Technical Report

UCAM-CL-TR-698
ISSN 1476-2986

Number 698

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

Pulse-based, on-chip interconnect

Simon J. Hollis

September 2007

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

http://www.cl.cam.ac.uk/

This technical report is based on a dissertation submitted June 2007 by the author for the degree of Doctor of Philosophy to the University of Cambridge, Queens' College.

Technical reports published by the University of Cambridge Computer Laboratory are freely available via the Internet:

http://www.cl.cam.ac.uk/techreports/

ISSN 1476-2986

Abstract

This thesis describes the development of an on-chip point-to-point link, with particular emphasis on the reduction of its global metal area footprint.

To reduce its metal footprint, the interconnect uses a serial transmission approach. 8-bit data is sent using just two wires, through a pulse-based technique, inspired by the *GasP* interconnect from Sun Microsystems. Data and control signals are transmitted bi-directionally on a wire using this double-edged, pulse-based signalling protocol, and formatted using a variant of dual-rail encoding. These choices enable a reduction in the number of wires needed, an improvement in the acknowledgement overhead of the asynchronous protocol, and the ability to cross clock domains without synchronisation hazards.

New, stateful, repeaters are demonstrated, and results from *spice* simulations of the system show that data can be transferred at over 1Gbit/s, over 1mm of minimum-sized, minimally-spaced metal 5 wiring, on a 180nm (0.18μm) technology. This reduces to only 926Mbit/s, when 10mm of wiring is considered, and represents a channel utilisation of a very attractive 45% of theoretical capacity at this length. Analysis of latencies, energy consumption, and area use are also provided.

The point-to-point link is then expanded with the invention and demonstration of a router and an arbitrated merge element, to produce a Network-on-Chip (NoC) design, called *RasP*. The full system is then evaluated, and peak throughput is shown to be 763Mbit/s for 1mm of wiring, reducing to 599Mbit/s for 10mm of the narrow metal 5 interconnect.

Finally, RasP is compared in performance with the *Chain* interconnect from the University of Manchester. Results for the metrics of throughput, latency, energy consumption and area footprint show that the two systems perform very similarly — the maximum absolute deviation is under 25% for throughput, latency and area; and the energy-efficiency of RasP is approximately twice that of Chain. Between the two systems, RasP has the smaller latency, energy and area requirements and is shown to be a viable alternative NoC design.

Acknowledgements

This thesis is dedicated to my parents, without whose help and belief it would have never been written. They have always encouraged me to do what I enjoy, whatever the cost. It is with this credo that I have found my niche in research. I will always thank them.

My eternal gratitude goes to my supervisor, Simon Moore, without whose expertise and help, I would have never got this far. Over the years, Simon has not only provided me with advice and funding, but has never complained when I wanted to go off on a tangent. This thesis is, I hope, proof of the success of such an approach; thank you.

John Whitington and Daniel Greenfield deserve special mention for proof-reading this mighty document, and providing a great deal of corrections (even if we did sometimes disagree about the finer points of grammar).

I also wish to thank my dear friends, Andy, Ash and Mark, who have filled my nights with experiences dear during the three and half years it has taken to prepare this dissertation.

Finally, I wish to express my appreciation to my housemates over the years — you have kept me sane (and frequently fed and watered) along the journey.

Contents

List of Tables

List of Figures

1 Introduction

There are more computer circuits on the radio of a new car than there were computer circuits on Apollo missions. These days everything is interconnected with computers. So unless you understand physics, you're not going to be able to fix a car

Mary Cantrell

Soon after the invention of the transistor, the integrated circuit was born. The cost of integration and the small number of transistors available made switching very precious. Connecting them up was a minor concern, and so the saying, "Wires are free, transistors are expensive." came into being. Now, this adage has gone full-circle to become, "Transistors are free, wires are expensive." Why is this so? In this dissertation, we will investigate the increasing importance to a chip designer of on-chip interconnect.

One of the greatest contributors to this turn-around in thinking has been Moore's Law [45], which has been a blessing for logic designers concerned with the increasingly demanding applications placed on semiconductor devices. The dramatic increase in the number of transistors available has enabled applications inconceivable by the fathers of the 'silicon revolution'.

However, all has not been smooth. By far, the greatest hurdle facing the semiconductor industry today is that of power consumption: the increasing density of devices has produced thermal fluxes approaching that of the surface of the sun [48]. Power density is now the most important factor when designing an integrated circuit, and designers are already incorporating power reduction measures such as variable-threshold gates, voltage and clock scaling and clock gating into their designs. These approaches are providing tractable solutions to the power problem.

So why is interconnect any different? Surely, once it becomes the prevalent factor in integrated design, then people will address it well? Unfortunately, interconnect is an all-together different problem. At the heart of the upcoming difficulty is the asymptotic complexity of the problem. Rent's Rule [8] predicts an exponential increase in the number of wires required as we increase the numbers of transistors in a device, and whilst other predictions are more conservative, one thing is certain: the small, incremental increases in available wires provided by additional metal layers in integrated circuits will not be able to keep up with such a blowup.

It is not only the complexity of wiring that causes headaches for device designers, we must also deal with a side-effect of Moore's Law: that devices get faster. Strictly, it is the increase in clock frequency of traditional, synchronous designs producing the problem. Combined with the slowness of the speed of light[†], this can cause problems where signals cannot physically traverse their interconnect in a single clock cycle. Most importantly, this means that it

[†]The speed of light in a vacuum, $c = 3 \times 10^8$m/s, is fast, but even light can travel only 10cm during one clock tick of a 3GHz processor. Wires on board a chip can easily extend to a significant fraction of this.

is impossible for a global clock, the cornerstone of synchronous design, to be distributed simultaneously across an entire die. This gives rise to *clock skew* [16], which has been a bane of integrated circuit designers for many years. Clock skew has no noticeable effect at a local (sub-circuit) scale, but when these are composed (by means of an interconnection), the skew becomes apparent and can cause a multitude of undesired effects (see §2.4 for more details).

These are fundamental issues, which need to be addressed. Yet, we have not even mentioned the requirement for interconnect to pass signals quickly around a chip. This goes almost without saying, since a shorter interconnect latency can lead to performance advantages by delivering data more quickly, perhaps so it may be operated on in the same clock cycle, or by decreasing the round-trip time of a protocol with acknowledgements. As an example of where this helps, consider the popular TCP protocol, where decreasing the round-trip time increases throughput for a fixed *window* size [10, pp.212–213].

All this has shown there is much work to be done on the problem of on-chip interconnect, and the remainder of this dissertation will focus on the main issues in turn. We shall take the standpoint of the desirability for small, yet flexible and highly performing interconnection systems.

These kinds of systems are ideally suited for use in custom Application Specific Integrated Circuit (ASIC) designs. However, the complexity involved when producing a modern ASIC means that designers would prefer not to have to consider the problems of interconnection, instead focusing on logical design.

Since interconnect design may well be the hardest task facing architects, the Holy Grail must surely be to produce a drop-in interconnect system for an ASIC design flow. To do this, the system should be modular, scalable and able to transfer data across differing clock domains. This latter point will enable the system to fit in easily with the common approach of connecting together a set of Intellectual Property (IP) blocks from different vendors. These often run at different clock speeds or voltages, and so make traditional communication difficult. Further, we desire high performance and energy-efficiency in any such design.

Therefore, the subject of this thesis is to produce such an interconnect system, with our target application being an 8-bit ASIC design, running synchronously at a clock rate of 66MHz. This gives us an interconnect end-to-end data rate requirement of at least 528Mbit/s, over chip-scale distances (which we will take to mean up to 1cm, the side dimension of a large modern die).

By the end of this document, our target will have been achieved; with the design of a point-to-point link capable of operating at over 1Gbit/s; and its incorporation into a Network-on-Chip (NoC) design, to produce a large scale interconnect that runs at over 700Mbit/s.

1.1 Overview of the proposed system

The problems outlined above will be tackled in two stages. First will be the creation of a pulse-based point-to-point link. Based on an asynchronous signalling protocol, this will be shown to run at speeds in excess of 1Gbit/s at distances of up to 1mm, and continue high performance operation up to 10mm of global wiring. It will, therefore, exceed our design specification.

The link will address all of the issues outlined above. To do this, it will employ a Globally Asynchronous, Locally Synchronous, or *GALS* approach. It will be based loosely on the *GasP* interconnect from Sun Microsystems [59], and a modified technique will allow it to transmit

data and control signals both *asynchronously* and *bi-directionally* using just two global metal wires; many fewer than a conventional parallel interconnect. The signalling protocol will be pulse-based, to fit in with the GasP mentality, and asynchronous to eliminate the problem of low-skew signal distribution, easily allowing clock domain crossing.

Asynchronous logic allows us to eliminate the need for a clock to be transmitted alongside our data stream. Its removal frees us to concentrate on increasing the frequency of data transmission without the worry of skew, whilst simultaneously removing the need for a clock wire. In order to interface cleanly and modularly with traditional designs, we present standard, FIFO-like, interfaces to the transmitting and receiving environments.

Second, this point-to-point link will be shown to be capable of being deployed at the heart of a NoC design. The additional elements needed will be introduced and evaluated, and a fully-functional NoC design, named *RasP*, will be demonstrated. End-to-end performance of RasP will be demonstrated to be less than that of the point-to-point link upon which it is built, but still in excess of the requirements of our ASIC application. To provide context for the results, we will compare it with the Chain interconnect system from the University of Manchester [3], and we will see that RasP provides competitive performance.

1.2 Approach

This thesis will start in Chapters 2 and 3 by introducing some basic concepts of interconnects, such as topologies and connectivities. Then, in Chapter 4, we will see the various physical limitations on interconnect, such as bandwidth limits and the impact of crosstalk, caused by the capacitive and inductive coupling between neighbouring wires. We then go on to an introduction of a potential point-to-point interconnect, and the use of repeaters to effectively cover long distances. This will be Chapter 5, and is based on an improved version of the system presented in two peer-reviewed papers, presented at the 2006 International Symposium on Circuits and Systems (ISCAS) [24], and the 19th International Conference on VLSI Design [25]. For both papers, the writer is the first author, and produced all of the content.

The system will then be evaluated in Chapter 6 with respect to our target ASIC application, and also some common interconnect performance metrics. Once this has been done, in Chapter 7 we will show how this basic link can be scaled into a Network-on-Chip application. To support this, we will design and incorporate router, arbiter and multiplexer elements. Again, this chapter is based upon a first-author paper, presented at the 24th International Conference on Computer Design (ICCD) [26]. Subsequently, in Chapter 8, there will be an evaluation of this NoC, which the author has named *RasP*, and we will provide a comparison with an implementation of a reference system called *Chain* [3]. Finally, in Chapter 9, we will draw some conclusions.

2 Interconnect design

We keep moving forward, opening new doors, and doing new things, because we're curious and curiosity keeps leading us down new paths

Walt Disney

2.1 Introduction

Approaches to interconnect design are much the same whether we deal with on-chip communication or off-chip transfers. The basic choices of parallel or serial have been around for a long time, and their respective advantages and disadvantages are well known. Traditional wisdom dictates that for high bandwidth applications, parallel interconnects are the correct approach. However, skew, power and packaging cost issues have caused a general trend towards more serial implementations.

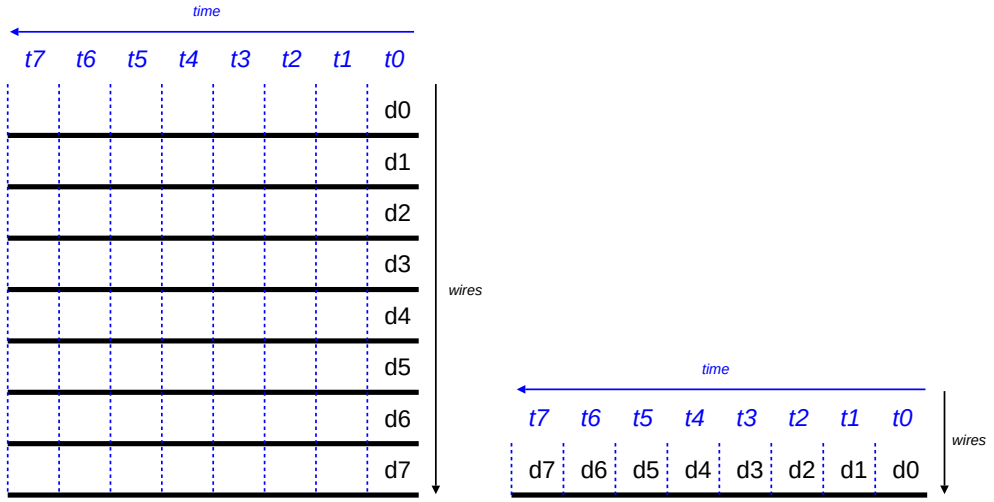
Serial interconnects are not without their disadvantages: the increase in logic complexity can cause increases in end-to-end latency, power consumption and even logic routing problems. In order to be chosen, these factors must be carefully weighed and, only then, a choice taken.

Once this broad choice of approach has been carried out, we can concern ourselves with finer details, such as the choice of signalling and coding techniques, in order to maximise performance and reliability. In the following sections, we will see how the various approaches compare, and when they should and should not be used.

2.2 Parallel interconnect design

Parallel interconnects have been with us since the dawn of computing: they are the simplest manifestation of the need to transfer multiple bits of data concurrently. Operation of a basic parallel interconnect is very simple, involving a separate wire for each bit of information to be transferred. An overview diagram is given as Figure 2.1(a), showing how an 8-bit parallel wire transfers all eight data bits `d0...d7` simultaneously in time slot $t0$. The major advantage this approach is its simplicity: to transfer a word of N bits, we just need N wires in parallel. In this way, the throughput of a parallel system can be increased by simply adding more wires. Thus, the throughput of M wires, with a base rate of B bit/s is $M \times B$.

The last statement is true in the ideal world, but it becomes incorrect when considering actual implementations, when we have to account for the interaction between wires. This interaction causes the impression of noise by switching (*active*) wires on non-switching (*passive*) wires, via capacitive and inductive coupling [33, pp.26–36]. The noise causes signal-integrity



(a) Transfers across an 8-bit parallel interconnect

(b) Transfers across an 8-bit serial interconnect

Figure 2.1: The difference between parallel and serial interconnect data transfers

problems, and the capacitive coupling causes an effective increase in wire capacitance, reducing its effective bandwidth, by increasing its time constant, from the equation $t = RC$, where t is the time constant of an RC wire, and R and C are its characteristic resistance and capacitance, respectively. This equation will be fully explained in Chapter 4.

Another problem faced by parallel interconnect is that of signal *skew*. Skew is the phenomenon of non-simultaneous arrival of signals to a common destination. Since many logical blocks require (near-)simultaneous arrival of their inputs to function correctly, skew can cause unreliable operation, and even metastability (see §2.4 for more details). Parallel wires induce relative skew when their lengths are not exactly matched [65, p.788] , and since we never get two identical wires in practice, skew will always be a problem for parallel interconnects.

Such concerns are very real for designers of on-chip interconnects, and have been the major motivating factors in the move to more serial links.

2.2.1 Performance

Parallel interconnects contain many closely spaced wires. This configuration has severe implications for capacitive and inductive coupling (see sections 4.1.2 and 4.1.3), and their respective impacts on signal integrity (Section 4.3).

Simulations of a 1000μm long, 16-bit, minimally-sized, minimally-spaced parallel bus in the capacitance extraction tool *Quickcap* [41] illustrate the high potential for this destructive effect — as shown in Table 2.1 below, the worst-case capacitive crosstalks are almost universally around 90% of `vdd`. For comparison, I show also the results from an identical bus, but where the width or spacing have been changed to twice their minimal values.

We see that increasing the width w of a wire increases its capacitance slightly, and does not decrease its inductance by an appreciable amount. Similarly, the worst-case (W/C) and RMS capacitive crosstalk (Xtlk) figures are not improved significantly. Given that the area requirements of this configuration entail an increase of 50% when including spacing, this is

Table 2.1: 16-bit, 1000μm parallel interconnect capacitances and inductances for various sizes and spacings (extracted, parallel-plane metal layers 4 & 6)

Configuration	W/C Xtlk	RMS Xtlk	C_{total}	$C_{mutual_{max}}$	L_{self}	$L_{mutual_{max}}$
Min w, min s	91%	60%	231fF	98fF	1.64nH	1.42nH
$2\times w$, min s	87%	59%	252fF	106fF	1.59nH	1.35nH
Min w, $2\times s$	79%	52%	149fF	55fF	1.63nH	1.33nH

not necessarily a trade-off we wish to make (although it may still be useful in practice since the wire's high frequency bandwidth will be increased).

Conversely, doubling the spacing s of a wire has an appreciable effect on its performance with these metrics. Capacitance is reduced by roughly one third, and inductance remains similar. The reduction in capacitance yields a drop in capacitive crosstalk of around 10%. Recalling that wire delay $t = RC$, these changes give a performance increase of about a third (directly in line with the reduction in capacitance; resistance remains fixed), plus signal integrity improvements. All the mutual inductances decrease linearly with distance from the victim wire.

2.2.2 Overheads

Parallel interconnects, which add one more wire for each extra data bit, are very area-inefficient. Their area cost grows linearly with word width. In extreme cases, an increase in word width can cause a reduction in core clock rate due to the added complexity. Then, overall throughput may static, or even decrease, despite the fact that additional area has been consumed.

When used in a clocked system, parallel interconnects may switch N wires in a clock cycle for an N bit word. This is very power inefficient, particularly if used with the popular precharging methodology [65, p.332]. In precharging's worst case (all wires `0`), a link running at a clock frequency of f will consume the energy of $f \times N$ "`0->1->0`" transitions. The energy consumption of a wire, modelling it as a pure capacitor with the following formula:

$$\Delta E = C\Delta V^2 \tag{2.1}$$

where E is energy, C is the capacitance and V is the supply voltage, `vdd`, of a capacitor. For a wire with capacitance of C Farads, and a supply voltage of V volts, this corresponds to $V^2CNf/2$ Joules every second [65, pp.190–191]. As wire length increases, so does energy consumption, and this is approximately linear with length, plus a constant term relating to the fixed energy consumption of the switching logic.

2.3 Serial interconnect design

Serial designs easily overcome the high wire overhead of their parallel cousins. Instead of many parallel wires, they take the approach of few wires, each carrying multiple bits. The style is characterised by the sequential nature of the bits, when looking at a single wire. To transfer eight bits, `d0...d7`, for example, the first, `d0` is transmitted fully, before `d1`, `d2`, and

so on. An illustration is given as Figure 2.1(b) where we see that the transfer of eight data bits `d0...d7` occurs sequentially using eight time slots $t0 \ldots t7$.

Typically, logical blocks do not understand serial data streams, but rather prefer data to be presented to them in parallel. Thus, to convert logical output data into a serial format for transmission, and back again for delivery, serial links must typically perform parallel-to-serial and then serial-to-parallel conversions. These conversions involve logical processing, and typically the latching of certain pieces of data. Naturally, this entails penalties in area, throughput, end-to-end latency and power consumption, degrading the overall performance of the link. Throughput and logical complexity can be ameliorated by pipelining the link [20, pp.A-2–A-4, A-10], although this increases the latency and power consumption. An alternative is to increase the frequency of operation. However, this is not a trivial task: with most modern systems this means increasing the frequency of a clock. This is a notoriously difficult problem, mostly due to difficulties in low-skew distribution of a high-speed signal [16].

For short links, the various overheads can mean that serial connections, whilst using fewer wires, have a larger footprint and worse performance than their more simplistic, parallel, counterparts.

One of the biggest selling points of serial interconnects is the low area footprint in global metal layers. Global metal is the most precious of a designer's routable layers, and is fast running out. Rent's rule [8], whilst a loose guideline to real events, predicts an exponential increase in the number of point-to-point connections required with shrinks in logic feature size. Compare this with the linear increase in metal layers with evolution of integrated circuits, and designers clearly have a problem. Serial approaches allow word-wide data links to be squeezed into the space of perhaps a single bit. In this way, serial links allow the evolution of technology whilst keeping the reigns on wiring complexity blowups. Further, and arguably more importantly for modern designs, the reduction in wiring area allows the use of wider wires for higher performance, or greater spacing of wires, reducing crosstalk and increasing signal integrity. Thus a narrow, serial interconnect can often provide better performance than a parallel counterpart over long runs of wire.

The reduction in the number of wires also makes it easier to equalise transmission distances, which help with skew problems (see §2.4). Serial interconnects can even use clock recovery or data detection schemes [51] in order to function independently from a global clock. This saves not only wires and power in distributing the clock, but also allows the problem of low-skew clock distribution to be removed. This approach can be considered a Globally Asynchronous, Locally Synchronous scheme, and is described fully in Section 2.4.1.

The situations where a serial or parallel choice provides optimum performance depend upon a combination of the technology used, design priorities and other factors. In the past the serial approach has not been the natural choice for high throughput requirements — rather parallel systems have been used, since one can simply increase the number of wires to add additional performance. However, at high signalling speeds this ceases to be true. Amongst the many reasons this is so is the problem of *crosstalk*. Crosstalk involves interference between neighbouring wires and can seriously degrade performance [65, 207–210],[33, 26–36] over what may naïvely be expected.

Naturally, though, the biggest perceived advantage of serial interconnection systems is its low wire count and correspondingly small area footprint. As previously mentioned, wires are critical in modern designs, and the reduction in number afforded by serial interconnect implementations can be a great boon for designers. Along with the reduction in number of

Table 2.2: 2-bit, 1000μm parallel interconnect capacitances and inductances for various sizes and spacings (parallel ground plane m4&6)

Configuration	W/C Xtlk	RMS Xtlk	C_{total}	$C_{mutual_{max}}$	L_{self}	$L_{mutual_{max}}$
Min w, min s	65%	65%	162fF	105fF	1.65nH	1.42nH
$2\times w$, min s	62%	62%	178fF	110fF	1.58nH	1.36nH
Min w, $2\times s$	42%	42%	120fF	58fF	1.66nH	1.35nH

wires come related benefits, such as a reduction in the overall power use (due in part to a reduction in drivers) leading to lower static power dissipation. Given the importance of power density in modern circuits, this can only help implementations.

Therefore, it is impossible to say definitively when one approach is superior to the other. However, most modern designs, for both on and off chip interconnect, perform some form of serialisation or packetisation. Thus, it is safe to assume that the attractiveness of serialisation increases with technology advances. Therefore, a key design choice of the interconnect we will see later in this thesis to use a serial approach.

2.3.1 Performance

Unlike its parallel counterpart, serial interconnect requires few wires. Consequently, the magnitudes of capacitive and inductive coupling are greatly reduced. As an illustration, results for crosstalk on a 1000μm dual-rail, minimally-sized, minimally-spaced interconnect are shown below. We see a reduction of over 20% of `vdd` compared with a parallel implementation.

In addition to these results, the simplicity of the dual-wire implementation makes it possible to accurately simulate the metal 4 and 6 layers with realistic wire layouts and spacings, rather than just as a ground plane. I performed this for our technology, with metals 4 and 6 minimally spaced, and varying arrangements for our metal 5 signal wires. However, for compactness, I display only the parallel-plate results as Table 2.2.

2.3.2 Overheads

In the absence of an advanced encoding scheme, the transmission of bits in a serial nature are mutually exclusive in time. This presents certain inefficiencies when using serial interconnects.

Of concern to many designers, with relation to serial interconnects is that the bits are ordered in time. In particular, all bits do not arrive simultaneously, and so the relative latency of different parts of a data word may vary (for example, Little Endian transmission will transfer the least significant bit (LSB) first [9]). This means that a data consumer may not get all the information it needs to make a computation when it needs it. For example, if the block is relying on a flag bit in the middle of a word, it will have to wait for at least half of the byte to be transferred. This is not a problem for parallel interconnects, since all bits arrive simultaneously.

Of course, this problem can be ameliorated by good interaction between designers when deciding the format of a word (e.g., always place the most critical information in the LSBs, which arrive first), but this is not always possible.

So, serial interconnects have some latency issues, and may not always be suitable for all applications. However, for the vast majority, their area-efficiency and crosstalk minimisation abilities outweigh all other considerations, and this explains their almost ubiquitous use for modern off-chip communications. This trend is also observed to be moving on-chip due to the increasing density requirements of modern integrated circuits.

2.4 Clock skew

Clock skew is simply a specialisation of the general signal skew problem, so I explain that first.

Skew is defined as the difference in time between the time a signal is expected to arrive (and wanted) at its destination in a circuit and the time it actually appears. Skew can be negative or positive or (ideally) zero. Skew exists for all real signals, but is only significant when considered relative to other signal arrival times (*relative skew*). Thus, if signal a arrives at time t_a and signal b at t_b, we say the relative skew, $\delta_{skew} = t_a - t_b$. More information can be found in Weste and Harris [65, pp. 786–789].

Low values of relative skew (i.e., those which are small compared with the logical delay of a functional block), do not generally cause any problems in a circuit. However, those which are larger can induce undesired behaviour. Ranging from additional power consumption, through glitches, to metastability on outputs, the consequences can be severe. Exactly what values of relative skew cause problems are beyond the scope of this thesis, but it is true to say that (until values approaching that of a clock cycle in a synchronous system) the higher the magnitude of the skew value, the worse a circuit fares.

Clock skew is the natural extension of relative skew between a single signal arrival time and that of the global clock signal. It is of particular concern when considering state-holding components such as latches. If data and clock inputs to such an element have a large relative skew, then it is possible for both to change simultaneously. This may violate the setup and hold times of a component, leading to unpredictable and/or incorrect operation. Therefore, skew is something circuit designers work very hard to avoid — but this is difficult to parametrise and solve when considering long distance interconnects, as we do in this thesis.

With today's high variations in CMOS processes, the necessary guarantees when designing for safe values of skew are harder to make. Uncertainties in wire thicknesses and propagation delays, as well as driver strengths and receiver thresholds, mean that two wires that appear identical on paper may end up wildly different when manufactured.

This makes skew-tolerant design even more necessary than ever, and there is an approach becoming more and more popular in order to give the necessary safety to logical blocks which are composed via long distance wiring. Called GALS, it is a popular contemporary approach, described in the following section.

2.4.1 GALS

GALS (shown as Figure 2.2) stands for 'Globally Asynchronous, Locally Synchronous', and is a paradigm for the implementation of systems composed of multiple functional blocks, where clock skew is expected across block domains [46]. Each block operates in a standard, clocked, synchronous manner, producing and consuming data on clock edges. Interconnection of blocks, however, is performed without need for clock synchrony. This allows the interconnection fabric to avoid any clock skew issues. Additionally, it allows it to operate at its

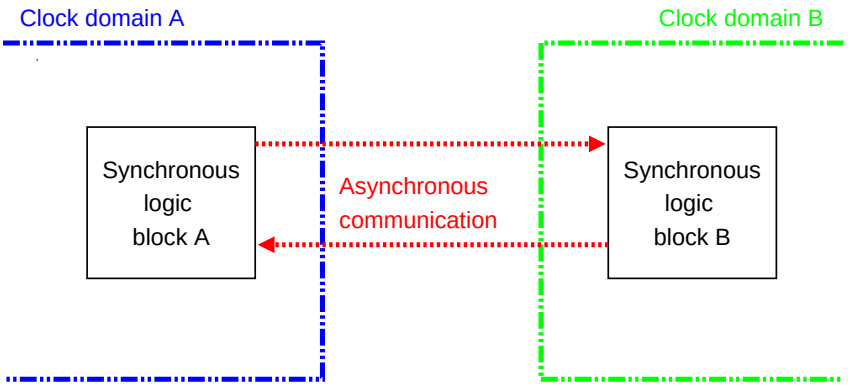


Figure 2.2: A Globally Asynchronous, Locally Synchronous (GALS) system

own frequency. This final point enables many performance enhancements, such as a higher transmission frequency than would be obtained if concerned with the global system's worst case, and also the ability to clock gate the power hungry interconnection subsystem (or an arbitrary logic block), without affecting other parts of the system. This independence of a global clock is what gives the interconnect its 'asynchronous' label.

GALS systems are finding favour with a large proportion of system designers since logical blocks remain fully synchronous, and thus are readily understood and designed using existing tools. The simplicity that the assumptions of clocked systems allow reduces development time and helps manage complexity. On the other side, not needing to guarantee clock and signal arrival times over long distances is a great relief when analysing wiring. In this way, the synchronous and asynchronous components of a GALS approach complement each other.

The final point to address then is the complexity of a full GALS system, and this varies. At its simplest, synchronisation can be added at a receiver as an after-thought, and may entail a very low logical overhead but an additional latency of one or two latch delays. In the middle of the range, *source synchronous* systems can use an asynchronous FIFO structure to decouple sender and receiver data transfers. The most complex systems are asynchronous networks and offer multipoint-to-multipoint connectivity (see Section 3.4 for an explanation) and very high throughputs. These are often referred to as Network on Chip (NoC) implementations.

So, there are a multitude of GALS systems out there, dependent on application; and the approach is set to become more applicable than ever with process variation and growing complexity.

2.5 Data encoding

Transmission efficiencies of all forms of interconnect can be increased by applying an encoding scheme to the data before transmission. Encoding can be as simple as the parallel-to-serial conversion required by serial interconnects (see §2.3), or as complicated as compression or encryption.

Most usefully for on-chip interconnect, data encoding can alter the format of data in order to minimise transitions. This property reduces power consumption, reduces crosstalk and can even increase the overall throughput of a system, since fewer transitions are needed. A popular encoding scheme is used as the template for ethernet [54, 469–471] and USB [29]

physical layers, and its aim is to reduce the number of transitions in a code word. This is unlike *Manchester Encoding* [51, pp18–21], the purpose of which is to provide edges at three times the base clock frequency. This very energy-expensive scheme facilitates clock recovery at the receiver.

Key to ensuring Electromagnetic Emissions Compatibility (EMC) is a low 'effective signalling rate' (i.e., a low frequency of transitions, regardless of the master clock frequency). Easy EMC compliance brings huge cost and complexity benefits to an interconnect systems designer. These concerns were once solely those of off-chip link architects but, with on-chip wires being proportionally larger and more power hungry than logic, EMC is a serious issue on-die as well. Again, the minimisation of transitions minimised the magnitude of electromagnetic radiation, by reducing the effective signalling frequency.

Data encoding may also be used to introduce redundancy into a data stream. Once considered necessary only for those inherently unreliable links, such as transatlantic satellites or long-distance, high-loss cables, redundancy is now also becoming used on-chip. The huge manufacturing variations in modern processes [65, 231–239] make for highly variable transistors and the need for either hugely pessimistic timing assumptions, or the tolerance of a number of errors at run-time.

To this end, *error detecting* and *error correcting codes* are set to play a crucial role in the implementation of modern chip interconnects. A taxonomy of such codes is beyond the scope of this thesis, but the interested reader is invited to consult the excellent books by Macwilliams and Peterson [40; 50]

To boost throughput of a link (almost certainly at the expense of latency), *compression* may be applied to a data stream. Compression takes data at a rate *Rate* and, using a transformation of effective rate *Compression ratio* > 1, produces a data stream of effective rate $Rate' = Rate \times Compression\ ratio$. Compression is an *orthogonal* transformation — it may be combined arbitrarily with other encodings, without effecting their efficiencies. Compression has one major drawback when considering on-chip communication, and this is that it works best when applied simultaneously to a large block of data, something that is rare to find on-chip, in one location. As such, it is generally suited better for off-chip transmissions although, on a wide on-chip link, simple compression schemes such as *run-length encoding* [55, pp.320–322] may still be effective.

Finally, link security may be improved by adding a layer of *encryption* to it. This is another form of encoding, but one where the data is transformed, and becomes unrecoverable to anybody without the correct knowledge of how to decode it [53]. Encryption may be performed in combination with other data encoding schemes: like compression, it is an orthogonal transform. Encryption generally incurs a negligible overhead on a data stream's rate.

Almost all encoding schemes introduce some overhead to the link to which they are applied. This arises through inefficiencies in data transfer, such as the padding required by USB with repeated bit values [29]; the transmission of redundant information, such as in error correcting codes ; the latency overheads of compression and encryption; and in the logical overhead caused by the need to perform the encoding at all. In all cases, except for the last three, the overheads are fundamental and unavoidable. For the others, techniques such as pipelining may help, but at the cost of additional logic.

2.6 Summary

In this chapter, we have seen a brief overview of the issues faced when developing an interconnection system, and how some of them may be overcome. All the topics are as important for on-chip communication that for the more traditional off-chip approaches.

For this reason, we will encounter most of them again in this thesis, and in particular the issues of skew, crosstalk, energy and serialisation.

We now continue our introduction with an overview of the various topologies available for interconnect systems, having stressed to the reader that topology is a concept independent of those presented so far. Thus, any topology may also be combined with any other technique from this chapter.

Connectivity of interconnects 3

Society does not consist of individuals; it expresses the sum of connections and relationships in which individuals find themselves

Karl Heinrich Marx, The Grundrisse, 1857–8

The topology of an interconnection network dictates many of its most important characteristics. Altering the connection graph can allow a designer to manipulate factors such as bandwidth, reliability and quality of service. For example, adding a dedicated, point-to-point link between two logical blocks will increase their bandwidth, and offer a predictable service.

Different topologies have different requirements, both at the resource level, and also at the protocol level. For example, point-to-point links can use a wide variety of signalling techniques, but multipoint ones must beware conflicting drivers, and perform arbitration/conflict-safe signalling[†].

I will now present a brief taxonomy of interconnect connectivities, followed by an evaluation of the performance and the drawbacks and benefits of each. The correct choice from the set is very application dependent, and so no one can be said to be the definitive solution for all designs.

3.1 Point-to-point links

Point-to-point links are the simplest form of interconnection: they directly connect two logical blocks. This allows designers a good deal of flexibility when choosing their interconnect requirements for the following reasons:

- Quality of service is not an issue, since the links are dedicated;
- Bandwidth can be trivially increased by scaling the number of links;
- Each link may use its own encoding scheme and voltage levels, as appropriate;
- Timing closure is necessary only for the two blocks being connected.

The independence of these factors means a designer has the most freedom possible and is essentially able to choose the optimal link methodology in most cases.

The reason point-to-point links are not exclusively used is their high requirements for area and power. Since each link requires its own routing lane, adding more blocks (and hence links) increases the area requirements spectacularly. In fact, to fully connect all n blocks on a die bi-directionally requires $n(n-1)$ links. This size blowup explains the need for alternative

[†]An example of a conflict-safe driver is an open-collector bus (see §3.4.1).

connection strategies and, in the following sections, we will see some other proposals, each with its own advantages and disadvantages.

First, however, let us make the distinction between a point-to-point link and a uni-directional link, since this is a common cause of confusion.

3.1.1 Uni-directional links

A uni-directional link is simply one where data always moves in a single direction. For example, when moving from block *A* to block *B*. This is not to say that there is no information transfer from *B* to *A*: indeed there will almost certainly be transfer of control information, often to ensure the reliability of data transfer. These control signals generally require dedicated wires, and so introduce area inefficiency.

Often, uni-directional links operate with a *push* data behaviour [56] where, once the data channel is idle, the transmitter (block *A*) may transfer data without further consultation of the receiver (block *B*), who must simply accept any arriving traffic. This protocol simplifies the arrangement, since no reservation of receiving resources is required.

3.1.2 Bi-directional links

The simplest bi-directional link is a pair of uni-directional links, connected in opposing directions. This naïve approach can work well for links where the utilisation is high and the traffic well balanced in both directions. However, since both links require their own control wiring, the area footprint is double that of a single link.

3.1.3 Packetisation

A simple way of increasing the area-efficiency is to remove the dedicated control wires and transmit one channel's control information over the other's data path. The area saving is obvious, at the expense of an increase in complexity of the signalling protocol (now each channel must monitor the other for control information), and be able to inject the relevant data. This approach is increasingly common, with the favourite technique for separating data and control being packetisation [35, pp.33–34], which causes switches to perform routing decisions on a case-by-case basis [49, pp.164–223] . In a *Network-on-Chip* application [5] (see also Section 3.4.2), packets are the *de-facto* standard, since they are well suited to traversing a switching fabric.

Many other ways of increasing the area-efficiency of a bi-directional link over that of a pair of links exist, too many to discuss here; but I wish to bring the reader's attention to one more important technique in this field: that of *half-duplex* links.

3.1.4 Half-duplex links

The principle behind a half-duplex link is straightforward: take a single link and allow data transfer over it in both directions, but only in one at a time. This can be thought of as a basic time-division multiplexing (TDM) scheme [51, pp.73–76],[18, pp.158–160] where the time intervals may or may not be fixed.

In the case of fixed intervals, a simple scheme could be that block *A* may transfer to block *B* on every odd clock cycle; and *B* to *A* on every even one. In this way, both *A* and *B* gain

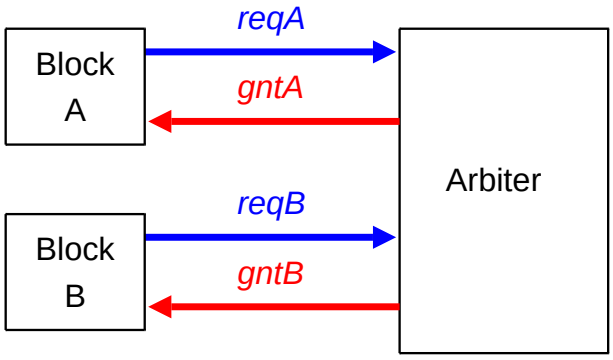


Figure 3.1: An abstract view of a two-way arbiter with request (`req`) and grant (`gnt`) inputs and outputs

50% utilisation of the link, with guaranteed quality of service (the latency is always two clock cycles). The logical overhead is also very low, requiring only a one-bit counter at either end, for an enable signal. This form obviously suits scenarios where traffic is predictable and balanced in the two directions. The allocation policy can be trivially altered to favour one link over another but, so far, it is defined statically.

If traffic is less predictable, or bursty, then an on-demand allocation policy may be favoured. Inevitably, this will require control signals, pushing up the area requirements of the link, albeit potentially by only a small amount. The simplest scheme may see blocks A and B possessing *request* wires, say $\texttt{req}_\texttt{A}$ and $\texttt{req}_\texttt{B}$, converging at an arbiter, which would reply by asserting one of two *grant* wires, say $\texttt{gnt}_\texttt{A}$ and $\texttt{gnt}_\texttt{B}$, to a winning block, which would then be able to transmit until releasing the link by de-asserting its *request* signal. An illustration of this approach is given as Figure 3.1.

We have now seen the distinction between directionality of interconnections, and clarified the varieties available. I will now present the orthogonal concept of connectivity. Each form will have a different degree of connectivity, aggregate bandwidth and availability. In the following sections, I will use the variables M and N to refer to the sets of transmitting and receiving nodes, respectively.

3.2 Point-to-multipoint links

A simple way to connect a single data producer to multiple consumers is to use a point-to-multipoint link (1-to-N). Where the logical topology allows, this scheme allows an increase in effective bandwidth from that of a single point-to-point link to that of N of them, with negligible increase in area over that of an elementary point-to-point link.

Since the aggregated bandwidth of this form of link scales with N, it seems that it can offer almost-miraculous throughput increases, without any increase in complexity. Since that which appears too good to believe always is, our scheme must have a down-side. In the case of a point-to-multipoint link, this is its method of operation: it can only be operated in a *master–slave* manner. The slaves (i.e., the N receiving nodes) may or may not have a shared control wire, which one may jam as a *busy* signal, in order to prevent further data transfer until it is ready. If the control does exist, it is possible for a single node to perform a kind of

'denial of service' attack on the entire network; something to be avoided wherever possible.

This broadcast nature somewhat limits the applicability of the topology, and explains why it is seldom used — after all, few structures involve parallel computation on the same data stream, with little allowance for differing completion times or guarantees on throughput.

3.3 Multipoint-to-point links

Multipoint-to-point links are the inverse of the point-to-multipoint variety. Featuring an M-to-1 topology, they are useful when there are multiple data providers, all wishing to communicate with a single consumer. Again, it is wire-efficient, since all producers may share a common data bus. However, bandwidth is strictly limited by that of the consumer — after all, there is little point placing new data on a bus if the only thing for which it is intended is busy.

Thus, this topology is only useful where the data providers are high latency, or data stream is sparse or sporadic. In this situation, contention on the data bus is limited and the system has a reasonable throughput. Arbitration may be implemented similarly to the previous topology, through a shared control line.

It is easy to see, however, that in most situations a multipoint-to-point link will result in an acute shortage of bandwidth, high contention and a distinct quality of service problem.

3.4 Multipoint-to-multipoint links

Multipoint-to-multipoint links seek to remove the disadvantages of the one-to-N and M-to-one topologies, whilst keeping close to the performance of dedicated, point-to-point links. Since this is a popular topology, such links come in many forms, the most common of which I now outline.

3.4.1 Buses

The simplest form of multipoint-to-multipoint link is that of a broadcast network. Here, all data providers and consumers are connected to the same wire (logically at least — with so many nodes connected, repeaters may be necessary). The broadcast network, often called a *bus*, offers an extremely simple implementation, low propagation latency and the minimal amount of wiring, but with one major disadvantage: contention. On a broadcast network, only one of the $M+N$ nodes may be communicating at any one time and, worse, without care, any node can cause denial-of-service to the entire network. For this reason, above all others, broadcast networks are uncommon on-chip, designers preferring instead to seek solutions where aggregate bandwidth is increased, and fairness (almost) guaranteed.

Off chip, where pin and trace counts are critical, buses are a lot more common — a modern PC contains several, such as the PCI, IDE and AGP buses. Here, a separate control structure links all nodes on the bus to a centralised arbiter, and it is this arbiter's job to ensure fairness and contention-free communication. Fail-safe physical-layer protocols are often used to ensure that in the worst case, where multiple nodes are attempting to drive the bus at the same time, short-circuits do not occur, causing damage. The *open-collector bus* is an example of this, and is illustrated in Figure 3.2 with four input nodes, `A`, `B`, `C`, `D` and an output `out`. We

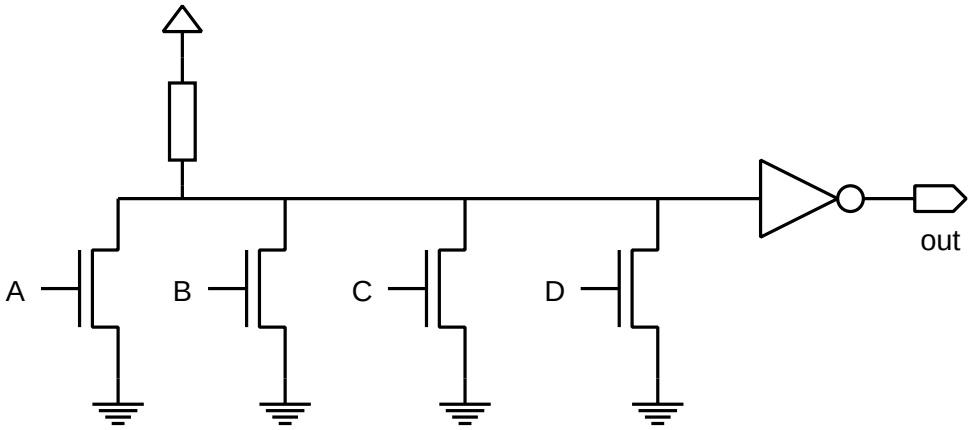


Figure 3.2: An open-collector bus

see that, since there is only ever a passive pull-up, conflicting signals are either masked by the always-on resistor (if they are high) or agree with one another (if they are low).

Given the passive pull-up nature of the open-collector bus, we see that it will have asymmetric rise and fall times. and that its maximum operational frequency will be limited by the charging time through the resistor. As with all passive pull-up schemes, it also features static power dissipation when any node is driving the bus low. For these reasons, it is rarely seen on-chip. It is, however, highly area-efficient, requiring only one transistor per node per bit.

3.4.2 Networks-on-Chip

Generalised as an *M*-to-*N* system, the most common real-world implementation of a multipoint-to-multipoint link is that of a Network-on-Chip (NoC). In this flavour of interconnect, each transmitting and receiving node has a point-to-point link to its nearest *router* node, which aggregates such traffic into *lanes* — point-to-point trunk links — and data will typically traverse several routers before being placed on its final leg on another point-to-point link for its destination node. A diagram of the most common configuration, a packet-switched mesh topology NoC, is seen as Figure 3.3, and was taken from the survey paper by Bjerregaard and Mahadevan [5].

Data producers provide their data with control information, generally in the form of a *packet header*, which allows *routing* of the data through the network to its destination. When packetised in such a way, there is no need for control wiring[‡], and the area efficiency of such a system is good.

By the end of this thesis, we will have seen a step-by-step guide to creating an on-chip NoC, so I give no more explanation here, but encourage the reader to look forward to the material to come.

3.4.3 Circuit-switched networks

Packet-based systems require quite a large logical overhead at the routing nodes, since headers must be looked up, queueing and arbitration may occur, and data packets may be of variable

[‡]In many implementations, control wiring is still required for the purposes of *flow control*.

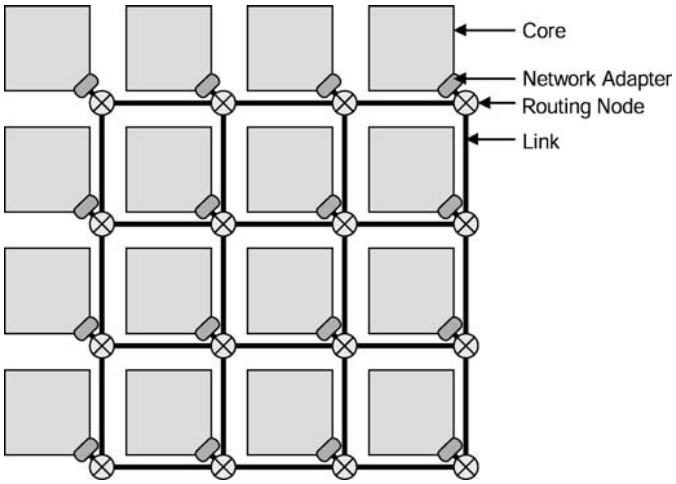


Figure 3.3: A packet-switched, mesh-topology network on chip overview, taken from [5]

length. Recalling from Section 3.1.4, the technique of Time Division Multiplexing, multipoint-to-multipoint systems can eliminate a large degree of routing complexity by forcing data streams to conform to predictable timing patterns. With TDM, a *circuit-switched network* is generated. Here, particular time slots are allocated to each end-to-end link, and a transmitter must comply with the allocation scheme to transfer data successfully. The allocations may be static, which gives the lowest complexity and greatest predictability, or produced on-demand, in response to control signals from the source and destination nodes. The latter increases the available bandwidth for irregular or bursty traffic, at the potential expense of arbitration problems.

Much work has been done on circuit-switched networks, particularly in the field of digital communications, and the interested reader is advised to read the following excellent book by Keshav on the subject [35, 162–174]. We will say no more here, since circuit switching is at a tangent to our approach.

3.5 Summary

We have now seen how both directionality and topology contribute to the selection of an interconnection system. The most scalable choice is the NoC structure, and this has been justified by its substantive deployment in real systems. Therefore, this thesis will focus on implementing such an NoC, using new techniques to minimise its cost per link. In order to do this, however, we must first understand the various physical properties and constraints associated with an integrated circuit link.

Physical characteristics and limitations of interconnects

4

Physical concepts are free creations of the human mind, and are not, however it may seem, uniquely determined by the external world

Albert Einstein

However clever we may be with our signalling schemes and protocols, the performance of an interconnect is ultimately bounded by physics. We can optimise other factors as much as we like, but we will never be able to breach the performance constraints of physical barriers. In this chapter, we will see many of these in turn, with particular relation to wires. First though, let us take on a theoretical bent, and investigate the absolute performance ceiling of *channel capacity*.

Given by Shannon's Coding Theorem [61, p.82], [12, pp.242,249] the maximum data rate *Rate* transferable down a channel, such as one of our interconnect wires, is bounded by the capacity of the channel *Capacity* in the simple relationship

$$Rate \leq Capacity. \tag{4.1}$$

Capacity is equally straightforward to derive, and is given by,

$$Capacity = Blog_2(1 + S/N) \tag{4.2}$$

where B is the channel *bandwidth*, and S/N is the *signal-to-noise ratio* of the channel (see Section 4.3 for a definition). Bandwidth is defined as the range of sinusoids, in Hertz, that may be transmitted from one end of the channel to another [61, pp.78–81], [19, pp.237–249], [12, pp.247–250].

Assuming we fix the type and material of wire to be used for an interconnect, then bandwidth is fixed. We see from Equation 4.2 that the capacity of our interconnect wire will then be directly related to the signal-to-noise ratio S/N. We will now see what this ratio means, and how we may improve its value using purely physical means. We can do all of this before worrying about optimising our transmission protocols.

4.1 Wires

The physical characteristics of the wires dictate their performance. Critically are their properties of resistance R, capacitance C, and inductance L. In this thesis, the parameters $\hat{R}, \hat{L}$ and $\hat{C}$ represent per-unit-length values for resistance, inductance and capacitance, whereas R, L and C are the total line values.

We will now investigate these in turn, show how they affect the performance of interconnects, and how we can optimise for them.

4.1.1 Wire Resistance

The resistance, R, of a wire is its opposition to the movement of current along its length. It is dictated solely by its resistivity, ρ, its cross-sectional area A and its length l.

$$R = \frac{\rho l}{A} \tag{4.3}$$

The equation can be rewritten to make the reliance on wire length and width explicit [65, p.198],

$$R = R_{\square} \frac{l}{w} \tag{4.4}$$

where $R_{\square} = \frac{\rho}{h}$ is coined as the *sheet resistance* or *resistance per square*; and is the resistance per unit area. h denotes the height, or thickness of the wire.

Resistivity is fixed by the material used for the wire, and the temperature it is at. Since temperature is not something not easily controlled at design time (although making a wire with a lower resistance will indeed cause it to run cooler), we ignore this parameter and concentrate on optimising A and l.

The length of the wire will most likely be fixed by the problem under consideration; for example, if two blocks 1mm apart need connecting, a wire of approximately 1mm length will be needed. Complications may arise in routing, where a wire may have to take a circuitous route in order to be placed, and this will adversely affect its resistance. Conversely, good layout and fortuitous routing can reduce the length considerably, and lead to increases in interconnect performance.

This leaves us with cross-sectional area as the main knob to turn when designing for low resistance. Area is, in fact, very easily varied by changing the width of the wire to be laid out. Height is fixed by the fabrication technique used, but different metal layers may have differing thicknesses of material. Thus, by increasing the width, we can reduce resistance to an (arbitrarily low, but still non-zero) level. The ratio of wire height h to that of its width w is called the *aspect ratio*,

$$\text{aspect ratio} = h/w \tag{4.5}$$

and gives an idea of the profile of a wire. I introduce this concept here, but it will not become relevant until we consider fringing capacitances in Section 4.1.2. The aspect ratio is also the key factor in determining the relative values of inter- and intra-layer coupling capacitances.

However, choosing a width is not that easy. By increasing the width of a wire, we have affected several other important factors — not least the area we have used on that metal layer. Increasing the width of a wire by two to three times (dependent on the minimal spacing rules) will halve the number of wires available to lay on that metal layer. Since we need to connect up many blocks, this will cause us tremendous routing headaches. The other major factor to be concerned with is that increasing the width also increases the capacitance of the wire, and we will see in the next subsection that this could end up *increasing* the RC delay (see Section 4.1.2) of a wire, rather than reducing it as intended.

Nevertheless, global wire designers do often favour few, wide, low resistance wires for the most critical signals since, in the vast majority of cases, this does reduce the overall RC delay but, critically, at the expense of a great deal of area

4.1.2 Wire Capacitance

The second important parameter of a wire is its capacitance. Capacitance is the ability of a wire to store energy in an electric field between it and another, charged surface, which is commonly taken to be ground. Capacitance does not retard signals — in fact it can accelerate their phase — however, it does increase the time to bring a wire to a desired value, since energy must be put in to raise a wire's voltage. In this way, capacitance is directly related to the charging time of a wire.

Capacitance occurs when two surfaces couple together capacitively. In an interconnection structure, this happens between a wire under interest and its surrounding wires, ground planes, power planes and substrates. With so many objects to consider, computing capacitance could become overwhelming. Luckily, capacitance is a reasonably short-range effect and is also shielded by closer wires from those further away. Thus, we generally only consider neighbouring surfaces when computing capacitance.

The *time constant* for a lumped capacitor is the time taken for a capacitor to reach a value of $(1/e)V_\Delta$ from ground, when a step voltage of V_Δ is applied at one terminal instantaneously, and the other is grounded.

It is very simply related to the capacitance:

$$\text{time constant}: \; t = RC \tag{4.6}$$

Here, R is the output resistance of the current source (and since no real-world component has zero resistance, t will always be non-zero). It is clear that reducing the value of either R or C will decrease the time constant, and that is the aim of much of interconnect optimisation, since a reduced time constant leads to faster signal edges.

We will now investigate how capacitance arises in interconnection wires. Shown in Figure 4.1 is the standard model used by designers for extracting the capacitance of a wire, in this case wire `m5.b`. Metals 4 and 6 (abbreviated hereafter to `m4` and `m6`) are approximated as planes, designed to simulate the orthogonally routed wires on alternate layers. However, they are most likely arrays of wires in real life (the model is only accurate if their contribution to the capacitance of `m5.b` is insignificant compared with the intra-layer wires on metal 5).

To verify this assumption, I used the capacitance extraction program *Quickcap* [41] with an accurate 0.18μm model to produce capacitance values for a 1mm line laid out as shown in Figure 4.2. I produced models of wires for worst, average and best case layouts for capacitive coupling in metals 5 and 6 wires in our 0.18μm technology.

The results from the configuration with maximum capacitive coupling were as follows: the total capacitance of one of the centre lines (e.g., `m5.b`) was 0.224pF, of which 0.178pF were produced by the two adjacent wires on the same metal layer (`m5.a` and `m5.c`). This represents a huge 79% of the total capacitance of the wire (a little larger than the claim by Ho in [23] that over 70% is possible, but then he models the adjacent layers as ground planes rather than the actual wire layout we consider here, so his estimate would be a little low). In contrast, the wires directly above and below (`m6.b` and `m4.b`) contributed only 0.023pF to the total — a mere 10%, yet note that, since `m4.b` is the same width and aligned and `m6.b` is wider, they should appear like a parallel-plate capacitor to the centre wire. All values are to within ± 3%, and this shows vividly the significance of intra-layer coupling, compared to inter-layer coupling. Hence, the parallel-plate assumptions are correct (and, equally unimportant) to a first approximation (the reader may be interested to know that the dominance of intra-layer

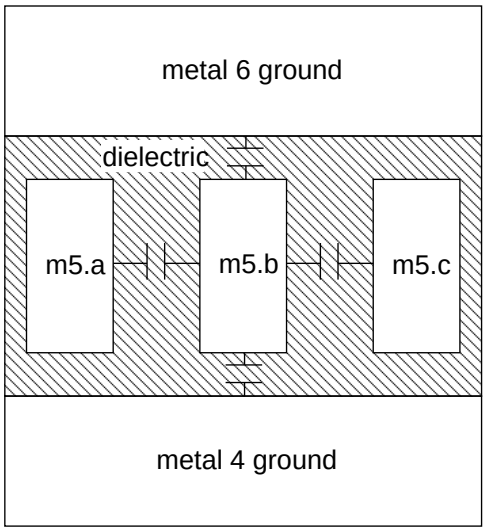


Figure 4.1: Standard surrounded wire model

capacitance is mostly due to the very tall and thin profile of modern metal layers — in this way, wires on the same layer more closely resemble parallel plates for capacitance purposes than do the physically parallel metal layers themselves).

Under this assumption, we may use the parallel-plate capacitor equation to quickly calculate capacitances:

$$C = \frac{\varepsilon_r \varepsilon_0 A}{d} \tag{4.7}$$

Here, ε_r is the *relative permittivity* of the *dielectric* separating the two plates, be it air or, more commonly for integrated circuits, silicon dioxide; ε_0 is a constant; A is the area of the smaller plate; and d the separation between the wire and the other plane in question. This is normally a neighbouring wire.

The parallel-plate equation does not though take into account the *fringing capacitances*. These are the capacitances caused by the non-parallel electric field lines that form at either end of a wire, and also to the edges of the wires along their lengths. Weste and Harris [65, p.201] suggest the follow equation to take these into account:

$$C = \varepsilon_0 \varepsilon_r l \left[\frac{w - \frac{h}{2}}{d} + \frac{2\pi}{\ln\left(1 + \frac{2d}{z} + \sqrt{\frac{2d}{z}\left(\frac{2d}{z} + 2\right)}\right)} \right] \tag{4.8}$$

where z is the height of the bottom of the wire above a ground plane. This equation is substantially more complex than the parallel-plate version and, since the fringing capacitances are only significant for short wires. It is also only valid for values of aspect ratios (see Equation 4.5) of less than two. Another formula, more computationally efficient, and also scalable to aspect ratios of less than 3.3 is shown in [65, p.201].

Since our wires are long, end fringing capacitance is negligible. Additionally, we have just seen that inter-layer capacitance only contributes 10% to total wire capacitance. Hence, the more minor inter-layer fringing effect will be even less important, and may be ignored. Therefore, we will leave consideration of fringing as a topic for the interested reader.

Table 4.1: Coupling capacitances for one millimetre of various track configurations

Configuration	C_{mutual}	C_{gnd}	C_{total}
Minimally-spaced, no gnd	0.115pF	—	0.115pF
Doubly-spaced, no gnd	0.0589pF	—	0.0589pF
Triply-spaced, no gnd	0.0426pF	—	0.0426pF
Minimally-spaced, gnd	0.142pF	0.0534pF	0.195pF
Doubly-spaced, gnd	0.0951pF	0.0624pF	0.157pF
Triply-spaced, gnd	0.0205pF	0.0663pF	0.0868pF
Minimally-spaced, gnd, M4&6 wires	0.0880pF	0.0771pF	0.224pF
Doubly-spaced, gnd, M4&6 wires	0.0454pF	0.0860pF	0.132pF
Minimally-spaced, guard wires, no gnd	0.0155pF	0.102pF	0.118pF
Minimally-spaced, guard wires, gnd	0.00290pF	0.139pF	0.142pF

If the two wires are on adjacent metal layers, the value of d is fixed by the process. If, however, they are on the same layer, d becomes their separation; which may be controlled at design time. For these two scenarios, A is the width times the length of the wire ($w \times l$), or a process-specific height, respectively.

A full table of the results for varying spacings and ground arrangements for a pair of 1mm metal 5 wires is given as Table 4.1. In the table, when I say 'ground', the ground is not a planar approximation as shown in Figure 4.1, but rather an accurate model with metal 4 and 6 wires in realistic locations. This is shown as Figure 4.2. Conventional models use ground planes in the metal 4 and 6 layers, to approximate a set of closely-spaced wires at right angles to the layer of interest (here, metal 5). This approximation is a little coarse since, especially at the lower metal layers, wires may be spaced as highly as they are wide; and so a plane approximation is only half correct, since it ought really be slotted. Choosing to extract worst-case figures, the setup of Figure 4.2 runs wires (i.e., `m4.b` and `m6.b`) parallel to, and the full length of the wire of interest (`m5.b`). Thus, a parallel plane approximation to these layers is maintained, but with fringing also taken into account by the extraction tool. I believe that this gives a higher accuracy of extraction than a simple planar approximation would offer.

Additionally, 'guard wires' means the addition of grounded shielding wires, at locations alternating with signal wires. Since these wires are themselves connected to ground, it is possible to have a non-zero ground capacitance when they exist, even if the ground plane has been omitted from the extraction.

The figures follow the same trend with those given by Weste and Harris [65, pp.203–204]: that increased spacing decreases mutual capacitances, and thus the ground capacitance becomes more significant, but total capacitance is always reduced. However, I believe that they overestimate the contribution from the neighbouring layers, since their figures are derived for the simplified model with planar metal 4 and 6 layers. They do, however, give an enlightening graph of total capacitance as a function of width and spacing on page 205, but again for the lower metal 2 layer, which cannot be directly applied to my work.

