hold.

Note that in most cases the delays of cell 0 and the environment are the only ones that are important. Because of this property, we can optimise cell 0 for speed and optimise cells deeper in the pipeline for power and area. For example. a designer may choose smaller transistors sizes for cells $1, \ldots, L$, thus trading speed for smaller area and power consumption. Or, if it is possible to scale down voltage, then cells $1, \ldots, L$ could be supplied with a lower voltage, again sacrificing speed for lower power consumption. In both cases the bounds on the average response time are based on the speed of cell 0 alone provided that $\Delta_{i,f} \leq \Delta_{0,f} \cdot n^i$ and $\Delta_{i,r} \leq \Delta_{0,r} \cdot n^i$ for $0 \leq i \leq L$.

- The average cycle time AC for pipelines with multiplication factor $n > 1$ is bounded as follows.

  $$\delta_{0,r} + \delta_e \leq \quad AC \quad \leq \Delta_{0,r} + \max\{\Delta_{0,f}, \Delta_e\}$$

  given $\Delta_{i,f} \leq \Delta_{0,f} \cdot n^i$ and $\Delta_{i,r} \leq \Delta_{0,r} \cdot n^i$ for $0 \leq i \leq L$. The lower bound still applies if the conditions on $\Delta_{i,f}$ and $\Delta_{i,r}$ do not hold. For this result a similar explanation can be given as for the previous result.

- Bounds for AR and AC are clearly related. We give the relationship between the two measures for the pipeline and tree architectures as discussed in this thesis. We derive bounds on AC by choosing a critical path between two level 0 events arbitrarily far apart. We then average the delay on this path over the number of handshakes at level 0. We calculate bounds on AR similarly except that we also subtract the environment delay. We call these unique bounds derived from the critical path "critical" bounds. We only define critical bounds for deterministic behaviours, i.e., those that have no conditional behaviour.

Let AR_{lb} and AR_{ub} be the critical lower bound and critical upper bound respectively for average response time. Let AC_{lb} and AC_{ub} be the critical lower bound and critical upper bound respectively for average cycle time. Let P_{lb} and P_{ub} be critical paths over which we derive the critical bounds for average cycle time AC_{lb} and AC_{ub}.

For upper bounds we have

$$AR_{ub} = AC_{ub} - \Delta_e$$

when P_{ub} is a level 0 path, i.e., P_{ub} contains edges only from cell 0 and the environment. We also have

$$AR_{ub} = AC_{ub} - \delta_e$$

when the level 0 part of P_{ub} is negligible. For lower bounds

$$AR_{lb} = AC_{lb} - \delta_e$$

when P_{lb} is a level 0 path, and

$$AR_{lb} = AC_{lb} - \Delta_e$$

when the level 0 part of P_{lb} is negligible. We do not deal with the case when P_{ub} or P_{lb} are split over segments at level 0, and segments not at level 0.

The formulae above allow us to calculate bounds on average cycle time from the bounds on average response time, and vice versa. This halves the number of proofs we have to do.

- Behaviour in which handshakes are conditionally passed deeper into the pipeline have the same worst-case bounds for AR and AC as those calculated for non-conditional behaviour. In conditional behaviour for pipelines, as defined in the previous chapter, each cell has a boolean guard that decides in each repetition step whether a handshake with a cell's right neighbour takes place or not.

4.2 Critical Bounds for Average Response Time and Cycle Time

Before calculating any average delay measures for pipelines, we formally describe our method for calculating tight bounds. We also show that for obtaining bounds on average response time and average cycle time, we can simplify our delay model to a model that uses fixed delays. We describe average cycle time first, since it is easier to explain.

4.2.1 Average Cycle Time

We define average cycle time to be the time between two events at level 0, arbitrarily far apart, divided by the number of handshakes at level 0 between the two events. For example, suppose that we are given a behavioural graph such as one of those given in Section 2.3 to describe the behaviour of a pipeline. To calculate the bounds on the average cycle time we find a critical path P between two events at level 0 in the graph. Since these events are arbitrarily far apart, we can choose events (r_0, i) and $(r_0, i+k)$, where $k \to \infty$. We find the delay on the critical path $P = (r_0, i) \rightsquigarrow (r_0, i+k)$ and divide by k, the number of handshakes at level 0 that occur between (r_0, i) and $(r_0, i+k)$. If we maximise the delays between (r_0, i) and $(r_0, i+k)$ we maximise the delay on path P. Then the above process gives us an upper bound on the average cycle time.

Let us formalise the method. Let P_k be a critical path between (r_0, i) and $(r_0, i+k)$ with maximum possible delay. Then,

$$AC_{ub} = \lim_{k \to \infty} \frac{\text{delay}(P_k)}{k} \tag{4.1}$$

We call AC_{ub} the *critical upper bound* for average cycle time, because it is derived from a critical path P_k of infinite length. We have that $AC \leq AC_{ub}$. The critical upper bound is the tightest possible bound and is therefore unique.

We can also calculate a lower bound on cycle time in similar fashion. Formally, let P_k be a critical path between (r_0, i) and $(r_0, i+k)$ with minimum possible delay. As above we derive,

$$AC_{lb} = \lim_{k \to \infty} \frac{\text{delay}(P_k)}{k} \tag{4.2}$$

We call AC_{lb} the *critical lower bound* for average cycle time. We have that $AC \geq AC_{lb}$. The critical lower bound is the tightest possible bound and is therefore unique. The formula is the same as for the upper bound, the only change is that we have minimised all delays between (r_0, i) and $(a_0, i+k)$ so that we minimise the delay on critical path P_k.

We note that our method of deriving critical bounds relies on knowing what the critical path is. Thus, although our method may apply to a wide variety of possible behaviours, these behaviours must be deterministic. If a behaviour is non-deterministic, we may not know what the critical path of a given execution will be. We have encountered non-deterministic behaviours in the conditional pipelines of Section 3.6. We will encounter these behaviours again in Section 4.9.

It is possible to simplify our delay model to a fixed delay model, when calculating bounds on average response time. This calculation of critical bounds is formalised in the following lemma.

Lemma 4.1 *The critical upper bound for average cycle time AC_{ub} for any deterministic behaviour can be obtained when we maximise all delays. The critical lower bound for average cycle time AC_{lb} for any deterministic behaviour can be obtained when we minimise all delays.*

Proof. Consider the calculation of AC_{ub}. We need to choose a critical path with maximum delay P between two events α and β at level 0. Since P is a critical path of maximum delay, all delays on this path are maximised. No other path $P' = \alpha \rightsquigarrow \beta$ can have a longer delay than path P, because path P is critical. No matter what delay assignments we make to path P', P remains a critical path. Without loss of generality we maximise all delays of all edges.

Consider the calculation of AC_{lb}. We need to choose a critical path with minimum delay P between two events α and β at level 0. We choose a critical path $P = \alpha \rightsquigarrow \beta$ on which all edge delays are minimised. No other path $P' = \alpha \rightsquigarrow \beta$ can have a longer delay than path P, since P is critical. Without loss of generality we minimise all delays of all edges.

□

Therefore we simplify a bounded delay model to a fixed delay model. This will greatly simplify proofs.

4.2.2 Average Response Time

Average response time differs from cycle time in that we exclude the delay of the environment. To maximise our upper bound on average response time, therefore, we minimise environment delay. We will prove this property formally in Lemma 4.2. To look at it another way, the upper bound on the average response time is produced when the pipeline is at its slowest, hence all edges have maximum delay, but the environment is putting requests into the pipeline as fast as possible. That is, environment edges have delay δ_e. When the pipeline is making requests as fast as possible, it gives the pipeline no "slack" to finish dealing with previous requests.

To calculate average response time we find a critical path between (r_0, i) and $(a_0, i+k)$ for some large k. To maximise the upper bound on average response time we maximise all delays between (r_0, i) and $(a_0, i+k)$ except those of the environment. We minimise all environment delays to δ_e.

Formally, let all delays between (r_0, i) and $(r_0, i+k)$ be maximised except the delays of the environment which are minimised to δ_e. Let P_k be the critical path between (r_0, i) and $(a_0, i+k)$. There are k environment edges $a_0 \rightarrow r_0$ between (r_0, i) and $(a_0, i+k)$ and therefore $k\delta_e$ environment delays between (r_0, i) and $(a_0, i+k)$. We subtract this delay from our formula. Thus

$$\begin{aligned} AR_{ub} &= \lim_{k\to\infty} \frac{\text{delay}(P_k) - k\delta_e}{k+1} \\ &= \left(\lim_{k\to\infty} \frac{\text{delay}(P_k) + \delta_e}{k+1}\right) - \delta_e \\ &= \left(\lim_{k\to\infty} \frac{\text{delay}(P_k)}{k}\right) - \delta_e \quad \{\text{Since } k \to \infty\} \qquad (4.3) \end{aligned}$$

We call AR_{ub} the *critical upper bound* for average response time. We have that $AR \leq AR_{ub}$.

Lastly, we define the critical lower bound for average response time AR_{lb}. In this case we maximise the environment delay to Δ_e. Let all delays between (r_0, i) and $(a_0, i+k)$ be minimised except the delays of the environment which are maximised. Let P_k be the critical path between (r_0, i) and $(a_0, i+k)$. There are k environment edges $a_0 \rightarrow r_0$ between (r_0, i) and $(a_0, i+k)$ and therefore $k\Delta_e$ environment delay between (r_0, i) and $(a_0, i+k)$. We subtract this delay from our formula as above and obtain

$$AR_{lb} = \left(\lim_{k\to\infty} \frac{\text{delay}(P_k)}{k}\right) - \Delta_e \tag{4.4}$$

We call AR_{lb} the *critical lower bound* for average response time. We have that $AR \geq AR_{lb}$. The critical bounds for response time are tight and therefore unique. As with critical bounds for average cycle time, the critical bounds for average response time are defined only for deterministic behaviours.

Again we have a lemma that allows us to use a fixed delay model.

Lemma 4.2 *The critical upper bound for average response time AR_{ub} for any deterministic behaviour can be obtained when we minimise the delay of the environment and maximise all other delays. The critical lower bound for average response time AR_{lb} for any deterministic behaviour can be obtained when we maximise the delay of the environment and minimise all other delays.*

Proof. We prove the lemma only for the upper bound, as the proof for the lower bound follows by a symmetrical argument. To find the upper bound for AR we choose two events at level 0 $\alpha = (e, i)$ and $\beta = (f, j)$ such that $j \gg i$. If $j - i$ is sufficiently large, we can choose e to be a request and f to be an acknowledgment without loss of generality. This reduces the number of cases we have to consider in the following calculations.

Now we find the longest delay path possible between α and β. Since we are trying to find the response time, not the cycle time, we subtract the time taken by the environment. Lastly we divide by the number of request/acknowledgement cycles between α and β at level 0 to obtain the bound on the average response time. That is,

$$AR \leq \max\{(\text{delay}(\alpha \rightsquigarrow \beta) - \text{env. delay})/(\text{num. cycles})\}$$

This equation corresponds to the unsimplified version of Equation 4.3.

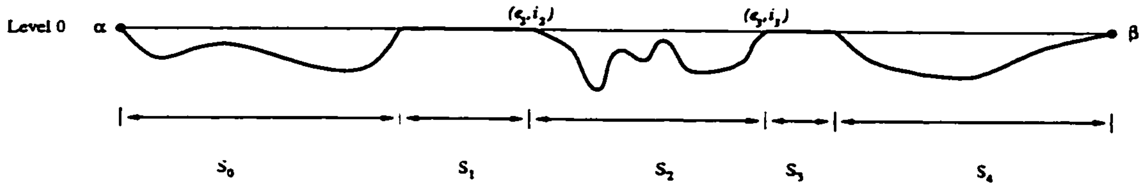


Figure 4.1: Path $\alpha \rightsquigarrow \beta$ broken into segments, $S_0, \ldots S_x$. Two example events, (e_2, i_2) and (e_3, i_3), are illustrated.

To maximise the formula we have to maximise delay($\alpha \rightsquigarrow \beta$) while minimising the environment delay. Let us break $\alpha \rightsquigarrow \beta$ into segments $S_0, S_1, \cdots S_x$ such that each segment S_i is either a path at level 0 or a path in $(0, L]$ as defined in Section 2.6.4. That is, either S_i consists of events at level 0 alone, or starts and ends with an event at level 0 and contains no other events at level 0. See Figure 4.1. Each segment S_k is a path $(e_k, i_k) \rightsquigarrow (e_{k+1}, i_{k+1})$ with $i_{k+1} \geq i_k$ and where e_k and e_{k+1} are level 0 events.

$$\begin{aligned} AR &\leq \max\{(\text{delay}(\alpha \rightsquigarrow \beta) - \text{env. delay})/(\text{num. cycles})\} \\ &= \max\{(\sum_k(\text{delay}(S_k) - \text{env. delay}(S_k)))/(\text{num. cycles})\} \qquad (4.5) \end{aligned}$$

where the environment delay(S_k) refers to the delay of the environment edges between (e_k, i_k) and (e_{k+1}, i_{k+1}) at level 0.

There are two cases.

1. If S_k is a path in $(0, L]$, then (delay(S_k) − env. delay(S_k)) is maximised when non-environment edges are maximised and environment edges are minimised.
2. If S_k is a level 0 path, (delay(S_k) − env. delay(S_k)) is also maximised when non-environment delays are maximised. It is irrelevant what the environment delays are, because they are cancelled out in this expression, and therefore also in expression 4.5 above.

Without loss of generality, we can therefore maximise non-environment delays and minimise environment delays, and the lemma holds.

□

The above lemma is a blueprint for all of the analyses carried out for average-case behaviour in the thesis. Equations 4.1, 4.2, 4.3, and 4.4 require that we choose a critical path P of infinite length. Instead we sometimes choose a finite critical path P' that we can repeat over and over again.

Note, that in the definition of average response time, the implicit assumption is made that the environment has a handshaking protocol and that we are averaging over the number of handshakes between the environment and cell 0. For example, between (r_0, i) and (a_0, j) in the above proof we know that there are $j - i + 1$ request-to-acknowledgment edges, which are responses from the first cell of the architecture, and $j - i$ acknowledgment-to-request edges, which is the time taken by the environment to issue a request. This handshaking assumption is unnecessary for this lemma. The number of cycles can be defined to be anything we want and is defined more generally in Chapter 6.

4.3 Average-Case Response Time

We now proceed to calculate bounds on average response time for pipelines. We show that we can have great freedom in designing asynchronous pipelines, in both cell ordering and pipeline length, without compromising the worst-case throughput.

In all proofs involving average-case bounds, any finite critical path P we use to derive the bound must be infinitely repeatable. For example, the critical path in a behavioural graph may be the path consisting of level 0 events only. We might calculate average response time using only a short segment of this path, viz., the single edge $r_0 \rightarrow a_0$. This is only one edge, but we can repeat the path ad infinitum. We need this property, because our average-case measures must be bounds on the average over the entire possible execution of the network.

We have the following theorem.

Theorem 4.3 *The average response time AR for any pipeline (as defined in Section 2.3 operating under a delay model of Section 2.6), and with multiplication factor $n = 1$, is bounded as follows*

$$mc_{\Delta_e} - \Delta_e \leq \ AR \ \leq MC_{\delta_e} - \delta_e$$

where $mc_{\Delta_e} - \Delta_e = AR_{lb}$ and $MC_{\delta_e} - \delta_e = AC_{ub}$ are critical bounds as defined in Section 4.2.

Proof. First we prove the upper bound $AC_{ub} = MC_{\delta_e} - \delta_e$.

From request (r_0, i_0) there is a critical path P to acknowledgment (a_0, j_0) with $i_0 \leq j_0$ and on this critical path there is no other event at level 0. That is, we have $P \in (0, L]$. Let this path be of length $2k+1$ with $k \geq 0$. There may be many paths between (r_0, i_0) and (a_0, j_0) of length $2k+1$. Note that the only possible path that is wholly at level 0 is the single-edge path when $k = 0$ and $i_0 = j_0$. We label each edge on each path with maximum delays and then we choose a critical path P. This path P is the longest path in terms of delay between (r_0, i_0) and (a_0, j_0) excluding paths through the environment. We call the delay on this critical path of length $2k+1$, D_k.

Between (r_0, i_0) and (a_0, j_0) on level 0 there are k environment edges (i.e, $a_0 \rightarrow r_0$) and $k+1$ "response" edges (i.e, $r_0 \rightarrow a_0$). Environment delays are at least δ_e. In k handshake cycles, the total time for the environment delays is at least $k\delta_e$. To obtain the upper bound on the average response time we subtract the environment delay over k handshake cycles and divide by the number of responses.

$$\begin{aligned} AR_{ub} &= \max\{(D_k - k\delta_e)/(k+1) \mid k \geq 0\} \\ &= \max\{(D_k + \delta_e)/(k+1) \mid k \geq 0\} - \delta_e \end{aligned}$$

We look in more detail at the first term $\max\{(D_k + \delta_e)/(k+1) \mid k \geq 0\}$.

Suppose there is either an edge $r_i \rightarrow r_{i+1}$ or an edge $a_{i+1} \rightarrow r_{i+1}$ in our path P. Event r_{i+1} is the output request signal of cell i. The path must pass back through cell i at some point which means there must be an acknowledgement a_{i+1} forthcoming from cell $i+1$. So, for each edge on the path that ends in r_{i+1} there must be a corresponding edge that ends in a_{i+1}. For each pair of such edges we have a maximum delay of $\Delta_{i,f} + \Delta_{i+1,r} \leq MC_{\delta_e}$. There are k edges ending in a request in the path and so we have k pairs with total delay less than or equal to

$k \cdot MC_{\delta_e}$. We also have one "loose" reverse edge at the end of the path. So the total delay on the path is bounded as $D_k \leq k \cdot MC_{\delta_e} + \Delta_{0,r}$.

$$\begin{aligned}
&\max\{(D_k + \delta_e)/(k+1) \mid k \geq 0\} \\
\leq\ &\max\{(k \cdot MC_{\delta_e} + \Delta_{0,r} + \delta_e)/(k+1) \mid k \geq 0\} \\
\leq\ &\max\{(k \cdot MC_{\delta_e} + MC_{\delta_e})/(k+1) \mid k \geq 0\} \\
=\ &MC_{\delta_e}
\end{aligned}$$

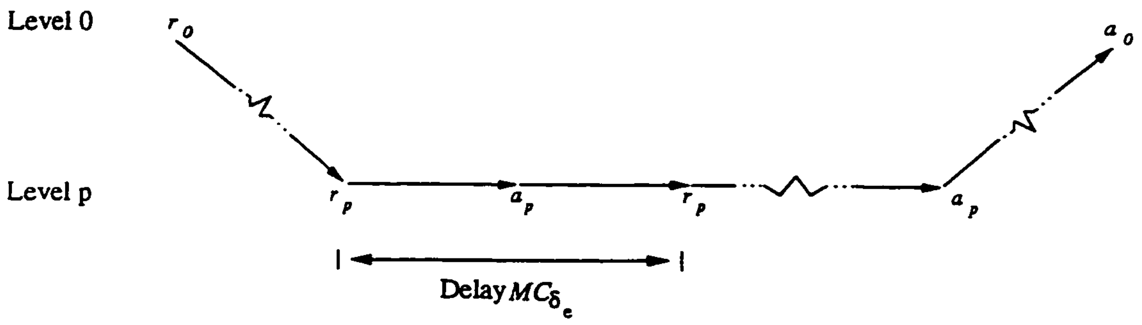


Figure 4.2: A possible critical path P where $MC_{\delta_e} = \Delta_{p,r} + \Delta_{p-1,f}$ with $p \geq 1$.

Suppose that

$$MC_{\delta_e} = \delta_e + \Delta_{0,r}$$

Then for $k = 0$, where P is a single edge path through cell 0, we have $D_k = \Delta_{0,r}$. Then,

$$\begin{aligned}
\max\{(D_k + \delta_e)/(k+1) \mid k \geq 0\} &= (\Delta_{0,r} + \delta_e)/(1) \\
&= MC_{\delta_e}
\end{aligned}$$

If on the other hand,

$$MC_{\delta_e} \;>\; \delta_e + \Delta_{0,r}$$

we can maximise the upper bound for AR at $k \to \infty$. We note that there exist paths P that almost exclusively consist of handshake cycles with delay MC_{δ_e}, except for a begin part and end part, and, because $MC_{\delta_e} > \delta_e + \Delta_{0,r}$, these handshake cycles are not at level 0. Suppose that $MC_{\delta_e} = \Delta_{p,r} + \Delta_{p-1,f}$ where $p \geq 1$. Then the critical path may look like Figure 4.2.

Let us break the delay on our critical path into two. Let $D_k = E_k + F_k$. The horizontal part of our critical path at level p has total delay $E_k = k_0 \cdot MC_{\delta_e} + \Delta_{p,r}$ for some k_0 such that $\lim_{k\to\infty}(k_0/k) = 1$. That is, k_0 approaches infinity as k does. Let the begin and end parts of our critical path have total delay F_k. We assume that all edge delays are finite, and we assume that the pipeline length L is also finite. Therefore the delay F_k is also finite. As $k \to \infty$, we have

$$\begin{aligned}
\lim_{k\to\infty} \frac{(D_k + \delta_e)}{k+1} &= \lim_{k\to\infty} \frac{(E_k + F_k + \delta_e)}{k+1} \\
&= \lim_{k\to\infty} \frac{(k_0 \cdot MC_{\delta_e} + \Delta_{p,r} + F_k + \delta_e)}{k+1} \\
&= \lim_{k\to\infty} \frac{k_0 \cdot MC_{\delta_e}}{k} \quad \{\text{Eliminating finite terms}\} \\
&= MC_{\delta_e} \quad \{\text{Since } \lim_{k\to\infty}(k_0/k) = 1\}
\end{aligned}$$

So in either case we find that our upper bound MC_{δ_e} is tight and that $\max\{(D_k + \delta_e)/(k+1) \mid k \geq 0\} = MC_{\delta_e}$. Consequently

$$\begin{aligned}
AR_{ub} &= \max\{(D_k + \delta_e)/(k+1) \mid k \geq 0\} - \delta_e \\
&= MC_{\delta_e} - \delta_e
\end{aligned}$$

We can use a similar explanation to prove the lower bound. Let the same paths exist as before except the delay on each non-environment edge is minimised. Let the critical path of length $2k+1$ have delay d_k. Since we are trying to find a lower bound, we have to maximise the environment delay when we subtract it. We obtain

$$\begin{aligned} AR_{lb} &= \max\{(d_k - k\Delta_e)/(k+1) \mid k \geq 0\} \\ &= \max\{(d_k + \Delta_e)/(k+1) \mid k \geq 0\} - \Delta_e \end{aligned}$$

Using the same steps as above we can show that

$$\max\{(d_k + \Delta_e)/(k+1) \mid k \geq 0\} = mc_{\Delta_e}$$

and the result follows.

□

The above theorem states that the average response time of a pipeline with multiplication factor $n = 1$ is bounded by two bottlenecks. One bottleneck is described by the minimum cycle time mc_{Δ_e} and the other by the maximum cycle time MC_{δ_e}. These bottlenecks need not occur at the same place in the pipeline. If one has the freedom to rearrange stages in the pipeline one might be tempted to split up stages that have a large cycle time between them so as to minimise MC_{δ_e}. Two remarks are in order, however. Firstly, if the reverse latencies of stages are identical, then there is no benefit in rearranging stages. Secondly, our formulae are only dealing with bounds. As a consequence, these formulae say what happens to the bounds of the average-case response time under any rearrangement of stages. They do not say what happens with the actual average-case response time.

The optimisation process for a synchronous pipeline has some similarities to that of an asynchronous pipeline. In a clocked circuit, we are also interested in minimising the worst-case bottleneck. Yet, in any synchronous design, the clock frequency must be slow enough such that the slowest component, i.e., the bottleneck, has time to complete its operation. This means that any redesign of the circuit requires global retiming if we are to take advantage of the redesign. If we do not do a global retiming we will not be able to increase the clock frequency. Any retiming in an asynchronous circuit need only be local. Lastly, a synchronous circuit will always be restricted by its worst-case bottleneck. An asynchronous circuit may possibly go faster as it does not have to wait for a clock pulse before proceeding.

Franklin and Pan [20] give a similar result to our lower bound mc_{Δ_e}, but they assumed a non-blocking model in which no data in the pipeline can ever be blocked. We do not need this assumption. They also give a result somewhat similar to our upper bound MC_{δ_e}, but they assume that each pipeline stage operates in lockstep, and they give no proof that this could happen in practice or that the bound they give is tight.

4.4 An Example Micropipeline

We refer back to the example given in Section 2.6.2 where we have a pipeline with length $L = 3$. We instantiate the delay elements as before and as in Figure 4.3. The minimum cycle time mc_{Δ_e} is calculated as follows.

$$\begin{aligned} mc_{\Delta_e} &= \max\{\Delta_e + \delta_{0,r}, \delta_{0,f} + \delta_{1,r}, \delta_{1,f} + \delta_{2,r}, \ldots, \delta_{L-1,f} + \delta_{L,r}\} \\ &= \max\{4+1, 2+1, 1+1, 1+2\} \\ &= \max\{5, 3, 2, 3\} \\ &= 5 \end{aligned}$$

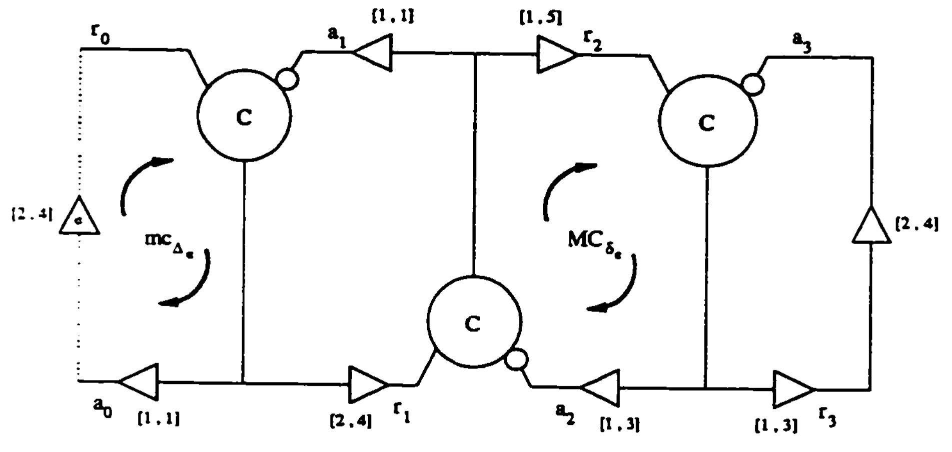


Figure 4.3: The environment delay is labelled with an e. The minimum and maximum cycle times are illustrated.

We calculate the maximum cycle time, MC_{δ_e}.

$$\begin{aligned}
MC_{\delta_e} &= \max\{\delta_e + \Delta_{0,r}, \Delta_{0,f} + \Delta_{1,r}, \Delta_{1,f} + \Delta_{2,r}, \ldots, \Delta_{L-1,f} + \Delta_{L,r}\} \\
&= \max\{2+1, 4+1, 5+3, 3+4\} \\
&= \max\{3, 5, 8, 7\} \\
&= 8
\end{aligned}$$

Since $mc_{\Delta_e} - \Delta_e \leq AR \leq MC_{\delta_e} - \delta_e$ we have $1 \leq AR \leq 6$.

Compare these results with the bounds for RT calculated in the previous chapter, where we found $RT \leq 7$. So, the bounds on the average response time can be different from the worst case response time. Any single response time of 7 time units must be compensated over time by one or more response times of less than 6 time units.

4.5 Average Cycle Time for $n = 1$

We can also calculate bounds on the average cycle time. Since the paths involved are potentially infinite, it does not matter whether we calculate from request to request or from acknowledgment to acknowledgment. The proof proceeds in the same way as for average response time AR.

Recall the definitions of mc_{δ_e} and MC_{Δ_e}.

$$mc_{\delta_e} = \max\{\delta_e + \delta_{0,r}, \delta_{0,f} + \delta_{1,r}, \delta_{1,f} + \delta_{2,r}, \ldots, \delta_{L-1,f} + \delta_{L,r}\}$$

$$MC_{\Delta_e} = \max\{\Delta_e + \Delta_{0,r}, \Delta_{0,f} + \Delta_{1,r}, \Delta_{1,f} + \Delta_{2,r}, \ldots, \Delta_{L-1,f} + \Delta_{L,r}\}$$

Theorem 4.4 *The average cycle time AC for any pipeline (as defined in Section 2.3 operating under a delay model of Section 2.6), and with multiplication factor $n = 1$, is bounded as follows*

$$mc_{\delta_e} \le AC \le MC_{\Delta_e}$$

This bound is tight.

Proof. First we prove the upper bound.

We are going to look at a path between (r_0, i_0) and $(r_0, j_0 + 1)$ where $i_0 \le j_0$. For acknowledgment (a_0, j_0) there is a critical path from some request (r_0, i_0) to (a_0, j_0) with $i_0 \le j_0$. We do not restrict the path to be in $(0, L]$ as we did with response time, since we do not need any special cases.

Let this path be of length $2k+1$ with $k \ge 0$. There may be many paths between (r_0, i_0) and (a_0, j_0) of length $2k+1$. We label each edge on each path with maximum delays and then we choose a critical path P. This path P is the longest path in terms of delay between (r_0, i_0) and (a_0, j_0). The delay on this critical path of length $2k+1$ is given by D_k.

Between (r_0, i_0) and (r_0, j_0) on level 0 there are k handshake cycles consisting of environment edges, i.e, $a_0 \rightarrow r_0$, and k "response" edges, i.e, $r_0 \rightarrow a_0$. We do not subtract the environment delay as we did with response time. To obtain the bound on the average cycle time we take the delay on path P $(= D_k)$ plus the delay on the edge $(a_0, j_0) \rightarrow (r_0, j_0+1)$ $(= \Delta_e)$ and divide by the number of cycles $(= k+1)$.

$$AC_{ub} = \max\{(D_k + \Delta_e)/(k+1) \mid k \geq 0\}$$

In the same way as in the proof of Theorem 4.3 we can show that the delay on path P is bounded as $D_k \leq k \cdot MC_{\Delta_e} + \Delta_{0,r}$.

$$\begin{aligned} AC_{ub} &= \max\{(D_k + \Delta_e)/(k+1) \mid k \geq 0\} \\ &\leq \max\{(k \cdot MC_{\Delta_e} + \Delta_{0,r} + \Delta_e)/(k+1) \mid k \geq 0\} \\ &\leq \max\{(k \cdot MC_{\Delta_e} + MC_{\Delta_e})/(k+1) \mid k \geq 0\} \\ &= MC_{\Delta_e} \end{aligned}$$

We can maximise the right-hand side at $k \rightarrow \infty$ for all cases. We note that there exist paths P that almost exclusively consist of handshake cycles with delay MC_{Δ_e}, except for a begin part and end part. Suppose that $MC_{\Delta_e} = \Delta_{p,r} + \Delta_{p-1,f}$ where $p \geq 1$. Then the critical path may again look like Figure 4.4. Alternatively, the critical path may be a level 0 path when $MC_{\Delta_e} = \Delta_{0,r} + \Delta_e$.

Let us break the delay on path our critical path into two. Let $D_k = E_k + F_k$. The horizontal part of our critical path at level p has total delay $E_k = k_0 \cdot MC_{\Delta_e} + \Delta_{p,r}$ for k_0 such that $\lim_{k \rightarrow \infty}(k_0/k) = 1$. That is, k_0 approaches infinity as k does. Let

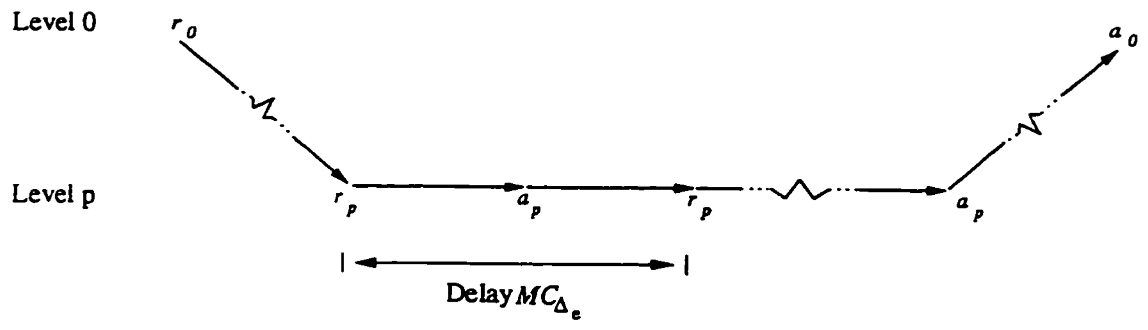


Figure 4.4: A possible critical path P where $MC_{\Delta_e} = \Delta_{p,r} + \Delta_{p-1,f}$ with $p \geq 1$.

the begin and end parts of our critical path have total delay F_k. We assume that all edge delays are finite, and we assume that the pipeline length L is also finite. Therefore the delay F_k is also finite. As $k \to \infty$, we have

$$\begin{aligned}
\lim_{k\to\infty} \frac{(D_k + \Delta_e)}{k+1} &= \lim_{k\to\infty} \frac{(E_k + F_k + \Delta_e)}{k+1} \\
&= \lim_{k\to\infty} \frac{(k_0 \cdot MC_{\Delta_e} + \Delta_{p,r} + F_k + \Delta_e)}{k+1} \\
&= \lim_{k\to\infty} \frac{k_0 \cdot MC_{\Delta_e}}{k} \quad \{\text{Eliminating finite terms}\} \\
&= MC_{\Delta_e} \quad \{\text{Since } \lim_{k\to\infty}(k_0/k) = 1\}
\end{aligned}$$

From the above equations we find that $\max\{D_k + \Delta_e)/(k+1) \mid k \geq 0\} = MC_{\Delta_e}$. Consequently

$$\begin{aligned}
AC_{ub} &= \max\{(D_k + \Delta_e)/(k+1) \mid k \geq 0\} \\
&= MC_{\Delta_e}
\end{aligned}$$

We can use a similar explanation to prove the lower bound. Let the same paths exist as before except the delay on each edge is minimised. Let the critical path of length $2k+1$ have delay d_k. We add the minimum environment delay δ_e to this delay and then divide by the number of cycles. We obtain

$$AC_{lb} \quad = \quad \max\{(d_k + \delta_e)/(k+1) \mid k \geq 0\}$$

Using the same steps as above we can show that $\max\{(d_k + \delta_e)/(k+1) \mid k \geq 0\} = mc_{\delta_e}$ and the result follows.