Thus, we see that doubling a wire's width may decrease its resistance, from Equation 4.3, but may also increase its capacitance, from Equation 4.7 by the same amount. Thus, its RC delay (Equation 4.6) may be unaffected. Yet, we have increased routing complexity and capacitance.

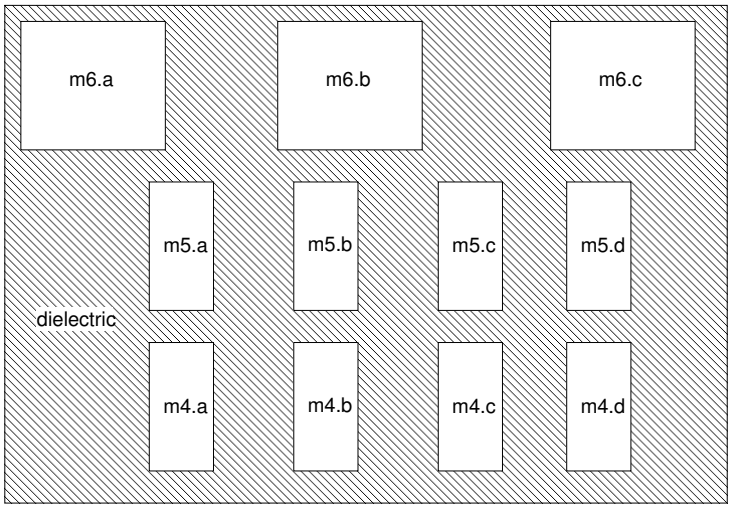


Figure 4.2: Wire layout for wire spacing simulations

In order for the *RC* delay to decrease significantly then, we must also increase the wire spacing. This manipulates the separation, d for same-layer wires, and allows the *RC* product to drop off linearly with increased separation. However, the same proviso applies here as for increasing the wire width: fewer wires may be accommodated for a given area footprint. Much work has focused in the past on addressing this tussle, such as Li et al.'s excellent paper [38] and Weste and Harris' consideration of widths and spacing [65, pp.219–221].

One important point to note about the effect of capacitance is that wires in close proximity will attract the vast majority of the electric field lines radiating from a wire under consideration. The net effect of this that these wires to a first order approximation contribute all the capacitance to a victim. As a by-product of this, any wires further away and in the same direction as an existing wire are effectively shielded from capacitive coupling. During evaluation, it was observed that the mutual capacitances for wires in the same layer drop to negligible levels after only one wire's distance from the victim wire (e.g., a factor of 45 less was seen). Even with far-away neighbours, capacitance falls off linearly with distance (this follows from from Equation 4.7).

This kind of behaviour leads us to say that capacitance is generally a short-range effect, strongly influencing near neighbours, but not a layout as a whole.

4.1.3 Wire Inductance

Inductance is the opposition to change of potential, realised by the ability to store energy in magnetic fields. It does not affect the fields in capacitors, since electric and magnetic fields are orthogonal, but it does effect signal propagation (see Equation 4.15), retarding velocity with increasing magnitude. Also, it contributes to the overall impedance of a wire, and thus impacts current flow. So, like all the wire properties so far, we wish to reduce the inductance of a wire wherever possible (although when a wire operates in the RC region, the contribution of inductance becomes negligible, as we will shortly see).

The inductance of a wire in a system consists of two principle part-inductances: the *self inductance*, L_{self} — the inductance caused by the wire on itself, and the *mutual inductance*,

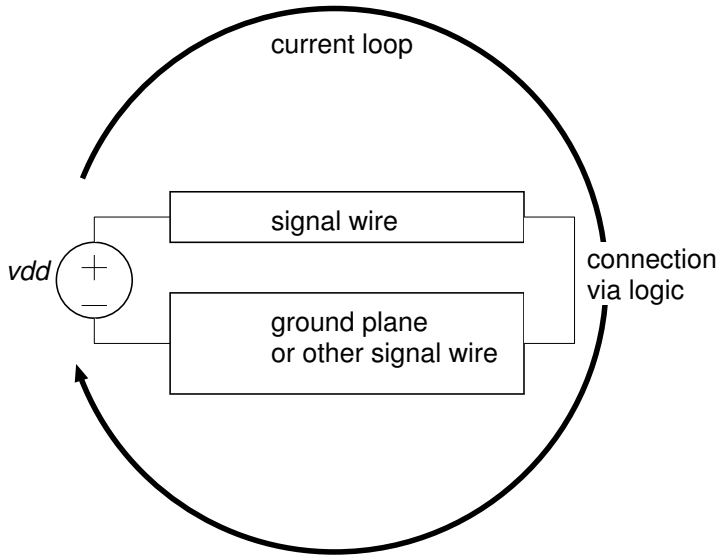


Figure 4.3: A current loop in an interconnection circuit

L_{mutual} — the component from two, coupling, wires.

Unlike resistance and capacitance, the value of a wire's inductance can be increased without increasing the size of the wire. What matters in determining the value of inductance is the size of the *current loop*. Illustrated in Figure 4.3, the current loop consists of a complete path from a power rail to ground. In many cases this may be the loop created by a signal wire, where current at the far end flows back along a ground or power plane, bearing in mind that the electrons themselves do not necessarily have to traverse this route, but rather inductive coupling allows the inductance of flow in the return path. In cases where the designer has been careful to minimise inductance, a *guard trace* may even be used to provide a close proximity return path. This proximity is vital to maintaining the value of a trace's inductance at a low level. This can be seen in the formula for the inductance of a loop:

$$L_{self} = r\mu_0\mu_r(\ln\frac{8r}{a} - 2) \tag{4.9}$$

This equation is for a circular wire, of diameter a producing a loop of radius r [33]. On-chip traces are not circular, but this approximation is close enough to that of a rectangular wire's inductance, given the indifference the result has on the value of a due to the log term. μ_0 is the permeability of free space, and μ_r is the relative permeability of the dielectric on chip.

The equation clearly shows the well-known dependence of inductance on the area enclosed by the current loop — it is directly proportional to r, and this phenomenon is not constrained to circular loops: for any shaped loop, the higher the area of the current loop, the higher the resultant inductance.

Given this, the approach is clear when managing inductance: place traces as close as possible to their return paths, be they other traces or power/ground planes.

Like capacitance, inductance not only affects the wire under consideration, but also neighbouring wires. In fact, unlike capacitance, inductance's range is virtually unbounded, meaning that every wire in a layout can potentially couple inductively to another. This coupling is called *mutual inductance*, and leads to voltages being induced between pairs of

current loops in a circuit. This phenomenon means that inductance contributes to all wires in a system (to a first approximation), and so they need all be taken into account when designing for inductance optimisation.

The mutual inductance between two parallel current loops is given as:

$$L_{mutual} = \alpha \frac{A_1 A_2}{r^3} \tag{4.10}$$

where α is a constant, given in [33, p.421] as 5.08, and A_1, A_2 are the areas of the two current loops under consideration. We again see the criticality of minimising the loop areas.

Also, with any inductor:

$$V_{induced\ across\ an\ inductor} = L \frac{dI}{dt} \tag{4.11}$$

We see here that, for any inductor, changing current flow induces a voltage across this. When we look at crosstalk in Section 4.3.1, we will see that this causes induction of voltages between coupled wires, and the impression of noise therefrom.

Finally, we note that, since minimisation of current loop area almost certainly involves placing selected wires closer together, inductance minimisation is often at odds with capacitance minimisation.

For the system in this thesis, a full inductance extraction was carried out using the tool *Quickind* [42] to gain inductance values for the layout shown in Figure 4.2, a likely scenario for global interconnection wires.

4.2 Delay models

The criticality of propagation delay in the performance of circuits has led to the development of many models, of varying accuracy, for its characterisation. The family ranges from analytical, through computationally infeasible but highly accurate, all the way to simplified but very quick to compute.

4.2.1 The Elmore delay model

The *Elmore delay model* is the most common model used when estimating wire RC delays. Splitting a wire with total resistance R and capacitance C into N sections, it models the distributed resistance and capacitance. Each 'L' shaped branch consists a resistor R_j and capacitor C_i, and is shown in Figure 4.4. When modelled in this way, the propagation delay is computationally efficient to determine:

$$t_{propagation} = \sum_{i=1}^{N} C_i \sum_{j=1}^{i} R_j \tag{4.12}$$

This simplifies to [7, p.18]

$$\tau = \frac{\hat{R}\hat{C}}{2} l^2 \tag{4.13}$$

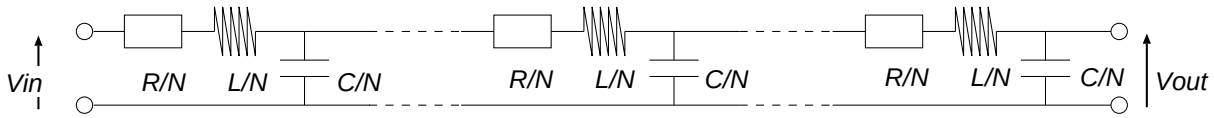


Figure 4.4: Elmore or 'L' delay model

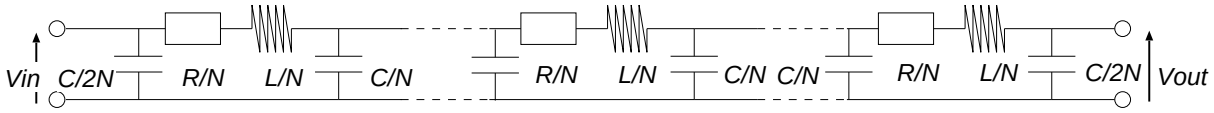


Figure 4.5: π delay model

As $i = j \to \infty$, the Elmore model tends to a true distributed model[†]. However, for high levels of accuracy, i, j may need to be large indeed, and this increases the computational complexity.

4.2.2 The π delay model

The π ($p_{\Lambda I}$) model is a variation of the Elmore model, where each stage's capacitance is split into two capacitances, each of size $C_i/2$. This simplification, shown in Figure 4.5, allows a much greater accuracy, with ranges of ±3% possible from only three stages [65, pp.205–207]. The computational efficiency of the π model is therefore that much better than the Elmore variant [65, pp.206–207], enabling speedups and high accuracies from a much smaller number of segments.

4.2.3 First-order RC delay approximation

If an even greater simplification is desired, then the following expression gives a good first-order approximation [65, p.170] to the time required for the voltage of a lumped RC line's far end to rise from 20%–80% of `vdd`, following the same rise at the near end:

$$t_{propagation} = (\ln 0.8 - \ln 0.2)RC \tag{4.14}$$

4.2.4 Effects on delay of resistance, capacitance and inductance

When combined, resistance R, capacitance C and inductance L provide both the propagation delay of a signal travelling down the length of the wire (i.e., the time for a delayed copy of the input waveform to arrive at the far end, generally taken to be the time between the 50% crossings of the input and output), and the time taken to charge the wire to a given voltage. Equally useful is the *contamination delay*, which is the minimum bound on how long the previous value remains on the wire before being changed by the new value.

[†]But for the assumption of drivers with linear output currents. For non-linear effects, an '$\ln 2$' term can be added to take account of these. The full equation is then $t = \frac{\ln 2 \hat{R}\hat{C}}{2} l^2$, often approximated to $0.37\hat{R}\hat{C}l^2$.

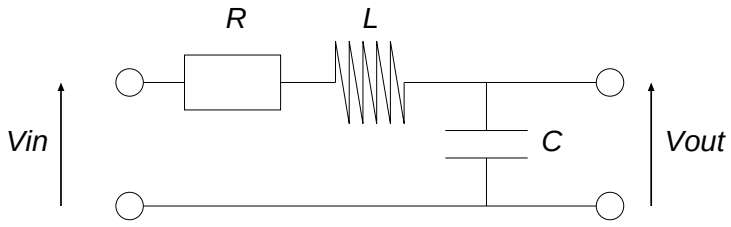


Figure 4.6: Lumped RLC model

The equations for propagation delay are trivial: for a transmission line, where inductance dominates resistance [33, p.423], *vis.*:

$$\text{propagation delay } (\tau_{prop}) = \sqrt{LC} \tag{4.15}$$

and, as we saw in Section 4.1.2, for an RC line, where inductance is insignificant:

$$\text{charging delay constant } (RC \text{ delay}) = \frac{RC}{2} \tag{4.16}$$

This is for the distributed wire model. For a lumped model, it increases to RC (see Equation 4.13). Note that, since R, L and C all increase linearly with length, transmission line delay increases linearly with length, but that of an RC line does so quadratically.

We can clearly see the effect of varying the three parameters R, C and L on the performance of the wire. Fortunately, the are many ways in which these values may be manipulated in order to increase the performance of a given wire. We have seen many of these already, but first I will show that the problem can often be simplified to reduce the consideration down to two variables: namely R and C.

Up to this point, we have been considering the model, shown in Figure 4.6, as a wire with R and L in series, and C in parallel. This is indeed a correct model, however it is the case that we may often approximate it to a wire possessing only R and C. This greatly simplifies analysis, but changes the propagation characteristics.

For a pure RC wire, delay increases quadratically rather than linearly with length (*vis.* Equation 4.16) since,

$$t_{propagation_{RC}} = RC \tag{4.17}$$

and both R and C will double with a doubling in wire length. Also, if a step input is applied at the transmitting end of an RC wire, the shape of the waveform at the receiving end resembles an inverse exponential function, and so waveforms are distorted [33]. For this reason, and RC line is called a *diffusion line*, since sharp edges are spread over time.

We saw in Section 4.1.2 that intra-layer capacitances are much more significant than inter-layer capacitances. So, doubling the width of a wire will reduce its resistance to half, but only increase its capacitance slightly. Hence, we can reduce the RC delay of a given wire. Making a wire fatter does also mean that its driver must be a lot stronger — capable of sourcing an increased amount of current, to charge the additional wire capacitance. We consider this in Section 4.6, but the interested reader may also wish to consult Ho's thesis [21] for information. The RC wire approximation is appropriate here, since inductance is significant only when line resistance is low or the waveform being driven down it is very quickly changing.

We care whether a wire is in the RC or RLC region since it alters its characteristics when a signal passes down it. Whilst attenuation is relatively low, an RC wire smooths out a square pulse into a logarithmic curve, significantly distorting it. This is since the phase of a signal on an RC line is non-linear with respect to the angular velocity of an input frequency component. This leads to a frequency-dependent response for a pure RC wire, and is given by [7, p.24]

$$v_{propagation} = \sqrt{\frac{\omega}{\hat{R}\hat{C}}} \tag{4.18}$$

where ω is the frequency component under consideration. Recalling that all waveform shapes may be split up into a sum of sinusoids by Fourier Analysis ([39, pp.211–279] has a good introduction), different parts of an input to an RLC wire will propagate at different rates. For example, sharp edges contain high-frequency components and these are affected more severely by inductance, and are slowed more than gently rising, or constant slopes. For a pulse edge, the maximum such frequency component is given by the following equation

$$\omega_{max} \approx \frac{2\pi}{6t_{rise/fall}} \tag{4.19}$$

The net effect of this action is to sharpen edges (to make them more upright by propagating the high frequency components preferentially), but also to increase the propagation time of the total pulse. Sharp edges are good for reliable logic operation, but increases in delay reduce throughput — so inductance can be thought of as good and bad here.

Conversely, an RLC wire attenuates in a frequency independent manner, but with a higher, fixed value than those available in the RC region. This so-called *low-loss* region prevents distortion, at the expense of an increased propagation delay.

The propagation velocity of a signal down a pure RLC wire is,

$$v_{propagation} = \frac{1}{\sqrt{\hat{L}\hat{C}}} \tag{4.20}$$

At frequencies higher still than those for the low-loss region, an effect called the *skin effect* comes into play, where attenuation is linearly related to wire length and the the square root of frequency. However, we need not consider this in this thesis since Weste and Harris [65, p.212] claim that this only significantly affects wires thicker than 1.6μm in current technologies like ours.

In the interest of completeness however, Hall, Hall and McCall [17] give the following formula for the skin effect region. The skin depth is the depth of a conductor which contains 63% of the total current, ρ is the resistivity of the conductor, μ_0 is the permeability of free space and ω is the angular velocity of the waveform of interest (or, correspondingly, f is the frequency of the signal).

$$\text{Skin depth} = \sqrt{\frac{2\rho}{\omega\mu_0}} = \sqrt{\frac{\rho}{\pi f \mu_0}} \tag{4.21}$$

Real on-chip wires combine aspects of RC and low-loss regions and, at low frequencies, resistance dominates and the RC equation is sufficient to describe behaviour. The opposite is true for high frequencies. The break-even point is when $\omega L = R$ (i.e., the impedance from inductance equals resistance), and near this region, a hybrid behaviour exists.

It has been shown [65, pp.212–214], [21, p.12 (with a few modifications)], [33, p.150] that inductance is unimportant if *either* of the following hold:

a) Resistive losses outweigh transmission line effects. This is the case when,

$$\frac{R}{2}\sqrt{\frac{C}{L}} > 1 \tag{4.22}$$

b) Edge transition time is more than twice the propagation time of a signal along the wire. This is true when,

$$\text{rise/fall time} > 2\sqrt{LC} \tag{4.23}$$

When in the RLC region, delay of interconnect wires grows linearly with increasing length. Unlike their RC counterparts, RLC lines generally preserve the shape of their input waveforms, and so a delayed copy of the input waveform arrives at their outputs.

4.2.5 Trade-offs

From what we have seen so far this chapter, it seems that all our problems can easily be solved: simply increase wire and driver sizes until a signal can be transmitted at the speed you want. Whilst this is true to a point, it rapidly becomes intractable.

Fatter wires means fewer wires for a given area, so whilst a given wire's bandwidth may be increased using the techniques above, the total bandwidth over an area — say an interconnection channel — may be reduced, since fewer wires are available to be aggregated.

Finding the sweet-spot can be a complicated task, but has been the subject of a good amount of work. For RC lines, good analyses of overall bandwidth optimisation are available. Ho [21, p.20] gives the following equation for the total available bandwidth for a given area, when no repeaters are used:

$$\text{Bandwidth over area} = \frac{1}{3(\texttt{FO4} + \frac{RC}{2})} \times \frac{\text{block width}}{w+s} \tag{4.24}$$

The symbol '`FO4`' represents the *fan-out-four* delay of the wire driver. In our 0.18μm technology, this is around 68ps. The sum of wire width and spacing $w+s$ is also known as the *wire pitch*; and 'block width' is the width of the area of interest, over which bandwidth is trying to be optimised. Equation 4.24 shows that bandwidth over a given area is decreased by increases in resistance or capacitance, and also by increases in wire pitch. However, changing the pitch is more involved than it may seem, since decreasing pitch will increase capacitance, and so careful analysis is needed before choosing to decrease the pitch of a system to increase performance. Naturally, sometimes pitch decreases will be preferred to make way for previously un-routable signals. Wire pitch also increases when wires are made fatter, which reduces resistance.

A more complete analysis of the trade-offs between wire width, spacing and bandwidth, including results on power dissipations for different configurations can be found in Li et al.'s excellent paper [38]. Again considering RC lines, they show that, with a 0.13μm technology and using optimal repeater insertion, maximum bandwidth is produced with minimally spaced, minimum width wires.

When repeaters are not used, the bandwidth is given by Li et al. as:

$$\text{bandwidth} = \frac{\text{block width}}{W+S} \times \frac{1}{\text{delay}} \tag{4.25}$$

W represents the wire widths, S their spacing, and the delay is that derived from considering the wire RC delay along with the output impedance of its driver — for more details the reader is invited to read their paper.

They come to the same conclusion as Ho: that maximum bandwidth is obtained when $W = S = W_{\text{min}}$. Later on, we will see that this result is the inspiration for my implementation's similar design choice.

4.3 Signal integrity

Moving signals quickly is all very well. However, if the receiver cannot determine their meaning, delay optimisation becomes a moot point. Therefore, we also care about signal integrity: how reliably a symbol transmitted at one end of an interconnect can be recovered at the other end.

Generally, we care about a signals voltage value in relation to the threshold values of the transistors in the receiving gate. For example, a 0V signal arriving at the gate of an n-type transistor with a grounded source will definitely not turn on the transistor, but a 1.8V signal almost certainly will. However, if the transistors threshold voltage V_t is 0.7V, what effect will an input close to 0.7V have? McCluskey [43, pp.100–103], Weste and Harris [65, pp.98–99] all consider this topic. For a given transistor, this answer may be characterisable but, with the wide variations in manufacturing experienced today, we cannot say anything about other transistors on the die.

These two uncertainties mean that we need a safety margin when transmitting signals, and this leads to a lower bound being set on the *voltage swing of the inputs*.

Voltage swing

All transistors have two ranges of input voltages, one which they consider to represent logic high (`1`), and one to represent logic low (`0`). To ensure reliable operation, these ranges should be disjoint, and the larger the separation, the higher the noise immunity, since a larger voltage swing is necessary to move from the high to low state and vice-versa. The range where a transistor will reliably interpret the voltage as neither high nor low is known as the *indeterminate region*, and is to be avoided in normal operation. By means of illustration, I reproduce Weste and Harris' diagram [65, pp.98–99] of noise margins as Figure 4.7. Another, good consideration of this topic is given by Johnson [33, pp.63–66].

In this way, the range of voltages used to denote high and low values on a wire influences its tolerance to noise by varying the effective magnitude of signal and that value of noise needed to perturb it. There are two main drawbacks though to increasing the magnitude of the swing: increased delay and increased power consumption.

Delay is increased since the driving transistor must raise the wire to a higher voltage, and this takes additional charge Q, and thus time (by $\Delta V = \Delta Q/C$). Similarly, dynamic power consumption is increased by:

$$P_{switching} = \frac{1}{2} C V^2 \tag{4.26}$$

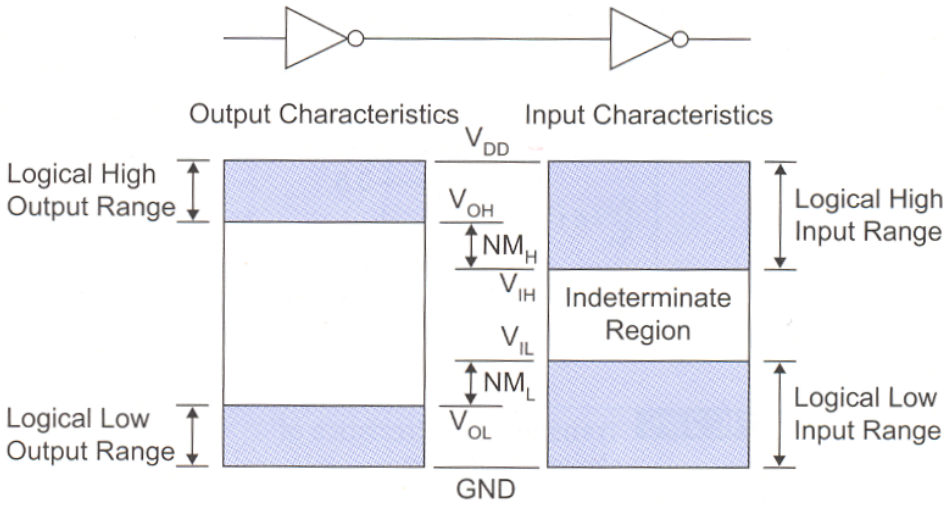


Figure 4.7: Noise margins of conventional CMOS logic (reproduced from [65])

So, high voltage swings have a double performance hit and, for this reason, we often seek lower swings. We will see in Section 4.3.1 that *differential signalling* can be used to give reliable signalling with low voltage swing levels — albeit at the expense of doubling the wiring and increasing logical requirements.

Some implementations trade static power for delay by placing the two thresholds very close together, so the swing needed to switch a transistor is minimal. However, when they are in such proximity, an effect called *sub-threshold leakage* means that the transistor has a greatly increased static power dissipation, and so this technique is ordinarily reserved only for those transistors on the critical path.

Noise

Noise may be induced in the wires, perturbing the voltage on them. Be it from thermal agitation (Johnson noise); Schottky/Shot noise [6], [28, pp.430–432]; leakages to power or other signal rails; or induced by capacitive or inductive coupling; all reduce the available margin for reliable operation at the receiver. We measure the impact of noise on signal integrity by producing a *signal-to-noise ratio*, denoted S/N. Increases in the noise magnitude degrade the value of S/N, whilst increases in the size of the signal increase it. Information theory then gives us the maximum theoretical capacity of the interconnect (see Equation 4.2 for the exact expression).

Of all the possible ways for noise to be induced on a wire transmitting information, the most significant is the mechanism of *crosstalk*, and we will now discuss this.

4.3.1 Crosstalk

Crosstalk is the combined effect of mutual capacitance (see Section 4.1.2) and mutual inductance (see Section 4.1.3) on a wire's signal value. It comes into play when a wire switches nearby to a wire under consideration. For example, if we have neighbouring wires at 0V and one of them switches to some `vdd`, the crosstalk will exert influence on the passive wire to try

to move it towards `vdd` as well. This force will cause some fluctuation in the value of the wire and, hence, some noise has been induced in it. The exact value of the noise depends on many factors, such as the relative capacitances and inductances of the wires, and also the location of the drivers.

Crosstalk due to mutual capacitance

A mutual capacitance C_{mutual} between active wire A and passive wire B causes a current I_{mutual} to flow when the voltage V changes magnitude in one wire [33, p.25]:

$$I_{mutual_B} = C_{mutual} \times \frac{dV_A}{dt} \tag{4.27}$$

We can see that, since voltages on coupled wires will indeed change in order to transmit information, this creation of current will cause voltage changes on wire B, leading to crosstalk, *vis.* [33, p.26]:

$$\%\ \text{crosstalk} = \frac{R_B C_{mutual}}{t_{rise_A}} \tag{4.28}$$

The above equation gives the percentage voltage crosstalk experienced by B when wire A switches with a transition time t_{rise}. R_B is the effective resistance of wire B (i.e., that value which may be gained by taking the real component of B's impedance).

The crosstalk from multiple wires is additive and has sign, dependent on switching directions. For example, if two coupled wires switch in the same direction simultaneously, then their net electric field potential will be unchanged, and no energy will be transferred between them — the wires at that point have an effective coupling capacitance of zero. Conversely, if the two wires simultaneously switch in opposing directions, the potential difference between them is doubled. This leads to a situation where the effective mutual capacitance is twice that of the rest value.

In both situations, the self capacitances (those to ground) are unaffected and may still contribute to crosstalk. If the switching directions combine favourably, it is possible for a highly-switching (i.e., noisy) environment to produce little net crosstalk, although worst cases naturally exist. We will see more of this shortly, in the summary section.

Before we leave this topic, we should examine the effect of varying drivers on wire crosstalk. Weste and Harris show [65, p.209] that, whilst the action of crosstalk on a floating wire is quite severe and undamped:

$$\Delta V_{victim\ wire} = \Delta V_{aggressor\ wire} \times \frac{C_{mutual}}{C_{mutual} + C_{victim}} \tag{4.29}$$

For a driven wire, the story is a little different. Driving transistors are able to reinforce voltage levels and replace or remove that charge which is impressed via crosstalk. This causes a damping of the fluctuations caused by crosstalk, and the interested reader is advised to see [65, p.209] for more details. The stronger the driver, the less influence crosstalk has on a victim's voltage levels, and this can be seen in the upcoming Equation 4.31, .

Ho [23] says that, for non-floating wires, assuming lumped capacitance, we get the following equation:

$$V_{noise} = V_{swing} \times \frac{C_{mutual}}{C_{total}} \times \frac{1}{1 + \frac{t_{rise/fall_{attacker}}}{t_{rise/fall_{victim}}}} \tag{4.30}$$

where V_{Swing} is the change in voltage on the attacking wire; $t_{rise/fall_{attacker}}$ is the fastest of the rise and fall times for a signal edge on the aggressing wire, and the same case holds for the victim wire. When used for balanced drivers, the right-hand term here reduces to one half.

When considering distributed, rather than lumped, capacitance the equation expands as follows:

$$V_{peak\_noise} = V_{swing}^{\ddagger} \times \frac{C_{mutual}}{C_{total}} \times \left(\frac{1+M}{k+M}\right)^{(k+M)/(k-1)} \tag{4.31}$$

where $M = nR_{wire}/2R_{attacker}$, and n is the number of segments in the model. k is the ratio of attacker to victim driving resistances, and V_{peak_noise} is the maximum voltage of noise which is impressed on the victim wire.

For matched drivers, the above equation reduces to:

$$V_{peak\_noise} = V_{swing}^{\ddagger} \times \frac{C_{mutual}}{C_{total}} \tag{4.32}$$

which is twice that of a lumped capacitance, as expected since only half the effective capacitance C_{total} is 'seen' at one end of a distributed capacitance model.

Note, though, that a stronger driver will allow faster edge transitions, and so may itself contribute to a crosstalk problem on other wires. This is a case where drivers must be designed carefully!

Crosstalk due to mutual inductance

For mutually coupled inductors, there are several equations to find the value of L_{mutual}. For example,

$$L_{mutual} = k\sqrt{L_1 L_2} \tag{4.33}$$

k is an arbitrary 'coefficient of coupling', representing the proximity and orientation of the two inductors.

Or, as in the case of a transformer,

$$L_{mutual} = N_1 N_2 P_{21} \tag{4.34}$$

Where N_1, N_2 are the number of turns of inductor 1 and 2, respectively and P_{21} is again a coupling coefficient.

When changes of current flow occur in one conductor, it induces a voltage both in itself and the other. Below we see how the voltage in inductor 1 depends on the current flows in itself and inductor 2.

$$V_1 = L_1 \frac{dI_1}{dt} + L_{mutual} \frac{dI_2}{dt} \tag{4.35}$$

Finally, these can be applied to find the inductive crosstalk equation, between wires A and B, from [33, p.31]:

$$\%\ \text{crosstalk} = \frac{L_{mutual}}{R_A t_{rise}} \tag{4.36}$$

Again, t_{rise} is the rise time of the signal on wire A, and R_A is the real component of A's impedance.

[‡]The V_{swing} term has been added to the original reference's formula, to correct the error there.

This shows that crosstalk can be minimised by:

1) Making the areas of both current loops as small as possible;

2) Reducing the impedance of the wire being aggressed;

3) Increasing the rise time of the signal (making it less sharp).

The second method has already been addressed fully in Section 4.1.3, and a nice side-effect of the interconnect system I will introduce in Chapter 5 of this dissertation will be to minimise the other two points here. This leads to the observation of negligible inductive crosstalk in its implementation.

Summary

In summary, we have seen that, with minimally-spaced wires, crosstalk can pose a serious problem. The layout shown in Section 4.1.2 has many wires closely spaced to the signal wire of interest. Under these conditions, figures from Quickcap show that capacitive coupling is able to impress a worst-case crosstalk of 80% of `vdd` on our wire. This very obviously would cause a signal integrity failure at the receiving end. These worst case figures come when all other wires in the circuit switch in simultaneously, so a more average case, with random switching is also given by my simulations. Here, total crosstalk is at the 56% of `vdd`. All these figures have an accuracy of ±1%. This magnitude of crosstalk will cause signal reliability problems when using an interconnect system that does not actively drive its wires all the time (and so voltage drifts can be significant). Similarly, for actively-driven systems, addition of crosstalk noise can degrade the rise times of components, and add to the delay of a line. Therefore, this noise must be addressed, and one method of doing so is to use *differential transmission*, which will be introduced presently.

Additional data on the effect of crosstalk on signal delay, taking into account wire width, spacing and also the metal layer they occupy (and so their resistance and height) are illustrated well by Bainbridge [2, pp.36–40]. The various graphs given in Bainbridge's book clearly show the expected trends of decreased delay when signals are moved to a higher layer, or routed on wider or staggered wires. It does not show much of a difference when spacing alone is adjusted, and this is due to the wires being on low metal layers, where their near-square profile favours capacitive coupling between adjacent layers, rather than between neighbouring traces. In the higher metal layers considered here, one would expect the trend to show a trend of increased spacing giving increased performance, but Bainbridge does not present this data.

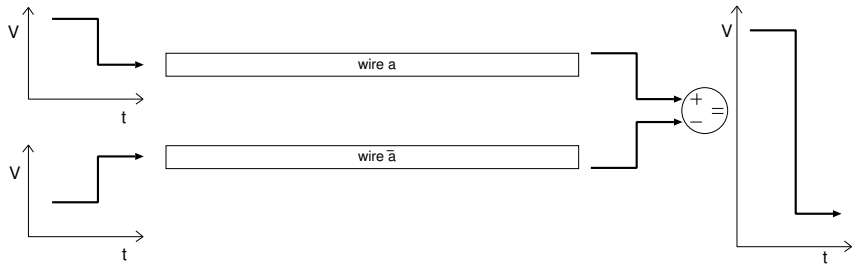


Figure 4.8: Differential transmission of logic values over two wires

Differential transmission

Differential transmission is a commonly used solution to the problem of signal lines with a low signal-to-noise ratio. It takes the approach, illustrated in Figure 4.8 that, if we consider two signal lines in close proximity (say, laid parallel and next to one another), we find that they generally experience the same amount of noise injected from the surrounding environment. This is easily visualised as a hollow sphere of aggressor wires all producing noise for our victim wires at the sphere's centre. At the limit, where our pair of wires are much closer together than any of the neighbours are to either one, we can approximate them as being co-located, and hence both of the pair must experience the same induced noise. This form of crosstalk is referred to as *common mode* crosstalk — the value is common, and equal in sign and phase to both our pair.

Now, suppose that, to transmit a signal of x volts, we transmit both x volts on one wire (A), and $-x$ volts on the other (B). The receiving end simply performs the computation $A-B$ and should obtain a value of $2x$ volts.

If noise is present, this approach has two main benefits:

1) The sum voltage is twice that of the original one, so the signal magnitude is greater than with one wire, hence the S/N ratio is increased;

2) If a common mode crosstalk voltage of ΔV is induced, it will be induced on both. After computing $(A+\Delta V)-(B+\Delta V)$, the noise is cancelled out, to leave just $A+B$ again, theoretically noise-free.

So, we see that differential transmission is extremely useful in high noise or low signal environments to help ensure error-free transmission of information over an interconnect.

The main problems with differential transmission are clear though: we require twice as many wires as normal, and some form of differential generator and comparator circuits are required. The latter is exacerbated by the fact that the generators may need to generate a negative supply voltage, or twice the standard value of `vdd`. Finally, placing two wires next to each other that simultaneously switch in opposite directions means they have a huge, and artificially induced, capacitance problem. Here, a lot of energy is required from a driver to swing them quickly, or degraded performance must be accepted.

To get over these disadvantages, and in particular to save power and increase signalling rates, a form of differential signalling known as LVDS has become popular.

LVDS

LVDS or 'Low Voltage Differential Signalling' is simply differential signalling, but where full excursions to `gnd` and `vdd` are not made. Typically, both wires idle at `vdd/2`, and switch either to `gnd` or `vdd` itself. In other implementations, even lower swings can be used. In this mode, LVDS does not provide as much reliability as its full-swing brother, but allows fast signalling rates, since less energy has to be put in to create a valid signal level. This allows the driving circuits to be smaller, and particularly suits complementary logic styles where, say, values a and $\bar{a}$ are produced and consumed. A brief overview can also be found in Weste and Harris' book [65, pp.227–231].

4.4 The routing problem

Optimisation of delays, crosstalk and overall performance are all very good, but if they cannot be realised then they have all been for nothing. As we have seen, almost all the solutions to these problems involve the resizing or re-spacing of wires in an upwards manner. Thus, they take additional area on their respective metal layers. Increases in area have three interlinked problems: cost, routing and, correspondingly, performance.

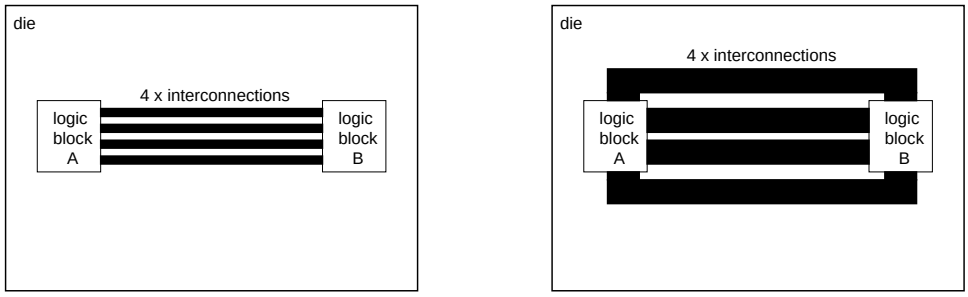
Cost is rather straightforward: if more area is required per trace, the total area required will increase. Thus, the die size must become bigger. Since the cost of a die is proportional in a roughly squared manner to its area [20, pp.17–20], this is a large motivator for compact designs.

Second, track routing becomes increasingly difficult with trace width. This occurs both because, for a given layer area, linearly fewer routes can be squeezed in if each is wider; and also because when corners have to be turned, routing requires the use of an area equal to the *square* of the trace widths. So the number of possible routes quickly dwindles.

Finally, if routing becomes more difficult, then traces are forced to take sub-optimal paths to their destination. This may lead to multiple kinks, vias, or simply a circuitous route. Unfortunately, all of these decrease the performance of the link, and may even make it worse than before the wire was widened.

We can see a 'before and after' picture of this in Figures 4.9(a) and 4.9(b). See how, with the wider wires, fewer can fit in the direct path from logic block A to block B, and so two of the four must take an extended route.

So, we have seen how there are inter-related factors to bear in mind when designing for a practical layout, and that sometimes these will prevent the use of otherwise optimal interconnect track widths. All of these points have shown that interconnect designers must certainly keep their wits about them!



(a) Four narrow wires may take a direct route

(b) But some thicker wires need diverting at a cost of w^2

Figure 4.9: The difficulty in routing wide wires

4.5 Conventional trade-offs in interconnect design

I hope that it has been clear when reading the preceding sections that interconnect design is a hugely complex task, with many inter-related factors to be considered when optimising for performance. Often, in the course of producing a layout, a designer will be working with optimisation of a particular variable in mind, most commonly system throughput. Whilst this may guide their strategy initially, it will often be the case that another will become more pressing when the design becomes closer to completion. A simple example of such a time would be when a system with many parallel channels, each using wide wires is chosen for performance reasons, but layout is impossible since the area requirement is too high.

At this point, a designer must rethink, potentially discarding many months' work. In this chapter, I wish to show the various trade-offs that must be made to produce a viable design for incorporation onto a real die. We will consider trade-offs for the performance metric of throughput (defined as the number of bits of data outputted per second) against those of latency, space, power and design complexity.

4.5.1 Throughput and latency

Throughput and latency are orthogonal concepts, but for many interconnect designers are equally important. Whilst throughput is concerned with the magnitude of data that can be transferred, latency is a measure of the time that data spent traversing the interconnect.

Latency is measured in seconds, and a low latency interconnect can transfer time-critical data reliably. A good example of where low latency is necessary is at bottlenecks such a microprocessor to cache connections. It is well known that a high cache latency can dramatically reduce the work that can usefully be done by a processor [20]. If the interconnection itself adds latency, the situation could get a lot worse. Low latency implementations require careful attention to all components on the critical path. Commonly, for low latency links, the control path will be as critical as the data path, as is our experience from microprocessor designs.

For some applications, however, latency is not critical. This is the case for non-interactive or so-called *bulk* traffic. Here, raw throughput is the sole concern, and hardware implementations can be optimised for highly efficient data transfers, since the constraint of latency has been relaxed.

4.5.2 Throughput and space

The trade-off between throughput and space is perhaps the most straightforward to understand. Most simply, if we have one point-to-point link running with a throughput of B bits/s, then placing N of them side-by-side will offer a total throughput of $N \times B$ bits/s. However, quite obviously, if one interconnect's area requirements is $A\mu\text{m}^2$ then N will take $N \times A\mu\text{m}^2$.

Increases in die area allow the inclusion of dramatically more transistors, and each of these has its own interconnection requirements. If the number of transistors doubles, then an analysis of Rent's Rule [8] suggests that the number of global connections might need to increase by $2^{0.36}$. Whilst this is a slow exponential, it suggests that a die with sides twice a nominal length would need 65% more global wires than its smaller counterpart. Therefore, simply tiling additional copies of interconnections in the manner described above will rapidly deplete both metal layers and driving resources. Therefore, if there is limited space, there must be limited bandwidth, and the two are intimately linked (in fact, we will see later, that Ho has a formula to show just such a relationship [22]).

4.5.3 Throughput and power

Radio aficionados will be aware that throughput and power consumption are often correlated. For example, a more powerful radio transmitter will generally be able to produce a signal with fewer errors at a receiver. This can then increase the available bandwidth, since fewer error-correction bits will be needed, and so the coding overhead decreases.

The physical reasoning is quite different for on-chip communications, but the end result is similar: the dedication of an increased number and/or wider driving transistors; more repeaters; or more wires can all result in being able to support a higher throughput. Needless to say, each of these techniques requires additional power to be expended.

One positive way that on-chip communication varies though is that, whilst a higher throughput may involve increased power consumption, for a fixed data payload communication will be completed in a shorter time period. Thus, when integrated over time, the total energy consumed may be constant across a choice of throughputs[§]. This leads to the notion that design for throughput may be a good idea from a power consumption point of view (provided, of course, that the peak dissipation does not melt your chip!)

4.6 Driving transistors

No transistor can source an infinite amount of current, with zero output resistance, as required by simplistic wire response models. Nor does it have zero output capacitance. These components add onto the intrinsic values of a wire when determining its actual charging time. For example, if the wire has characteristic resistance R_{wire}, and capacitance C_{wire}, and the driver has an output resistance R_{driver}, and capacitance C_{driver}, then the total charging time will be:

$$t = (R_{wire} + R_{driver})(C_{wire} + C_{driver}) \tag{4.37}$$

When we produce a wider wire, we have seen in Section 4.1.2 that its capacitance will increase. Therefore, in order to try and maintain the same rise time, our driver transistor

[§]This assumption ignores static leakages, which is not necessarily a good approximation for modern processes.

must decrease in resistance (the value of C_{driver} is negligible compared to C_{wire} for our long interconnects, and so we will ignore its contribution on rise time. For a similar reason, Equation 4.37 does not include the receiver's input capacitance — if the wide driver is insignificant, so will be a narrow receiving transistor). The output resistance of a transistor is inversely proportional to its channel/gate width, and directly proportional to its channel length. This simple relationship makes it easy for a designer to vary R_{driver} on demand (subject to design rules), and so timing requirements can reasonably easily be satisfied.

The penalties a designer pays for larger driving transistors are threefold though:

1) Area — as mentioned, a lower driving resistance needs a wider channel, and this takes additional area.

2) Power — charging and discharging an increased capacitance requires more power, since the energy per switching $E_{switch} = (1/2)CV^2$, and, for a fixed frequency of operation f, the total power is then $P = (f/2)CV^2$.

3) Input delay — whilst the driver may be able to drive more current, the increased channel size causes an increased gate capacitance. This in turn must be charged by some preceding circuit with a finite resistance. Thus, counter-intuitively, the circuit may actually become slower overall [32].

The contributions of these three overheads can mean that all of a designer's good work has been in vain. All three problems have stemmed from the decision to use a thicker interconnection wire. In particular, the additional driver area overhead, added to that of that produced from the wire re-sizing can lead to some real headaches in routing and layout and, as we saw in Section 4.4, cost.

These factors are a huge catalyst for the development of area-efficient designs. Thus, this is where a large proportion of the motivation of the author's work stems from for this thesis.

4.6.1 Models

There are many models of transistors and how they behave under load. The most simple and ubiquitous is that of *Logical Effort* [32]. This straightforward approach is a good way of making a first-order approximation to which of a set of designs is likely to operate with the smallest delay. However, it omits many factors, and it is not clear how applicable it will be to technologies below the 90nm node. It also optimises solely for delay, at the expense of power and area.

Other models are more complicated, taking into account many more factors about the transistors themselves, such as leakage or layout variation. Weste and Harris provide a good tutorial [65, pp.67–108], and Horowitz and Hill give a more thorough treatment of switching characteristics [28, pp.113–171].

For the transistors performing wire driving, we should understand their operation, since they are so critical to interconnect performance.

4.6.2 Driving transistor behaviour

We now consider the behaviour of the interconnect's most critical components: the wire driving transistors.

The ability of a driving transistor to source a large amount of current is key to driving an interconnection wire effectively, and so we need to check that it is able to do so.

The *alpha power law* [52] is a common way of determining the current sourcing ability of a transistor, when it operates predominantly in either the *saturation* or the *linear* operating region [65, pp.71–75]. At the border, components of both equations are needed to fully describe transistor output behaviour.

In the linear region, the drain-source current ability of an nMOS transistor is:

$$I_{ds_{linear}} = k\frac{w}{l}(V_{gs} - V_{th})^{\alpha/2}V_{ds} = \frac{V_{ds}}{R_{transistor}} \tag{4.38}$$

where k is a compound process constant, w, l are the width and length of the nMOS channel, V_{gs} is the gate-source voltage, V_{th} is the threshold voltage, V_{ds} is the drain-source voltage and α is a variable in the range $1 \leq \alpha \leq 2$, which characterises the shape of the transistor's I-V curve. $R_{transistor}$ is the effective resistance of the transistor, which is constant in the linear region, and so we see the transistor acts as a normal resistor, with the current passing through it being directly proportional to the voltage across the drain and the source terminals.

For the saturated region, where current saturates, the transistor's effective output resistance rises with drain-source voltage and the output current becomes:

$$I_{ds_{saturated}} = P_C\frac{w}{l}(V_{gs} - V_{th})^{\alpha} \tag{4.39}$$

where the symbols are the same as those defined above, plus a process constant P_C.

The main effect of the current driving ability of a transistor is the impact a low value will have on the rise and fall times of a signal going onto a wire, and the contribution of this time to the overall delay of the interconnect system. Ismail and Friedman [30] give some formulæ to calculate the overall wire propagation delay for an RLC wire, considering charge/discharge times.

In the linear region of a transistor, the RLC propagation delay is given as

$$t_{pd_{linear}} = \frac{e^{-2.9\zeta^{1.35}} + 1.48\zeta}{\omega} \tag{4.40}$$

where:

$$\omega = \frac{1}{\sqrt{L_{wire}C_{wire}}\sqrt{1 + C_{driver}}} \tag{4.41}$$

and,

$$\zeta = \frac{R_{wire}}{2}\sqrt{\frac{C_{wire}}{L_{wire}}} \times \frac{R_{driver} + C_{driver} + R_{driver}C_{driver} + 0.5}{\sqrt{1 + C_{driver}}} \tag{4.42}$$

When the driver's output impedance and parasitic capacitance are insignificant with respect to those of the wire, then the right-hand terms of equations 4.41 and 4.42 may be omitted. When this is done, the formulæ become familiar as Equation 4.20 and the left-hand side of Equation 4.22.

When in saturation, the transistor's effect on the propagation delay is given as:

$$t_{pd_{saturated}} = \frac{V_{dd}}{2}\frac{l}{w}\frac{(C_{wire} + C_{load})}{P_C(V_{dd} - V_{th})^{\alpha}} \tag{4.43}$$

Note that, for a fixed gate voltage and size, the transistor's current sourcing ability is constant in the saturation region, and so we can model it as a constant current source. Also note that the saturation equations are frequency-independent, whilst the linear range ones are frequency-dependent. The independence further simplifies our calculations when we are in saturation.

With good design, our driving transistors ought to be able to be in saturation the majority of the time, since we hope to be in the situation where wire delay is much more significant than transistor gate delay. The saturation voltage V_{dsat} is given by the formula:

$$V_{dsat} = P_v(V_{gs} - V_{th})^{\alpha/2} \tag{4.44}$$

and, as an example, the value of V_{dsat}, given by Weste and Harris [65, pp.83–87] is 0.36V. When we come to consider a point-to-point interconnect in the next chapter, we will apply these results, along with some other techniques, to ascertain the optimal sizings for the wire driving transistors.

4.7 Summary

In this chapter, we have seen how physical factors affect our notion of an ideal interconnect system, and how they can interfere with our planned layout. It is very difficult for a designer to wrest with all of them simultaneously, and so prioritisation of one, or a small set of factors must be carried out if a design is to be completed at all.

Of all the parameters we have seen, it is clear that a reduction in track width ameliorates almost all negative effects (e.g., capacitance, crosstalk and routing densities), whilst allowing the designer greater flexibility to add more wires or increase spacings. All of these points have the potential to increase system bandwidth, and so must be welcomed by a designer.

For this reason, the author decided that the production of the following would be of great interest: an on-chip interconnect requiring only minimally-spaced metal 5 wires, and thus yielding an area-efficient and high throughput design. We will see this system in the rest of this thesis, beginning with the next chapter, which will introduce the concept of an area-efficient, pulse-based point-to-point link.

An area-efficient, pulse-based interconnect

5

A journey of a thousand miles must begin with a single step

Chinese proverb

It is clear from the preceding chapters that an area-efficient, power-efficient, high-throughput and low latency interconnect system would be a silver bullet to the needs of modern-day I.C. designers. However, as the previous chapter has also shown, having all of these properties at once is also wishing for the impossible!

So, we must take a subset of these requirements and aim to produce a system as highly optimised as possible for this choice. To this end, I have produced an on-chip interconnect, intended for use with on-chip global data signals, with the following characteristics:

1) Point-to-point: a transmitting and receiving block have a dedicated link, with the reliability and availability that this entails;

2) Serial: with conversion to and from parallel interfaces. This means that it needs only a few global metal wires (in fact, just two per link), and therefore is;

3) Area-efficient: the global metal footprint is very small indeed. This enables many advantages, as described in Chapter 4, such as wide interconnect spacing to reduce signal degradation, or the placement of many different interconnections, to produce a massively point-to-point interconnection system. In keeping with the theoretical results of Equation 4.24 and 4.25, minimum width, minimally-spaced wires are used, to maximise the bandwidth over a given area;

4) Power-efficient: the approach given means that energy consumption is within 25% of an optimised existing approach (and this will be justified in the following chapter);

5) Clock-skew tolerant: to an arbitrary level. My system requires no global clock signal to be transferred with data, and so is able to transcend issues such as clock skew or clock-domain crossing. This quality is very useful in providing easy composition of multiple IP blocks or when interfacing differing logic styles.

In exchange for these benefits, the interconnection system has throughput and latency that is not cutting-edge, but still perfectly acceptable for non-aggressive ASIC designs. The interconnection system has been designed specifically to fit in to conventional design practices and, to this end, includes standard, synchronous FIFO interfaces, black-boxing the data from transmitter to receiver. This allows it to be used as a 'drop-in' replacement for existing interconnection systems in a design. The target application for the interconnect is as

a replacement in an 8-bit ASIC design, running at a clock frequency of 66MHz. This gives a target throughput for the system of 528Mbit/s.

Before I can describe this interconnect properly, I first introduce some related basic concepts; namely *asynchronous logic*; and some prior work called *GasP*.

5.1 Asynchronous logic

Following Chapter 2.4, we are familiar with the issues involved when composing multiple synchronous logic blocks together to comprise a bigger system. Of these, the most difficult is ensuring that a common notion of time is shared between blocks. This reference is needed in order to guarantee that data values, passed between blocks, are sampled at a time when they are valid, and not potentially transitioning. To do this, most synchronous designs introduce a *global clock* signal.

The problem with distributing this common clock is that of clock skew (which I explained in Section 2.4). Clock skew is very difficult to eliminate, or even to reduce to acceptable levels, and yet we still need to compose our blocks.

An approach to solve this problem is to use a flavour of logic called *asynchronous logic*. Asynchronous logic shuns the notion of a global clock, electing instead to generate a set of localised 'clocks', on demand, when the interfacing between two neighbouring logic blocks (or *environments*) requires. These clocks take the form of control wires, shared between the environments, and comprise a protocol called *handshaking* [56].

Since asynchronous logic operates 'on-demand' — handshaking, and therefore activity, is only performed when data is available to be processed — it is power-efficient and, for an interconnect, able to offer power savings over traditional interconnect by shutting down operation when there is no data to be transmitted.

5.1.1 Handshaking

Handshaking involves two, neighbouring components participating in a common control protocol. Synchronous designers will already be familiar with the FIFO control signals (i.e., `FIFO_full`/`FIFO_empty`, `data_read`, `data_write` etc.) Cycling these signals in their permitted orders is a form of handshaking, where the input and output environments are handshaking with the FIFO via the control signal transitions.

With asynchronous logic, all computation involves performing handshaking when receiving or transmitting data. There are many varieties of handshake, each with differing resource requirements and overheads. For a survey, Bainbridge [2, pp. 12–13, 44–45] and Sparsø and Furber [56, pp. 17–22, 57–8] describe in detail both a variety of protocols, and their implementations. I now discuss the two most common ones. These are *bundled data* and *completion detection*.

Bundled data

The bundled data approach operates using separated control wires and data path. This simplifies implementation, allowing the data path to remain unaltered and yet supports handshaking. In this approach, there are two control wires, `request`, which allows a transmitting environment to tell a receiving environment that valid data has arrived; and `acknowledge`,

whereby the receiving environment marks its operation completed, and informs the transmitting environment of this. Following an acknowledge, but strictly not before it arrives, the transmitting environment is permitted to change the data wires. Then we cycle the protocol and a new data word begins to transfer.

Overall, the bundled approach requires the use of additional global wiring for the two control signals, albeit wiring not needing strict timing requirements. Bundled data therefore helps the composability problem, but at the expense of area-efficiency in the form of global metal wiring.

Completion detection

Fundamentally, completion detection embeds control information inside a data flow, removing the need for separate control wires in the forward direction. A backward-flowing `acknowledgement` is still necessary, however, so not all control wires are eliminated.

Completion detection involves altering the format of transmitted data signals to embed information about their current validity in themselves. For example, a simple way to do this might be to take a binary data bits d_1, d_2 and add a data value d_3, which is the odd parity of d_1 and d_2 if they are valid, otherwise it is the even parity. d_1, d_2, d_3 are then transmitted simultaneously. A receiver is then able to detect valid data by simply checking that $d_1 \oplus d_2 \oplus d_3 = 0$, where $\oplus$ signifies a binary XOR operation.

There is a flaw with this particular encoding (e.g., what happens if d_3 changes before new values of d_1, d_2 are received), but it serves to illustrate the point. However, it is also clear that the efficiency of data transfer is reduced: instead of transmitting one data bit per wire, we have two bits per three wires, which is significantly worse. In general though, this is the kind of penalty we pay for completion detection.

The particular form of completion detection I use in my interconnect is that of *dual-rail* logic [56, pp. 11–13]. As its name suggests, dual-rail encoding involves encoding a single bit of data on a pair of wires, d_0, d_1. States are explained in Table 5.1, with the system initialised to the `00` (idle) state, and operation of a cycle proceeds as follows:

1) A data bit is signalled by asserting exactly one of d_0 and d_1, the former to indicate logic `0`, and vice-versa for logic `1`.

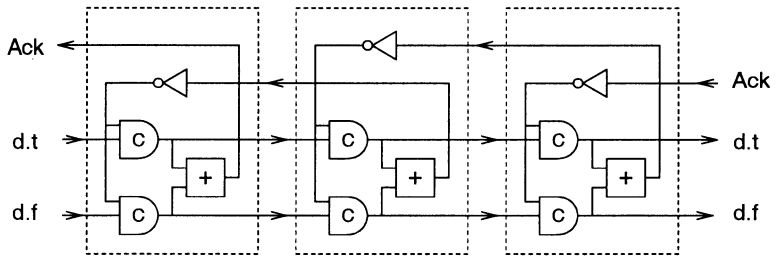
2) The receiver logically `OR`s together the bits, and the output of the `OR` constitutes a `request` signal, so a transition causes processing at the receiver.

3) Having completed processing or latching of the data, the receiver asserts the `acknowledgement` wire.

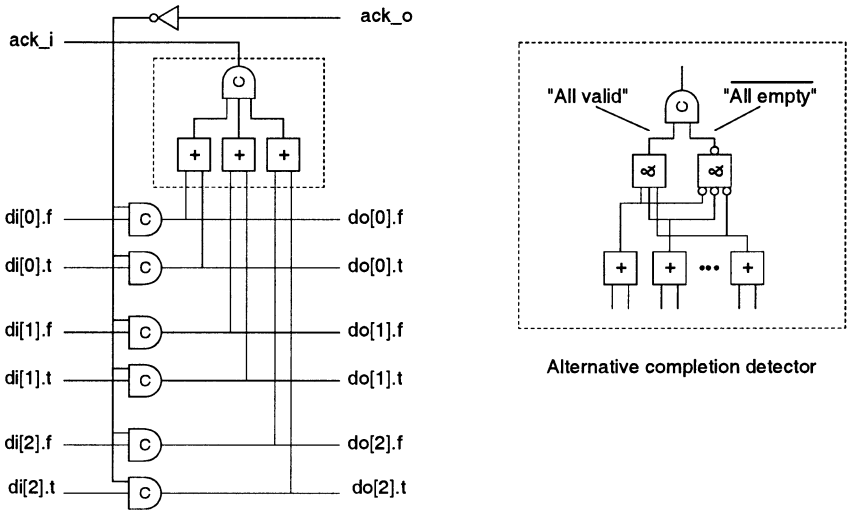
4) At which time the transmitter is free to return the wires to the `00` state (4-phase protocol), or transition exactly one to indicate the next data bit, and repeat from the beginning (2-phase protocol).

5) For a 4-phase protocol only, we now repeat from the beginning.

Also shown, as Table 5.2, is a protocol I call *inverted dual-rail*, whereby the states correspond to inverted wire values. This, inverted, protocol is identical to dual-rail, but with the polarity of the data wires inverted. It is more amenable to implementation in certain logic styles, for example pre-charged logic.



A simple 3-stage 1-bit wide 4-phase dual-rail pipeline.



Alternative completion detector

An N-bit latch with completion detection.

Figure 5.1: Completion detection, as illustrated by Sparsø and Furber [56]

Table 5.1: Dual-rail semantics

Wire Values	Meaning
00	Idle/Invalid data
01	0
10	1
11	Undefined/Error

Table 5.2: Inverted Dual-rail semantics

Wire Values	Meaning
00	Undefined/Error
01	1
10	0
11	Idle/Invalid data

We will see later that it is also suited to implementation with my interconnect solution. By means of illustration of completion detection, I reproduce Figure 5.1 from the book by Sparsø and Furber [56].

Two-phase or four-phase?

Both the bundled data and completion detection protocols can operate in a *two-phase* mode or a *four-phase* one (see Figure 5.2).

With 2-phase, a data bit is transmitted by toggling the state of a data bit, say d_0 to transmit a `0`; or d_1 to transmit a `1`. Following transmission of a bit, the receiver transmits an acknowledgement signal back to the transmitter, who then knows that data has been safely received, and the next bit may be transferred.

The 4-phase dual-rail protocol involves an additional transition per bit transferred, and it thus natively slower than its 2-phase cousin. However, the 2-phase variety requires that state be kept at the receiving end (to determine which wire it was that toggled, since the wires may start in any of four combinations). This state contributes significantly to receiver complexity, and therefore entails a slowdown of its own. 4-phase removes the need for receiver state by ensuring that each handshake begins at a fixed value, which is `00`. As before, exactly one wire will change state to signal data transfer but, unlike before, after an acknowledgement, it will transition back to `0` before the next data bit may be transmitted. This, resetting phase, is known as the *return-to-zero* (RTZ) phase. As previously mentioned, RTZ protocols take longer to perform than ones without this phase, but greatly simplify receiver design (with the associated speedup).

To illustrate the 4-phase dual rail protocol clearly, I include Table 5.1, which displays the four wire states and their meanings. Note that there are two data states, and idle (RTZ) state, and one, unused and therefore invalid state. The existence of the invalid state implies some inefficiency in the encoding of 4-phase dual-rail, but is necessarily wasted by the RTZ nature.