□

4.6 The Relationship Between AR and AC

There is clearly a relationship between the values of AC and AR. For example, when the multiplication factor is $n = 1$, an upper bound on AR can be given by $AC_{ub} - \delta_e$ in most cases, though there are special cases if the critical path passes only through the environment. If we could construct a general relationship between the bounds for AR and AC, then we would only have to write one proof to calculate both values. We will be using this in future proofs for deriving the average cycle time later in the thesis. For example, we calculate the average cycle time for a pipeline with multiplication factor $n > 1$ in Section 4.8 by using this relationship. We can thereby avoid repeating a longwinded proof. The chief problem with finding such a relationship is that while environment delays are maximised to compute an upper bound for average cycle time, the environment delays are minimised when computing an upper bound on average response time.

In this section we give a general relationship between the *critical bounds* for AR and AC under certain restrictions. First we require that the environment has a handshaking behaviour with the first cell, such that every other edge at level 0 is an environment edge. Next, critical bounds as described in Section 4.2 prescribe a particular method of calculating the bounds. We calculate critical bounds by finding a critical path between two level 0 events that are a large distance apart and then averaging over the number of handshake cycles at level 0. This critical path is made up of segments $\{S_g \mid g \in G\}$ at level 0 for some set G, and segments $\{S_h \mid h \in H\}$ each in $(0, L]$ for some set H. We assume that we have to consider only two cases.

Case (i) The critical path is a level 0 path. Therefore $\{S_h \mid h \in H\} = \emptyset$.

Case (ii) The delay on the critical path caused by level 0 segments is negligible. That is, if k is the difference in occurrence indices between the first event on the critical path and the last event of the critical path

$$\lim_{k \to \infty} \frac{\sum_{g \in G} \text{Delay}(S_g)}{k} = 0$$

The second case is more restrictive than we actually need, but it makes proofs simpler. All the behaviours in this thesis meet these restrictions, as they all have handshaking environments, and we can always choose a suitable critical path that is either a level 0 path or a critical path that consists largely of an infinitely long segment in $(0, L]$. In general we might need to consider mixed paths rather than the two cases presented above.

We define $\overline{AC}_{ub}$ as the upper bound on the cycle time calculated if we only consider the horizontal paths at level 0. We define $\widehat{AC}_{ub}$ as the upper bound on the cycle time calculated if we only consider paths that have delay that is negligible at level 0. These concepts are formalised in the following definitions. The definitions are based on the formulae for critical bounds defined in Section 4.2. As with critical bounds, these values are only defined for deterministic behaviour.

First let us define what we mean by *path delays.*

Definition 4.5 *Let the* path delay *be the sum of the edge delays on a path.*

For example, if all edge delays were maximised, a path $r_0 \to r_1 \to r_2$ would have path delay $\Delta_{0,f} + \Delta_{1,f}$. It would have this path delay whether or not it was a critical path. If, however, a path $\alpha \to \beta$ is critical, then path delay$(\alpha \rightsquigarrow \beta) = T(\beta) - T(\alpha)$.

Definition 4.6 *Let B represent a process graph where the environment handshakes with the component. Let $\overline{P}_k$ be a path at level 0 between (r_0, i) and $(r_0, i+k)$ with all edge delays maximised. Then,*

$$\overline{AC}_{ub} = \lim_{k \to \infty} \frac{path\ delay(\overline{P}_k)}{k} \tag{4.6}$$

Again, let all edge delays be maximised. Let $\widehat{P}_k$ be the path with the greatest path delay between (r_0, i) and $(r_0, i+k)$ when the path has negligible delays at level 0. That is, if $\widehat{P}_k$ is made up of segments $\{S_g \mid g \in G\}$ at level 0 for some set G, and segments $\{S_h \mid h \in H\}$ each in $(0, L]$ for some set H, then

$$\lim_{k \to \infty} \frac{\sum_{g \in G} Delay(S_g)}{k} = 0$$

Given $\widehat{P}_k$ we have

$$\widehat{AC}_{ub} = \lim_{k \to \infty} \frac{path\ delay(\widehat{P}_k)}{k} \tag{4.7}$$

$\overline{AC}_{lb}$ and $\widehat{AC}_{lb}$ are similarly defined.

□

Notice that if we assume that the reverse delay of the cell interfacing with the environment is bounded by $\delta_{0,r}$ and $\Delta_{0,r}$, then we can compute $\overline{AC}_{lb} = \delta_e + \delta_{0,r}$ and $\overline{AC}_{ub} = \Delta_e + \Delta_{0,r}$ for all behaviours with a handshaking environment. We can perform this calculation easily because there is only one possible path of length k at level 0. The values of $\widehat{AC}_{lb}$ and $\widehat{AC}_{ub}$ are dependent on the behaviour we are examining. For example, these values will be different for pipelines and for trees.

We define $\overline{AR}_{ub}$, $\widehat{AR}_{ub}$, $\overline{AR}_{lb}$, and $\widehat{AR}_{lb}$ in a similar fashion to the definitions above.

Definition 4.7 *Let B represent a process graph where the environment handshakes with the component. Let $\overline{P'}_k$ be a path at level 0 between (r_0, i) and $(a_0, i+k)$ when the delays of all edges are maximised except the environment delays which are minimised. Then,*

$$\overline{AR}_{ub} = \left(\lim_{k\to\infty} \frac{\textit{path delay}(\overline{P'}_k)}{k}\right) - \delta_e$$

Again, let all edge delays be maximised except the environment delays which are minimised. Let $\widehat{P'}_k$ be the path between (r_0, i) and $(a_0, i+k)$ with greatest path delay when $\widehat{P'}_k$ has negligible delays at level 0. That is, if $\widehat{P'}_k$ is made up of segments $\{S_g \mid g \in G\}$ at level 0 for some set G, and segments $\{S_h \mid h \in H\}$ each in $(0, L]$ for some set H, then

$$\lim_{k\to\infty} \frac{\sum_{g\in G} \textit{Delay}(S_g)}{k} = 0$$

Given $\widehat{P'}_k$ we have

$$\widehat{AR}_{ub} = \left(\lim_{k\to\infty} \frac{\textit{path delay}(\widehat{P'}_k)}{k}\right) - \delta_e$$

$\overline{AR}_{lb}$ and $\widehat{AR}_{lb}$ are similarly defined as

$$\overline{AR}_{lb} = \left(\lim_{k\to\infty} \frac{path\ delay(\overline{P'})}{k}\right) - \Delta_e$$

and

$$\widehat{AR}_{lb} = \left(\lim_{k\to\infty} \frac{path\ delay(\widehat{P'}_k)}{k}\right) - \Delta_e$$

where all delays are minimised except those of the environment which are maximised.

□

If we assume that the reverse delay of the cell interfacing with the environment is bounded by $\delta_{0,r}$ and $\Delta_{0,r}$, then we can compute $\overline{AR}_{lb} = \delta_{0,r}$ and $\overline{AR}_{ub} = \Delta_{0,r}$ for all behaviours that have a handshaking environment. Note that all of these definitions define a unique value for a particular behaviour.

We can calculate the critical bounds for the tree and pipeline architectures in this thesis by only examining the two cases described above. We formalise this property in a lemma.

Lemma 4.8 *Let B represent a process graph where the environment handshakes with the component. Let (r_0, i) and $(r_0, i+k)$ be events at level 0. Let AC_{ub} be the critical bound (as defined in Section 4.2) derived from path $P_k = (r_0, i) \rightsquigarrow (r_0, i+k)$ in B. Let $\overline{AC}_{ub}$ and $\widehat{AC}_{ub}$ be defined between (r_0, i) and $(r_0, i+k)$ as in Definition 4.6. If P_k is either (i) a path at level 0, or (ii) has negligible delays at level 0, then*

- $AC_{ub} = \max\{\overline{AC}_{ub}, \widehat{AC}_{ub}\}$

Using similar definitions we have

- $AC_{lb} = \max\{\overline{AC}_{lb}, \widehat{AC}_{lb}\}$

Let (r_0, i) and $(a_0, i+k)$ be events at level 0. Let AR_{ub} be the critical bound (as defined in Section 4.2) derived from path $P'_k = (r_0, i) \leadsto (a_0, i+k)$ in B. Let $\overline{AR}_{ub}$ and $\widehat{AR}_{ub}$ be defined between (r_0, i) and $(r_0, i+k)$ as in Definition 4.7. If P'_k is either (i) a path at level 0, or (ii) has negligible delays at level 0, then

- $AR_{ub} = \max\{\overline{AR}_{ub}, \widehat{AR}_{ub}\}$
- $AR_{lb} = \max\{\overline{AR}_{lb}, \widehat{AR}_{lb}\}$

Proof. We show only $AC_{ub} = \max\{\overline{AC}_{ub}, \widehat{AC}_{ub}\}$, since the other results follow similarly. If the critical path is a level 0 path, then from Definition 4.6 we have $\overline{AC}_{ub} \geq \widehat{AC}_{ub}$. We also have $AC_{ub} = \overline{AC}_{ub}$. Therefore $AC_{ub} = \max\{\overline{AC}_{ub}, \widehat{AC}_{ub}\}$ for this case. If the critical path has negligible delays at level 0, then $AC_{ub} = \widehat{AC}_{ub}$ and $\overline{AC}_{ub} \leq \widehat{AC}_{ub}$ and the result follows.

□

Given Lemma 4.8 we demonstrate the relationship between the critical bounds for AC and AR.

Theorem 4.9 *Let B be a behavioural graph. Let $\overline{AC}_{ub}$, $\widehat{AC}_{ub}$, $\overline{AR}_{ub}$ and $\widehat{AR}_{ub}$ be defined as in Definition 4.6. Then*

$$\begin{aligned} \overline{AR}_{ub} &= \overline{AC}_{ub} - \Delta_e \\ \widehat{AR}_{ub} &= \widehat{AC}_{ub} - \delta_e \end{aligned}$$

Similarly we have

$$
\begin{aligned}
\overline{AR}_{lb} &= \overline{AC}_{lb} - \delta_e \\
\widehat{AR}_{lb} &= \widehat{AC}_{lb} - \Delta_e
\end{aligned}
$$

Proof. Let us show that $\overline{AR}_{ub} = \overline{AC}_{ub} - \Delta_e$. Let $\overline{P}_k$ be a level 0 path between (r_0, i) and $(r_0, i+k)$ with all edge delays on the path maximised.

The environment delay on path $\overline{P}_k$ in each handshake cycle has been maximised at Δ_e, since this is how we calculate $\overline{AC}_{ub}$. Since the path $\overline{P}_k$ is a level 0 path with k handshakes, there must be k environment edges. Hence the environment delay on path $\overline{P}_k$ is $\Delta_e \cdot k$. Let $\overline{Q}_k$ be the level 0 path $(r_0, i) \rightsquigarrow (r_0, i+k)$ with all delays maximised except the environment delays which are minimised. We have

$$\text{path delay}(\overline{Q}_k) = \text{path delay}(\overline{P}_k) - k \cdot \Delta_e + k \cdot \delta_e$$

We calculate $\overline{AR}_{ub}$ between a request and an acknowledgment, not between two requests. Let $\overline{P'}_k$ be the level 0 path $(r_0, i) \rightsquigarrow (a_0, i+k-1)$ with all delays maximised except for environment delays which are minimised. That is, $\overline{P'}_k$ is the same as path $\overline{Q}_k$ minus the last edge $(a_0, i+k-1) \rightarrow (r_0, i+k)$. We have path delay$(\overline{P'}_k)$ = path delay$(\overline{Q}_k) - \delta_e$.

From Definition 4.7 we calculate $\overline{AR}_{ub}$ over path $\overline{P'}_k$ as

$$
\begin{aligned}
\overline{AR}_{ub} &= \left(\lim_{k\to\infty} \frac{\text{path delay}(\overline{P'}_k)}{k}\right) - \delta_e \\
&= \left(\lim_{k\to\infty} \frac{\text{path delay}(\overline{Q}_k) - \delta_e}{k}\right) - \delta_e \\
&= \left(\lim_{k\to\infty} \frac{\text{path delay}(\overline{P}_k) - k \cdot \Delta_e + k \cdot \delta_e - \delta_e}{k}\right) - \delta_e
\end{aligned}
$$

$$\begin{aligned}
&= \left(\lim_{k\to\infty} \frac{\text{path delay}(\overline{P}_k) - \delta_e}{k}\right) - \Delta_e + \delta_e - \delta_e \\
&= \left(\lim_{k\to\infty} \frac{\text{path delay}(\overline{P}_k) - \delta_e}{k}\right) - \Delta_e \\
&\quad \{\text{Eliminating finite terms}\} \\
&= \left(\lim_{k\to\infty} \frac{\text{path delay}(\overline{P}_k)}{k}\right) - \Delta_e \\
&\quad \{\text{Equation 4.6}\} \\
&= \overline{AC}_{ub} - \Delta_e
\end{aligned}$$

Let us now show that $\widehat{AR}_{ub} = \widehat{AC}_{ub} - \delta_e$. Let all edge delays be maximised. Let $\widehat{P}_k$ be the path between (r_0, i) and $(r_0, i+k)$ with greatest path delay when the level 0 parts of the path have negligible delay. That is, $\widehat{P}_k$ is made up of segments $\{S_g \mid g \in G\}$ at level 0 for some set G, segments $\{S_h \mid h \in H\}$ each in $(0, L]$ for some set H, and that

$$\lim_{k\to\infty} \frac{\sum_{g\in G} \text{Delay}(S_g)}{k} = 0$$

Consider the segments of $\widehat{P}_k$ in $\{S_g \mid g \in G\}$. These segments are the parts of path $\widehat{P}_k$ that are at level 0. The environment delay on path $\widehat{P}_k$ in each handshake cycle has been maximised at Δ_e, since this is how we calculate $\widehat{AC}_{ub}$. Therefore all environment edges in segments at level 0 have delay Δ_e. Let the total delay of environment edges in segments at level 0 be $k_0 \cdot \Delta_e$ for some $k_0 \geq 0$. The total environment delay over the level 0 segments is less than the total delay of the level 0 segments themselves. That is, $k_0 \cdot \Delta_e \leq \sum_{g\in G} \text{Delay}(S_g)$. Then

$$\lim_{k\to\infty} \frac{\sum_{g\in G} \text{Delay}(S_g)}{k} = 0 \;\Rightarrow\; \lim_{k\to\infty} \frac{k_0 \cdot \Delta_e}{k} = 0 \tag{4.8}$$

The environment edges traversed by the critical path contribute negligible delay to the overall path. Let $\widehat{Q}_k$ be the same path as $\widehat{P}_k$ between (r_0, i) and $(r_0, i+k)$ with all delays maximised except the environment delays which are minimised. We have

$$\text{path delay}(\widehat{Q}_k) = \text{path delay}(\widehat{P}_k) - k_0 \cdot \Delta_e + k_0 \cdot \delta_e$$

We calculate $\widehat{AR}_{ub}$ between a request and an acknowledgment, not between two requests. Let path $\widehat{P'}_k$ be the same as path $\widehat{Q}_k$ minus the last edge $(a_0, i+k-1) \to (r_0, i+k)$. We have path delay$(\widehat{P'}_k)$ = path delay$(\widehat{Q}_k) - \delta_e$.

From Definition 4.7 we calculate $\widehat{AR}_{ub}$ over path $\widehat{P'}_k$ as

$$\begin{aligned}
\widehat{AR}_{ub} &= \left(\lim_{k\to\infty} \frac{\text{path delay}(\widehat{P'}_k)}{k}\right) - \delta_e \\
&= \left(\lim_{k\to\infty} \frac{\text{path delay}(\widehat{Q}_k) - \delta_e}{k}\right) - \delta_e \\
&= \left(\lim_{k\to\infty} \frac{\text{path delay}(\widehat{P}_k) - k_0 \cdot \Delta_e + k_0 \cdot \delta_e - \delta_e}{k}\right) - \delta_e \\
&\quad \{ \text{ Equation 4.8, } \delta_e \le \Delta_e \text{, eliminating finite terms } \} \\
&= \left(\lim_{k\to\infty} \frac{\text{path delay}(\widehat{P}_k)}{k}\right) - \delta_e \\
&\quad \{\text{Equation 4.7}\} \\
&= \widehat{AC}_{ub} - \delta_e
\end{aligned}$$

We can construct a similar proof for the lower bounds $\overline{AR}_{lb}$ and $\widehat{AR}_{lb}$.

□

Corollary 4.10 *Let B be a behavioural graph and let AR_{ub} be derived from a critical path either (i) at level* 0, *or (ii) having negligible delays at level* 0. *Let $\overline{AC}_{ub}$ and $\widehat{AC}_{ub}$ be defined as in Definition 4.6. Then*

$$AR_{ub} = \max\{\overline{AC}_{ub} - \Delta_e, \widehat{AC}_{ub} - \delta_e\}$$

where AR_{ub} is the critical upper bound for average response time. Similarly we have

$$AR_{lb} = \max\{\overline{AC}_{lb} - \delta_e, \widehat{AC}_{lb} - \Delta_e\}$$

where AR_{lb} is the critical lower bound for average response time.

Proof. The result follows directly from Lemma 4.8 and Theorem 4.9.

□

Let us demonstrate Corollary 4.10 for pipelines with multiplication factor $n = 1$.

$$\begin{aligned} AC_{ub} &= MC_{\Delta_e} \\ &\qquad \{\text{ By defn. of } MC_{\Delta_e}. \text{ }\} \\ &= \max\{\Delta_e + \Delta_{0,r}, \Delta_{0,f} + \Delta_{1,r}, \Delta_{1,f} + \Delta_{2,r}, \ldots, \Delta_{L-1,f} + \Delta_{L,r}\} \\ &= \max(\{\Delta_e + \Delta_{0,r}\} \cup \{\Delta_{0,f} + \Delta_{1,r}, \Delta_{1,f} + \Delta_{2,r}, \ldots, \Delta_{L-1,f} + \Delta_{L,r}\}) \end{aligned}$$

We have

$$\overline{AC}_{ub} = \Delta_e + \Delta_{0,r}$$

and

$$\widehat{AC}_{ub} = \max\{\Delta_{0,f} + \Delta_{1,r}, \Delta_{1,f} + \Delta_{2,r}, \ldots, \Delta_{L-1,f} + \Delta_{L,r}\}$$

We derive the upper bound on average cycle time AC_{ub} from the critical path. For pipelines with multiplication factor $n = 1$, the critical path can either be a path at level 0, or a path with negligible delay at level 0. Then Corollary 4.10 gives us,

$$\begin{aligned} AR_{ub} &= \max\{\overline{AC}_{ub} - \Delta_e, \widehat{AC}_{ub} - \delta_e\} \\ &= \max\{\Delta_{0,r}, \max\{\Delta_{0,f} + \Delta_{1,r}, \Delta_{1,f} + \Delta_{2,r}, \ldots, \Delta_{L-1,f} + \Delta_{L,r}\} - \delta_e\} \\ &= \max\{\delta_e + \Delta_{0,r}, \Delta_{0,f} + \Delta_{1,r}, \Delta_{1,f} + \Delta_{2,r}, \ldots, \Delta_{L-1,f} + \Delta_{L,r}\} - \delta_e \\ &= MC_{\delta_e} - \delta_e \end{aligned}$$

Similarly, from $AC_{lb} = mc_{\delta_e}$ we have

$$\overline{AC}_{lb} = \delta_e + \delta_{0,r}$$

and

$$\widehat{AC}_{lb} = \max\{\delta_{0,f} + \delta_{1,r}, \delta_{1,f} + \delta_{2,r}, \ldots, \delta_{L-1,f} + \delta_{L,r}\}$$

Then Corollary 4.10 gives us,

$$\begin{aligned} AR_{lb} &= \max\{\overline{AC}_{lb} - \delta_e, \widehat{AC}_{lb} - \Delta_e\} \\ &= \max\{\delta_{0,r}, \max\{\delta_{0,f} + \delta_{1,r}, \delta_{1,f} + \delta_{2,r}, \ldots, \delta_{L-1,f} + \delta_{L,r}\} - \Delta_e\} \\ &= \max\{\delta_{0,r} + \Delta_e, \delta_{0,f} + \delta_{1,r}, \delta_{1,f} + \delta_{2,r}, \ldots, \delta_{L-1,f} + \delta_{L,r}\} - \Delta_e \\ &= mc_{\Delta_e} - \Delta_e \end{aligned}$$

So the theorem correctly derives critical bounds for AR from the critical bounds for AC in the case of a linear pipeline with a multiplication factor of $n = 1$.

Clearly we can derive bounds for AC from the bounds for AR as well as bounds for AR from the bounds for AC.

Corollary 4.11 *Let B be a behavioural graph and let AC_{ub} be derived from a critical path either (i) at level 0, or (ii) having negligible delays at level 0. Let $\overline{AR}_{ub}$ and $\widehat{AR}_{ub}$ be defined as in Definition 4.7. Then*

$$AC_{ub} = \max\{\overline{AR}_{ub} + \Delta_e, \widehat{AR}_{ub} + \delta_e\}$$

where AC_{ub} is the critical upper bound for average cycle time. Similarly we have

$$AC_{lb} = \max\{\overline{AR}_{lb} + \delta_e, \widehat{AR}_{lb} + \Delta_e\}$$

where AC_{lb} is the critical lower bound for average cycle time.

Proof. Lemma 4.8 and Theorem 4.9 together give us the following formulae.

$$\begin{aligned}
\overline{AR}_{ub} &= \overline{AC}_{ub} - \Delta_e \\
\widehat{AR}_{ub} &= \widehat{AC}_{ub} - \delta_e \\
AC_{ub} &= \max\{\overline{AC}_{ub}, \widehat{AC}_{ub}\}
\end{aligned}$$

By rearranging the first two equations and substituting them into the third we obtain the first part of the corollary. Similarly we can prove the lower bound.

□

4.7 Average Response Time for $n > 1$

We now turn our attention to pipelines with handshaking behaviours that have multiplication factor $n > 1$. When the multiplication factor of a cell is $n > 1$ we might expect that cell 0 determines the delay, because cell 0 communicates n times more frequently than the next cell. The following theorem proves that the delays of cell 0 do indeed dominate the bounds. In fact, given certain conditions, cell 0 and the environment are the *only* factors in determining the bounds. These conditions actually simplify the problem considerably. We restrict the range on the delays of the cells, such that cells deeper in the pipeline can be no more than n times slower than their predecessor cell. Since cell $i > 0$ may be exponentially slower than cell 0, these restrictions are slight.

We have the following theorem.

Theorem 4.12 *Let the maximum delays for each cell be bounded as follows.*

$$\begin{aligned}
\Delta_{i,f} &\leq \Delta_{0,f} \cdot n^i \\
\Delta_{i,r} &\leq \Delta_{0,r} \cdot n^i
\end{aligned}$$

The average response time AR for any pipeline (as defined in Section 2.3 operating under a delay model of Section 2.6), and with multiplication factor $n > 1$, is bounded as follows

$$\delta_{0,r} \leq AR \leq \max\{\Delta_{0,r}, \Delta_{0,r} + \Delta_{0,f} - \delta_e\}$$

The lower bound even applies when the cell delays $\Delta_{i,f}$ and $\Delta_{i,r}$ are not bounded. The upper bound is a critical bound when $\Delta_{i,f} = \Delta_{0,f} \cdot n^i$ and $\Delta_{i,r} = \Delta_{0,r} \cdot n^i$.

□

The proof is considerably more convoluted than that for $n = 1$. Before we prove the main theorem, we will need an auxiliary theorem. By using this auxiliary theorem we can reduce the number of possible paths we have to examine when calculating the bounds on AR.

Theorem 4.13 *Let $\Delta_{i,f} = \Delta_{0,f} \cdot n^i$ and $\Delta_{i,r} = \Delta_{0,r} \cdot n^i$ be given. Let paths P and P' exist between (r_h, i) and (a_h, j), for $h > 0$ and $j > i$. Let path $P \in (h, h+k]$ have maximal delays on each edge, and let P' be a level h path, also with maximal delays on each edge. Then the delay on P is the same as the delay on P'.*

In other words, if we have a path between request and acknowledgment events at level h, then the "direct" level h path has the same maximum delay as a path that goes deeper. A corollary of this is that all paths $P \in [h, h+k]$ between the two events have the same maximum delay.

Note that we explicitly exclude $h = 0$ from our reckoning, because the environment delays have no restrictions.

The proof proceeds by a nested induction. The outer induction is on the maximum depth of the path in the pipeline, k. The inner induction is on m, where $m \cdot n$

is the difference in occurrence indices of the first and last event in the path. So, intuitively, we have a two-dimensional induction. One dimension, k corresponds to the vertical axis, the other dimension, m corresponds to the horizontal axis.

Proof of Theorem 4.13. By induction on k.

Basis. Let $k = 1$. That is, we are exploring only paths in [h,h+1]. We prove the base case by a separate induction on m.

Basis. Let $m = 1$. Then $j = i + n$. Refer to Figure 4.5.

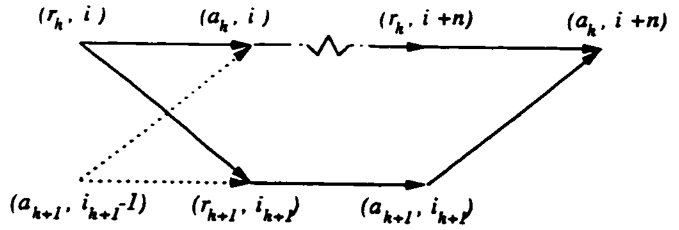


Figure 4.5: Example of path when $k = 1$ and $m = 1$.

We can trace a path P' directly at level h between $T(r_h, i)$ and $T(a_h, i + n)$. Assume that P' is a critical path. We have $n + 1$ edges of the form $r_h \rightarrow a_h$ on this path and n edges of the form $a_h \rightarrow r_h$. The delay on each $r_h \rightarrow a_h$ is maximised at $\Delta_{h,r}$. Likewise the delay on each $a_h \rightarrow r_h$ is at most $\Delta_{h-1,f}$. Then,

$$\begin{aligned}
\text{Delay}(P') &= T(a_h, i+n) - T(r_h, i) \\
&= (n+1)\Delta_{h,r} + n \cdot \Delta_{h-1,f} \\
&= (n+1)n^h\Delta_{0,r} + n^h \cdot \Delta_{0,f} \quad \{\text{By assumptions}\}
\end{aligned}$$

This is the delay on the level h path P'.

Now assume path $P \in (h, h+1]$ with $j = i + n$ is critical. There is only one such path, and it is $r_h \rightarrow r_{h+1} \rightarrow a_{h+1} \rightarrow a_h$. We can calculate the corresponding maximum delay.

$$\begin{aligned}
\text{Delay}(P) &= T(a_h, i+n) - T(r_h, i) \\
&= \Delta_{h,f} + \Delta_{h+1,r} + \Delta_{h,r} \\
&= n^h \cdot \Delta_{0,f} + n^{h+1} \cdot \Delta_{0,r} + n^h \cdot \Delta_{0,r} \quad \{\text{By assumptions}\} \\
&= (n+1)n^h \Delta_{0,r} + n^h \cdot \Delta_{0,f}
\end{aligned}$$

So the delay on P is the same as the delay on P'.

Step. Assume that our property holds for $j = i + m \cdot n$. Now we prove for the case when $j = i + (m+1) \cdot n$. Note that we still have $k = 1$. We are trying to prove that the property holds for paths in $[h, h+1]$ of arbitrary width. Refer to Figure 4.6.

By the inductive assumption the delay between $T(r_h, i)$ and $T(a_h, i + m \cdot n)$ is the same independent of whether the critical path is $P \in (h, h+1]$, or whether it is $P' \in [h, h]$. Let $T(a_h, i + m \cdot n) - T(r_h, i) = x$.

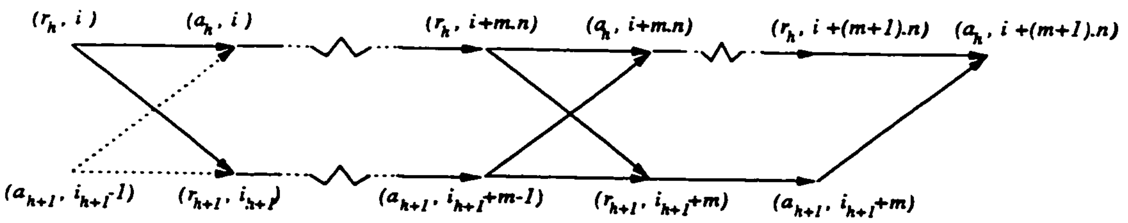


Figure 4.6: Example of path when $k = 1$ and $m \geq 1$.

We derive the delay $T(a_h, i + (m+1) \cdot n) - T(r_h, i)$. First we assume that the level h path, P', is critical. Between $(a_h, i + m \cdot n)$ and $(a_h, i + (m+1) \cdot n)$ there are n $r_h \rightarrow a_h$ and n $a_h \rightarrow r_h$ edges.

$$\begin{aligned}
\text{Delay}(P') &= T(a_h, i + (m+1) \cdot n) - T(r_h, i) \\
&= (T(a_h . i + (m+1) \cdot n) - T(a_h, i + m \cdot n)) \\
&\quad +(T(a_h . i + m \cdot n) - T(r_h, i)) \\
&= (T(a_h, i + (m+1) \cdot n) - T(a_h, i + m \cdot n)) + x \\
&= n \cdot \Delta_{h,r} + n \cdot \Delta_{h-1,f} + x \\
&= n^{h+1} \cdot \Delta_{0,r} + n^h \cdot \Delta_{0,f} + x \quad \{\text{By assumptions}\}
\end{aligned}$$

Now assume path $P \in (h, h+1]$ between (r_h, i) and $(a_h, i + (m+1) \cdot n)$ is a critical path. Path P is restricted to events at level $h+1$, except for the endpoints, and so must pass through point $(a_{h+1}, i_{h+1} + m - 1)$. See Figure 4.6. For brevity, we call this event α. We determine $T(\alpha)$.

First note that both the level h path and the path in $(h, h+1]$ between $T(r_h, i)$ and $T(a_h, i + m \cdot n)$ have delay x by our inductive assumption. The delays on each edge of each path are maximised. Now, observe that the two incoming edges to $(a_h, i + m \cdot n)$ must have maximal delay of $\Delta_{h,r}$. Thus both predecessors of $(a_h . i + m \cdot n)$, i.e., $(r_h . i + m \cdot n)$ and $(a_{h+1}, i_{h+1} + m - 1) = \alpha$, must occur at $T(a_h, i + m \cdot n) - \Delta_{h,r}$.

$$\begin{aligned}
T(\alpha) &= T(a_{h+1}, i_{h+1} + m - 1) \\
&= T(a_h, i + m \cdot n) - \Delta_{h,r} \\
&= T(r_h, i) + x - \Delta_{h,r}.
\end{aligned}$$

Path P is restricted to events at level $h+1$ except for the endpoints and so must pass through point α, i.e., $(a_{h+1}, i_{h+1} + m - 1)$. We have a path

$$(r_h, i) \rightsquigarrow \alpha \rightsquigarrow (a_h, i + (m+1) \cdot n)$$

The path $\alpha \rightsquigarrow (a_h, i+(m+1)\cdot n)$ consists of one $a_{h+1} \rightarrow r_{h+1}$ edge, one $r_{h+1} \rightarrow a_{h+1}$ edge, and one $a_{h+1} \rightarrow a_h$ edge. These have delays respectively of $\Delta_{h,f}$, $\Delta_{h+1,r}$ and $\Delta_{h,r}$.

$$\begin{aligned}
\text{Delay}(P) &= T(a_h, i+(m+1)\cdot n) - T(r_h, i) \\
&= (T(a_h, i+(m+1)\cdot n) - T(\alpha)) + (T(\alpha) - T(r_h, i)) \\
&= (\Delta_{h,f} + \Delta_{h+1,r} + \Delta_{h,r}) + (T(\alpha) - T(r_h, i)) \\
&= (\Delta_{h,f} + \Delta_{h+1,r} + \Delta_{h,r}) + (T(r_h, i) + x - \Delta_{h,r} - T(r_h, i)) \\
&= (\Delta_{h,f} + \Delta_{h+1,r} + \Delta_{h,r}) + (x - \Delta_{h,r}) \\
&= \Delta_{h.f} + \Delta_{h+1,r} + x \\
&= n^{h+1} \cdot \Delta_{0,r} + n^h \cdot \Delta_{0,f} + x
\end{aligned}$$

This is the same as the delay on path P' and we are done. We have now completed the base case for when $k = 1$.

Step. Assume that the property holds for k. We prove that the property holds for $k+1$. We wish to show that if we have a path $P \in (h, h+k+1]$, between (r_h, i) and (a_h, j), and we have a level h path P' between the two, then the maximum delay on each path is the same.

Note that path P must begin with an $r_h \rightarrow r_{h+1}$ edge and finish with an $a_{h+1} \rightarrow a_h$ edge. So we have a path $r_h \rightarrow r_{h+1} \rightsquigarrow a_{h+1} \rightarrow a_h$. The path $r_{h+1} \rightsquigarrow a_{h+1}$ is a path in $[h+1, h+k+1]$, or by setting $g = h+1$, path $r_g \rightsquigarrow a_g$ is a path in $[g, g+k]$.

The path $r_g \rightsquigarrow a_g$ is either a path at level g, or consists of segments S, some of which are paths at level g and some of which are paths $S \in (g, g+k]$.

Suppose that somewhere on the path $r_g \rightsquigarrow a_g$ there is a segment S of the form $(r_g, i_g) \rightsquigarrow (a_g, j_g)$ where $j_g = i_g + m \cdot n$, $m > 0$. Refer to Figure 4.7. By the

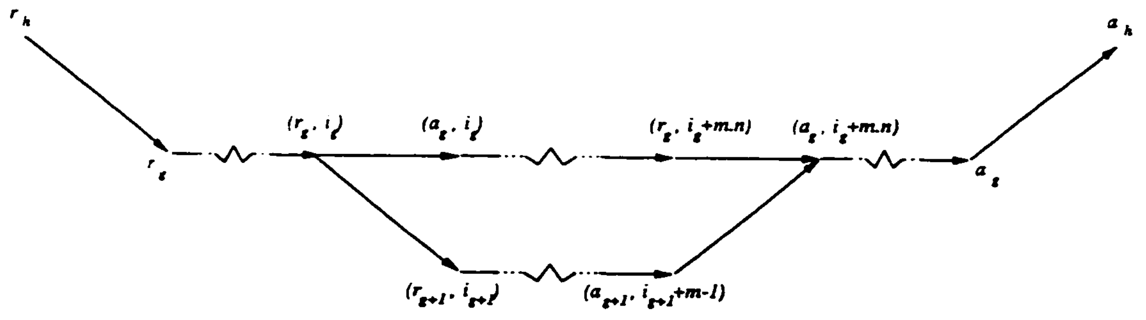


Figure 4.7: Example of possible critical path in $(h, h+k+1]$.

induction hypothesis, the maximal delay on this segment is identical whether it is a segment at level g or if it is a segment in $(g, g+k]$. Without loss of generality we assume that segment S is a level g path, and do the same for all other such segments. So, without loss of generality, we may assume that path $r_g \rightsquigarrow a_g$ is a level g path, and therefore path $P \in (h, h+1]$. We now have the base case, which we have already proven and we are done.

□

Now we are ready for the proof of Theorem 4.12.

Proof of Theorem 4.12

Because all $r_0 \rightarrow a_0$ edges have a minimum delay of $\delta_{0,r}$, we know that $AR \geq \delta_{0,r}$ and we have proven the lower bound.

Now we prove the upper bound, viz,

$$AR \leq \max\{\Delta_{0,r}, \Delta_{0,r} + \Delta_{0,f} - \delta_e\}$$

Let there be a critical path P, from from event i at level 0 to event j at level 0, with $j \gg i$. If j and i are sufficiently far apart then, without loss of generality, we let event i be a request and event j be an acknowledgement. We can do this because edge delays are assumed to be finite, and on an infinite path a single edge causes negligible extra delay. We choose i and j such that $j = i + m \cdot n$, for some $m > 0$. Because we are looking for a bound for worst-case behaviour we may assume that $\Delta_{i,f} = \Delta_{0,f} \cdot n^i$ and $\Delta_{i,r} = \Delta_{0,r} \cdot n^i$ and we derive our critical bound based on this assumption.

We assume the critical path P is within $[0,1]$ due to Theorem 4.13 above. Remember that Theorem 4.13 does not apply to level 0 paths as the environment delays are unbounded.

Suppose that the critical path P is a level 0 path. Then $T(a_0, j) - T(r_0, i) \leq (j - i + 1) \cdot \Delta_{0,r} + (j - i) \cdot (\text{Environment Delay})$. To determine the average response time between (r_0, i) and (a_0, j) we subtract the environment delay and divide by the number of cycles. The number of cycles is $j - i + 1$, so if the critical path P is a level 0 path, the average response time satisfies $AR \leq \Delta_{0,r}$.

If the critical path P is not a level 0 path then we can break P into segments. There is at least one segment S of the form $S \in (0, 1]$. Let S start at (r_0, i_0) and end at (a_0, j_0). Then $j_0 = i_0 + m' \cdot n$ with $m' > 0$. We can calculate the maximum delay on S. S consists of the first request/request edge, $2m' - 1$ "horizontal edges", and one final acknowledgment/acknowledgment edge.

$$\begin{aligned}
& T(a_0, i_0 + m' \cdot n) - T(r_0, i_0) \\
= \;& \Delta_{0,f} + m' \cdot \Delta_{1,r} + (m' - 1) \cdot \Delta_{0,f} + \Delta_{0,r} \\
& \qquad \{\text{ From assumptions}\} \\
= \;& \Delta_{0,f} + m' \cdot n \cdot \Delta_{0,r} + (m' - 1) \cdot \Delta_{0,f} + \Delta_{0,r} \\
& \qquad \{\text{ Since } n \geq 1\} \\
\leq \;& \Delta_{0,f} + m' \cdot n \cdot \Delta_{0,r} + (m' - 1)n \cdot \Delta_{0,f} + \Delta_{0,r}
\end{aligned}$$

We derive the average response time over this segment by subtracting the number of environment delays, and dividing by the number of cycles at level 0.

$$\begin{aligned}
AR_{ub} &= (\Delta_{0,f} + m' \cdot n \cdot \Delta_{0,r} + (m'-1)n \cdot \Delta_{0,f} \\
&\quad + \Delta_{0,r} - m' \cdot n \cdot \delta_e)/(m' \cdot n + 1) \\
&= (\Delta_{0,f} - \Delta_{0,r} + (m'-1)n \cdot \Delta_{0,f} + \Delta_{0,r} + \delta_e)/(m' \cdot n + 1) \\
&\quad + \Delta_{0,r} - \delta_e \\
&= (\Delta_{0,f} + (m'-1)n \cdot \Delta_{0,f} + \delta_e)/(m' \cdot n + 1) + \Delta_{0,r} - \delta_e \\
&= (\Delta_{0,f} + (m' \cdot n + 1) \cdot \Delta_{0,f} - n \cdot \Delta_{0,f} - \Delta_{0,f} + \delta_e)/(m' \cdot n + 1) \\
&\quad + \Delta_{0,r} - \delta_e \\
&= (\delta_e - n \cdot \Delta_{0,f})/(m' \cdot n + 1) + \Delta_{0,f} + \Delta_{0,r} - \delta_e
\end{aligned}$$

First let us consider the case where $\delta_e > n \cdot \Delta_{0,f}$. We maximise over $m' > 0$.

$$\begin{aligned}
AR_{ub} &= \max\{(\delta_e - n \cdot \Delta_{0,f})/(m' \cdot n + 1) + \Delta_{0,f} + \Delta_{0,r} - \delta_e \mid 1 \leq m'\} \\
&\quad \{\text{ Maximise at } m' = 1\} \\
&= (\delta_e - n \cdot \Delta_{0,f})/(n + 1) + \Delta_{0,f} + \Delta_{0,r} - \delta_e \\
&= (\Delta_{0,f} - n \cdot \delta_e)/(n + 1) + \Delta_{0,r} \\
&\leq (n \cdot \Delta_{0,f} - n \cdot \delta_e)/(n + 1) + \Delta_{0,r} \quad \{\Delta_{0,f} \geq 0\} \\
&< (\delta_e - n \cdot \delta_e)/(n + 1) + \Delta_{0,r} \quad \{\text{ From assumption above}\} \\
&\leq (0)/(n + 1) + \Delta_{0,r} \quad \{n > 1 \text{ and } \delta_e \geq 0\} \\
&= \Delta_{0,r}
\end{aligned}$$

We have shown that if $\delta_e > n \cdot \Delta_{0,f}$ then $AR < \Delta_{0,r}$. But $\Delta_{0,r}$ is the delay through cell 0 alone and is the value of AR when using a level 0 path. We can therefore surmise that if $\delta_e > n \cdot \Delta_{0,f}$, the critical path is at level 0 and goes no deeper in the pipeline.

Let us now consider the case when $\delta_e \leq n \cdot \Delta_{0,f}$. We maximise over $m' > 0$.

$$\begin{aligned}
AR_{ub} &= \max\{(\delta_e - n \cdot \Delta_{0,f})/(m' \cdot n + 1) + \Delta_{0,f} + \Delta_{0,r} - \delta_e \mid 1 \leq m'\} \\
&\quad \{\text{ Maximise at } m' \to \infty\} \\
&= \max\{(0)/(m' \cdot n + 1) + \Delta_{0,f} + \Delta_{0,r} - \delta_e \mid 1 \leq m'\} \\
&= \Delta_{0,f} + \Delta_{0,r} - \delta_e
\end{aligned}$$

The contribution of delay by level 0 paths is negligible here.

Combining the above results, and maximising over all possible segments, we get for the average response time

$$AR \leq \max\{\Delta_{0,r}, \Delta_{0,r} + \Delta_{0,f} - \delta_e\}$$

□

4.7.1 Interpreting the Results.

As with worst-case response time, cell 0 is the only cell that matters for a bound on average response time. We can allow cells deeper in the pipeline to be exponentially slow without affecting the bounds on average response time. So, for both average and worst-case response time we need to optimise only cell 0 for speed and make sure that the other cells satisfy the delay bounds.

4.8 Average Cycle Time for $n > 1$.

We now extend the results to average cycle time, AC.

Theorem 4.14 *Let the maximum delays for each cell be bounded as follows.*

$$\begin{aligned}\Delta_{i,f} &\leq \Delta_{0,f} \cdot n^i \\ \Delta_{i,r} &\leq \Delta_{0,r} \cdot n^i\end{aligned}$$

The average response time AC for any pipeline (as defined in Section 2.3 operating under a delay model of Section 2.6) and with multiplication factor $n > 1$ is bounded as follows

$$\delta_{0,r} + \delta_e \leq AC \leq \Delta_{0,r} + \max\{\Delta_e, \Delta_{0,f}\}$$

The lower bound even applies when the cell delays $\Delta_{i,f}$ and $\Delta_{i,r}$ are not bounded. The upper bound is a critical bound when $\Delta_{i,f} = \Delta_{0,f} \cdot n^i$ and $\Delta_{i,r} = \Delta_{0,r} \cdot n^i$.

Proof. Because all $r_0 \to a_0$ edges have a minimum delay of $\delta_{0,r}$, and all $a_0 \to r_0$ edges have a minimum delay of δ_e, we know that $AC \geq \delta_{0,r} + \delta_e$ and we have proven the lower bound.

There are two ways we could derive the upper bound. We could derive the value by using the same path tracing algorithm or we can derive the bound for AC from the bound AR simply by using Corollary 4.11 and Lemma 4.8 from Section 4.6. Lemma 4.8 defines AR_{ub} as

$$AR_{ub} = \max\{\overline{AR}_{ub}, \widehat{AR}_{ub}\}$$

where $\overline{AR}_{ub}$ is derived from a level 0 path, and $\overline{AR}_{ub}$ is derived from a path in which the contribution of level 0 paths is negligible. The proof of the upper bound for AR in Theorem 4.12 splits the path calculation into two. One path is the level 0, and one path is a path in $(0,1]$. One of these paths is the critical path. These paths respectively derive $\overline{AR}_{ub} = \Delta_{0,r}$ and $\widehat{AR}_{ub} = \Delta_{0,r} + \Delta_{0,f} - \delta_e$. Theorem 4.12 gives

$$\begin{aligned} AR_{ub} &= \max\{\Delta_{0,r}, \Delta_{0,r} + \Delta_{0,f} - \delta_e\} \\ &= \max\{\overline{AR}_{ub}, \widehat{AR}_{ub}\} \end{aligned}$$

Because the critical path is either (i) a path at level 0, or (ii) a path with negligible delay at level 0, we can apply Corollary 4.11 which states

$$\begin{aligned} AC_{ub} &= \max\{\overline{AR}_{ub} + \Delta_e, \widehat{AR}_{ub} + \delta_e\}\} \\ &= \max\{\Delta_{0,r} + \Delta_e, \Delta_{0,r} + \Delta_{0,f}\} \\ &= \Delta_{0,r} + \max\{\Delta_e, \Delta_{0,f}\} \end{aligned}$$

□

4.9 Conditional Behaviour

As with the worst-case analysis of the previous chapter, let us assume that we have a behaviour of the form

$$(r_i?; *[\textbf{if } B_i \textbf{ then } (a_i!;\ r_i?)^n \,\|(r_{i+1}!;\ a_{i+1}?) \textbf{ else } (a_i!;\ r_i?)^n \textbf{ fi }])$$

The guard B_i may depend on data values that have been communicated. If $B_i =$ false, no handshake takes place with the right-hand neighbour.

We prove that the upper bounds for AR and AC still hold for this behaviour. First we show that the AR bound still holds. The guards B_i restrict the depth to which any critical path may reach, but do not introduce any new paths. The bounds on average response time are derived from the critical path. This critical path is the maximum delay path of a set of paths. Therefore, reducing the number of paths can only reduce the bound. The bound for non-conditional behaviour AR may not be tight for conditional behaviour. A similar argument holds for AC.

The lower bounds for AR and AC may no longer hold. For example when we find a lower bound for AC, we minimise the delays on all edges. We then find the critical path, i.e, the longest path between two events arbitrarily far apart when all delays are minimised. By a similar argument to that given above, the more paths that are possible, the larger the delay is likely to be. With fewer paths we will likely have a smaller delay in the best case. The minimum possible number of paths occurs when all guards are false. Therefore we can calculate lower bounds by assuming that the pipeline has minimum length, i.e. $L = 0$. Hence, $AR \geq \delta_{0,r}$ and $AC \geq \delta_{0,r} + \delta_e$.

It is important to note that these bounds are no longer critical bounds. Conditional behaviour is non-deterministic. We do not know what the critical path is, so we cannot compute the critical bound.

4.10 An Example: The Eager Stack of Hulgaard et al.

Hulgaard and Burns [27] give an example of a pipeline called an eager stack. The eager stack is a linear array of data-storing cells. On a *Put* operation, all data already in the stack is moved one place deeper. On a *Get* operation, all data already in the stack is moved one place closer to the top. We choose this example, because they have done a separation analysis similar to ours. We can compare our results, derived by formula, with their results, derived by an iterative computation. We will be calculating cycle time, as this is one of the measures they use.

We make two simplifications to their model. We are only interested in the communications between cells, so we do not distinguish between a "put" or a "get". Since the pipeline is a stack, the decision to communicate with the next cell is dependent on the number of data elements currently stored in the stack. We can model this as the conditional behaviour given in Section 4.9. We know that the upper bound for AC in this conditional behaviour is the same as for non-conditional behaviour.

We have two problems translating their model to ours.

- The possible delay assignment allowed by Hulgaard et al. is more restrictive than our model allows. We assign a delay to every edge in our behavioural graphs independent of all other edges. This may not be true for all behaviours. For example, some behaviours may assign identical delays to edges $(r_0, i) \rightarrow (a_0, i)$ and $(r_0, i) \rightarrow (r_1, j)$. That is, these edges cannot be assigned delays independent of each other. This lack of independence may mean we may not be able to give tight results for RT and CT, though our results will still be bounds.

- Since our model uses handshakes, and the model used in Hulgaard and Burns assumes an underlying synchronization mechanism, we have to massage the model somewhat to make sure our model is consistent with theirs. For ease of understanding, we model their pipeline as a micropipeline. See Figure 4.8. There are four cycles within this model, synchronized by C elements. Each of these cycles represents one of the four Petri nets in Figure 12 of [27] that are composed to form the original net.

Our model is shown in Figure 4.8. The P_i markings give some idea of the original synchronisation points. As with the model in Hulgaard et al. all delays have the range $[1,2]$ except the environment and end-cell which have zero delay. Note that our method calculates the delay between requests at the environment, whereas Hulgaard et al. calculate it between consecutive P_0 operations. We solve this problem by discarding the left-hand stage since the loop between P_0 and the environment has maximum delay less than MC_{δ_e} and minimum delay less than mc_{Δ_e} and do not influence our results below. Our trimmed pipeline is illustrated in Figure 4.9. In Section 6.2 we will show, that for the bounds on average cycle time, this pruning step is unnecessary.

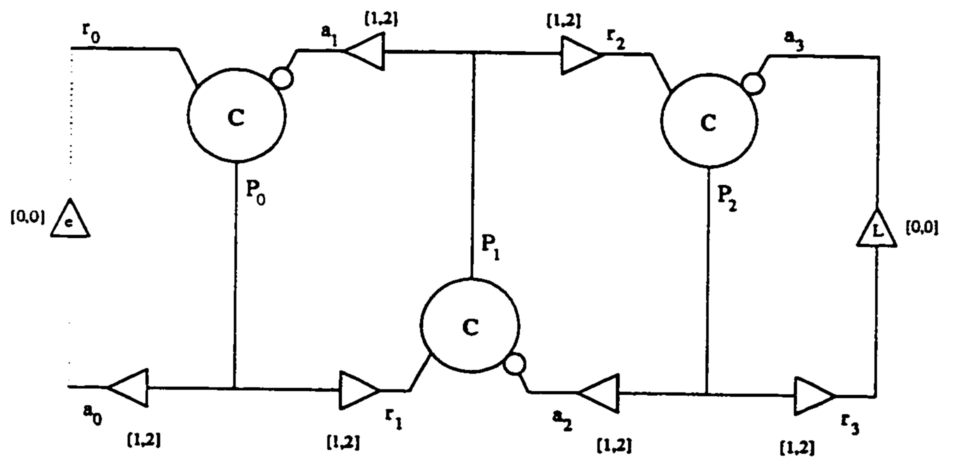


Figure 4.8: The eager stack modelled by a micropipeline with length $L = 3$ and multiplication factor $n = 1$. The original synchronisation points in the model of [27] are labelled $P_0 \ldots P_2$.

For this example the parameters are $n = 1$ and $L = 3$. We can now easily

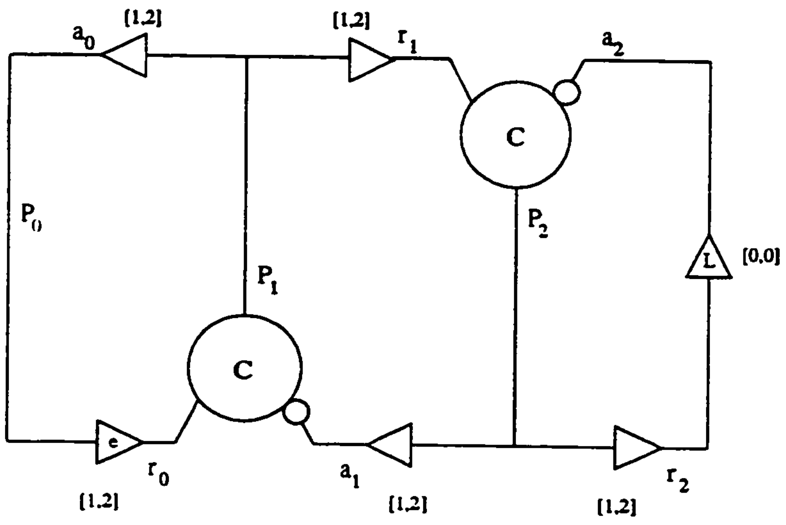


Figure 4.9: The eager stack modelled by a micropipeline with length $L = 2$ and multiplication factor $n = 1$, modified so that P_0 is at the entrance of the pipeline.

calculate the value of AC. Since $AC \leq MC_{\delta_e}$, and $MC_{\delta_e} = 4$ in this example, we have that $AC \leq 4$. This is also the value given in the Hulgaard et al. paper.

We calculate an upper bound for CT also. Below we skip some steps.

$$\begin{aligned}
CT &\leq \Delta_e + \max(\{\Delta_{k,r} + \Delta_{k-1,f} + \sum_{h=0}^{k-1}(\Delta_{h,r} - \delta_{h,r}) - \delta_e \mid 1 \leq k \leq L\} \\
&\quad \cup\{\Delta_{0,r}\}) \\
&\quad\quad \{\text{ Maximum occurs at } k = 1\} \\
&= \Delta_e + \Delta_{1,r} + \Delta_{0,f} + \sum_{h=0}^{0}(\Delta_{h,r} - \delta_{h,r}) - \delta_e \\
&= 2 + 2 + 2 + (2 - 1) - 1 \\
&= 6
\end{aligned}$$

Their result is $CT \leq 5$. The discrepancy occurs, because we take cycle time from request to request rather than acknowledgment to acknowledgment. See Lemma 3.6.

Most of the work in this example is taking another model and translating it to ours. Our model is slightly different, and the measures we use are slightly different, but the bounds provided by the formulae are still relevant and valid. Once the example is translated to our model, the results can be calculated trivially.

Note that our formulae easily generalise to stacks with other parameter values. Hulgaard et al. would have to execute their algorithm for every new set of values.

4.11 Summary

We have calculated bounds on average response time and average cycle time in this chapter. This chapter has proven the following results.

- The average response time, AR, for pipelines with multiplication factor, $n = 1$ is bounded as follows.

  $$mc_{\Delta_e} - \Delta_e \leq \; AR \; \leq MC_{\delta_e} - \delta_e$$

  The two bottlenecks mc_{Δ_e} and MC_{δ_e} constrain the throughput and these bottlenecks are calculated from the delays between each neighbouring pair of cells. The formula is largely independent of cell ordering, though we must consider which cells are adjacent. The formula is independent of the pipeline length L. Therefore, when we optimise such a pipeline, we have some freedom to rearrange the pipeline or add cells without affecting the bounds on the average response time.

- The average cycle time AC for pipelines with multiplication factor $n = 1$ is bounded as follows.

  $$mc_{\delta_e} \leq AC \leq MC_{\Delta_e}$$

- The average response time AR and average cycle time AC for pipelines with multiplication factor $n > 1$ are bounded as follows.

$$\begin{aligned} \delta_{0,r} &\leq AR \leq && \max\{\Delta_{0,r}, \Delta_{0,r} + \Delta_{0,f} - \delta_e\} \\ \delta_{0,r} + \delta_e &\leq AC \leq && \Delta_{0,r} + \max\{\Delta_{0,f}, \Delta_e\} \end{aligned}$$

  provided that $\Delta_{i,f} \leq \Delta_{0,f} \cdot n^i$ and $\Delta_{i,r} \leq \Delta_{0,r} \cdot n^i$ for $0 \leq i \leq L$. We optimise such a pipeline by optimising cell 0 and the environment for speed. We have some freedom to optimise other cells for low power or low area without affecting the worst-case bounds on average response time or average cycle time.

- We relate the bounds for AR and AC so that we may easily calculate one value from the other.

- The bounds for average response time and average cycle time calculated for non-conditional behaviour are also bounds for the conditional behaviour described in Section 3.6.

Chapter 5

Trees

5.1 Tree-Based Networks

In software, many data structures are implemented by means of trees. For example, by ordering the data stored in a tree one can construct efficient data structures suitable for searching, insertion, deletion, or sorting. To implement tree-based data structures in hardware one could use a tree architecture, where each cell contains one item of data. For example, a stack can be implemented as a tree where we can use an efficient algorithm for push and pop operations.

An advantage of tree architectures is that they minimise the number of communications between cells, thereby reducing power consumption. The idea is that the number of cells a communication passes through is smaller than with a linear pipeline. For example, the depth of a balanced tree is only logarithmic in the number of cells, and hence the maximum number of communications may be less than for a pipeline with similar capacity. Potentially the tree has small response times for the same reason. A high-level model of a tree is given in Figure 5.1.

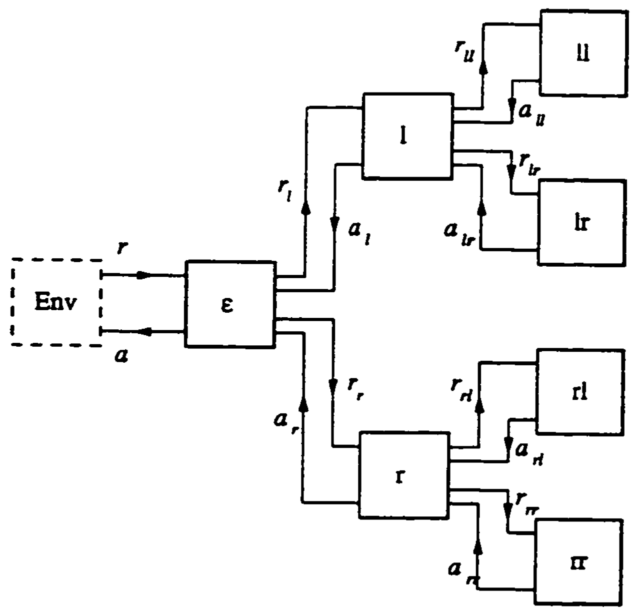


Figure 5.1: A tree of cells with depth $L = 2$ and arity $n = 2$.

Again we can ask questions about the delay behaviour of a tree architecture much as we did for the pipelines in the previous chapter. We calculate response time, average response time, and the corresponding analogues for cycle time in this chapter. This time we have to worry about how the branching of a tree affects the tree's speed. It is not immediately obvious how communications on one branch of the tree interfere with the timing of communications on another branch. Moreover, we still have to worry about variation in cell delays and the differences between forward and reverse delays.

Another interesting tree architecture can be seen in Figure 5.2, where we abut two trees. We can construct a FIFO like this, as shown in [7]. Data passes from the left-hand environment to the right-hand environment. Each consecutive item of data takes a slightly different path from its predecessor. For example, each cell in the left half of the FIFO alternates between *sending* data to one child and its other child. In the right half of the FIFO each cell alternates between *receiving* data from one child or from its other child. This architecture presents us with new analysis problems, since we need to understand how the two trees interact. Combining two trees in this way introduces many new possible critical paths. We will need the results of both this chapter and the next to answer questions about the throughput of this FIFO.

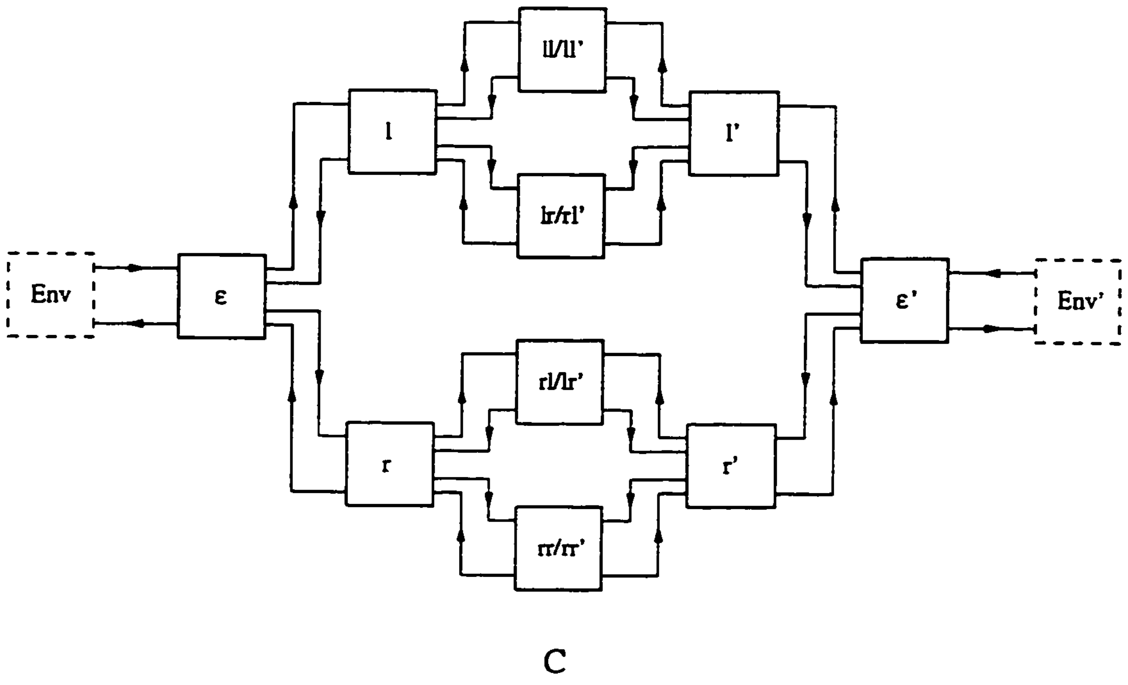


Figure 5.2: Two trees joined.

There are several possible disadvantages of tree architectures. One problem is the delay and extra circuitry required for steering data. That is, we need extra circuitry to switch data to one branch of a tree or to another. This is one of the key problems described by Brunvand in his discussion of tree-based FIFOs [7].

Another problem is layout. When implementing a tree architecture on a chip, there are two problems. Area directly affects the cost of circuit fabrication, and tree-based circuits waste a lot of space. Ideally all circuits are square and tile nicely on a chip. Another problem in layout is that the wires that pass between the cells are longer in a tree than between cells in a pipeline. Wires are an important component of the delay of a circuit and transitions on wires dissipate energy. Longer wires mean less speed and higher energy consumption. The speed penalty is unlikely to be too costly, as we will see below, but the power consumption may be a factor.

We can analyse trees in the same way as we did pipelines. Our parameter n refers to the arity of the tree. For example, binary trees would have arity $n = 2$. When $n = 1$ we have a pipeline, and when $n = 3$ we have a ternary tree. We consider a restricted subset of possible tree behaviours, viz, trees that direct communications to each branch in turn. For example, in a binary tree we specify that each cell alternates between communicating between its left child and its right child. We will generalise this behaviour at the end of this chapter.

For the most part, we will consider trees where the arity is $n > 1$, and we will use a binary tree for examples. The arity of the tree is similar to the multiplication factor given for pipelines. As one might expect, the results for trees have similarities to those given for pipelines with $n > 1$.

- Response Time, RT, is bounded as follows

$$\begin{aligned} RT \leq\ & \max(\{\Delta_{k,r} + \Delta_{k-1,f} + \sum_{h=0}^{k-1}(\Delta_{h,r} - \delta_{h,r}) \\ & -(\delta_{0,r} + \delta_e)n^{k-1} + \delta_{0,r} \mid 1 \leq k \leq L\} \cup \{\Delta_{0,r}\}) \end{aligned}$$

The parameters are analogous to those used for pipelines. Each cell has a set of delay parameters based on the depth h of a cell in the tree. All cells at depth h have the same delay bounds. Again, simplicity is the main reason for these parameter choices. In particular, choosing delay bounds that are identical for all cells at level h simplifies the analysis greatly, and is a reasonable assumption in general. Typically one might expect one branch of the tree to have functionality (and hence circuitry) identical to other branches. Therefore each branch of the tree would have identical delay bounds at each level. By choosing parameters that are analogous to those for pipelines, it makes it easier for us to compare trees with pipelines.

This differs from the result given in Chapter 3 only in the factor n^{k-1}. For pipelines this factor was n^k. If $n = 1$, the result is the same as for pipelines. When $n > 1$, the first cell and the environment play a slightly smaller role than for linear pipelines.

Since the parameter n is greater than 1, and since delays δ_e and $\delta_{0,r}$ are positive, the upper bound for response time of pipelines $RT_{ub,pipelines}$ given in Chapter 3 is less than or equal to the upper bound for the response time of trees RT_{ub} given here. That is,

$$RT_{ub,pipelines} \leq RT_{ub}$$

This occurs, because cell 0 in a tree performs more communications, as it has to deal with all branches rather than just one, and delay is associated with each communication.

- Cycle time for trees, CT, is bounded as follows

$$CT_{ub} \;=\; RT_{ub} + \Delta_e$$

where RT_{ub} is an upper bound on the response time RT. This is as predicted by Lemma 3.5.

We then have

$$CT_{ub,pipelines} \leq CT_{ub}$$

- The average response time, AR, for trees with arity $n > 1$ is bounded as follows.

$$\delta_{0,r} \leq AR \leq \max\{\Delta_{0,r}, \Delta_{1,r} + \Delta_{0,f} - \delta_e\}$$

given $\Delta_{i,f} \leq \Delta_{0,f} \cdot n^i$ and $\Delta_{i,r} \leq \Delta_{0,r} \cdot n^i$ for $0 \leq i \leq L$. That is, a cell may be exponentially slower than cell 0 depending on its depth in the pipeline. The lower bound even applies when the cell delays $\Delta_{i,f}$ and $\Delta_{i,r}$ are not bounded.

Compare these bounds for trees with those for pipelines. The bounds for pipelines with $n > 1$ are

$$\delta_{0,r} \leq AR_{pipelines} \leq \max\{\Delta_{0,r}, \Delta_{0,r} + \Delta_{0,f} - \delta_e\}$$

We then have

$$AR_{ub,pipelines} \leq AR_{ub}$$

The lower bound is the same for both pipelines and trees.

As with pipelines with $n > 1$ in most cases the delays of cell 0 and the environment are the only ones that are important. This would indicate that problems with wire delays beyond the root may not be significant. Secondly, note that we can optimise the layout as we wish, and optimise all cells, except the root cell, without affecting the speed, as long as we are careful not to exceed the exponential delay bounds of the cells. For example, we can make the transistors smaller which will typically reduce speed, energy consumption, and area for individual cells. We could conceivably arrange the tree in a squarer configuration at the expense of longer wiring. This would increase energy consumption, and increase some delays. In both cases, we should be able to avoid increasing the bound on AR.

- The average cycle time, AC, for trees with arity $n > 1$ is bounded as follows.

  $$\delta_{0,r} + \delta_e \leq \quad AC \quad \leq \max\{\Delta_{0,r} + \Delta_e, \Delta_{1,r} + \Delta_{0,f}\}$$

  given $\Delta_{i,f} \leq \Delta_{0,f} \cdot n^i$ and $\Delta_{i,r} \leq \Delta_{0,r} \cdot n^i$ for $0 \leq i \leq L$. The lower bound even applies when the cell delays $\Delta_{i,f}$ and $\Delta_{i,r}$ are not bounded.

  Compare these bounds with the bounds for pipeline average cycle time when $n > 1$.

  $$\delta_{0,r} + \delta_e \leq \quad AC_{pipelines} \quad \leq \Delta_{0,r} + \max\{\Delta_{0,f}, \Delta_e\}$$

  We then have

  $$AC_{ub,pipelines} \leq AC_{ub}$$

  The lower bound is the same for both pipelines and trees.

- We discuss more general behaviours in which the handshakes are conditionally passed deeper in the tree.

5.2 The Behaviour

We formalise our notion of the behaviour of a tree. There are many possible behaviours to choose from. We choose a behaviour that is an extension of the behaviour we have for pipelines. If we choose a tree with arity $n = 1$, we should get a pipeline with multiplication factor $n = 1$. We change only the behaviour, not the behavioural model itself. Moreover we do not change the delay model, and also the parameters and their meanings are kept the same as those used for pipelines.

Consider a tree where each internal cell for each handshake with its parent has one handshake with one of its children. The handshakes with the children, however, alternate between left and right child for a binary tree. The communication behaviours are given by

$$(r_i?;\ *[\,(a_i!;\ r_i?)\|(rl_{i+1}!;\ al_{i+1}?);\ (a_i!;\ r_i?)\|(rr_{i+1}!;\ ar_{i+1}?)\,])$$

The terminals rl and al connect the internal node with its left child, the terminals rr and ar with its right child. See Figure 5.3.