Summary of handshaking

We have seen both the bundled data and completion detection approach to handshaking. Bundled data allows a completely conventional data path design, which makes it more suitable as a wrapper around existing systems. Conversely, completion detection requires a customised data-transfer protocol, but enables additional gains in wire savings and a removal of the strict length matching requirement, that data should arrive with, or before, the `request` signal.

We have also had a taste of the sub-varieties available in the form of two-phase and four-phase implementations. Here, a trade-off is available between complexity of implementation and the number of transitions in a handshaking cycle; with an associated impact on data throughput.

Note that in all completion detection protocols, signal transitions may occur at any time they are valid given the protocol, and we may have to wait arbitrarily long between them. However long we wait, though, data will never be lost at the receiving end, since the next phase of the protocol may not complete out of order. Thus, completion detection avoids the need of a synchronising clock, and yet is *data safe* — no data values may be lost during the run of the protocol due to timing violations.

Finally, all of these protocols have an additional, backwards-flowing *acknowledgement* wire, which contributes further to the area-inefficiency. My approach removes the need for even this wire.

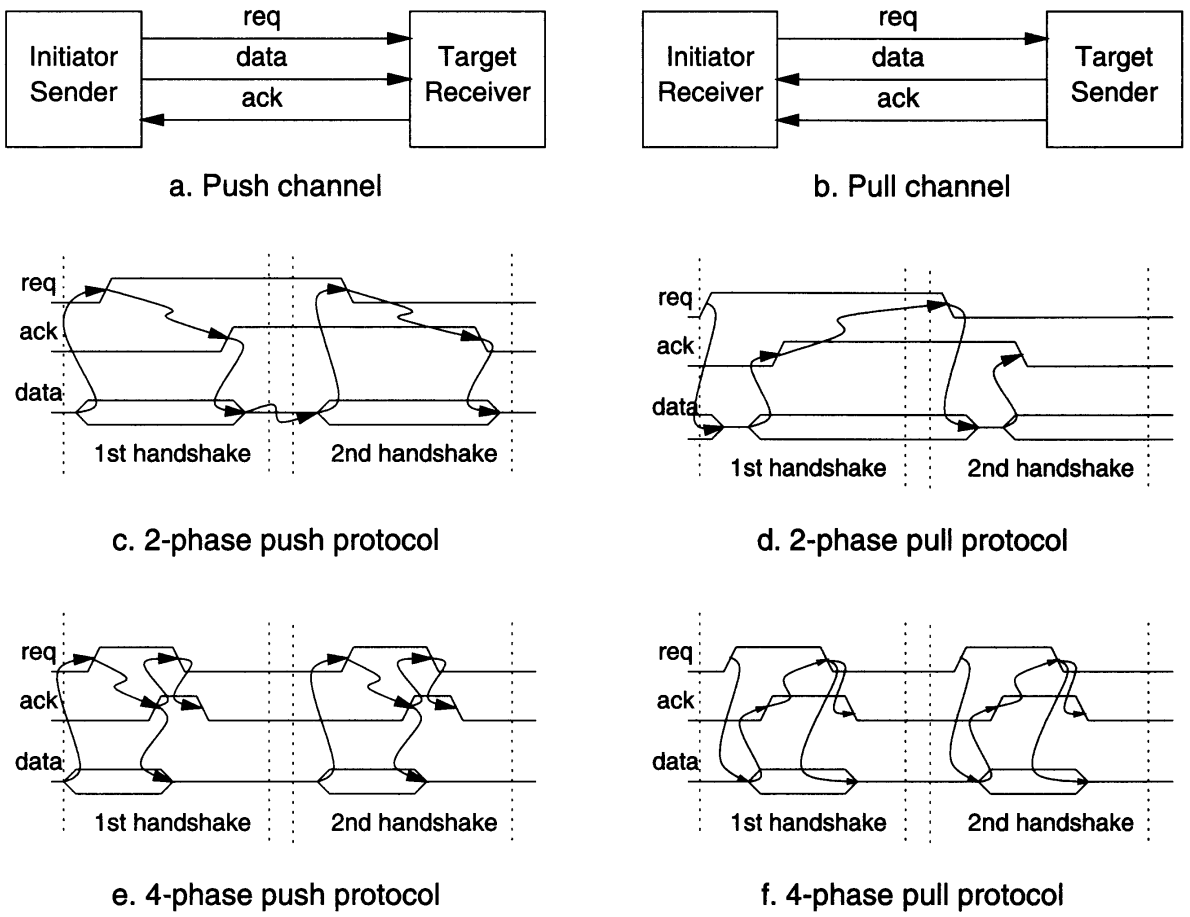


Figure 5.2: Two- and four-phase signalling protocols, as shown by Bainbridge [2]

5.2 The GasP control system

In this section, I describe GasP. Standing for 'Global asynchronous signalling Protocol', GasP is a technique developed at Sun Microsystems [59] for the control of *micropipelines* [58], an asynchronous design paradigm resembling a FIFO structure.

GasP was designed with the intention of being an extremely lightweight control structure, featuring very low latency and fast cycle times. GasP is an implementation scheme for handshaking (see §5.1.1) on the control path of micropipelines.

5.2.1 Micropipelines

I show micropipelines in Figure 5.3, which I reproduce from Sparsø and Furber's book [56]. The idea is to generate control signals (C_i) for use in controlling the sequence of latches deployed in a FIFO element. In the generalised structure, logic can be added between the various FIFO stages to produce a pipelines, asynchronous data flow with computation. Micropipelines use a bundled-data approach, with the control path being shown in that shown in Figure 5.3,. They are primitive asynchronous design elements, and often serve as data buffers between to logic blocks with differing latencies.

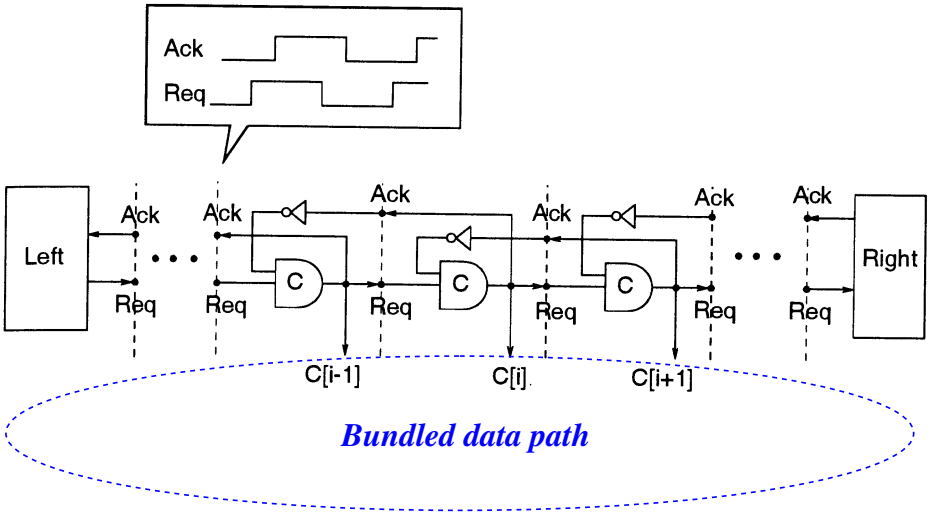


Figure 5.3: A micropipeline, reproduced from Sparsø and Furber [56]

5.2.2 GasP and micropipelines

The presence of both `req` and `ack` signals in the micropipeline is an unwanted area overhead. GasP's key innovation was that it requires only a single control wire to perform bundled-data control. Replacing both the forward-moving `req` and the backward-moving `ack` control wires with a single one, GasP therefore also reduces the control wire count by one.

How does it accomplish this? The key innovation was for GasP's designers to realise that wires are not uni-directional by physical necessity, but merely by popular convention. So, they decided to create a single, bi-directional control wire, capable of propagating both `request` *and* `acknowledge` signals. These wires they call *state conductors*, and they form the basis of GasP.

GasP manages bi-directional communication safely by running an event-based protocol over the wire. The protocol is return-to-one (RTO), for implementation reasons, which means that the idle state is logic high. To perform a `request`, a transmitter applies an input pulse to its driving transistor, causing its state conductor wire to fall to zero, which is seen at the receiver, and noted. Once the receiver has completed use of the data stream (sent bundled data style, on separate data wires, in a conventional binary format), it resets the wire to one via the application of another pulse. Visualised, a cycle creates a waveform looking like a longer pulse, along the state conductor: `1` ↓ `0` (`req`) ; `0` ↑ `1` (`ack`).

The wire transitions are driven by a structure called a *distributed inverter*, which I illustrate as Figure 5.4. Distributed inverters are very simple structures, comprising the same n- and p-type transistors of a standard inverter, but without physical co-location. This allows the n- and p- components to be placed in disparate locations on-chip. Both request and acknowledgement inputs are driven by pulses.

If we assume that the left-hand side of the diagram is the requesting circuitry, and the right-hand side the acknowledging, then we can see how the protocol described above is materialised with this structure: since with the distributed inverter scheme each end may

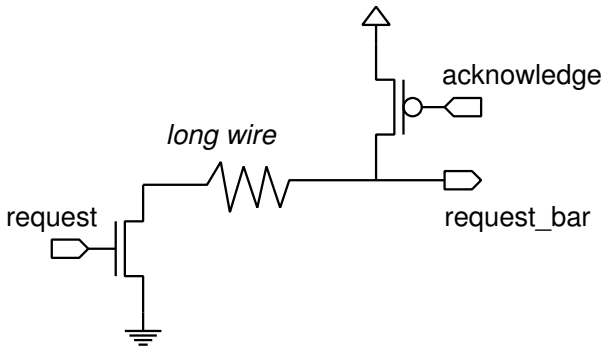


Figure 5.4: A distributed inverter

perform only one *event* (i.e., the left may drop the line, and the right may raise it). Furthermore, it is obvious to see that they must occur in lock-step; and so a similarly-minded protocol is easily enforced by the use of a distributed inverter.

GasP is simply an expansion of the distributed inverter concept, with the distributed inverter's 'long wire' being a GasP state conductor. As previously described, GasP's purpose is as a micropipeline control and, to do this, it is responsible for the propagation of storage *bubbles* — empty storage locations in a FIFO-like structure. A way to visualise bubbles is to imagine the backward-propagating holes left when a data item moves forward though a FIFO. It transpires that these bubbles have exactly the same behaviour as an acknowledgement in a handshaking protocol, when the handshake is talking about the forward-flowing data.

Thus, GasP simply uses the `request` line of a distributed inverter to control the passage of data items through a pipeline (by using it to toggle data latching in locations they call *places*). In GasP, the acknowledgement is automatically created if the succeeding place is free. Hence, data flows as quickly as possible through the micropipeline structure. Since the rate of flow is essentially bounded by the cycle time of a handshake, GasP's very low latency handshake entails high micropipeline throughput.

5.2.3 The unsuitability of GasP for area-efficient interconnect

Good as it is, GasP is not directly suited to the task of controlling data transfers over long distance interconnect, especially one intended to be very area-efficient, such as the design I propose. This is for two main reasons.

The first is that is that GasP is only a control system. Its design merely allows the control of a separate data path, assumed to be propagating in parallel with the control flow (the state conductor is alongside the data wires). This control still requires a global control wire (the state conductor), which is an area price too high for a very area-efficient design.

Second, GasP assumes that all control and arbitration occurs in exactly three inversion cycles. This assumption implies that, whilst wire delays will be tolerated (the data would be delayed alongside the handshaking), there cannot be any significant relative skew between data and control.

This assumption is, unfortunately, not valid for the kind of long-distance interconnect we consider in this dissertation. A separate data path takes up area and also is unlikely to be

routed uniformly, and identically to the control path. On top of this, driver logic variation may result in data skew. The main cause of skew is the probability that, over global distances, wires not be routed uniformly. In fact, at high data rates, even minor differences such as whether a line is on the inside or outside of a turn can cause significant data skew, relative to the control signals. Without being correctly addressed, this could cause data corruption if all data values do not arrive in line with the sampling signal.

My solution, then, is to combine aspects of the GasP signalling system and a dual-rail based protocol, to produce an interconnect approach that is able to serially transmit data bits with completion detection *and* acknowledgement. The total wire requirement will be shown to be *two*, making this a very area-efficient solution[†].

5.3 Introduction to my point-to-point interconnect

I have already described the key features of my interconnect system in the introduction to this chapter. I introduce the components, and highlight how they contributed to produce a high-performance system.

The overall form of the interconnection is shown as Figure 5.5, where we see a block diagram of the system components. The 'zones' I will shortly talk about correspond to the labels across the top of this diagram. Clearly shown on the extreme left and right of the diagram (zones *a* and *g*) are the synchronous input and output interfaces. Standard FIFO interface signals are provided, along with an eight-bit data path.

Following creation of data by an input environment (a *data producer*), the system latches the value in a set of parallel latches (zone *b*), before launching it on its journey across the interconnect. In this way the latches provide a simple *input buffer* functionality.

After the data has been safely captured, the interconnection system performs parallel-to-serial conversion (zone *c*) in preparation for transmission over a serial interconnection. However, it is at this stage that it differs from conventional approaches to interconnection. As will be explained later on in this chapter, the global interconnection of my system is based on a *dual-rail, pulse-based* protocol. I leave the details aside here, but it suffices to say that conversion to this format is required. To perform parallel-to-serial conversion, my design uses a multiplexer tree; with data bits from a parallel word being selected and interleaved in turn to produce a serial stream of data. This interleaving process is the reason that the data bit values `d0...d7` are shown out of order. The level-based data is converted to pulses by truncation of signals at the leaf nodes, triggered by the reception of acknowledgements.

Once the data is in the correct format, and parallel-to-serial conversion has been completed, it is ready for transmission across the global interconnection. This is the zone *d* box at the very centre of Figure 5.5, and most importantly, contains some (potentially very) long wires. These long wires are the most area-hungry component of a global interconnection system and my system is able to transmit the eight-bit data word using just two, compared with eight for a comparable parallel interconnection system. In addition, they do not need to be any wider than a standard trace, which is often a drawback of conventional serial interconnect designs. This factor-of-four decrease in global wiring gives my system a great advantage in area-efficiency.

[†]For comparison, dual-rail requires two data wires per data bit, plus one `acknowledge` wire; and GasP needs one `request / acknowledgement` wire plus *n* data wires, for *n*-bit data.

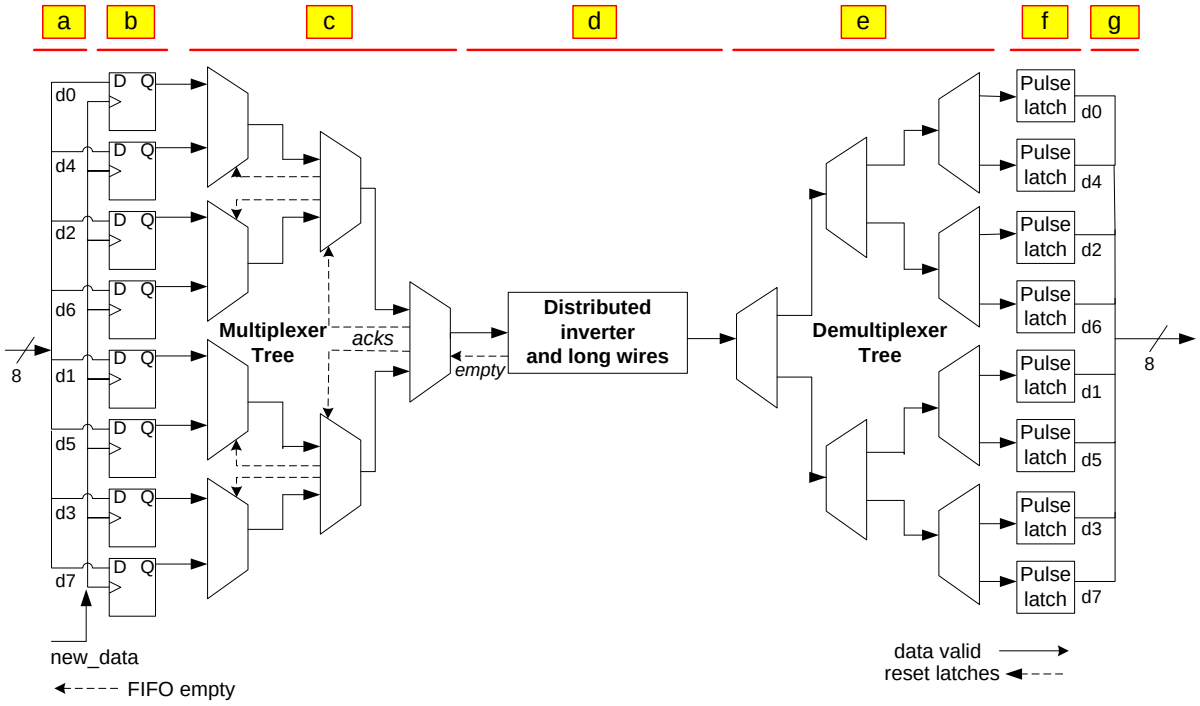


Figure 5.5: Block diagram of my point-to-point interconnect

Once data has successfully travelled down the global wiring, the inverse processes occur: namely that it is serial-to-parallel converted in a demultiplexer (DEMUX) tree (zone *e*); latched; and made available at a synchronous FIFO output interface via some pulse latches in zone *f*.

This entire process of data flows through the interconnect may be summarised in the following manner:

1) Synchronous parallel data is placed in a FIFO-like latch by a transmitting environment;

2) The data is converted to a pulse-based protocol;

3) Data is parallel-to-serial converted by a tree of multiplexers;

4) The, now serial, data stream is dispatched over the global interconnect, driven by a distributed inverter;

5) Serial data is received at the far end, and placed into a demultiplexer tree, where it is serial-to-parallel converted;

6) The data is converted back from a pulse-based form to conventional, level-based signalling;

7) Finally, when a full word of data has been collected, the receiving environment is alerted through a `new_data` signal.

The basic version of my interconnect supports eight-bits with the intention that wider data widths may easily be supported by placing multiple copies of the system side-by-side. Different data width may also be supported, however, by altering the number of levels in the MUX and DEMUX trees (see sections 5.7 & 5.8).

5.4 Chosen data encoding

I now introduce my interconnect's data encoding, and the protocol used to transmit data over the global link. We will see that it is:

- Pulse-based;
- Dual-rail based;
- Control signals are embedded in the data path, requiring no extra wires;
- The total global wire requirement is *two* minimum-sized, minimally-spaced wires;
- The system uses serial transfers to accomplish this.

The data encoding my interconnect uses is pulse based, and the protocol based on a variant of dual-rail encoding (recall from Section 5.1.1). This variant is inverted dual-rail encoding, which was shown earlier as Table 5.2. We see that it is a return-to-one protocol (very much akin to pre-charging), run over two wires, with the transition of exactly one of those wires to zero indicating a single binary value.

The inversion of dual-rail value is just an optimisation, given implementation details. The main innovation comes through the conversion of dual-rail from a level-based system to an edge, or *event* based one. Basing communication on edges, rather than levels offers an number of advantages, many of which boost performance over a more conventional design.

As we have seen, 4-phase dual rail requires a return-to-one phase, during which no data may be transferred. This leads to a performance hit, since we must wait for this cycle to complete. Under my protocol, this corresponds to the need for two edges to be observed per data bit transferred. Since my design uses pulses, which are naturally double-edged, we can roll both of these edges into a single atom, with an associated increase in performance. Essentially, we are able to use a 4-phase protocol with its benefit of simple receiving logic, but the increased utilisation ratio of a 2-phase design (see Section 5.1.1).

This is not the only way I transmit data in an area-efficient manner. Besides eliminating control signals, which reduces the per-bit wire penalty, I choose a serial-based approach.

5.4.1 Serial transmission

Serial transmission was introduced in Chapter 2, and is a well known approach to reducing the number of wires needed to transmit a fixed width of data across an interconnect. Rather than n data bits being sent on n data wires (near) simultaneously, they are queued and sent sequentially across a smaller number of data wires; at the limit, one.

My system performs parallel-to-serial conversion to allow multiple data bits (nominally eight) to be sent across an interconnection that natively supports just a single bit transfer. In

Table 5.3: Area requirements for various interconnection methods, over 10mm of global wiring (optimally repeated and logic buffered)

Configuration	Logic Area	Wire Area	Total Area
Parallel, 8-bit simplex	$4320\mu m^2$	$44800\mu m^2$	$49120\mu m^2$
Serial, 8-bit simplex	$2367\mu m^2$	$11200\mu m^2$	$13567\mu m^2$
This interconnect, 8-bit simplex	$5984\mu m^2$	$11200\mu m^2$	$17184\mu m^2$

this way, my interconnection has a total of only two global wires for an eight-bit data transfer. This is highly area efficient.

To illustrate the point, I show Table 5.3, where I show the various interconnection designs for parallel and serial systems, and we see that mine is extremely lightweight. For the standard systems, I perform optimal repeater sizing and insertion to give a logic area size. The serial link uses a simple shift-register implementation and transmits one data bit in parallel with a clock. All are for repeated implementations.

5.4.2 Voltage swing

Low voltage swing interconnects [65, 229–231] are well know to increase the performance of an interconnect, at the expense of signal-to-noise ratio. The performance increase comes about due to a lower magnitude of slew being required, as was seen in Section 4.3.1.

In my protocol, where a receiving node can acknowledge a data transfer immediately upon observing it, the voltage fall need only be big enough to be seen. After this, the receiver can begin resetting the signal to its idle value. In simulations, however, the swing has generally been for the full line voltages (`vdd` to 0V), except when pulses have been too narrow to charge the entire wire capacitance — and we will discuss this later.

5.5 The core interconnect (dual distributed inverter structure)

At the heart of any interconnect system is a method for physically transferring a chosen data format over global wiring, and my system is no different. The previous few sections have outlined the required properties, and I now show how these may be achieved.

The core is based on the distributed inverter design I showed in Section 5.2, but is adapted to be able to carry *data*, rather than only *events*. The design is shown as Figure 5.6, and we see that it consists of two distributed inverters side-by-side, with some additional logic. To transfer a data bit, exactly one of the wires makes a transition, in accordance with the inverted dual-rail semantics (refer to Table 5.2); and this is received at the far end as data, with a value dependent on which wire transitioned. The receiver latches in the value, then asserts its acknowledgement signal, resetting the line high. Once the high signal propagates back to the transmitter, it is interpreted as an `ack`, and the next data bit is allowed to be sent.

The protocol can be summarised thus:

1) We see that the wire is initially high;
2) The transmitter drives the line low, for a certain time, to signal a data bit;
3) This edge propagates until it reaches the receiver;
4) Observation of the edge caused the data value to be interpreted, and simultaneously the reset phase begins, consisting of the wire being driven high again;
5) After a delay, the pull-up is deactivated, in preparation for transmission of the next data bit;
6) Loop.

Physically, on the active line this looks like the sequence I show in Figure 5.7, where the total protocol is a collaboration between the two logic blocks at either end of the wire, to produce a *data pulse*. The transmitting end creates a leading (high→low) edge by asserting its n-type transistor for a while, to cause a discharge of the wire, with the semantics of a data bit. Later, the trailing (rising) edge is caused by the partner at the far end of the wire. Put together, these create a pulse on the active data wire.

There is one more component to the operation of this link, and it stems from the observation that, were both ends of the wire to be driven simultaneously, both data corruption and power wastage would occur. Therefore, we realise that the two drivers must themselves be driven by pulse-shaped inputs.

If we arrange that each driver is turned on for a period long enough to cause the wire to transition correctly (at around three wire delays), yet short enough so one driver stops before the other begins (so less than the wire delay plus the logic delay of the receiver), we will get effective, reliable and power-efficient operation. The question of how long the pulses should be is critical to the correct operation of this protocol: too long and throughput is degraded by an overly-long cycle time; or worse, an attempt to transmit a subsequent data bit may be swamped, resulting in data loss. Too short a pulse, and the data wire will not be fully charged to one, violating the protocol. Neither prospect is attractive, and so pulse width must be carefully tuned to the length and capacitance of the transmission wires.

Thankfully, though, this tuning turns out to be rather coarse, and we can get away with a range of delays and still have correct operation. This will be shown more clearly in Section 5.6. The range of acceptable values is sufficiently broad that a designer need only roughly estimate process parameters, and yet still end up with a design that works reliably. I now explain how the pulses may be generated.

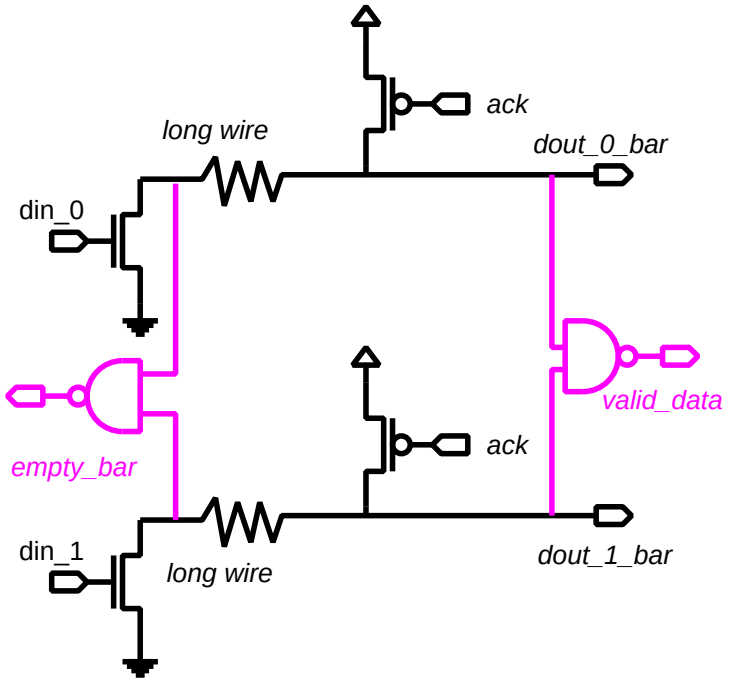


Figure 5.6: Core interconnect, based on a dual-rail distributed inverter

5.5.1 Pulse generation

To generate our pulses, we make use of a form of logic called self-resetting logic (Sutherland and Lexau demonstrate the advantages of this approach [60]), where the presence of a valid incoming signal edge causes a delay to be triggered, and the line to be driven to its active value. Driving continues until the delay has been fulfilled, at which point it ceases. It is the responsibility of the delay to ensure that the output is asserted sufficiently long to create a proper pulse.

Some example implementation details may be seen from the two pulse generators shown in Figure 5.18; or the `n-transistor→delay→p-transistor` loop, part of the stack arrangement of a GasP stage (Figure 5.15). There are many more examples, but these are two that are used in this development process, and they will be fully explained later.

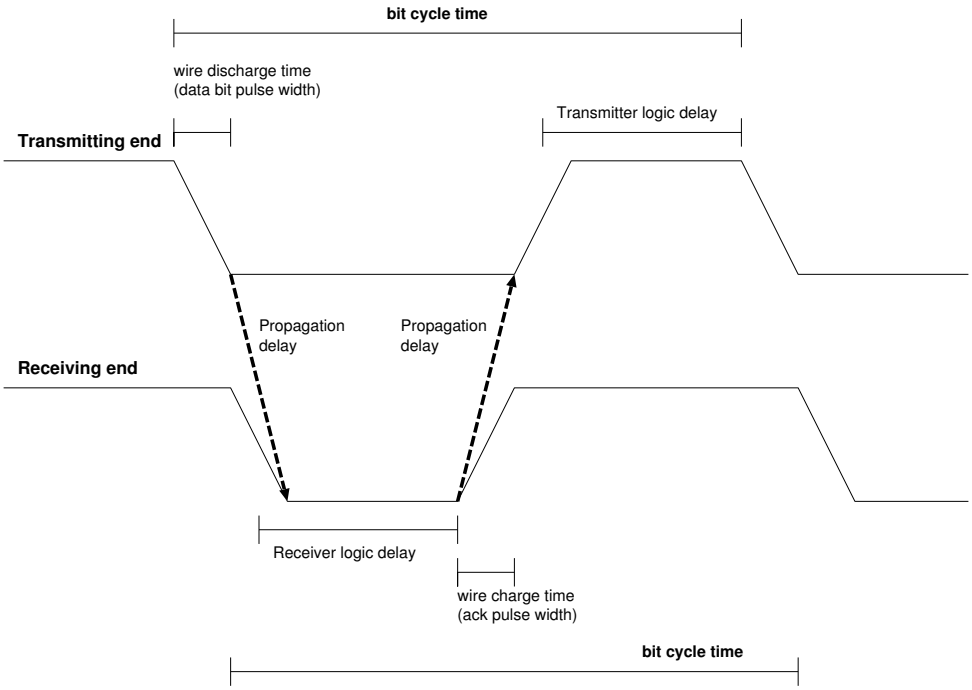


Figure 5.7: Sequence of events as observed on an active core interconnect wire

5.6 Point-to-point interconnect implementation

We have now seen how the core of my system operates. I now go through the full setup of Figure 5.5 and explain the implementation of the remaining components. I will proceed left-to-right through the diagram, and so start with consideration of the multiplexer (MUX) element.

5.7 MUXs

To perform parallel-to-serial conversion of a data stream, my interconnect does not use the standard technique of a parallel-load shift register. Rather it uses a tree of two-input multiplexers (MUXs) operating by alternating their input selection after each data bit flows through them. Parallel data begins at the leaves of the tree and, by the time data reaches the root MUX, it is a fully-serial stream. Refer to Figure 5.5 to see how a tree accomplishes this.

The data flowing down the MUX tree is transmitted using the normal, non-inverted dual-rail protocol. The change to the inverted protocol, used by the rest of the interconnect system occurs directly after the root MUX has been traversed, and data has exited the tree at the interface with the distributed inverter. Thus, in the idle state, MUX outputs are in the `00` state.

A tree structure has numerous advantages over a shift register for our needs. First, for each level in the MUX tree we go towards the leaves, the data rate seen by that MUX halves. This means that, for an n-deep tree, the required speed of a leaf MUX is a mere $1/2^{n-1}$ of the root data rate. For an eight-bit data word, and a line rate of 1Gbit/s, this means we need only design some multiplexers to run at 250Mbit/s. This allows us to optimise these elements for a parameter other than speed, for example power or area. Even if all MUXs are designed equally,

a quarter of the data rate implies a quarter of the power consumption, since the MUXs are asynchronous logic elements and shut down when data is not passing through them. This is a native ability of asynchronous logic, and can offer large dynamic power savings.

Much work was put into attempting to make the multiplexers as lightweight as possible. As such, each MUX has the minimum functionality required, allowing the full, three-stage MUX tree to be travelled in only 1.59ns — just over two FO4 delays in our technology.

Second, remembering that our data protocol involves bi-directional communication along the same wire, it is difficult to see how a conventional parallel-to-serial converter will fit in. However, it is very easy to design a MUX to comply with this protocol: simply use pass transistor logic as the path switch. Then, data values can flow freely in either direction. My MUXs are constructed this way (see Figure 5.9(a)). A further benefit of pass transistor logic is that the data propagation delay is minimal, since a path is selected before its data stream arrives and there is no logic on the data path; so the signal simply passes through.

Overall then, a MUX tree will have a higher performance in my pulse-based system than a more conventional design, and so I have implemented my parallel-to-serial conversion using this approach.

The MUX data path implementation is straightforward: use pass transistors. So, now we turn our attention to the control path. Two-way MUXs require a single control signal, to select between input a or input b.

Because there are small differences in what a MUX must do, depending upon whether it is at the root, leaf, or another location in the tree, there are three different MUX designs. Called the *vanilla, leaf* and *root* designs, each differs slightly from the other. I now outline them briefly.

Leaf MUXs (Figure 5.9(b)) are connected to the input data latches. As we will see shortly, they deal with interfacing to this element of the synchronous input environment. There is only one root MUX, and it (Figure 5.9(c)) interfaces between the MUX tree and the core interconnect's distributed inverter. Finally, vanilla MUXs (Figure 5.9(a)) are used for all other locations in the tree. The major difference between the three designs is the generation and acceptance of handshaking signals.

Vanilla MUXs

The required operation of our MUX blocks is to interleave data bits from the two inputs. This corresponds to toggling the selection of a and b. A natural suggestion, then, is to implement the selection using a toggle flip-flop, and this is, indeed, the approach I take in my implementation. We can see the design of a basic (or *vanilla*) MUX element in Figure 5.9(a). Once a flip-flop is in existence and pass-transistors have been chosen, the design is obvious: connect one path's transistor to flip-flop output Q, and the other to $\overline{\text{Q}}$. The control signal now becomes the flop's clock signal. The only remaining question is how this signal is generated.

It transpires that, if we choose to use the asynchronous logic paradigm, with local handshaking as the control path for the MUX tree, then clock generation is trivial. We simply rename the clock as acknowledgement (ack_in in the diagrams), and the problem is solved. This leaves us solely with the issue of acknowledgement generation. Again, this issue is extremely straightforward: simply use the enable outputs, from Q or $\overline{\text{Q}}$, as the acknowledgements to the next level down the tree. Since Q and $\overline{\text{Q}}$ are mutually exclusive, only one branch will be enabled at any one time, and the selected branch will toggle strictly every acknowledgement:

exactly the behaviour we want. In addition to toggling the state at a MUX, an acknowledgement is also used as the signal to reset the output channel to its idle state (clearing the data value currently on that branch). Each new handshake selects a new data bit for transmission, in the order `d0...d7`, as shown in Figure 5.5, and the termination of the old creates the trailing edge of its respective data pulse.

To further clarify this, I illustrate in Figure 5.8 the paths taken by the first three data bits. We see that transfers always transmit down the opposite branches that were taken by the directly preceding bit; and thus every level of the tree structure operates at half the data rate of its parent. Propagating these acknowledgements deals with all of the non-root nodes, so the final detail is how to generate the single root node's acknowledgement-in signal. The answer to this will also turn out to be remarkably simple.

Root MUXs

First, recall that the root multiplexer interfaces with the input to the core interconnect's distributed inverter. Now recall that a distributed inverter returning from a state with data undergoing transmission to an idle state performs a transition from one line being low to a `11` state. This corresponds to a completion of a cycle. Therefore, the same point in time is when the root MUX should receive an acknowledgement. Further observe that attaching a NAND gate to the distributed inverter data wires will produce a signal that may be interpreted as an indication of whether it is busy with transmission or not. This `distinv_full` signal (active high, see Figure 5.9(c)) is exactly what we use to clock the root MUX node. So, a high→low transition causes the root MUX node to reset its output, toggle its enabled path and generate another (high→low) pulse for the next MUX element downstream, and so on, in a cascade of acknowledgements. The customisable delay between `distinv_full` being asserted and generating the acknowledgement determines the width of the data pulse passing down the MUX tree. Note that an acknowledgement will not be generated by the distributed inverter if it is not able to reset. Not resetting could be caused by the receiving DEMUX tree (see later) being full, and thus we block and 'buffer' data for transmission. However, this is data-safe — no data can be lost.

Leaf MUXs

We have now dealt with the root and vanilla multiplexers, so to complete the discussion of the multiplexer environments, we now consider the leaf MUXs. Interfacing with the synchronous data environment, leaf MUXs see a data word, as latched into a set of input latches. Therefore, they must enforce some data safety property, and only accept data when those latches are actually displaying valid data (i.e., when it is not stale or transitioning). An additional task it to inform the transmitting environment when all bits have been successfully serialised, and the data buffer is ready to be refilled.

The first task is rather easy to accomplish: an `output_enable` signal is produced by a combination of signals from the input environment, and MUX state. Upon loading data into the input latch array, the FIFO protocol calls for the environment to set a `new_data` signal (see Figure 5.5). After a delay, for the data to be correctly latched, this signal can be used as the `output_enable` signal, to indicate that the data in the latches is valid. However, this does not safeguard against stale data potentially being re-transmitted once the MUX tree has cycled through all eight path combinations, and returns to the d_0 point. This issue is

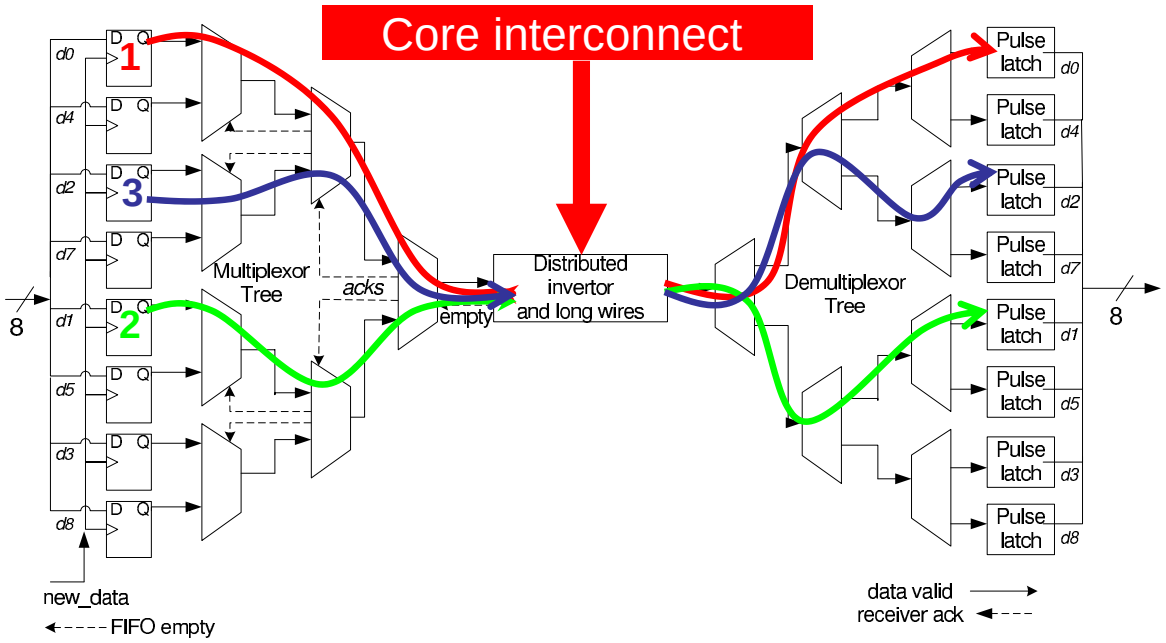


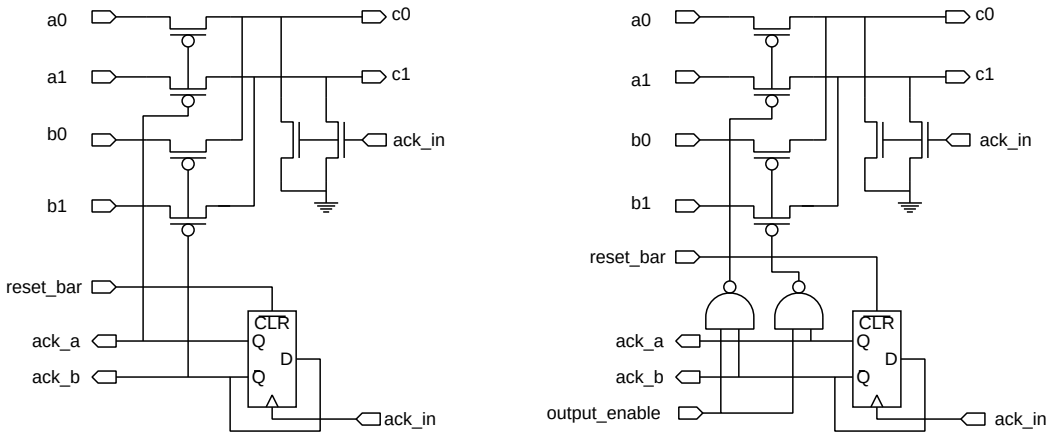
Figure 5.8: Interleaved paths taken by three bits through the interconnect MUX and DEMUX trees

solved in parallel with that of informing the input environment of successful serialisation completion. The latter is easily done by observing when d_7's latch signals an acknowledge-out. A transition here means that the the final tree branch (that containing d_7) has finished its data bit transmission. Thus the word transmission process must be complete. This signal then forms the basis for producing an `FIFO_empty` signal. Since the acknowledgement is, by its very nature, transient the `FIFO_empty` signal is implemented by latching this acknowledgement. The latch is only cleared by the environment's application of the next `new_data` signal. Finally, this `FIFO_empty` signal can be inverted and ANDed with the `output_enable` signal to ensure that the FIFO is not empty (i.e., with stale data at the latches) before really enabling the MUX leaf outputs.

5.7.1 State

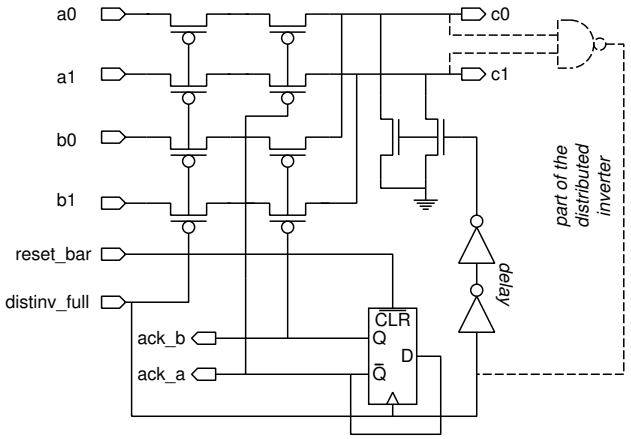
We notice that, other than a single flip-flop to toggle path selection, my MUXs do not have any state. This implies several things about the data path. First, once a data path is enabled, data needs to be able to propagate all the way to the root without encountering any blocking, or it may be lost. This seemingly important issue is actually irrelevant, due to the way a path setup is handled. A new path always begins being established at the root node (after it has successfully forwarded data on the core link), and ripples monotonically outwards to a leaf node. Therefore, by the time a leaf node has enabled a path, and dispatched data down it, the path is guaranteed to be fully established. And, given that MUXs pass data transparently, we see that blocking can never occur, and hence we need no state on the data path.

Such a characteristic, where data passing through an element, such as a MUX, may



(a) The vanilla multiplexer block

(b) Leaf node multiplexers



(c) Root multiplexer

Figure 5.9: Three flavours of multiplexer

assume the next element is ready to accept it, and simply move on without any checks can be expressed by saying that the MUX tree operates in *fundamental mode* [56, pp.83–85]. Strictly, operation in fundamental mode states that an input to a circuit (here a MUX) may change, and then it may not change again until the circuit has stabilised its state and output. Since my MUXs do not have any state, and the output is simply a (barely) delayed copy of the input, this property is trivial to verify.

Further, there exists another theoretical mode called *burst mode*, which is an extension of fundamental mode behaviour. In burst mode, a set of input changes causes a 'burst' of output changes, which all complete before any inputs change again. The whole MUX tree could be considered to also be operating in this mode, since a data word is input (changing multiple inputs), causing a burst of outputs at the root (the stream of data bits to be send over the core interconnect). Once the tree is emptied, a new data word can be loaded, and so the tree as a whole is 'bursty'.

5.8 DEMUXs

Just as a multiplexer tree is used to convert parallel data into a serial format for transmission over the core interconnect, a demultiplexer (DEMUX) tree on the other side re-expands it into parallel. Many DEMUX design features are the same as in the MUXs. For example, we find that the most practical path selector is built from pass transistor logic. However, where a naïve implementer might assume that the design of the control path would be similar, I show that this is not the case. We now see why this is the case.

The control paths differ since the DEMUX tree does not, and can not, operate in fundamental mode. Recall from Section 5.7 that an assumption of fundamental mode allows us to safely ignore emptiness checking of a following element (there, the subsequent MUX in the tree). We see how it is not the case for the DEMUX tree.

The first reason fundamental mode may not hold for the entire tree is that we may wish to make the same optimisation as for the MUX tree: namely to have non-root DEMUXs run more slowly than the root, for power or area reasons. If we implement in this manner, we are no longer guaranteed that a non-root DEMUX can accept a data bit as quickly as the root DEMUX would wish to distribute it. This assumption held for the MUX tree, since MUX performance increases in the direction of data flow, and so a data bit always leaves a MUX destined for another, at least as fast as its current location. With DEMUXs, speed *decreases* in the direction of data flow, hence the hazard.

Secondly, we may wish to purposefully decouple the DEMUX elements' control flows. Whereas the MUX tree features local handshaking, the DEMUX tree does not. The main reason for this is obvious, and is based on the ordering of control and data flows. With the MUXs, a control cycle sets up a path, originating at the root MUX, and flowing in the opposite direction to data propagation. With this behaviour, it is able to complete a path setup in its entirety before any data is sent. With the DEMUXs, the root node would only be able to initiate a control sequence *after* it had already observed a data symbol. Thus, the control flow would run behind the data flow in time, and be the critical path in the system, either delaying data's propagation, or providing obsolete control signals. We can categorise this behaviour by saying that, in the DEMUX tree, data uses a *push channel* protocol [56, pp.10,115,156]. This means that the data is performing a request for servicing to the receiving element, which must *then* process it (i.e., processing occurs *after* data arrival). Here, this corresponds to data arriving off the core interconnect, and expecting to be serviced. For comparison, the MUX tree was a *pull channel*, where an element requests new data (via handshaking) before its arrival.

Another reason for the desire to decouple may be to increase performance. We will see in a later chapter (7), that handshaking can sometimes be a bottleneck, and that decoupling can reduce the handshake round-trip (Figure 5.14).

5.8.1 State

To solve these problems, and to add an additional element of output buffering, the DEMUX design both generates its control signals locally (there are no external control signals), and relies instead on state. State is provided by the addition of D-latches on the data path, in addition to the ones for toggling path selection (this latch is very similar to its cousin in the MUX implementation).

State is both a blessing and a curse. On one hand, it allows simplicity of design by eliminating timing assumptions on the data path, allowing DEMUXs to be interfaced to arbitrarily-slow elements on their outputs, and the removal of inter-DEMUX handshaking. On the other hand, the insertion of a D-latch on the data path entails the addition of a large delay, which will ultimately translate into increased interconnect latency. However, for DEMUXs, the benefits offered by state outweigh the drawbacks, and so we choose to latch the data path. We also gain a boon through latching at every stage in the tree, in the form of free output buffering. If data at the output latches is not consumed quickly, each DEMUX element can hold its respective data bit, and the entire tree can hold just under two data words[‡].

Finally, the lack of inter-DEMUX handshaking allows the use of a universal design, regardless of their location in the tree, so we do not require differing flavours, as was the case for the MUXs.

5.8.2 Operation

We will now see how DEMUX elements generate their local control signals. Please refer to Figure 5.10 for a visualisation of the element. A very neat feature of the DEMUX design is that all state is derived solely from observing the flow of data through that particular DEMUX (and is thus independent of other DEMUXs' state). Hence, we see that the data flow can also be thought of as the control flow; and this is exactly what we intended when we decided upon a dual-rail based implementation.

Since the only interesting events we have in our system are those which transition from high to low, we need only be concerned about transmitting logic `0`s. Therefore, we choose a design utilising n-type pass transistors to produce our path selection circuitry. We also include a global reset signal `reset_bar`.

Clocking the D-flop is perhaps the most interesting aspect of the DEMUX, and is very simple given our protocol. We know that when we see a transition from a `11` input state to a state with either input `0` then we have gained new data. We can use this information to disable the current data path, reset the input and enable an new path.

Input resetting is provided by strong p-type transistors on the input side of our n-type pass transistors. These are able to override any latched low signal coming from the previous DEMUX stage, signalling successful capture. This is then detected by the preceding stage's output, and causes the D-flops to cease driving their outputs via the latch $\overline{\texttt{preset}}$ signal. As with most of our design, the p-types are triggered by a pulse, generated by a pulse generator. The pulse's parameters are vital to correct operation and high performance, and it has been designed so that the initial signal passes through immediately, before being truncated or extended later. This configuration means that it has virtually no impact at all on cycle time.

[‡]Recall that the total number of nodes in a binary tree with w leaf nodes is $2w-1$. Here, the number of leaf nodes corresponds to the data word width, and so the entire tree may store up to one fewer bit than two words.

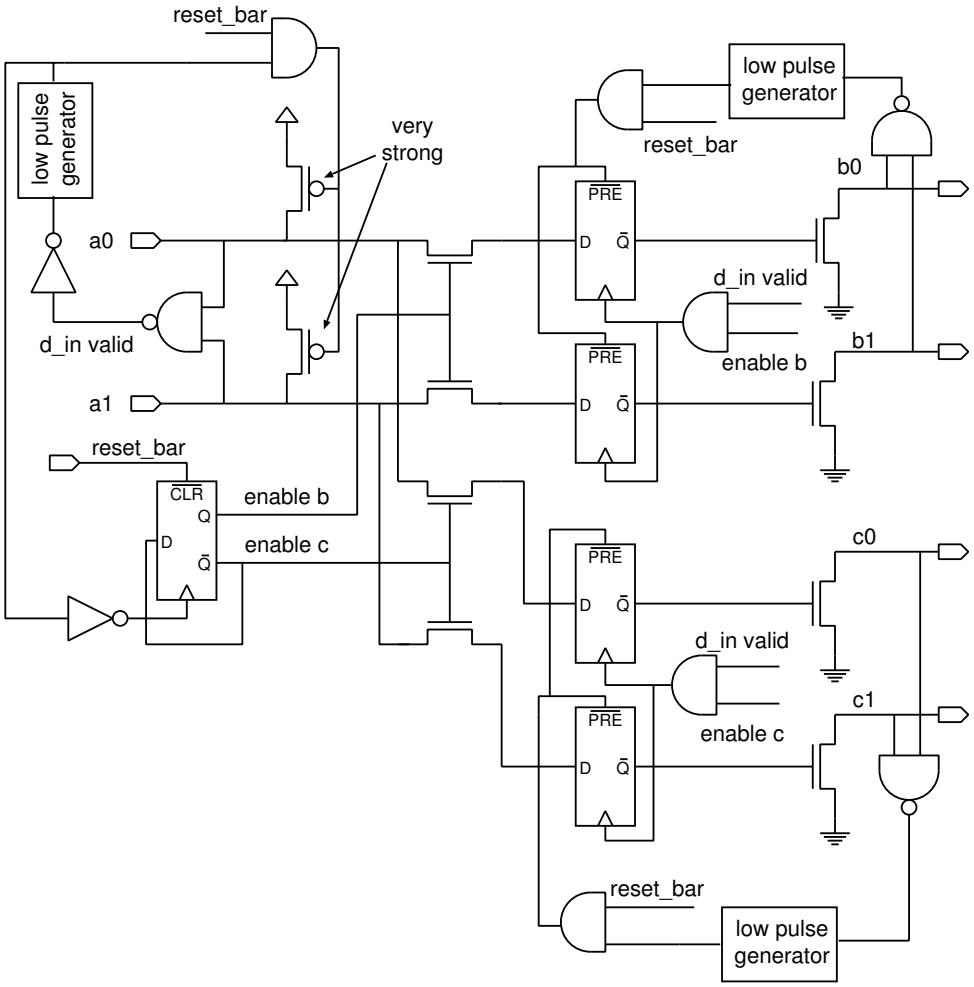


Figure 5.10: Demultiplexer element

5.9 Pulse latches

Following the DEMUX tree, we have a data word, formatted in parallel, at its destination. All that is left is for the output environment to read it in and inform the system of this.

However, up until this point, we have put to the back of our minds that the data symbols are transmitted as pulses. As such, one detail we have not yet cleared up is how the data is transformed from a pulse format, back to a conventional level-based binary format, for consumption by an output environment. It turns out that this is, perhaps, the simplest of all operations in the interconnect. I introduce the concept of a *pulse latch*, its relevance, and describe its operation.

Pulses are, by their very nature, transient. If we are not careful, we risk missing them, if we evaluate too long after they arrive. Therefore, some form of static capture is needed; and this looks rather like a requirement for a latch. However, not any latch will do, since most latch designs operate on logic levels, not events. Therefore, I use the pulse latch shown in Figure 5.11. We see how simple it is, constructed from just two transistors. Also only a single gate capacitance is presented to the input circuitry (one-third that of a balanced inverter), and this makes it very fast to respond to an input. We use one per bit of our inverted dual-rail

protocol. Here, we also note that the pulse latch inverts its input, and even this is of use to us, since it allows us to flip data bits from the inverted dual-rail encoding, and back into the conventional dual-rail one. The output environment's reading of the binary value is is simply a matter of discarding a_0, and reading a_1 (for a dual-rail symbol $\{a_1, a_0\}$).

Why do we transmit a_0 at all then, if we are merely to discard at the readout stage? Well, the answer is simple: the output stage only has valid data once all eight data bits have been received. To know when a bit has been received, we NAND together the two bits, and so a_0 is necessary. Otherwise, we would not be able to distinguish between `01` being transmitted, and the 00 idle state. We extend this reasoning to determine how the FIFO's `data_valid` signal is generated. This signal is the one that informs a receiving environment that a new data word has been successfully received and it should read it out at the earliest opportunity. Following readout, the protocol is for the output environment to reset the latches with the `reset_latch` control signal (equivalent to a more conventional `done` signal). This signal is simply attached to the gate of the n-type featured in the pulse latch, as well as the reset signals for the DEMUX elements. This clears the entire parallel-to-serial logic, and makes it, and its buffer space, available for the transmission of the next data word.

The pulse latches use dynamic logic [65, p.376], and so we must be careful when considering what happens if the output environment has an arbitrary delay before reading out data, since there is the possibility of data loss. Even so, as presented so far, the interconnection will never lose data. This is because the pulse latches get their inputs from leaf DEMUXs, and all DEMUXs feature D-latches on their data outputs. Thus, even though the pulse latch may be dynamic, their inputs are static, and so the pulse latches' values are constantly being refreshed, and they can never lose data due to leakage.

However, this coupling to a D-latch begs the question of redundancy — why bother having a second tier of latches, when all that could possibly add is latency? This is indeed a valid question, and the answer depends on the context of operation. If arbitrary output environment delays are possible, one should use the safe implementation as above. As an aside, if the environment delay is possibly high, then the addition of some latency in the interconnect path is likely to be insignificant in comparison, and so our penalty is not too high. If, however, we can guarantee the output environment always reads its data word within the reliable retention time of the pulse latches, we can optimise our design and eliminate the output latches of the leaf DEMUXs.

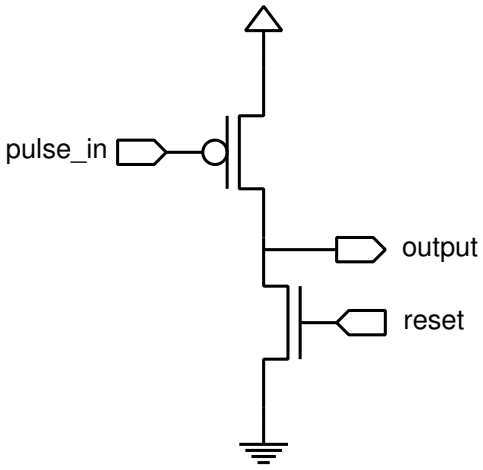


Figure 5.11: Pulse latch

5.10 Synchronisers and metastability

We have now seen the full description of the interconnect's data path, and almost all of the control path (which mostly consists of localised handshaking). We have also seen how data is interfaced to the synchronous input and output environments, and the control signals they provide and receive. The one topic we have not yet covered is the reliability of this interfacing and, in particular, the issue of *metastability*.

Metastability is the process whereby a logic block's output value becomes indeterminate for some period of time. Potentially oscillating between either intermediate, or valid logical values, before finally settling down to a decision one way of the other, metastability can cause a cascade of undesired and erroneous operation in an electronic circuit. Kinniment [36; 37] gives a very good examination of this topic.

The root cause of metastability is generally an indeterminate or changing input value to a logic block. After all, if the input to a function is not fixed, how could the output possibly be? Thankfully, it is rare that an arbitrary piece of logic becomes metastable, and in particular this is thanks to the inclusion of a clock in synchronous circuitry. The definition of the minimum clock period is that time in which all clocked logic blocks have resolved their outputs, given fixed inputs. Thus, in a pipelined design, we can prevent intermediate (and potentially changing) results from propagating through to a subsequent logic block (the latches get in the way, and store a valid value from a previous clock cycle).

However, given that a clock enforces a strict notion of when data should be guaranteed valid, it also causes problems when interfacing with the real world. Real life quantities do not only change on a clock tick, and so violate the very neat assumption of synchronous design. A good example of this is the interfacing of buttons or other mechanical devices to some clocked electronics. The fact that a transition may occur at any time raises the possibility of that chance coinciding with a clock edge, and thus the spectre of metastability.

As I have said, metastability can cause erroneous operation if it is not dealt with carefully. One way to combat it is to redesign a component to be more resistant to it. In general this involves increasing the size of transistors, making the design faster (which decreases the vulnerable window size), but also larger and more power hungry. These factors mean that we

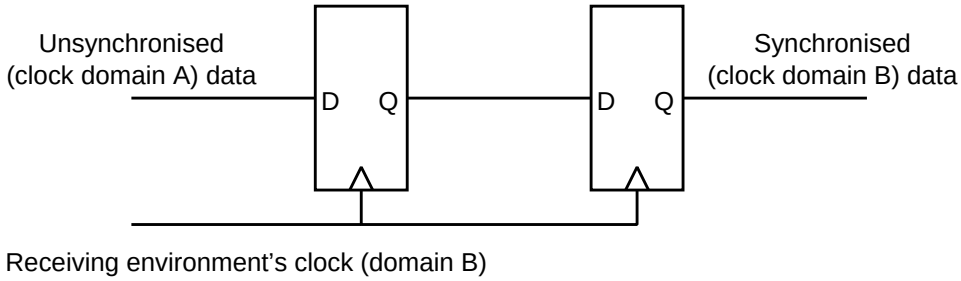


Figure 5.12: Two-flop synchroniser

would like to have as few a number of components as possible exposed to metastability.

A popular way to do this is to make use of so-called *synchronisers* [13, pp.462–486]. A synchroniser is a dedicated piece of electronics whose sole purpose is to reduce the chance of metastability on its output, even if its input may be oscillating or metastable. The simplest, and probably the most effective synchroniser design is to use two D-flip-flops, with the potentially unstable input connected to one, then the output feeds into the next, which then produces the cleaned-up output. The two flops are connected directly to the destination environment's clock. An illustration of this is Figure 5.12.

The downside of a synchroniser is that it increases the latency of the signal passing through it. This can be neutralised in my design, however, by pre-emptively generating the signal, say when the penultimate bit is observed, rather than waiting for the final one. This does, of course, require some assumptions about the speed of data transfer and the clock speed of the synchronous domain, but these should be straightforward enough. If they cannot be made, abort circuity could be added to allow pre-emption to be safe, even if there is the chance or guessing incorrectly.

When crossing clock domains (or as we do, from an asynchronous domain to a synchronous one), the issue of metastability is also encountered, so we must take measures to ensure it does not affect the reliability of our interconnect. Hence, synchronisation is required on those FIFO control signals flowing from the interconnect *to* the environment (i.e., `FIFO_empty` and `data_valid`). Note that synchronisation is *not* required on those signals originating in a synchronous domain and arriving in the interconnect's asynchronous domain (e.g., `new_data` and `receiver_ack`). This is another advantage of an asynchronous logic implementation.

It would be easy at this point to simply say, "Perform synchronisation, and you are safe from metastability", and we would probably be correct. However, we can be more rigorous than this, since there is a formula for the chance of metastability being encountered by a logic circuit, given its parameters. The full calculation is taken from Sparsø and Furber [56, p.80], and is (with some variable renaming for clarity):

$$\text{MTBF} = \frac{e^{t/\tau}}{T_W f_c f_d} \tag{5.1}$$

where t is the gate resolution time; τ is a process-specific metastability constant, approximately equal to 35ps in a 0.18μm technology; T_W is the gate propagation delay; f_c is the clock frequency; and f_d is the input data frequency.

Using values for our process, the formula gives a Mean Time Before Failure (MTBF) for a synchronous environment receiving data from our interconnect, and running at 1GHz, of

greater than 10^{135} years. This is certainly longer than the expected lifespan of any integrated circuitry. using this design we may conclude that, with synchronisation, metastability cannot occur when using the proposed interconnect.