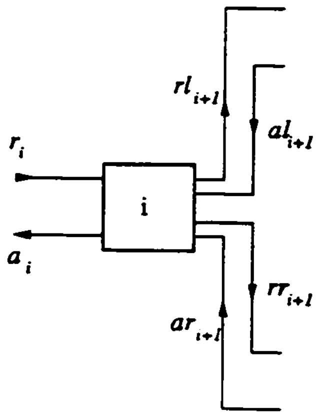


Figure 5.3: One cell of a tree.

To adapt our old proofs we use a few simplifying definitions. Let Σ be a set of subscript labels. Σ contains n elements. In the $n = 2$ case, i.e., a binary tree, we can choose labels r and l representing right and left. The example tree is shown in Figure 5.1. In this case, $\Sigma = \{r, l\}$. In the following examples s is a string of elements from Σ. That is, $s \in \Sigma^*$. Rewriting the communication behaviour for a binary tree, we have.

$$(r_s?;\ *[\ (a_s!;\ r_s?) \| (r_{sl}!;\ a_{sl}?);\ (a_s!;\ r_s?) \| (r_{sr}!;\ a_{sr}?)\ ])$$

For the more general n-ary tree we have a behaviour

$$\begin{aligned}(r_s?;\ *[\ (a_s!;\ r_s?) \| (r_{sx_1}!;\ a_{sx_1}?);\ (a_s!;\ r_s?) \| (r_{sx_2}!;\ a_{sx_2}?);\\ \cdots;(a_s!;\ r_s?) \| (r_{sx_n}!;\ a_{sx_n}?)\ ])\end{aligned}$$

where x_i, $1 \le i \le n$ are the distinct elements of Σ. We generalise this behaviour later in the chapter.

The delay model is the same as that used for linear pipelines. Edges ending in a request are considered "forward" delays. Edges ending in an acknowledgment are considered "reverse" delays. For the response-time proof we assume that all cells at level i have the same delay bounds. We allow a bit more flexibility for the average-time proofs.

As well as using the string s in examples, we will use the following conventions. Labels $b, c \in \Sigma$ are single elements. For example sb is the concatenation of element b to the end of string s. $|\cdot|$ is the function from Σ^* to the non-negative integers that determines the length of a word in Σ^*. For example, ϵ is the empty word, and $|\epsilon| = 0$. Note that $|s|$ gives the depth in the tree at which an event occurs. For example, let r_{bc} be an event. Since $|bc| = 2$, r_{bc} occurs at depth 2.

We are going to show for trees that

$$\begin{aligned} RT \ \le\ & \max(\{\Delta_{k,r} + \Delta_{k-1,f} + \sum_{h=0}^{k-1}(\Delta_{h,r} - \delta_{h,r}) \\ & -(\delta_{0,r} + \delta_e)n^{k-1} + \delta_{0,r} \mid 1 \le k \le L\} \cup \{\Delta_{0,r}\}) \end{aligned}$$

Note the factor n^{k-1} that occurs here, in contrast to the factor n^k that occurred in the formula for pipelines. This is the only difference between the equations.

The proof is almost identical to the original proof for worst-case response times of linear pipelines. The major difference is that we relabel the subscripts. We can use $|\cdot|$ as a mapping between the proof for trees and the linear pipeline case. Suppose that $|s| = i$. We replace i in the original proof with s. We replace $i+1$ with sb. Since all cells at level i are identical it doesn't matter which s and b we choose (as long as $|s| = i$).

The graph corresponding to the binary tree of Figure 5.1 is given in Figure 5.4. A ternary tree would not be much harder to draw, but increasing the depth of the tree beyond $L = 2$, would make the graph exceedingly complex.

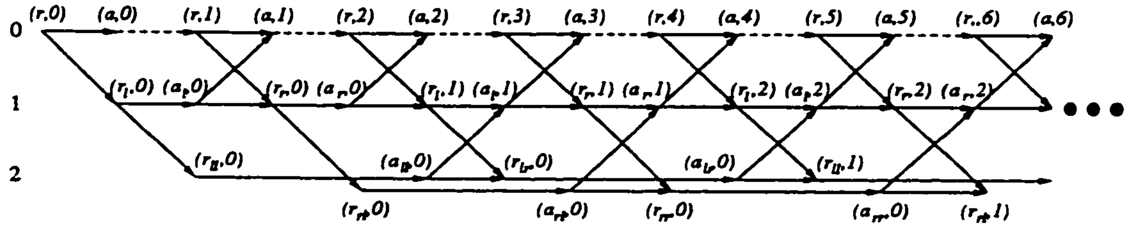


Figure 5.4: Unfolded process graph for a binary tree of cells with tree depth $L = 2$.

Response depth is defined as in Chapter 3. Hence, response depth refers to the number of edges in a critical path of acknowledgments.

Lemma 3.2 still holds without further proof. This lemma states that if the response depth of an acknowledgment (a_k, j_k) is zero, then the separation between that acknowledgment and the corresponding request is bounded to

$$T(a_k, j_k) - T(r_k, j_k) \leq \Delta_{k,r}$$

The proof of this lemma for trees is identical to that for pipelines. Now we are ready for the following lemma.

Lemma 5.1 *If the response depth of (a_s, j_s) is $k, 1 \le k \le L - |s|$, then event $(a_s, j_s - n^{k-1})$ exists and we have*

$$T(a_s, j_s) - T(a_s, j_s - n^{k-1}) \le \Delta_{|s|+k,r} + \Delta_{|s|+k-1,f} + \sum_{h=0}^{k-1}(\Delta_{|s|+h,r} - \delta_{|s|+h,r})$$

□

Note that $n = 2$ for binary trees.

Proof. By induction on k.
Basis. Assume that (a_s, j_s) has a response depth of $k = 1$. Let the edge $(a_{sb}, j_{sb}) \to (a_s, j_s)$ exist, then (a_{sb}, j_{sb}) has response depth 0. We have a dependency graph as in Figure 5.5.

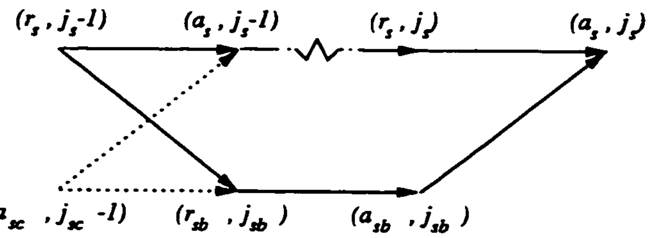


Figure 5.5: Example of paths used in the base case. Dotted edges may not necessarily be present. The wavy line is a path of one or more edges.

First we deal with the statement that event $(a_s, j_s - n^{k-1})$ exists. Let us look at the behaviour of a cell at level $|s|$.

$$(r_s?;\ *[\ (a_s!;\ r_s?)\|(r_{sx_1}!;\ a_{sx_1}?);\ (a_s!;\ r_s?)\|(r_{sx_2}!;\ a_{sx_2}?);$$

$$\cdots ; (a_s!;\ r_s?) \| (r_{sx_n}!;\ a_{sx_n}?)\ ])$$

where x_k are the distinct elements of Σ. If (a_s, j_s) exists and has a response depth greater than 0 then it must depend on an acknowledgment from a deeper level. Therefore, from the behaviour given above, (a_s, j_s) must be at least the second occurrence of a_s. Therefore $(a_s, j_s - 1)$ exists. This can easily be seen from Figure 5.5.

From the definition of $T_{pred}(c)$ and the dependency graph in Figure 5.5, we have $T_{pred}(r_{sb}, j_{sb}) = T_{pred}(a_s, j_s - 1)$. In the proof below we take $T_{pred} = T_{pred}(a_s, j_s - 1)$. We wish to derive an upper bound for $T(a_s, j_s) - T(a_s, j_s - 1)$. We observe

$T(a_s, j_s) - T(a_s, j_s - 1)$

$=$ { Break path into 4 edges. }

$(T(a_s, j_s) - T(a_{sb}, j_{sb})) + (T(a_{sb}, j_{sb}) - T(r_{sb}, j_{sb})) + (T(r_{sb}, j_{sb}) - T_{pred}) + (T_{pred} - T(a_s, j_s - 1))$

$\leq$ { (a_s, j_s) has positive response depth so timing dependency exists. Also, ack/ack pair is a "reverse" delay. }

$\Delta_{|s|,r} + (T(a_{sb}, j_{sb}) - T(r_{sb}, j_{sb})) + (T(r_{sb}, j_{sb}) - T_{pred}) + (T_{pred} - T(a_i, j_i - 1))$

$\leq$ { (a_{sb}, j_{sb}) has resp. depth 0, Lemma 3.2 }

$\Delta_{|s|,r} + \Delta_{|s|+1,r} + (T(r_{sb}, j_{sb}) - T_{pred}) + (T_{pred} - T(a_s, j_s - 1))$

$=$ { $T_{pred} = T_{pred}(a_s, j_s - 1) = T_{pred}(r_{sb}, j_{sb})$ }

$\Delta_{|s|,r} + \Delta_{|s|+1,r} + (T(r_{sb}, j_{sb}) - T_{pred}(r_{sb}, j_{sb})) + (T_{pred} - T(a_s, j_s - 1))$

$\leq$ { From definition of timing assignments and because edges ending in request are forward delays }

$\Delta_{|s|,r} + \Delta_{|s|+1,r} + \Delta_{|s|,f} + (T_{pred} - T(a_s, j_s - 1))$

$\leq$ { Minimum edge delay is $\delta_{|s|,r}$ }

$\Delta_{|s|,r} + \Delta_{|s|+1,r} + \Delta_{|s|,f} + (-\delta_{|s|,r})$

$=$ { calc. }

$\Delta_{|s|+1,r} + \Delta_{|s|,f} + (\Delta_{|s|,r} - \delta_{|s|,r})$

$=$ { $k = 1$, calc. }

$\Delta_{|s|+k,r} + \Delta_{|s|+k-1,f} + \sum_{h=0}^{k-1}(\Delta_{|s|+h,r} - \delta_{|s|+h,r})$

Step. Assume the theorem holds for $k > 0$ and that (a_s, j_s) has response depth $k+1$. Then there is a node (a_{sb}, j_{sb}) such that dependency $(a_{sb}, j_{sb}) \rightarrow (a_s, j_s)$ exists and (a_{sb}, j_{sb}) has response depth k. From the induction hypothesis it then follows that node $(a_{sb}, j_{sb} - n^{k-1})$ exists. Given that node $(a_{sb}, j_{sb} - n^{k-1})$ and dependency $(a_{sb}, j_{sb}) \rightarrow (a_s, j_s)$ exist, we show that node $(a_s, j_s - n^k)$ and dependency $(a_{sb}, j_{sb} - n^{k-1}) \rightarrow (a_s, j_s - n^k)$ also exist and that we can depict the dependencies for levels $|s|$ and $|s| + 1$ only as in Figure 5.6.

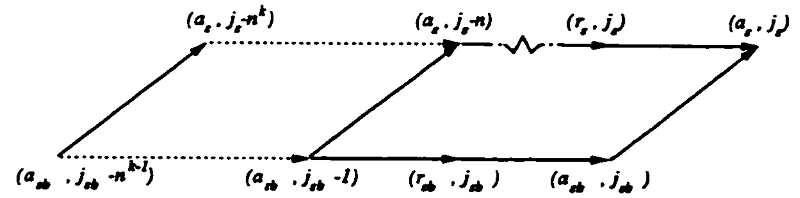


Figure 5.6: A part of the behaviour graph at levels $|s|$ and $|s| + 1$.

Let us look at the behaviour of a cell at level $|s|$.

$(r_s?;\ *[\ (a_s!;\ r_s?) \| (r_{sx_1}!;\ a_{sx_1}?);\ (a_s!;\ r_s?) \| (r_{sx_2}!;\ a_{sx_2}?);$

$$\cdots; (a_s!;\ r_s?) \| (r_{sx_n}!;\ a_{sx_n}?)\ ])$$

where x_k are the distinct elements of Σ.

Let $b = x_k \in \Sigma$. From this behaviour it follows that between any two successive dependencies $a_{sb} \rightarrow a_s$ for a particular $b \in \Sigma$ there are n dependencies $a_s \rightsquigarrow r_s$. It follows that dependency $(a_{sb}, j_{sb} - 1) \rightarrow (a_s, j_s - n)$ exists. For example, there are $n = 2$ edges of the form $a \rightarrow r$ at level 0 in Figure 5.4, between each two $a_l \rightarrow a$ edges. Note that the dependency $a_s \rightsquigarrow r_s$ is a sequence of edges if $s \neq \epsilon$. Note also that $a_{sb} \rightarrow a_s$ is a direct dependency, i.e., a single edge, as is $(a_{sb}, j_{sb} - 1) \rightarrow (a_s, j_s - n)$. Both of these facts are easily seen from the graph in Figure 5.4.

If $(a_{sb}, j_{sb} - 1) \rightarrow (a_s, j_s - n)$ exists, we can iteratively show that $(a_{sb}, j_{sb} - 2) \rightarrow (a_s, j_s - 2n)$ exists. Repeating this step n^{k-1} times yields the dependency graph above with direct dependency $(a_{sb}, j_{sb} - n^{k-1}) \rightarrow (a_s, j_s - n^k)$.

Note that the above proof also gives us the existence of $(a_s, j_s - n^k)$.

Now we observe

$T(a_s, j_s) - T(a_s, j_s - n^k)$

$=$ { Break path into three: edge, path, edge. }

$((T(a_s, j_s)) - T(a_{sb}, j_{sb})) + (T(a_{sb}, j_{sb}) - T(a_{sb}, j_{sb} - n^{k-1})) + (T(a_{sb}, j_{sb} - n^{k-1}) - T(a_s, j_s - n^k))$

$\leq$ { (a_{sb}, j_{sb}) has resp. depth k, induction hyp. }

$((T(a_s, j_s)) - T(a_{sb}, j_{sb})) + \Delta_{|sb|+k,r} + \Delta_{|sb|+k-1,f} + \sum_{h=0}^{k-1}(\Delta_{|sb|+h,r} - \delta_{|sb|+h,r}) + (T(a_{sb}, j_{sb} - n^{k-1}) - T(a_s, j_s - n^k))$

$\leq$ { calc. }

$((T(a_s, j_s)) - T(a_{sb}, j_{sb})) + \Delta_{|s|+1+k,r} + \Delta_{|s|+1+k-1,f} + \sum_{h=0}^{k-1}(\Delta_{|s|+1+h,r} - \delta_{|s|+1+h,r}) + (T(a_{sb}, j_{sb} - n^{k-1}) - T(a_s, j_s - n^k))$

$\leq$ { (a_s, j_s) has positive response depth, and ack/ack pair is a reverse latency }

$\Delta_{|s|,r} + \Delta_{|s|+k+1,r} + \Delta_{|s|+k,f} + \sum_{h=1}^{k}(\Delta_{|s|+h,r} - \delta_{|s|+h,r}) + (T(a_{sb}, j_{sb} - n^{k-1}) - T(a_s, j_s - n^k))$

$\leq$ { $(a_{sb}, j_{sb} - n^{k-1}) \to (a_s, j_s - n^k)$ exists, minimum edge delay is $\delta_{|s|,r}$ }

$\Delta_{|s|,r} + \Delta_{|s|+k+1,r} + \Delta_{|s|+k,f} + \sum_{h=1}^{k}(\Delta_{|s|+h,r} - \delta_{|s|+h,r}) + (-\delta_{|s|,r})$

$\leq$ { calc. }

$\Delta_{|s|+k+1,r} + \Delta_{|s|+k,f} + \sum_{h=0}^{k}(\Delta_{|s|+h,r} - \delta_{|s|+h,r})$

□

5.3 Worst-Case Response Time

We now calculate the worst-case response time so we have bounds on the worst-case behaviour of the tree. The result for *RT* follows identically to the original proof for pipelines, given the mappings $a_\epsilon \leftrightarrow a_0$, and $r_\epsilon \leftrightarrow r_0$. We have the following theorem.

Theorem 5.2 *The worst-case response time RT for any tree (as defined in Section 5.2 operating under a delay model of Section 2.6) is bounded from above as*

$$\begin{aligned} RT \;\leq\; & \max(\{\Delta_{k,r} + \Delta_{k-1,f} + \sum_{h=0}^{k-1}(\Delta_{h,r} - \delta_{h,r}) \\ & -(\delta_{0,r} + \delta_e)n^{k-1} + \delta_{0,r} \mid 1 \leq k \leq L\} \cup \{\Delta_{0,r}\}) \end{aligned}$$

Proof. We have to find an upper bound for $T(a_\epsilon, j) - T(r_\epsilon, j)$ over all $j \geq 0$ and all valid delay distributions. Each acknowledgment has a response depth k, where $0 \leq k \leq L$. For a response depth $k = 0$, we have from Lemma 3.2, $T(a_\epsilon, j) - T(r_\epsilon, j) \leq \Delta_{0,r}$. For a response depth of $k > 0$, we use the result of the previous section. We observe

$$T(a_\epsilon, j) - T(r_\epsilon, j)$$

$= \quad \{ \text{ calc. } \}$

$$(T(a_\epsilon, j) - T(a_\epsilon, j - n^{k-1})) + (T(a_\epsilon, j - n^{k-1}) - (T(r_\epsilon, j))$$

$\leq \quad \{ \ (a_\epsilon, j) \text{ has response depth } k\text{, Lemma 5.1, } s = \epsilon \ \}$

$$\Delta_{|s|+k,r} + \Delta_{|s|+k-1,f} + \textstyle\sum_{h=0}^{k-1}(\Delta_{|s|+h,r} - \delta_{|s|+h,r}) + (T(a_\epsilon, j - n^{k-1}) - (T(r_\epsilon, j))$$

$\leq \quad \{ \ |s| = |\epsilon| = 0 \ \}$

$$\Delta_{k,r} + \Delta_{k-1,f} + \textstyle\sum_{h=0}^{k-1}(\Delta_{h,r} - \delta_{h,r}) + (T(a_\epsilon, j - n^{k-1}) - (T(r_\epsilon, j))$$

Notice that there are n^{k-1} handshakes at level 0 between occurrences (a_ϵ, j) and $(a_\epsilon, j - n^{k-1})$. The handshakes at level 0 experience delays through the environment and through cell 0. Assuming that the environment's minimum delay is δ_e (for edge $a_\epsilon \rightarrow r_\epsilon$) and the minimum delay of cell 0 (for a reverse latency edge $r_\epsilon \rightarrow a_\epsilon$) is $\delta_{0,r}$, the minimum duration of one cycle through environment and cell is $\delta_{0,r} + \delta_e$. Hence the minimum duration from $T(a_\epsilon, j - n^{k-1})$ to $T(r_\epsilon, j)$ is $(\delta_{0,r} + \delta_e)n^{k-1} - \delta_{0,r}$. This then leads to the inequality

$$T(a_\epsilon, j) - T(r_\epsilon, j) \leq \Delta_{k,r} + \Delta_{k-1,f} + \sum_{h=0}^{k-1}(\Delta_{h,r} - \delta_{h,r}) - (\delta_{0,r} + \delta_e)n^{k-1} + \delta_{0,r}$$

Maximising these upper bounds over all $k, 0 \leq k \leq L$ gives the desired result.
□

5.3.1 Simplifying the Bound

We simplify the bound in the same fashion as we did in Section 3.3.1. We ignore the case where $n = 1$, as this gives us rather uninteresting trees. When $n > 1$, we use the following simplifying assumptions. We restrict the possible delay ranges of the maximum forward and reverse latencies of each cell. We assume that

$$\begin{aligned}
\Delta_{h,r} &\leq \Delta_{0,r} \cdot n^{h-1} \\
\Delta_{h,f} &\leq \delta_e \cdot n^h
\end{aligned}$$

where $h \geq 1$. To simplify the formula further, we again assume that forward and reverse latencies are equal, i.e., $\Delta_{h,r} = \delta_{h,r}$ where $h \geq 0$. That is, the reverse latencies are constant at each depth, though the reverse latency of a cell at level h may still have a different reverse latency from a cell at level g, where $g \neq h$.

$$\begin{aligned}
RT &\leq \max(\{\Delta_{k,r} + \Delta_{k-1,f} + \sum_{h=0}^{k-1}(0) - (\delta_{0,r} + \delta_e)n^{k-1} + \delta_{0,r} \mid 1 \leq k \leq L\} \\
&\quad \cup\{\Delta_{0,r}\}) \\
&\qquad \{\text{Using } \Delta_{h,r} \leq \Delta_{0,r} \cdot n^{h-1}\} \\
&\leq \max(\{\Delta_{0,r} \cdot n^{k-1} + \Delta_{k-1,f} - (\delta_{0,r} + \delta_e)n^{k-1} + \delta_{0,r} \mid 1 \leq k \leq L\} \\
&\quad \cup\{\Delta_{0,r}\}) \\
&\qquad \{\text{Using } \delta_{0,r} = \Delta_{0,r}\} \\
&= \max(\{\Delta_{0,r} \cdot n^{k-1} + \Delta_{k-1,f} - (\Delta_{0,r} + \delta_e)n^{k-1} + \Delta_{0,r} \mid 1 \leq k \leq L\} \\
&\quad \cup\{\Delta_{0,r}\}) \\
&= \max(\{\Delta_{k-1,f} - \delta_e \cdot n^{k-1} + \Delta_{0,r} \mid 1 \leq k \leq L\} \cup \{\Delta_{0,r}\}) \\
&\qquad \{\text{Using } \Delta_{h,f} \leq \delta_e \cdot n^h\ \} \\
&\leq \max(\{0 + \Delta_{0,r} \mid 1 \leq k \leq L\} \cup \{\Delta_{0,r}\}) \\
&= \Delta_{0,r}
\end{aligned}$$

The simplifying assumptions about cell delays for pipelines given in Section 3.3.1 were

$$\begin{aligned}\Delta_{h,r} &\leq \Delta_{0,r} \cdot n^h \\ \Delta_{h-1,f} &\leq \delta_e \cdot n^h\end{aligned}$$

where $h \geq 1$. There is a factor of n difference between these assumptions and those given for trees. The assumptions about cell delays for trees are more restrictive than those given for pipelines, even though delays of cells deep in the tree may be exponentially slow. Note that if we set $h = 1$ we restrict the delays of the first two levels of a tree as follows $\Delta_{1,r} \leq \Delta_{0,r}$. Another is that $\Delta_{0,f} \leq \delta_e$ which might seem a little stringent. We can use the following assumptions instead.

$$\begin{aligned}\Delta_{h,r} &\leq \Delta_{0,r} \cdot n^{h-1} \\ \Delta_{h,f} &\leq \delta_e \cdot n^h\end{aligned}$$

where $h \geq 2$. We leave the delays unrestricted when $h = 1$ and this is the only change from the previous assumptions for trees. We again assume that reverse latencies are constant, i.e., $\Delta_{h,r} = \delta_{h,r}$ with $h \geq 0$.

$$\begin{aligned}RT &\leq \max(\{\Delta_{k,r} + \Delta_{k-1,f} + \sum_{h=0}^{k-1}(0) - (\delta_{0,r} + \delta_e)n^{k-1} + \delta_{0,r} \mid 1 \leq k \leq L\} \\ &\quad \cup\{\Delta_{0,r}\}) \\ &\quad\quad \{\text{Using steps as above, we are left with one last case when } k = 1\} \\ &\leq \max(\{\Delta_{k,r} + \Delta_{k-1,f} + \sum_{h=0}^{k-1}(0) - (\delta_{0,r} + \delta_e)n^{k-1} + \delta_{0,r} \mid k = 1\} \\ &\quad \cup\{\Delta_{0,r}\}) \\ &\leq \max(\{\Delta_{1,r} + \Delta_{0,f} - (\delta_{0,r} + \delta_e) + \delta_{0,r}\} \cup \{\Delta_{0,r}\}) \\ &\leq \max(\{\Delta_{1,r} + \Delta_{0,f} - \delta_e\} \cup \{\Delta_{0,r}\})\end{aligned}$$

This is still a simple result. With these assumptions one can say that if the reverse latencies have no variance (i.e., $\Delta_{h,r} = \delta_{h,r}$) the delays in the first two levels are the only ones that matter. The rest of the pipeline can be exponentially slow in the depth of the cells in the pipeline and may be optimised for power consumption or area rather than for speed.

5.4 Average Response Time

We would like to find the throughput of a tree-based circuit. Again we calculate the bounds on the average response time AR for trees, but, as with the proof for response time, we have to worry about possible interference between the signals travelling on different branches of the tree.

In the following proofs we use $s \in \Sigma^*$, and we let $h = |s|$. If we specify an event a_h, we are referring to any acknowledgment event at the hth interface. If we refer to a_s, however, we are referring to a particular acknowledgment. For example, if $s_1 = ll$, $s_2 = lr$, $s_3 = rl$, and $s_4 = rr$, and $|s_1| = |s_2| = |s_3| = |s_4| = h = 2$, then we have a_{ll}, a_{lr}, a_{rl}, and a_{rr} are four a_2 events. We define things similarly for requests.

Theorem 5.3 *Let the maximum delays for each cell be bounded as follows.*

$$\begin{aligned}\Delta_{i,f} &\leq \Delta_{0,f} \cdot n^i \\ \Delta_{i,r} &\leq \Delta_{0,r} \cdot n^i\end{aligned}$$

where $i \geq 0$ is the depth of the cell in the tree.

The average response time AR for any tree (as defined in Section 5.2 operating under a delay model of Section 2.6) and with $n > 1$, is bounded as follows

$$\delta_{0,r} \leq AR \leq \max\{\Delta_{0,r}, \Delta_{1,r} + \Delta_{0,f} - \delta_e\}$$

The lower bound even applies when the cell delays $\Delta_{i,f}$ and $\Delta_{i,r}$ are not bounded. The upper bound is a critical bound when $\Delta_{i,f} = \Delta_{0,f} \cdot n^i$ and $\Delta_{i,r} = \Delta_{0,r} \cdot n^i$.

□

Before we prove the main theorem, we need an auxiliary lemma. We again use the definition of level h paths and paths $\in (h, h+k]$ as defined in Section 2.6.4.

Lemma 5.4 *Let $\Delta_{i,f} = \Delta_{0,f} \cdot n^i$ and $\Delta_{i,r} = \Delta_{0,r} \cdot n^i$ be given. Let paths P and P' exist between (r_s, i) and (a_s, j), for $|s| = h$, $h > 0$, and $j > i$. Let path $P \in (h, h+k]$ have maximal delays on each edge, and let P' be a level h path, also with maximal delays on each edge. Then the delay on P is the same as the delay on P'.*

□

Note, as with the $n = 2$ average-case analysis we considered in Chapter 4, we explicitly exclude $h = 0$ from our reckoning, because the environment delays are unconstrained. Note also that the statement of the problem, i.e., path P exists and $P \in (h, h+k]$, means that the first edge in P is an $r_h \rightarrow r_{h+1}$ edge and the last edge in P is an $a_{h+1} \rightarrow a_h$ edge. This constraint also implies that $j = i + m$ with $m > 0$.

As with the similar proof in Chapter 4 we do a nested induction. The outer induction is on the maximum depth of the path in the pipeline, k. The inner induction is on m, the difference in occurrence indices of the first and last event in the path.

Proof of Lemma 5.4. We induct on the number of levels, k. Note that $h = |s|$.

Basis. Let $k = 1$. Let $j = i + m$. We induct on m, the number of occurrence indices between (r_h, i) and (a_h, j).

Basis. Let $m = 1$. Then $j = i + 1$. Refer to Figure 5.7.

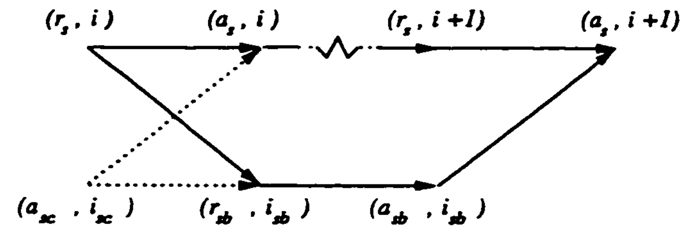


Figure 5.7: Example of path when $k = 1$ and $m = 1$.

We can trace a path directly at level h between $T(r_s, i)$ and $T(a_s, i+1)$ where $|s| = h$. We have $n+1$ edges of type $r_h \rightarrow a_h$ on this path and n edges of type $a_h \rightarrow r_h$, recalling that a_h and r_h are any events at level h, not just events a_s and r_s. To convince ourselves, let us look at the behaviour of a cell at level $|u|$ $(= h-1)$.

$$(r_u?;\ *[\ (a_u!;\ r_u?)||(r_{ux_1}!;\ a_{ux_1}?);\ (a_u!;\ r_u?)||(r_{ux_2}!;\ a_{ux_2}?);$$
$$\cdots;(a_u!;\ r_u?)||(r_{ux_n}!;\ a_{ux_n}?)\ ])$$

where $x_q \in \{x_0, x_1, \cdots, x_n\}$ are the distinct elements of Σ. Let $u \in \Sigma^*$ such that $ux_1 = s$. Between (r_{ux_1}, i) and $(r_{ux_1}, i+1)$ there are n edges of the form $r_{ux_q} \rightarrow a_{ux_q}$ and n of the form $a_{ux_q} \rightarrow r_{ux_{(q \cdot mod\ n)+1}}$. Lastly, from $(r_{ux_1}, i+1) \rightarrow (a_{ux_1}, i+1)$ we have one more $r_h \rightarrow a_h$ edge.

As an example, consider Figure 5.4. Let $s =$"ll". Between $(r_{ll}, 0)$ and $(a_{ll}, 1)$ we have the edges $(r_{ll}, 0) \rightarrow (a_{ll}, 0)$, $(a_{ll}, 0) \rightarrow (r_{lr}, 0)$, $(r_{lr}, 0) \rightarrow (a_{lr}, 0)$, $(a_{lr}, 0) \rightarrow (r_{ll}, 1)$, and $(r_{ll}, 1) \rightarrow (a_{ll}, 1)$. We have $n+1$ $r_h \rightarrow a_h$ edges on this path and n $a_h \rightarrow r_h$ edges as predicted.

The delay on each edge $r_h \rightarrow a_h$ is maximised at $\Delta_{h,r}$. Likewise the delay on each edge of type $a_h \to r_h$ is at most $\Delta_{h-1,f}$. Then,

$$\begin{aligned} T(a_s, i+1) - T(r_s, i) &= (n+1)\Delta_{h,r} + n \cdot \Delta_{h-1,f} \\ &= (n+1)n^h\Delta_{0,r} + n^h \cdot \Delta_{0,f} \end{aligned}$$

This is the delay on the level h path P'.

Now we derive the delay of the path $P \in (h, h+1]$ between $T(r_s, i)$ and $T(a_s, i+1)$ with $j = i+1$. There is only one such path and it is $r_s \to r_{sb} \to a_{sb} \to a_s$. We can calculate the corresponding maximum delay.

$$\begin{aligned} T(a_s, i+1) - T(r_s, i) &= \Delta_{h,f} + \Delta_{h+1,r} + \Delta_{h,r} \\ &\qquad \{\text{From statement of theorem}\} \\ &= n^h \cdot \Delta_{0,f} + n^{h+1} \cdot \Delta_{0,r} + n^h \cdot \Delta_{0,r} \\ &= (n+1)n^h\Delta_{0,r} + n^h \cdot \Delta_{0,f} \end{aligned}$$

So the delay on P is the same as the delay on P'.

Step We have $k = 1$ as above. Assume that our property holds for $j = i + m$. Now we prove for the case when $j = i + m + 1$. Refer to Figure 5.8. Let $T(a_s, i+m) - T(r_s, i) = x$.

We derive the delay $T(a_s, i+m+1) - T(r_s, i)$. First we assume that the level h ($= |s|$) path P' is critical. Using the same reasoning as for the base case we know that there are n edges of type $r_h \to a_h$ and n edges of type $a_h \to r_h$ edges between $(a_s, i+m)$ and $(a_s, i+m+1)$. Consequently,

$$\begin{aligned} \text{Delay}(P') &= (T(a_s, i+m+1) - T(a_s, i+m)) \\ &\quad + (T(a_s, i+m) - T(r_s, i)) \end{aligned}$$

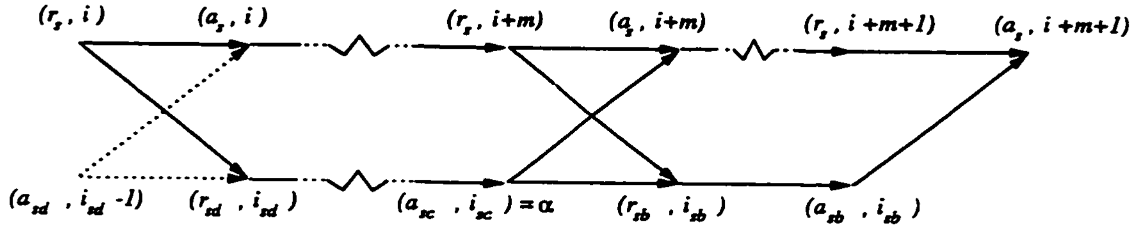


Figure 5.8: Example of path when $k = 1$ and $m \geq 1$.

$$\begin{aligned}
&= (T(a_s, i+m+1) - T(a_s, i+m)) + x \\
&= n \cdot \Delta_{h,r} + n \cdot \Delta_{h-1,f} + x \\
&= n^{h+1} \cdot \Delta_{0,r} + n^h \cdot \Delta_{0,f} + x \quad \{\text{By assumptions}\}
\end{aligned}$$

Now assume the path $P \in (h, h+1]$ between (r_s, i) and $(a_s, i+m+1)$ is critical. Path P is restricted to events at level $h+1$ except for the endpoints and so must pass through point (a_{sc}, i_{sc}). See Figure 5.8. For brevity, we call this event α. We determine $T(\alpha)$.

First note that both the level h path and the path in $(h, h+1]$ between $T(r_s, i)$ and $T(a_s, i+m)$ have delay x by our inductive assumption. The delays on each edge of each path are maximised. Now, observe that the two incoming edges to $(a_s, i+m)$ must have maximal delay of $\Delta_{h,r}$. Thus both predecessors of $(a_s, i+m)$, i.e. $(r_s, i+m)$ and $(a_{sc}, i_{sc}) = \alpha$, must occur at $T(a_s, i+m) - \Delta_{h,r}$.

$$\begin{aligned}
T(\alpha) &= T(a_{sc}, i_{sc}) \\
&= T(a_s, i+m) - \Delta_{h,r} \\
&= T(r_s, i) + x - \Delta_{h,r}.
\end{aligned}$$

We know path P is a path $(r_s, i) \rightsquigarrow \alpha \rightsquigarrow (a_s, i+m+1)$. The path $\alpha \rightsquigarrow (a_s, i+m+1)$ consists of one $a_{h+1} \rightarrow r_{h+1}$ edge, i.e., $(a_{sc}, i_{sc}) \rightarrow (r_{sb}, i_{sb})$, one $r_{h+1} \rightarrow a_{h+1}$ edge, i.e., $(r_{sb}, i_{sb}) \to (a_{sb}, i_{sb})$, and one $a_{h+1} \to a_h$ edge, i.e., $(a_{sb}, i_{sb}) \to (a_s, i+m+1)$. These have delays respectively of $\Delta_{h,f}$, $\Delta_{h+1,r}$ and $\Delta_{h,r}$. Consequently,

$$\begin{aligned}
& T(a_s, i+m+1) - T(r_s, i) \\
= \; & (T(a_s, i+m+1) - T(\alpha)) + (T(\alpha) - T(r_s, i)) \\
= \; & (\Delta_{h,f} + \Delta_{h+1,r} + \Delta_{h,r}) + (T(\alpha) - T(r_s, i)) \\
= \; & (\Delta_{h,f} + \Delta_{h+1,r} + \Delta_{h,r}) + (T(r_s, i) + x - \Delta_{h,r} - T(r_s, i)) \\
= \; & (\Delta_{h,f} + \Delta_{h+1,r} + \Delta_{h,r}) + (x - \Delta_{h,r}) \\
= \; & \Delta_{h,f} + \Delta_{h+1,r} + x \\
= \; & n^{h+1} \cdot \Delta_{0,r} + n^h \cdot \Delta_{0,f} + x
\end{aligned}$$

This is the same as the delay on path P', and we are done. We have now completed the base case for when $k = 1$.

Step. Assume that the property holds for k. We prove that the property holds for $k+1$. We wish to show that if we have a path $P \in (h, h+k+1]$, between (r_s, i) and (a_s, j), and we have a level h path P' between the two, then the maximum delay on each path is the same.

Note that path P must begin with an $r_s \to r_{sc}$ edge and finish with an $a_{sb} \to a_s$ edge. So we have a path $r_s \to r_{sc} \rightsquigarrow a_{sb} \to a_s$. The path $r_{sc} \rightsquigarrow a_{sb}$ is a path in $[h+1, h+k+1]$, or by setting $g = h+1$ $(= |sb| = |sc|)$, path $r_{sc} \rightsquigarrow a_{sb}$ is a path in $[g, g+k]$.

The path $r_{sc} \rightsquigarrow a_{sb}$ is either a path at level g, or consists of segments S, some of which are paths at level g, and some of which are paths in $(g, g+k]$.

Let $t \in \Sigma^*$ be a string of length $|s|+1$. Let S be any segment on the path $r_{sc} \rightsquigarrow a_{sb}$ of the form $(r_t, i_g) \rightsquigarrow (a_t, j_g)$. Either this is a path at level g or it is a

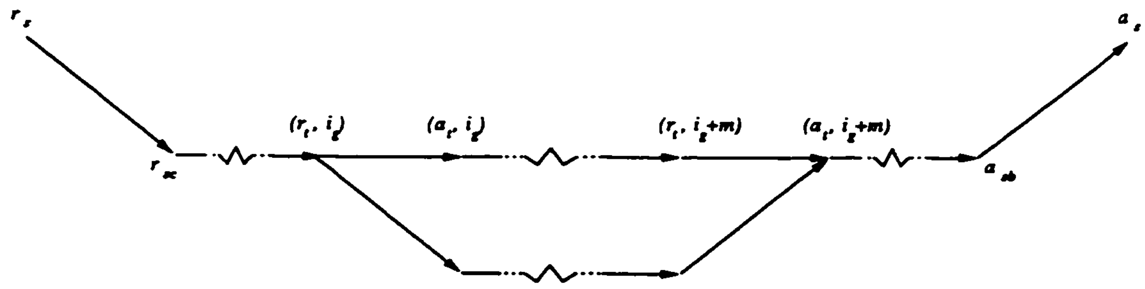


Figure 5.9: Example of path when $k = 1$ and $m \geq 1$.

path in $(g, g+k]$ with $j_g = i_g + m$, $m > 0$. Refer to Figure 5.9. By the induction hypothesis, the maximal delay on segment S is the same whether it is a segment at level g, or if $S \in (g, g+k]$. Without loss of generality we assume that segment S is a level g path.

So, we may assume that path $r_{sc} \rightsquigarrow a_{sb}$ is a level g path, and therefore path P in $(h, h+1]$. We now have the base case, which we have already proven and we are done.

□

We are ready to prove Theorem 5.3.

Proof of Theorem 5.3. Because all $r_0 \rightarrow a_0$ edges have a minimum delay of $\delta_{0,r}$, we know that $AR \geq \delta_{0,r}$ and we have proven the lower bound.

Now we prove the upper bound.

$$AR \leq \max\{\Delta_{0,r}, \Delta_{1,r} + \Delta_{0,f} - \delta_e\}$$

Let there be a critical path P, from event (e, i_0) at level 0 to event (f, j_0) at level 0, with $j_0 \gg i_0$. If e and f are sufficiently far apart then, without loss of generality, we let event e be a request and event f be an acknowledgement. We can do this, because edge delays are assumed to be finite, and on an infinite path a single edge causes negligible extra delay. We choose i_0 and j_0 such that $j_0 = i_0 + m$, for some $m > 0$. Because we are looking for a worst-case bound, we assign delays to cells that make them as slow as they can be. That is, $\Delta_{i,f} = \Delta_{0,f} \cdot n^i$ and $\Delta_{i,r} = \Delta_{0,r} \cdot n^i$.

We may assume path $P \in [0, 1]$ due to Lemma 5.4 above. Remember that Lemma 5.4 does not apply to level 0 paths as the environment delays are unbounded.

Suppose that the critical path P is a level 0 path. Then

$$T(a_\epsilon, j) - T(r_\epsilon, i) \leq (j - i + 1) \cdot \Delta_{0,r} + (j - i) \cdot (\text{Environment Delay})$$

To determine the average response time between (r_ϵ, i) and (a_ϵ, j) we subtract the environment delay and divide by the number of cycles. The number of cycles is $j - i + 1$, so, if the critical path P is a level 0 path, the average response time satisfies $AR \leq \Delta_{0,r}$.

If the critical path P is not a level 0 path then we can break P into segments. There is at least one segment S of the form $S \in (0, 1]$. Let S start at (r_ϵ, i_ϵ) and end at (a_ϵ, j_ϵ). Then $j_\epsilon = i_\epsilon + m'$ with $m' > 0$. We can calculate the maximum delay on S. S consists of the first request/request edge, $2m' - 1$ "horizontal edges", and one final acknowledgment/acknowledgment edge.

$$\begin{aligned} T(a_\epsilon, i_\epsilon + m') - T(r_\epsilon, i_\epsilon) &= \Delta_{0,f} + m' \cdot \Delta_{1,r} + (m' - 1) \cdot \Delta_{0,f} + \Delta_{0,r} \\ &= m' \cdot (\Delta_{1,r} + \Delta_{0,f}) + \Delta_{0,r} \end{aligned}$$

We derive the average response time over this segment by subtracting the number of environment delays, and dividing by the number of cycles at level 0.

$$AR_{ub} = (m' \cdot (\Delta_{1,r} + \Delta_{0,f}) + \Delta_{0,r} - m' \cdot \delta_e)/(m' + 1)$$

First let us consider the case where $\delta_e \geq \Delta_{1,r} + \Delta_{0,f} - \Delta_{0,r}$ We maximise over $m' > 0$.

$$\begin{aligned}
AR_{ub} &= (m' \cdot (\Delta_{1,r} + \Delta_{0,f}) + \Delta_{0,r} - m' \cdot \delta_e)/(m'+1) \\
&\quad \{\text{From assumption above}\} \\
&\leq (m' \cdot (\Delta_{1,r} + \Delta_{0,f}) + \Delta_{0,r} - m' \cdot (\Delta_{1,r} + \Delta_{0,f} - \Delta_{0,r}))/(m'+1) \\
&= (\Delta_{0,r} + m' \cdot \Delta_{0,r})/(m'+1) \\
&= \Delta_{0,r}
\end{aligned}$$

We have shown that if $\delta_e \geq \Delta_{1,r} + \Delta_{0,f} - \Delta_{0,r}$ then $AR \leq \Delta_{0,r}$. But $\Delta_{0,r}$ is the delay through cell 0 alone and is the value of AR when using a level 0 path. We can therefore surmise that if $\delta_e > n \cdot \Delta_{0,f}$, the critical path is at level 0 and goes no deeper in the tree.

Let us now consider the case when $\delta_e < \Delta_{1,r} + \Delta_{0,f} - \Delta_{0,r}$ We maximise over $m' > 0$.

$$\begin{aligned}
AR_{ub} &= (m' \cdot (\Delta_{1,r} + \Delta_{0,f}) + \Delta_{0,r} - m' \cdot \delta_e)/(m'+1) \\
&= (\delta_e + \Delta_{0,r} - \Delta_{1,r} - \Delta_{0,f})/(m'+1) + \Delta_{1,r} + \Delta_{0,f} - \delta_e \\
&\quad \{\text{From assumption above}\} \\
&< (\Delta_{1,r} + \Delta_{0,f} - \Delta_{0,r} + \Delta_{0,r} - \Delta_{1,r} - \Delta_{0,f})/(m'+1) \\
&\quad + \Delta_{1,r} + \Delta_{0,f} - \delta_e \\
&= (0)/(m'+1) + \Delta_{1,r} + \Delta_{0,f} - \delta_e \\
&= \Delta_{1,r} + \Delta_{0,f} - \delta_e
\end{aligned}$$

The contribution of delay by level 0 paths is negligible here.

Combining the above results, and maximising over all possible segments, we get for the average response time

$$AR \leq \max\{\Delta_{0,r}, \Delta_{1,r} + \Delta_{0,f} - \delta_e\}$$

□

As we saw with linear pipelines with $n > 1$, the delay of the first cell is the most important. Even if we let the cells deeper in the tree become exponentially slow, the bound on the average cycle time still depends on the first cell.

So, what do these results mean for optimising tree structures? Presumably we wish to size transistors to optimise the speed of cell 0. Smaller transistors are usually slower, but obviously take up less chip area. As our results show, we can minimise the size of transistors deeper in the tree without affecting the bounds on the average throughput. Consequently we can shrink the transistors and reduce the area penalty of using a tree without compromising the worst-case bounds. We also have more freedom to use longer wires, with their accompanying greater delay, to suit our layout requirements. We still have a problem with the delay of directing data to different branches, but this will only be a problem at cell 0.

5.5 Cycle-Time of Tree Architectures

Bounds on the worst-case cycle time and average-case cycle time are easy to compute. We have the following theorems.

Theorem 5.5 *The worst-case cycle time CT for any tree (as defined in Section 5.2 operating under a delay model of Section 2.6) is bounded from above as*

$$CT \leq \Delta_e + \max(\{\Delta_{k,r} + \Delta_{k-1,f} + \sum_{h=0}^{k-1}(\Delta_{h,r} - \delta_{h,r})$$

$$-(\delta_{0,r} + \delta_e)n^{k-1} + \delta_{0,r} \mid 1 \le k \le L\} \cup \{\Delta_{0,r}\})$$

Proof. By using Lemma 3.5 we know that $CT \le RT_{ub} + \Delta_e$ where RT_{ub} is an upper bound on response time. We use the value of RT_{ub} calculated by Theorem 5.2 and the result follows.

□

Theorem 5.6 *Let the maximum delays for each cell be bounded as follows.*

$$\begin{aligned}\Delta_{i,f} &\le \Delta_{0,f} \cdot n^i \\ \Delta_{i,r} &\le \Delta_{0,r} \cdot n^i\end{aligned}$$

where $i \ge 0$ is the depth of the cell in the tree.

The average cycle time AC for any tree (as defined in Section 5.2 operating under a delay model of Section 2.6) and with $n > 1$ is bounded as follows

$$\delta_{0,r} + \delta_e \le AC \le \max\{\Delta_{0,r} + \Delta_e, n \cdot \Delta_{0,r} + n \cdot \Delta_{0,f}\}$$

The lower bound even applies when the cell delays $\Delta_{i,f}$ and $\Delta_{i,r}$ are not bounded. The upper bound is a critical bound when $\Delta_{i,f} = \Delta_{0,f} \cdot n^i$ and $\Delta_{i,r} = \Delta_{0,r} \cdot n^i$.

Proof. Because all $r_0 \to a_0$ edges have a minimum delay of $\delta_{0,r}$, and all $a_0 \to r_0$ edges have a minimum delay of δ_e, we know that $AC \ge \delta_{0,r} + \delta_e$ and we have proven the lower bound.

To prove the upper bound, we use Corollary 4.11 and Lemma 4.8 from Section 4.6. Lemma 4.8 defines AR_{ub} as

$$AR_{ub} = \max\{\overline{AR}_{ub}, \widehat{AR}_{ub}\}$$

where $\overline{AR}_{ub}$ is derived from a level 0 path, and $\overline{AR}_{ub}$ is derived from a path in which the contribution of level 0 paths is negligible. The proof of the upper bound for AR in Theorem 5.3 splits the path calculation into two. One path is the level 0, and one path is a path in $(0,1]$. One of these paths is the critical path. These paths respectively derive $\overline{AR}_{ub} = \Delta_{0,r}$ and $\widehat{AR}_{ub} = \Delta_{1,r} + \Delta_{0,f} - \delta_e$. Theorem 5.3 gives

$$\begin{aligned} AR_{ub} &= \max\{\Delta_{0,r}, \Delta_{1,r} + \Delta_{0,f} - \delta_e\} \\ &= \max\{\overline{AR}_{ub}, \widehat{AR}_{ub}\} \end{aligned}$$

Because the critical path is either (i) a path at level 0, or (ii) a path with negligible delay at level 0, we can apply Corollary 4.11 which states

$$\begin{aligned} AC_{ub} &= \max\{\overline{AR}_{ub} + \Delta_e, \widehat{AR}_{ub} + \delta_e\} \\ &= \max\{\Delta_{0,r} + \Delta_e, \Delta_{1,r} + \Delta_{0,f}\} \end{aligned}$$

□

We will apply the above formulae to a tree-based FIFO in the next chapter.

5.6 Generalised Behaviours

We can generalise the behaviour given in Section 5.2 in a number of ways and therefore widen the scope of the results. For example, consider the following implementation of a stack with a binary tree. On a push operation data is inserted at the root, and the data at the root is displaced down into the tree. The root decides whether to send the displaced data to the left child or to the right child and its choice might depend on the number of items in the stack, or where it sent previously displaced data. On the next pop operation it may retrieve data from the same child. That is, we may have consecutive communications with the left-hand child.

The behaviour given in Section 5.2 is

$$(r_s?;\ *[\,(a_s!;\ r_s?)||(r_{sx_1}!;\ a_{sx_1}?);\ (a_s!;\ r_s?)||(r_{sx_2}!;\ a_{sx_2}?);$$
$$\cdots;(a_s!;\ r_s?)||(r_{sx_n}!;\ a_{sx_n}?)\,])$$

where $x_q \in \{x_0, x_1, \cdots, x_n\}$ are the distinct elements of Σ. This behaviour requires that a node communicates with each child in turn. We generalise the behaviour as

$$(r_s?;\ *[\,(a_s!;\ r_s?)||(r_{sx}!;\ a_{sx}?)\,])$$

where x is a random element of Σ at each repetition step. That is, the behaviour specifies in each repetition step a handshake at interface $|s|$, i.e., $(a_s!;\ r_s?)$, with the parent, and a handshake on interface $|s|+1$, i.e., $(r_{sx}!;\ a_{sx}?)$, made with an arbitrary child.

This behaviour is extremely flexible, but note that it is a form of conditional behaviour. We assume that the probability of choosing any particular child is greater than zero. If we look for worst-case behaviour we always choose the worst-case child. If we look for best-case behaviour, we always choose the best-case child. For our purposes, therefore, the choice is deterministic.

Let us extend the behaviour even further.

$$(r_s?;*[\textbf{if } B_s \textbf{ then } (a_s!;\ r_s?)\,||(r_{sx}!;\ a_{sx}?) \textbf{ else } (a_s!;\ r_s?)\,\textbf{fi}\,])$$

where x is a random element of Σ at each repetition step. Not only can a cell choose any to communicate with, but a cell can also choose not to communicate deeper into the tree. This is analogous to the conditional behaviour for pipelines described in Section 3.6.

Consider again the stack example. We insert data into the stack with a push operation, which may or may not displace data already at the root node. If there is no data to displace, maybe no further communication is necessary. In this case guard B_ϵ is false. If we do displace the data at the root node, the root then has to make a choice as to where to send the data. Suppose the displaced data is moved to the left child, which in turn chooses to displace data to its left child. In some cases, therefore, we may simply shift data in the left-hand branches of the tree. Similarly, on a pop operation we might shift data in the left-hand branches of the tree only. Hence, the case may occur on consecutive push and pop operations that the stack only uses the left-hand branches. Effectively this would be a pipeline of left-hand branches.

This leads us to the following conjectures. Given a tree of depth L, where L is $\log(N)$ and N is the number of cells, and the tree behaviour described above, then the worst-case response time RT of the tree is the same as that of a linear pipeline of length L and multiplication factor $n = 1$. Secondly, given a tree of depth L, where L is $\log(N)$, and the tree behaviour described above, then the bounds on the average response time, AR, of the tree are those of a linear pipeline of length L and multiplication factor $n = 1$.

We can give similar conjectures for the cycle time and average cycle time. We note that none of these conjectures will give us a critical bound as we only define critical bounds for deterministic behaviours.

Therefore, given the most general behaviour above, the response time and average response time of a tree may be no better than a pipeline in the worst case.

Note, however, that a balanced tree is going to have less depth, L, than a pipeline with equivalent capacity. Since worst-case response time depends on the depth, a tree may well be better. The upper bound on average response time, on the other hand, is independent of depth for a linear pipeline. Using a tree would offer no obvious advantage for the above behaviour, and the extra hardware required for switching between branches would probably add delay. Note, however, the limitations of our model of choice. If we always choose the worst-case path, the tree may fare worse than the pipeline. In many cases, such as the stack example above, we might expect a more favourable distribution of communications that used all parts of the tree evenly. In this case the average response time is bounded as with the non-conditional behaviour described earlier in this chapter. That is, the average response time for a tree with arity $n > 1$ is bounded from above by $\max\{\Delta_{0,r}, n \cdot \Delta_{0,r} + n \cdot \Delta_{0,f} - \delta_e\}$ given $\Delta_{i,f} \leq \Delta_{0,f} \cdot n^i$ and $\Delta_{i,r} \leq \Delta_{0,r} \cdot n^i$. So we can have exponentially slow cells deeper in the tree without affecting the upper bound on the average response time. The average response time for a pipeline with multiplication factor $n = 1$ is bounded from above by $MC_{\delta_e} - \delta_e$. Exponentially slow cells will be reflected in the worst-case delay of such a pipeline. So, if we can guarantee a more favourable distribution of communications than the worst-case, a tree may well be a better choice for average-case response time also.

5.7 Summary

We have applied the timing measures of Chapters 3 and 4 to tree-based architectures. We have assumed that each level of the tree has the same delay parameters. This chapter has proven the following results.

- Response Time, RT, is bounded as follows

$$\begin{aligned} RT \;\leq\; & \max(\{\Delta_{k,r} + \Delta_{k-1,f} + \sum_{h=0}^{k-1}(\Delta_{h,r} - \delta_{h,r}) \\ & -(\delta_{0,r} + \delta_e)n^{k-1} + \delta_{0,r} \mid 1 \leq k \leq L\} \cup \{\Delta_{0,r}\}) \end{aligned}$$

  This differs from the result given in Chapter 3 only in the factor n^{k-1}. As with pipelines, the worst-case response time depends on the variation in reverse delays. Again, if there is no variation in reverse delays, the bottleneck is the cycle of delays between levels.

  We compute the cycle time $CT_{ub} = RT_{ub} + \Delta_e$ using Lemma 3.5.

- The average response time AR and average cycle time AC for trees with arity $n > 1$ are bounded as follows.

$$\begin{aligned} \delta_{0,r} \;&\leq AR \leq\; \max\{\Delta_{0,r}, \Delta_{1,r} + \Delta_{0,f} - \delta_e\} \\ \delta_{0,r} + \delta_e \;&\leq AC \leq\; \max\{\Delta_{0,r} + \Delta_e, \Delta_{1,r} + \Delta_{0,f}\} \end{aligned}$$

  given $\Delta_{i,f} \leq \Delta_{0,f} \cdot n^i$ and $\Delta_{i,r} \leq \Delta_{0,r} \cdot n^i$ for $0 \leq i \leq L$. We optimise such a tree by optimising cells at the first two levels and at the environment for speed. We have some freedom to optimise other cells for low power or low area without affecting the worst-case bounds on average response time or average cycle time.

Chapter 6

Joining Components

So far we have considered only pipelines and trees. If we have formulae for the performance of these architectures, would it be possible to find formulae for networks of pipelines and trees? Would it be possible to find formulae for other behaviours? It may be possible to connect components, such as pipelines and trees, to form new structures. In some restricted cases it is possible to derive bounds on the average-cycle time of the new structure from the bounds previously derived from the components. For example, if we connect two of our pipelines with multiplication factor $n = 1$ together to form one long pipeline, we can derive results for the composite pipeline. Moreover we could join a pipeline with multiplication factor $n = 1$, to a pipeline with multiplication factor $n = 2$, or to a tree. We can still derive throughput results for the composite structure by looking at the components. We do not need to prove results from scratch.

In this chapter we only apply one of our measures, average cycle time, AC. Average cycle time is the easiest measure to compute, because we can simplify the delay model to fixed delays due to Lemma 4.1. Moreover, we need not worry about which edges represent the environment, as all edges are treated identically. Both of these properties allow us to generalise the computation of bounds for average cycle time to arbitrary graphs. We work with critical bounds as defined in Section 4.2, because these bounds are unique.

In this section:

- We generalise average cycle time of pipelines and trees to average cycle time of arbitrary events in arbitrary behaviours. We introduce the concept of *ratio*, which relates the average cycle time of two events to each other. If we know the average cycle time of one event, and we know the ratio of that event relative to another event, we can calculate the average cycle time of that other event.

- We give a method for composing two arbitrary graphs by a synchronisation set of events and give bounds on the average cycle time for the composite. The synchronisation set is a matching of events from two components. The components are joined by fusing together each pair of events.

  Let two graphs A and B be joined to form a composite C. Let event e be an event in the synchronisation set that is used to join A and B. Therefore event e is the fusing of an event in graph A, and an event in graph B. Then $AC_{ub,C}(e)$, the critical upper bound on the average cycle time of event e in C, satisfies

  $$\max\{AC_{ub,A}(e), AC_{ub,B}(e)\} \leq AC_{ub,C}(e) \leq AC_{ub,A}(e) + AC_{ub,B}(e)$$

  where $AC_{ub,A}(e)$ and $AC_{ub,B}(e)$ are the critical upper bounds on the cycle times of event e in A and B before composition.

In words, we are given an event in A and an event in B. These are fused to become one event e in the composite C. The critical bound on the average cycle time of e in C, $AC_{ub,C}(e)$, cannot be smaller than the average cycle time bounds of the event in either of the original graphs. Moreover, $AC_{ub,C}(e)$ cannot be larger than the sum of the original average cycle time bounds. The formula gives us a straightforward estimate of the critical upper bound of a composite behaviour. Note that this formula is very general. We can choose arbitrary behaviours A and B as long as both behaviours are deterministic and the behaviours can be composed into a composite C.

We give a similar result for the lower bounds.

$$\max\{AC_{lb,A}(e), AC_{lb,B}(e)\} \leq AC_{lb,C}(e) \leq AC_{lb,A}(e) + AC_{lb,B}(e)$$

- We give tight bounds on the average cycle time on two special cases of combining two components.

  1. When the two components are "mirror images" of each other (as defined later), the critical bounds on the average cycle times of the composite are exactly the critical bounds of the average cycle time of the components. Thus, if behaviours A and B are "mirror images" and behaviour C is the composite, we have

     $$AC_{ub,C}(e) = AC_{ub,A}(e) = AC_{ub,B}(e)$$

     where $AC_{ub,C}(e)$ is the critical upper bound on the average cycle time of an event e in the synchronisation set, and where $AC_{ub,A}(e)$ and $AC_{ub,B}(e)$ are the critical upper bounds of event e in the original graphs A and B. Using similar definitions we have

     $$AC_{lb,C}(e) = AC_{lb,A}(e) = AC_{lb,B}(e)$$

2. When the synchronisation set has only occurrences of one periodic event (as defined later), the critical bound on the average cycle time of the composite, is exactly the maximum of the critical bounds on the average cycle time of the components. Thus, if behaviours A and B are composed to form behaviour C, we have

$$AC_{ub,C}(e) = \max\{AC_{ub,A}(e), AC_{ub,B}(e)\}$$

where $AC_{ub,C}(e)$ is the critical upper bound on the average cycle time of event e in the synchronisation set, and where $AC_{ub,A}(e)$ and $AC_{ub,B}(e)$ are the critical upper bounds of event e in the original graphs A and B. Using similar definitions we have

$$AC_{lb,C}(e) = \max\{AC_{lb,A}(e), AC_{lb,B}(e)\}$$

Examples are given of both special cases of combining components.

As we deal with critical bounds in this chapter, we only deal with deterministic behaviours. If we do not have a deterministic behaviour it is difficult to determine what the critical path is. This chapter can be broken into two distinct parts. The first part discusses generalising cycle time and the concept of ratio of events. The second part discusses composing behaviours.

6.1 Generalising Cycle Time

Throughout the calculations in this thesis, the events of interest are those at level 0. Specifically we are interested in a request at level 0 and its corresponding acknowledgment. More generally, the cycle time is the time separation between an event (e, i) and the next occurrence of that event $(e, i+1)$. This delay is called the cycle period by Burns [10].

Burns states that in a strongly connected subgraph, i.e., when there is a path from every vertex to every other vertex in the subgraph, the cycle time of each event in a cycle is the same as all other events in a cycle. Below, we generalise this notion of cycle time of events by including the relative freqency of an event occurring within a cycle. That is, within a cycle of a graph an event e may occur more times than another event f. Consider, for example, a pipeline with multiplication factor $n = 2$. Events at the end cell occur far less frequently than those at the environment. Therefore, the ratio of events states how often an event occurs relative to another event.

Note that Burns's formulation used fixed delays. We also use fixed delays, as Lemma 4.1 allows us to do this when we calculate bounds on average cycle time.

6.2 Ratio

The measures we have calculated so far in this thesis are based on separations between events at level 0, viz., paths through cell 0 and the environment. If we can find a timing relationship between arbitrary events and the events at level 0, we may be able to easily calculate other separations. In general we consider the relationship between two events e and f. We make the restriction that between occurrences of events e and f there must always be a path with a finite number of edges and vice-versa. As noted in the introduction, we also restrict the behaviours to deterministic behaviours.

We calculate the ratio of e relative to f, denoted by ratio(e/f), which is the number of times an event e occurs for every occurrence of event f. Formally, let there be two events e and f. For each occurrence (e, i) of e let there be some occurrence (f, j) of f such that there is a path with a finite number of edges from (e, i) to (f, j). Let there be a similar path between (f, j) to occurrence $(e, i+x), x > 0$. We assume that all edge delays are finite, and therefore the paths between e and f have finite delay. Let us construct an infinite subgraph of the unfolded process graph that contains *all* paths between (e, i) and $(e, i+k-1)$ where $k \rightarrow \infty$. By definition of occurrence indices, this graph contains k instances of event e. We compare this number with the number of times that f occurs. The ratio is then defined as

$$\begin{aligned} \text{ratio}(e/f) &= \lim_{k \rightarrow \infty} \frac{num(e)}{num(f)} \\ &= \lim_{k \rightarrow \infty} \frac{k}{num(f)} \end{aligned}$$

Note that

- ratio$(e/f) \geq 0$ always, since when $k \rightarrow \infty$ there is a positive number of occurrences of e and f. WE have ratio$(e/f) = 0$ when $num(f)$ is infinite and $num(e)$ is finite.
- ratio(e/f) is the reciprocal of ratio(f/e).
- $num(f)$ is a function of k.

Consider our pipeline with multiplication factor n. When $n = 1$, we can pick any events e and f and their ratio is 1. This case is dealt with by Burns [10]. Typically, we would compare an event f with another event at the environment, for example, $e = r_0$. When $n = 2$ in a linear array, for example, the events r_0 and

a_L have ratio$(r_0/a_L) = 2^L$. In general, the ratio of events a_i and r_i relative to r_0 or a_0 is n^i for linear arrays with multiplication factor n.

6.2.1 Using Ratio to Compute Cycle Time

If we know the bounds on the average cycle time AC at one event, we can derive the average cycle time at any other event, if we know the ratios of the events.

Theorem 6.1 *Let e and f be events with finite non-zero ratio(e/f). Suppose the average cycle time of event e is bounded from below by finite critical bound $AC_{lb}(e)$ and from above by finite critical bound $AC_{ub}(e)$. Then the average cycle time of event f is bounded from below by finite critical bound $AC_{lb}(f)$ given by*

$$AC_{lb}(f) = AC_{lb}(e) \cdot \mathit{ratio}(e/f)$$

and from above by finite critical bound $AC_{ub}(f)$ given by

$$AC_{ub}(f) = AC_{ub}(e) \cdot \mathit{ratio}(e/f)$$

□

Before proving the theorem, we give an example. Suppose the cycle time of r_0 in a linear pipeline with multiplication factor $n = 2$, and length $L = 3$ is bounded from below by 2 and from above by 5. Now suppose we wish to find the cycle time of event a_2. Let r_0 be event e, and event a_2 be event f.

$$\begin{aligned} \text{ratio}(e/f) &= \frac{1}{n^{-2}} \\ &= \frac{1}{1/4} \\ &= 4 \end{aligned}$$

Then

$$\begin{aligned} AC_{lb}(f) &= AC_{lb}(e) \cdot \text{ratio}(e/f) \\ &= 2 \cdot 4 = 8 \\ AC_{ub}(f) &= AC_{ub}(e) \cdot \text{ratio}(e/f) \\ &= 5 \cdot 4 = 20 \end{aligned}$$

Thus the average cycle time of f is bounded as $8 \le AC(f) \le 20$.

Proof. We will prove only that $AC_{ub}(f) = AC_{ub}(e) \cdot \text{ratio}(e/f)$ is the critical upper bound for the average cycle time of f. The lower bound proof is identical.

Let us choose a path P of the form $(e,i) \rightsquigarrow (f,j) \rightsquigarrow (f,j+l) \rightsquigarrow (e,i+k)$, where the index j is chosen to be as small as possible, and the index l is chosen to be as large as possible, for a given k. Since the behaviour is deterministic, the values j and l are uniquely determined for a given i and k.

We know from Section 4.2 that

$$\begin{aligned} &\lim_{k\to\infty} \frac{(\text{max. delay}\{(e,i) \rightsquigarrow (e,i+k)\})}{k} \\ = \;& AC_{ub}(e) \end{aligned} \tag{6.1}$$

as this is the definition of the upper bound. Similarly we have

$$\begin{aligned} &\lim_{l\to\infty} \frac{(\text{max. delay}\{(f,j) \rightsquigarrow (f,j+l)\})}{l} \\ = \;& AC_{ub}(f) \end{aligned} \tag{6.2}$$

Note that l and k are related by a monotonically non-decreasing function $l = h(k)$. That is, l gets bigger as k does. We can change limits thus.

$$\begin{aligned} & \lim_{k\to\infty} \frac{(\text{max. delay}\{(f,j) \leadsto (f,j+l)\})}{l} \\ = & \lim_{l\to\infty} \frac{(\text{max. delay}\{(f,j) \leadsto (f,j+l)\})}{l} \end{aligned}$$

We also have

$$\lim_{k\to\infty} \frac{h(k)}{k} = \lim_{k\to\infty} \frac{l}{k} = \text{ratio}(f/e) \tag{6.3}$$

Similarly we have $\lim_{l\to\infty} \frac{k}{l} = \text{ratio}(e/f)$. For the analysis below we let $k \to \infty$ and hence $l \to \infty$ also.

We note that the paths $(e,i) \leadsto (f,j)$ and $(f,j+l) \leadsto (e,i+k)$ have finite maximum delays because:

1. for ratio to be defined, we need to have a path between e and f with a finite number of edges.
2. we assume edge delays to be finite.
3. j is as small as possible.
4. l is as large as possible.