5.11 Wire repeaters

Signal integrity is not the only thing impeded by long wires. Performance is also reduced due to the impact of an increased delay. The delay of a line is responsible not only for end-to-end latency of a system, but also for throughput in a system where acknowledgements are required. My system uses this scheme, and so it has such a potential performance penalty — as an analogy, it may be helpful for the reader to consider the performance of a computer network using a sliding window protocol with window size of one [61]. It is clear that an increased rate of acknowledgements (more strictly a decreased latency on them) increases system throughput (this is illustrated succinctly in Figure 5.13).

Hence, the throughput of a section of my system is, indeed, limited by the delay (and therefore the length) of its wires. There are two approaches to ameliorating this problem: reduce the delay; or segment the wires, producing multiple sections with lower unit delays. Repeaters do both.

The delay of a line can be minimised by the technique of optimal repeater-insertion [21; 31]. This is a well-known approach, and reduces the delay of a signal travelling down a long wire by placing repeaters of optimal sizes at optimal locations.

The Elmore delay model explains why repeaters offer performance enhancements. Recalling from Equation 4.13, where propagation delay, τ, depends quadratically on the length of a wire, splitting the total length l into n segments of length l' reduces the per-segment delay linearly with the number of segments, and so we now simply have to sum n, reduced, delays. Optimal repeating, thus, makes propagation delay linear with total length (see Chang [7, p.18]):

$$\tau_{repeated} = \frac{RC}{2} \times (l')^2 \times n \tag{5.2}$$

Rearrangement makes delay's linear dependence on the total wire length much more apparent:

$$\tau_{repeated} = \frac{RC}{2} \times l' \times l \tag{5.3}$$

Optimal repeating is equally applicable to a our implementation, but additional care must be taken since its links require bi-directional communication, and repeaters traditionally offer only uni-directional operation.

To solve this problem, and also that of the critical impact of acknowledgements on the performance of my system, I chose to pipeline, rather than repeat. In this approach, we split a long wire at intervals with repeaters, and also add elements that provide a full signalling-protocol cycle (they consume the data and re-transmit it, but also provide backward-propagating acknowledgement signals). Our technique, illustrated as Figure 5.14, is similar to the one used by Ho et al. to pipeline an asynchronous channel [22]. Our approach therefore increases system performance, as well as boosting signal integrity.

Repeaters are not only necessary for delay minimisation, they are also invaluable for reducing signal-integrity problems. Repeaters may be used for signal regeneration when a segment with a low signal-to-noise ratio has been diagnosed. Naturally, there is a balance

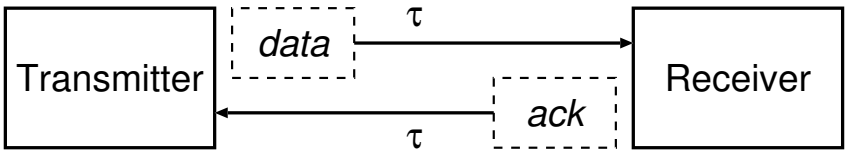


Figure 5.13: Round-trip time of a *data* bit and an *ack* is 2τ

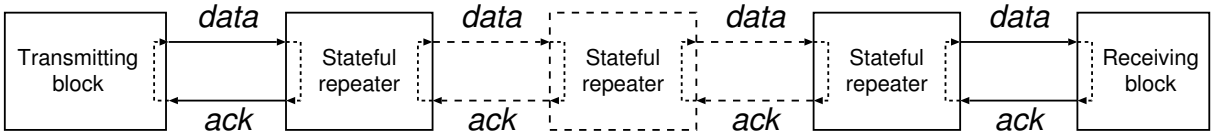


Figure 5.14: The 'Pipelining' effect of Stateful Repeaters — *acks* are generated at each element traversed, and circulate locally in a 'racetrack-like' manner

with S/N ratio being traded off against the additional logic delay of extra repeaters. Signal attenuation can occur due to resistive losses in interconnection wires or due to energy losses in the electric and magnetic fields caused by wire capacitance and inductance. Similarly, these phenomena can create noise on a wire in the form of cross-coupled noise, further degrading the S/N ratio.

In these circumstances, repeaters can be used to boost the signal magnitudes back to their original levels, whilst filtering out a good proportion of the unwanted noise component. The latter is often difficult to achieve though, since simple (i.e., fast) repeaters will often amplify the noise component of the signal, leaving the S/N ratio largely unchanged, albeit at higher magnitudes of S and N.

My stateful repeaters, which we will see shortly, do not suffer from this problem, since the input and output sides are decoupled by a state-holding element. Thus, a noise-free output can be generated, once a value has been successfully latched. State also allows arbitrary delay to be tolerated between NoC components.

I use two types of repeater, dependent on the required function at a particular position in a flow. I will now describe the stateless (and most lightweight) repeater first, followed by the stateful one.

5.11.1 Stateless (GasP) repeater

A pipelined approach has a drawback in the form of increased latency, induced by the addition of state-holding elements on the data path. There are some occasions when state is not necessary, such as for pure signal regeneration applications where the propagation delay of a wire is insignificant and does not warrant state and the premature acknowledgement that comes with it. A prime example of its applicability is as a voltage re-generator for the output stage of some of the other interconnect blocks, where pass-transistor logic can cause some voltage attenuation and signal-integrity problems.

For these applications, I instead use a stateless repeater, derived from the original GasP

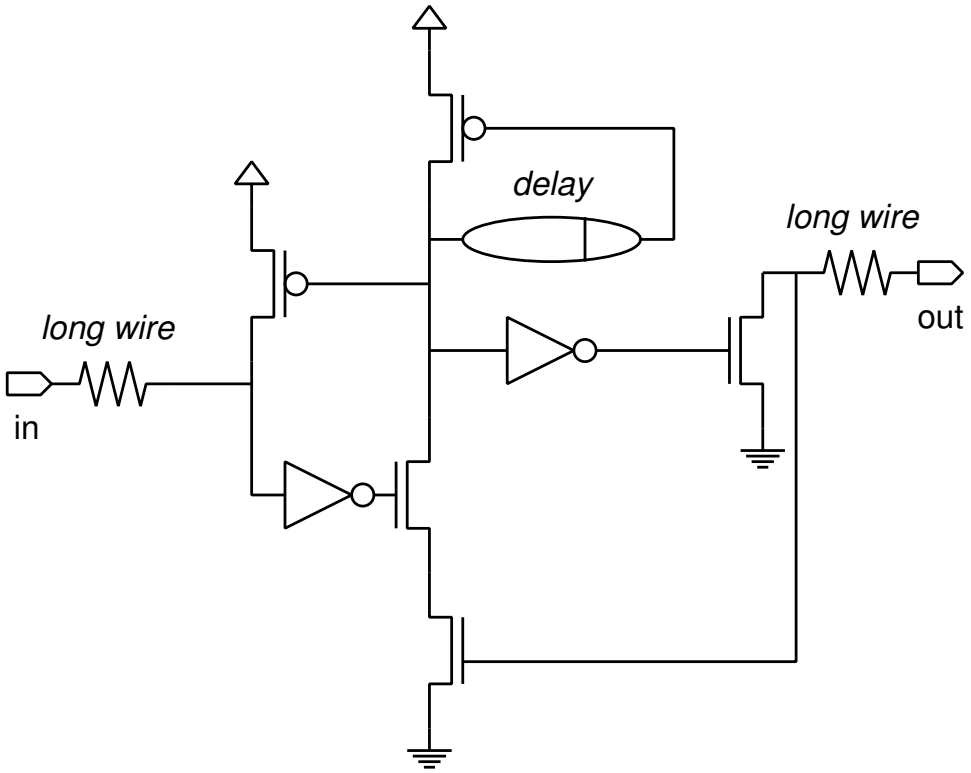


Figure 5.15: Standard GasP repeater

work [59], which inspired my point-to-point interconnect. It is essentially a single GasP pipeline control stage (albeit for a dual-rail data path), and I illustrate it as Figure 5.15. Its simplicity and high performance is obvious from the brevity of the schematic. Requiring only nine transistors in total, it's performance is very good, with a forward delay of 235ps, approximately three and a half fan-out-four delays in the 180nm technology used — high for a path transistor count of six, but note that the output driver transistor must be large and drive a large load. When loaded only by another repeater stage, latency is 198ps (=2.9×FO4) with a z0 input, or 159ps (=2.3×FO4) with an artificial, 100ps slew, voltage input.

5.11.2 Stateful repeater

I will now describe my preferred wire repeater, which is a stateful design using D-type latches. It is the component I use for repeating long wire lengths in the presented system, and I illustrate it as Figure 5.16. The repeater can be separated into three main sections:

1) Input acknowledgement and data validity detection;

2) State holding and resetting;

3) Output driving.

We can see from the schematic that the input section is more or less the same as for the other components: it consists of an input with a NAND gate detecting valid input symbols, and resetting circuitry in the form of p-type pull-up transistors. In the case of the repeater, I

have chosen to add a set of dedicated reset p-type pull-ups on the input of the second, rather than logically ANDing the reset signal into the acknowledgement p-types. This choice is made to eliminate an additional gate delay on the reverse, acknowledgement, path; I believe that the parasitic delay from the additional p-types is less than the corresponding logical overhead.

The state holding section is very simple, comprising D-type flip-flops. Of the inputs and outputs available from these devices, I use only `D`, $\overline{\texttt{PRE}}$, $\overline{\texttt{Q}}$ and the clock input. Operation is straightforward, with new data being latched into the flip-flops by upward transitions of the `d_in_valid` signal, which occurs when a new, non-idle data symbol has appeared on the inputs. Data is cleared by the application of a low $\overline{\texttt{PRE}}$ signal, which is created by a pulse generator that is triggered by the clearing of the outputs `z0` and `z1` from a valid data symbol to the idle state. As a reminder to the reader, this is caused by the succeeding stage acknowledging the data. Full D-flops are needed since a simpler element such as an RS-flop will not suffice. To understand why, we need to examine the possible orderings of the input and output cycles. Imagine that the input side is presenting a valid data bit, and the output side manages to dispatch a copy, and the receiver resets the output wires before the input side has finished clearing. Here, the use of a simple RS-latch would see the input side still asserting an input, and hence the output may never be reset. This would result in an duplicate transmission of the input data bit, and so we must use an edge-sensitive latch, such as a D-flop.

The final section of the repeater is the lightweight wire driving circuity. The active-high outputs $\overline{\texttt{Q}}$ are fed directly into wide n-type driving transistors, the strength of which affects the speed of swing on the output wires. This completes the propagation of a symbol, from input to output.

Inside the repeater, two different forms of pulse generator (or *chopper*) are used, to generate either high or low pulses. They are based on the simple edge detector circuit in Johnson and Graham's book [64, p.181], and are very similar. Depending on the polarity of their input, I use either a NOR-based design, (Figure 5.18(a)), or a NAND-based one, (Figure 5.18(b)). The delay is tunable by adding or subtracting inverters in a long chain, directly influencing the width of the output pulse. In both cases, the output pulse is active low.

This repeater design does make one timing assumption: that the output environment can consume a symbol and reset the lines at least as quickly as the input environment produces data. If this is not the case, it is possible for data to be lost.
Consider the following sequence of events:

1) The repeater begins with all lines idle (at `11`);

2) A symbol, say `01`, appears on the input;

3) It is latched, and the input is reset to `11`;

4) The complementary symbol, in this case, `10` is received on the inputs;

5) The symbol will be latched, without the latches being reset.

this pattern results in a `00` value on the repeater's outputs, which is an illegal value.

To safeguard against this erroneous behaviour, protection circuitry can be added to a stateful repeater, resulting in the schematic shown as Figure 5.17. Safety is ensured by disal-

lowing an input reset unless the output wires are at the `11` idle state. This has the side-effect of increasing the cycle time of the inputs to an arbitrary level, dependent on the speed of output consumption. At this point, the stateful repeater acts much more like a FIFO stage than it does a conventional repeater or pipelining stage.

Additional to the difference in behaviour, the safe design also imposes a latency penalty due to the additional NOR gate, and inverter (to change the NAND gate into an AND). In the technology I consider, these gates may be merged into an AND-NOR compound gate and the increase in latency calculated: with safety, the repeater design has an additional latency of 99ps and occupies an extra 9.66μm^2. The area difference is small, but the increase in latency may be unacceptable for more aggressive designs. If this is the case, I suggest that repeaters be placed non-uniformly, with decreasing spacing in the direction of data flow. This should ensure that each repeater's output environment is faster than its input one; with the result that the simpler design may be used without fear of data loss.

Often, the addition of a large state-holding element such as a D-type flip-flop adds significantly to the propagation delay of a repeater. This is true for my design as well, but the fact that its inclusion enables acknowledgements to be more locally generated than would otherwise be the case means that this overhead is less critical than the enhancement to the throughput of the overall interconnect system, and so inclusion is advantageous overall.

5.11.3 A potential optimisation

Since evaluation began, the following optimised layout has been suggested by Simon Moore. I describe it here, but do not update the diagrams, since the results we will see in the following chapter are based on the design indicated. The improved design has the D-flop fed a constant `0` value into its `D` input and, clocked instead not by the NAND gate, but by the falling edge of the data line to which it is attached. This requires a flop operating on the falling, rather than rising, clock edge, but that is easily accomplished. Performance is high, since we save a NAND gate delay, and the flop is fast, since its data input is always strongly asserted by being tied to `vdd`.

5.12 Summary

In this chapter, I have introduced a point-to-point link capable of connecting together two 8-bit synchronous parallel interfaces over a long distance on-chip. It uses just two minimally-sized, minimally-spaced global wires, and can cross arbitrary clock domains without ill effect.

I have introduced the building blocks and shown how they are composed. I have also presented the concept of a pulse-based transmission protocol, its phases and how pulses may be produced.

Long links need repeating to give high performance, and I have outlined how this may be performed with the link presented.

In the chapters that follow, I will evaluate this link, show that is has good performance and that is meets our target ASIC application. Later, I will show how the link can be scaled up to a fully-fledged Network on Chip implementation. We now continue with the remaining chapters, and see this for ourselves.

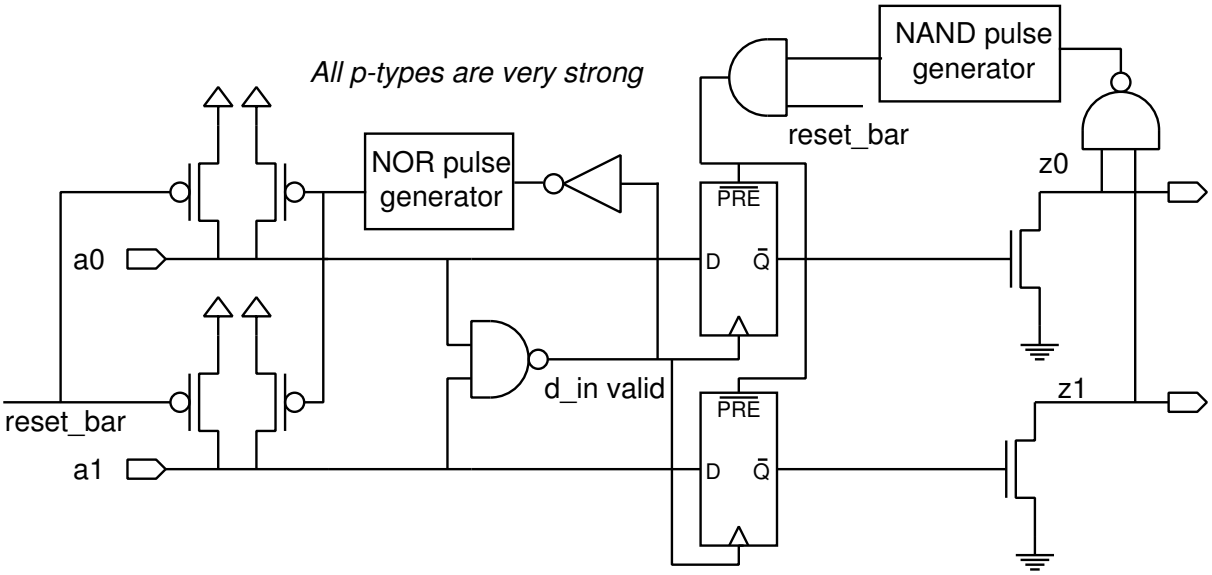


Figure 5.16: Stateful wire repeater

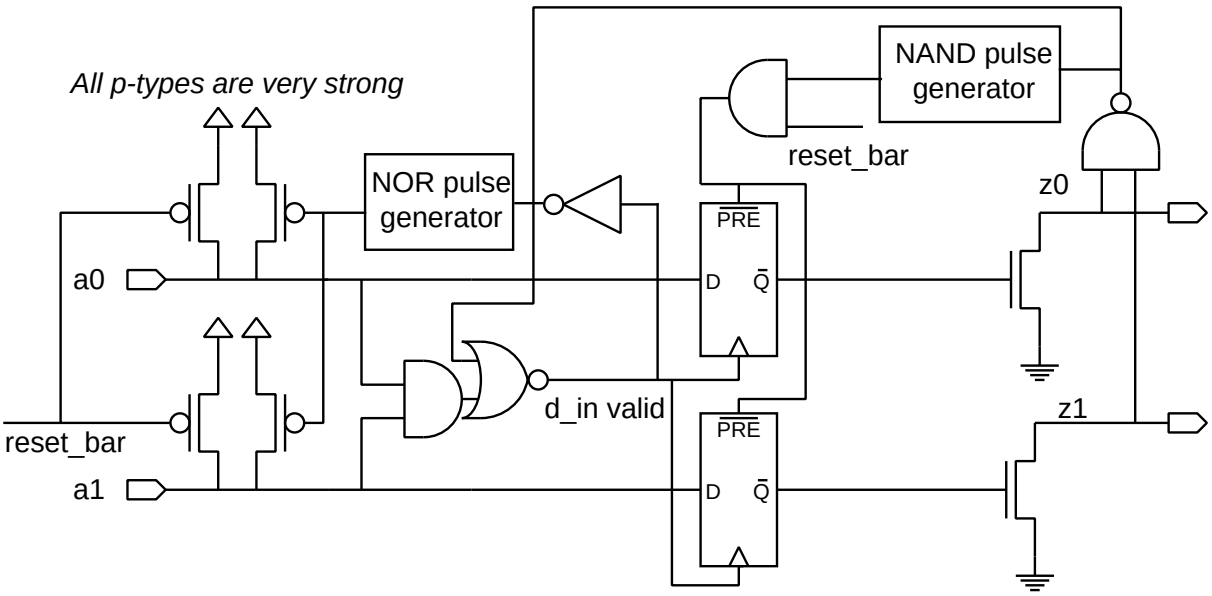
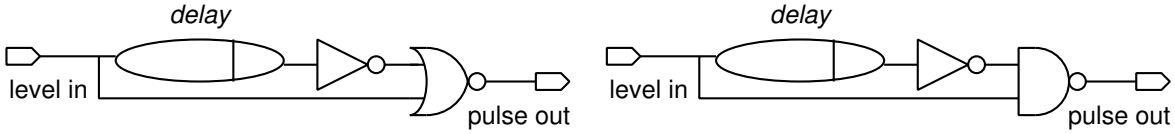


Figure 5.17: Stateful wire repeater with an output safeguard



(a) NOR pulse generator (Generates a high pulse from a low input level)

(b) NAND pulse generator (Generates a low pulse from a high input level)

Figure 5.18: Two types of pulse generator

Evaluation of the area-efficient interconnect

6

I must begin with a good body of facts
and not from a principle,
(in which I always suspect some fallicy),
and then as much deduction as you please

Charles Darwin, Letter to J. Fiske, 8 Dec 1874

6.1 Introduction

In the previous chapter, I outlined an implementation of an area-efficient interconnect. Here, I will evaluate its performance, consider some optimisations, and compare it with other designs in the interconnect space.

6.2 Methodology

To test the operational correctness and performance of the interconnect system, I decided to simulate it in its entirety. Since the system involves both analogue and clock-less components, a conventional hardware description language such as Verilog [63] or VHDL is inappropriate (there do exist variants of these languages with support for analogue components, e.g., VHDL-AMS, but the lack of a global clock makes circuit description problematic).

Taking all of this into account, I decided that the best solution would be to describe and simulate in the analogue component simulation language *spice*. The spice languages are the most developed and numerically stable such tools. Of the family, *hspice* [62] is the most popular and highly accurate instance, and so I chose to use hspice for the simulations of my system.

In spice, all components are described as instantiations of a basic component, created with a netlist and a basic component name (with optional parameters).

Spice lacks modern structure support, such as objects or modules, so the nearest thing is a 'sub-circuit', a circuit declaration which may itself be instantiated.

In general though, spice is very low level and the verbosity of declarations, combined with the need for unique names and the importance of declaration case and ordering makes it a language unsuitable for large systems. Despite these shortcomings though, it is still the best tool for the job of simulating my system, albeit with an increased development time compared with higher level languages.

Hspice is capable of outputting results in a number of formats, as textual measurements or as graphs of the voltages or currents of various nodes through time. The latter is the most useful tool for debugging operation, and is also a good way of verifying the correctness of

operation of wires, units and other components. Some of the graphs presented in the section have been generated in this way.

Simulation performance

Simulations of a complex system in hspice are slow. The basic point-to-point interconnect design contains around 1000 nodes, and simulation of a single word transfer takes around 45 minutes on a modern PC. For a sweep over the length space between 0μm and 10,000μm, a simulation run takes around 12 hours. These times are for simulation runs with a maximum *time-step*, or resolution, of 1ps. In an attempt to reduce the run time, coarser resolutions were experimented with. It was found that, if the resolution was decreased from 1ps to 5ps, the speedup is significant (around five times), whilst results varied by only plus or minus one at the third significant figure. This gives an decrease in accuracy of less than one per-cent, but a great deal of a speedup. Therefore, a resolution of 5ps was used throughout.

6.2.1 Wire model

A good wire model is vital when simulating long-distance interconnect. Not only does it determine the signal delay, but also other effects such as the crosstalk and noise injected and the shaping of signals. In Chapter 4, I introduced these concepts and showed how a model could be accurately extracted. I performed an extraction for a minimally-spaced wire array in metal 5 in our 180nm technology, in order to generate an accurate model for spice simulations. The wire physical model was as seen in Figure 4.2, and metal 5 is considered a 'semi-global' wire in this technology. The choice of minimally-spaced wires is a product of the conclusions of both Ho [21] and Li [38], and Chapter 4: namely that interconnect bandwidth is optimised when wire spacing and width are both set to their minimum values. Therefore, I attempted to create an effective interconnect under this constraint.

The model's spice description is included as Appendix A. I will not list the full details now, but will say that it is based on the distributed π model (see Figure 4.5), and models wires as bundles of up to five traces, cross-coupled via inductance and capacitance, and sharing a common reference plane. This models our setup perfectly, since a dual-rail implementation uses two wires, routed together. The spice model definition scales automatically to any wire length, so the performance of a range of interconnect lengths can be simulated without the need for further extraction or alterations to the model.

Displayed in Table 6.1 are the extracted values per metre for resistance, capacitance and inductance for a two wire model. The resistance is 221kΩ/m. Were they to be for a metal 6 wire, the resistance would decrease to 93kΩ/m, but the self capacitance increases to only 238pF/m from 224pF/m; and the mutual capacitance from 88pF/m to 100pF/m, a total increase of only 8%. Therefore, the RC constant is much smaller than for metal 5 (only 45% of it over 1mm), as is to be expected for a top level wire. To place the metal 5 values in context, Horowitz et al. [27] claim that a 'mid-layer' metal (as metal 5 could be considered) wire will have a resistance of 0.11Ω/μm(or 110kΩ/m), but use 50% wider wires than presented here. Therefore, their estimate extrapolated for use here would be around 165kΩ/m, not too distant from the extracted value shown. In their later paper, [23] they update the value to 96kΩ/m for a 'semi-global' wire

Table 6.1: Extracted wire model parameters (0.18μm, two wires, min. width, min. spacing)

Parameter	Value (per metre)
Self-capacitance of line 1	224pF
Self-capacitance of line 2	224pF
Resistance of line 1	221.43kΩ
Resistance of line 2	221.43kΩ
Self-inductance of line 1	1.652μH
Self-inductance of line 2	1.652μH
Mutual capacitance	88.2pF
Mutual inductance	1.428μH

Cong and Pan [11] give a more accurate estimation based on NTRS [1][†] figures. They claim a sheet resistance of 0.0679$\Omega/\square$ at the 0.18μm node. This would give a resistance for my wires of 242.5kΩ/m.

Similarly, both publications give values for capacitance at the 0.18μm node. Horowitz et al. state that capacitance is 0.22fF/μm, and Cong and Pan claim (0.0596fF/μm^2+ 0.0641fF/μm). These equate to 220pF/m and 81pF/m respectively. Note that Horowitz et al.'s result is in excellent agreement with my data for the self capacitance of the line. Also, in their second paper, Ho et al. include coupling capacitances to generate a total wire capacitance of 414pF/m, which compares well to my total extracted value of 312pF/m. My value is smaller since I consider actual wire placements on neighbouring planes, rather than parallel grounds as normally assumed. Not presented in Table 6.1 is the percentage of capacitance I see coming from intra-layer cross-capacitance, but the extracted figure is 79.8%. This compares with the 76% Ho et al. publish [23].

Therefore, given there are published results erring on either side of my extracted figures, I have confidence that they are very close to the real world values.

One striking result is how large some of the values actually were. For example, trace self inductance was a huge 1.65μH per metre, and self capacitance was 0.22nF per metre. These are very large, and so capacitance and inductance will contribute strongly to signal degradation and crosstalk. Mutual inductance was very similar in value to self inductance, whilst mutual capacitance was nearer a third of self inductance. These, of course, mean that cross-coupled noise is likely to be fairly strong.

Since the wire model includes a large inductive component, we must see if we need to consider all interconnect behaviour in the context of a full RLC model. This model, which was introduced in Chapter 4, tells us a few important things about wire operation. The most important for consideration in this chapter is that pure RLC operation involves a linear propagation delay with length, as opposed to the behaviour of sluggish RC wires, where propagation delay scales quadratically the longer the line. There is not a clear changeover between a line's operation in the RC and RLC regions, but rather a continuum over which one effect becomes more and more pronounced, as the other's influence wanes. For this reason, it is best to consider the line as fully RLC. However, if we can determine that we are

[†]The 'NTRS' (National Technology Roadmap for Semiconductors) was the precursor to the well-known ITRS (International Roadmap for Semiconductors).

modelling the degenerate case, when inductance is insignificant, we will be able to assume RC behaviour.

Let us now check whether inductance should be significant for the interconnect used here, or whether we can safely treat its effects as negligible. We saw in Section 4.2.4 that is is possible to perform two rule-of-thumb calculations to find out. We do these now, for the shortest interesting length in our design of 1000μm:

1) Is,

$$\frac{R}{2}\sqrt{\frac{C}{L}} > 1\,?$$

For 1mm, with our model we get values of $110\sqrt{\frac{3.12\times10^{-13}}{3.08\times10^{-9}}} = 1.11 > 1$, so resistance is significant (and grows in significance linearly with length).

2) And is,

$$\text{rise/fall time} > 2\sqrt{LC}\,?$$

We see that $2\sqrt{LC} \simeq 6$ps, so yes again, since even moderately loaded gate delays are well in excess of this.

Both answers are 'yes', allowing us to assume that inductance is not significant for our model over lengths above 1mm, the range we are interested in. Thus, we will ignore its effects during calculations.

The interconnect 50%–50% of `vdd` delay values are illustrated for the model in Figure 6.1, where we see the wire delay rising quadratically with length. This is exactly as expected with an RC wire model, and concisely illustrates the problems faced by designers when creating long wires: the delay rises sharply with additional length. We can now use Equation 4.13 to verify the figures for the propagation delay shown in Figure 6.1, where we see the value for 1mm is 35ps.

6.2.2 Optimal repeater insertion

We saw in Section 5.11 that the delay of a line can be minimised by the technique of optimal repeater-insertion [21; 31]. Now, we consider what the best sizings are for such repeaters.

To do this theoretically, we use the following equations from Weste and Harris [65, pp.222–223], when the repeaters are single inverter elements:

$$l_{segment} = \sqrt{\frac{2\hat{R}_{inverter}\hat{C}_{inverter}}{\hat{R}_{wire}\hat{C}_{wire}}} \tag{6.1}$$

and,

$$w_{inverter\_output} = \sqrt{\frac{\hat{R}_{inverter}\hat{C}_{wire}}{\hat{R}_{wire}\hat{C}_{inverter}}} \tag{6.2}$$

The inverter characteristics are for a unit-sized inverter, and the wire per unit length. Those for non-inverting buffers are used are similar, but involve a cascading factor, k, representing the ratio of transistor widths between the first stage inverter and the output one. It is suggested

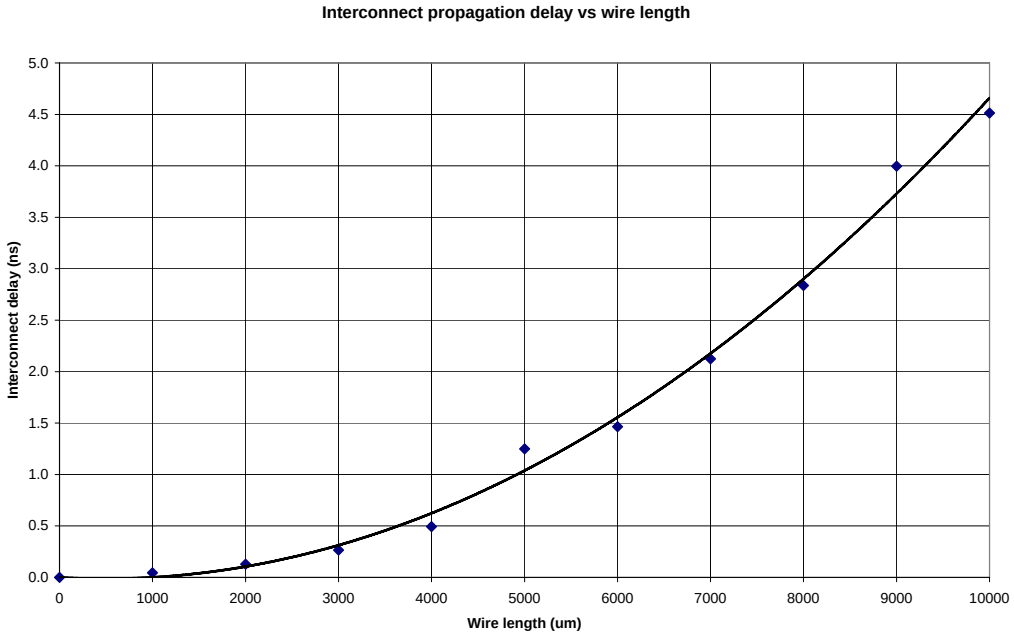


Figure 6.1: Rising propagation delay with length for our wire model

that this should be very close to two as an optimal value, but extraction of our technology suggests that three is a better selection, and so k is set to three for our system (see Figure 6.3).

$$l_{segment} = \sqrt{\frac{2\hat{R}_{inverter}\hat{C}_{inverter}(k+\frac{1}{k})}{\hat{R}_{wire}\hat{C}_{wire}}} \tag{6.3}$$

and,

$$w_{inverter1} = \sqrt{\frac{\hat{R}_{inverter}\hat{C}_{wire}}{\hat{R}_{wire}\hat{C}_{inverter}}}/\sqrt{k} \;\; ; \;\; w_{inverter2} = \sqrt{\frac{\hat{R}_{inverter}\hat{C}_{wire}}{\hat{R}_{wire}\hat{C}_{inverter}}} \times \sqrt{k} \tag{6.4}$$

I remind the reader that $\hat{R}_{inverter}$ and $\hat{C}_{inverter}$ are the unit inverter resistance and capacitances. Similarly $\hat{R}_{wire}$ and $\hat{C}_{wire}$ are the per-unit length wire resistance and capacitances.

During development, I used these equations to give a first approximation to the transistor sizings for repeater insertion. Further information was then gained by performing sizing sweeps, centred around the formulæ's results. These are described in the next section.

6.2.3 Driving transistor characterisation

In order to determine the exact relationship between driving transistor performance and wire swing times, we need a little more information in the form of transistor characterisation. Since we do not have the values of the process constants used in equations 4.38 and 4.39 to hand, yet we should like to know the relationship between the current sourcing ability of a transistor and the voltage presented to its output, we need to extract transistor characteristics by another method. Known as I-V characterisation, we can do this by considering the drain-source current (I_{ds}) as a function of drain-source voltage (V_{ds}). Given that our driving transistors are

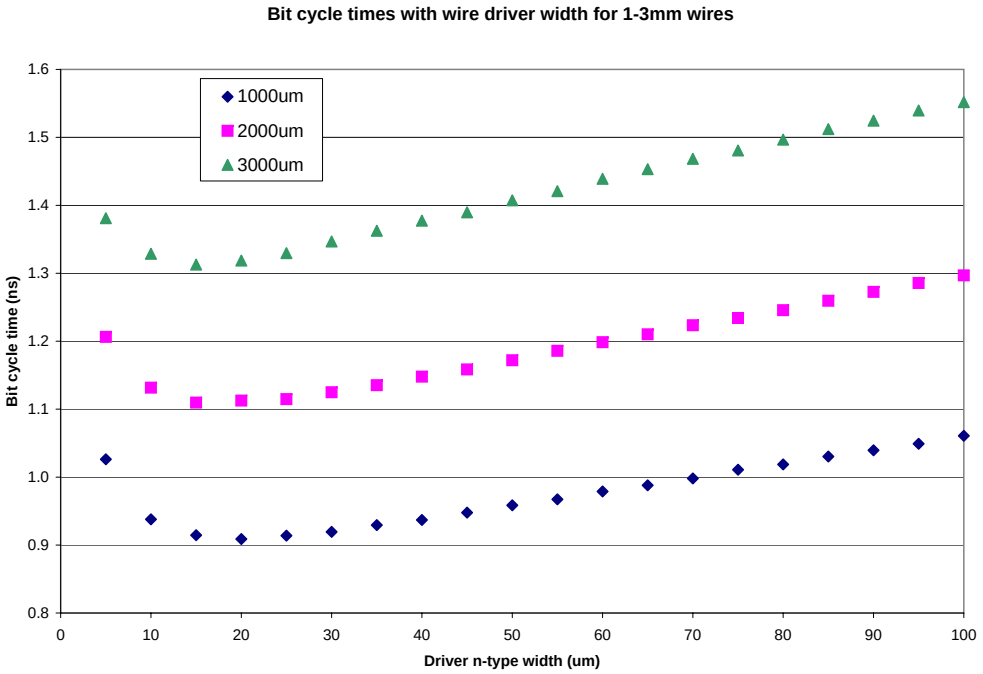


Figure 6.2: Wire driving transistor width sweep: 1–3mm wires

driven strongly by logic gates, which should rapidly (with respect to the interconnect wire) switch between high and low, we need only consider I-V for a fixed gate voltage; `vdd` for an n-type, and 0V for a p-type.

Following simulation of the I-V characteristics, I plotted the values for the technology used. The results follow the well understood I-V curve [65, pp.293–296], involving an initial linear region, followed by a saturation region. We wish to know the crossover point, V_{dsat}, between these two, and inspection suggests that it occurs for the n-type transistor at at V_{ds} of approximately 0.6V, and a V_{ds} of -0.5V for the p-type transistor. These equate to both drivers working in saturation when the interconnect voltage is in the range 0.6–1.3V. If the significant majority of our interconnect's operation occurs in this range, we can treat the drivers as always working in saturation, which will simplify our analysis.

As we will see in more detail later, when simulating our interconnect with very short wires, excursions are observed from `vdd` to ground. However, simulations show that an average case (5000μm length wire) maximum voltage swing is between 0.2–1.3V, with approximately two thirds of the total time of a pulse being in the 0.6–1.3V range.

Recall also that the useful swing of an interconnect is the range which causes transitions between the two threshold voltages of its receiving buffer's transistors, in order to transmit information. Simulations show that a value is considered logic 'high' once it passes 1.12V, and 'low' upon falling below 0.61V. We have just seen that the entirety of this range corresponds to drivers in saturation, and can assume that both the p- and n-types are in saturation, for our entire period of interest. Therefore, we can consider them as constant current sources (refer to Equation 4.39) without incurring much loss of accuracy. So, we need to check that they are sufficiently large that their output impedance is much lower than that of the wires, to ensure their impact on delay is minimal.

We could do this from the equations in Section 6.2.2, but good as theory is, it is often more instructive and accurate to perform a parameter sweep for a design such as this. This way, and additional factors not shown in the model can be accounted for, and an optimal design rapidly chosen.

I performed such a sweep of driver size versus interconnect performance, and I display the results as Figure 6.2.

We clearly see the optimum driver width is 20μm. The reason the optimum size reduces with longer wires is that the output resistance is not on the critical path of the wire delay, yet the larger input capacitance of a wider transistor adversely effects the receiver logic delays via reduced slew rate. We see that the performance at the 20μm size is very nearly optimal for all three wire lengths, and so it is an excellent choice for our driver width, and so it is fixed at this size for all simulations. p-type transistors use double the width, at 40μm. Many implementations use p-transistors of three times the width of their n-type counterparts, to create an inverter with balanced pull-up and pull-down resistances. However, we find here that the increased input capacitance when widening p-types from two to three times slows impacts overall performance more than output resistance. Thus, we find that p-types twice the width of n-types are optimal for our performance. Further, the reader may be interested to note that this sizing is different from the conventionally-taken value of $\sqrt{2}$ for a repeater's inverter since the p- and n-types are independently driven, and so we do not consider their combined delay, as for a normal inverter.

6.2.4 Output buffer cascading factor

Transistors as wide as 20μm offer a high loading to any logic driving them. In order to keep transitions sharp, their inputs must be reinforced by a well designed output buffer.

The first stage in designing an output buffer is to set the inter-stage cascading factor, k. This is the ratio of inverter sizes between subsequent stages of a multi-inverter buffer. Logical effort calculations typically set this value to be as close to e, the natural number, as possible, with the nearest integer being two. However, since the theory does not cover all real-world parameters, a preliminary sweep was performed to select the optimal value.

To ensure that the simulations were accurate, an artificial input slope was first passed through a set of inverters to give it a more realistic slew profile, before encountering a cascade of four inverters, each a factor k larger than the previous. The output of the cascade drives an RC load, modelling a wire of length 1000μm.

The optimal value from these simulations can be determined from Figure 6.3, where we see the output from such a four-inverter buffer with different values of k, from two to five. We see that the earliest and sharpest transition (i.e., the best) occurs with a value of $k = 3$. Therefore, buffers using this scale ratio were inserted into wire driving components between their outputs and the wire driving transistors, to efficiently drive the latter.

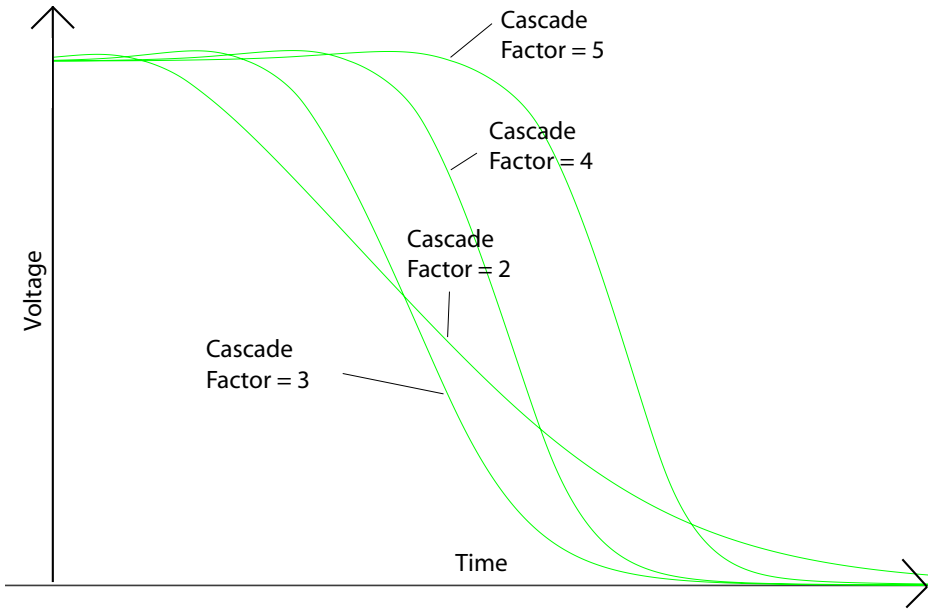


Figure 6.3: Inverter cascading factor sweep. Note the optimal value is three.

6.3 Simulation of correctness

Using the results from hspice simulations, I am able to confidently say that the area-efficient interconnect solution proposed in the previous chapter does indeed operate correctly. By 'correctly' I mean that a transmitted data word is received error-free, with a high probability by the receiving environment. As an illustration of correct operation, I show as Figure 6.4 the plot of the outputs of the output pulse latches, compared for an input stimulus of `00001011`, transmitted via the dual-rail MUX-DEMUX base-link, with the distributed inverter running over 1000μm wires. The outputs are dual-rail, and so consist of two wire per bit, suffixed '0' and '1' (refer back to Section 5.1.1 and Table 5.1 for a reminder of the dual-rail protocol). Assertion of exactly one of those wires to high corresponds to the corresponding logical value being transmitted. For example, the penultimate trace, labelled `out0_0` transitions high around a third of the way across the plot. This indicates that bit 0 equals logic `0`.

6.3.1 Test results

Figure 6.4 that the data values received correspond exactly to those transmitted, with reception of each bit offset in time. Transmission occurs in the order $d_0 \ldots d_7$. Upon reception of d_7, the `data_out_valid` control line (third trace) is asserted, to inform the receiving environment that a full data word has been received correctly. Also displayed are two more control signals: `new_data_bar` (top), sent low when the input environment supplies a fresh data word; and `input_fifo_empty` (second), which indicates back once that word has been consumed and the MUX tree is prepared to accept a new word. Note from the traces that this particular signal transitions to low (the FIFO empty state) just before bit six is successfully seen at the receiving end. This indicates that, at this point, bits six *and* seven are simultaneously in flight

through the interconnect structure. However, the transmitting environment has been pre-emptively instructed to refill the transmit buffer. This pattern of control signal cycles enables the transmit buffer to be refilled with minimal delay, and increases the overall throughput of the system.

Test data value

The transmitted data value of `00001011` was chosen since it contains both rapid value changes (`0101` sub-word) and runs of length one, two and four of the same value (`0, 11, 0000`). I discovered early on in simulations that these two extremes of bit run-length caused the most unreliable operation, and so if a simulation is successful on the shown data word, it will most likely be reliable on all others.

I believe the reason this word is the most error prone is that rapid changes may occur before a previous value has had time to settle, and may cause data corruption, or even a `1` and a `0` to be transmitted simultaneously, resulting in an erroneous output.

On the other side of the fence, large runs of the same value have been observed to cause a drifting of the bias level of the passive (non-transmitting) wire away from its idle value towards its active one. Over multiple bits, the drift accumulated and sometimes passed the receiver's threshold voltage, again causing the simultaneous transmission of a `0` and a `1`; and an error.

Since high accuracy simulation run-times were in the order of half a day, it was not possible to perform an exhaustive search to demonstrate correctness with all bit patterns. Therefore, it is this author's belief that the word presented above is the best solution to the problem of fault coverage in the base link system, and that the results from a simulation with it are reliable indicators of the correctness of system operation for all data words.

Figure 6.4: The outputs and control signals for a transmitted data value of `00001011`

6.4 Point-to-point link results

We will now see some results from an implementation of the point-to-point link, as presented in Chapter 5. Ultimately, it will consist of a synchronous data environment, feeding a MUX tree, then data being passed over a correctly repeated wire, arriving at a DEMUX tree, before finally being passed to a different synchronous receiving environment.

First, though, we test the limits of performance of the pulse-based dual-rail protocol. To do this, we configure the fastest possible free-running system: a loop oscillator.

6.5 A loop oscillator

Since our repeaters are non-inverting, we can configure any number N of them in a loop to produce a free-running oscillator where, for a single circulating data item, the frequency of transition at any point is equal to one upon the sum of the delays of all the repeaters, plus the delays of all the wire segments connecting them together. For a regular system, the delay of a single repeater and wire can thus be found by dividing the sum by N.

This configuration, illustrated in Figure 6.7, is perfect for testing the maximum performance of our point to point system. In particular, the ease of tweaking the width of the pulses driven and acknowledged by each repeater allows us to test the minimum pulse widths required to successfully transfer data over a variety of wire lengths. Shown in Figure 6.8 is a graph of the pulse widths needed to drive data transfer along different wire lengths.

Due to the slightly different lengths of the logic paths in the data and acknowledgement pulse generation circuitry, it is not possible to exactly match the widths of data and acknowledgement pulses, and the latter is always shorter by approximately 0.5–1×`FO4` (30–60ps). For this reason, the acknowledgement pulse width is more critical than its data counterpart, and all data graphed is with relation to it. Before we proceed to the results discussion, please take the time to familiarise yourself with Figure 6.5, which illustrates a cycle of the protocol, and the naming convention we will use to denote its various phases.

Throughput

There are several possible measures of throughput for our link, dependent on which section(s) of the interconnect one is considering. The simplest, and perhaps most informative is the *bit cycle time*. This metric indicates the amount of time elapsed between the beginning of one bit being sent to the beginning of the next, all as observed at the transmitter side. Measurements are triggered by downward passings of `vdd/2`.

We see the bit cycle times (the time for a two way `data`→`ack`→*start of the next data bit* handshake to be performed between two neighbouring repeaters) and the time taken for a single data bit to travel completely around the loop (from hereon in, the *end-to-end latency*[‡]). Both have been swept with the length of inter-repeater wires, and are displayed as Figures 6.10 and 6.9, respectively.

An important note is that, for each wire length setup, the pulse widths have been tuned to provide the highest possible performance. Figure 6.8 shows how widths increase with wire length, from as little as 233ps to a whopping 2.90ns. As expected, after an initial plateau, where

[‡]The reader may note with a smile the irony of an 'end-to-end' metric for a circular configuration, but we use it here to maintain compatibility with other interconnect evaluations.

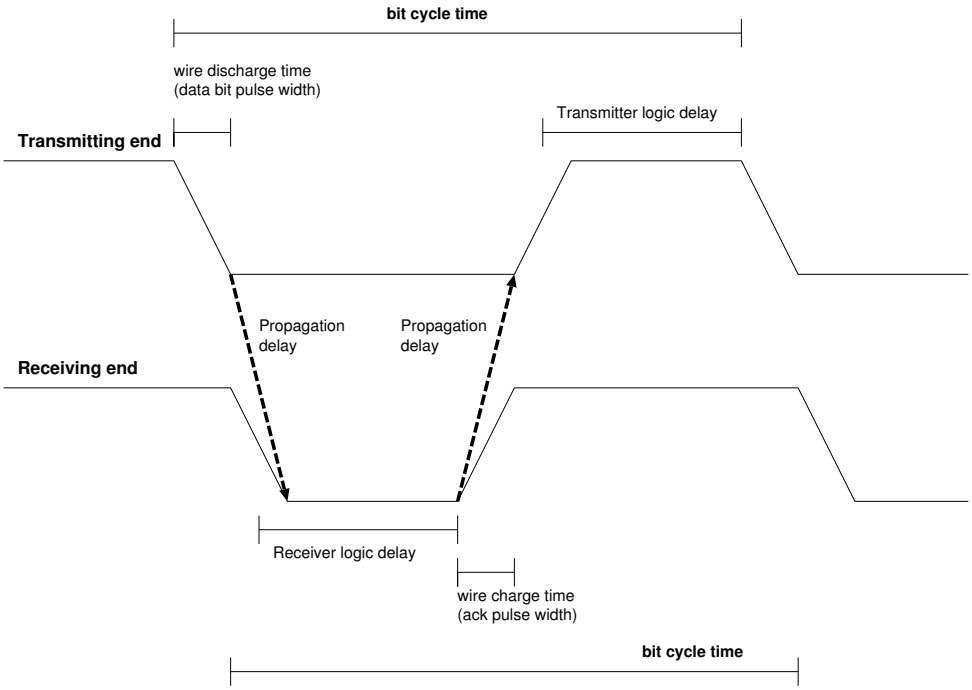


Figure 6.5: The major events during the pulse-based signalling protocol

the minimum producible pulse width is able to service a multitude of short wire lengths, increase is quadratic, as determined by the wire charging delay. The 'plateau' section indicates that, for a minimum pulse width of 233ps, we gain reliable operation in the wire segment length range of 1–3mm.

We see both pulse width and throughput are influenced by the increasing effect of increasing wire delay with length. Rising, as it does, quadratically with length, it impacts end-to-end latency and bit cycle times in a similar fashion. Bit cycle times rise from as little as 755ps at 1mm (which equates to 1.33Gbit/s of throughput) to 5.65ns for a 10mm wire (only 177Mbit/s).

It is hard to discern an exact pattern by eye since the data past the 3000μm point may be linear or quadratic. Data with wire lengths of less than 3000μm shows similar pulse width values since, due to logic delays in the system, it does not operate at all with pulse widths under 233ps, and an $RC/2$ line charging delay of only 60ps. Therefore, all wire lengths for which this width is sufficient (i.e., the 1000–3000μm range) show up as requiring this pulse width when they may, perhaps, accept a lower value.

Therefore, if we discard these data points, we can perform a regression on the remainder to attempt to find the shape of the graph: and both lines fit a quadratic with values of $R^2 = 0.999$, much higher than those of a linear regression, suggesting that the pulse charges the wire in the RC mode.

Latency

Latencies follow an almost identical pattern, but, since they involve a data bit travelling all the way around the ring, have a larger magnitude, in the range 2.74–15.5ns. This comprises four wire delays plus four repeater forward delays, whereas the bit cycle time is made up from two wire delays and a repeater data and acknowledgement time. This explains the difference in

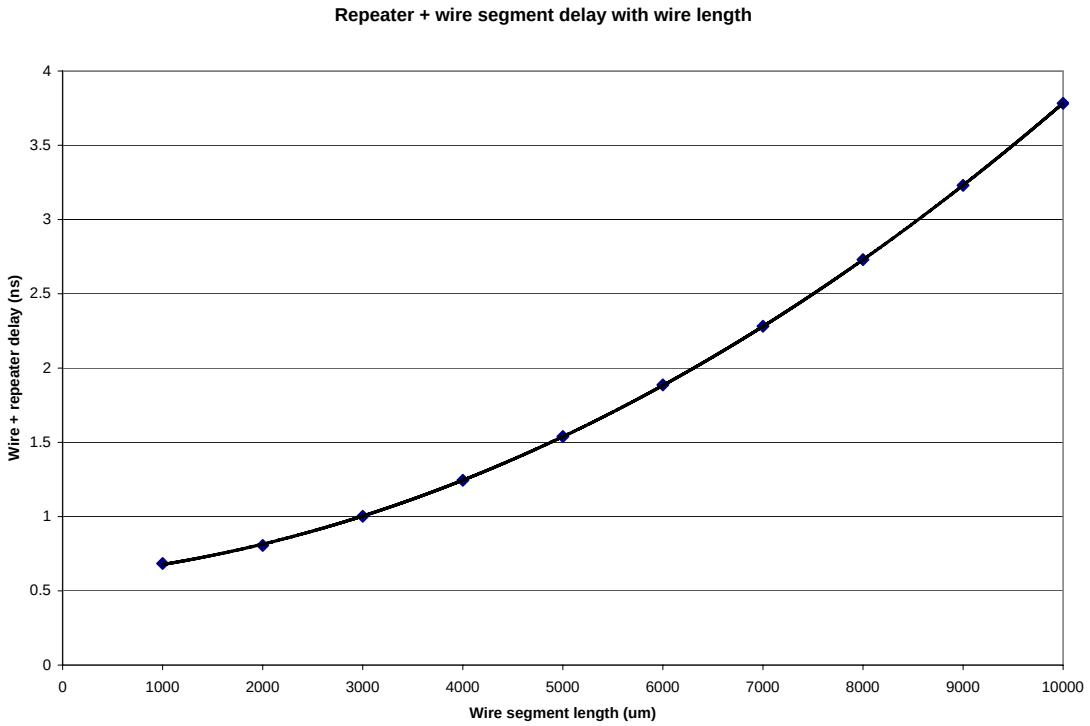


Figure 6.6: Observed segment delay (total wire + repeater forward delay upon four) for various wire segment lengths in an four repeater oscillator loop

scaling and allows us to say something about the relative importance of the repeater logic and wire delays — the former is more important at shorter wire lengths, but the wire eventually dominates.

We find that the total delay of a single repeater and wire segment grows quadratically with with length. In Figure 6.6 we see a plot of this delay for input pulse widths providing maximum link performance. The form is quite clearly quadratic, and a regression even provides an almost-unheard of value of $R^2 = 1.000$, implying perfect matching. If all the delay increase with length can be attributed to wire delays, then the regression even gives us the fixed logic delay of our repeater element: 592ps. However, we might realistically expect the repeaters to slow with wire length, since their inputs rise more slowly, and so should their internal signals. Therefore, the perfect correlation may be more of a coincidence, but still strongly suggests that the wire delays are quadratic, and dominating.

6.5.1 Additional repeater insertion

Since our interconnect may reasonably be expected to run over several millimetres (the edge of a large ASIC die may extend to one centimetre or more), and we have just seen that performance plummets as soon as we scale up to such lengths, we require a strategy for increasing performance. Fortunately, repeater insertion [21; 65, pp.221–226] comes to the rescue.

From the metrics we have seen above, we can calculate when the insertion of an additional repeater element will increase overall performance over that of the nominal four repeater loop configuration. Terminology has the potential to get confusing here, with an overloading

of the word 'repeater', so we will use the convention of 'repeater insertion' referring to the insertion of additional intermediate repeater elements between the original loop repeater 'nodes'.

Insertion for latency

First we can formalise when performance is actually increased by repeater insertion. It occurs when:

$$2 \times \text{wire delay}\left(\frac{l}{2}\right) + \text{repeater logic delay} < \text{wire delay}(l) \tag{6.5}$$

Since wire delay increases quadratically with length, it intuitively makes sense to keep wires short. However, what is the optimum length?

To get a feel for the answer, let us look at a few test cases. First we need the information, gained from multiple simulation runs, that the forward logic delay of a stateful repeater is more-or-less constant at around 600ps. Then we are in a position to see when this overhead is justified by savings in wire delay.

We approach this calculation in two ways: theoretical and results-based. Our theoretical version considers the inequality 6.5, and when the left-hand side is satisfied.

We know that repeater logic delay is 600ps, so we can re-arrange the equation to find when:

$$\text{wire delay}(l) - 2 \times \text{wire delay}\left(\frac{l}{2}\right) > 600\text{ps} \tag{6.6}$$

Additionally, Figure 6.1 indicates the value of wire delays with length, so we need only find two equally-spaced points where the value of interconnect delay changes by 600ps. Given the poor resolution of this graph, an alternative calculation remembers that the wire delay over 1mm is 35ps, rising quadratically with length. Both inspection and calculation give somewhere in the range $l = 5000$–6000μm to satisfy Equation 6.6. By calculation, we have that $l = 5000\mu$m yields a difference of 437ps, and $l = 6000\mu$m gives $36(35) - 2 \times 9(35) = 630$ps. Therefore, the optimal value of l for additional repeater insertion is within this range.

Our second method of calculating this value comes from the results presented in Figure 6.11, where end-to-end latency is shown for loop oscillators containing four, eight and twelve repeater elements, for a fixed range of total loop wire lengths. We observe the following:

1) At a 4000μm length, the end-to-end latency of the loop oscillator is 2.74ns. We also know that the wire delay is four lots of 44ps. Inserting a repeater here clearly makes no sense, since it could save at most 22ps per segment, but at a cost of four times 600ps.

2) At 24mm (magenta arrow), we see the line for latency of an eight repeater loop duck below that for the four repeater version. This suggests that, for per-segment lengths of greater than 6mm, additional repeater insertion is useful.

3) At 32mm (green arrow), the same becomes true for a twelve repeater loop. This corresponds to double repeater insertion per segment. And, since $32/3 \simeq 24/2$ the increase is roughly linear.

4) Whilst there is insufficient data to say for certain, it looks likely that the eight- and twelve-repeater lines will cross over at somewhere in the 44–48mm range, thus suggesting that the repeater insertion relationship is a linear one: whenever a segment exceeds 6mm, we should insert another repeater to keep latency under control.

So, we see that the optimal time to insert a repeater for latency reasons must be when total segment length is somewhere around twenty-four millimetres, and this corresponds to optimal performance where wire segment lengths for the oscillator are at most 6mm. Thus, we draw the design rule that we should never create a system with an inter-repeater gap larger than 6mm.

Insertion for throughput

Repeater insertion does not only reduce latency, but can also be used to boost throughput. Happily, the analysis for throughput is trivial. Throughput is simply equal to one upon the bit cycle time, since one bit of data (and its corresponding acknowledgement) is transferred every cycle.

Figure 6.10 shows how bit cycle times vary with wire lengths, from 750ps for a 1mm segment and rising quadratically to 5.65ns for a 10mm one. Throughput, therefore, goes as its inverse.

Equally simple is a consideration of when inserting an additional repeater makes the oscillator increase in throughput — always. This is very unlike conventional interconnects where data must flow end-to-end in a single clock cycle, and so inserting additional logic reduces throughput (by increasing the end-to-end delay) at the point where it outweighs the reduction in wire delay a repeater may afford.

With my system, however, inserting an additional repeater reduces the wire length and leaves the 'per-hop' logic delay unchanged. Therefore, the cycle time always decreases. The reader may wish to liken this approach to deeper pipelining being applied to a logic path — operating frequency always increases (albeit with a law of diminishing returns) with the addition of more pipeline stages; but so does latency, and this is the same situation we have here.

The positive impact of repeater insertion is vividly illustrated in the next section, and in particular a forward glance at Figure 6.29 will give an idea of the improvements available, under a law of diminishing returns.

Summary

We have seen how a loop oscillator composed of stateful repeaters operates over various wire lengths. Insertion of additional repeaters to further pipeline long wire segments always increases throughput, but may either increase or decrease end-to-end latency, dependent on the segment length and the number of repeaters being inserted. For latency purposes, we have seen the optimal configuration is to constrain the distances between two pipeline stages to the range 0–6mm.