We call the maximum delays on these paths D_1 and D_2 respectively.

$$\begin{aligned} & AC_{ub}(e) \\ = & \lim_{k\to\infty} \frac{(\text{max. delay}\{(e,i) \leadsto (e,i+k)\})}{k} \quad \{\text{From equation 6.1}\} \\ \geq & \lim_{k\to\infty} \frac{(\text{max. delay}\{(e,i) \leadsto (f,j) \leadsto (f,j+l) \leadsto (e,i+k)\})}{k} \end{aligned}$$

$$
\begin{aligned}
&= \lim_{k\to\infty} \frac{\text{max. delay}\{(e,i) \rightsquigarrow (f,j)\} + \text{max. delay}\{(f,j+l) \rightsquigarrow (e,i+k)\}}{k} \\
&\quad + \lim_{k\to\infty} \frac{(\text{max. delay}\{(f,j) \rightsquigarrow (f,j+l)\})}{k} \\
&= \lim_{k\to\infty} \frac{D_1 + D_2}{k} + \lim_{k\to\infty} \frac{(\text{max. delay}\{(f,j) \rightsquigarrow (f,j+l)\})}{k} \\
&\quad \{\text{Since } D_1 \text{ and } D_2 \text{ are finite}\} \\
&= 0 + \lim_{k\to\infty} \frac{(\text{max. delay}\{(f,j) \rightsquigarrow (f,j+l)\})}{k}
\end{aligned}
$$

A property of limits is

$$\lim_{k\to\infty} f(k)\cdot g(k) = \left(\lim_{k\to\infty} f(k)\right)\cdot\left(\lim_{k\to\infty} g(k)\right)$$

when $\lim_{k\to\infty} f(k)$ and $\lim_{k\to\infty} g(k)$ are finite. Then

$$
\begin{aligned}
AC_{ub}(e) &= \lim_{k\to\infty} \frac{(\text{max. delay}\{(f,j) \rightsquigarrow (f,j+l)\})}{k} \\
&\quad \{\text{Properties of limits}\} \\
&= \left(\lim_{k\to\infty} \frac{l}{k}\right)\cdot\left(\lim_{k\to\infty} \frac{(\text{max. delay}\{(f,j) \rightsquigarrow (f,j+l)\})}{l}\right) \\
&\quad \{\text{Changing limits}\} \\
&= \left(\lim_{k\to\infty} \frac{l}{k}\right)\cdot\left(\lim_{l\to\infty} \frac{(\text{max. delay}\{(f,j) \rightsquigarrow (f,j+l)\})}{l}\right) \\
&= \left(\lim_{k\to\infty} \frac{l}{k}\right)\cdot AC_{ub}(f) \quad \{\text{From equation 6.2}\} \\
&= \text{ratio}(f/e)\cdot AC_{ub}(f) \quad \{\text{From equation 6.3}\}
\end{aligned}
$$

So we have that

$$
\begin{aligned}
AC_{ub}(e) &\geq \text{ratio}(f/e)\cdot AC_{ub}(f) \\
\text{ratio}(e/f)\cdot AC_{ub}(e) &\geq AC_{ub}(f) \quad \{\text{Since ratio}(e/f) = 1/\text{ratio}(f/e)\} \qquad (6.4)
\end{aligned}
$$

If we swap e and f in the above proof we can show that

$$AC_{ub}(f) \geq \text{ratio}(e/f)\cdot AC_{ub}(e) \qquad (6.5)$$

since the relationships between e and f are identical. From equations 6.4 and 6.5 we obtain equality.

$$AC_{ub}(f) \;=\; \text{ratio}(e/f) \cdot AC_{ub}(e)$$

Lastly, since ratio(e/f) and $AC_{ub}(e)$ are finite, $AC_{ub,(f)}$ must also be finite.

□

We have now greatly extended the concept of cycle time and can now determine the cycle time of any event in a system. Consider the example of an eager stack in Section 4.10. We wished to calculate the average cycle time of event P_0, where P_0 was not an event at the environment. We solved this in Section 4.10 by discarding the left-hand cell, making cell 0 the environment, and so bringing event P_0 to the environment. With the results above, there would be no need to make this hack. Since the multiplication factor of the eager stack is $n = 1$, the ratio of any two events is 1. From Theorem 6.1, the bounds on the average cycle time of all events in the system must therefore be equal. We can then calculate the bounds on the average cycle time at the environment, and apply it to event P_0.

6.3 Defining Composition of Behaviours

Previously we have restricted ourselves to pipelines and trees. What we would like to do is to take components for which we know the average cycle time bounds, and to combine them into new components. We would like to do so in a manner that allows us to easily compute the average cycle time bounds for the composite from the average cycle time bounds already computed for the components. If we can do this, then we will have greatly expanded the behaviours we can apply our method to.

In the rest of the chapter we discuss

- a method whereby we join two components to form one new component.
- the timing behaviour of the composite.

We need to decide which compositions are possible and to define the behaviour that results from the composition.

6.3.1 Generalising the Delay Model

The delay model we describe in Section 2.6 uses delay parameters for forward and reverse delays for individual cells. That is, we have a set of parameters $\delta_{h,r}, \Delta_{h,r}, \delta_{h,f}, \Delta_{h,f}$ for a cell h. The formulae we use to calculate time stamps use the direction of an edge, i.e., whether the edge is a "forward" or a "reverse" edge. A graph of an arbitrary behaviour may not have cells enumerated by their depth within the pipeline or tree. An arbitrary behaviour may also not have "forward" and "reverse" delays either. It is easy to generalise our delay model, however. In fact, the model formalised in Section 2.6 is a restriction of the more general model described by an arbitrary behavioural graph. For example, suppose an event c has predecessors a and b. The event c cannot happen until both a and b have occurred and the delays across $a \to c$ and $b \to c$ have occurred. Our old model prescribed the values that the delays across $a \to c$ and $b \to c$ take depending on whether the edges were considered "forward" or "reverse" edges.

In general, let c be an event. Let c_{pred} be the set of direct predecessors of c. Then

$$\begin{aligned} & \max\{T(b) + \text{min. delay}(b \rightarrow c) \mid b \in c_{pred}\} \\ \leq \; & \\ & T(c) \\ \leq \; & \\ & \max\{T(b) + \text{max. delay}(b \rightarrow c) \mid b \in c_{pred}\} \end{aligned} \tag{6.6}$$

The definitions given in Section 2.6 are as follows for an event c.

$$T_{pred}(c) + \delta_{h,f} \leq T(c) \leq T_{pred}(c) + \Delta_{h,f}$$

if node c is a request, and

$$T_{pred}(c) + \delta_{h,r} \leq T(c) \leq T_{pred}(c) + \Delta_{h,r}$$

if node c is an acknowledgment. $T_{pred}(c)$ is defined as

$$T_{pred}(c) = \max\{T(b) \mid b \text{ is a direct predecessor of c}\}$$

A simple substitution of $T_{pred}(c)$ gives us the general formula of Equation 6.6, albeit with a restricted set of possible edge delays.

6.3.2 Composition of Two Graphs

To increase the flexibility of our model we would like to be able to compose behaviours in a simple, consistent way. One method would be to synchronise behaviours by adding extra dependencies. In our graphical behavioural model this would mean adding extra edges between graphs. While this model is visually quite intuitive, the actual synchronisation occurs at the events. For example, in the pipeline examples we have an event a_0 that has direct predecessors r_0 and a_1. The event a_0 acts as a synchronisation point for the two incoming edges from r_0 and a_1. Therefore, rather than synchronise using edges, we synchronise using events. That is, we compose two behaviours, represented by behavioural graphs A and B, by means of a synchronisation set of events. An event A is paired with and event in B and these events are fused together into one event in the composite. This new event in the composite retains all the predecessors and successors that it had in both behavioural graphs.

A problem with synchronising on events is that we might construct a composite graph that is messy and difficult to understand. In the following example, we synchronise *events* and then transform the resulting graph into something more familiar. This transformation does not alter the overall timing behaviour. Note that we assume that the synchronisation itself has negligible delay. In practice, we need something like a C-element to perform the synchronisation of two components, and the C-element will add delay to the synchronised paths. For mathematical simplicity we ignore this delay.

In the example below, let us synchronise r_3 that belongs to process graph A and a'_0 that belongs to process graph B. That is, we wish to make occurrence (r_3, i) occur at the same time as (a'_0, i'). See Figure 6.1. For later use, example predecessors and successors of the two events are given.

We can synchronise two events by joining them together to become one event. See Figure 6.2. We synchronise on events, not on edges. As will become clear in the next section, transforming the graph into another graph with equivalent timing behaviour can sometimes give us a cleaner and more intuitive representation of the

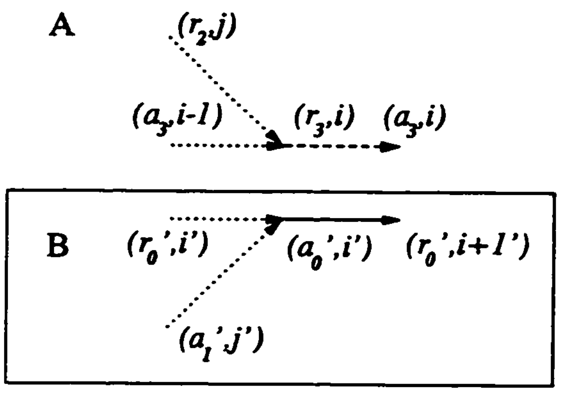


Figure 6.1: Events (r_3, i) in A and (a_0', i') in B drawn with their respective successors (a_3, i) in A and $(r_0', i' + 1)$ in B. Example predecessors are also drawn. The edge in A illustrated by a dashed line has delay bounds $[\delta_{3,r}, \Delta_{3,r}]$. The edge in B illustrated by a solid line has delay bounds $[\delta_e', \Delta_e']$.

behaviour. We transform the graph of Figure 6.2 to the graph of Figure 6.3 by splitting the event r_3/a_0' back into two events (r_3, i) and (a_0', i') and adding two edges. The delay bounds on the dotted edges are $[\delta_{3,r}, \Delta_{3,r}]$ and the delay bounds on the solid edges are $[\delta_e', \Delta_e']$. That is, we have preserved the delays of the original edges, and also conveniently preserved the property that "forward" edges end in a request and that "reverse" edges end in an acknowledgment.

We can show that the timing behaviour of Figure 6.2 is equivalent to that of Figure 6.3, at least at the points at which we measure the delay. The events (r_3, i) and (a_0', i') are no longer synchronised in Figure 6.3. What we show instead is that the bounds on the timestamps $T(a_3, i)$ and $T(r_0', i' + 1)$, i.e., the timestamps of the successor events, are the same for both the behaviour described by Figure 6.2 and the behaviour described by Figure 6.3. For completeness, below is the proof.

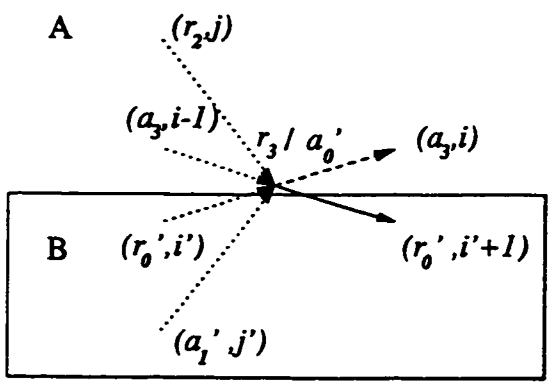


Figure 6.2: Events (r_3, i) in A and (a'_0, i') in B synchronised to become one event. The occurrence index of r_3/a'_0 is left unspecified as we do not know it.

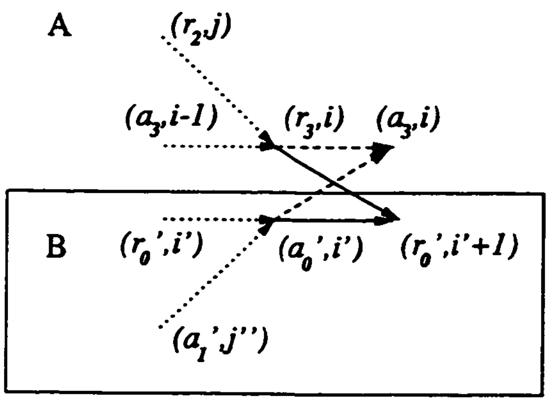


Figure 6.3: The transformed graph. Dashed edges have delay bounds $[\delta_{3,r}, \Delta_{3,r}]$ and solid edges have delay bounds $[\delta'_e, \Delta'_e]$.

Proof. Consider $T(a_3, i)$. Let us find the upper bound on the possible timing assignments for (a_3, i) in both the behaviours. Let us look at the behaviour given in Figure 6.3 first. From Section 6.3.1 we have

$$\begin{aligned}
T(a_3,i) &\leq \max\{T(b) + \text{max. delay}(b \rightarrow (a_3,i)) \mid b \in (a_3,i)_{pred}\} \\
&= \max\{T(b) + \Delta_{3,r} \mid b \in \{(a_0',i'),(r_3,i)\}\} \\
&= \max\{T(b) \mid b \in \{(a_0',i'),(r_3,i)\}\} + \Delta_{3,r} \qquad (6.7)
\end{aligned}$$

We also have

$$\begin{aligned}
T(a_0',i) &\leq \max\{T(b) + \text{max. delay}(b \rightarrow (a_0',i')) \mid b \in (a_0',i')_{pred}\} \\
&= \max\{T(b) + \Delta_{0,r} \mid b \in \{(a_1',j'),(r_0',i')\}\} \\
&= \max\{T(b) \mid b \in \{(a_1',j'),(r_0',i')\}\} + \Delta_{0,r} \qquad (6.8)
\end{aligned}$$

$$\begin{aligned}
T(r_3,i) &\leq \max\{T(b) + \text{max. delay}(b \rightarrow (r_3,i)) \mid b \in (r_3,i)_{pred}\} \\
&= \max\{T(b) + \Delta_{2,f} \mid b \in \{(a_3,i-1),(r_2,j)\}\} \\
&= \max\{T(b) \mid b \in \{(a_3,i-1),(r_2,j)\}\} + \Delta_{2,f} \qquad (6.9)
\end{aligned}$$

Substituting Equations 6.8 and 6.9 into Equation 6.7 and we have

$$\begin{aligned}
T(a_3,i) &\leq \max\{T(b) \mid b \in \{(a_0',i'),(r_3,i)\}\} + \Delta_{3,r} \\
&\qquad \{\text{ Substituting }\} \\
&\leq \max\{ \\
&\quad \max\{T(b) \mid b \in \{(a_1',j'),(r_0',i')\}\} + \Delta_{0,r}, \\
&\quad \max\{T(b) \mid b \in \{(a_3,i-1),(r_2,j)\}\} + \Delta_{2,f}\} + \Delta_{3,r} \qquad (6.10)
\end{aligned}$$

Consider $T(a_3,i)$ in Figure 6.2. We have

$$\begin{aligned} T(a_3,i) &\leq \max\{T(b) + \text{max. delay}(b \rightarrow (a_3,i)) \mid b \in (a_3,i)_{pred}\} \\ &= \max\{T(b) + \Delta_{3,r} \mid b \in \{r_3/a_0'\}\} \\ &= T(r_3/a_0') + \Delta_{3,r} \end{aligned} \tag{6.11}$$

We also have that

$$\begin{aligned} T(r_3/a_0') &\leq \max\{T(b) + \text{max. delay}(b \rightarrow r_3/a_0') \mid b \in (r_3/a_0')_{pred}\} \\ &= \max\{T(b) + \text{max. delay}(b \rightarrow r_3/a_0') \mid \\ &\quad b \in \{(a_1',j'),(r_0',i'),(a_3,i-1),(r_2,j)\}\} \\ &= \max\{ \\ &\qquad \max\{T(b) + \Delta_{0,r} \mid b \in \{(a_1',j'),(r_0',i')\}\}, \\ &\qquad \max\{T(b) + \Delta_{2,f} \mid b \in \{(a_3,i-1),(r_2,j)\}\}\} \\ &= \max\{ \\ &\qquad \max\{T(b) \mid b \in \{(a_1',j'),(r_0',i')\}\} + \Delta_{0,r}, \\ &\qquad \max\{T(b) \mid b \in \{(a_3,i-1),(r_2,j)\}\} + \Delta_{2,f}\} \end{aligned} \tag{6.12}$$

Substituting Equation 6.12 into Equation 6.11 and we get Equation 6.10. Similarly we can show that the lower bound on $T(a_3,i)$ is the same for both Figure 6.2 and Figure 6.3. Lastly, we can show that the bounds on $T(r_0',i'+1)$ are likewise the same for both behaviours.

□

Thus, for our purposes, the transformation does not alter the timing of (a_3,i) and $(r_0',i'+1)$.

6.3.3 Path Notation

We refer to a path of events and edges contained only in subgraph A by the notation $\alpha \leadsto_A \beta$, where α and β are events. The edges of path $\alpha \leadsto_A \beta$ are also contained in A, since all the vertices on the path are in A. As we shall see, some events in A are also in B; the path $\alpha \leadsto_A \beta$ may include events in the intersection of A and B.

6.3.4 Compatibility

Before we compose two behaviours, we must decide whether they are compatible. That is, can the behaviours be joined in a sensible fashion? We try to make restrictions on compatibility as loose as possible.

When we compose two behaviours, we will do so by composing the behavioural graphs at certain vertices. We choose a subset of vertices in each graph so that each of these chosen vertices is synchronised with a matching vertex in the other graph. These matched vertices become one vertex in the new composite graph and these vertices are what we call the synchronisation set. Compatibility is a restriction on the choice of synchronisation set.

Formally, let two unfolded process graphs $A =< V_A, E_A >$ and $B =< V_B, E_B >$ be given. V_A and V_B are the vertex sets, and hence represent the events of the system. Note that each event $\alpha \in V_A \cup V_b$ consists of an event label and an occurrence index. E_A and E_B are the edge sets and represent the dependencies. Let S be the synchronisation set of *events* such that $S = V_A \cap V_B$ (after composition). Each event in S, (e_i, λ_i), consists of the event label e_i and an occurrence index λ_i. Not every occurrence of a particular event need be represented in the synchronisation set. In all interesting cases S is an infinite set of events.

Let $\alpha_i = (e_i, \lambda_i)$ and $\alpha_j = (e_j, \lambda_j)$ be events in S. When A and B are *compatible* with respect to synchronisation set S, then there exists a path $\alpha_i \rightsquigarrow_A \alpha_j$ iff there is also a path $\alpha_i \rightsquigarrow_B \alpha_j$.

Let us give an example of a composition. We might compose two linear pipelines of length $L = 3$ and $L = 2$ respectively, by synchronising r_3 of pipeline A and a'_0 of pipeline B. We synchronise every occurrence of each event. It is fairly easy to prove that the two pipelines are compatible. The synchronisation set is $S = \{(e, \lambda) \mid \lambda \in I, e = r_3 = a'_0\}$ where I is the range of occurrence indices. Let us pick $(r_3, i), (r_3, j) \in S$ with $j > i$. Then we have $(r_3, i) \rightsquigarrow_A (r_3, j)$. The event r_3 in A is matched with a'_0 in B. We also have $(a'_0, i) \rightsquigarrow_B (a'_0, j)$. Therefore the two behaviours are compatible w.r.t. synchronisation set S. We will build on this example again in the next section when we compose A and B. In general, we would like to compose components that do not have the same behaviour or architecture; that is, we would like to be able to analyse more complicated circuits than we have done previously in the thesis. Combining two identical pipelines to form one long pipeline is of little practical interest, but, since we have already analysed linear pipelines, we can show that our composition method indeed produces correct results. We will address the more interesting problem of a tree-based FIFO later in the chapter.

6.3.5 Composition

If A and B are compatible w.r.t. synchronisation set S, then we may compose graphs A and B to form a composite behavioural graph.

The graph of the composite consists of the two unfolded process graphs A and B synchronised at events in the synchronisation set. An event $\alpha \in S$ has the predecessors of α in both the original graph A and the original graph B.

Suppose we have graphs A and B that represent linear pipelines of length L_A and L_B respectively and multiplication factor n_A and n_B. We abut the two pipelines to form one long pipeline of length $L_A + L_B + 1$. Intuitively, what we would like to do is as in Figure 6.4, which joins two micropipelines, with multiplication factors $n_A = n_B = 1$, of lengths $L = 3$ and $L = 2$ respectively to form a micropipeline of length $L = 6$. We choose to form the composite by synchronising r_3 and a'_0 at every occurrence. Therefore our synchronisation set for these two pipelines is $S = \{(e, \lambda) \mid \lambda \in I, e = r_3 = a'_0\}$ where I is the range of occurrence indices.

Now, suppose we compose the graphs by synchronising events r_3 and a'_0. We can compose graphs as in Figure 6.5. We have made our task simpler by starting pipeline B with an acknowledgment, viz, a'_0, rather than the usual request. Each synchronisation is illustrated using the method of Figure 6.3. We can see that composing the graphs this way gives us a graph that looks more or less like a graph of a pipeline with $L = 6$.

Note that we can easily apply our composition method to folded dependency graphs, by ensuring that every occurrence of a particular event is in the synchronisation set.

Lastly note that composition does not affect the ratio of events. Since the original graphs do not change, the relative frequency of events within the original graphs does not change either.

6.3.6 Bounds on Cycle Time for Composite Graphs

Now that we have a method of composition, we give the following theorem that relates the average cycle time of the composite graph to that of its components.

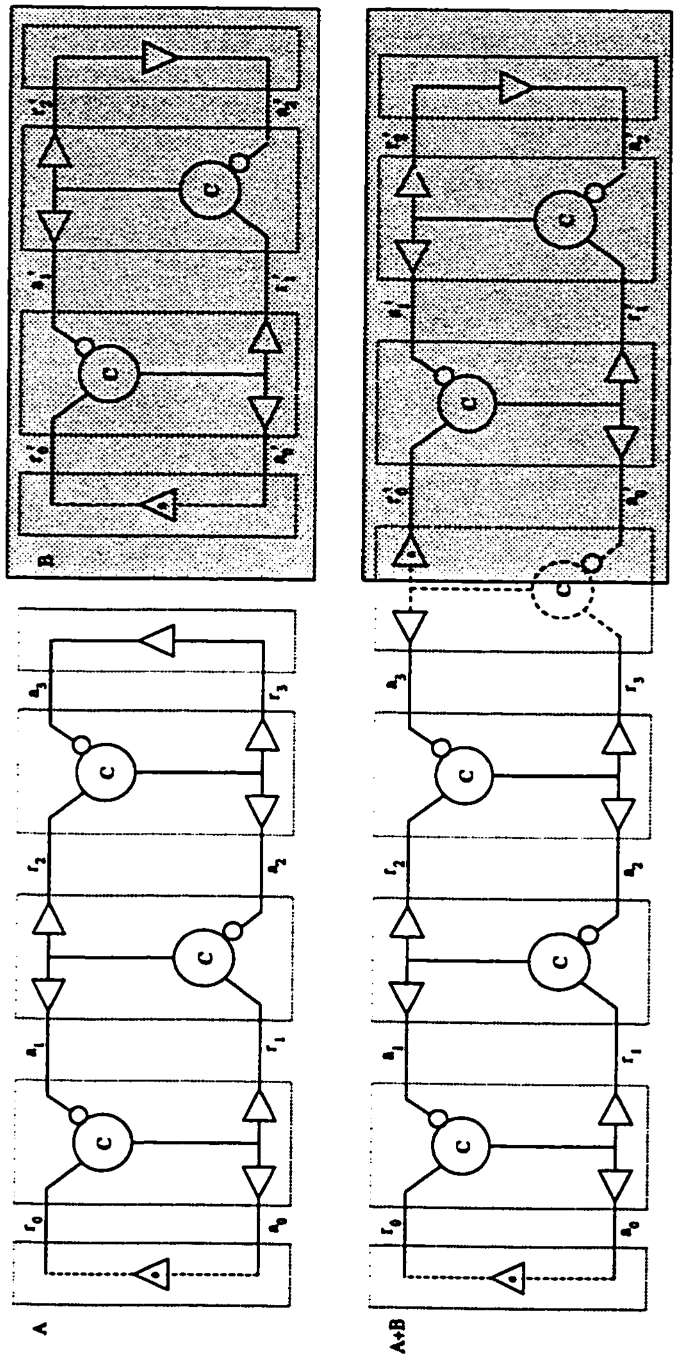


Figure 6.4: Joining two pipelines A and B of length $L_A = 3$ and $L_B = 2$, to form one new pipeline of length $L = 6$. The synchronisation of events r_3 and a_0' is done by means of a C-element. The new section is drawn with dashed lines.

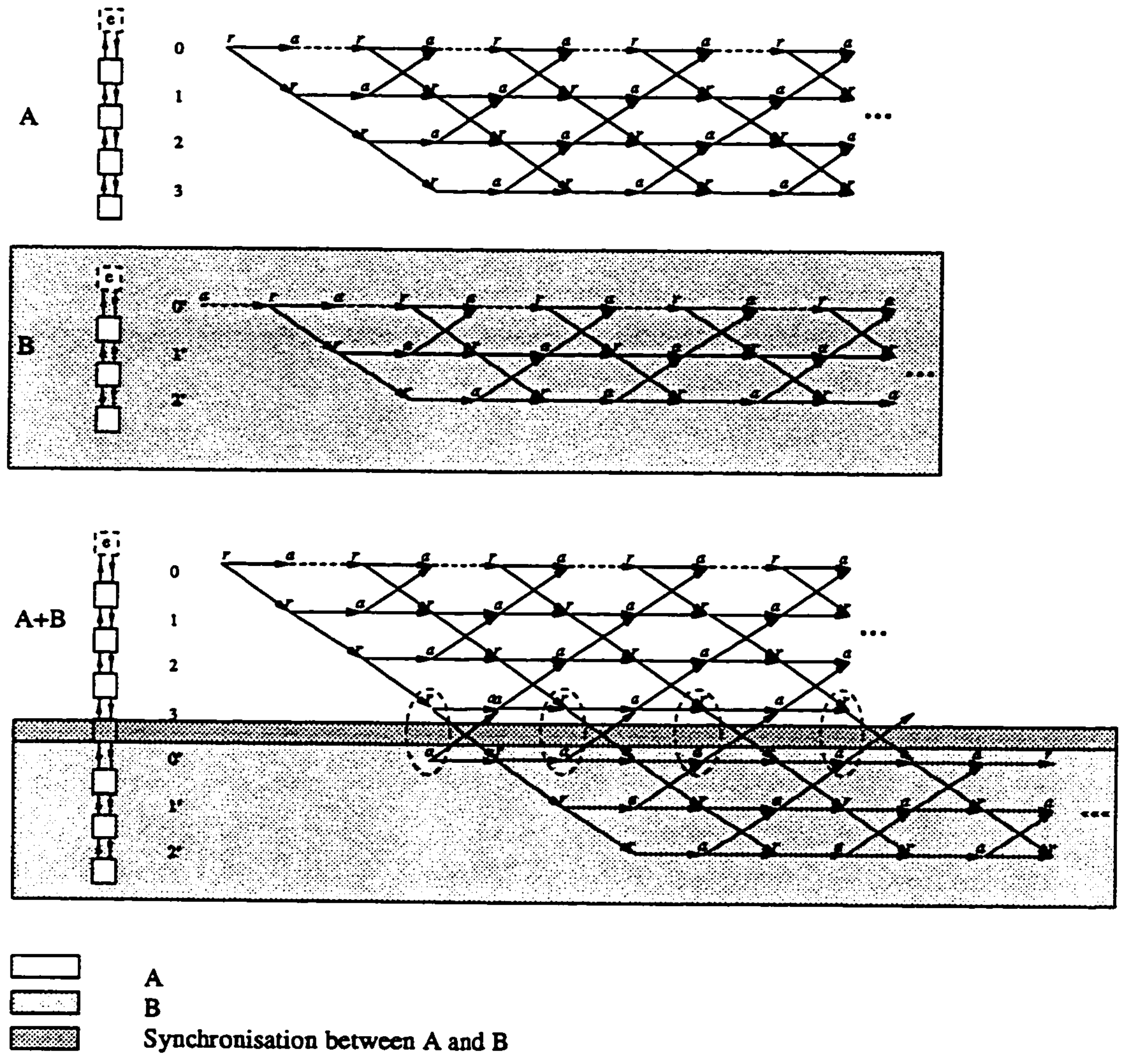


Figure 6.5: Joining two pipelines A and B of length $L_A = 3$ and $L_B = 2$, to form one new pipeline of length $L = 6$. The pipelines are synchronised at r_3 and a'_0. These events are enclosed in dashed ovals in the composite graph.

Theorem 6.2 *Let two process graphs A and B be compatible graphs with synchronisation set S. Let C be the composite graph of A and B w.r.t. S. Let $\alpha = (e, \lambda) \in S$ be an event.*

Let $AC_{ub,A}(e)$ be the critical upper bound on the average cycle time of e in original graph A.

Let $AC_{ub,B}(e)$ be the critical upper bound on the average cycle time of e in original graph B.

Let $AC_{ub,C}(e)$ be the critical upper bound on the average cycle time of e in composite graph C.

Then $AC_{ub,C}(e)$ satisfies

$$\max\{AC_{ub,A}(e), AC_{ub,B}(e)\} \le AC_{ub,C}(e) \le AC_{ub,A}(e) + AC_{ub,B}(e)$$

Using similar definitions for lower bounds, we also have

$$\max\{AC_{lb,A}(e), AC_{lb,B}(e)\} \le AC_{lb,C}(e) \le AC_{lb,A}(e) + AC_{lb,B}(e)$$

Proof. Let us prove first that $\max\{AC_{ub,A}(e), AC_{ub,B}(e)\} \le AC_{ub,C}(e)$

Let (e, i) and $(e, i+k)$ be occurrences of e in C. We have

$$\lim_{k\to\infty} \frac{\text{max. delay}\{(e,i) \rightsquigarrow_A (e,i+k)\}}{k} = AC_{ub,A}(e)$$

and

$$\lim_{k\to\infty} \frac{\text{max. delay}\{(e,i) \rightsquigarrow_B (e,i+k)\}}{k} = AC_{ub,B}(e)$$

The original paths $(e,i) \rightsquigarrow_A (e,i+k)$ in A and $(e,i) \rightsquigarrow_B (e,i+k)$ in B are not destroyed by the composition process. That is, the same paths exist in the composite C. Therefore, the critical path $(e,i) \rightsquigarrow_C (e,i+k)$ through C must be at least as long as the path through either of the subcomponents. Then

$$\begin{aligned} AC_{ub,C}(e) &= \lim_{k\to\infty} \frac{\text{max. delay}\{(e,i) \rightsquigarrow_C (e,i+k)\}}{k} \\ &\geq \max\{AC_{ub,A}(e), AC_{ub,B}(e)\} \end{aligned}$$

Now let us prove that $AC_{ub,C}(e) \leq AC_{ub,A}(e)+AC_{ub,B}(e)$. Let (e,i) and $(e,i+k)$ be occurrences of e in C.

$$\lim_{k\to\infty} \frac{\text{max. delay}\{(e,i) \rightsquigarrow_C (e,i+k)\}}{k} = AC_{ub,C}(e)$$

as before. Let P represent the critical path $(e,i) \rightsquigarrow_C (e,i+k)$. Let us break path P into two sets of segments corresponding to paths through A and paths not through A. Let there be h_0 segments in A and l_0 segments in B. We call the segments in A, A_h, where $0 < h \leq h_0$ and the segments not in A, B_l, where $0 < l < l_0$. Formally each segment A_h starts and ends in an event in S (noting that $S \subset A$), and all events on the path are events in A. Each segment B_l, starts and ends in an event in S. The other events on the path of the segment are not in A, i.e., they are in $B - S$.

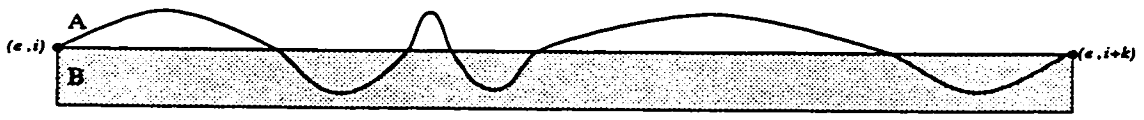


Figure 6.6: Illustration of a possible critical path, P, from (e,i) to $(e,i+k)$ made up of segments in A and B.

Each segment is constructed to be as long as possible. We can therefore reconstruct the path P from alternating segments from $\{A_h \mid 0 \leq h \leq h_0\}$ and $\{B_l \mid 0 \leq l \leq l_0\}$. Refer to Figure 6.6.

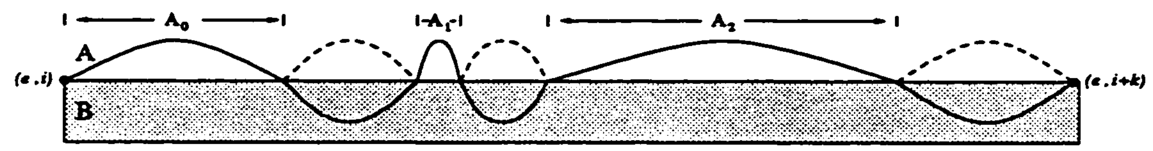


Figure 6.7: Illustration of a path, P_A, from (e,i) to $(e,i+k)$ made up only of events in A, using segments A_h and extra segments represented by dashed lines.

Let us link each segment A_h to segment A_{h+1} within A. Suppose segment A_h ends in event (f,λ_f) and segment A_{h+1} starts with event (g,λ_g). Since path P consists of $\cdots \rightarrow A_h \rightarrow B_l \rightarrow A_{h+1} \rightarrow \cdots$, there is a segment B_l in P of the form $(f,\lambda_f) \leadsto_B (g,\lambda_g)$. From the definition of compatibility, there is a path in A of the form $(f,\lambda_f) \leadsto_A (g,\lambda_g)$. Therefore we can construct a path of events only in A between (e,i) and $(e,i+k)$ of the form $A_1 \leadsto A_2 \leadsto \cdots \leadsto A_{h_0}$. We call this path P_A. See Figure 6.7.

By a similar construction we construct P_B.

We then have

$$\begin{aligned}
AC_{ub,C}(e) &= \lim_{k\to\infty} \frac{\text{max. delay}\{(e,i) \leadsto_C (e,i+k)\}}{k} \\
&= \lim_{k\to\infty} \frac{\sum_{h=1}^{h_0} (\text{delay}\{A_h\}) + \sum_{l=1}^{l_0} (\text{delay}\{B_l\})}{k} \\
&\leq \lim_{k\to\infty} \frac{\text{delay}\{P_A\} + \text{delay}\{P_B\}}{k} \\
&= \lim_{k\to\infty} \frac{\text{delay}\{P_A\}}{k} + \lim_{k\to\infty} \frac{\text{delay}\{P_B\}}{k} \\
&= AC_{ub,A}(e) + AC_{ub,B}(e)
\end{aligned}$$

□

The bounds given may seem somewhat wide, but the worst case, $AC_{ub,C}(e) = AC_{ub,A}(e) + AC_{ub,B}(e)$, can occur. Consider Figure 6.8. The two folded dependency graphs represent a couple of cooked-up behaviours.We are going to synchronise the two graphs by matching every occurrence of e_0 and f_0, and also e_1 and f_1. We use a synchronisation set consisting of all occurrences of e_0/f_0 and e_1/f_1. The cycle time of e_0 in graph A is 20 time units, and the cycle time of f_0 in graph B is 30 time units.

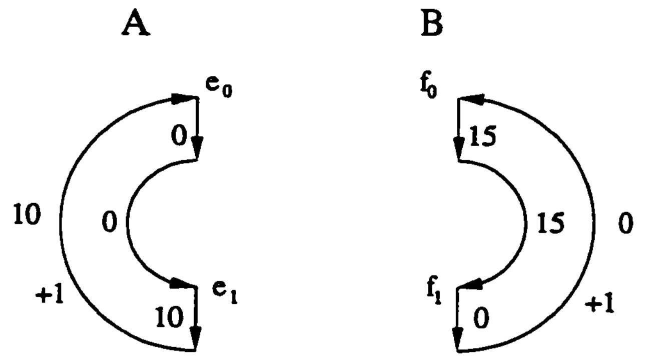


Figure 6.8: Two folded dependency graphs. The edges are annotated with (fixed) delays. The outer edges of graph A and B are also annotated with +1 indicating the occurrence index offset. Four events e_0, e_1, f_0, and f_1 are labelled.

Now consider the composite graph of Figure 6.9. The critical path (illustrated as the solid line) has a total delay of $20 + 30 = 50$. Therefore, the worst case may occur in some cases. By restricting the types of graphs, we can avoid this kind of worst case.

6.3.7 Mirror Images

A special case of joining components occurs when one component is a mirror

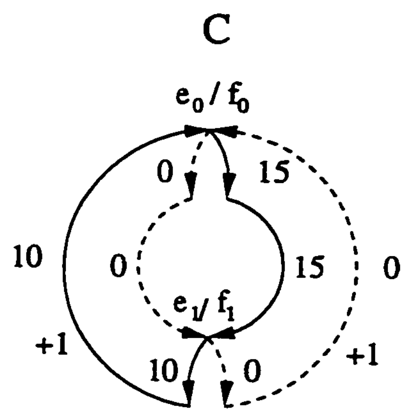


Figure 6.9: The composite graph. The solid line is the critical path.

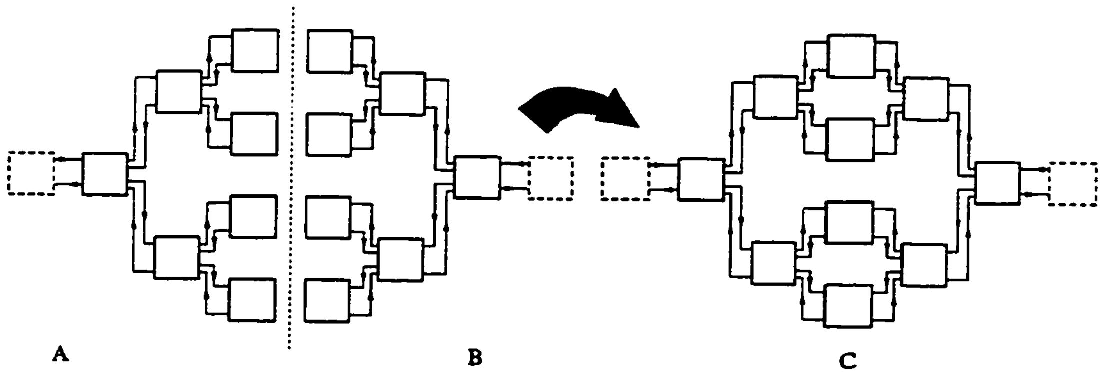


Figure 6.10: Two mirror-image trees become a FIFO.

image of another. A component A is a "mirror image" of another component B if a set of paths that exists between particular events in A has a matching set of paths in B. Roughly speaking, we take a component A and place a "mirror" somewhere over it and "reflect" the behaviour in it to get a new component B. To compose A and B we join the components at the point where we place a "mirror". Defining mirror images gives us a convenient way of composing two behaviours that are similar. We will be demonstrating this by abutting two trees together to form a FIFO as in Figure 6.10. Then we can derive bounds on the average cycle time of the FIFO from bounds we already know for trees without having to go through a full-blown proof. Another architecture we may be able to analyse by using mirroring is a ring. See Figure 6.11.

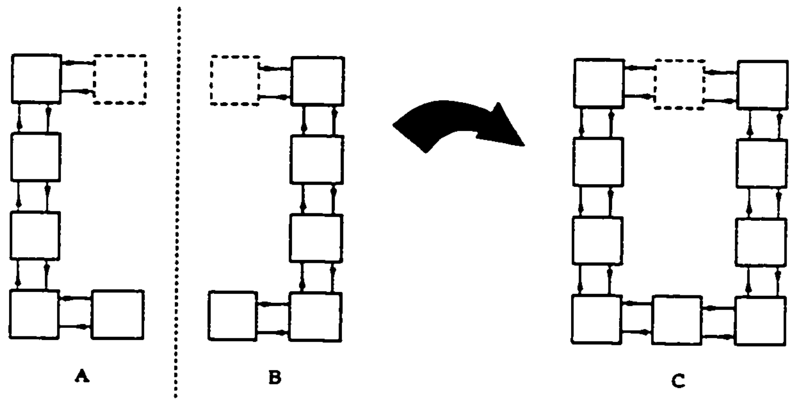


Figure 6.11: Two mirror-image pipelines become a ring.

Definition 6.3 . *Let A and B be process graphs with synchronisation set S. Let C be the composite of A and B w.r.t. synchronisation set S. Let two process graphs $A =< V_A, E_A >$ and $B =< V_B, E_B >$ be given. Let S be the synchronisation set of events such that $S = V_A \cap V_B$. Let $\alpha_i = (e_i, \lambda_i)$ and $\alpha_j = (e_i, \lambda_j)$ be events in S.*

A and B are mirror images *of each other iff*

1. *A and B are compatible with respect to synchronisation set S.*
2. *If all delays of all edges are maximised in C, the critical path, $\alpha_i \rightsquigarrow_A \alpha_j$, has the same total delay as the critical path, $\alpha_i \rightsquigarrow_B \alpha_j$.*
3. *If all delays of all edges are minimised in C, the critical path, $\alpha_i \rightsquigarrow_A \alpha_j$, has the same total delay as the critical path, $\alpha_i \rightsquigarrow_B \alpha_j$.*

□

Note that this definition says nothing about the events in A and B, only the paths in A and B with starts and end events in the intersection of A and B, i.e., the events in S. Note also, that the "mirroring", so to speak, is done around the synchronisation set. The mirror image of a component depends on what the synchronisation set is. In words, the mirror image of a component may depend on where we place the "mirror".

As an example of mirror images consider mirroring a pipeline with $n = 1$ and length $L = 3$. We can mirror about any single event here, but for the sake of this example we mirror about r_3. This becomes r'_3 in the mirrored pipeline. Our synchronisation set for composition consists of all occurrences of r_3/r'_3. We mirror the pipeline as in Figure 6.12. We also "mirror" the latencies. Thus edge $r_0 \rightarrow r_1$ has delay bounds $[\delta_{0,f}, \Delta_{0,f}]$ in the original pipeline A. In the mirror image B these delay bounds are applied to edge $r'_0 \rightarrow r'_1$.

Let us show that the three conditions of the definition of mirror images are satisfied for pipelines A and B. The synchronisation set is $S = \{(e, \lambda) \mid \lambda \in I, e = r_3 = r'_3\}$ where I is the range of occurrence indices.

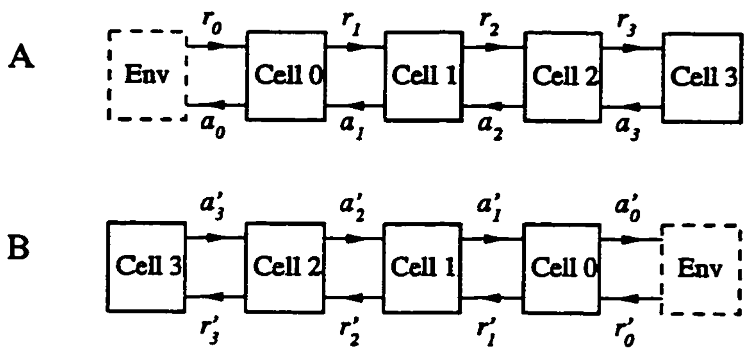


Figure 6.12: Pipeline is A is mirrored to become pipeline B.

1. Let $\alpha_i = (e_i, \lambda_i)$ and $\alpha_j = (e_j, \lambda_j)$ be events in S. When A and B are compatible with respect to synchronisation set S, then there exists a path $\alpha_i \rightsquigarrow_A \alpha_j$, iff there is also a path $\alpha_i \rightsquigarrow_B \alpha_j$. In this case we have $\alpha_i = (r_3/r'_3, \lambda_i)$ and $\alpha_j = (r_3/r'_3, \lambda_j)$. From the behaviour of a pipeline described in Section 2.3 we have a path path $(r_3, \lambda_i) \rightsquigarrow_A (r_3, \lambda_j)$ for any $\lambda_j \geq \lambda_i$. Similarly, path $(r'_3, \lambda_i) \rightsquigarrow_B (r'_3, \lambda_j)$ exists. Therefore A and B are compatible w.r.t. synchronisation set S.

2. Let all delays in C be maximised. Let $\alpha_0 \rightarrow_A \alpha_1 \rightarrow_A \alpha_2 \rightsquigarrow_A \alpha_i$ be a critical path for some $i > 0$ and where $\alpha_0, \alpha_i \in S$. Let the delay of edge $\alpha_k \rightarrow_A \alpha_{k+1}$ be Δ_k where $0 \leq k < i$. By our construction of pipeline B, there exists a path $\alpha'_0 \rightarrow_B \alpha'_1 \rightarrow_B \alpha'_2 \rightsquigarrow_B \alpha'_i$ in B, where α'_k in B is the reflection of α_k in A and where $\alpha'_0 = \alpha_0$ and $\alpha'_i = \alpha_i$. Moreover, by construction, the delays of edge $\alpha'_k \rightarrow \alpha'_{k+1}$ are Δ_k where $0 \leq k < i$. Thus, given a critical path in A with delay $\sum_{k=0}^{k=i-1} \Delta_k$ there exists a path in B with delay $\sum_{k=0}^{k=i-1} \Delta_k$. The path in B must also be critical. If there is a longer path $\alpha'_0 \rightsquigarrow_B \alpha'_i$ we could show that path $\alpha_0 \rightsquigarrow_A \alpha_i$ was not critical. This would be a contradiction.

3. The last condition is proven similarly to the previous condition.

We will give a theorem that relates the cycle time of the events in each original graph to their cycle time in the composite. First we give a preparatory lemma.

Lemma 6.4 *Given mirror image process graphs A and B, synchronisation set S, and event $e \in S$, then $AC_{ub,A}(e)$, the critical upper bound on the average cycle time of e in graph A, equals $AC_{ub,B}(e)$, the upper bound on the average cycle time of e in graph B. Similarly, $AC_{lb,A}(e) = AC_{lb,B}(e)$.*

Proof. When all delays are maximised the critical path between (e,i) and $(e,i+k)$ in A has the same delay as the critical path between (e,i) and $(e,i+k)$ in B by condition 2 of the definition of mirror images. Then

$$\begin{aligned} AC_{ub,A}(e) &= \lim_{k\to\infty} \frac{\text{max. delay}\{(e,i) \rightsquigarrow_A (e,i+k)\}}{k} \\ &\quad \{\text{By definition of mirror images.}\} \\ &= \lim_{k\to\infty} \frac{(\text{max. delay}\{(e,i) \rightsquigarrow_B (e,i+k))}{k} \\ &= AC_{ub,B}(e) \end{aligned}$$

The proof for lower bounds is similarly trivial.

□

Now we give the main theorem.

Theorem 6.5 *Let two process graphs A and B be mirror image graphs with synchronisation set $S = \{(e_i, \lambda_j) \mid i \in I, j \in J\}$ for some sets I and J. Let C be the composite graph of A and B. Let $e \in S$ be an event. Let $AC_{ub,A}(e) = AC_{ub,B}(e)$ be the critical upper bound on the average cycle time of e in both the original graph A, and in the original graph B. Let $AC_{ub,C}(e)$ be the critical upper bound on the average cycle time of e in the composite graph C. Then*

$$AC_{ub,C}(e) = AC_{ub,A}(e) = AC_{ub,B}(e)$$

Using similar definitions for critical lower bounds, we also have

$$AC_{lb,C}(e) = AC_{lb,A}(e) = AC_{lb,B}(e)$$

Proof. As usual we give the proof only for the upper bounds. We assume all edge delays are maximised. Let (e, i) and $(e, i + k)$ be occurrences of e in C.

$$\lim_{k \to \infty} \frac{\text{max. delay}\{(e,i) \rightsquigarrow (e,i+k)\}}{k} = AC_{ub,C}(e)$$

Let path P be a critical path between (e, i) and $(e, i + k)$. Let $(e, h) \rightsquigarrow (e, h + l)$ be a segment on path P, such that $i \leq h$ and $l \leq k$ and such that all events in the segment path are contained either entirely in A or entirely in B. That is, path P is of the form $(e, i) \rightsquigarrow (e, h) \rightsquigarrow (e, h + l) \rightsquigarrow (e, i + k)$.

The definition of compatibility ensures that there is a path in A between two events in S, if and only if there a path in B between those two events. Therefore, segment $P' = (e, h) \rightsquigarrow (e, h + l)$ could be contained either in A or in B. If there is a path in one, there is a path in the other. We note that the longest delay path between (e, h) and $(e, h + l)$ in A has the same delay as the longest delay path between (e, h) and $(e, h + l)$ in B by condition 2 of the definition of mirror images. Then without loss of generality, we assume that the segment is contained entirely within A.

In a similar fashion we assume that all of path P is contained entirely within A. It follows trivially that $AC_{ub,C}(e) = AC_{ub,A}(e)$, and from Lemma 6.4 we have that $AC_{ub,A}(e) = AC_{ub,B}(e)$.

□

We now use this theorem to demonstrate a tree FIFO circuit [7]. The tree FIFO circuit is similar to the wagging buffer described by van Berkel [1]. This is the circuit we described in the previous chapter, and we are now ready to tackle it.

The two trees in Figure 6.13 are mirror images of each other. We synchronise at the leaf cells. For example, r_{ll} is synchronised with a'_{ll}, and so on for r_{lr} with cell rl', etc. As we did with the pipeline example earlier we reflect the delays. For example, if $r_l \rightarrow r_{ll}$ has delay bounds $[\delta, \Delta]$, then so does edge $r'_l \rightarrow r'_{ll}$.

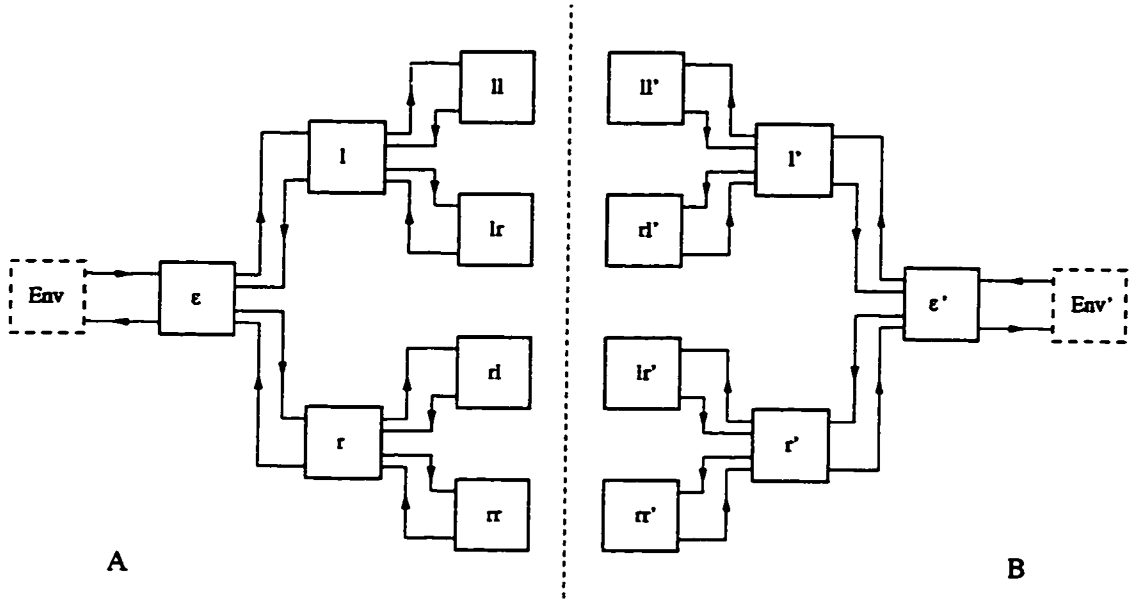


Figure 6.13: Two mirror image trees.

We combine the trees as in Figure 6.14. For a FIFO, we assume that data enters from the left-hand environment and exits via the right-hand environment. Note that when calculating cycle times, we don't actually need to differentiate the environment from other cells. The first item of data is passed to the left subtree, the second is passed to the right, and so on where left and right subtrees are defined by the labels in Figure 6.14. In the second half of the buffer, the reverse happens. Data is input first from one child and then the other. The idea is that an item of data need only travel a distance logarithmic in the capacity of the FIFO, instead of linear, thereby saving power and reducing the time a piece of data spends in the pipeline.

Now we restrict the delays to conform with Theorem 5.6.

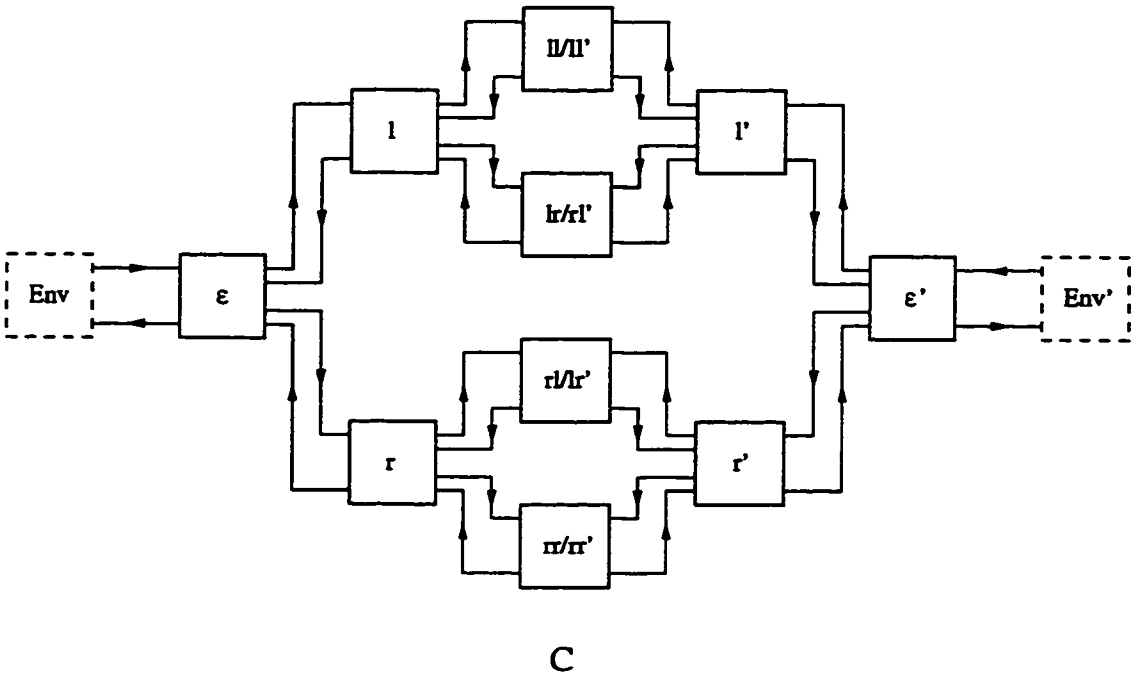


Figure 6.14: Two trees joined.

Consider the split trees again, as in Figure 6.13. Let the maximum delays for each cell in tree A be bounded as follows.

$$\begin{aligned}\Delta_{i,f} &\leq \Delta_{0,f} \cdot n^i\\ \Delta_{i,r} &\leq \Delta_{0,r} \cdot n^i\end{aligned}$$

where $0 \leq i \leq 2$ is the number of cells between the cell in question and the environment of tree A. The forward delay of the leaf cells, i.e., $\Delta_{2,f}$, is irrelevant because there is no edge that corresponds with that delay. For example, there is no $r_{ll} \rightarrow r_{lll}$ edge.

The maximum delays for each cell in tree B are bounded similarly, but, if we assume "forward" is always horizontally to the right, the forward and reverse directions are switched.

$$\begin{aligned} \Delta'_{i,r} &\leq \Delta_{0,f} \cdot n^i \\ \Delta'_{i,f} &\leq \Delta_{0,r} \cdot n^i \end{aligned}$$

where $0 \leq i \leq 2$ is the number of cells between the cell in question and the environment of tree B and where $\Delta_{0,f}$ and $\Delta_{0,r}$ are the delays of the root cell in tree A. The reverse delays of the leaf cells, i.e., $\Delta'_{2,r}$, are irrelevant in this tree. The environment delays of tree B are assumed to be the same as the environment delays of tree A. That is, $\delta'_e = \delta_e$ and $\Delta'_e = \Delta_e$.

We join the trees as in Figure 6.14. The central cells synchronise the two trees via the synchronisation set $S = \{(e, \lambda) \mid e \in \{r_{ll}/r'_{ll}, r_{lr}/r'_{rl}, r_{rl}/r'_{lr}, r_{rr}/r'_{rr}\}, \lambda \in J\}$ where J is the set of all occurrence indices. The central cells have delay bounds governed by $\Delta'_{2,f}$ and $\Delta_{2,r}$.

We have constructed the two trees so that they are "mirror images" of each other when reflected around the leaf cells. It is easy, albeit a little tedious, to prove that they are mirror images. If we pick any path $\alpha \rightsquigarrow_C \beta$ where $\alpha, \beta \in S$ we can show that there exist paths $\alpha \rightsquigarrow_A \beta$ and $\alpha \rightsquigarrow_B \beta$. Each path has an equivalent set of edges, and the same set of delay bounds on each edge. From this we can easily prove: (i) A and B are compatible, (ii) that if the delays on each path are maximised the delays on each path are the same (iii) that if the delays on each path are minimised the delays on each path are the same.

Since the trees are mirror images, we can use Theorem 6.5 to show that for any event $e \in S$.

$$AC_{ub,C}(e) = AC_{ub,A}(e) = AC_{ub,B}(e) \tag{6.13}$$

Let f be an event in A that is not necessarily in S. We wish to show that f has the same upper bound for average cycle time after composition as it did before composition. Since we can derive the ratio of events in a tree, we can derive ratio(e/f) where e is in the synchronisation set. Then we can use Theorem 6.1 to show $AC_{ub,C}(f) = AC_{ub,C}(e) \cdot \text{ratio}(e/f)$ in the composed graph and $AC_{ub,A}(f) = AC_{ub,A}(e) \cdot \text{ratio}(e/f)$ in the original graph. Note that ratio(e/f) is the same in both the original graph and in the composite. Note also that $AC_{ub,A}(e) = AC_{ub,C}(e)$ from Equation 6.13. Trivially, we then have $AC_{ub,A}(f) = AC_{ub,C}(f)$, and similarly we have $AC_{ub,B}(f') = AC_{ub,C}(f')$. In other words, any event in either of the original graphs has the same critical upper bound on cycle time in the composite as in the original graph.

From Theorem 5.6 we have $AC_{ub,A}(r_0) \leq \max\{\Delta_{0,r} + \Delta_e, n \cdot \Delta_{0,r} + n \cdot \Delta_{0,f}\}$ for a single tree. At the left-hand environment of the joined trees we therefore also have

$$AC_{ub,C}(r_0) \leq \max\{\Delta_{0,r} + \Delta_e, n \cdot \Delta_{0,r} + n \cdot \Delta_{0,f}\}$$

Consequently, for this FIFO example, an upper bound on average cycle time at the environment is dominated by the same values as that for a simple tree.

6.3.8 Periodicity and Single Synchronisation Points

As a last special case, we connect components at one synchronisation point. That is, we synchronise at a set of occurrences of one event. We will demonstrate this by connecting two pipelines together as we did in Figure 6.4 and analysing the composite. If we compose two components A and B at only one synchronisation point, we introduce no new possible critical paths. That is, we need only examine the critical paths in A and B alone. We do not need to examine any new critical paths in C. This makes the problem tractable.

We restrict the synchronisation to one type of event, hereafter described as *periodic events.*

Definition 6.6 *Let I be the range of possible occurrence indices. Event e of a graph A is a* periodic *event iff for every $h \in I$ and $l > 0$ such that a path $P = (e,h) \rightsquigarrow (e,h+l)$ exists, there exists a path $P' = (e,h+l) \rightsquigarrow (e,h+2l)$. Moreover, the delay on path P when all edges have maximum delay is the same as delay on path P' when all edges have maximum delay. Similarly, the delay on path P when all edges have minimum delay is the same as the delay on path P' when all edges have minimum delay.*

□

In other words, a periodic event is one that occurs at regular intervals in a graph, and for each periodic event, the path between two successive occurrences has the same maximum and minimum delay. For example, suppose we are given a behaviour $*[e; S_0]$. That is, event e occurs, followed by some behaviour S_0 and then the behaviour repeats. As long as the delay in S_0 is deterministic, then e is periodic. Consider in contrast the behaviour $*[e; S_0; e; S_1]$. That is, event e occurs, then some behaviour S_0, then e again, then some behaviour S_1, and then the behaviour repeats. Periodicity stipulates that the best-case and worst-case path delays for paths between occurrences of e is the same if the offset in occurrences is the same. Unless S_0 and S_1 have identical delay bounds, e is *not* periodic. Lastly, consider behaviour $*[e; S_0; e'; S_1]$. Both event e and event e' are periodic.

Note that all the vertices of a folded dependency graph are periodic events as long as each vertex has a unique label. Our model of periodic events is similar to that of *pseudorepetitive behaviour* given in Burns's thesis [10]. Pseudorepetitive behaviour consists of an initial behaviour followed by a repetitive behaviour that can be described by a folded dependency graph. All events in the repetitive behaviour would be periodic in our model. Our model of periodicity is more general in two ways.

- We describe periodicity of an event. It need not be true that every event in a repetitive behaviour is periodic.
- We avoid problems with folded dependency graphs as described in Section 2.4. That is, we can deal with cases such as pipelines with multiplicity $n > 1$. In this case, all events are periodic, but events at the environment have a different period from those at the end cell. The above definition of periodicity handles this.

We give the following lemma, that relates periodic events to the average cycle time bounds. This lemma allows us to bound the cycle time on *finite* paths.

Lemma 6.7 *Let A represent a behaviour in which e is a* periodic *event. Let $AC_{ub,A}(e)$ and $AC_{lb,A}(e)$ be the critical upper and lower bound respectively on the cycle time of e in A.*

If the path $(e,i) \rightsquigarrow (e,i+k)$ exists, where $k > 0$, $i \in I$ and I is the range of occurrence indices, then the critical upper and lower bounds on the cycle time of e satisfy

$$AC_{ub,A}(e) \geq \frac{max.\ delay((e,i) \rightsquigarrow_A (e,i+k))}{k}$$

and

$$AC_{lb,A}(e) \geq \frac{min.\ delay((e,i) \rightsquigarrow_A (e,i+k))}{k}$$

Proof. Let us prove the bound on $AC_{ub,A}(e)$ first. We note that if there exists a path $(e,i) \rightsquigarrow (e,i+k)$ there also exists a path $(e,i+k) \rightsquigarrow (e,i+2k)$ by the definition of periodic events. Inductively there exists a path $(e,i) \rightsquigarrow (e,i+x\cdot k)$ for $x \geq 0$ that has delay $x \cdot \{\text{max. delay } ((e,i) \rightsquigarrow (e,i+k))\}$. The maximum time separation $T(e,i+x\cdot k) - T(e,i)$ must be at least the delay on path $(e,i) \rightsquigarrow (e,i+x\cdot k)$.

$$\begin{aligned} AC_{ub,A}(e) &= \lim_{x\to\infty} \frac{\text{max. delay}\{(e,i) \rightsquigarrow_A (e,i+x\cdot k)\}}{x\cdot k} \\ &\geq \lim_{x\to\infty} \frac{x\cdot(\text{max. delay } \{(e,i) \rightsquigarrow_A (e,i+k)\})}{x\cdot k} \\ &= \frac{\text{max. delay } \{(e,i) \rightsquigarrow_A (e,i+k)\}}{k} \end{aligned}$$

We can do the same proof for the lower bound.

□

Suppose we have two components such that we know the critical bounds on the average cycle time of the events in each. If we compose those two components using only one synchronisation point, we would like to determine the bounds on the average cycle time of the events in the composite. For example, consider the pipelines of Figures 6.4 and 6.5. Remember that we are composing a pipeline of length $L_A = 3$ with a pipeline of length $L_B = 2$ to form a pipeline of length $L = 6$. From Figure 6.4 it is clear that we only need one synchronisation point, as we can synchronise the pipeline with only one C-element. This C-element synchronises r_3 in A and a'_0 in B. We might ask what the bounds on the average cycle time of event r_3/a'_0 would be.

We have the following theorem.

Theorem 6.8 *Let two process graphs A and B be compatible graphs with synchronisation set $S = \{(e, \lambda_i) \mid i \geq 0\}$, such that the only elements of S are occurrences of a* single *event e that is* periodic *in both A and B.*

Let C be the composite graph of A and B.

Let $AC_{ub,A}(e)$ be the critical upper bound on the average cycle time of e in original graph A.

Let $AC_{ub,B}(e)$ be the critical upper bound on the average cycle time of e in original graph B.

Let $AC_{ub,C}(e)$ be the critical upper bound on the average cycle time of e in composite graph C.

Then

$$AC_{ub,C}(e) = \max\{AC_{ub,A}(e), AC_{ub,B}(e)\}$$

Using similar definitions for lower bounds, we also have

$$AC_{lb,C}(e) = \max\{AC_{lb,A}(e), AC_{lb,B}(e)\}$$

Proof. From Theorem 6.2 we have that $AC_{ub,C}(e) \geq \max\{AC_{ub,A}(e), AC_{ub,B}(e)\}$. We prove that $AC_{ub,C}(e) \leq \max\{AC_{ub,A}(e), AC_{ub,B}(e)\}$.

Without loss of generality, we will assume that $AC_{ub,A}(e) \geq AC_{ub,B}(e)$.

Let (e, i) and $(e, i+k)$ be occurrences of e in C where $k \to \infty$. Let all edge delays be maximised. By the definition of critical bounds contained in Section 4.2, we have

$$\lim_{k \to \infty} \frac{\text{delay}\{(e,i) \rightsquigarrow_C (e,i+k)\}}{k} = AC_{ub,C}$$

Let path P be the critical path between (e,i) and $(e,i+k)$. Let us break P into consecutive segments of the form $P_j = (e,h_j) \leadsto_A (e,h_j+l_j)$ or of the form $P_j = (e,h_j) \leadsto_B (e,h_j+l_j)$. That is each P_j is a segment starting at event e and ending at event e and consisting of events and edges in A only or in B only. Note that there may be an infinite number of segments P_j or there may be a finite number of segments that have infinite length.

From periodicity, and Lemma 6.7, we have $\text{delay}(P_j) \le l_j \cdot AC_{ub,A}(e)$ if P_j is a path in A and $\text{delay}(P_j) \le l_j \cdot AC_{ub,B}(e)$ if P_j is in B. Because we have $AC_{ub,A}(e) \ge AC_{ub,B}(e)$, we then have
$\text{delay}(P_j) \le l_j \cdot AC_{ub,A}(e)$ for both types of segment.

Then

$$\begin{aligned}
AC_{ub,C}(e) &= \lim_{k\to\infty} \frac{\text{delay}\{(e,i) \leadsto_C (e,i+k)\}}{k} \\
&= \frac{\sum_j \text{delay}(P_j)}{\sum_j l_j} \\
&\qquad \{\text{Using Theorem 6.7 and } AC_{ub,A}(e) \ge AC_{ub,B}(e)\} \\
&\le \frac{\sum_j l_j \cdot AC_{ub,A}(e)}{\sum_j l_j} \\
&= AC_{ub,A}(e) \cdot \frac{\sum_j l_j}{\sum_j l_j} \\
&= AC_{ub,A}(e)
\end{aligned}$$

From the assumption that $AC_{ub,A}(e) \ge AC_{ub,B}(e)$ we have that $AC_{ub,C}(e) \le \max\{AC_{ub,A}(e), AC_{ub,B}(e)\}$ as desired.

We can follow a similar proof to show $AC_{lb,C}(e) = \max\{AC_{lb,A}(e), AC_{lb,B}(e)\}$.

□

Note that this theorem does not require every occurrence of the periodic event e to be in the synchronisation set S, just that e is periodic in both A and in B.

The set $\{\lambda_i \mid i > 0\}$ is simply a subset of the possible occurrence indices.

Let us return to our example of Figure 6.4. We synchronise the two pipelines at all occurrences of $e = r_3 = a'_0$. The cycle time of r_0 in pipeline A is critically bounded as $AC_{lb,A}(r_0) = mc_{\delta_e,A}$ and $AC_{ub,A}(r_0) = MC_{\Delta_e,A}$ where

$$mc_{\delta_e,A} = \max\{\delta_e + \delta_{0,r}, \delta_{0,f} + \delta_{1,r}, \delta_{1,f} + \delta_{2,r}, \delta_{2,f} + \delta_{3,r}\}$$

and

$$MC_{\Delta_e,A} = \max\{\Delta_e + \Delta_{0,r}, \Delta_{0,f} + \Delta_{1,r}, \Delta_{1,f} + \Delta_{2,r}, \Delta_{2,f} + \Delta_{3,r}\}$$

From Theorem 6.1 and since ratio$(r_0/r_3) = 1$, we have that $AC_{lb,A}(r_3) = mc_{\delta_e,A}$ and $AC_{ub,A}(r_3) = MC_{\Delta_e,A}$. Similarly, we can show that $AC_{lb,B}(a'_0) = mc_{\delta_e,B}$ and $AC_{ub,B}(a'_0) = MC_{\Delta_e,B}$.