6.5.2 Comparison to published literature

Ebergen, Furber et al. perform a similar evaluation to the one just presented for a pulse-based interconnect system [15], and plot the maximum wire length vs pulse width, which I reproduce as Figure 6.13(a). I use a cascading factor of three, and so my trace corresponds to the black line on the results they display. To ease comparison, Figure 6.13(b) re-plots my results with the axes swapped to map to those they use. Note that their results are generated from an eleven stage repeater ring, presumably since they deal with shorter wire lengths,

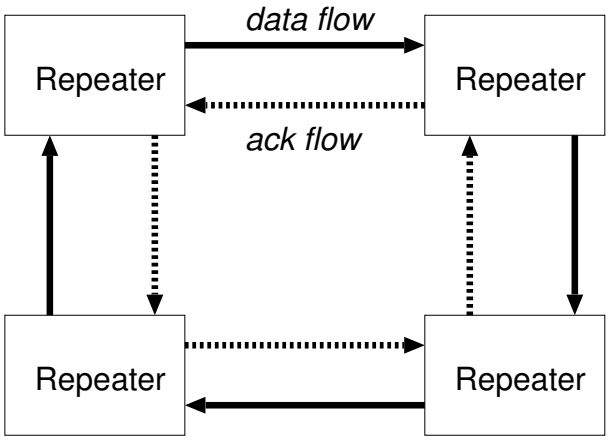


Figure 6.7: Four repeater oscillator loop configuration

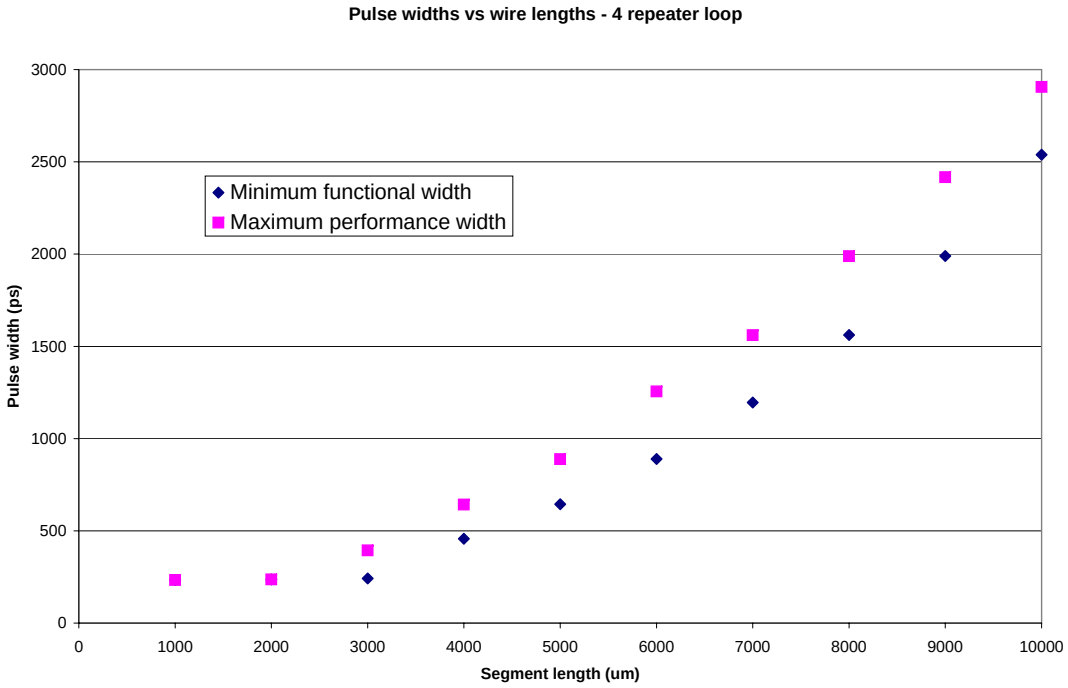


Figure 6.8: Required pulse widths for various wire segment lengths in an four repeater oscillator loop

they require more stages then used here to increase the accuracy and provide stable initial conditions. This difference should not significantly affect the comparison since, in both cases, we are evaluating a single stage's maximum performance.

Looking at their graph, it is impossible to be certain about the best regression without accurate data, but their plot does indeed seem to have the same form as ours, including the cluster of vertical data points at the extreme bottom end of wire length (I explained how these come about in the previous section). Note, however, the difference in scale: my results are over the wire range 1–10mm, whereas theirs are only in the range 0.5–3mm; the upper

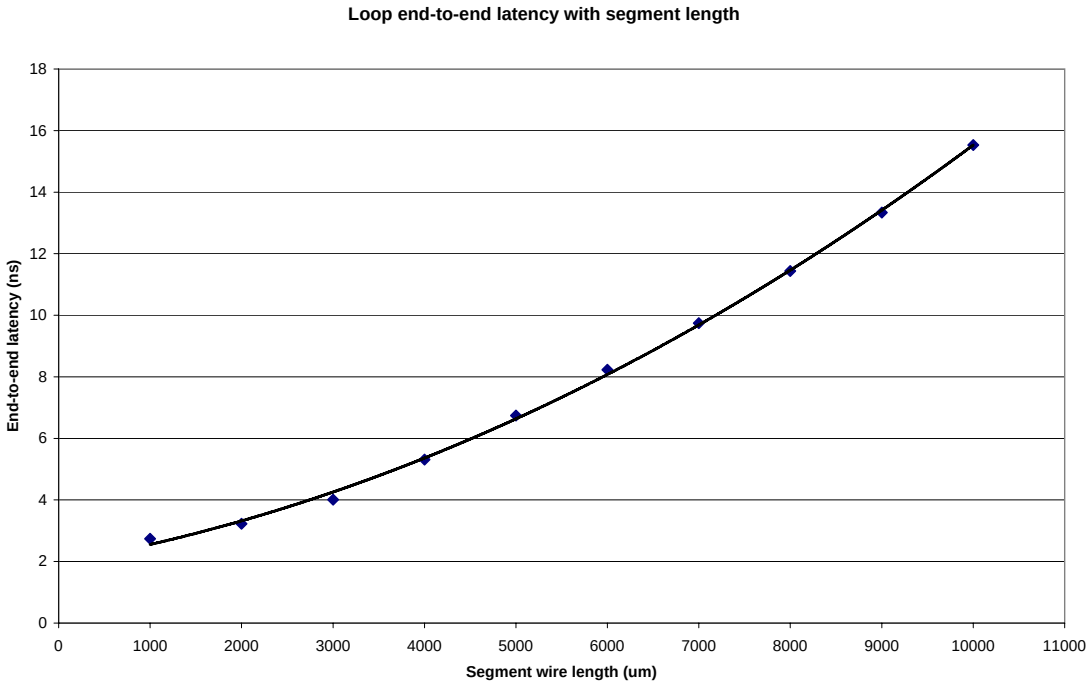


Figure 6.9: End-to-end latencies for full loop traversal for various wire segment lengths in an four repeater oscillator loop

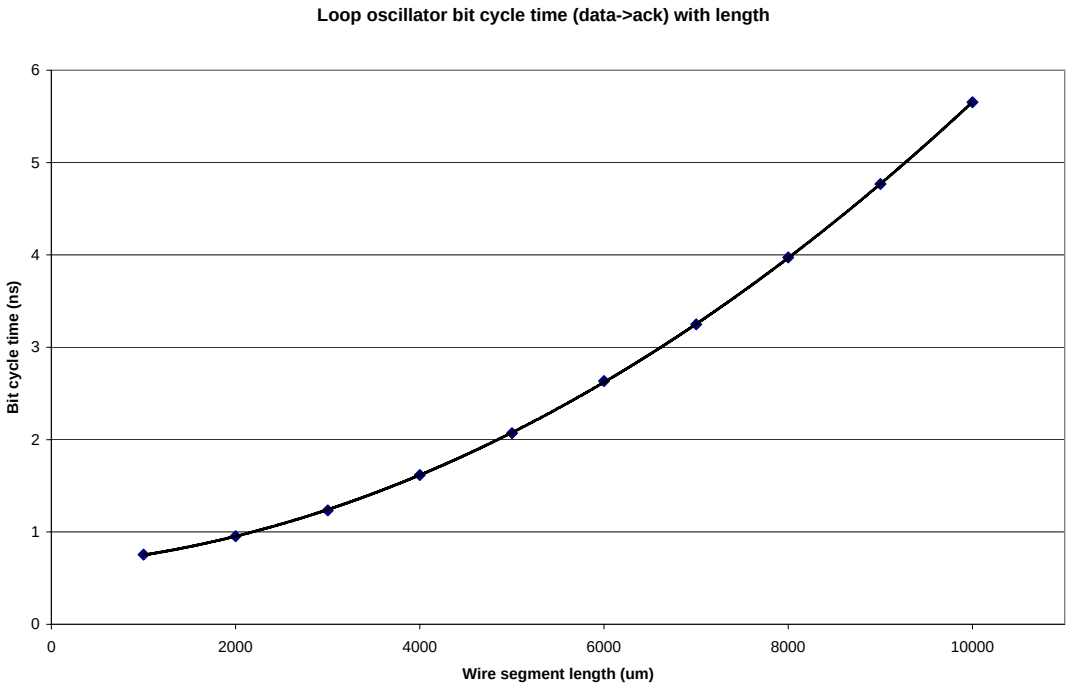


Figure 6.10: Observed bit cycle times between neighbouring stages for various wire segment lengths in an four repeater oscillator loop

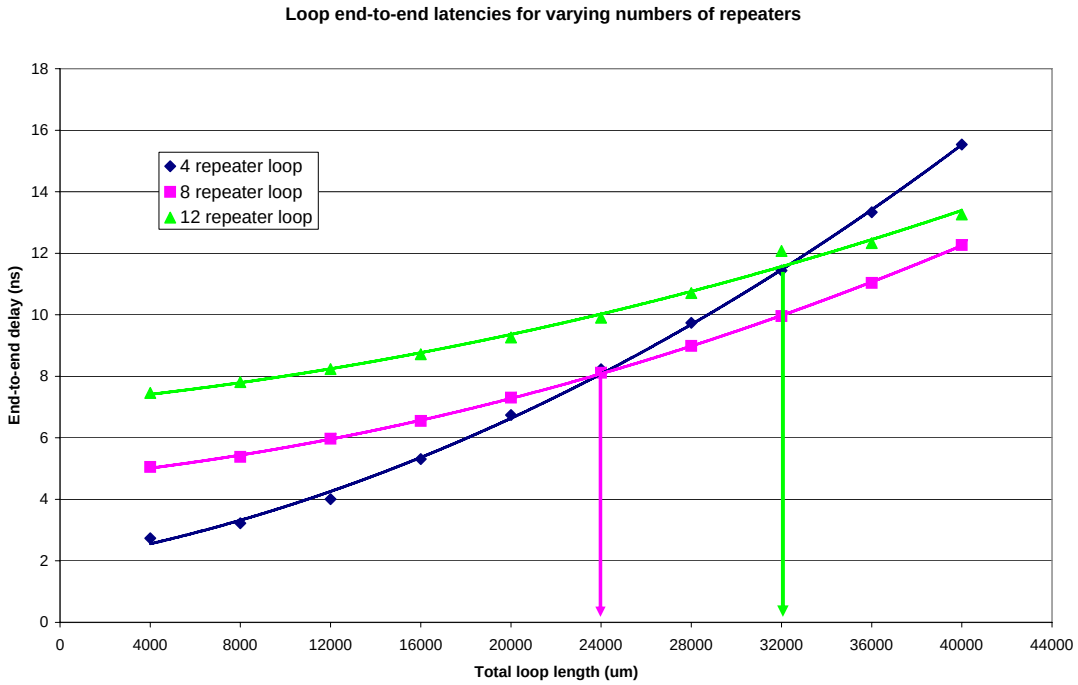


Figure 6.11: End-to-end latencies for full loop traversal for various wire segment lengths in an four repeater oscillator loop with repeater insertion

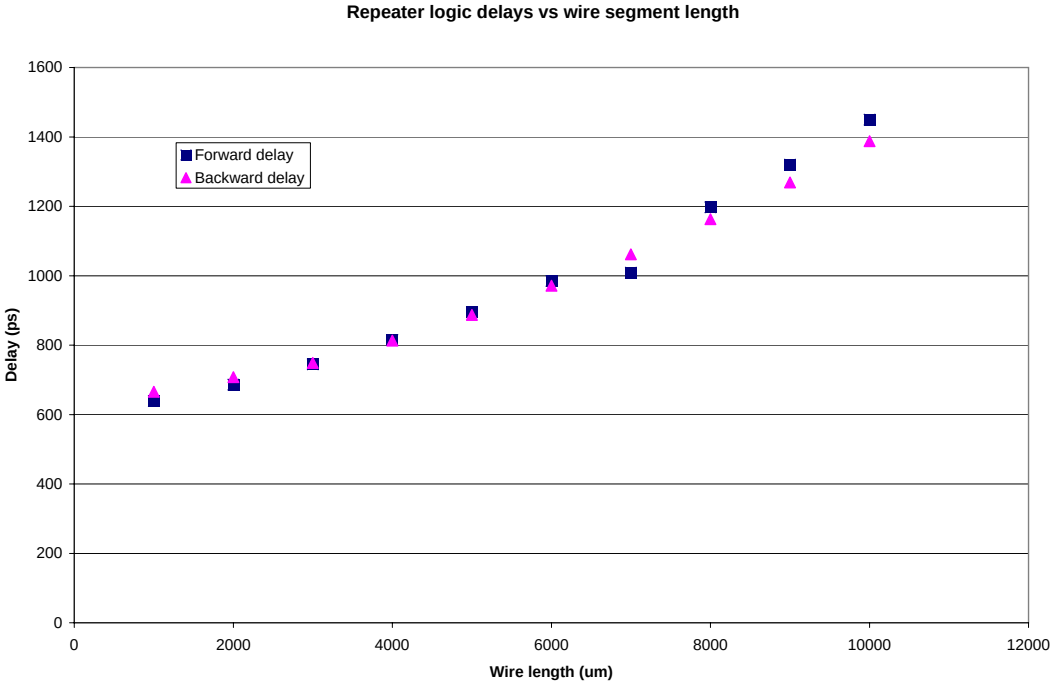
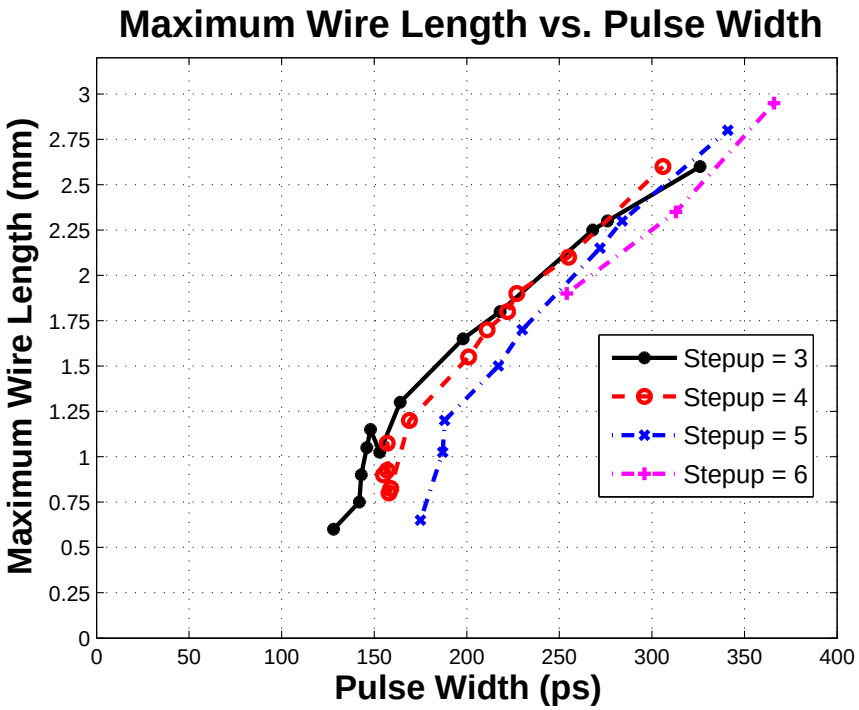


Figure 6.12: Repeater logic plus wire delays in a four repeater loop vs segment length

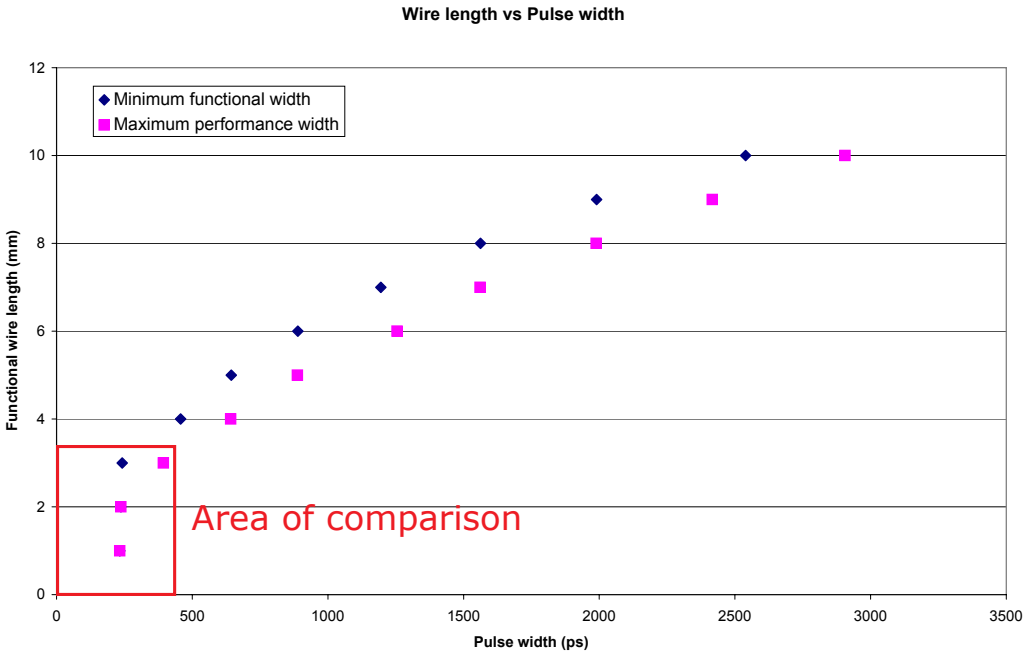
limit being imposed by the lack of robustness with long wire lengths. This is supported by their own admission that pulse widths must be carefully matched, and this arises due to the implementation of fewer safeguards and the lack of data latching with stateful elements.

In addition, they are able to produce finer-grained changes to their pulse widths than can be done with my system. Therefore, their results graph displays equivalently more data points in a given range. Due to the way that pulse widths in my system must be multiples of two inverter-delays, I would only be able to generate four data points in the 100–350ps range they display. This is the reasoning behind the very different scales on the their graph and mine. I include my graph to show the similarity of form, even if not of scale.

In Figure 6.14(b) we also see a trace of a pulse from a repeater loop simulation, as captured at the receiving end of a 1000μm wire segment. Displayed in Figure 6.14(a) is the same trace, reproduced from Ebergen, Furber et al. [15]. We see that the traces are very similar, featuring the same, exponential discharge and charge profiles, and pre-shoot at the beginning of both the discharge and charge phases. The lack of any undershoot shows that the system is well damped, implying RC wire operation, rather than having any obvious inductive influence. Note that the pulse from my system has an observed width of approximately 600ps, whereas Ebergen, Furber et al.'s has a width of 300ps. Whilst they may therefore seem to be a factor of two different, the pulses are actually comparable, since the driver in my scheme drives for 294ps, with the remainder being taken up by the acknowledge (233ps) and the wire delay ($RC/2 = 35$ps). Therefore, the total time the wire is driven low in the forward direction is the same in both cases.

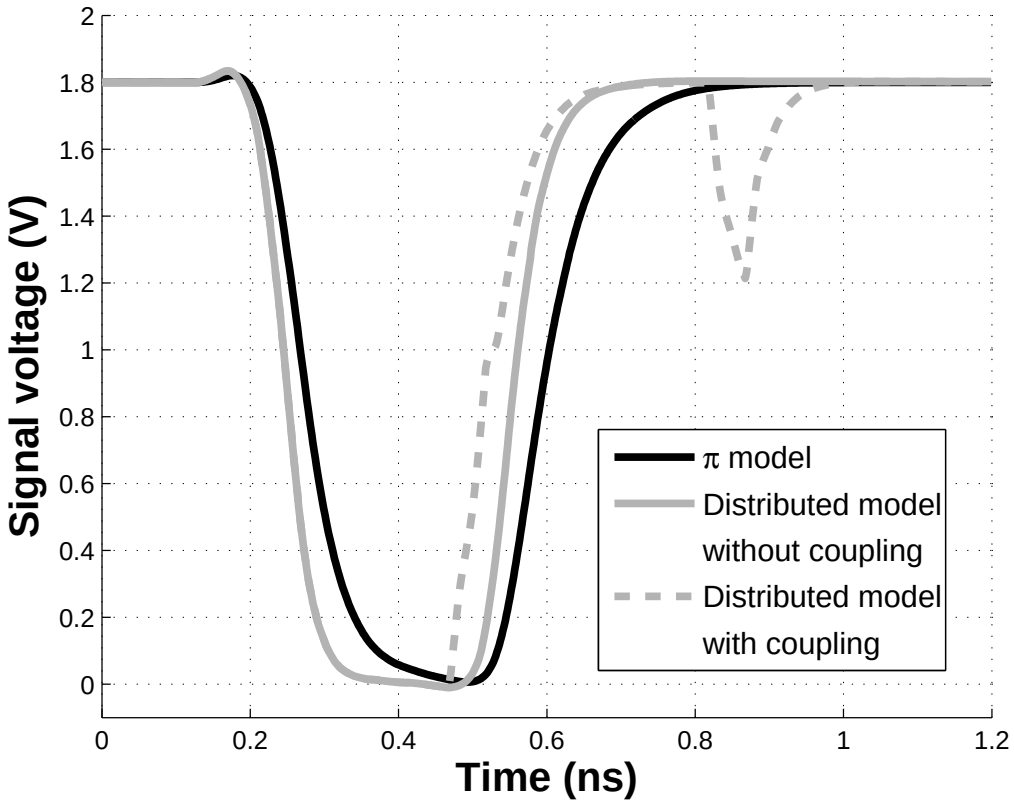


(a) Maximum wire length vs pulse width (reproduced from Ebergen, Furber et al. [15]) — results from an eleven repeater loop oscillator

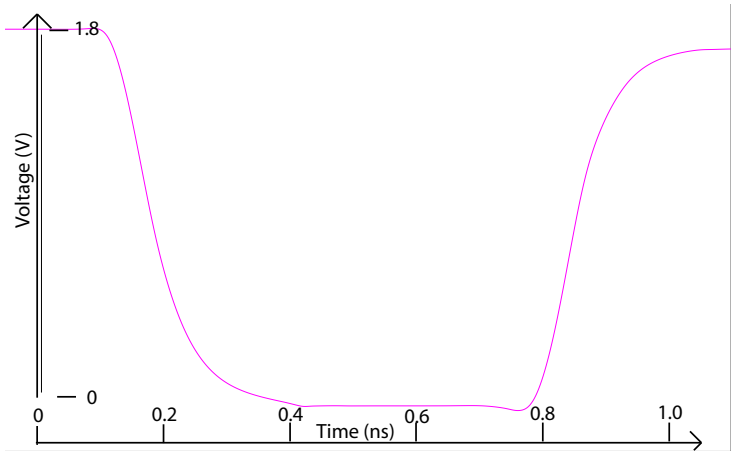


(b) Wire length vs pulse widths — results from a four repeater loop oscillator

Figure 6.13: Wire length versus pulse width sweeps



(a) Received pulse shapes for different wire models (reproduced from Ebergen, Furber et al. [15])



(b) Received pulse shape at a stage of a four repeater loop oscillator

Figure 6.14: The shape of received pulses over 1mm of wiring

6.6 Basic link evaluation

In Chapter 5 we saw how multiplexer (MUX) and a demultiplexer (DEMUX) elements may be connected together via a wiring segment. This comprises the simplest possible link configuration, providing a data path between two eight-bit parallel interfaces. In any interconnection system, we desire that the wires joining together data sources and sinks be the critical path, rather than any logic. In this section, we will see if this holds for our basic interconnect, and several other important performance metrics will be evaluated.

Operation of the link consists of a low wire-driving pulse being produced at the MUX end, in order to transmit data, followed later by a high pulse at the DEMUX end to transmit the acknowledge pulse and reset the wires to high. The widths of these pulses can be varied so that the elements drive for shorter or longer periods of time. Longer periods are of use for long wire segments, where a minimum pulse width may be insufficient to charge the whole wire. Conversely, for short wire segments, the smallest possible pulse is likely to give optimal performance and minimal power consumption.

6.7 Pulse widths

We now look at the effect that varying the transmitted pulse width has on the reliability of communication over different lengths of interconnect. Generated at the transmitting end is a pulse of width varied by the number of inverter pairs in a delay chain. After a fixed minimum amount, to ensure the pulse generator gates operate correctly, this value, which I call 'the delay' is varied to produce an output pulse of different widths. The situation is identical at the receiving end, to produce the acknowledgement pulse. Each additional pair of inverters inserts a delay to the wire driving circuitry of approximately 65ps. However, this is not necessarily reflected in the pulse observed on the wire, since it must first travel through buffering circuitry that may alter its profile. Table 6.2 shows how inverter pair insertion relates to pulse widths. When these widths are input to lines 0–3mm in length, their edges are smeared by the charging of the wire, and this produces swings on the wire with timings as shown in Figure 6.15. As expected, longer wires make swing slower, and require wider input pulses. At 4mm correct function can only be observed with a delay factor of at least four.

I displayed a simplified representation of the phases of the pulse-based protocol as Figure 6.5, and the reader would be well advised to familiarise themselves with this, since much discussion will refer to the basic operations presented there. Correctness of data transfer is dependent on a data pulse being long enough on the wire to cause a transition at the far end to a valid logic value, but not so long that it causes a prolonged fight with the acknowledgement pulse, resulting in the loss of the latter, and an associated drift in the idle wire voltage.

Up to this point, I have merely given the best results available for a loop oscillator, regardless of how wide a pulse was needed to generate them. We will now see a full consideration of pulse widths and their impact on designs.

Thankfully, the range of valid pulse widths is fairly large and some error can be tolerated when choosing the correct width for a given wire length.

The basic design of a pulse generator from Chapter 5 uses the chopper circuit of Figure 6.17(a) to generate the output pulses. During runs of the loop oscillator simulation, I noticed that this design has a failure mode if it observes too long a pulse on the wire. At this point, multiple acknowledgement pulses may be generated for a single input data bit, if the

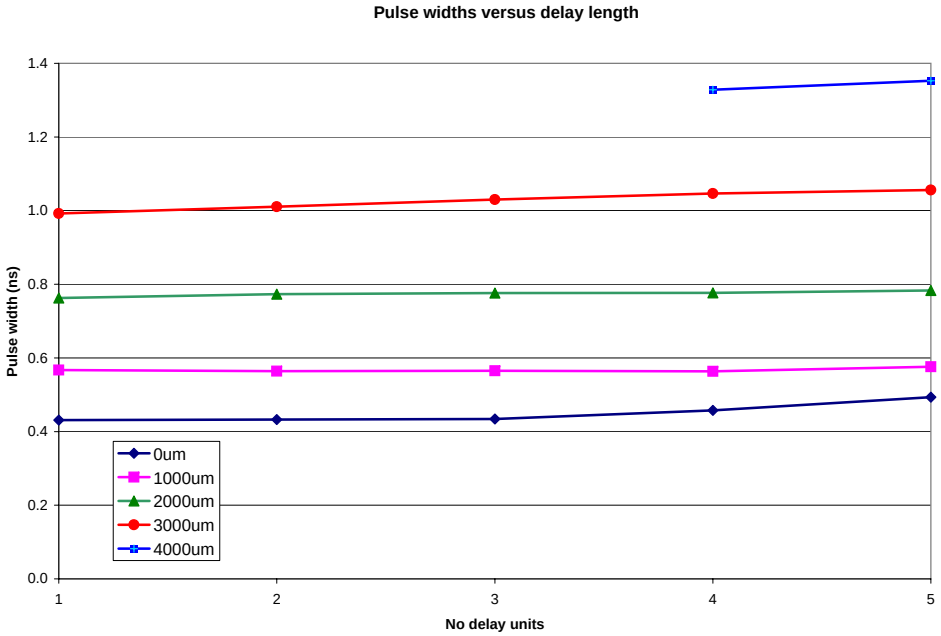


Figure 6.15: Wire pulse widths vs delay units for varying wire lengths

Table 6.2: Driver pulse widths produced for various configurations of pulse chopper pulse generator

Delay factor	No. inverters	data pulse width	matched ack pulse width	unmatched ack pulse width
1	1	246ps	262ps	136ps
2	3	320ps	342ps	261ps
3	5	385ps	413ps	343ps
4	7	445ps	467ps	414ps
5	9	506ps	498ps	468ps

logic path of the receiver has a delay much smaller than the pulse width. Seen in Figure 6.16 this can lead to multiple triggerings at the data receiver, and the repeated latching of a single bit of data, and an associated corruption of the data stream.

Failure modes like this are alluded to by Ebergen, Furber et al.[15], where they say, "...the pulse must have a width at most equal to the loop delay." They talk here about repeater design, but it is a good observation for our receiver design. Equally, they say that one can make an implementation, "more robust against too wide input pulses", but omit any details of how to do so.

It is possible to design a much more robust circuit for the link here, where the use of a latch (Figure 6.17(b)) prevents the problem of too short an input pulse generating too short an output pulse, by holding the notion of an event in state. However, the delay must be designed to be not longer than the ack-to-data bit cycle time on the wire, otherwise the $\overline{\texttt{clr}}$ signal is not de-asserted in time to receive a subsequent bit. In this case, the bit would be ignored and our protocol violated, resulting in lost data and a potential seizing up of the system.

Table 6.3: Valid base link pulse widths

Wire length	valid delay factor range	tx end driver pulse	rx end driver pulse
1000μm	1–12	318–1033ps	260–975ps
2000μm	1–12	318–1033ps	260–975ps
3000μm	1–14	318–1163ps	260–1105ps

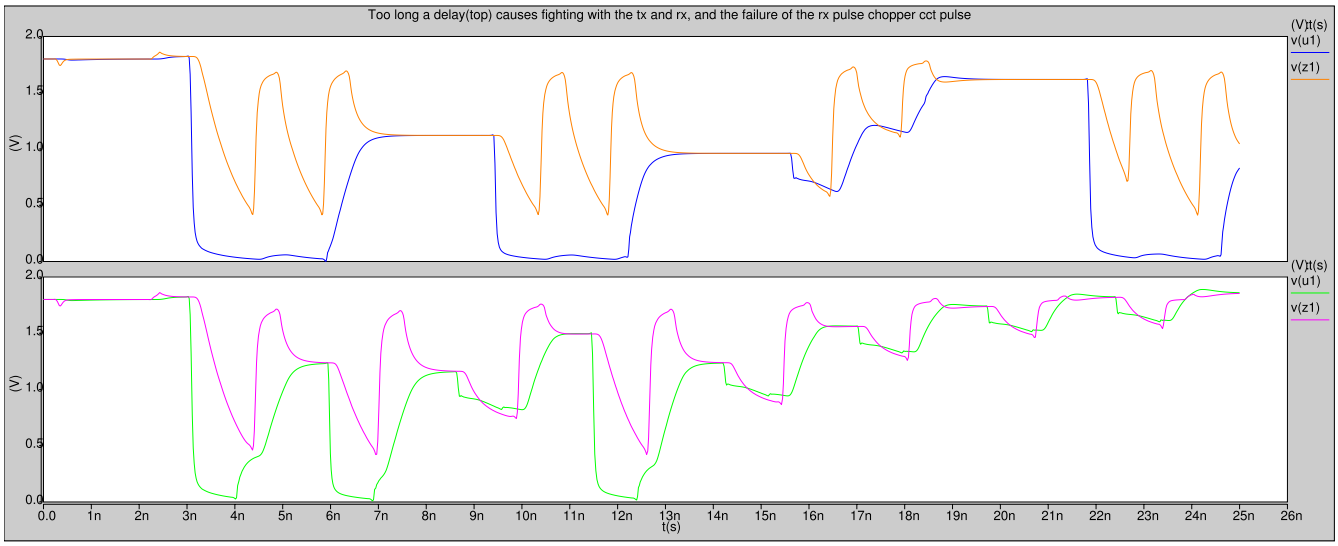


Figure 6.16: Too wide a data pulse can cause multiple acknowledge cycles (orange)

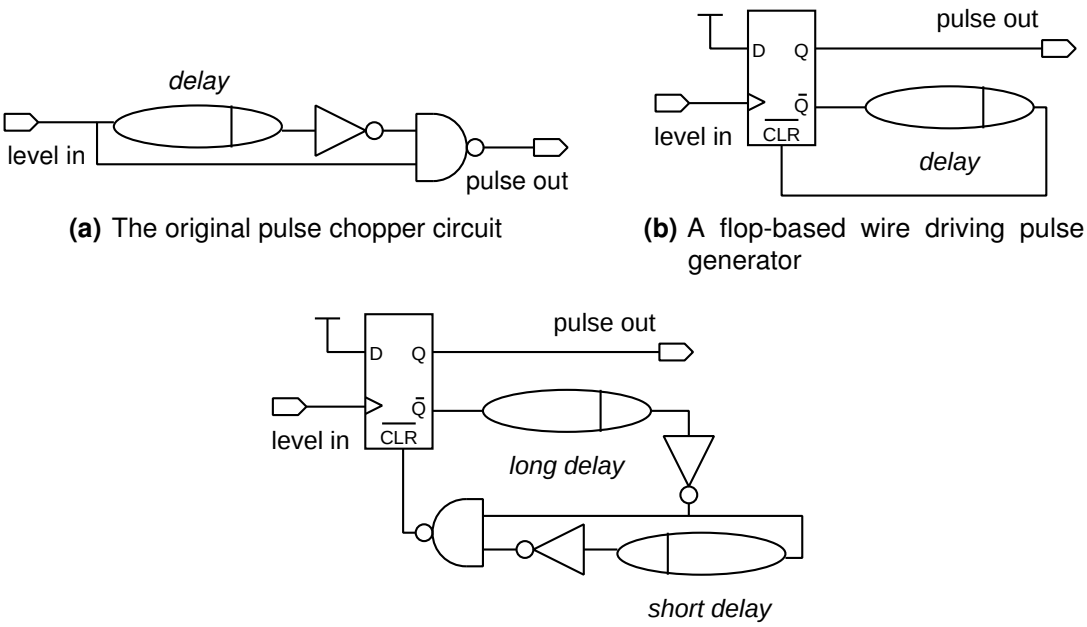
This calibration problem can be overcome by the addition of a chopper on the pulse generator's $\overline{\texttt{clr}}$ input, as shown in 6.17(c), but the additional overhead of all these safety measures is to decrease the overall system performance by around 10%.

In the next section, I compare the performance of the basic and flop-based designs, and make a choice about the correct one to proceed with for the rest of the evaluation.

Matching pulse widths

As an aside, one may expect that some symmetry of design would ensure that the driven data and ack pulse widths are near identical. In fact, this is not the case, and there is some flexibility in their sizes. In particular, the simpler ack pulse's path means that acknowledgement pulse widths can be made much smaller (by about two inverter delays) than the data ones. This means that an ack pulse can be made over 100ps shorter than the data one. Surely doing this will result in increased performance, especially with short wire lengths since there is less time given over to driving?

Well, as Figure 6.18, I show that this is, in fact *not* the case. The graph shows the total bit cycle times for a link using pulse widths of the minimum possible ('unmatched', which is 246ps data / 136ps ack, at their lowest value), or with the two matched to the larger, by the artificial insertion of two inverters' delay on the ack path ('matched', lowest values 246ps data / 262ps ack). Points are plotted when these widths are scaled up, in multiples of two inverter delays, and data is transmitted over a 1000μm link.



(a) The original pulse chopper circuit

(b) A flop-based wire driving pulse generator

(c) A flop-based wire driving pulse generator with over-wide input pulse protection

Figure 6.17: Two improvements to the basic wire pulse generating chopper circuit

Bit cycle times are observed to be shortest when the two pulse widths are matched, and operation is even more energy-efficient, as shown by the energy-delay product in Figure 6.19. Although the differences are slight, the result is still counter-intuitive. The explanation lies in the fact that, at this length of wire, the acknowledgement, whilst still long enough in the unmatched case to reliably drive the wire, drives it for less time. This then impacts the slew rate at the far end, and it takes longer for the ack to be detected, than in the matched case, where it gets to drive for longer. The speed increase then affects the energy-delay product by decreasing delay and mirroring the conclusion here.

In Figure 6.19 we also see the energy-delay product values for the matched and unmatched MUX-DEMUX link, and come to the same conclusion, that matched delays are best.

Both result sets for both configurations have the same form, with a flat section on the left-hand side, with a roughly polynomial rise afterwards. Both begin their climb at a total pulse width of 1.0ns.

What does this tell us? Well, first we see that it does not matter if the pulses are balanced or not — from an energy-delay perspective it is the total pulse width that counts, and this is intuitive if we consider that the total width is the total time that a wire is being driven for and driving energy should be the same per-unit-time constant.

Second, and perhaps most importantly, is the explanation of the shape. The flat section shows that pulse widths may be varied in a range of roughly 500–1000ps without impacting the delay-energy product. We will see in Equation 6.8 that, for a range of short pulses, bit cycle times remain constant since the logic delays in the MUX and DEMUX elements are instead the critical path. The same is true here, for the energy-delay product. Essentially, all pulses in the flat range charge the wire fully, and so dissipate the same energy. They also all produce the same bit cycle time, thus their product is also constant.

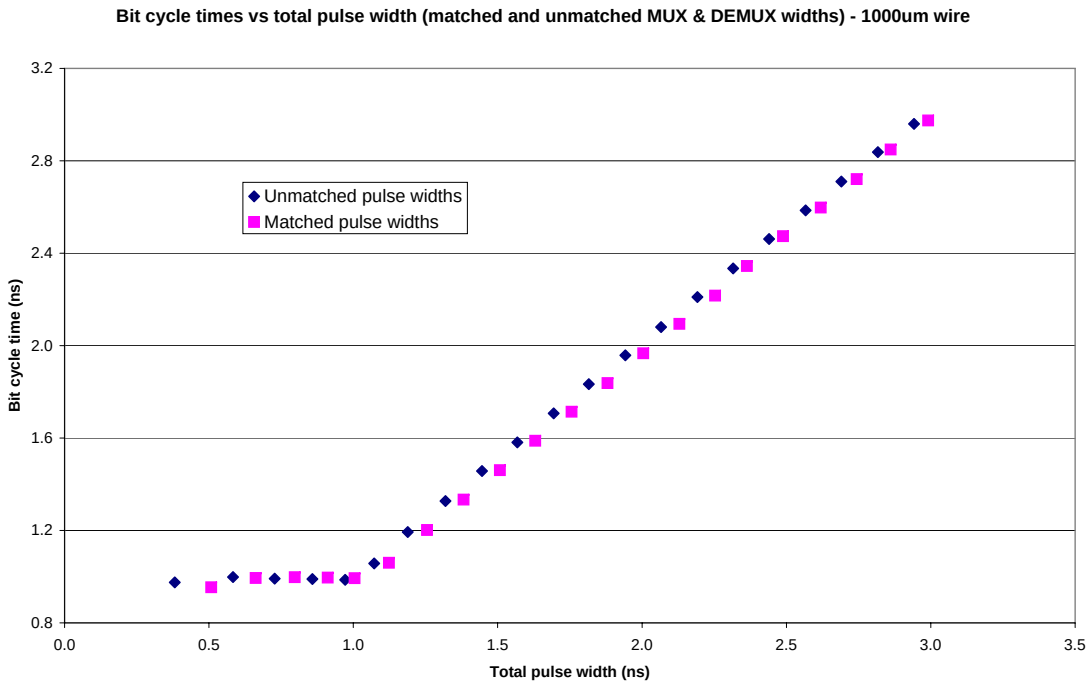


Figure 6.18: Bit cycle times for a 1000μm MUX-DEMUX link

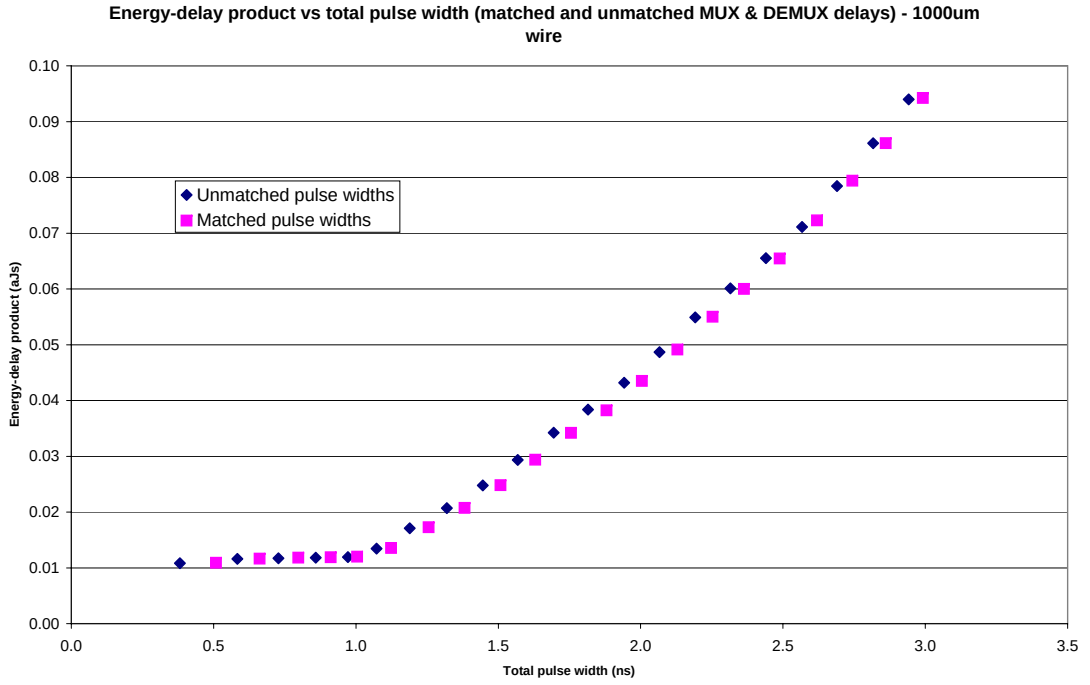


Figure 6.19: Energy-delay products for a 1000μm MUX-DEMUX link

Pulse width matching conclusion

We have come to the following conclusion about the best setup for pulse widths: *latency and throughput are optimal* when the pulse widths of the data and acknowledgement drivers are *matched and the narrowest possible for reliable operation at a given wire length.*

With this knowledge, we can proceed to evaluate the performance of the two pulse generator designs and how they may be scaled. We will do this, but first I explain how pulse widths affect the voltage swing on the wire.

6.8 Voltage swing

Voltage swing on interconnect wires is a key factor when determining the reliability of signals. In this section, we will examine the effect that wire length has on the voltage swing, and how it can be increased by the widening of driving pulses.

As a base case, compare Figures 6.20 and 6.21, where the traces for a minimum driving pulse are displayed for wire lengths of 1000μm and 3000μm, respectively. For both, the transmitting end voltage is drawn in green, and the receiving end in magenta. Measurement lines are also drawn to show the magnitude of peaks and troughs. Striking is the difference in shape between the two traces. Whilst the 1mm drawing shows clear transitions, as would be expected by any well-behaved system, the 3mm version shows a much more complex picture. Here, whilst the transmitting end makes a full transition from `vdd` to 0 volts during its swing, the wire capacitance means this is not so for the far end, and a discharge to a mere 0.8V is seen. Note how, after a period of driving the transmitting end low, the voltage then drifts back up to equalise at that of the receiving end. This is the effect of *charge sharing* — driving has stopped, so the capacitance of the line eventually equalises the voltages of the two sides by averaging them out, with the effect that both drift toward a median value. The situation is made even worse when a change in data value causes a wire to settle to a bias voltage, rather than going rail-to-rail.

In total, we see that swings down at the receiving end can be as high as 0.8V, and swings up at the receiving end as low as 1.2V. When combined, this makes a differential signal swing of only 0.4V, compared to a `vdd` of 1.8V. At this point, signal integrity becomes a problem — what bit value should the logic at either end detect? For this reason, we find that the interconnect system begins to fail at the 3mm length.

Increasing the driver pulse width

We know the reason voltage swing is degraded when driving long wires is that charge sharing equalises two unlike voltages after positive drives have stopped (i.e., the pulse width is over). Can we, therefore, arrange that there is less charge sharing?

The answer is yes: by altering the driving time (which is the pulse width), we can change how much the far end is charged and discharged before sharing occurs. To ensure signal integrity, the pulse width can be widened to 506ps (see Figures 6.22 and 6.23). The change is obvious: both the one and three millimetre traces look similarly well-shaped, and so 3mm is easily reachable with this pulse width.

What is the downside of using the wider pulse then? Well, whilst hard to see from the diagrams, the time taken to transfer a bit is increased, and the analysis of this can be seen in other sections of this chapter.

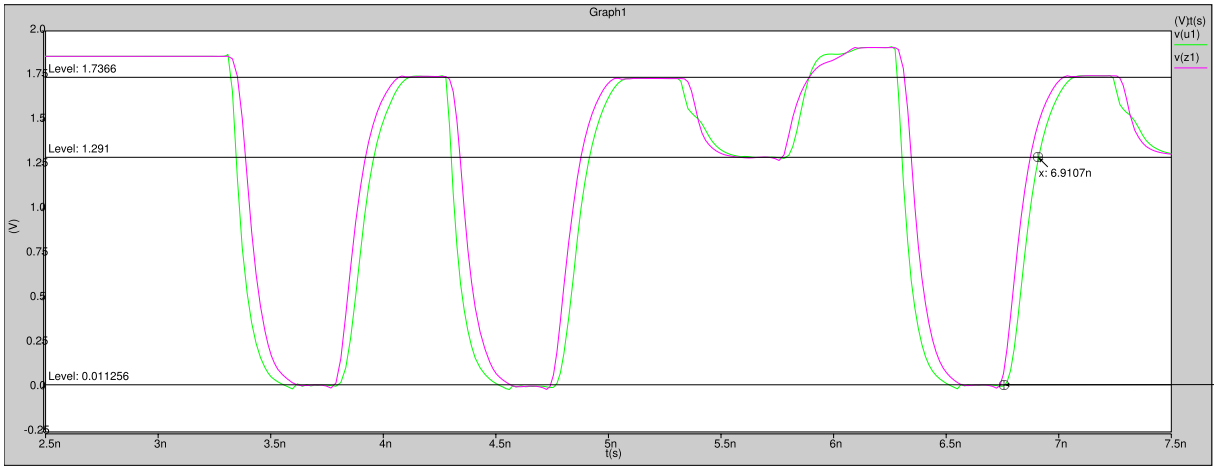


Figure 6.20: The voltage swing observed over a 1000μm wire with a minimal driving pulse width of 246ps

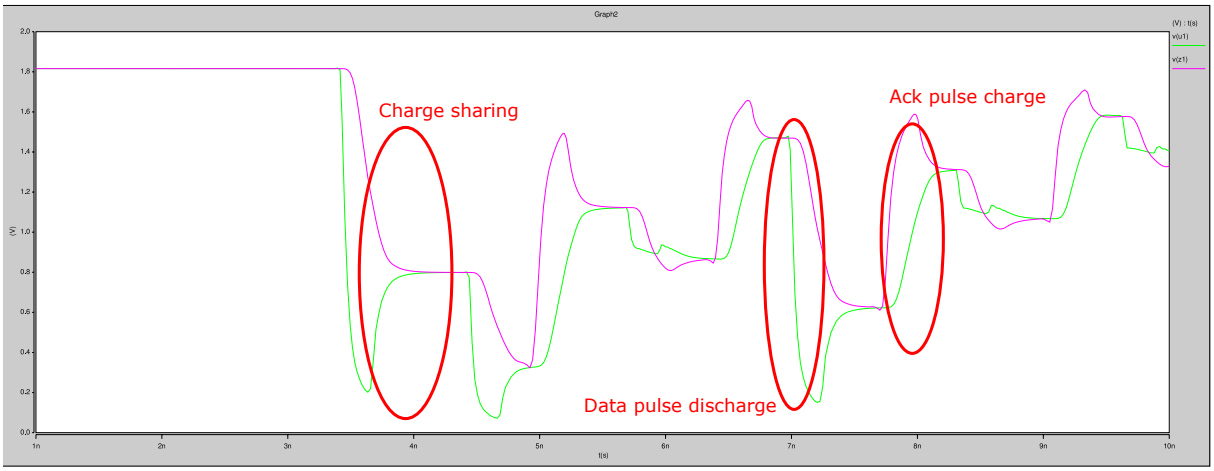


Figure 6.21: The voltage swing observed over a 3000μm wire with a minimal driving pulse width of 246ps

If the designer is confident of the wire lengths used, and trims the driver pulse widths as short as possible, then an advantage may be gained, since low swing operation offers potential power and time savings (since less wire capacitance must be charged in each cycle).

6.8.1 Crosstalk

Now we have seen how the transmitted signal quality varies with wire length and pulse width, it is time to consider the other factor in the S/N ratio: noise. The injection of noise has a detrimental effect on operation, since it reduces the received signal quality. With an on-chip interconnect, noise manifests itself mainly through *crosstalk*. The effect of crosstalk is to couple together two interconnect wires, so that transitions on one (the *aggressor wire*) cause

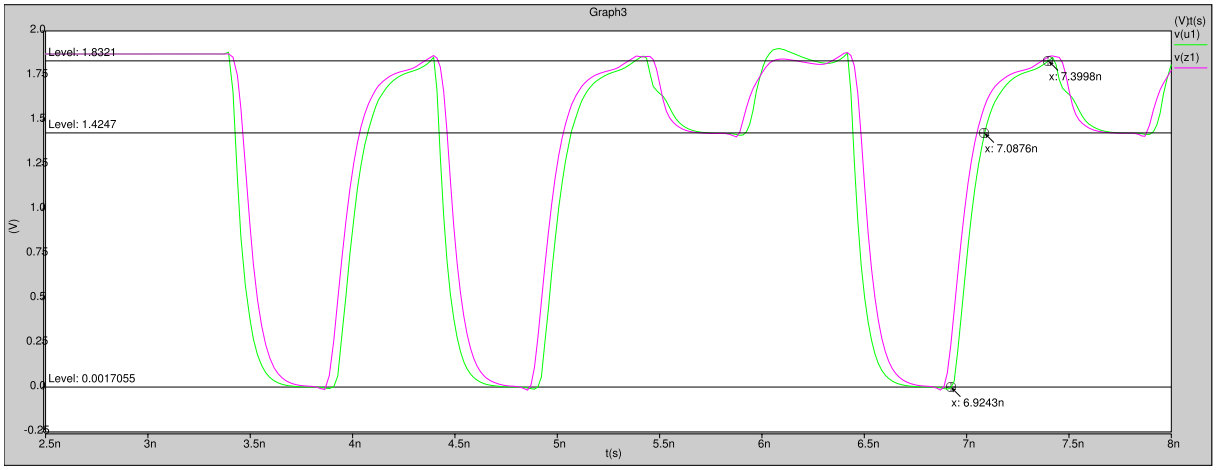


Figure 6.22: The voltage swing observed over a 1000μm wire with a driving pulse width of 506ps

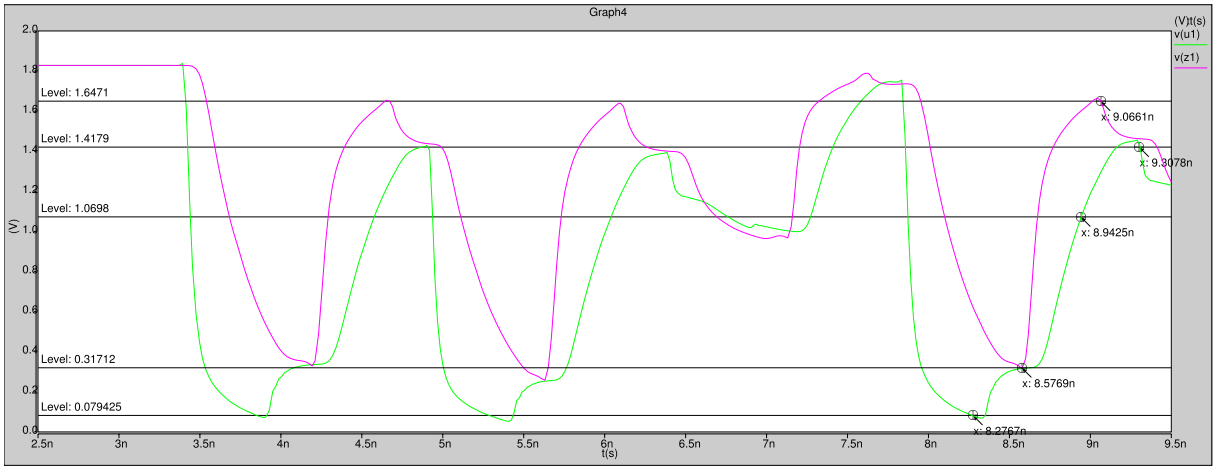


Figure 6.23: The voltage swing observed over a 3000μm wire with a minimal driving pulse width of 506ps

voltage fluctuations on the other (the *victim wire*). How large are these fluctuations then, when considering our interconnect situation?

With crosstalk, it is the rate of change of voltage at a particular point on the aggressor wire, rather than its absolute value that effects the victim (see Section 4.3.1 for a refresher), so we can actually discount the effect of pulse width, since all widths are driven with the same strength. This simplifies our analysis down to the effect of wire length. In Figures 6.24 and 6.25, I show the effect on 1mm and 3mm lines, and I outline this now.

At 1000μm, swings on an aggressor wire from 1.8V down to 0V cause a drop down to 1.5V on the victim wire, implying a crosstalk value of 0.3V on the transmitting end. The analysis is the same in this case for the receiving end.

Additionally, when data changes on the active wire (e.g., when the bit sequence is a 1 followed by a 0), the bias left by the previous transmission can add up to an extra 0.2V of drift to the new victim.

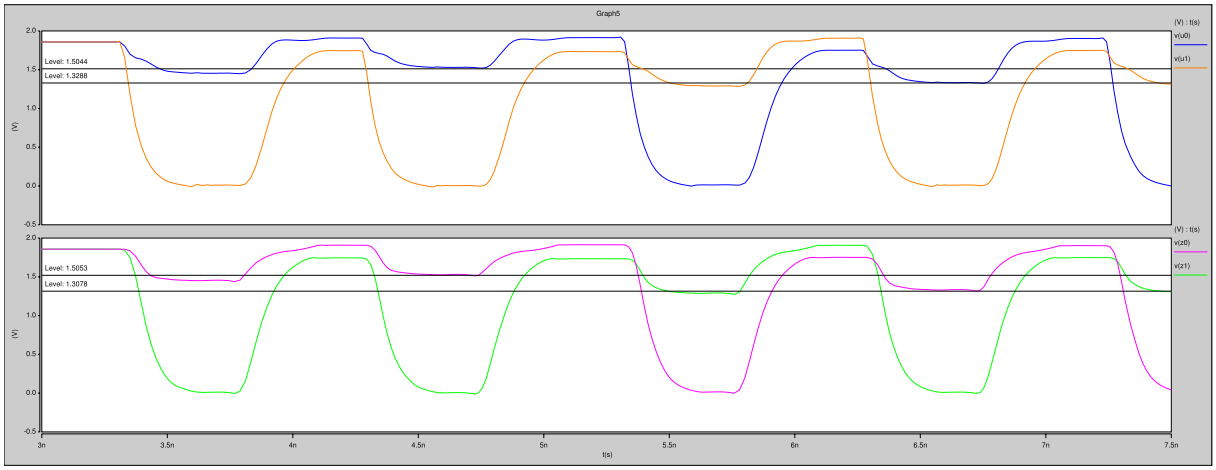


Figure 6.24: Crosstalk on a 1000μm line

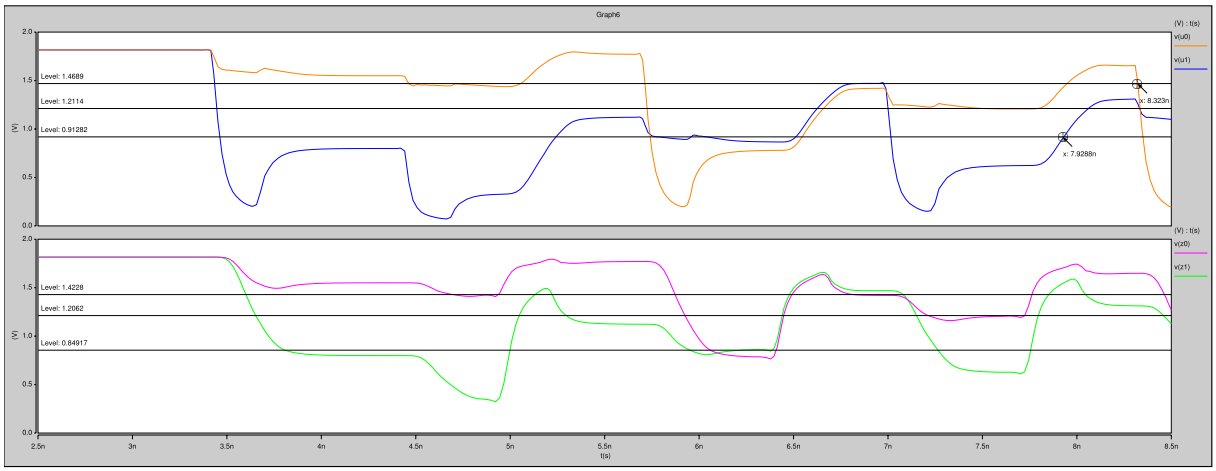


Figure 6.25: Crosstalk on a 3000μm line

For a 3000μm wire, the aforementioned distortion of the waveform leaves it hard to determine exactly what is crosstalk and what is signal attenuation, but it appears that crosstalk swings are down to 1.45V from `vdd`, and to 1.2V after a bias has been impressed — crosstalk levels of 0.35V and 0.6V, respectively.

6.8.2 Reflections and ringing

Ringing [33, pp.160–164], caused by reflections on the wire can be observed on lines where the pulse width is smaller than a few times the propagation delay. Since ringing is not observed, we can confirm that the pulse widths (~ 250ps) is much greater than the calculated propagation delay of the line (35ps).

6.8.3 Effects of inductance

In Section 6.2.1 I made the argument that inductance is unimportant in the design here. It would be nice if the profile of our traces backs this up.

Were inductance to be significant, we would expect to see some degree of resistance to changes of voltage on the wire when undergoing charge or discharge, altering the profile to one with sharper rises or a truncated length. These are not seen, and so our hypothesis does, indeed, appear to be correct.

6.9 Pulse generator design choice

In Section 6.7, I proposed two varieties of pulse generation circuit, it is prudent to evaluate the performance of each before proceeding any further.

When operating with minimal output pulse widths, both designs provide results for wire lengths up to 3000μm, for an unrepeated MUX-DEMUX link (recall the pulse generators here stem from the delay elements in the root MUXs and root DEMUXs). But, as we will see, their quality of performance varies. We have also seen from the loop oscillator (Section 6.5) that, up to 3000μm, operation is reliable with a fixed and minimal generated pulse width of 233ps. Therefore consider we choose to simplify analysis by considering only the range 0–3mm, with this fixed, minimum pulse size. Results from the loop oscillator can easily be used to scale any results we gain here, if needed.

Throughput

Figure 6.26 shows a comparison of the performance of the two designs, measured by their bit cycle times. Throughput of the interconnects is simply one over this value, and so lower is better on the graph. Both implementations operate with their minimum output pulse widths.

We see that the flop-based design has a slightly better performance over much of the range, but this is probably due to it being more heavily optimised at design time. I am confident that the pulse chopper implementation could obtain this level of performance given enough time and resources to improve its implementation. Note also that both designs are capable of achieving a throughput of over 1Gbit/s for a 1mm wire, and so both satisfy the interconnect's original design criteria.

Latency

The two latencies are shown for comparison as Figure 6.27, and we see that, here, the pulse chopper has the edge[§] over the flop-based design. Further, for these, minimal width output pulses, reliable end-to-end transfer of a data word (not shown) is observed for the full 3000μm length for the pulse chopper, but only up to 2500μm for the flop-based design, since the pulse ceases to be wide enough for correct logic operation at this length. We will see more about this shortly.

So, the flop-based design is fine if one has the time to calculate the delay value needed for correct operation at a certain wire length, but the chopper based design is simpler and gives good performance over a wider range of wire lengths, without parametrisation.

[§]No pun is intended, nor warranted funny by the author.

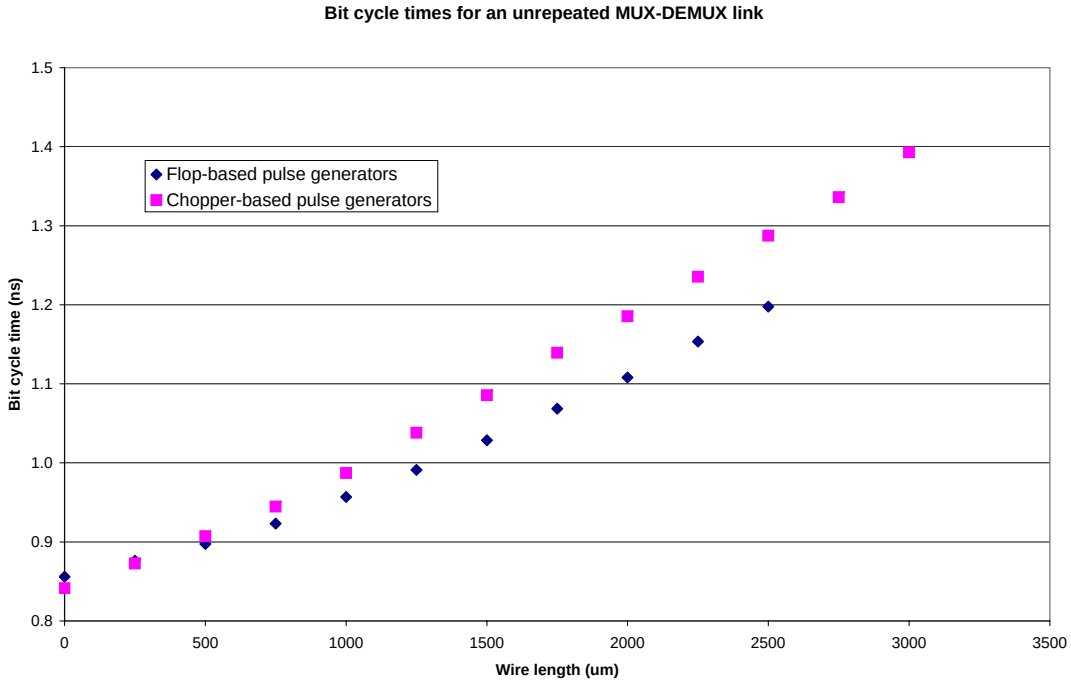


Figure 6.26: Bit cycle time comparison between the chopper- and flop-based implementations

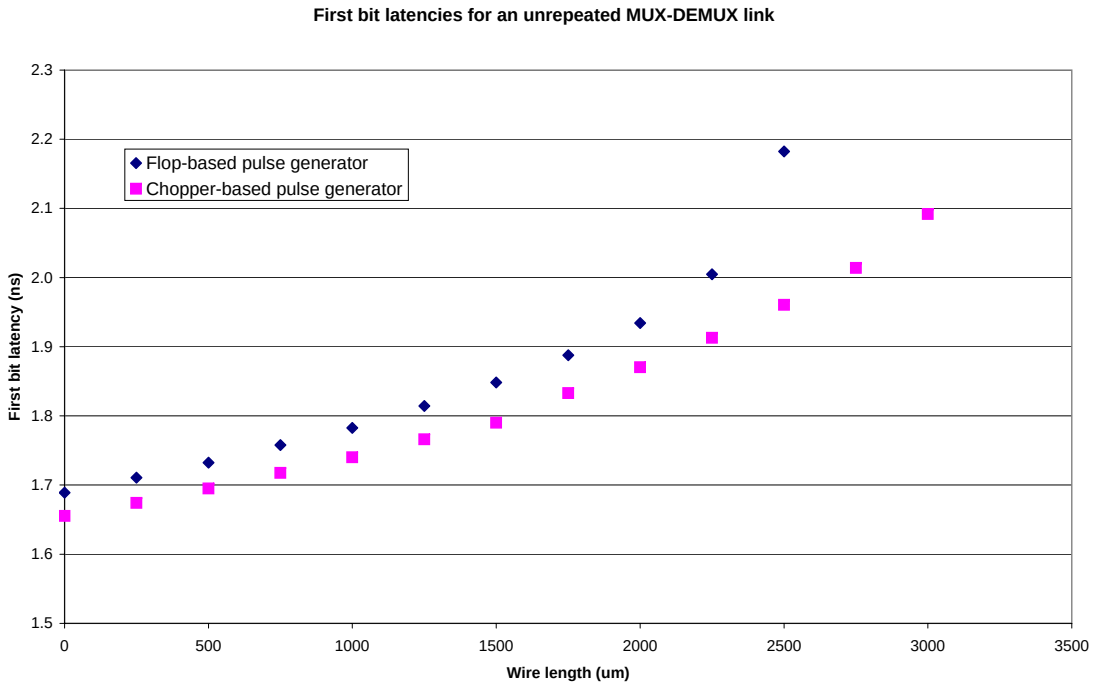


Figure 6.27: First bit latency comparison between the chopper- and flop-based implementations

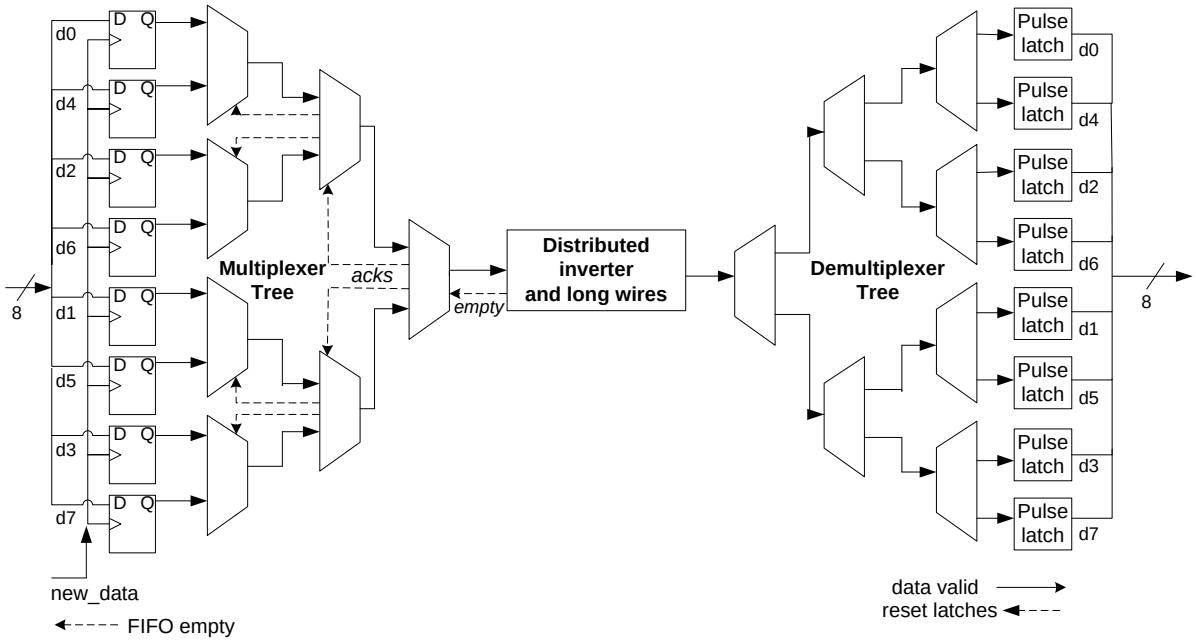


Figure 6.28: Block diagram of the basic MUX-DEMUX link

Summary

Given that each implementation design wins on one of the two metrics, choice will be driven by which of the two is more important to a particular design instantiation. We will see later that bit cycle time is comparatively easier to optimise than forward latency, so it makes sense to choose the lower latency candidate; and this is the chopper design.

To be rigorous though, and since there may still be circumstances where the flop-based design's lower cycle times may be of use, we evaluate it also. We do this next, where we will see that is has undesirable features, making it of little use. An evaluation of the chopper will follow, where we will fully evaluate all of its characteristics, and see it is the design of choice for our system.

6.10 Evaluation of the flop-based design

Having introduced the flop-based design, we proceed to evaluate its performance more fully. Since we have already seen latency and bit cycle time results in the previous section, all that remains is to examine performance with relation to energy, and how the design scales with repeater insertion.

The setup used is a full point-to-point link implementation from Chapter 5 and reproduced here as Figure 6.28, with a MUX tree feeding a link with varying numbers of repeaters, based on the flop pulse generator. At the far end, a DEMUX tree takes the data and it is checked for correctness.

For initial results, the simplest design, using minimal pulse widths was chosen, with the

intention that these be examined at a later stage. As we will see, however, analysis did not get this far, since faulty operation caused a full analysis to be abandoned after the initial sweep. For now though, note that all results operate on pulse widths of 294ps in the data direction, and 233ps in the acknowledgement one.

When we look at the results in Figures 6.30 and 6.29, the first thing to strike you will be the sparsity of data points, compared to what may be expected. It is easy to understand many of the lines do not make it to the right-hand side of the graph: their segment lengths eventually exceed 2500–3000μm and, as we have previously seen, this is the limit of the known operating range with the minimum delay setup we have here. Less obvious, though, is the reason why all traces do not extend fully to the left-hand side (to begin at 0mm, but instead they start at 1mm), as was the case for the unrepeated system (and will also be the case for the repeated pulse-chopper). The answer lies in the implementation details of the flop-based repeaters, and is a rather unsatisfactory and stubborn problem.

When used with short links, a complex and non-obvious interaction occurs between the latch inputs and the transitions on the wire. This causes the latch to produce only one acknowledgement signal during a run, regardless of the number of transitions on its input wire. Essentially, there is insufficient time for the internal nodes of the latch to fully stabilise after generating one output pulse, before receiving its next clock input. Hence, the first data points occur at wire lengths of 1mm or more, and for these points of correct operation I have generated graphs of the end-to-end latencies, per bit energies and energy-delay products, and we will examine these now.

6.10.1 Throughput

As we know, the bit cycle time gives the best indication of throughput of the link. Figure 6.29 shows how this scales with length and number of repeaters.

We see that, at the point where each repeater line begins operation, the bit cycle time is the same (approximately 1.2ns), regardless of actual end-to-end wire length. This corresponds to a throughput of 833Mbit/s. This slightly counter-intuitive result is actually easily explained by noticing that each plot line begins when that number of repeaters causes the segment (inter-repeater) length to be the same: 1000μm (as an example, the two repeater line begins at 3mm, creates three segments, and 3mm/3 = 1mm). Only two repeater configurations are able to successfully transfer data up to 10mm, those of four and five repeaters. Similarly, these two succeed where the others fail since they are the only ones capable of sub-dividing 10mm into segments of less than 2.5mm (10/5 = 2, but 10/4 = 2.5). Therefore, we can say that correct operation occurs only for segment lengths in the range 1–2.5mm.

Just noticeable is the characteristic of the various lines to diverge slightly with increases in number of repeaters (they are not exactly parallel) — bit cycle time climbs more slowly with increased repeating, since a constant wire length change is less and less significant the more segments it is divided into.

In summary though, we can see that, with proper repeater insertion, correct operation can be obtained over the full 1–10mm range, and bit cycle times can be maintained as constant, regardless of total wire length. The latter is, in itself, an attractive point and an advantage of a system using stateful repeater elements.

6.10.2 Latency

End-to-end latency of a real world link can be the critical design factor if blocks are unable to buffer data bits, and data must arrive before computation can occur. Figure 6.30 shows how the latency for our setup changes with varying of total wire length, and repeater insertion. The range is from 0–10mm, with correct operation being observed from 1mm upward.

The latency graph exhibits a high degree of symmetry for the repeater insertion plots. We see that each number of repeaters creates a line parallel to all others, and the greater the number of repeaters, the higher and more offset to the right the line. The meaning of this is quite simple: latency is increased by additional repeaters, since they insert a high logic delay into the path (this is the move upwards), and the greater the number of repeaters, the longer the total wire length that can be spanned (the horizontal translation).

However, were one to draw a regression line through the start, middle or end points of the individual traces (as illustrated by the green dashed line), you would find it a good approximation to linear, and so our repeater insertion increases latency linearly overall. This point is an important one. The main feature of optimal repeater insertion is that it produces delays that increase linearly with wire length, and that is what we see here too. So, we are happy that repeater insertion has successfully converted a quadratically-decreasing performance curve into a linearly-decreasing one.

In comparison to the unrepeated case, repeaters reduce overall latency when the total wire length is 3000μm. Here, the wire delay, plus the degradation in signal rise time outweighs one repeater forward delay in latency. Therefore, from a latency point of view, repeaters are useful for wire lengths over 3000μm.

This result differs substantially from the loop oscillator case, and arises due to the relatively poorer performance of the MUX and DEMUX element's logic inputs and pulse generators, compared to a system consisting solely of repeaters. Were the elements not to be equally spaced, one might reasonably expect that, if MUX and DEMUX elements were repeated at three millimetres, then the spacing between remaining repeaters could be stretched to nearer six millimetres to improve performance. However, to keep things simple and uniform, this analysis is not done here.

6.10.3 Energy use

Now that we have seen how our link performs, how much energy does it take to transfer a bit, and how does that relate to overall energy optimisation? The key metric relating to the power efficiency of any modern VLSI implementation is the *energy-delay product* [65, p.193]. Simply put, it highlights the optimal balance between performing power-hungry computation quickly and more modest computations stretched over a longer period of time. The lowest value of energy-delay product indicates the most efficient system, but not necessarily the highest performance one.

Energy per bit

However, since the energy-delay product is not necessarily the most intuitive one to look at and understand what is going on, first I display as Figure 6.31 the energy taken per bit transferred, with our 0–5 repeater setups.

We see two things from this graph: that the greater the number of repeaters, the higher the power consumption; and also that power consumption per bit is mostly constant for each trace, but peaks when a configuration is in the middle of its operating range. The first is easily explained by realising that each additional repeater consumes additional logical power, whilst the overall wire capacitance to be charged remains unchanged. Therefore, the increases indicate the energy cost of a repeater.

The second is a little more difficult to understand, and will be explained more fully in the repeater evaluation section (§6.13), where the same characteristic is observed.

Energy-delay product

Excepting the unrepeated case, the energy-delay product plot shows a continuous line of increasing energy-delay product with total wire length. Also, the different repeater configurations' extents blend well to form a solid line, much longer than any individual trace. Finally, to a best approximation, the energy-delay product is linear with length.

Again, this suggests an instance of optimal repeater insertion. We have just seen that energy consumption is nearly constant. Therefore, we can deduce that it must be delay that increases linearly (and, in fact this is confirmed by our earlier analysis of latencies).

6.11 Summary of the flop-based design

In the previous few sections, we have evaluated the performance of an end-to-end link design using the flop-based pulse generator from Figure 6.17(b). We have seen that a functional link can be produced when segment lengths are in the 1–2.5mm range unrepeated, up to 10mm (or more) if repeated using our stateful repeaters.

Latency has been shown to increase linearly with length if properly repeated, but throughput can be maintained near constant at 833Mbit/s.

The energy-delay product also rises linearly with wire length, which is roughly optimal given that so does the wire capacitance that must be charged.

Despite all these promising results, however, reliability problems with long *and* short wire segments means that further evaluation was abandoned by the author, in favour of producing a more robust design. We will see this in the next section, where the pulse chopper circuit is resurrected to produce a simpler, more lightweight and reliable drop-in replacement for segment lengths of 0–3mm, and yet still uses a fixed pulse width.

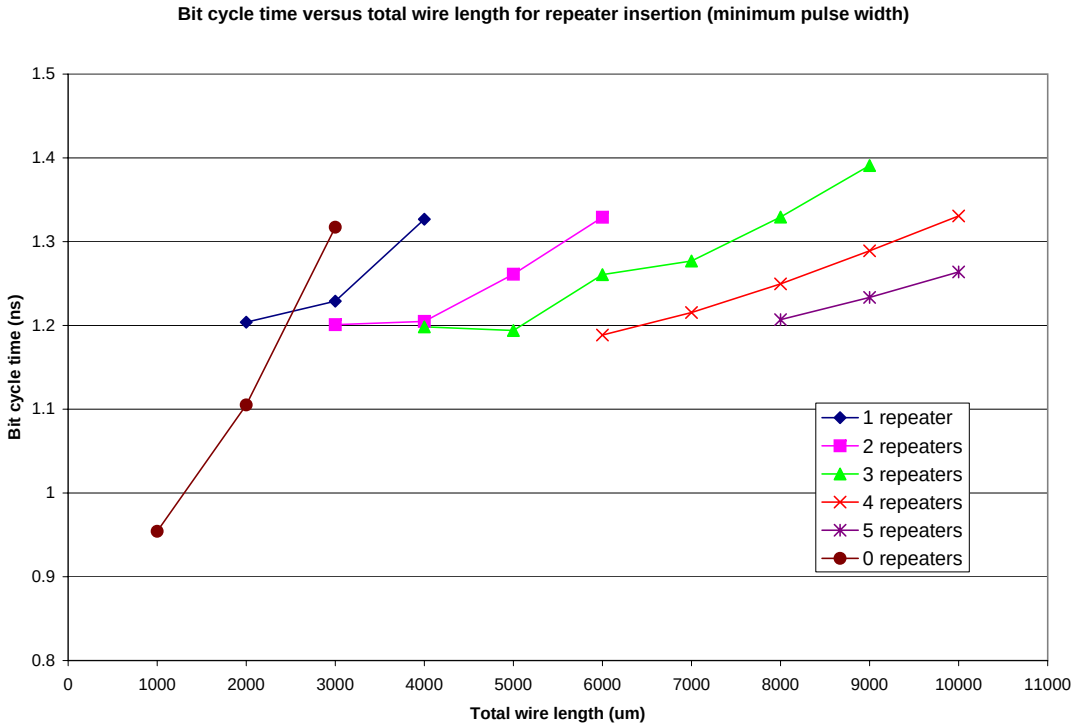


Figure 6.29: Bit cycles times for a MUX-DEMUX link with total wire length, for varying numbers of repeaters inserted — flop pulse generator, 1 delay

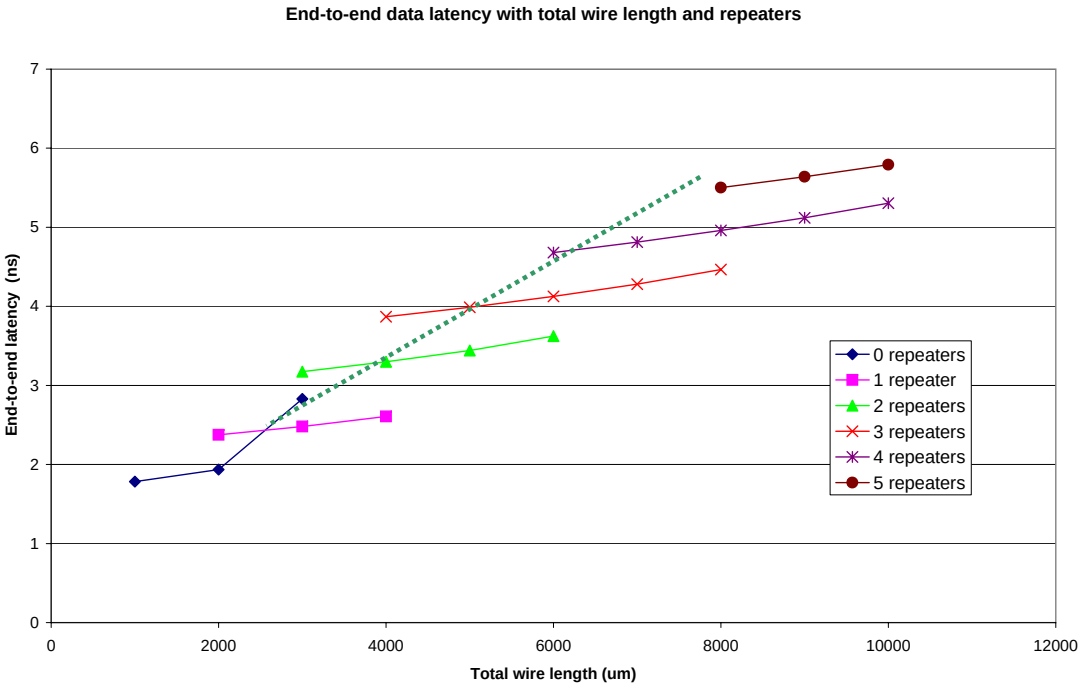


Figure 6.30: End-to-end latencies for a MUX-DEMUX link with total wire length, for varying numbers of repeaters inserted — flop pulse generator

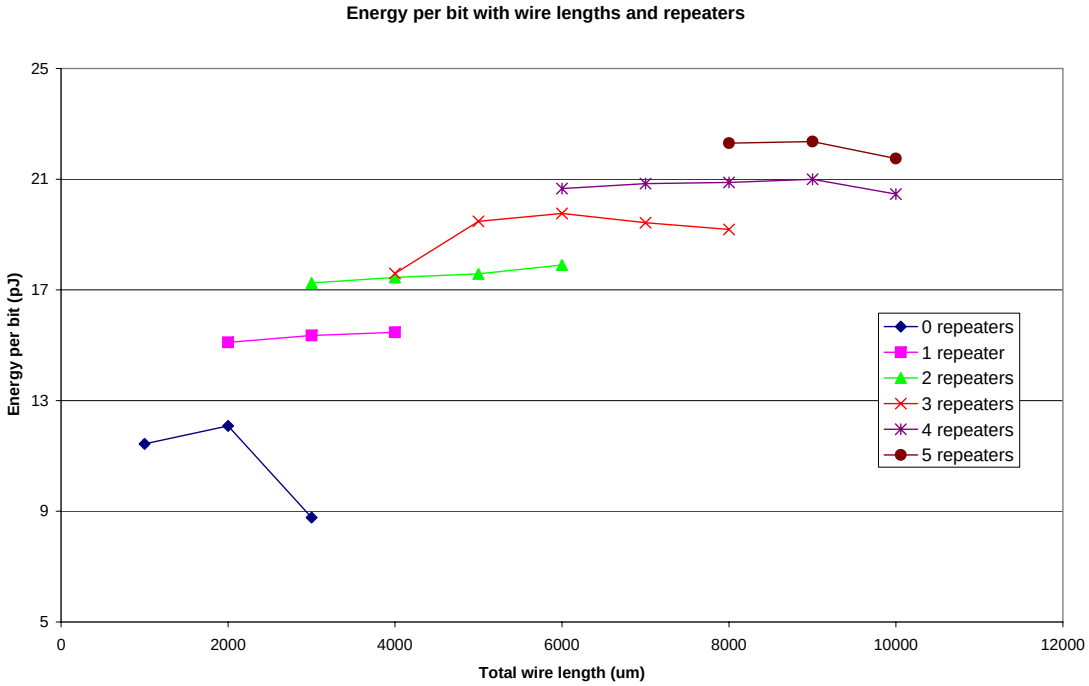


Figure 6.31: Energy per bit for MUX-DEMUX link with total wire length, for varying numbers of repeaters inserted — flop pulse generator

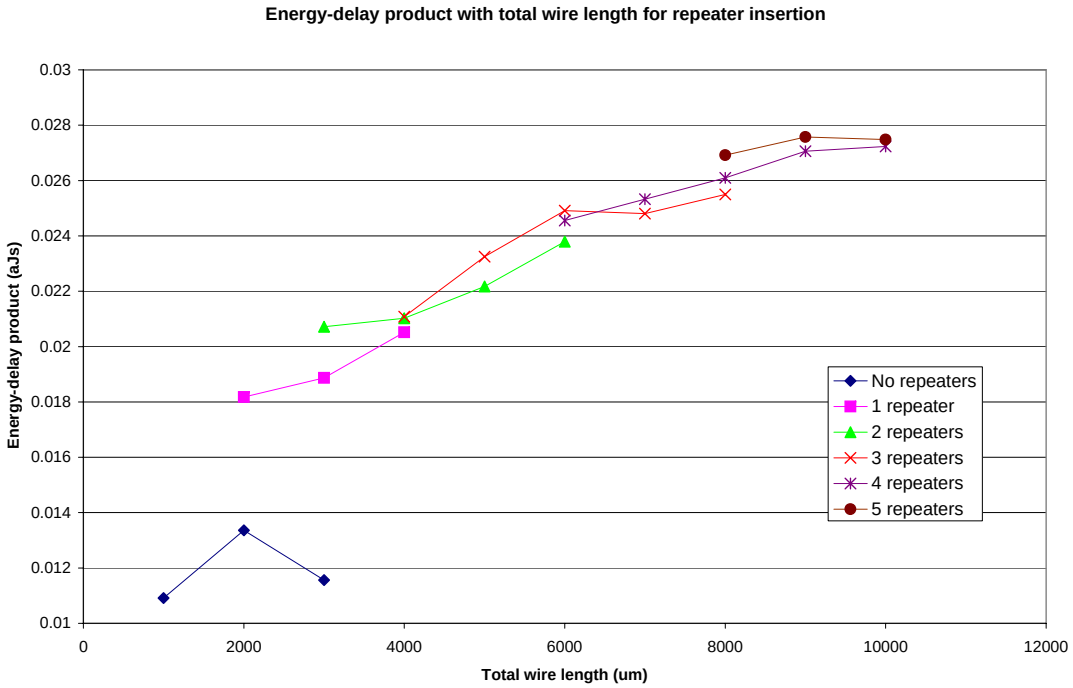


Figure 6.32: Energy-delay products for MUX-DEMUX link with total wire length, for varying numbers of repeaters inserted — flop pulse generator with minimum delay

6.12 Evaluation of the pulse-chopper based design

I now display a similar, but more complete analysis of the point-to-point link; the only difference being the use of chopper-based pulse-generators in the MUX and DEMUX elements, in place of the latch-based design. For a circuit diagram, see Figure 5.18(b). The implementation is very simple, but incorporates none of the over- and under-width pulse protection mechanisms of the flop-based design. However, fault-free data transfer using this generator can be observed from 0–3mm wire segment lengths with a minimal pulse width. Therefore, as long as we remain in this range, we can easily deploy and evaluate the chopper-based system. So, from hereon in, that is all we will consider.

6.12.1 Latency

As before, we evaluate the end-to-end latency of our link. We saw in Figure 6.27 that a pulse-chopper based implementation offers a lower latency than its latch-based counterpart, due to being more lightweight. In fact, the chopper line in that graph gives us all the latency figures we need for a standard implementation: that latency is 1.66ns at 0μm, rising quadratically to 2.09ns over 3000μm.

It is at this stage when it is interesting to consider the effect of changing the pulse width, to corroborate the, previously unjustified, assumption that the best performance over the range of interest is with a minimal pulse width. Therefore, I re-ran the latency simulations, with pulse widths increased by four and eight inverter delays, corresponding to observed DEMUX driven pulse widths of 262, 413 and 498ps, respectively (which I call 1, 3 and 5 delays). These figures are similar for the MUX and repeater widths, and so total (data + ack) pulse widths are approximately double this. Table 6.2 shows the actual figures. The latency results from this evaluation for the first bit are shown in Figure 6.33, and also for a full byte transfer (Figure 6.34). We see that the smallest available delay (blue diamonds) always gives the lowest end-to-end latency, and this can be easily reasoned about if we realise that a wider data driving pulse results in a delay before acknowledgements can be successfully transmitted. And, for all except the very first data bit through the network this holds up a bit's transmission.

So, for latency, the smaller the pulse width the better, as long as it is long enough to reliably transmit data.

6.12.2 Throughput

In Figure 6.35 we see how the value of the bit cycle time varies with the length of the interconnect wire. We saw earlier that the most effective and reliable operation occurs over the range 0–3000μm, and so that is the range we show here. As we also saw, practical pulse width sizes for this range are in the region 1–5 delay elements, and so we see curves for one, three and five delays here.

We observe many interesting things. Perhaps the most important are the general shape of the plots: they are all quadratics. Whilst this is hard to fully appreciate visually, a quadratic regression applied to the data yields an R^2 value of 0.999 in the one and three delay cases, and 0.998 in the five delay case. This suggests that the curves are indeed very good fits to a polynomial function. The regression also suggests that the x^2 term only dominates the linear term for lengths over 10mm for one delay, 5mm for three delays, and 1.5mm for five delays.

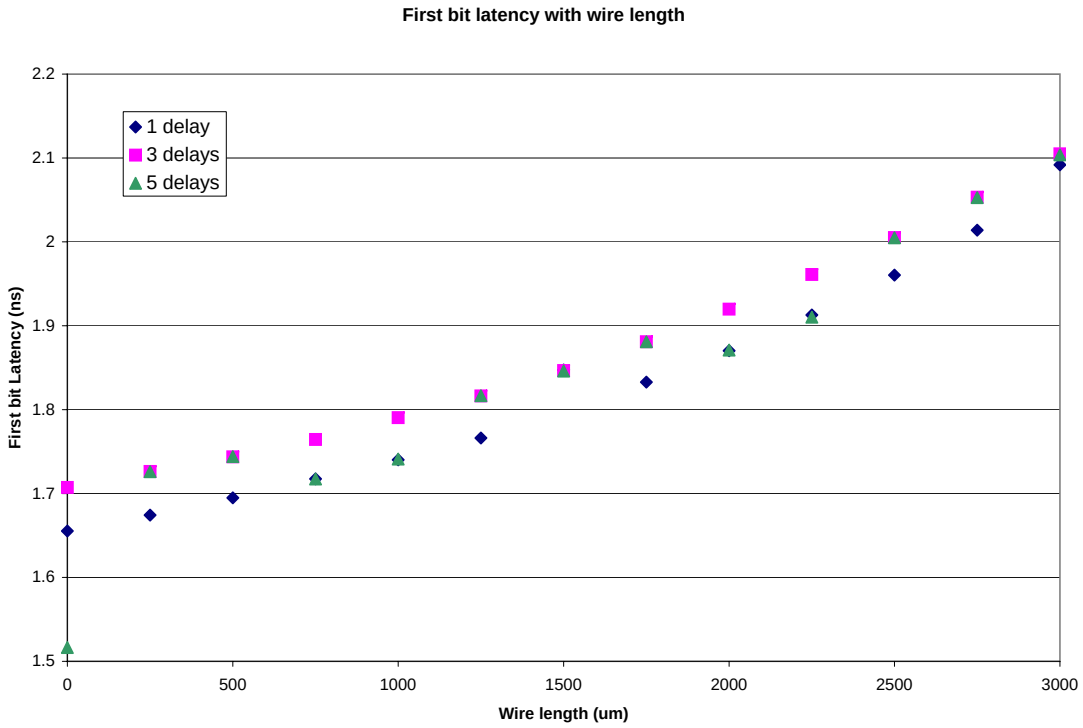


Figure 6.33: First bit latencies with unrepeated wire length – pulse chopper unrepeated base link

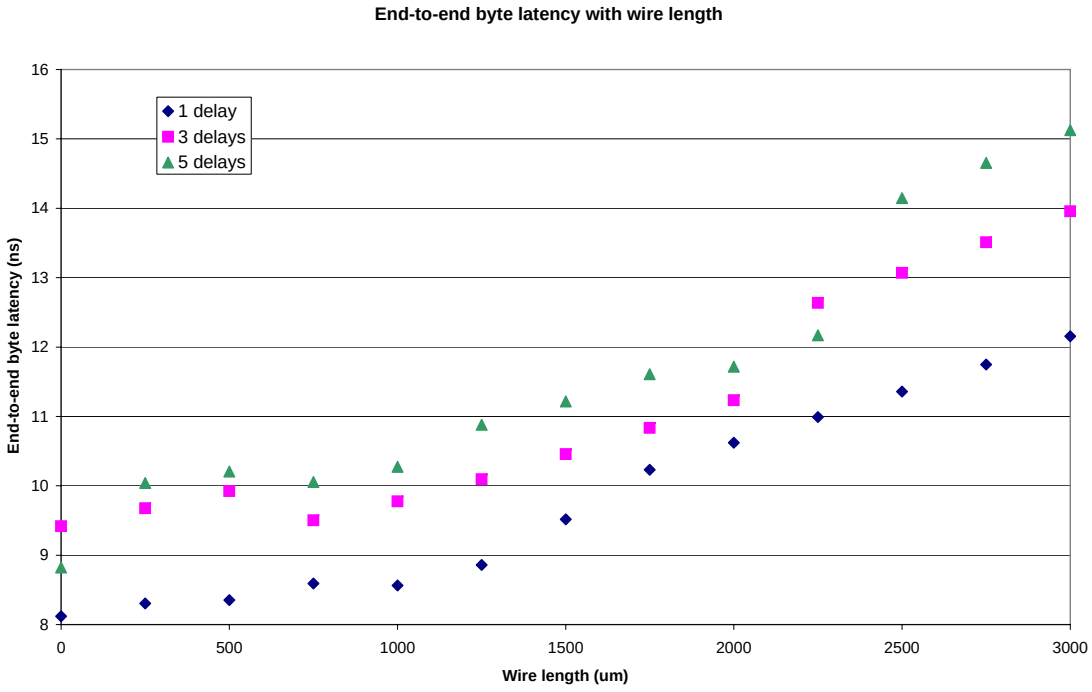


Figure 6.34: Byte end-to-end latencies with unrepeated wire length — pulse chopper unrepeated base link

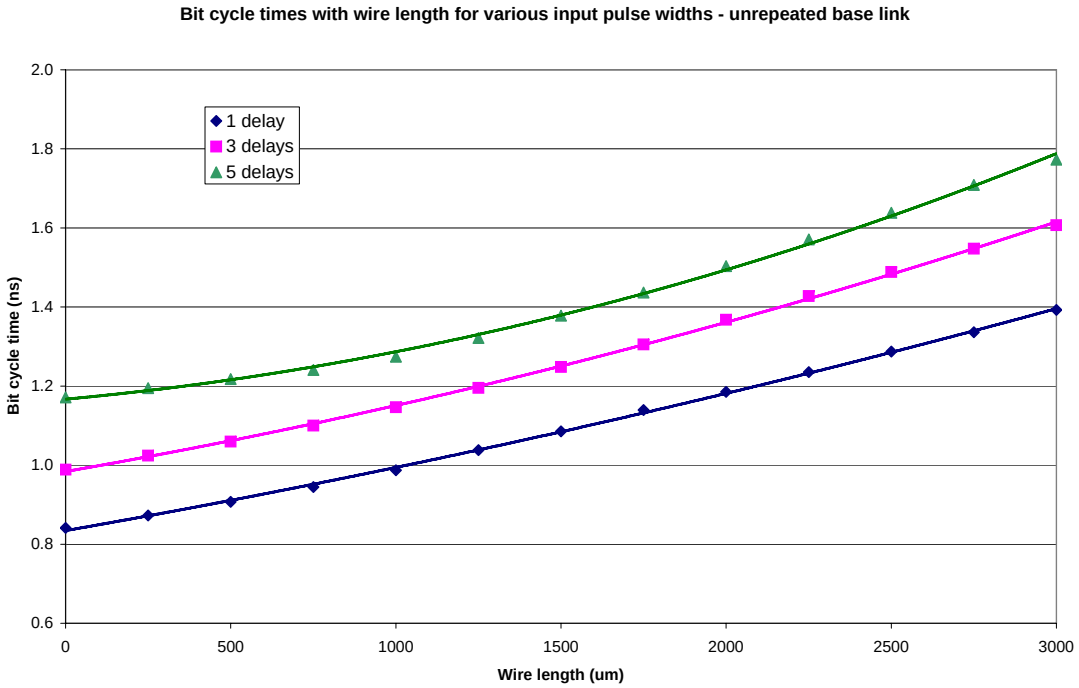


Figure 6.35: The effect of interconnect length on bit cycle times — pulse chopper unrepeated base link

This information is consistent with our hypothesis, since we expect the transmission time of a bit to rise quadratically with wire length, since the physical RC delay rises in this manner (see Section 4.2.1).

Again, we have already seen the performance of the vanilla chopper interconnect in Figure 6.26; the results are a bit cycle time of 841ps at 0μm, which rises to 1.39ns at 3000μm.

We can even derive a formula for the shape of these curves by recalling the sequence of events on our interconnect that leads to a bit transfer taking place. Illustrated in Figure 6.5, we see that the transmission of a data bit by the transmitter is followed by a acknowledgement transmission at the receiver. An acknowledgement is only generated after an internal logic delay at the receiver, and equally, subsequent data bits may only come after a transmitter logic delay. Added to these is the RC delay of the wire in each direction, which as a distributed capacitance takes the value $t = RC/2$. Thus:

$$\text{bit cycle time} = \frac{RC}{2} + \text{receiver logic delay} + \frac{RC}{2} + \text{transmitter logic delay} \tag{6.7}$$

however, we have so far ignored the interaction of the width of the data/acknowledge pulses. If the pulse width in a direction is longer than the wire $RC/2$ delay plus the logic delay of the end it arrives at, then it will cover these functions up and insert an additional delay, since the next phase of the protocol (in the opposing direction) cannot commence until the pulse has ceased.

Therefore, the complete equation describing a bit cycle time should be :

$$\begin{aligned}\text{bit cycle time} = \textbf{MAX}(\frac{RC}{2} + \text{receiver logic delay}\ ,\ \text{transmitter pulse width}) \\ + \textbf{MAX}(\frac{RC}{2} + \text{transmitter logic delay}\ ,\ \text{receiver pulse width})\end{aligned} \tag{6.8}$$

where **MAX** is the 'maximum' function, returning the greater of its two arguments.

Summary

We have compared the operational correctness and performance of two types of pulse generator circuit for deployment in a base link implementation. A pulse chopper circuit has been shown to operate correctly, and with good performance over the range of wire segment lengths 0-3mm. A flop-based pulse generator has also been shown to have good performance, but with reduced operational range, 1-2.5mm.

When repeated, the pulse chopper circuit has been shown to have the best performance and give the most reliable operation of the two designs, operating flawlessly with any number of repeaters, from zero to five over a design range sweep of 0–10mm, as long as segment lengths are kept no longer than 3mm.

The reliability problems with a flop-based design, however, highlight the hazards when deploying a stateful element with setup and hold time constraints in an otherwise asynchronous design. For this reason, and since the unrepeated performance is comparable, but the repeated performance better, I choose the pulse-chopper circuit for use with simulation for the remainder of the evaluations in this chapter.

For both implementations, we can simplify and improve performance by the use of repeaters, when long wire segments are involved. The following section will demonstrate their use and how performance scales with them.

6.13 Repeater insertion with the chopper-based link

Having evaluated the chopper link, we now consider how its performance and reachable span scales with the addition of repeater insertion. I have already introduced this concept in several sections, including Section 6.10, so I move straight on to the evaluation.

6.13.1 Latency

We saw that, for the flop-based system, latency was always increased by repeater insertion with segment lengths up to 2500μm, but that repeaters were able to extend the total wire length over which data could be transferred successfully. Therefore, we would expect to find similar results for the chopper-based implementation, since nothing fundamental has changed. And, in fact, we do find this, with latency increases of approximately 600ns per repeater, which is the same penalty as found during the loop oscillator evaluation.

In Figure 6.36 we see that very much the same kind of thing is going on. In contrast though, all lines are able to start at the 0μm point, before tailing off when per-segment lengths reach 3000μm. The general trend is that each additional repeater insertion permits an additional 3mm of length to be successfully traversed, whilst also very slightly shallowing the increase

of latency with length over the previous number of repeaters. If the regression line is to be believed (and there may not be sufficient data points to make it significant), at around 6mm a single repeater insertion provides a net latency improvement over the unrepeated version. This may be a moot point since unrepeated operation cannot occur up to this length and a repeater is demanded anyway, but it does neatly agree with the results we got for the loop oscillator previously.

6.13.2 Throughput

From the loop oscillator results, we already know that our stateful repeaters are very helpful from a throughput point of view; they are able to pipeline our interconnect and reduce bit cycle times. The question here then is, "By how much, and when is an increase in number useful?" We investigate this now.

Figure 6.37 shows a plot of bit cycle times for total wire lengths of 0–10mm, with 0–5 repeaters inserted. Peak throughputs are 1.01Gbit/s for 1mm, and 926Mbit/s for 10mm of wiring. We see that, as expected, for a given total wire length, repeater insertion increases interconnect throughput (decreases the cycle time), and the greater the number of repeaters, the higher the improvement; albeit with a trend of diminishing returns. This trend is easily explained since, whilst the first repeater insertion causes the average wire segment length to halve, the next one only causes a decrease by a third, then a quarter, then a fifth, and so on. Strictly, the reduction given by inserting the Nth repeater is $N/(N+1)$ of the previous value. And the counter-intuitive result of more repeaters decreasing bit cycle time when performance in a conventional system is normally hurt by this extra logic is due to the 'pipelining effect' of repeater insertion (see Section 5.11, and in particular Figure 5.14). The metric of bit cycle time talks only about the time taken between adjoining repeaters, and nothing about end-to-end performance.

We see once again that the graph shows data points for the respective repeater configurations until each reaches a segment length of greater than 3000μm, when operation ceases. This corresponds to the operational limit of a segment with a minimum pulse width.

An interesting point from the plot is that performance can always be increased by inserting additional repeaters — even when the wire length is zero. This phenomenon is caused by the fact that a repeater's input cycle time (data low to acknowledge high) is slightly faster than the DEMUX's. Therefore, a transmitting MUX will see a slightly higher performance than direct connection for some of the data bits being sent due to a decoupling effect provided by the repeaters.

6.13.3 Repeater logic delay

Repeaters may reduce wire delays by segmenting long sections, but they also insert an additional logic delay for each repeater used. If the logic delay outweighs the saving along the wire, then repeater insertion is pointless and power inefficient.

The stateful repeaters offer a forward (data path) logic delay of 639ps, independent of input pulse width, but rising with segment wire lengths. This is measured as the time from the 50% of `vdd` falling at the input, and the equivalent fall at the output. The loop oscillator result of Figure 6.12 shows this, and also that the backward delays have a very similar pattern. Points are taken from the corresponding maximum performance pulse widths.

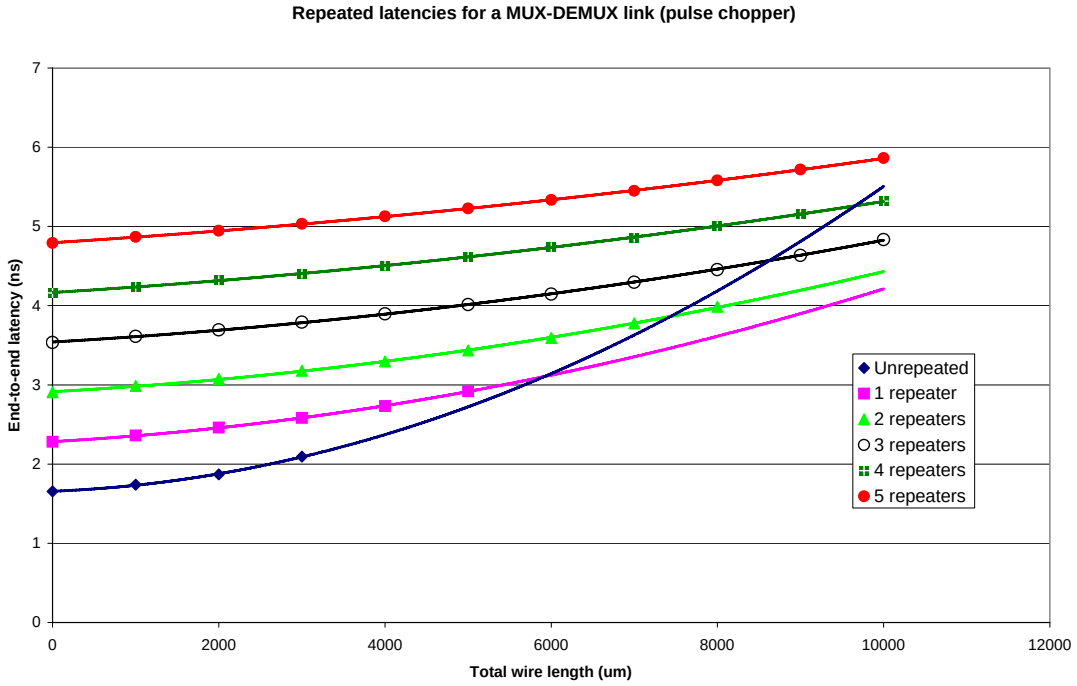


Figure 6.36: First bit latencies for the repeated interconnect

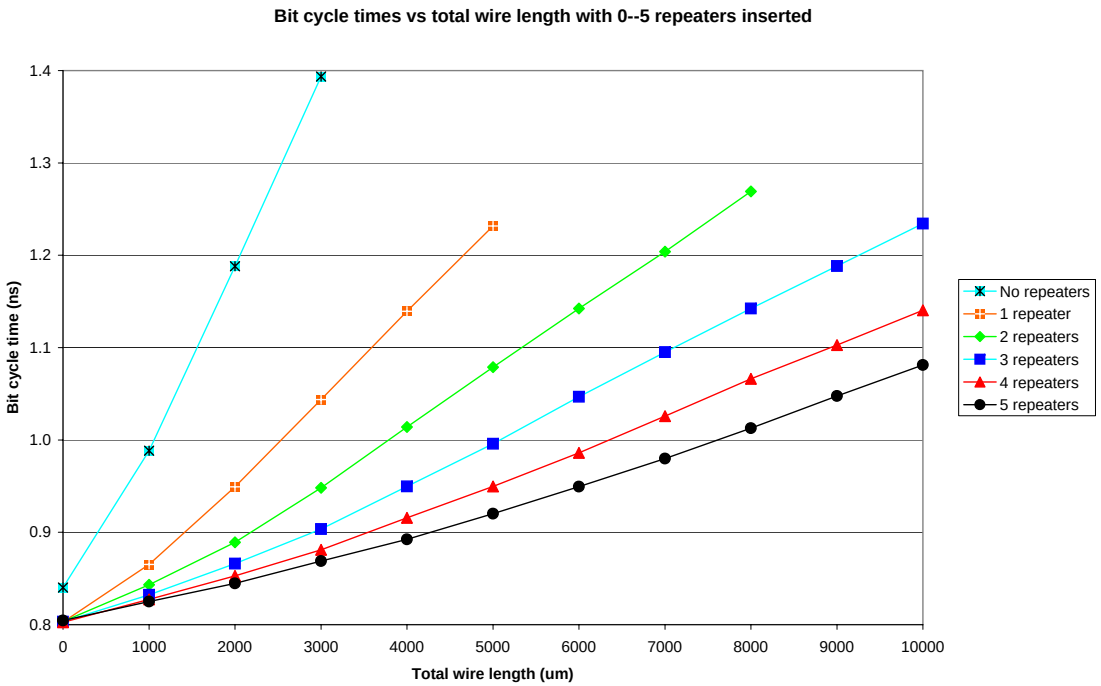


Figure 6.37: Bit cycle times for the repeated case — chopper pulse generator

6.13.4 Energy

Now that we have seen how the conventional performance metrics of latency and throughput apply to our vanilla and repeated links, it is time to move on to what, today, is perhaps *the* most important factor when designing a large-scale system: power consumption. Power constraints are the most pressing in today's design environment, and so it would be very nice if my system not only had good throughput characteristics, but also transferred data in an energy-efficient way. Therefore, we examine the metric of energy-delay product, which has been introduced previously as the best way of evaluating the efficiency of an implementation.

We have just seen that additional repeater insertion boosts throughput, so we may expect that as high a number of repeaters as possible would be optimal from an energy-delay product point of view, since (data→ack loop) delay per bit would be minimised.

Looking at the actual energy-delay product results (Figure 6.38) however, we see a different situation: the energy overhead of inserting additional repeater elements outweighs the increase in performance they provide. Further, for every repeater inserted, the higher the energy penalty, and there are no situations when it is advantageous from an energy-delay point of view to insert a repeater. Therefore, when implementing this interconnect, a designer must make a choice to where their priorities lie; power or performance?

This trade-off between power and performance is a classic dilemma for interconnect designers.

6.13.5 Energy use breakdown

To try and help a designer to make an informed decision when using my point-to-point interconnect, it is useful to understand where the energy consumption comes from. We will now see how the energy consumption breaks down, and what components in particular are energy intensive.

Wire driver energy

To start, we look at the straightforward Figure 6.39, where we plot the energy consumption per bit of the unrepeated full system, and also display how much of that consumption is actually being taken up by driving the wires. We see that, unhappily for an efficient point-to-point interconnect, only a fifth of the energy consumed is for this vital task.

More interestingly, if we actually look carefully at the energy consumed, we can see something a little odd: energy consumption increases with wire length, as we would expect, but only up to a point. After this, it remains more or less constant, despite the fact that the wire capacitance is still increasing. How can this be?

Well, theory says, by $E = CV^2/2$ [28, p.970], that if I drive the wire full swing up and down for one bit and 1mm, it should dissipate 1.011pJ of energy, and that this should scale linearly with wire length, since capacitance does, and all other variables remain constant. However, we have, so far, omitted any thought of the finite length of our driving pulses. So, to understand better, Figure 6.40 shows how the driver energy fares with different pulse widths.

If one subtracts the fixed internal logic power dissipation of the 0mm case (~1.5pJ with a `vdd` of 1.8V), we see that the actual wire dissipation with a single delay is 0.246pJ at 250μm, 0.501pJ at 500μm, 0.912pJ at 1000μm, increasing in even steps, until a break-point at around

1250μm, after which it tends to a constant value. In fact, this behaviour is mirrored by the 3 delay line, but with a higher asymptote; whilst the 5 delay line carries on growing. Why is this?

The answer is simple: energy consumption increases with the capacitance increase of the longer wires until there is no energy left to consume. At 1250μm, the charging time of the wire matches the pulse width. Given that the pulse width is fixed, longer wires will simply not be fully charged. However they will still absorb all of the energy[††] available in the pulse.

Let us consider two cases from the graph, when power is still increasing (l = 1mm), and when it becomes constant (l = 2mm). We know from before that at 1mm, $t = RC/2 = 33$ps (where t is the charging constant of the wire, equal to the time taken to charge to $\texttt{vdd}(1-(1/e))$ from 0V), and a pulse width of around 250ps easily charges the wire.

However, the situation is different at 2mm. Here, t = 132ps, which is approximately a half of the pulse width. Therefore, the wire charges for approximately $2t$. Now, in $2t$ the wire charges to only $\texttt{vdd}(1-(1/e^2)) = 1.56$V. So, we see that not all the wire capacitance is charged. Further, at 3mm, the calculation gives a swing of only 1.14V, which is very much reduced from the nominal 1.8V, and explains why 3mm is the extent of reliable operation with a minimum width pulse. For further enlightenment, the reader is referred to the plots of voltage swing presented before in Section 6.8.

This tells us two things: that the pulse width equals the charge time at 1250μm, and that the wire will be operating without a full voltage swing for longer lengths. The latter point is a boon and a potential for disaster. Low swing systems (see §4.3.1 for more details) offer reduced energy consumption and faster response times that their full-swing counterparts. However, since the difference between voltage levels is reduced, they are more susceptible to noise, and so erroneous operation may occur. This is one reason that there are minimum pulse widths for the varying lengths of wire in this system.

The '5 delay' line is an exception since its pulse is still wide enough to fully charge wires in excess of 3mm, and so it keeps increasing with length.

A final thing to mention is that the displayed value of 0.912pJ/bit/mm is well in keeping with the reported energy consumptions of contemporary high performance interconnects of around 0.8pJ/bit/mm for a charge and discharge cycle (at a vdd of 1.8V). For example, Ho et al. state [23] that, for a 0.18μm process, 1mm of semi-global interconnect has a capacitance of 414fF. So, if I charge and discharge this wire, then by $E = CV^2/2$, full swing consumption is 1.34pJ. And, recalling that the capacitance I extracted totals 3.08fF/mm, including mutuals, this calculation becomes 0.998pJ/mm. Both of these are close enough to the value of 0.912pJ/mm to give the author high levels of confidence in his model and simulation results.

Dynamic power from driver conflicts, compared to pulse widths

We have dealt with the plateau of driver energy with a fixed pulse width, but how about the rapid increase observed with increases in pulse width for a fixed wire length from Figure 6.19?

Well, Equation 6.8 shows that, once the pulse width exceeds the processing time of these components, bit cycle times increase linearly with pulse width. Additionally, being longer than the processing delay of the elements means that, say, a data pulse will still be being driven by the transmitting end when the receiving end is attempting to acknowledge it: and so conflict will occur on the wire, with both ends driving and excess energy being dissipated.

[††]Most electronics discussions would talk about current here, rather than energy. This is certainly also true, but the author stays with the terminology of 'energy' to avoid confusion.

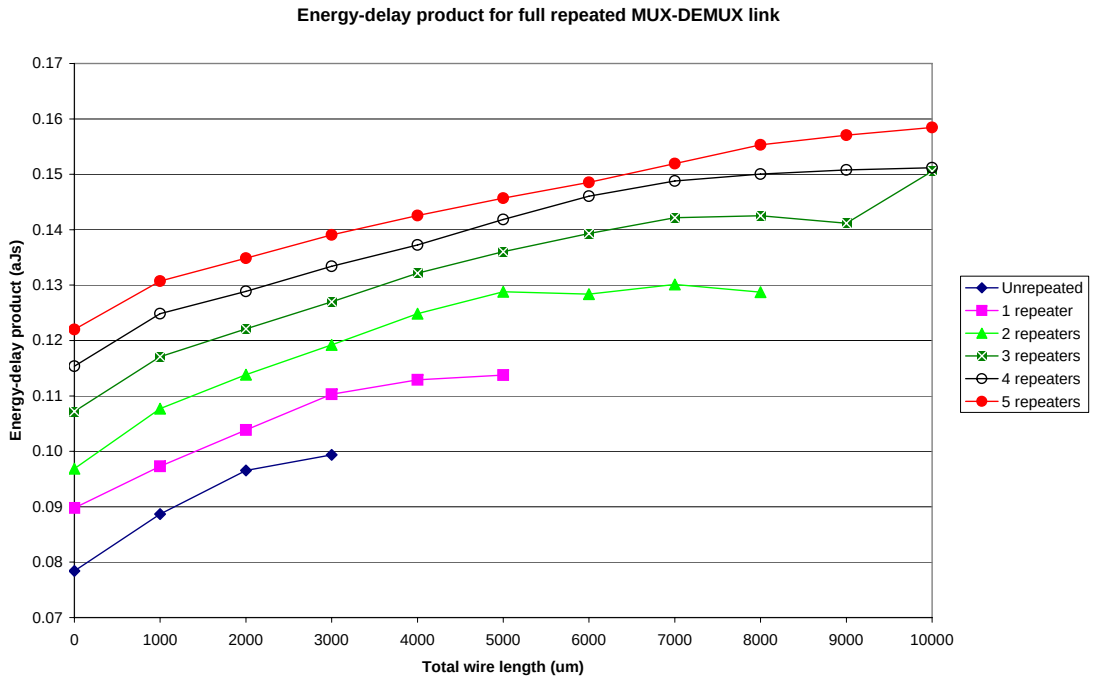


Figure 6.38: Full system energy-delay product for various repeater insertion configurations — chopper pulse generator, 1 delay

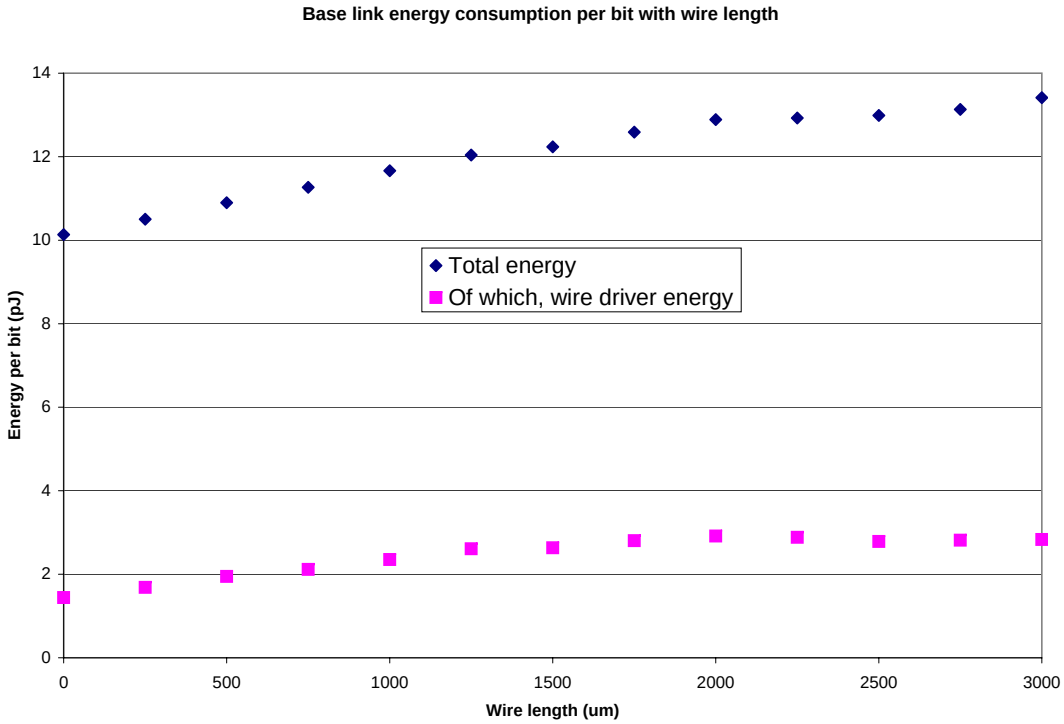


Figure 6.39: Base link energy consumption with wire length

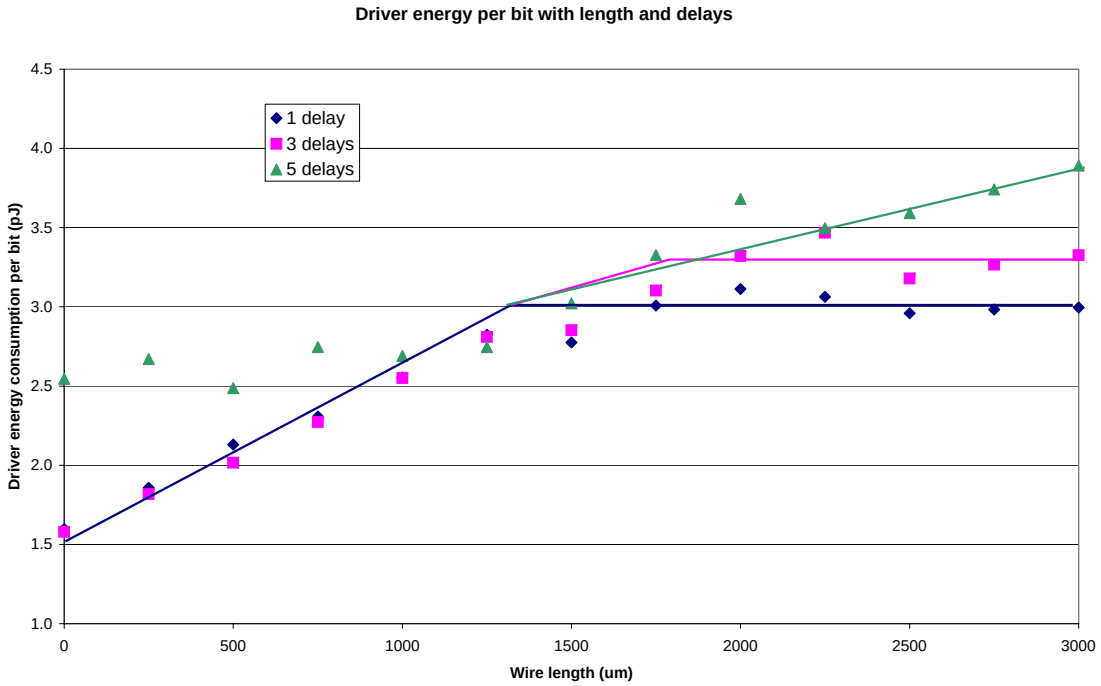


Figure 6.40: The increase in energy consumption of the wire drivers with wire length

The longer the pulse, the longer the conflict, in a linear relationship. Combined in a product with the linear increase of bit cycle time with width, and the energy-delay product shoots up quadratically; thus explaining the shape of the graph and illustrating, once again, the importance of picking the minimum sized delay to do the job with this interconnect.

The remaining energy

The remaining energy goes to the MUX and DEMUX logic, and I beg the reader's indulgence when I say that the exact breakdown will be presented later, in Chapter 8.