Using Theorem 6.8 above, and synchronising at the single event r_3/a'_0 we have.

$$AC_{lb,C}(r_3/a'_0) = \max\{mc_{\delta_e,A}, mc_{\delta_e,B}\} \tag{6.14}$$

$$AC_{ub,C}(r_3/a'_0) = \max\{MC_{\Delta_e,A}, MC_{\Delta_e,B}\} \tag{6.15}$$

Observe that

$$\begin{aligned} mc_{\delta_e,C} &= \max(\{\delta_e + \delta_{0,r}, \delta_{0,f} + \delta_{1,r}, \delta_{1,f} + \delta_{2,r}, \delta_{2,f} + \delta_{3,r}\} \cup \\ &\quad \{\delta'_e + \delta'_{0,r}, \delta'_{0,f} + \delta'_{1,r}, \delta'_{1,f} + \delta'_{2,r}\}) \\ &= \max\{\delta_e + \delta_{0,r}, \delta_{0,f} + \delta_{1,r}, \delta_{1,f} + \delta_{2,r}, \delta_{2,f} + \delta_{3,r}, \\ &\quad \delta'_e + \delta'_{0,r}, \delta'_{0,f} + \delta'_{1,r}, \delta'_{1,f} + \delta'_{2,r}\} \end{aligned}$$

Therefore $\max\{mc_{\delta_e,A}, mc_{\delta_e,B}\} = mc_{\delta_e,C}$ and similarly we can show

$\max\{MC_{\Delta_e,A}, MC_{\Delta_e,B}\} = MC_{\Delta_e,C}$. Consequently, Equation 6.14 becomes

$$AC_{lb,C}(r_3/a_0') = mc_{\delta_e,C}$$

and Equation 6.15 becomes

$$AC_{ub,C}(r_3/a_0') = MC_{\Delta_e,C}$$

which match the calculations we made in Chapter 4.

6.3.9 Parallel Pipelines

We can create a more interesting example of composition by considering architectures with parallel pipelines. For example, consider a system where input is split into parts which are then processed in parallel. Each part is processed independently of and concurrently with the other parts. All of the behaviours are synchronised so that they start at the same time on each iteration.

Let us consider a twin pipeline of the form of Figure 6.15. This can be constructed using our methods by synchronising two pipelines as in Figure 6.16. The difference between this example and the example in the previous section is that we synchronise two acknowledgments rather than synchronising a request and an acknowledgment. That is, we synchronise a_0 and a_0'.

For simplicity, assume that both pipelines have multiplicity factor of $n = 1$. We synchronise the two pipelines at all occurrences of $e = a_0 = a_0'$. From Theorem 4.4 the cycle time of r_0 in pipeline A is critically bounded as $AC_{lb,A}(r_0) = mc_{\delta_e,A}$ and $AC_{ub,A}(r_0) = MC_{\Delta_e,A}$. Since we have $\text{ratio}(r_0/a_0) = 1$, we can show that

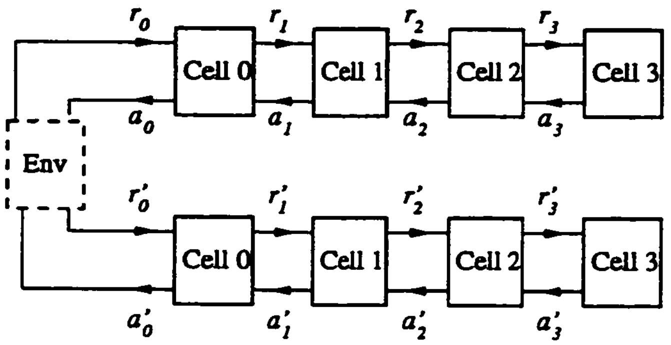


Figure 6.15: Two pipelines joined to form parallel pipelines. The requests from the environment are forked to both pipelines. The acknowledgments to the environment are synchronised.

$AC_{lb,A}(a_0) = AC_{lb,A}(r_0)$ by using Theorem 6.1. Therefore we can show

$$\begin{aligned} AC_{lb,A}(a_0) &= mc_{\delta_e,A} \\ AC_{ub,A}(a_0) &= MC_{\Delta_e,A} \end{aligned}$$

and similarly

$$\begin{aligned} AC_{lb,B}(a'_0) &= mc_{\delta_e,B} \\ AC_{ub,B}(a'_0) &= MC_{\Delta_e,B} \end{aligned}$$

for pipeline B.

Using Theorem 6.8, and synchronising at the single event a_0/a'_0 we have.

$$\begin{aligned} AC_{lb,C}(a_0/a'_0) &= \max\{mc_{\delta_e,A}, mc_{\delta_e,B}\} \\ AC_{ub,C}(a_0/a'_0) &= \max\{MC_{\Delta_e,A}, MC_{\Delta_e,B}\} \end{aligned}$$

We could easily extend the results to fatter pipelines simply by synchronising more pipelines. For example, suppose we synchronise four pipelines A, B, C and D to produce a composite parallel pipe E. We could then calculate

$$\begin{aligned} AC_{lb,B}(a_0/a_0'/a_0''/a_0''') &= \max\{mc_{\delta_e,A}, mc_{\delta_e,B}, mc_{\delta_e,C}, mc_{\delta_e,D}\} \\ AC_{ub,B}(a_0/a_0'/a_0''/a_0''') &= \max\{MC_{\Delta_e,A}, MC_{\Delta_e,B}, MC_{\Delta_e,C}, MC_{\Delta_e,D}\} \end{aligned}$$

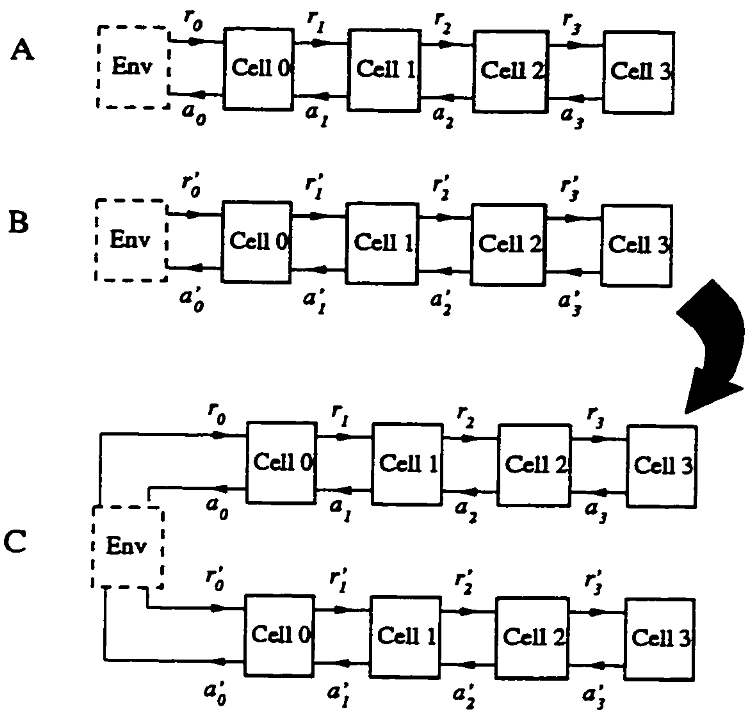


Figure 6.16: Pipelines A and B are joined by synchronising events a_0 and a_0'.

Note that we have connected two similar types of pipeline. Instead, we could have connected a pipeline with multiplication factor $n = 1$ to a pipeline with multiplication factor $n > 1$, to a pipeline with conditional behaviour, or to a tree, and then apply Theorem 6.8 above. In general, we can connect any two architectures at one synchronisation point, as long as the synchronisation events are periodic.

6.4 Summary

In this chapter we have:

- Generalised average cycle time to apply to any event. By using the concept of ratio of events, we can compute the bounds on the average cycle time of one event from the average cycle time of another event.
- We give a method of synchronising two behaviours by "fusing" events together. Given the bounds on the average cycle time of events in the original behaviours we can then calculate bounds on the average cycle time of the events in the composite behaviour. We find tight bounds for two special cases: the two composed behaviours are "mirror images" of each other, and when the two composed behaviours are joined at only one place.

Chapter 7

Conclusions and Future Work

7.1 Summary of Results

We have given closed-form formulae for the timing behaviour of several asynchronous system architectures. We looked at four basic timing measures:

- worst-case response time
- worst-case cycle time
- average response time
- average cycle time

We have given results for pipelines with various handshaking behaviours and extended these results to trees. Furthermore, we have extended bounds on average cycle time to more general architectures. These formulae allow us to experiment with a number of parameters easily and quickly and, therefore, experiment with a number of designs.

Perhaps the greatest contribution of this thesis is the insight these formulae give into the timing behaviour of various architectures. The formulae give us an understanding of how best to optimise a design, and what the various tradeoffs can be. For example, we have shown that asynchronous pipelines may be made arbitrarily (though finitely) large without affecting the bounds on the average response time or average cycle time. As another example, we have shown that the response time and cycle time of tree architectures may not need to be affected by layout problems as they seem to be, and we can therefore take advantage of their potential for low energy consumption.

We can speak with confidence about our results, because we have proofs. Since simulations often only use a (random) sample of the possible inputs, they may only give us a sample of possible outputs. Exhaustive simulation is usually computationally infeasible unless large amounts of detail are discarded from the model. Our proofs exploit the regularity of the architectures investigated.

When calculating bounds on the average-case measures we have shown that we can greatly simplify the computations by reducing our variable delay model to a fixed delay model. We have shown that we can do this without affecting the results. It is much harder to calculate worst-case response time and worst-case cycle time than it is to calculate bounds on average-case measures, because we still need the variable delay model. That is, when we calculate worst-case response time, we find the maximum event separation between a request and an acknowledgment. We make the request happen as early as possible and the acknowledgment happen as late as possible. This results in a set of paths to the request that have minimum delays and a set of paths to the acknowledgment that have maximum delays.

We have given a general definition of cycle time that applies to any event in a system. The notion of "ratio" of events allows us to calculate average cycle time bounds of any event in a deterministic system. That is, we can calculate the bounds on the average cycle time of an event if we know the bounds on the average cycle time of another event and we know the ratio of the two events.

We have given a method of composing general behaviours given by arbitrary graphs. We have given bounds on the average cycle time of the composite behaviour. In particular, we have given tight bounds for two special cases, viz., components synchronised at one point, and components that have so-called mirror-image behaviour. This greatly extends the number of architectures to which we can apply our formulae to. In this thesis, for example, we have considered a buffer based on the connection of two trees and a pipeline constructed from shorter pipelines. We can also connect pipelines to other pipelines with different handshaking behaviours or pipelines to trees.

We have given examples of a micropipeline and an eager stack and determined bounds on the worst-case response time and average-case response time. We have also calculated bounds on the worst-case cycle time and average-case cycle time for these examples. Lastly we have used our formulae to calculate the bounds on the average cycle time of a tree buffer.

7.2 Future Work

The most obvious limitation of this thesis is the restriction of architectures. One set of architectures that is not addressed is the set of array-based architectures. It is likely that the proofs in this thesis could be adapted to rectangular arrays of cells. Each column of a rectangular array could be viewed as a linear pipeline and maybe one could use the techniques of Chapter 6 to join columns of cells into an array. A ring is another interesting architecture. A ring is like a pipeline, where the outputs are fed back to the inputs, so any proof would have to worry about critical paths doubling back on themselves in a ring. We may be able to analyse a ring by unrolling it into a pipeline. Since the results for average cycle time are independent of the pipeline length, given a finite length pipeline, it may be possible to adapt the proofs to rings. That is, the results for average cycle time and average response time may apply to rings.

Chapter 6 briefly delved into the problem of joining components when the components are described by arbitrary graphs. We have only dealt with the simplest of cases. We could extend the theory by allowing two synchronisation points between components. This would allow us to model a more realistic environment. We only deal with one environment for the pipelines and trees in this thesis. In general a circuit might have one environment supplying input data and another environment receiving the output data. For example, FIFOs get input data from one part of their environment and output it to another. The end cell in the pipelines in this thesis can effectively function as a second environment, but this assumes that this second environment had no interaction with the original environment other than through the component. With two synchronisation points, we could model both environments as one component. Our main circuit component would then synchronise with the environment at two places. For example, a FIFO would synchronise with the environment where it input data and it would synchronise with the environment where it output data.

While it would be nice to allow two synchronisation points, this opens a can of worms. We can use our formulae to tell us how long the critical path in each subcomponent is. When there is only one synchronisation point, we do not introduce any new critical paths, and we can use our previous results. As we saw from the example in Section 6.3.6, when we have two synchronisation points we can introduce new critical paths. The new critical path may weave in and out of subcomponents. To compose arbitrary graphs we would need to know the critical paths between every two synchronisation points, something that our current formulae do not tell us.

We have considered only two-phase handshaking. That is, a handshake is complete after one request and its acknowledgment. In this case, voltage transitions are important, but not the direction of the transition. In contrast, consider four-phase handshaking. Here, we discriminate between a rising transition and a falling transition. As with alternate architectures, we can redraw our behavioural graphs and apply our proof techniques. There is likely to be a large commonality between results for two-phase and four-phase handshaking.

It would be nice to find further applications. Digital Signal Processors frequently use pipelines, though there is often a feedback loop between stages. Asynchronous arithmetic circuits typically use rings. There are other FIFO implementations that use a hybrid of architectures described in this thesis. For example, one can describe FIFOs that are linear pipelines bent into a U shape to become a $2 \times L$ rectangular array. Another example is that of FIFOs that split into multiple pipelines.

Lastly, it would be nice to reduce the complexity of the proofs, perhaps avoiding inductions. Two important simplifying steps have been made already in this thesis. The first is defining the relationship between average cycle time and average response time, which allows us to derive one result from the other easily. The second is simplifying the delay model to fixed delays for average case analysis. Not only does this make proofs much simpler to construct, but it often allows us to use the same proof for upper and lower bounds.

Appendix A

Trace Descriptions

Behaviour can be described as a sequence of atomic actions or events. For example we can describe the behaviour of a wire with input a and output b as

$ababababababa....$

We can more concisely define such sequences of actions by means of a "program". We can represent the behaviour of a wire component as

$*[\, a \,;\, b \,]$

where we use a sequence operator ";" and a repetition operator "*[ ]". The repetition operator allows the enclosed sequence to be repeated any number of times. We represent a fixed finite number of repetitions by means of a superscript. For example,

$(\, a \,;\, b \,)^3$

produces the trace *ababab*.

The operator "$||$" represents parallel composition. For example, the behaviour

$*[\,(a||b); c\,]$

represents a behaviour whereby events a and b may occur "concurrently" followed by event c. This behaviour may be repeated any number of times. This behaviour might generate a trace of actions such as *abcabcabcabc...* or *bacba....* or *abcbacbacabc.....* Parallel composition is described by an interleaving semantics. In the above example, a and b are atomic actions and cannot occur at exactly the same time. They can, however, occur in either order, viz., *ab* or *ba*. As another example, consider a behaviour

$((a; b)||c)$

The following traces are possible: *abc*, *acb* and *cab*.

We can describe conditional behaviour using a boolean guard B. The value of the guard B is determined by the data values that have been communicated. Since our notation abstracts from the actual data communications, the value of guard B can be considered to be non-deterministically true or false. We can then specify alternate behaviour dependent on the value of the guard. For example,

$*[\,a\,;\ \textbf{if}\ B\ \textbf{then}\ b\ \textbf{else}\ c\,]$

describes a behaviour *ab* or *ac* depending on the value of the guard B. The behaviour then repeats; the guard is checked again on each iteration as its value may have changed. A trace produced by this behaviour might be *ababacabac.....*

Lastly, some behavioural descriptions have a notation that indicates whether an action is an input or an output. In our example component of the wire we could have

$$*[\ a?\ ;\ b!\ ]$$

where ? denotes an input and ! denotes an output.

Appendix B

Symbols

Table B.1 describes the parameters and symbols used in the delay and behavioural models. It also describes abbreviations used for the timing measures. The chapter most relevant to the parameter or abbreviation is noted at the right.

Symbol	Usage	Chapter
n	Handshake multiplication factor. ($n = 1$ is typical of micropipelines)	2
$\delta_{h,r}$	Lower bound on "reverse" latency of cell h	
$\Delta_{h,r}$	Upper bound on "reverse" latency of cell h	
$\delta_{h,f}$	Lower bound on "forward" latency of cell h	
$\Delta_{h,f}$	Upper bound on "forward" latency of cell h	
δ_e	Lower bound on the environment delay	
Δ_e	Upper bound on the environment delay	
L	Number of non-end cells and index of end cell	
mc_{Δ_e}	Minimum cycle time between cells (slow environment) $mc_{\Delta_e} = \max\{\Delta_e + \delta_{0,r}, \delta_{0,f} + \delta_{1,r}, \delta_{1,f} + \delta_{2,r}, \ldots, \delta_{L-1,f} + \delta_{L,r}\}$	
MC_{δ_e}	Maximum cycle time between cells (fast environment) $MC_{\delta_e} = \max\{\delta_e + \Delta_{0,r}, \Delta_{0,f} + \Delta_{1,r}, \Delta_{1,f} + \Delta_{2,r}, \ldots, \Delta_{L-1,f} + \Delta_{L,r}\}$	
mc_{δ_e}	Minimum cycle time between cells (fast environment) $mc_{\delta_e} = \max\{\delta_e + \delta_{0,r}, \delta_{0,f} + \delta_{1,r}, \delta_{1,f} + \delta_{2,r}, \ldots, \delta_{L-1,f} + \delta_{L,r}\}$	
MC_{Δ_e}	Maximum cycle time between cells (slow environment) $MC_{\Delta_e} = \max\{\Delta_e + \Delta_{0,r}, \Delta_{0,f} + \Delta_{1,r}, \Delta_{1,f} + \Delta_{2,r}, \ldots, \Delta_{L-1,f} + \Delta_{L,r}\}$	
RT	Response time	3
CT	Cycle time	
AR	Average response time	4
AR_{lb}	Critical lower bound on average response time	
AR_{ub}	Critical upper bound on average response time	
AC	Average cycle time	
AC_{lb}	Critical lower bound on average cycle time	
AC_{ub}	Critical upper bound on average cycle time	

Table B.1: Summary of symbols used in this thesis.

Appendix C

C-elements

The "Muller C-element" is a component used to synchronise voltage transitions in an asynchronous circuit. See Figure C.1. Informally, if a voltage transition occurs on both inputs, a voltage transition is produced at the output. If only one input changes, then no change is produced at the output.

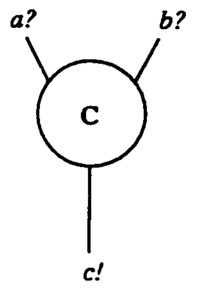


Figure C.1: A C-element with inputs $a?$ and $b?$ and where $c!$ is the output.

The behaviour of a C-element can be described as

$*[\,(a?\|b?);c!\,]$

That is, output c is produced after inputs a and b have occurred. Inputs a and b

may occur in any order. The behaviour then repeats; the C-element waits again for inputs to occur on both a and b before producing an output c.

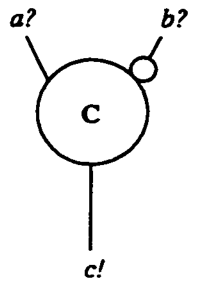


Figure C.2: An initialised C-element with inputs $a?$ and $b?$ and where $c!$ is the output. Input $b?$ is the initialised input.

Often one input of a C-element is initialised. See Figure C.2. The behaviour of this C-element can be described by

$$(a?\,;\; c!\,;\; *[\,(a?\|b?); c!\,])$$

That is, after the first voltage transition on $a?$ output $c!$ is produced. Thereafter the behaviour is the same as a normal C-element. Informally, it is as if input $b?$ has already occurred once initially.

The behaviours above are sufficient to describe the C-elements in this thesis. A more general model of behaviour allows inputs to be withdrawn. Suppose a voltage transition on a occurs followed by a second voltage transition on a to return a to its original value before any voltage transition on b occurs. The input on a is said to be withdrawn. One can specify the behaviour of a C-element so that this behaviour is permitted and no c output is produced when an input is withdrawn. No inputs are withdrawn in the circuits in this thesis.

Bibliography

[1] C. H. K. v. Berkel, C. Niessen, M. Rem, and R. W. J. J. Saeijs. VLSI programming and silicon compilation. In *Proc. International Conf. Computer Design (ICCD)*, pages 150–166, Rye Brook, New York, 1988. IEEE Computer Society Press.

[2] K. v. Berkel. *Handshake Circuits: An Intermediary between Communicating Processes and VLSI.* PhD thesis, Eindhoven University of Technology, 1992.

[3] K. v. Berkel. VLSI programming of a modulo-N counter with constant response time and constant power. In S. Furber and M. Edwards, editors, *Asynchronous Design Methodologies*, volume A-28 of *IFIP Transactions*, pages 1–11. Elsevier Science Publishers, 1993.

[4] K. v. Berkel and M. Rem. VLSI programming of asynchronous circuits for low power. In G. Birtwistle and A. Davis, editors, *Asynchronous Digital Circuit Design*, Workshops in Computing, pages 152–210. Springer-Verlag, 1995.

[5] M. Berkelaar. Statistical delay calculation, a linear method. In *Proc. ACM/IEEE International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems (TAU '97)*, Dec. 1997. Participant's proceedings.

[6] R. Berks and J. Ebergen. Response time properties of linear pipelines with varying cell delays. In *Proc. International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems (TAU)*, Dec. 1997.

[7] E. Brunvand. Low latency self-timed flow-through FIFOs. In W. J. Dally, J. W. Poulton, and A. T. Ishii, editors, *Advanced Research in VLSI*, pages 76–90. IEEE Computer Society Press, 1995.

[8] E. Brunvand and R. F. Sproull. Translating concurrent programs into delay-insensitive circuits. In *Proc. International Conf. Computer-Aided Design (ICCAD)*, pages 262–265. IEEE Computer Society Press, Nov. 1989.

[9] J. A. Brzozowski and C.-J. H. Seger. *Asynchronous Circuits.* Springer-Verlag, 1995.

[10] S. M. Burns. *Performance Analysis and Optimization of Asynchronous Circuits.* PhD thesis, California Institute of Technology, 1991.

[11] S. M. Burns and A. J. Martin. Performance analysis and optimization of asynchronous circuits. In *Advanced Research in VLSI*, pages 71–86. MIT Press, 1991.

[12] S. Chakraborty and D. L. Dill. Approximate algorithms for time separation of events. In *Proc. International Conf. Computer-Aided Design (ICCAD).* IEEE Computer Society Press, 1997.

[13] S. Chakraborty, K. Y. Yun, and D. L. Dill. Practical timing analysis of asynchronous systems using time separation of events. In *Proc. International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems (TAU)*, Dec. 1997.

[14] G. B. Dantzig, W. O. Blattner, and M. R. Rao. Finding a cycle in a graph with minimum cost to time ratio with application to a ship routing problem. In *Theory of graphs: International Symposium*, pages 77–83, 1966.

[15] A. Davis and S. M. Nowick. Asynchronous circuit design: Motivation, background, and methods. In G. Birtwistle and A. Davis, editors, *Asynchronous Digital Circuit Design*, Workshops in Computing, pages 1–49. Springer-Verlag, 1995.

[16] J. Ebergen and R. Berks. Response time properties of some asynchronous circuits. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 76–19. IEEE Computer Society Press, Apr. 1997.

[17] J. Ebergen and R. Hoogerwoord. A derivation of a serial-parallel multiplier. *Science of Computer Programming*, 15:201–215, 1990.

[18] J. C. Ebergen and A. M. G. Peeters. Design and analysis of delay-insensitive modulo-N counters. *Formal Methods in System Design*, 3(3):211–232, Dec. 1993.

[19] J. C. Ebergen, J. Segers, and I. Benko. Parallel program and asynchronous circuit design. In G. Birtwistle and A. Davis, editors, *Asynchronous Digital Circuit Design*, Workshops in Computing, pages 51–103. Springer-Verlag, 1995.

[20] M. A. Franklin and T. Pan. Clocked and asynchronous instruction pipelines. In *Proc. 26th ACM/IEEE Symp. on Microarchitecture*, pages 177–184, Austin, TX, Dec. 1993.

[21] M. A. Franklin and T. Pan. Performance comparison of asynchronous adders. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 117–125, Nov. 1994.

[22] S. Furber. Computing without clocks: Micropipelining the ARM processor. In G. Birtwistle and A. Davis, editors, *Asynchronous Digital Circuit Design*, Workshops in Computing, pages 211–262. Springer-Verlag, 1995.

[23] J. D. Garside. A CMOS VLSI implementation of an asynchronous ALU. In S. Furber and M. Edwards, editors, *Asynchronous Design Methodologies*, volume A-28 of *IFIP Transactions*, pages 181–207. Elsevier Science Publishers, 1993.

[24] M. R. Greenstreet and K. Steiglitz. Bubbles can make self-timed pipelines fast. *Journal of VLSI Signal Processing*, 2(3):139–148, Nov. 1990.

[25] H. Hulgaard. *Timing Analysis and Verification of Timed Asynchronous Circuits*. PhD thesis, Department of Computer Science, University of Washington, 1995.

[26] H. Hulgaard, S. Burns, T. Amon, and G. Borriello. An algorithm for exact bounds on the time separation of events in concurrent systems. Technical Report TR-94-02-02, University of Washington, Department of Computer Science and Engineering, Feb. 1994.

[27] H. Hulgaard and S. M. Burns. Bounded delay timing analysis of a class of CSP programs with choice. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 2–11, Nov. 1994.

[28] K. Hwang. *Computer Arithmetic: Principles, Architecture and Design*. John Wiley & Sons, 1979.

[29] M. B. Josephs, P. G. Lucassen, J. T. Udding, and T. Verhoeff. Formal design of an asynchronous DSP counterflow pipeline: A case study in handshake algebra. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 206–215, Nov. 1994.

[30] D. Kearney and N. W. Bergmann. Bundled data asynchronous multipliers with data dependant computation times. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 186–197. IEEE Computer Society Press, Apr. 1997.

[31] D. Kearney and N. W. Bermann. Performance evaluation of asynchronous logic pipelines with data dependant processing delays. In *Asynchronous Design Methodologies*, pages 4–13. IEEE Computer Society Press, May 1995.

[32] E. Lawler. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart & Winston, 1976.

[33] T. K. Lee. *A General Approach to Performance Analysis and Optimization of Asynchronous Circuits.* PhD thesis, California Institute of Technology, 1995. Technical report CS-TR-95-07.

[34] J. Liu. The design of an asynchronous multiplier. Master's thesis, University of Manchester, 1995.

[35] A. J. Martin. Programming in VLSI: From communicating processes to delay-insensitive circuits. In C. A. R. Hoare, editor, *Developments in Concurrency and Communication*, UT Year of Programming Series, pages 1–64. Addison-Wesley, 1990.

[36] A. J. Martin. Synthesis of asynchronous VLSI circuits. In J. Straunstrup, editor, *Formal Methods for VLSI Design*, chapter 6, pages 237–283. North-Holland, 1990.

[37] K. McMillan and D. Dill. Algorithms for interface timing verification. In *Proc. International Conf. Computer Design (ICCD)*. IEEE Computer Society Press, Oct. 1992.

[38] T. H.-Y. Meng, R. W. Brodersen, and D. G. Messerschmitt. Asynchronous design for programmable digital signal processors. *IEEE Transactions on Signal Processing*, 39(4):939–952, Apr. 1991.

[39] C. D. Nielsen and M. Kishinevsky. Performance analysis based on timing simulation. In *Proc. ACM/IEEE Design Automation Conference*, pages 70–76, June 1994.

[40] S. M. Nowick, K. Y. Yun, and P. A. Beerel. Speculative completion for the design of high-performance asynchronous dynamic adders. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 210–223. IEEE Computer Society Press, Apr. 1997.

[41] P. Pang and M. Greenstreet. Self-timed meshes are faster than synchronous. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 30–39. IEEE Computer Society Press, Apr. 1997.

[42] P. Patra and D. S. Fussell. Optimization of delay-insensitive circuits – a case study. Technical report, Dept. of Computer Sciences, The Univ of Texas at Austin, July 1994. Technical Report.

[43] A. M. G. Peeters. *Single-Rail Handshake Circuits*. PhD thesis, Eindhoven University of Technology, June 1996.

[44] P. J. Schweitzer and T. Altiok. Aggregate modelling of tandem queues without intermediate buffers. In H. G. Perros and T. Altiok, editors, *Queueing networks with blocking*. 1989.

[45] J. Sparsø and J. Staunstrup. Design and performance analysis of delay insensitive multi-ring structures. In *Proc. Hawaii International Conf. System Sciences*, volume I, pages 349–358. IEEE Computer Society Press, Jan. 1993.

[46] R. F. Sproull, I. E. Sutherland, and C. E. Molnar. The counterflow pipeline processor architecture. *IEEE Design & Test of Computers*, 11(3):48–59, Fall 1994.

[47] I. E. Sutherland. Micropipelines. *Communications of the ACM*, 32(6):720–738, June 1989.

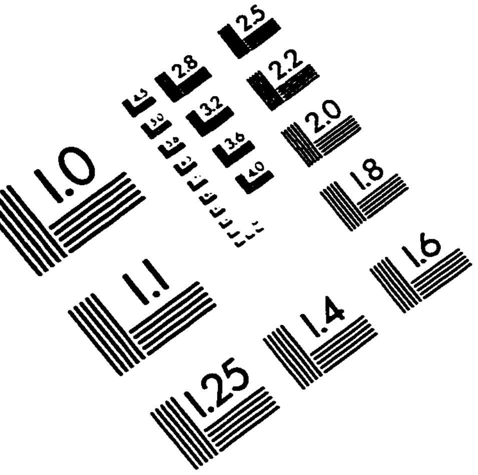
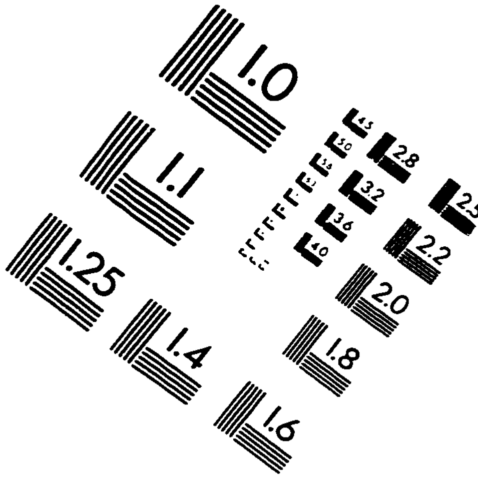
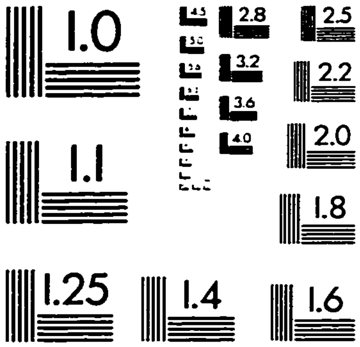
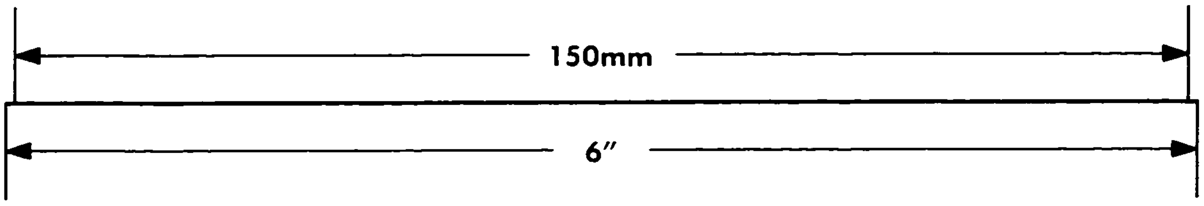
[48] E. A. Walkup and G. Borriello. Interface timing verification with application to synthesis. In *Proc. ACM/IEEE Design Automation Conference*, pages 106–112. IEEE Computer Society Press, June 1991.

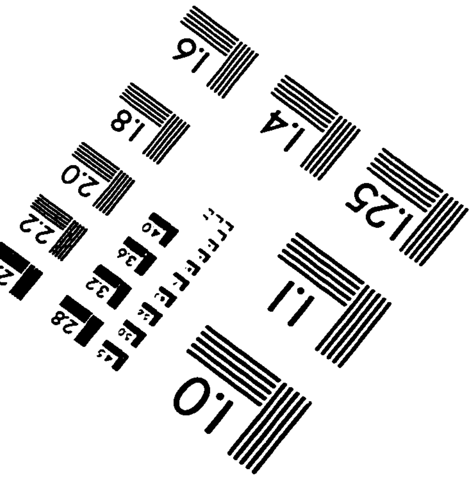
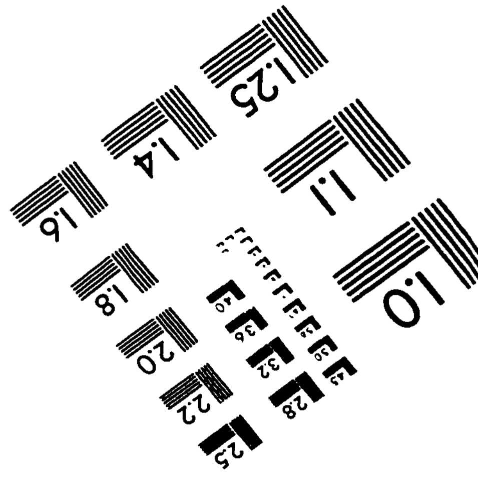
[49] T. E. Williams. Performance of iterative computation in self-timed rings. *Journal of VLSI Signal Processing*, 7(1/2):17–31, Feb. 1994.

[50] T. E. Williams and M. A. Horowitz. A zero-overhead self-timed 160ns 54b CMOS divider. *IEEE Journal of Solid-State Circuits*, 26(11):1651–1661, Nov. 1991.

[51] A. Xie and P. A. Beerel. Symbolic techniques for performance analysis of timed systems based on average time separation of events. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 64–75. IEEE Computer Society Press, Apr. 1997.

IMAGE EVALUATION
TEST TARGET (QA-3)

APPLIED IMAGE . Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989