6.13.6 Repeater energies

How much energy goes on supplying our repeaters' logic? We see Figure 6.41, which informs us that, for most points, the energy increases by approximately 2pJ for every repeater inserted. The figures exclude energy consumed from driving wires. The value of 2pJ is, then, a good approximation to the energy repeater logic consumes per bit.

More interesting, perhaps is the general shape of the graph. Visible for the one, two and three repeater cases, and starting to be noticeable for the four number case as well is a dome-like shape to the curves. This tells us that repeater energy consumption starts off relatively low, then increases to a peak, before slowly dropping off again. The reason behind the initial rise is straightforward and stems from the fact that the repeater inputs slew less quickly when their wire segments become longer. This then leads to increased dynamic power dissipation in the repeater logic gates, as each spends more time with both pull-up and pull-down circuitry partially conducting. The fall-off is a little less obvious, but again relates the degradation

of input rise and fall times. When the wire becomes so long it is pushing the operational boundaries of a repeater's capability, the reduced swing voltage and rate causes the repeater input to see a shorter pulse than was transmitted. This results in the input detecting `data valid` for a reduced period of time (this may be reduced by as much as 40ps for an extra 1mm of wire length). This, reduced period then gives internal logic less time to settle, and some internal signals swing over a reduced range, suppressing total power consumption.

This energy consumption shape has a knock-on effect to the energy-delay product of the repeater configurations, and we see that Figure 6.42 shows the same form, albeit a little stretched out. The energy-delay product graph then, deserves no additional explanation.

Energy-delay product

A very similar analysis than for raw energy can be given for the energy-delay product, shown in Figure 6.19.

Essentially, all pulses in the flat range charge the wire fully, and so dissipate the same energy. They also all produce the same bit cycle time, and so their product is also constant.

Whilst for both these graphs, there are many more data points shown to improve the quality of the regression, reliable data transfer only actually occurs in the system with total pulse widths of less than 1.8ns, for both the matched and unmatched cases.

So, for 1000μm of wire length and no repeaters, the pulse widths should be less than 1ns in length to produce optimal performance. Note that this will be different once repeaters are inserted.

6.13.7 Logic delays

The logic delays for the MUX tree at maximum performance are 393ps, 381ps and 374ps, for segment lengths of 1000μm, 2000μm and 3000μm, respectively. The DEMUX tree is a little higher, at 434ps, 459ps and 473ps. Again, these are measured between successive falls past the 50% of `vdd` at their output / input nodes. Compared to the wire delay, these are high and explain why logic is on the critical path for short wire lengths. This is a price paid for having a wire-efficient implementation and robust operation at the same time.

We can perform a cursory correctness check of several of the results presented so far by seeing that the total of the logic and propagation delays equals the bit cycle time. The calculation is 393ps+434ps+(2×35) = 897ps, compared with a stated bit cycle time at 1000μm of 957/988ps (latch/chopper). The values are within margin of error of our first-order wire delay model, and so we are self-consistent.

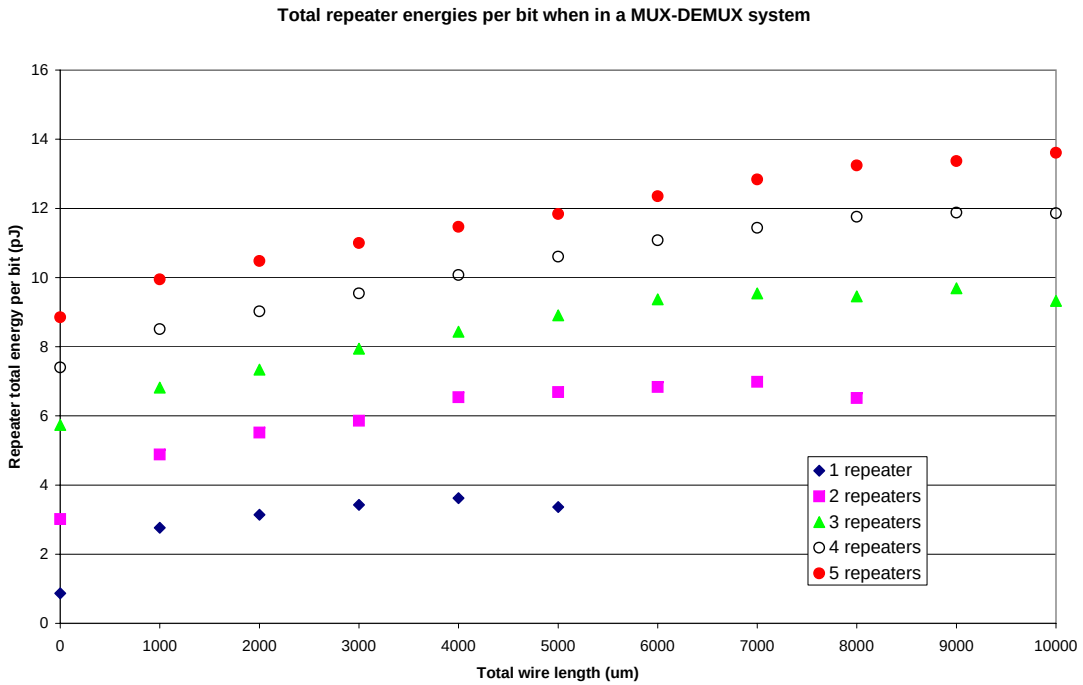


Figure 6.41: Energy per bit transferred of repeaters, excluding MUX and DEMUX logic

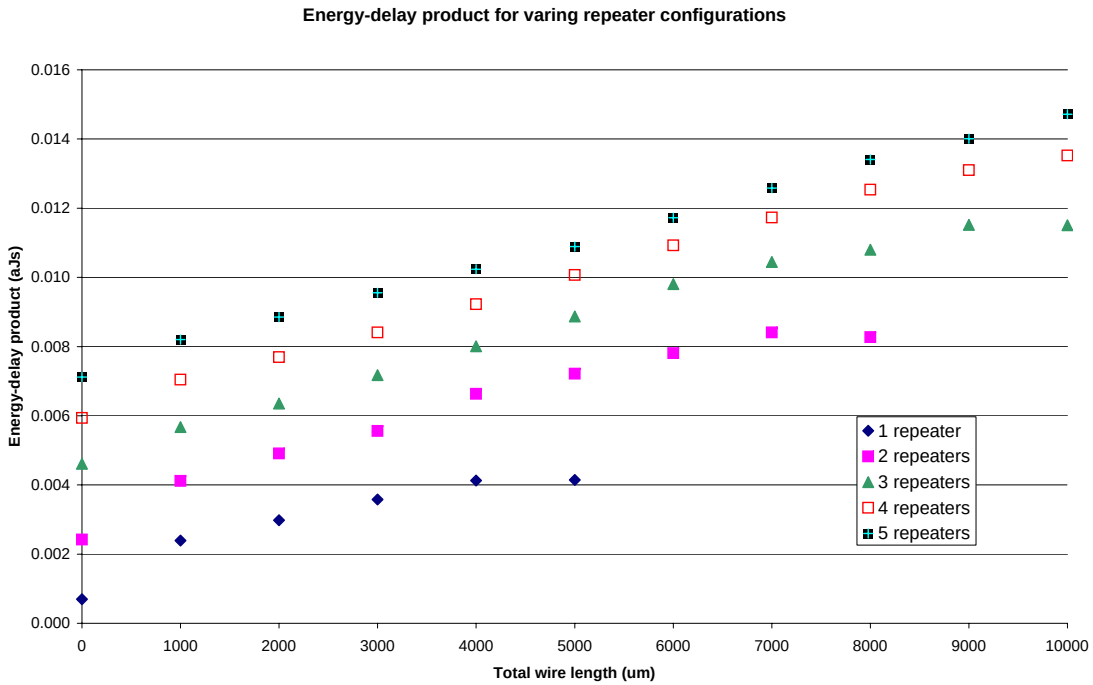


Figure 6.42: Repeater only (excluding MUX & DEMUX elements) energy-delay product for various repeater insertion configurations — chopper pulse generator, one delay

6.14 Area

Since the primary motivation for developing the interconnect system we have been evaluating was to provide a very area-efficient link, its area performance demands a close inspection. In this section, we will see that the system is, indeed, very area-efficient, when related to more conventional interconnection strategies.

I start by asserting that the area-efficiency we desire most is that of global metal footprint. Global metal is that which is available at the highest metal layers (metals 5 & 6 in the technology we consider), and is typically used to provide wide, low resistance tracks for signals of a global nature. Good examples of its use are for power nets and long-distance interconnects. Since the interconnect we are considering is designed for long distance, any implementation would typically be routed on global metal layers.

We have seen previously that a reduction in the area footprint of a channel using these layers offers a multitude of benefits: from offering space for additional channels; to minimising wire delays via increasing trace sizes; to a reduction in inter-trace noise. All of these are enabled by area-efficient channels.

Conversely, logic area is expendable. With every shrink of a process technology, more and more transistors become available to be used. This is not so for wiring, which remains a more-or-less fixed resource, in terms of quantity and, often, size. However, smaller transistors make weaker drivers, and so are less able to drive their wires quickly. The explanation for this is simple under the constant field model of scaling, as discussed by Streetman and Banerjee [57, p.307–316]. Both transistor width and length are reduced by a scaling factor α, leading to a transistor with the same bulk resistance as the original one. Without the additional reduction of gate oxide thickness, the electric field will not be strong enough to firmly control transistor operation. However, if the gate oxide thickness is reduced, then the supply voltage must also be to avoid dielectric breakdown under a higher field strength. This also has the side effect of keeping leakage currents under control. Since the overall resistance of the transistor has remained largely unchanged, the reduction in supply voltage lowers the current driving ability of the transistor. Under the assumption that global wires do not scale with transistors and that their capacitance remains roughly constant with shrinks, the end result is a reduced driving ability, and increased rise and fall times on the wire. Naturally, this implies that unrepeated wires suffer in performance.

Under these conditions, we should be more than happy to burn transistors if it means a reduction in the number of global wires that have to be routed and driven. This is the approach my interconnect takes. We will now see that it does, indeed, save a large amount of global wiring area over a parallel interconnect, and also saves in overall area after wiring lengths of around 1000μm.

In Table 6.4, we see the area requirements of the interconnect we have been considering, alongside those of a parallel interconnect to transfer the same quantity of data. Area usage is displayed for both the native 8-bit width of the system, and also for a 32-bit width, as may be more applicable for high-end designs. The interconnect is scaled from eight to thirty-two bits by aggregating four copies of the structure in parallel, and so it uses a total of eight wires. For comparison, the area use of a standard, parallel interconnect is shown, and we consider both one-way (simplex) and bi-directional (duplex) schemes. We see data for lengths of 1mm and 5mm, and the area usage values are split between logic area (transistor area) and wire area (global metal area), with a total being available on the right. Estimates were produced by

Table 6.4: Area requirements (in μm^2) for various configurations

Configuration	Logic Area	Wire Area	Total Area
8-bit, simplex, this scheme, 1mm	5,984	1,320	7,304
8-bit, simplex, std. parallel, 1mm	416	6,925	7,341
32-bit, duplex, this scheme, 1mm	47,872	10,560	58,432
32-bit, duplex, std. parallel, 1mm	3,328	55,440	58,768
32-bit, duplex, this scheme, 5mm	47,872	52,800	100,672
32-bit, duplex, std. parallel, 5mm	4,560	277,000	281,560

summing the contributions from standard cell footprints, where available, and an estimate of the size of any custom logic used.
The results show several things:

- This interconnect design always consumes less global metal area than a parallel design (since it is serial based);
- It also always requires more logic area than an equivalent parallel design (since it must perform parallel-to-serial and serial-to-parallel conversion);
- The total area is always less than that for a parallel interconnect (so it has no bad use cases);
- As wire lengths increase, so does the improvement my system offers over that of parallel interconnect — at 1mm, they are comparable in total area, but at 5mm, my system requires just over a third of the total area of a parallel system (and under a fifth of the global metal area).

So, my interconnect system always requires around a *fifth* of the metal of a similar parallel interconnect, and thus is *very* area-efficient.

6.15 Theoretical analysis of interconnect efficiency

Ho [22, p.20] gives the following equation for the maximum possible bandwidth for unrepeated wires, such as those discussed in this section. The formula[‡‡] was previously seen as Equation 4.24:

$$\text{Bandwidth over area} = \frac{1}{3(\text{FO4} + \text{wire\_delay})} \times \frac{\text{Block\_width}}{w + s} \tag{6.9}$$

It is interesting to calculate the maximum theoretical bandwidth of the proposed interconnect and a traditional parallel scheme, and compare both to the measured available bandwidths. This will generate a measure of interconnect implementation efficiency.

The table below shows the result of the theoretical bandwidth calculations for both the distributed and lumped RC wire models.

[‡‡] Ho's actual formula states $R_{wire}C_{wire}/2$, the equation for a distributed wire delay, where I write 'wire_delay'. I substitute this term both for clarity and to generalise the expression somewhat.

Table 6.5: Theoretical maximum bandwidths for a single minimal-sized, minimal-spaced wire

Wire length	Wire model	Theoretical Maximum Summary
1000μm	distributed RC	3.25Gbit/s
10000μm	distributed RC	94.6Mbit/s
1000μm	lumped RC	2.43Gbit/s
10000μm	lumped RC	47.8Mbit/s

Table 6.6: Theoretical maximum bandwidths for a single minimal-sized, minimal-spaced wire — inter-symbol interference insignificant

Wire length	Wire model	Theoretical Maximum Bandwidth
1000μm	distributed RC	9.75Gbit/s
10000μm	distributed RC	284Mbit/s
1000μm	lumped RC	4.88Gbit/s
10000μm	lumped RC	142Mbit/s

We see that the models are all comparable in value for short wires (1000μm), but the variation rises dramatically for a longer wire, to a maximum variation of 91% between the distributed RC and lumped RC models. This result shows the importance of choosing a model carefully, and the negative impact that inductance can have on an interconnect signal's performance.

If we now look at the observed throughput of the area-efficient interconnect, we can get an implementation efficiency ratio. We saw in Section 6.5 that the loop repeater throughput is measured at 1.33Gbit/s for a 1000μm link, and scales down to 177Mbit/s at the 10,000μm length.

We see immediately that there must be something wrong here, since observed throughput at 10mm is around three times the theoretical maximum! Rather than being an error in simulation, this error derives from Ho's overly-pessimistic bandwidth calculation. Equation 6.9 includes a '3' under the line to account for the settling time of a wire, in order to prevent inter-symbol interference. However, observations show that this is an unnecessary precaution with my system, since rise and fall times are too slow to cause reflections of any appreciable magnitude. Therefore, this tripling constant can be removed safely, and the values for our table recalculated in Table 6.6.

We now see all values treble, and whilst this much reduces the efficiency of my interconnect at short wire length to 15% of theoretical maximum, it runs at a plausable 91% efficiency at 10mm, again, assuming the distributed RC model. However, since two wires are used in our interconnect (and only one is ever active), across the link area, these efficiencies halve to 7.5% and 45%, respectively.

Note two things about this figure: the per wire bandwidth is twice that, and is actually dominated by logic delays at short wire lengths, but these are easily amortised at lengths as large as 10mm (see §6.13.7). Were the critical path to be the wires themselves, the efficiency would increase dramatically. A fully optimised logic design for the proposed interconnect

would, therefore, offer much high wire efficiencies.

To place the performance in relation to other designs with practical implementations, let us first consider what it means to operate at the theoretical maximum. For any system, it is the maximum possible bandwidth, but for a synchronous system it also subsumes the clock frequency. Therefore, a synchronous serial link running over two wires, one for data and one for a clock, could possibly run at the 4Gbit/s mark. However, this performance is unlikely to be achieved for many reasons. First, the data sourcing and sinking logic (for a synchronous serial link this would be a parallel-to-serial converter and vice-versa at the receiving end) must be capable of supporting this rate. For example, the standard cell latches in our 0.18μm technology have a maximum clock rate of very close to 1GHz, and so would be unable to support data transfer faster that this. Whilst full-custom design may be able to improve over this many-fold, the effort may be undesirable, and timing margins, and therefore safety, would decrease.

Second, having two very quickly transitioning wires in such close proximity to each other as would be the case for a serial interconnect would pose immense problems through cross-coupling and the effects of inductance and capacitance at high frequencies. The noise impressed could rapidly rise to the point of causing false transitions at a receiver (Ho's '3' covers only self-impedance induced inter-symbol noise).

A third major problem is the need to distribute both clock and signal with very low relative skew, if reliable data transmission is to be assured. This problem encompasses those of matched layout and process variations. My interconnect solution is much more tolerant to these.

Finally, recall that the intended application for the area-efficient interconnect presented in this thesis is ASIC design. Many ASICs will not run at a high frequency, and so it may often be the case that an interconnect with very performance will remain idle for the majority of the time, and so nothing is gained from improving the available bandwidth — better to have a more reliable and modular approach, such as the system I propose.

All of these issues lead us to the conclusion that the efficiency of the proposed area-efficient interconnect, whilst appearing low, is in fact competitive with the other realistic designs available.

6.16 Comparison to other interconnect designs

How does this point-to-point interconnect compare to other common interconnect designs? In this section, I provide a brief summary of the nominal performances of the two main interconnect paradigms (serial and parallel), plus that of GasP. I evaluate each for its performance when providing an 8-bit point-to-point link. For energy, I assume that a bit value toggles in the cycle under consideration.

6.16.1 Parallel wires

The simplest interconnect to transfer eight bits of data is a parallel interconnect. Consisting of eight data wires and one or two control wires, (near) simultaneous transitions on all wires allows the transmission of a byte of data. Most commonly, one control wire is used, to transmit a synchronous clock signal to a receiving end, informing it when a new data bit has arrived on a wire.

Needless to say, this methodology, when used over long wire lengths, runs rapidly into problems of data skew and other signal integrity issues, as we have discussed in previous chapters.

An alternative is to use two control wires and an asynchronous transmission style, with one wire being a `request` wire, and the other an `acknowledge`.

We look first at the synchronous design, which requires eight data wires and a clock wire to do the job.

Area: 9 global metal wires + 9 wire drivers + receiving latch logic.

Latency: 1 Driver delay + 1 wire delay + 1 latch delay

Throughput: $8B$, where B is the maximum theoretical throughput given by Ho's equation, and assuming that the clock runs at BHz.

Energy: $1\frac{1}{8}$ transitions per bit. However, crosstalk is likely to increase this value. Therefore, energy consumption is $1\frac{1}{8} \times \frac{\mathtt{vdd}^2 \times C_{wire}}{2}$ Joules/bit.

6.16.2 Synchronous serial interconnect

In this approach, I consider the use of a single data wire, with serial data produced by a parallel to serial converter feeding a serial to parallel converter at the far end. Bits are clocked using the signal on a second wire.

Area: Two global metal wires + parallel to serial conversion logic (~ 8 flops), + serial to parallel (~ 8 flops) + a misc. control logic (~ 8 more flops)

Latency: Assuming that the clock runs at the maximum theoretical frequency B, latency is $8/B$ + 2(latch delay) + 1 wire delay

Throughput: Bbit/s

Energy: 2 transitions are per bit transferred (one clock plus one data bit). So, energy consumption is $2 \times \frac{\mathtt{vdd}^2 \times C_{wire}}{2}$ Joules/bit.

6.16.3 GasP

Since the GasP system was the basis for the development of my link, it is only fair to compare with it. GasP uses a parallel data path, but replaces the clock with an advanced handshaking system, using just one extra wire.

Area: 9 global metal wires + 9 wire drivers + receiving latch logic + the inverters needed for the control logic, including a delay line

Latency: Driver latency + wire delay + the delay of the '3 inverter loop' (approx 3 gate delays)

Throughput: $\frac{8}{\frac{1}{B}+3\times\text{inverter delay}}$

Energy: $1\frac{1}{16}$ transitions per bit (GasP uses double edged control). Plus the energy to overcome crosstalk. Total is thus, $> 1\frac{1}{16} \times \frac{\mathtt{vdd}^2 \times C_{wire}}{2}$ Joules.

6.17 Summary

We have seen the throughput for a parallel interconnect can be very high, but recall that the global metal footprint it has is over four times that of my system. If we were to use this area for my link, then extraction shows that we could:

- Double space the lines, resulting in a reduction of capacitance to two-thirds, and an associated reduction in wire delay.
- Quadruple space the lines. The wire capacitance halves, and therefore so does the wire delay.
- Double space the lines *and* double their width, halving resistance as well. Here, we get new values of $2C/3$ and $R/2$, resulting in a hugely improved wire delay of a one third the original wire delay.

So, my link could easily make use of this area and, over long distances increase performance by 2–2.5 times.

Finally, we have to consider if any of this optimisation is necessary — recall our target application is to transfer somewhere around 500Mbit/s. All designs have already exceeded this rate, and so we should be happy to sit back and enjoy the area efficient benefits of the interconnect evaluated in this chapter. Anything more is just a waste of wires and power.

RasP: a network-on-chip implementation

7

Parvis imbutus tentabis grandia tutus
(When you have mastered small things, you shall attempt great ones safely)

Motto of Barnard Castle School, Co. Durham, UK

In the previous chapters, I have introduced my point-to-point link, based loosely on the GasP control system. I have shown that it has good performance and, critically, that it is extremely area-efficient.

Now that we have such a basic link, can we apply it to a large-scale problem, such as the issue of chip-wide communication, encountered by modern IC designers?

The answer is 'yes' but, before we see how it can be done, let us first take in an overview of the problem. Modern chips typically incorporate a large number of logic blocks, each of which may wish to communicate with any of the others. As process shrinks occur, the number of such blocks increases rapidly, to the point where a traditional interconnect (such as a bus) becomes intractable due to scale.

Therefore, many modern designs make use of the Network-on-Chip (NoC) paradigm (see Section 3.4.2 for an introduction to NoCs) to provide an interconnection fabric between logic blocks. As an example of how ubiquitous NoCs now are, Dally and Towles show in their 2001 paper [14] that the scalability and ease of interconnecting many IP blocks enabled by NoCs means they will soon become indispensable. The overriding reason for which is the plethora of desirable properties they offer, some examples of which are given below.

- Scalability — since NoCs often offer a regular structure, an increase of blocks to be interconnected increases the number of nodes in a system linearly. Therefore, NoC overheads grow no faster than the logic they support, even with the increases in density given by process shrinks;
- Regular layout — this is similar to scalability, but also makes design and layout easier;
- Global interconnections — even logic with only a local-scale connection to a NoC fabric is able to route data to any block on-chip;
- Manageable complexity — for each node and IP block, only local relationships need to be considered for routing, timing closure, etc. This eases design and verification;
- The ability for dedicated interconnections between blocks to be supported if needed — guarantees of bandwidth, latency etc. can be offered via virtual networks without the need for additional point-to-point links;

- A large total bandwidth — the NoC bandwidth is up to the sum of the links in the system;
- Potential run-time configurability of topology and characteristics — a 'soft fabric' can allow dynamic resource allocation, and give a higher utilisation ratio than a set of dedicated links could offer;
- Often, a GALS based approach is used (see Section 2.4.1 for details on GALS) — this allows multiple clock domains to be interconnected without detailed knowledge of timing required at any logic node. This reduces logic system complexity, and speeds design.

Standard on-chip buses are unable to offer all these advantages. For buses, the complexity and scalability issues are particularly acute, since link quality degradation and the number of connections required increases rapidly with additional nodes.

The NoC trend offers scalable multipoint-to-multipoint connections, with ease of bandwidth aggregation. Most of all, perhaps, NoCs offer a template-style implementation, where black-box routers and data lanes can be placed in pre-reserved areas of a floor-plan, in a very predictable manner. The routers or data aggregators then offer standard input and output interfaces, allowing those blocks supplying data to — or sinking data from — them to be swapped around or modified without affecting the connections between the other blocks in a system. This makes a layout designer's job much easier, allows design changes late in the flow, and allows multiple IP providers to connect together seamlessly. Further, the a priori knowledge of the floor-planning needed and a good estimation of the time needed for implementation using black-boxed NoCs allows design projects to be more predictable and the engineering resources and time required to be estimated more accurately. This, of course, offers cost savings.

Their desirable properties mean that there are many NoC implementations now available (and Bjerregaard catalogues a few types [5]), but the most popular and effective choice in the asynchronous logic space is the Chain interconnect system from the University of Manchester. I now describe it.

7.1 Introducing the Chain interconnect system

Bainbridge et al. have spent many years developing the *Chain* interconnection system [3]. Intended as a scalable system to connect together logic blocks from multiple IP providers, it provides standard interfaces to blocks which, critically, need not all operate at the same frequency. This speed-independent nature gives it great flexibility when composing systems whose exact implementation details are not necessarily know early on in the design process.

Chain consists of a sequence of modular blocks, each with a specific purpose. In their paper, Bainbridge et al. demonstrate a pipeline latch, router, arbiter, and multiplexer [3]. Each use C-elements [56] as their basic gate, and all are connected together with a one-of-five data encoding, which allows two data bits to be transmitted per cycle. This wire-intensive arrangement makes it highly robust, and insensitive to delays of any block or data producer or consumer. It uses a 4-phase signalling protocol (this was described in Section 5.1), entailing a performance overhead, since all active wires must complete two transitions to transfer a single data symbol.

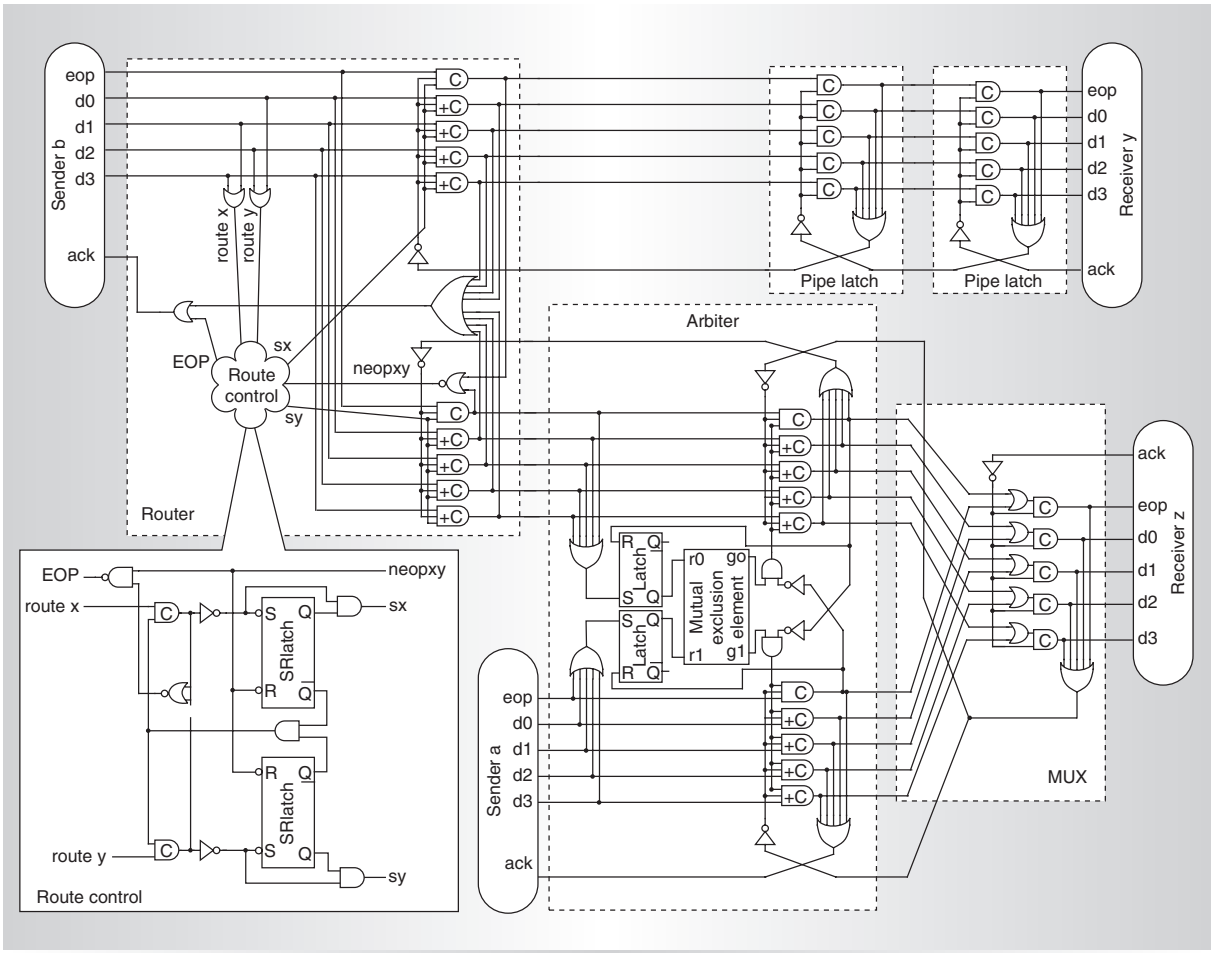


Figure 7.1: Chain implementation collage (Reproduced from [3])

As mentioned, Chain uses *one-of-five encoding* [56] for its data transmission, which requires five data wires to transmit two bits of data. Further, they add an acknowledgement wire, bringing the total to *six wires* (for only two bits of data). Chain, unfortunately, suffers performance penalties: asynchronous designs are based on *C-elements*, state-holding multiple-input gates. C-elements are generally slow, and so Chain's latency and cycle times are dominated by these sluggish components, even when properly buffered. A full performance evaluation of Chain will be displayed in the next chapter.

So far we have seen that Chain is indeed a flexible system, but lacks very obviously in one respect: area-efficiency. Nothing ever comes for free, and asynchronous logic is no different: the price paid for clock independence is area and, potentially, power. Asynchronous logic typically takes around fifty percent more than that of more conventional types. Amazingly, this does not always entail a power disadvantage, since asynchronous designs consume switching power only when there is data to be processed.

I show a collage, reproduced from Bainbridge and Furber's paper [3], of the reference Chain configuration as Figure 7.1. To ensure full fairness of comparison, I implemented the Chain elements as shown, but with the configuration modified to mirror the RasP test system. I will explain the modification in more detail in Section 8.7.

One thing is not clear from the collage, so I mention it here: Chain deletes the first symbol of every packet in the router, using its contents to determining the routing path. Thus, complex paths can be made by appending multiple preamble symbols, with the associated packet overhead.

One of the greatest motivators for choosing Chain as a comparison system is that, like our point-to-point link, it does not make use of conventional wire repeaters. In fact, Chain employs the same technique of pipelining, with elements its designers call *pipe latches*. Evaluations in the next chapter will show how these affect Chain's performance in a very similar manner to the stateful repeaters used for our link.

7.2 Improving Chain

Chain, with its speed-independent design, neatly addresses the issues of clock skew and composability. These advantages go on to give it scalability and a low implementation complexity. However, it lacks in one crucial respect: efficiency of global metal wiring. With its one-of-five encoding plus an acknowledgement signal, Chain uses six wires to transfer two bits. Its efficiency, therefore, is three wires per bit, and we note that this is worse than the two wires per bit offered by out point-to-point link of Chapter 5.

The most important trade-offs for NoCs mirror those of a point-to-point link: throughput-per-area, and latency-per-area. Therefore, it is natural to wonder if my point-to-point link would do a better job of producing an NoC than Chain does with its native signalling scheme. The Chain elements neatly address the problems of scalability and data routing, and so it is helpful choose to use it as a blueprint for a new NoC implementation.

In the remainder of this chapter, I will present *RasP*, an area-efficient NoC, based on Chain. I show its components, and evaluate their performance individually. Then, in Chapter 8, we will see how RasP performs when compared with Chain.

7.3 RasP: a network-on-chip implementation

I have demonstrated in Chapter 5 my highly area-efficient point-to-point serial link. It requires only *two wires* to transfer a data word, giving it tremendous area and performance advantages over more traditional forms of interconnect.

We will now consider how to scale my point-to-point link into RasP: a network-on-chip application that inherits the properties of area-efficiency, high bandwidth, routability and signal integrity. In comparison to more traditional Carallel interconnect structures it will occupy only 20% of the global metal footprint, and a comprable amount of logic area, for line lengths of up to 1mm. At lengths greater than this, its implementation area becomes significantly less than that of a parallel interconnect (see Table 6.4), whilst retaining the high bandwidth and throughput characteristics that are so important for NoCs.

My basic point-to-point link is ideal for deployment as the node-to-node interconnection of a NoC. Its clockless operation gives it numerous advantages over conventional NoC implementations, but the two most important are the ability to cross arbitrary clock domains and tolerate skew, plus the flexibility of on-demand operation, made available by the use of handshakes rather than clock-synchronous operation.

I briefly outline these and other basic link advantages now:

- The link internally runs our pulse-based, dual-rail signalling protocol. Conversion from a parallel format occurs immediately after data needs to be transferred, and data is only restored to parallel at the very last opportunity before reaching the receiver. This means that all nodes operate on, and are interconnected by, dual-rail signals. Therefore, the metal area footprint of RasP is much smaller than an alternative parallel fabric; and so it retains the area-efficiency of its base link.

- Arbitrary clock-domain crossing enables the link to interconnect logical or IP blocks running at different clock frequencies without any compositional problems being posed. With my system, the block frequencies need not have any known relationship (e.g., rational clocks) for reliable operation. This property also extends to provide skew-tolerant design.

- The free-running operation of the link means that it always offers the highest possible throughput to the blocks individually, and to the system as a whole. The effective frequency of the individual links to data sources and sinks will always run at the maximum rate at which data can be provided or consumed. This minimises latency and maximises throughput at the nodes. Similarly, the routers and arbiters of RasP operate as soon as data arrives at them, and not at some clock cycle boundary. This again maximises throughput and minimises latency, which is such a critical parameter when designing an NoC implementation. With RasP it comes 'free by construction'.

- The modularity of the link allows plug-in operation at design time. Since each node operates independently from a designer's point of view, they need only concern themselves with the most local of connections and dependencies. This greatly speeds up design time, reduces complexity and increases the chance of the system working first time.

- Scaling the link is trivial: simply add more nodes, and RasP will do the rest. This is in main part due to its self-synchronising nature via clock-free operation. An increased number of nodes naturally leads to increased contention for resources, but a sensible structuring of routers and arbiters will allow this to be controlled. For a good example of this, look at tree arbitration, as described in [2; 34; 47].

7.3.1 Overview of RasP

RasP consists of several basic elements: routers, arbiters and repeaters. To explain their interaction and operation, I start with a block diagram of its overall form. Shown as Figure 7.2, we see a sample configuration with two data providers and four data consumers. Data flows left-to-right, along a path selected by a configurable address.

The occurrence of data fan-in and fan-out requires that some scheduling and/or arbitration be performed. This is accomplished in RasP by a combination of routers and arbiters. A router is able to take in a stream of data and an address, which may either be encoded in the data stream, or transmitted via separate address wires, and selects one out of four mutually-exclusive outputs. In this way, the data has 'value added', and routing is possible to

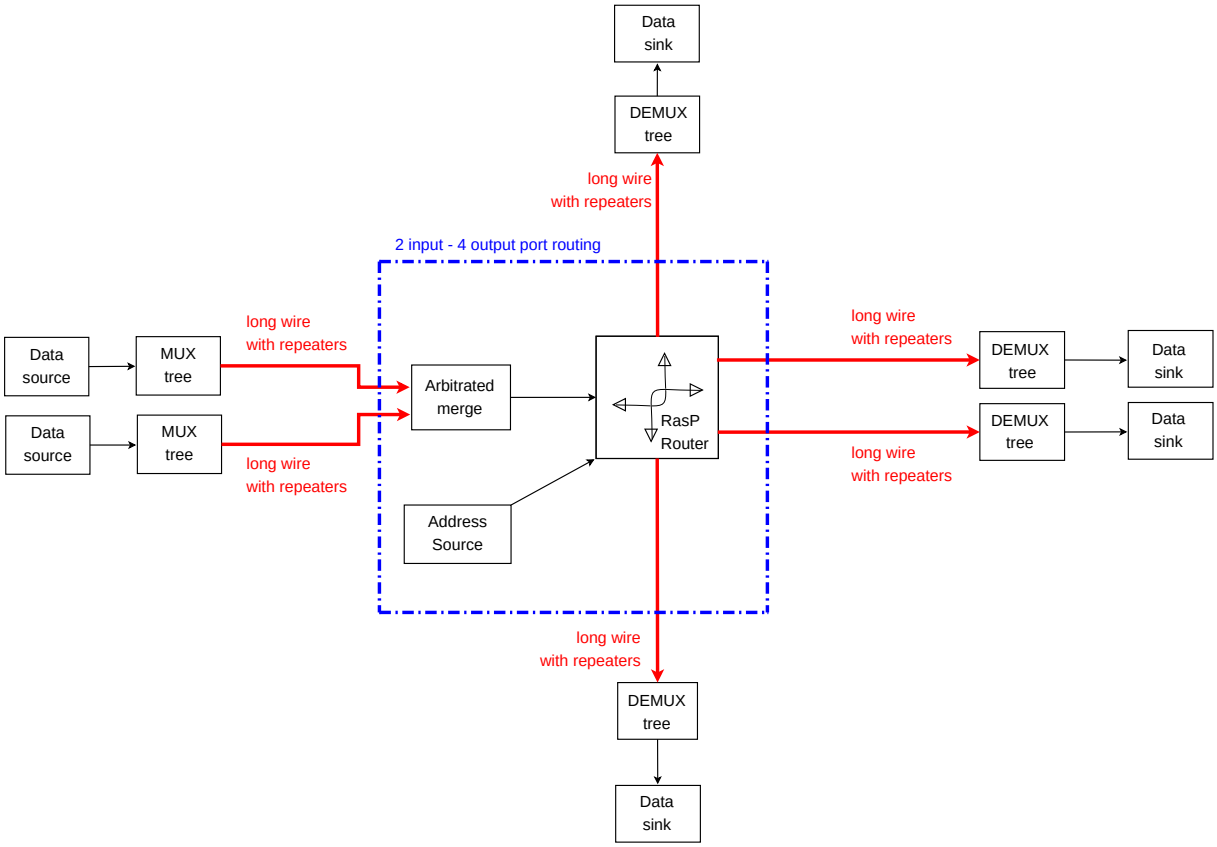


Figure 7.2: Overview of RasP test system components

a selected destination. This procedure can be thought of as applying a `fork` function [56] to the data path.

Arbiters are able to take in two streams of data, potentially arriving simultaneously, and select at most one to be transmitted at that point in time on its single output channel. In this way, arbiters ensure data-safe transmission, by preventing data collisions on merging channels. This can be thought of as implementing the complimentary `merge` function to the router element's `fork`. The reader may be interested in Molnar's elegant solution to a two-phase arbiter implementation, and novel uses for such arbiters [44].

For all elements, unserviced data is easily queued by the simple act of not resetting (acknowledging) the line after data is placed on it.

If an element is ever busy when data is presented to its input interface, the data can be easily queued for processing at a later time. If we simply do not reset the input interface (i.e., do not generate an acknowledge signal), the transmission protocol states that data must remain there arbitrarily. Thus, in these circumstances, the data will just sit on the wires until it is serviced at some later point. Critically, no data will be lost due to queueing — transmission and processing is always *data safe*.

Finally, since long wires will be involved in an NoC implementation, since it is a global venture, wires may suffer from the effects of the phenomena outlined in Chapter 4. For example, attenuation and crosstalk will be encountered, and signal-to-noise ratios will be degraded. This may culminate in erroneous values, or even phantom transmissions if steps are

not taken to restore logical values at appropriate stages throughout their flight. Traditionally these issues can be overcome by the use of a repeater element [65, pp.221–226]. With this in mind, I have designed two forms of RasP repeater element, depending on the exact needs of a particular repeater location. Repeaters are also often able to decrease the end-to-end delay of a link (see Section 6.2.2) since they 'shield' downstream capacitances from any line driving logic, but in RasP they are more valuable for their level-restoring ability.

I will now introduce the implementation of the two building-blocks of RasP, starting with the the router, and then the arbitrated merge element.

7.3.2 RasP router

Key to the RasP system is a router element, shown as Figure 7.3. It takes destination address information and a single data input channel, and chooses one of four output channels to route that data to.

For simplicity, I illustrate operation with two address wires indicating one of four routes by their states and generating an enable signal for the relevant routing direction. In a larger system, the first transmitted data word could be used to indicate one of 256 possible routes; or we could convert the two address wires to a fully-fledged point-to-point link, serially transferring a long address. I show the schematic for the 'north' output channel. The logic is repeated for each of the four routable directions, with a central route controller shown in dashed lines. Our simplified address wires are steered with AND gates, from the same direction enable signals since, unlike the data wires, they do not use the pulse-based protocol. In this example, where we have only four possible destinations, the address does not actually need to be routed to the following stage, but I have included the circuitry in the diagram, to indicate how this may be accomplished for larger systems. It should be clear from the schematic that the addressing logic is 'plug-in', with respect to the routing circuitry, and this is a key aspect of the flexibility of my system: regardless of the particular physical-layer protocol being used by a system, RasP can trivially adapt to it.

Performance evaluations are sufficiently accurate using our simple addressing scheme, so a more complex one is not necessary. This is mainly because address decoding need only happen once per data word, and so any overhead would be amortised over several data bits, making almost all performance penalties negligible compared to the cycle time of the rest of the router. Further, for the example shown, address decoding is very much off the critical path of the router element, and so most alternative designs would, likewise, have little or no impact on performance. For these reasons, the author is confident that the evaluation results given in this, and the following sections, are accurate.

In our example 'north' channel, we can see that the `enable_north` channel is logically ANDed via transistor *n1* in an n-type with two other signals, before allowing the passage of data. The bottom transistor, *n2*, is active whenever the output channel (both `north_0` and `north_1`) is empty of a data symbol. Thus, when the output channel is safely enabled (addressed and empty), transmission of a data symbol relies solely on the activation of transistor *n0*, which occurs as soon as a valid data symbol appears on the `data_in` inputs. Presently, the output enabling inverter is caused to go to high, and the two output drivers, *p1*, and its mirror on the `north_1` wire are enabled. The arrangement of the enable and data signals follows that of a best practice tristate design (as shown by Weste and Harris [65, p.103]), to provide the lowest output switching latency. The driver corresponding to the input data

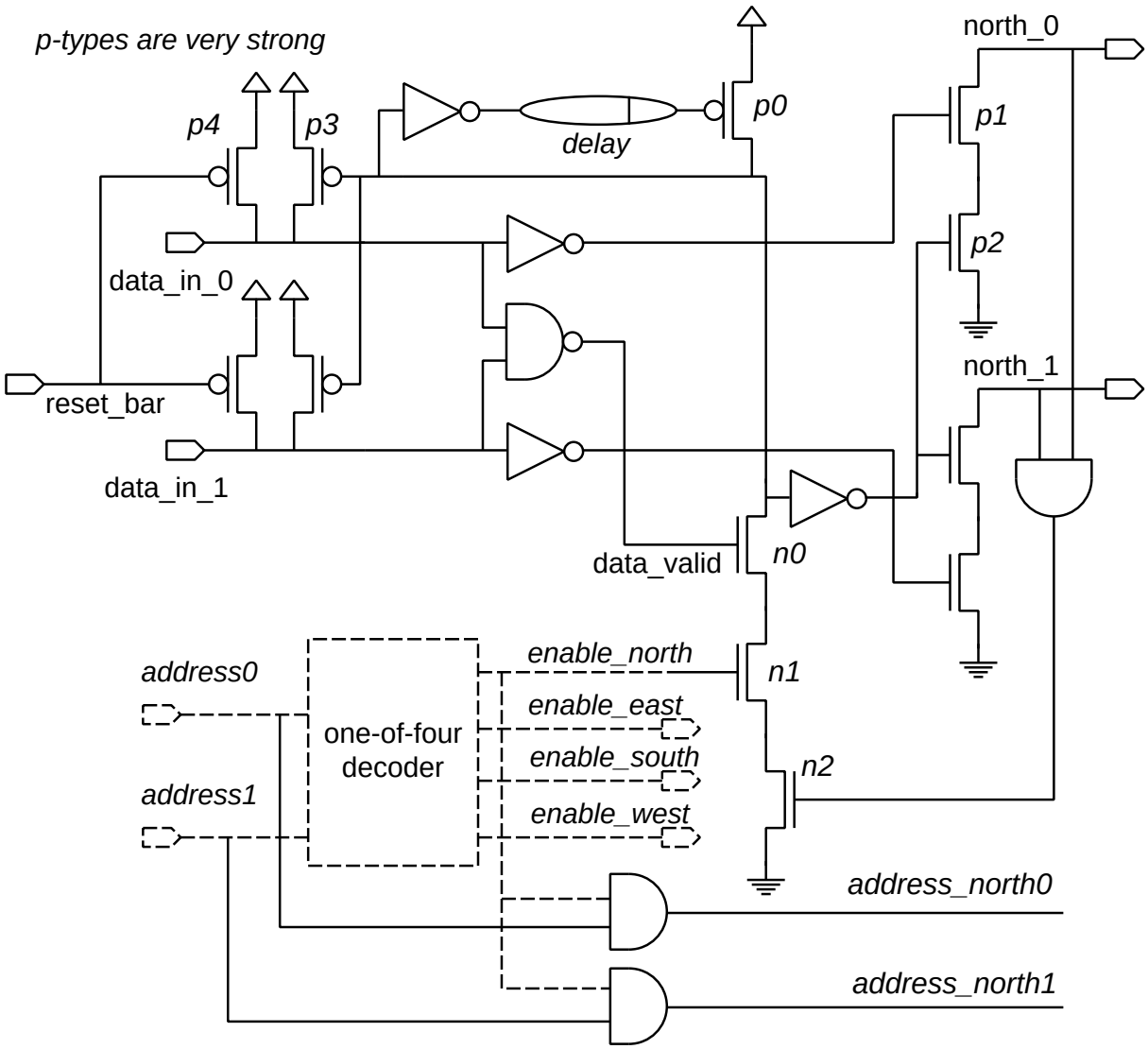


Figure 7.3: RasP router

symbol wire being activated (driven low) becomes the one which copies the input data symbol on the outputs. Thus, the router stage has successfully propagated the forward-going data symbol, and now only needs to complete the acknowledge stage. This is completed, as is the case for the stateless GasP repeater of Section 5.11.1, by the inclusion of a self-resetting pull-up stage on the transistor stack (consisting of transistor *p0*), which drives both the output stage to the high-impedance idle state, and the input stage to the all high idle state.

All of this functionality is contained in a very simple design, comprising only an inverter and an n-type transistor on the critical forward data path. Needless to say, this gives a very good latency characteristic, with data propagation taking merely around 310ps when attached to a system, which corresponds to under five fan-out-four delays in our technology.

The router has a critical path in the form of the stack of n-type transistors, comprising the `data_valid`, `enable` and `output_clear` transistors. This stack forms the discharge path of the output-enabling inverter and so no output may be driven without a transition here. Good

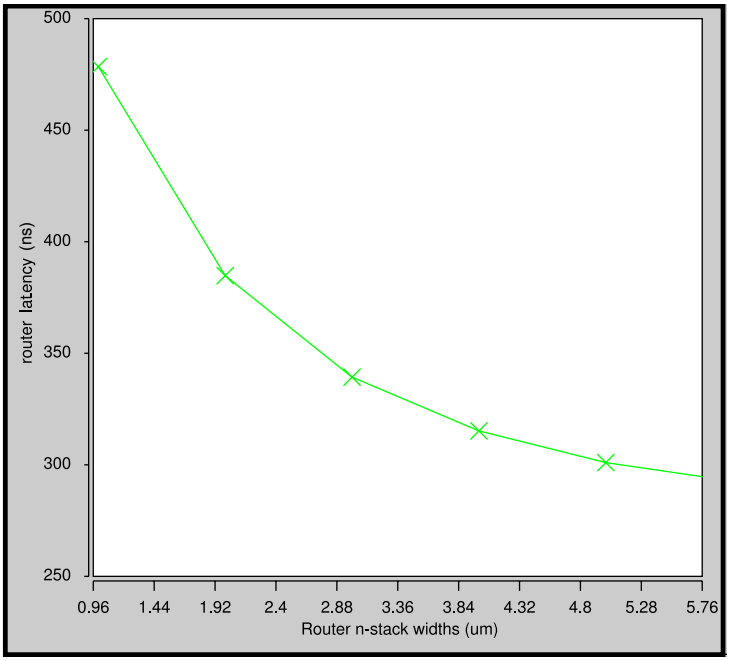


Figure 7.4: Reduction in RasP router latency with increasing n-Stack transistor widths

discharge performance depends on a low resistance path to ground, and this implies that the stack transistors should be wider than nominal. To illustrate the effect varying the width of these transistors has in performance, I present Figure 7.4, where the width of each n-type is plotted against the overall router latency. For example, we see that doubling the width from 1.28μm to 2.54μm reduces the router delay from 412ps to 310ps, a reduction of around 25%.

These figures come from a router attached to a long output wire, measuring an input 80%→20% of `vdd` transition, in-situ in the RasP system, to the same values on the output wires.

To perform unit testing on the router, its output was connected to a standard load for the 0.18μm technology (0.004pF). The input was connected to a linear ramp voltage, with the technology nominal slew rate of 0.08ns. When in this configuration, the router's performance is much improved over its normal configuration: propagation of 50% `vdd` input to 50% `vdd` output takes 290ps. This equates to 4.25×`FO4` delays in our technology. The increase in performance compared to the in-system case is easily related to the reduction in output loading, and we see the overall efficiency of implementation is high.

7.3.3 RasP arbitrated merge element

I call this element an 'arbitrated merge element' (from hereon simply 'arbiter') because it performs both functions (which are traditionally separated) simultaneously. Merging two RasP signals is simply a matter of connecting the two paths together, and ensuring that no driver conflicts occur. Since the latter is essentially the job description of an arbiter, MUXing comes for free. Our arbiter design uses pass transistor outputs, in the form of transmission gates, to ensure mutual-exclusivity when driving an output.

The input side of the arbiter element is much the same as for all RasP elements, with reset

circuitry and validity detection. The validity detection occurs in parallel on both input channels, and is funnelled into a bit counter and a MUTEX request input simultaneously. Before I go on to explain the operation of these first note that, unusually, no acknowledgements are generated in the arbiter. This is because it uses transmission gates, and the acknowledgement from a succeeding stage will flow through to a preceding one. Since the arbiter's data lines are not latched, to ensure signal integrity following these transmission gates, I recommend the immediate use of a GasP-style repeater (Figure 5.15), to boost line levels. Staticisers are added to the inputs to ensure that input line data values are not lost when there is a long wait for arbitration. Further, the active transistors of the staticising inverters also help to restore acknowledgement signals, after passing though the transmission gates.

The heart of the arbiter (shown as Figure 7.5) is a two-way mutual-exclusion (MUTEX) element, which takes in two request signals, `req0` and `req1` (corresponding to a valid data symbol on input `a` or `b` respectively), and grants at most one of `gnt0` and `gnt1` (corresponding to enabling the pass-through of input `a` or `b` respectively). If both requests are asserted simultaneously, one channel will be chosen at random to be granted and, when the request is de-asserted, the other grant will be enabled. This property ensures a degree of fairness: once one input channel has had a packet transmitted, the other one will get to transmit, if it is waiting.

An interesting feature of my arbiter is the operation of the transmission gates: they are driven asymmetrically. The inverted dual-rail protocol uses low pulses in the forward direction, and high ones in the backward. This places the n-type pass transistors on the critical forward path, and the p-types on the equally-critical backward one. Hence, we activate the n-type pass transistors immediately after a grant signal is produced, to allow the fastest possible data propagation.

Activation of the p-types is more interesting: since they are not critical on the forward path, we can afford to delay their activation after receiving data. However, they must remain on for a period after the backward-going acknowledge signal is detected, to ensure it has time to charge the input line fully. This involves a period of time where the MUTEX's request line is reset since all wires have gone high locally, even if the signal has not yet propagated fully to the preceding stage. Ordinarily, the `data_valid` signal would then be de-asserted; removing the MUTEX grant; and de-activating the transmission line. Instead, I insert a delay to keep the grant activated for a time sufficient to charge the input line. This is easily implemented if one delays p-type activation for an equal time after valid, forward-going data is detected on an input. Since I have already shown that the p-types are not on the forward critical path, this additional delay in activation does not affect the performance of the arbiter.

Output enabling is dependent on one additional factor: the current state of the output. Since the p-types are the last transistors to switch, they may still be on when the following input data symbol arrives at an arbiter input. Therefore, to prevent successive symbols from crashing into each other and potentially merging, enabling of the an output is conditional on its p-types being inactive (a high input signal), and this is the role of the AND gates on the `gnt` signal paths.

Finally, in this description, I will explain how the arbiter arbitrates on a per-word basis, in order to reduce the arbitration overhead. In the example presented here, we use 8-bit words to fit in with the implementation of the point-to-point link shown in Chapter 5. After arbitration on the first bit of a word, the output path remains enabled until the end of the word. This is accomplished by a counter that tracks the number of bits transmitted since arbitration began,

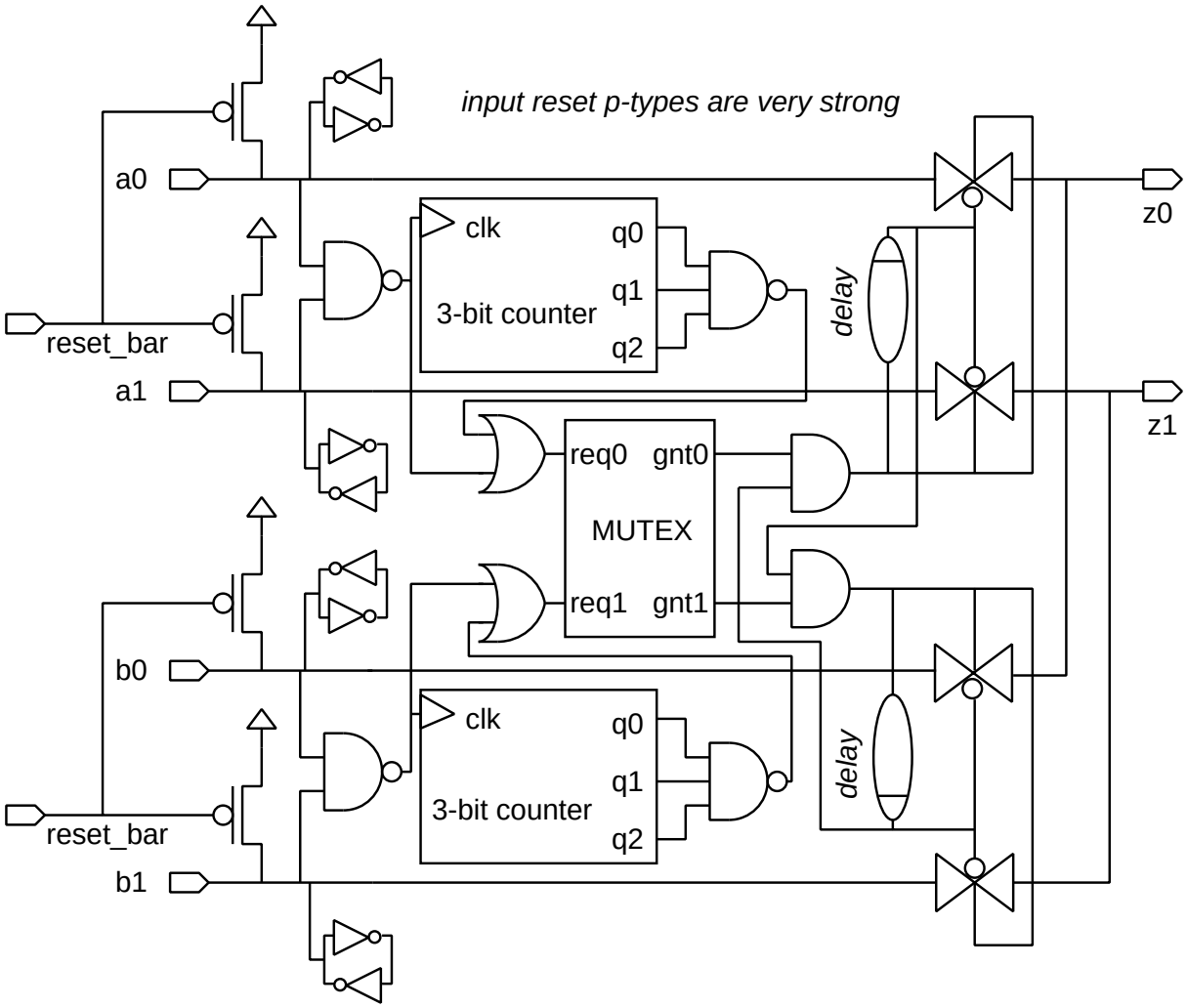


Figure 7.5: RasP arbitrated merge element

and only releases the MUTEX request after all eight bits of a data word have been transmitted. The counter need not be glitch-free, since the hysteresis inherent in the MUTEX smooths out such problems, and leads to error-free operation. Using a counter increases the amount of logic, but removes the need for another global wire signalling 'end-of-packet'. Per-word consideration also amortises the time cost of arbitration over a whole word, making it less critical than might otherwise be the case.

For wider data paths, where we aggregate multiple point-to-point links, the counter should not be clocked until all data wires are showing `valid_data` signals. This is easily implemented by clocking using a tree of C-elements: stateful gates whose output only changes once *all* inputs have changed in the same direction. These are further described by Sparsø and Furber in [56, p.21–23], where an example implementation is also given.

7.4 Summary

In this chapter, I have introduced the problem of creating a large scale multipoint-to-multipoint interconnect. To address this issue, I have outlined how a Network-on-Chip (NoC) can provide the desired functionality and performance, and so is ideal for use in these circumstances.

To this end, I have demonstrated the *Chain* interconnect system, a NoC implementation and shown that it is suitable for the task. However, I have also shown that it suffers from the problem of inefficiency at the global metal level (it needs to use six wires to transfer just a pair of data bits).

Realising how the point-to-point link designed in Chapter 5 could help to ameliorate this problem, I have produced a new NoC, *RasP*. To enable its construction, I have demonstrated designs for a router and an arbitrated merge element for the RasP system. Both operate on the pulse-based, dual-rail protocol of the point-to-point interconnect. Therefore, a RasP implementation appears similar to the Chain structure, but decreases the wire count to two: *an area reduction of two-thirds*.

8 Evaluation of RasP

Every man has a right to his opinion,
but no man has a right to be wrong in his facts

Attributed to Bernard M. Baruch

8.1 Introduction

RasP possesses all of the advantages outlined in Section 7.3 it is an area-efficient NoC, with cutting-edge performance. RasP is modular, scalable and area and power efficient. It offers standard interfaces to logic blocks, appearing as a simple FIFO structure. This makes it a very easy, drop-in replacement for other NoCs. I believe that RasP offers substantial complexity and area savings over conventional interconnect structures, such as those featured in Chapter 3. We now evaluate its performance, and compare it to Chain.

There are many factors of interest when evaluating the performance of an NoC implementation. Generally, they centre around factors relating to the data rates available, but designers are increasingly giving priority to those involving implementation details, for example size and ease of use. In this way, the priorities attached to the different metrics may vary, depending on the particular constraints of a design. Therefore, I make no claim to the exact ranking of them, but outline the four most important below.

- Throughput: both at the link and end-to-end level
- Latency: end-to-end and node level
- Energy consumption: generally in the form of pJ/bit/mm
- Area: requirements of global metal, routing nodes and a full system; in μm^2 of die space used

In this chapter, I will address each, and show how the performance of RasP measures up. For fairness of comparison, I present not only my own results, but also those from a reference implementation of the *Chain* interconnect system [3]. Chain is the system most similar to ours that is available in the literature and was designed by Bainbridge et al. at the University of Manchester. Like RasP, Chain is intended as a NoC design to easily connect together multiple IP blocks on a single die. I introduced it in Section 7.1. Let me also point out that, to produce a fair comparison, I re-implemented Chain in the CMOS process I use for RasP. This was done to bring the technology values up-to-date from those published in the Chain papers, and give a fair comparison with my system.

8.2 Evaluation methodology

System testing

The results in the following sections are all produced from hspice simulations. Following verification of the correct operation of all the circuitry, a spice test harness was produced to simulate the inputs and outputs of the systems under test. Rather than use artificial inputs, we evaluate the performance of each RasP system block in-situ, as part of a full system implementation. This allows us to more accurately model the transition characteristics of the system and, in particular, the performance of its drivers; and therefore to provide realistic implementation performance figures. The configuration of the RasP system can be seen as Figure 8.1, and this setup is used for all the following evaluation.

Inside the blue box in Figure 8.1, I show how the arbitrated merge element and router elements may be combined to produce a two-input, four-output router. This forms a primitive element used in many NoC implementations. Further, the node can trivially be extended into an n-input, 4-output router ($\forall n \geq 2, n \bmod 2 = 0$) by creating a tree of $n-1$ MUX nodes.

Throughout the results, simple data patterns were used, in most part because of the data-independence of the performance of RasP's point-to-point link (this was described in Section 6.3.1). For the arbiter evaluation, both non-overlapping/non-conflicting and co-incident inputs were considered, with functional correctness for coinciding inputs proved separately. The only observed effect of these worst-case inputs is to increase delay whilst metastability is resolved [36]. Given the very small probability of very coincident inputs occurring, amortisation over many data words makes its impact negligible. The RasP system figures are for designs with fully buffered inputs and outputs to logic blocks.

Unit testing

In addition to the full system evaluation, some performance figures for individual RasP components have already been given in Chapter 7. These data values come from test as stand-alone units. To perform this unit testing, RasP elements were isolated and configured as follows:

- Their outputs were connected to a standard load of 0.004pF[†], with an initialisation to the appropriate idle state (most frequently the `11` state);
- Inverted dual-rail inputs were connected with one input to `vdd`, and the other a voltage source, ramping from `vdd` to `gnd` in 0.08ns[†].

Chain comparison

For the comparison Chain system, optimal logic buffering and wire repeating was both calculated before and measured empirically after implementation. Results for overall system performances therefore come from such optimally-buffered systems.

[†]These values are not arbitrary; rather they are taken from the nominal values used in our technology for determining the performance of standard cells. Thus, it is prudent to use the same evaluation metric here. In fact, in the data book, the input values are for 80%→20% of `vdd` transitions, and so our evaluation is, perhaps, even a little pessimistic in comparison.

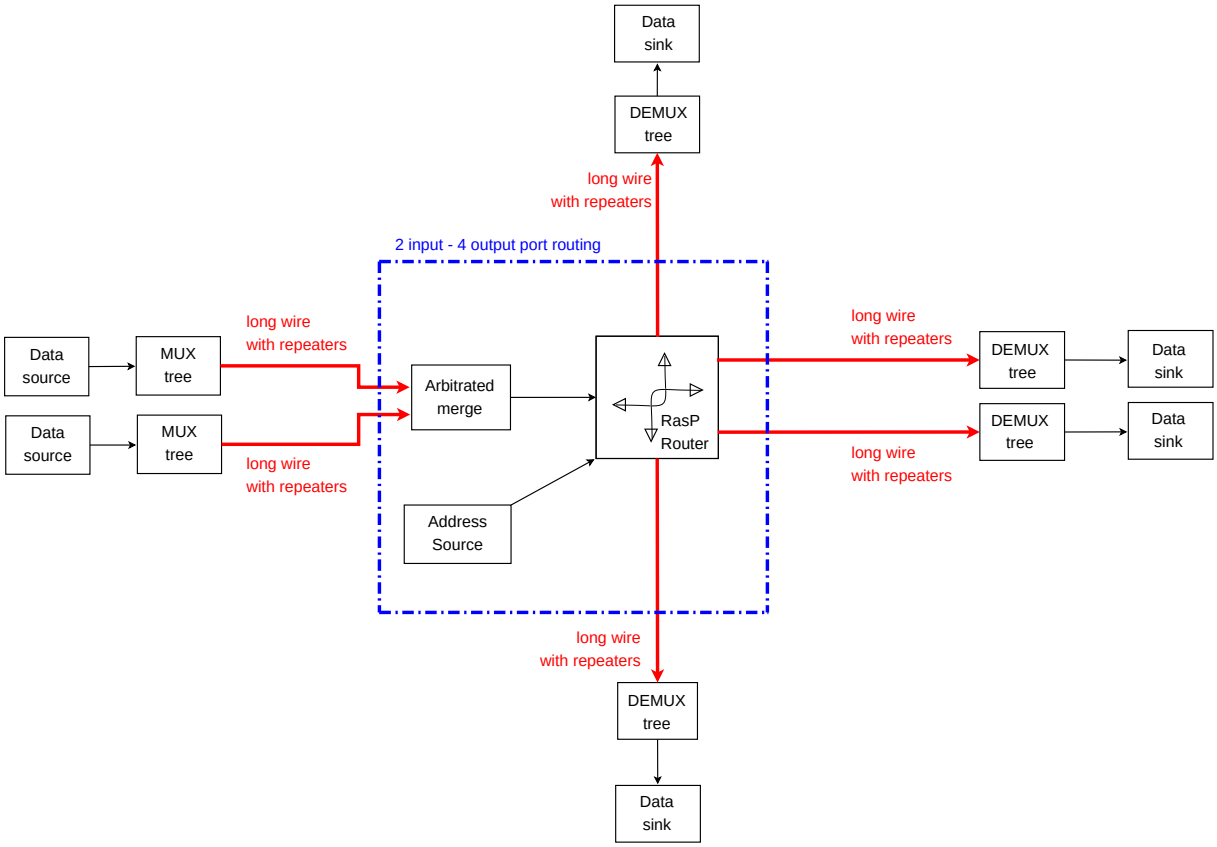


Figure 8.1: RasP test system setup

8.3 RasP base link

Since RasP uses the point-to-point link of Chapter 5 as its base case, the wire performance figures for RasP are the same as seen previously. Therefore, they are not reproduced fully in this chapter, though it is instructive to re-read that data and compare it with the performance of the RasP system, as observed end-to-end. Similarly to its base link, RasP's throughput is also improved by the insertion of stateful repeaters.

8.4 RasP performance evaluation

I now give the results for the performance of the RasP test system. As for the base link, we are concerned about the latency, throughput, area requirements and energy consumption of the system. Therefore, those are the metrics we will see results from.

8.4.1 Latencies

I characterised the logic latencies of the various RasP components, and these are shown in Table 8.1. The arbiter element has variable latency, dependent on how co-incident in time data arrives at its two input ports; the closer together, the longer its latency, since arbitration time is inversely proportional to the proximity of data arrival times. Therefore, I show arbiter latencies

Table 8.1: Logic latencies of the various RasP components (10mm wire)

Component	Latency
MUX tree	917ps
DEMUX tree	1500ps
Router	384ps
Arbiter (single data input asserted)	653ps
Arbiter (co-incident data inputs)	694ps
End-to-end handshake latency (`new data in`→`data_out_valid`)	24.3ns

for a data stream with no conflicting arbitration requests, and for one with co-incident data. We see the penalty for simultaneous arrival is 41ps.

The results do vary with wire length, since the wire logic inputs are presented with different rates of input slew for different lengths, but the variance observed was negligible compared to the magnitude of the latencies, and so is fixed with wire length to a first approximation, and I see no need to present additional data here.

Now that we have seen the latency of the individual components, we are in a position to see how they effect the end-to-end latency of a data word (which is one byte), when connected into a full RasP system. The figure for a 10mm total length is given at the bottom of Table 8.1 as 24.3ns for the first word. This is the time between `new_data_in` being asserted by the input environment, and `data_out_valid` being displayed to the output environment.

Figure 8.2 shows how this value changes as we not only range from one to ten millimetres, but also as we insert repeaters. We see the expected trend: longer wires take a longer time; and repeater insertion always increases latency, but with a lessened impact at longer lengths. The overall shape is very much like that for the repeated base link (Figure 6.36), with the minor alteration that one extra repeater is needed for correct operation with 10mm wires. Both figures show that, after the initial latency penalty is taken, latencies increase by only 40ps/mm when five repeaters are used. This compares very favourably to the known wire propagation delay of 35ps/mm, and shows that we can get within 15% of optimal performance.

8.4.2 Throughput

We will now see how RasP behaves in terms of throughput. First, in Table 8.2, I give the bit cycle times of the components, when presented with their first data bit of the day, and all other stages are empty. Very much like pipelining a conventional logic block, we wish all the times to be very well matched, so that no one component is obviously the critical path for the RasP design. Also, we understand that the system's steady-state throughput will be determined by the speed of the slowest component, so we wish none to be overly slow.

Whereas logic latencies are fixed regardless of wire length, bit cycle times also take into account wire propagation delays and so alter with total wire length and number of wire repeaters inserted. In Table 8.2 I display the bit cycle times associated with the optimal number of repeaters inserted for one and five millimetre wire segment lengths.

We see that the arbiter element has the worst bit cycle time. This is because it is a more passive component than the others; so is unable to strongly drive signals; and in turn receives weaker ones. Thus, both its reception and transmission operations are liable to be slower than the more active blocks.

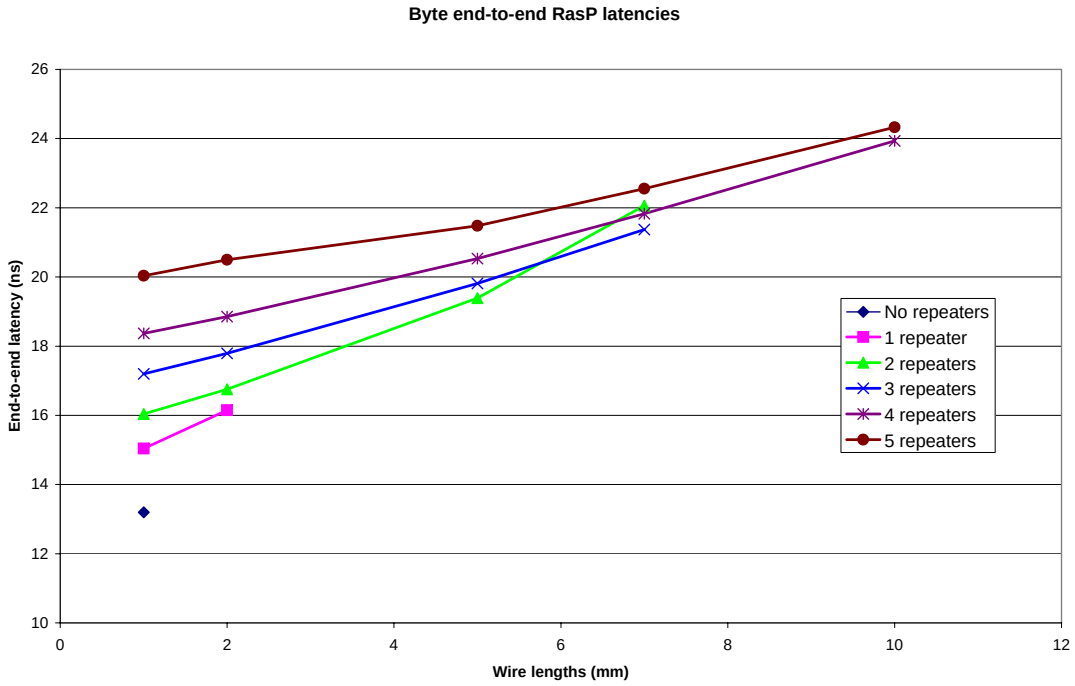


Figure 8.2: RasP end-to-end byte latencies

Table 8.2: Minimum bit cycle times of the various RasP components

Element	Cycle time (1mm wires)	Cycle time (5mm wires)
MUX tree	955ps	1033ps
DEMUX tree	1213ps	1388ps
Router	1249ps	1422ps
Arbiter (single data input asserted)	1311ps	1523ps
Arbiter (co-incident data inputs)	1352ps	1564ps

Now that we have identified the critical component for performance as the arbiter, we can observe its cycle times for varying lengths of wire, to determine overall RasP performance. Figure 8.3 shows how throughput scales. We see that RasP is just as amenable to repeater insertion than the link it is based on (compare with Figure 6.37). For example, at 7mm, changing from a one repeater to a five repeater setup results in a speedup of 30%.

The maximum throughput at 1mm is 763Mbit/s, falling to 599Mbit/s at 10mm (both when optimally repeated). This represents a 16% drop in throughput over a ten times increase in distance, which is a slow decline, and so indicates a scalable interconnect.

8.4.3 Energy

I include a table (8.3) of the power consumption of the various principle RasP components. Data is collected from multiple word runs and then averaged out to give a per-bit energy value.

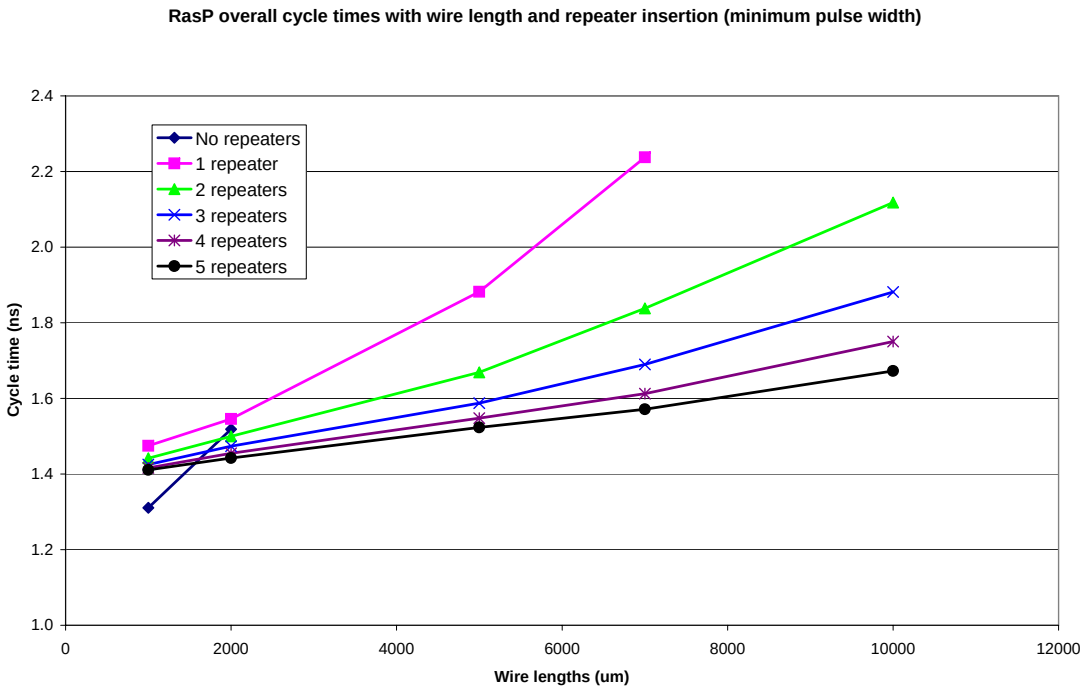


Figure 8.3: RasP test system cycle times, with wire length and repeaters inserted per link

Table 8.3: Energy consumption for RasP elements for a total link length of 10mm, with three repeaters inserted

Component	Energy consumption per bit
MUX tree	0.297pJ
DEMUX tree	3.37pJ
Arbiter	0.328pJ
Router	1.03pJ
Repeaters	5.67pJ

For the repeaters, multiple repeaters were measured and an average taken to increase the accuracy of measurement. `vdd` is 1.8 Volts in the system.

We see that, whilst the DEMUX tree may be area heavy, it still consumes significantly less energy than the repeater, which is what one would hope from an efficient implementation, since the capacitance of long wires should require the largest share of system power.

The breakdown of energy consumption is best illustrated as a pie chart, and we can see this as Figure 8.4. As expected, wire driving, repeating and buffering takes around three quarters of the power consumption, indicating that logic optimisation is not the critical factor in the design's energy efficiency.

The full graph of the repeater power versus wire length can be seen as Figure 8.5. Once again, the non-wire-driving components increase their energy consumption with wire lengths due to decreased rates of input slew and the associated extra dynamic power dissipation.

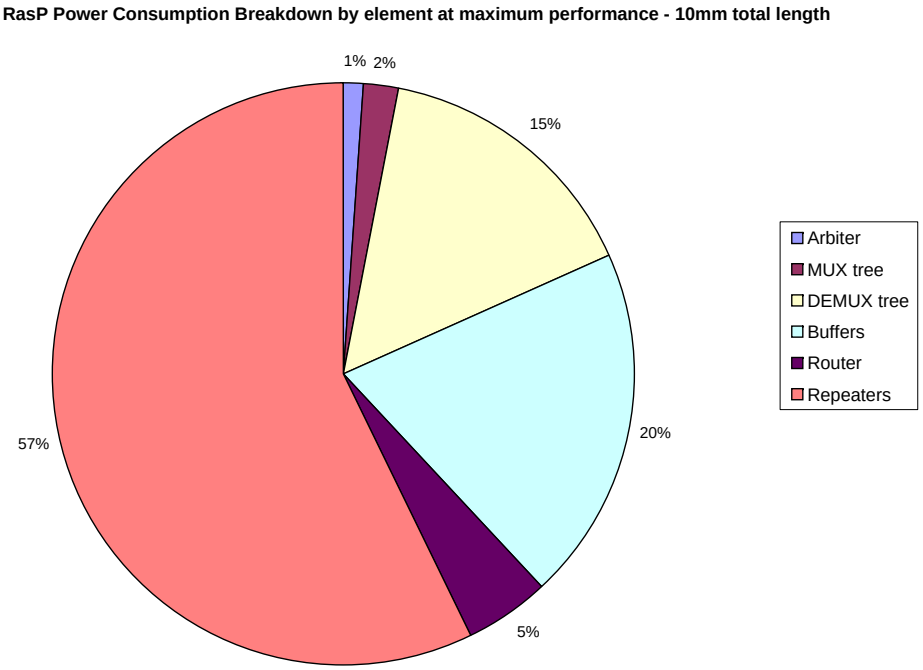


Figure 8.4: RasP energy consumption by element breakdown (5000μm wire)

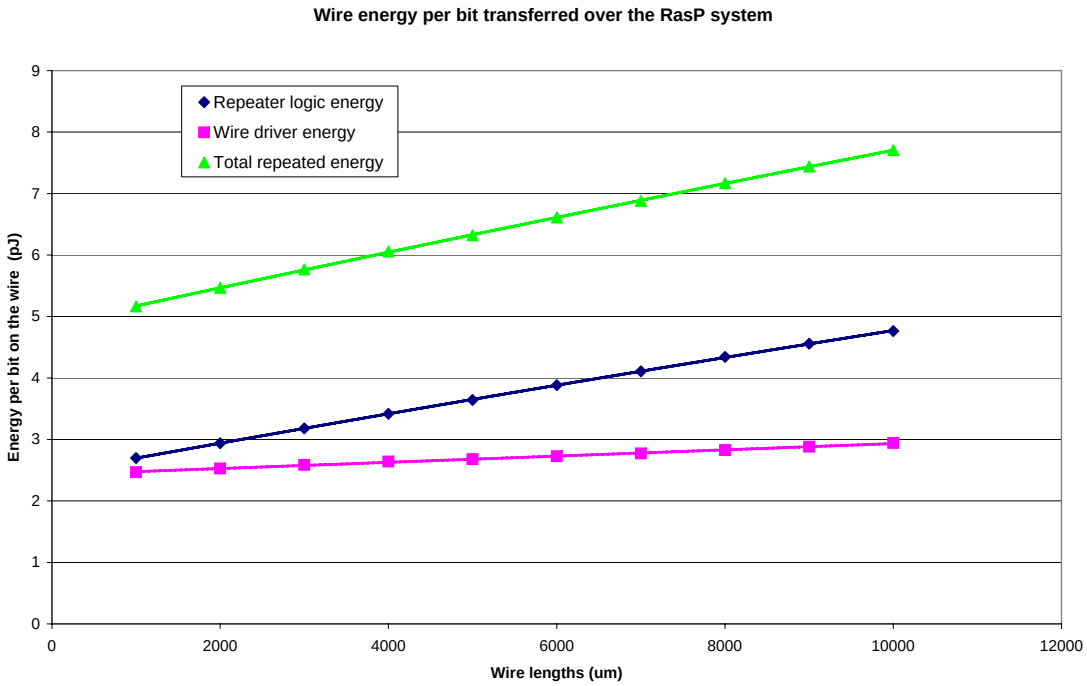


Figure 8.5: RasP test system wire energy with length (3 repeaters inserted, minimum pulse width)

Table 8.4: RasP test system area breakdown

Element	Area
Stateful RasP repeaters	2712μm^2
RasP router	2533μm^2
RasP arbiter	841μm^2
Logic buffering	288μm^2
Wire drivers	160μm^2
DEMUX tree	6044μm^2
MUX tree	855μm^2

We see a counter-intuitive result: for three repeaters with a minimum pulse width, the energy required in repeater logic increases roughly linearly (by approximately 0.25pJ/mm)with length, whilst the energy needed to drive the wire capacitance increases much more slowly. In fact, the increase tails of at longer wire lengths (although this is not easy to see from the plot).

Why is this? Well, the logic takes more energy than the wires since stateful repeaters have a lot more logic than conventional ones, and so gives a less efficient implementation. Their energy needs increase with length due, once again, to the detrimental effect of reduced input slew rate on dynamic power consumption. For drivers, the slowing of the increase at long lengths is again due to the phenomenon of not fully charging the wire at long lengths for a given pulse width (recall the explanation from §6.13.5). This could lead to signal integrity problems, and so for these lengths a longer pulse duration could be necessary in practice.

Overall, the total RasP system power consumption follows the shape of the wire driver total energy consumption (green triangle) line from the graph, but totals for the entire RasP setup range from 11.3–17.5pJ/bit for the wire range 1–10mm. This is twice the variation than can be explained by wire energy alone from the graph, and so approximately 3pJ/bit is additionally expended by RasP logic blocks when wire lengths are increased from 1–3mm.

8.4.4 Area

Like the base link, area estimates for RasP elements were produced by summing the contributions from standard cell footprints, where available, and an estimate of the size of any custom logic used.

The breakdown by element is shown in Table 8.4. Figure 8.6 presents the table information as a pie chart of area consumed, were a single instance of each element to comprise the system.

The test RasP system logic uses a total of 29,548μm^2 (of which 25,886μm^2 is taken up by the six MUX and DEMUX trees, leaving the core system taking only 3,662μm^2), plus repeater area dependent on the number inserted of 0–13,650μm^2.

We see that roughly half the area footprint is taken up by the DEMUX tree, with a fifth going in wire repeaters, and another fifth in the router element.

We expect to have to pay in area for wire repeating and routing, but the DEMUX tree value is slightly surprising, and is due to the number of state-holding elements it contains. However, bearing in mind that only one DEMUX tree is needed per end-to-end connection, and there may be a number of intermediate routing nodes, its area cost can be amortised

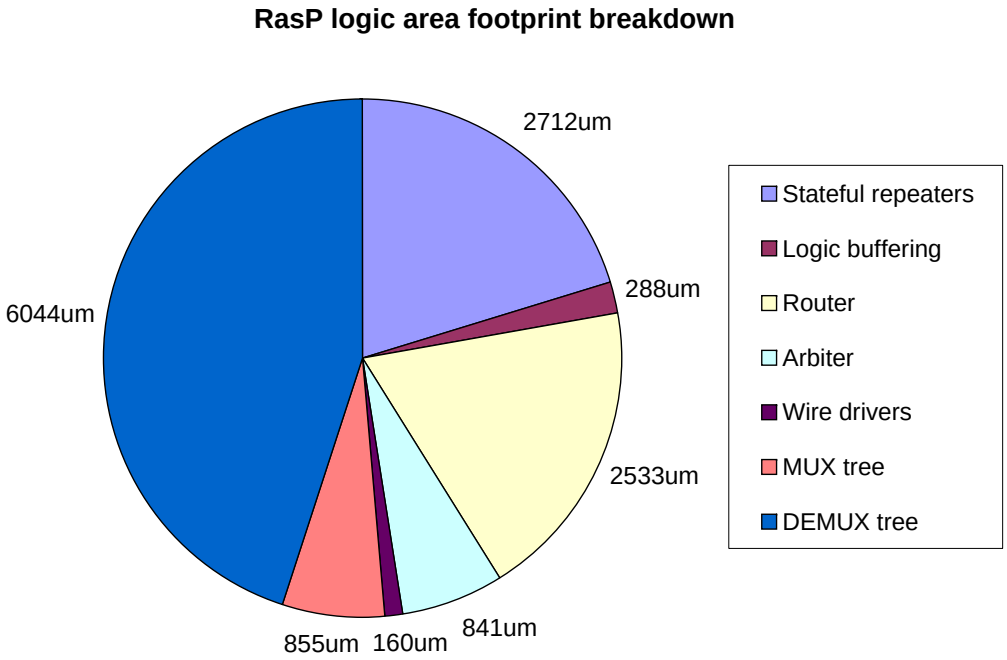


Figure 8.6: RasP test system area breakdown by element, in μm^2

somewhat. In the test system, however, we use four DEMUX trees and only one routing node, so its importance is artificially inflated in this chapter.

Finally, we compare the system's logic area footprint to that of the global wiring. For the RasP test system, wiring area ranges from 1,120μm^2 for a 1000μm long wire, to 11,120μm^2 for 10mm. Thus, a system with 10mm of interconnect has a total footprint of 40,748μm^2.

So, we see that the total area footprint for long wires is shared, three-quarters in the logic and only one-quarter in the wiring. Since our original assumptions were that logic is much more disposable than wiring, our goal has been met when we find that we have created a logic-heavy design, with a resulting decrease in global metal requirement.

8.5 Comparison to the base link

How does RasP's performance compare to the link upon which it is based? We expect to see a performance hit in terms of latency, since more components must be traversed, and their increased complexities is also likely to hit throughput.

Both of these assumptions are correct: at 10mm, RasP throughput is 599Mbit/s, compared with 926Mbit/s for a solo link, and latency is 16.1ns, compared with 13.0ns. At 1mm, these change to 763/1012Mbit/s and 13.2/8.2ns, for the RasP system and the base link, respectively. So, we see that the RasP system's performance is 30% lower than that of the base link, for both throughput and latency. Given the additional complexity of RasP over the base link, this is quite reasonable.

8.6 Comparison with the Chain interconnect system

At this stage, we have seen results from RasP, but are not quite sure of their relevance in comparison to other interconnect systems. Therefore, in the next few sections, we will see how our reference Chain system performs with an identical setup, and technology parameters. Then, at the end of this chapter, I will compare the figures, and the pros and cons of RasP and Chain.

8.6.1 Optimal buffer sizing for Chain

Before performance figures can be accurately generated, the sizing of Chain drivers must be determined. Given the custom nature of the Chain pipe latches, none of the standard repeater insertion equations we saw in Section 6.2.2 apply. Therefore, a repeater sizing and placement sweep was performed to determine optimal performance.

Repeaters

To provide a fair comparison between the performance of the RasP and Chain systems, we need to ensure that both operate on a level playing field. For Chain, this entails providing full optimal repeater insertion along long wire segments. This reflects likely real-life practice, were the design to be implemented. To determine the optimal values, we first perform a calculation of the optimal driver widths and inter-repeater length.

Simulations show that Chain operates most effectively when wires are driven by a two inverter buffer, with an optimal output n-type transistor width of 13.5μm. The sweep results for 5mm wires are shown as Figure 8.7, with other lengths telling a very similar story. The shape is perfectly characteristic of a repeater insertion sweep, as characterised by Sutherland in his book on Logical Effort [32].

We see the expected curve of an initial rapidly reducing delay, followed by a levelling off period, and then a hint of a slow increase again. This is as well known [32]. We see that the optimal size from a delay point of view is about 13.5μm for the output width.

Like the RasP sweeps, the results were for a complete Chain implementation, and so logic inputs have an equivalently imperfect rise time and realistic behaviour.

Power consumption versus width is not illustrated here, but data was extracted and it scales linearly with transistor width (this is also in line with received theory). When taken into account to create the metric of *delay-energy product*, then smaller transistor sizes rapidly become more attractive: a 1μm width consumes just over half the power of a 2μm one (from extracted data), and yet the delay penalty is not nearly as bad as two-fold. So, in practice smaller driving transistors would be chosen. However, we choose to consider the optimal performance configuration for the evaluations that follow.

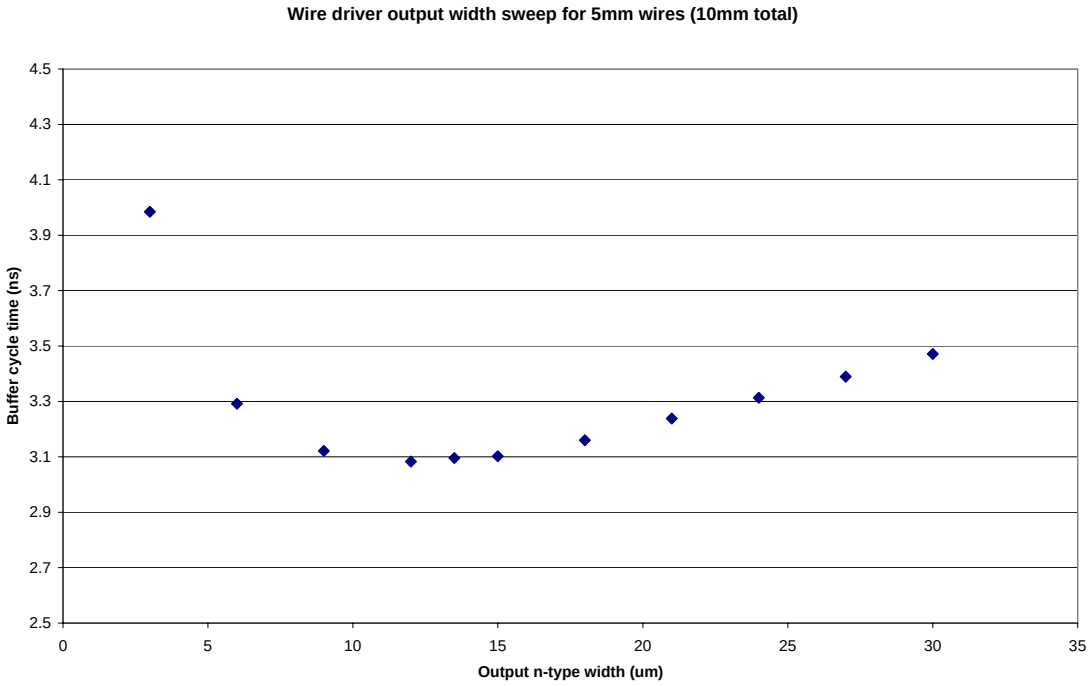


Figure 8.7: Chain wire driver input buffer sweep for a 5000μm wire

8.7 Chain system results

The setup for the Chain system is the same as that for the RasP implementation, to ensure a fair comparison. To recap, it involves the following: two data sources, each going into a parallel-to-Chain converter; two repeated long wire sections, feeding into a MUX, then an arbiter, and then a router. The routed data is buffered, fed onto a second long wire, and is finally converted back to parallel, before being consumed and acknowledgements generated.

In their literature [3], Bainbridge et al. omit the implementation details of how to convert a standard parallel data channel into the one-of-five encoding they use for Chain, and back again. Since it is only fair to compare systems with identical input and output interfaces, I implemented such a converter. I then consider Chain's 'base link' to be this parallel-to-one-of-five conversion, fed into a pipe latch, driving a long wire, followed by conversion back to parallel.

8.7.1 Latency

The latency of various Chain components contribute to the overall end-to-end latency, and the values are shown in Table 8.5. For Chain, the latencies are constant with wire length, except for the output buffers (where the increase in wire capacitance causes a decrease in output voltage slew) and the arbiter (where a reduced input slew causes the resolution time to increase). I show a range, at one, five and ten millimetre wire lengths, and the maximum performance figure for each. At the bottom, I show the word end-to-end latency, which increases as expected with wire length. We see that the word latency increases by 800ps for

Table 8.5: Chain element forward latencies

Element	Forward latency (1mm wires)	Forward latency (5mm wires)	Forward latency (10mm wires)
Parallel-to-Chain converter	1130ps	1130ps	1140ps
Chain-to-parallel converter	960ps	980ps	1010ps
Router	215ps	216ps	217ps
Arbiter	745ps	870ps	990ps
Multiplexer	408ps	408ps	410ps
Pipe latch	375ps	399ps	399ps
Output driver buffering	1520ps	1590ps	1680ps
Word end-to-end latency	18.2ns	22.2ns	26.2ns

every additional millimetre of wire length, which is substantially more than can be explained by the wire delay alone (at 35–40ps/mm, when optimally repeated).

8.7.2 Throughput

We now look at how quickly Chain is able to deliver data to its endpoints: its throughput. Like RasP, Chain's throughput depends on the maximum bit cycle time of its component elements. With Chain this was found to be the router element, as expected for the most complex element. Again, in steady-state operation, all other components' cycle times become synchronised to that of the router; therefore I present only the values for the router in this section.

Throughput with Chain is shown in Figure 8.8, and is affected in exactly the same way as RasP: decreasing quadratically with increasing wire length. However, the scale factor of this quadratic is reduced by the addition of repeater stages (pipe latches). As is the case for RasP, the addition of an infinite number of repeater stages would theoretically allow throughput to remain unaffected by length changes; whilst an unrepeated link rapidly becomes slow. Obviously, the exact number of stages in a real design will more likely be based on power and area trade-offs, rather than raw throughput figures.

The maximum throughput figures are 823Mbit/s over 1mm, decreasing to 752Mbit/s for 10mm.

8.7.3 Energy

We now perform the same energy consumption analysis as we did for RasP. The values for the energy consumed per bit (recalling that Chain transfers two bits per symbol), at maximum efficiency are displayed in Table 8.6 for transfers over 5mm wire sections.

We see that Chain consumes considerably more energy per bit transferred than RasP, even though 62% of energy is consumed with wire driving and pipe latches, which is percentage-wise a more efficient implementation than RasP. The main reason for Chain's larger value is its need to drive several wires to transmit a symbol; and every transition has an energy cost. Therefore, when compared to RasP, which requires only one transition per bit, Chain's energy consumption is much higher.

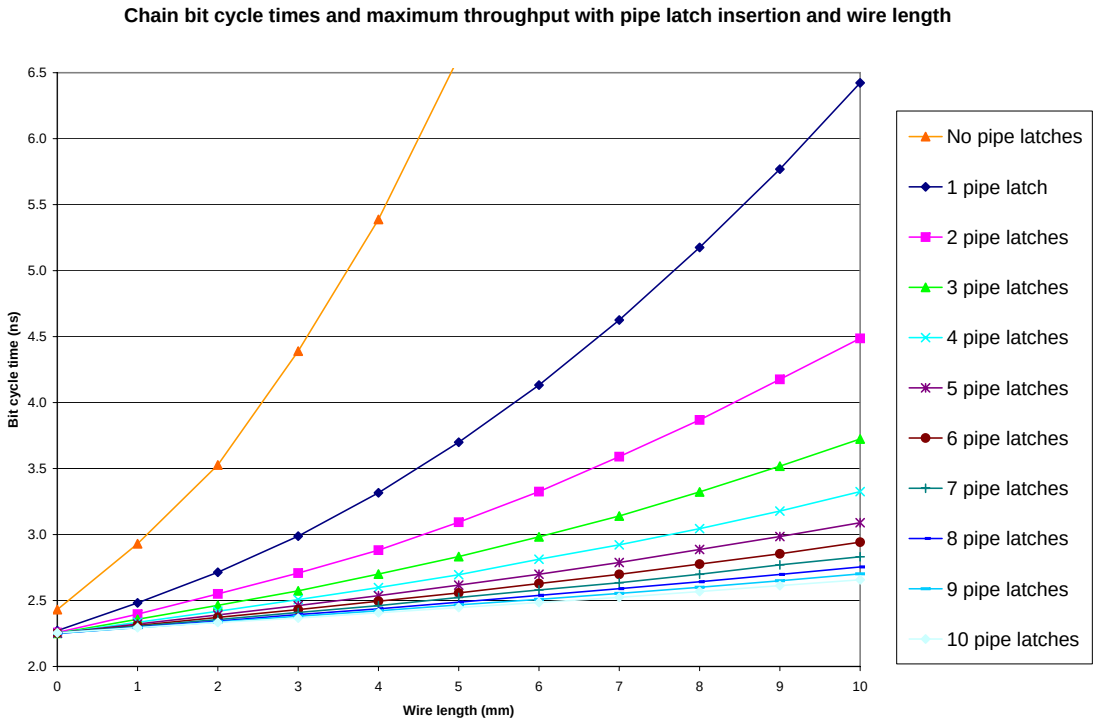


Figure 8.8: Bit cycle times and maximum throughput with repeater insertion length for the Chain system

Table 8.6: Chain element energy consumptions per bit with 5mm wires

Element	Energy consumption
Parallel-to-Chain converter	1.02pJ/bit
Chain-to-parallel converter	0.16pJ/bit
Non-pipe latch wire buffers	1.13pJ/bit
Router	0.17pJ/bit
Arbiter	0.26pJ/bit
Multiplexer	0.52pJ/bit
Pipe latches (one off)	1.32pJ/bit
Pipe latches (three off)	2.29pJ/bit
Pipe latches (five off)	3.42pJ/bit

Again, I display an alternative representation at 5mm with three pipe latches as a pie chart in Figure 8.9. To compare this with RasP, the total per-bit energy over the range 1–10mm, with the same number of latches, varies from 9.5–32pJ. This is a much larger variation than with RasP, although it starts off from a lower value. Therefore, we can say that Chain is most energy-efficient at short wire lengths, and RasP better for longer ones.

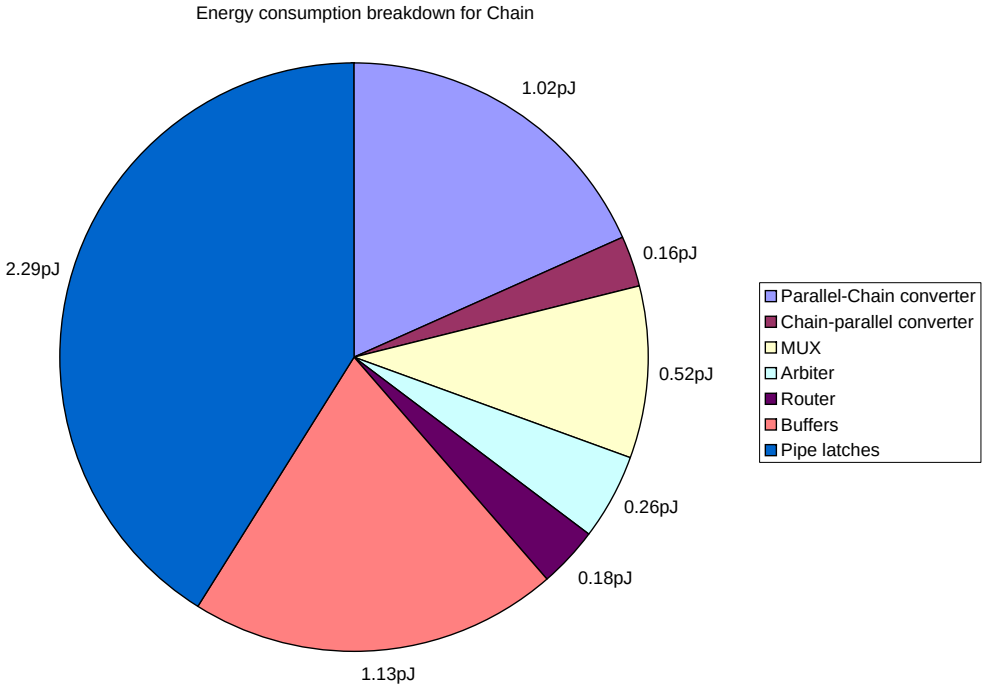


Figure 8.9: Chain element energy consumptions per bit with 5mm wires

Table 8.7: Complex Chain system area breakdown

Element	Area
Parallel-to-Chain converter	2013μm^2
Chain-to-parallel converter	1171μm^2
Non-pipe latch wire buffers	3240μm^2
Router	789μm^2
Arbiter	701μm^2
Multiplexer	259μm^2
Pipe latch (inc. buffer)	3444μm^2
Other	23μm^2

8.7.4 Area

The breakdown of the area-requirements for Chain elements may be seen as Table 8.7. Comparing the values with RasP's, we see that the opposite conversions are area-expensive — parallel-to-Chain conversion takes around 2,000μm^2 for Chain, yet the RasP MUX tree takes less than 1,000μm^2. Conversely, Chain-to-parallel takes only around 1,000μm^2, but RasP DEMUXing is the most area-expensive operation, at around 6,000μm^2. Also, Chain's router is around a third of the size of the RasP variant. However, the Chain Arbiter and MUX elements are slightly bigger.

Overall, then the total Chain test system area, *excluding wires*, comes to 12,180μm^2. So, with 5mm of wire added taking 16,800μm^2, the grand total of area becomes 28,980μm^2.

8.7.5 Comparison to original paper results

It is always good practice to compare a re-implementation of a system such as Chain with the originally published results, and I do this now.

In their later paper [4], Bainbridge, Plana and Furber state that the peak performance of a single Chain link on a 180nm process, and over one pipe latch stage, is 750Mbit/s over 2mm of wiring. To compare, the result I get is 566Mbit/s, although this rises rapidly to 858Mbit/s once two repeater stages are inserted over the same distance.

8.8 Comparison between RasP and Chain

So far in this chapter, I have presented results from the RasP interconnect system, based on the point-to-point link of Chapter 5 and also the Chain interconnect system as a reference implementation. In this section, I compare their relative performance.

Our discussion has been based on the four main performance metrics of throughput, latency, area footprint and energy requirements per bit. I therefore summarise these for the two systems in Table 8.8. We see that, for a total wire length of ten millimetres, the two systems are within 25% of each other for all metrics except energy. RasP has the edge on latency, area and energy-efficiency, with Chain being better for throughput. At shorter wire lengths, Chain's energy efficiency increases, and here the selection of the most energy-efficient becomes blurred.

In summary, both approaches produce viable NoC implementations, and performance against the metrics varies between the two designs, and with the wire lengths under consideration. Therefore, the choice of which to use in a system will ultimately depend on what relative priorities a designer assigns to the various parameters, but RasP has certainly been shown to be a viable alternative for an NoC implementation.

To conclude the evaluation between RasP and Chain, I now outline in bullet-point form the various pros and cons of the two systems, as observed by the author during development and evaluation. I first start with a discussion of RasP, and then go on to Chain.

Table 8.8: Test system comparison between RasP and Chain (total wire length 10mm)

Characteristic	RasP	Chain
End-to-end latency of first byte	24.3ns	26.2ns
Throughput over 10mm total length, optimally pipelined	599Mbit/s	752Mbit/s
Energy per bit transferred over 10mm total length	17.5pJ	32.2pJ
Two-input, four-output router logic area	3,682μm^2	1,772μm^2
Test system global metal area	11,200μm^2	33,600μm^2
Test system total logic area	29,548μm^2	12,180μm^2
Total test system area	40,748μm^2	45,780μm^2

RasP pros:

- Very low wire complexity, greatly simplifying routing;
- Uses only standard gates and conventional transistors;
- Clock domain crossing is free;
- Operates independently of data producer / consumer rates safely;
- Energy efficient.

RasP cons:

- Pulse length must be varied according to wire length or inefficiency results;
- Potential routing complexity in the MUX / DEMUX tree;
- Signal integrity needs to be analysed closely if minimum pulse widths and/or minimum spaced wires are used;
- Semi-analogue operation may be unfamiliar to a designer, and can be harder to verify.

Chain pros:

- Clock domain crossing is free;
- Operates independently of data producer / consumer rates safely;
- Is speed-independent — tolerates arbitrary wire delays without any ill effect or design consideration needed;
- One-of-five encoding is power-efficient;
- Digital transition paradigm is easy for a designer to understand.

Chain cons:

- Needs custom-designed C-elements, which are inherently slow;
- Requires six wires to transfer two bits, which has a high routing complexity;
- Uses up to an eight input gate, with associated stability and glitch issues.

All the above points are more complex manifestations of the properties of their respective base links. Thus, the trade-offs for each test system closely mirror those of the underlying units.

8.9 Summary

In the preceding two chapters, I have shown how the pulse-based point-to-point link of Chapter 5 may be scaled into an NoC with routing, multiplexing and arbitration facilities. We have seen that end-to-end performance is promising, and easily meets our target ASIC application requirements, even over distances as long as 10mm. To do this successfully, stateful repeaters must be deployed, and I have shown the various trade-offs (in particular that of throughput versus energy) associated with use. We have seen end-to-end throughputs of 763Mbit/s for the test RasP system with 1mm wire lengths, dropping slowly to 639Mbit/s for a 10mm wire. Latency scales near linearly over the same range from 13.2ns to 24.3ns, for an 8-bit data word.

To provide a reference comparison, I re-implemented the Chain interconnect system, and employed it to produce a second NoC, with an identical system layout. Chain's throughput just exceeded RasP's at 1mm, with 823Mbit/s, and dropped similarly at 10mm, to 752Mbit/s. However, Chain's end-to-end latency was observed to be substantially higher, at 18.2–26.2ns.

In other metrics, over 10mm RasP consumed just over half the energy per-bit than Chain, even taking into account that Chain transfers two bits per data cycle. This is a key result for RasP, since energy efficiency is a key factor in new designs.

The two systems also displayed values within 25% of each other for area-efficiency, with RasP being the smaller. However, this similarity in the total figures hides the more interesting comparison: global metal area. RasP uses just one third of the global metal of Chain, and so is likely to be much more easily routed, and to have better performing links, since additional wire spacing could be allocated. Therefore, in the "Transistors are free, wires are expensive." ideology of the modern VLSI design, RasP can be said to be *much* more area efficient than Chain.

9 Conclusions

Great things are done by a series of small things brought together

Vincent Van Gogh

Interconnect design is always difficult, and on-chip interconnect design is no different. The large number of factors simultaneously in play during development is greatly complicated by their high degrees of interdependency (for example, increasing throughput may also increase latency, and we may wish both to be optimised).

For many years, throughput and latency have been the major parameters to consider when creating a link, but with modern designs, power consumption and area-efficiency have become equally important. Additionally, the scale of integrated systems, and the deployment of multi-core circuits have led to the need for even more widespread and coordinated interconnection. The solution to this is clearly a move to Network-on-Chip fabrics, as outlined in this thesis (and in particular Section 3.4.2).

At the heart of every good NoC design is a high performance point-to-point link, and so this dissertation has shown how to produce such a link. Particular emphasis was given to reducing the global metal footprint of the interconnect (and global metal is our most important on-die routing resource). To this end, a serial transmission scheme was employed to improve area-efficiency.

The link operates unconventionally, using pulse-based transmission techniques, rather than the, more conventional, level-based logic style. The approach has been shown to present advantages such as compactness and the ability to use power-efficient low-swing operation over long wire lengths. Equally, though, it has posed challenges, since a non-standard technique with analogue voltages has proved more difficult to debug or to theoretically guarantee correct operation. The latter point is particularly noticeable from the need to calibrate the widths of transmitted pulses to the length of the wires, if fault-free transactions are to be assured.

With the use of novel *stateful repeaters*, we have shown how to pipeline transmissions over long wire lengths, maintaining throughput, despite the system's use of an acknowledgement-based protocol. Peak throughput has been shown to be 1.01Gbit/s over 1mm of wiring, and 926Mbit/s over 10mm of wiring, with a steady decrease between the two.

The point-to-point link thus compares favourably with conventional wiring schemes. Not only does it support a high data rate, but its ability to transfer both data and control signals, using a bi-directional signalling scheme, leads to an increase in area efficiency over native wiring. The evaluation in Section 6.15, based on Ho's work [22], compared the available bandwidth of the link to that theoretically available. The results were promising, particularly for long wire lengths: 91% efficiency for a single wire at 10mm, assuming a distributed RC

model. However, since two wires are used for the interconnect (only one is ever active), across the link area, this drops to 45% of the theoretical maximum — still a very good utilisation ratio.

To demonstrate that the point-to-point link was not a purely academic exercise, we have also seen how to scale it into a fully-fledged NoC design called *RasP*. RasP's performance is slightly reduced from the basic link, due to the additional overhead of the extra components needed to perform networking functions. The overall throughput is still high, however, at 763Mbit/s over 1mm of wiring, and 599Mbit/s over 10mm.

To provide a realistic benchmark for RasP, and to put it properly in context, a version of the Chain interconnect from the University of Manchester [3] was re-implemented in the same 180nm process technology as RasP. Chain is an NoC implementation, based on more conventional asynchronous logic structures. It provided the perfect comparison for RasP, due to its like-for-like component mapping and, in particular, its similar use of stateful elements to pipeline long links. Chain produced better throughput results, but was worse performing than RasP in the key metrics of latency, power and area.

This, pleasing comparison shows that the implementation of RasP was a realistic and effective one. Finally, with performances as stated, the author is pleased to say that both the point-to-point link and RasP have met the target ASIC application, set out in the introduction to this thesis.

9.1 Future work

There is always more research to be performed on the topic of on-chip interconnect, and the RasP implementation's future opportunities are no different.

Perhaps the biggest piece of work omitted from this dissertation is to consider the impact of changing the shape of the wires over which data is transmitted. Here, we talk only of minimally-sized, minimally-spaced wires, since our goal was to provide the smallest possible area footprint. Whilst, earlier on, I provided a table of how capacitance and crosstalk are affected by changes in width and spacing, further investigations could show how this causes changes in overall system performance. Changes need not be limited to width and spacing, but may extend to shape and the positions of neighbouring wires. In fact, there are many variables when considering a wire's performance, and this is why investigation here was limited to a single case.

Additional future topics could be to expand the NoC with respect to data formatting; particularly with an emphasis on optimisation for low latency routing; and also the associated decoding schemes needed at the router nodes.

Bibliography

[1] Semiconductors Industry Association. National technology roadmap for semiconductors. 1997.

[2] John Bainbridge. *Asynchronous System-on-chip Interconnect*. Kluwer Academic Publishers, 2001.

[3] John Bainbridge and Steve Furber. CHAIN: A delay-insensitive chip area interconnect. *IEEE Micro*, 22:16–23, 2002.

[4] W. J. Bainbridge, L. A. Plana, and S. B. Furber. The design and test of a smartcard chip using a chain self-timed network-on-chip. In *DATE '04: Proceedings of the conference on Design, automation and test in Europe*, page 30274, Washington, DC, USA, 2004. IEEE Computer Society.

[5] Tobias Bjerregaard and Shankar Mahadevan. A survey of research and practices of network-on-chip. *ACM Comput. Surv.*, 38(1):1–51, 2006.

[6] R. E. Burgess. Electronic fluctuations in semiconductors. *British Journal of Applied Physics*, 6:185–190, June 1955.

[7] Richard Chang. *Near Speed-of-Light on-chip Electrical Interconnects*. PhD thesis, Stanford University, 2003.

[8] Philip Christie and Dirk Stroobandt. The interpretation and application of Rent's rule. *IEEE Trans. Very Large Scale Integr. Syst.*, 8(6):639–648, 2000.

[9] Danny Cohen. On holy wars and a plea for peace. *IEEE Computer Magazine*, October 1981.

[10] Douglas E. Comber. *Internetworking with TCP/IP: Principles, Protocols, and Architectures (Fourth Edition)*. Prentice Hall, 2000.

[11] Jason Cong and David Zhigang Pan. Interconnect performance estimation models for synthesis and design planning. Technical Report 980017, University of California, Los Angeles, 10, 1998.

[12] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley, 1991.

[13] William J. Dally and John W. Poulton. *Digital Systems Engineering*. Cambridge University Press, 1998.

[14] William J. Dally and Brian Towles. Route packets, not wires: On-chip interconnection networks. In *Design Automation Conference*, pages 684–689, 2001.

[15] Jo Ebergen, Steve Furber, Arash Saifhashemi, Naela Nissar, and Alex Chow. Notes on pulse signalling. In *Proc. 13th International Symposium on Asynchronous Circuits and Systems*. IEEE Computer Society Press, March 2007.

[16] E. Friedman. Clock distribution networks in synchronous digital integrated circuits. In *Proceedings of the IEEE, Vol 89, No. 5*, pages 665–692, May 2001.

[17] Stephen H. Hall, Garrett W. Hall, and James A. McCall. *High-Speed Digital System Design — A Handbook of Interconnect Theory and Design Practices*. Wiley, 2000.

[18] Fred Halsall. *Data Communications, Computer Networks and Open Systems*. Addison-Wesley, 1996.

[19] Richard W. Hamming. *Coding and Information Theory (Second Edition)*. Prentice-Hall, 1986.

[20] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach (Third Edition)*. Morgan Kaufmann, 2003.

[21] Ron Ho. *On-Chip Wires: Scaling and Efficiency*. PhD thesis, Department of Electrical Engineering, Stanford Universiy, Aug 2003.

[22] Ron Ho, John Gainsley, and Robert Drost. Long wires and asynchronous control. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 240–249. IEEE Computer Society Press, April 2004.

[23] Ron Ho, Kenneth W. Mai, and Mark A. Horowitz. The future of wires. In *Proceedings of the IEEE*, volume 89, pages 490–504, April 2001.

[24] Simon Hollis and Simon W. Moore. An area-efficient, pulse-based interconnect. In *Proc. International Symposium on Circuits and Systems (ISCAS)*, May 2006.

[25] Simon Hollis and Simon W. Moore. An asynchronous interconnect architecture for device security enhancement. In *Proc. 19th International Conference on VLSI Design*, Jan 2006.

[26] Simon Hollis and Simon W. Moore. RasP: An area-efficient, on-chip network. In *Proc. 24th International Conference on Computer Design (ICCD)*, Oct 2006.

[27] M. Horowitz, R. Ho, and K. Mai. The future of wires, 1999.

[28] Paul Horowitz and Winfield Hill. *The Art of Electronics (Second Edition)*. Cambridge University Press, 1989.

[29] John Hyde. *USB Design by Example*. Intel University Press, 1999.

[30] Y. Ismail and E. Friedman. Optimum repeater insertion based on a CMOS delay model for on-chip RLC interconnect. In *Proceedings of the IEEE ASIC Conference*, pages 369–373, September 1998.

[31] Yehea I. Ismail and Eby G. Friedman. Repeater insertion in RLC lines for minimum propagation delay. In *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, May 1999.

[32] Bob Sproull Ivan Sutherland and David Harris. *Logical Effort, Designing Fast CMOS Circuits*. Morgan Kaufmann, 1999.

[33] Howard Johnson and Martin Graham. *High-speed Digital Design: A Handbook of Black Magic*. Prentice Hall, 1993.

[34] Mark B. Josephs and Jelio T. Yantchev. CMOS design of the tree arbiter element. *IEEE Trans. Very Large Scale Integr. Syst.*, 4(4):472–476, 1996.

[35] S. Keshav. *An Engineering Approach to Computer Networking*. Addison-Wesley, 1997.

[36] David Kinniment. Synchronisation and arbitration circuits in digital systems. In *Proceedings of the IEE*, volume 123, pages 961–966, October 1976.

[37] David Kinniment. Synchronizers and arbiters. In *Proceedings of ACiD Working Group*, January 2005.

[38] Xiao-Chun Li, Jun-Fa Mao, Hui-Fen Huang, and Ye Lui. Global interconnect width and spacing optimization for latency, bandwidth and power dissipation. *IEEE Transactions on Electron Devices*, pages 2272–2279, 2005.

[39] Paul A. Lynn and Wolfgang Fuerst. *Introductory Digital Signal Processing with Computer Applications*. Wiley, 1994.

[40] F.J. Macwilliams and N. J. A. Sloane. *The Theory of Error-correcting Codes*. North-Holland, 1977.

[41] Magma Design Automation Ltd. Quickcap.

[42] Magma Design Automation Ltd. Quickind.

[43] Edward J. McCluskey. *Logic Design Principles*. Prentice-Hall, 1986.

[44] Charles E. Molnar and Ian W. Jones. Simple circuits that work for complicated reasons. In *ASYNC '00: Proceedings of the 6th International Symposium on Advanced Research in Asynchronous Circuits and Systems*, page 138, Washington, DC, USA, 2000. IEEE Computer Society.

[45] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8):114–117, 1965.

[46] Simon W. Moore, George Taylor, Robert Mullins, and Peter Robinson. Point to point GALS interconnect. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 69–75, April 2002.

[47] Robert Mullins and Simon W. Moore. Demystifying data-driven and pausible clocking schemes. In *Proceedings of the 18th UK Asynchronous Forum*, September 2006.

[48] David Newport. Cooling the communications revolution, *Available online at: http://www.ul.ie/elements/Issue5/David%20Newport.htm*.

[49] Larry L. Peterson and Bruce S. Davie. *Computer Networks: A Systems Approach (Third Edition)*. Morgan Kaufmann, 2003.

[50] W. Wesley Peterson and Jr. E. J. Weldon. *Error-correcting Codes (Second Edition)*. The MIT Press, 1972.

[51] D. Russell. *Principles of Computer Networking*. Cambridge University Press, 1989.

[52] T. Sakurai and A. R. Newton. Alpha-power law mosfet model and its applications to CMOS inverter delay and other formulas. In *IEEE Journal of Solid-State Circuits*, volume SC-25, pages 584–593, April 1990.

[53] Bruce Schneier. *Applied Cryptography*. Wiley, 1996.

[54] Mischa Schwartz. *Telecommunication Networks*. Addison-Wesley, 1988.

[55] Robert Sedgewick. *Algorithms (Second Edition)*. Addison Wesley, 1988.

[56] Jens Sparsø and Steve Furber, editors. *Principles of Asyncronous Circuit Design: A Systems Perspective*. Kluwer Academic Publishers, Boston, 2001.

[57] Ben G. Streetman and Sanjay Banerjee. *Solid State Electronic Devices (Fifth Edition)*. Prentice Hall, 2000.

[58] I. E. Sutherland. Micropipelines (the Turing award lecture). *Comm. A.C.M.*, 32(6):720–738, June 1989.

[59] Ivan Sutherland and Scott Fairbanks. GasP: A minimal FIFO control. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 46–53. IEEE Computer Society Press, March 2001.

[60] Ivan Sutherland and Jon Lexau. Designing fast asynchronous circuits. In *Proceedings of the Seventh International Symposium on Advanced Research in Asynchronous Circuits and Systems*. IEEE Computer Society Press, March 2001.

[61] Andrew S. Tanenbaum. *Computer Networks*. Prentice Hall, 1981.

[62] Synopsis ®. hspice.

[63] Donald E. Thomas and Philip R. Moorby. *The Verilog ®Hardware Description Language (Fourth Edition)*. Kluwer Academic Publishers, 1998.

[64] Ronald J. Tocci. *Digital Systems: Principles & Applications, 6th Edition*. Prentice Hall, 1995.

[65] Neil H.E. Weste and David Harris. *CMOS VLSI Design (Third Edition)*. Wesley, 2005.

All I know is what I read in the papers

Will Rogers

Spice wire model definition

A

Below I display the wire model as used for spice simulations of interconnects in this thesis. I display first the definition for a two-wire model, as used by my point-to-point and RasP interconnects, and then a five-wire model used for the forward data direction in the Chain system.

A.1 Two-wire model

```
.model U_RLC_lines2_newR U LEVEL=3 ELEV=2 DLEV=0 PLEV=1 LLEV=0 NL=2

** number of lumps per line
+ MAXL=100

** values are per metre
** gnd capacitors
+ CR1=0.224n CR2=0.224n R11=221.43k L11=1.652u R22=221.43k L22=1.652u
+ C12=88.2p L12=1.428u

** set conductance to zero and gnd plance resistance to 10 Ohms/m
+ RRR=10 GR1=0 GR2=0 G12=0
```

A.2 Five-wire model

```
.model U_RLC_lines5_newR U LEVEL=3 ELEV=2 DLEV=0 PLEV=1 LLEV=0 NL=5

** number of lumps per line
+ MAXL=100

** values are per metre
** gnd capacitors
+ CR1=0.224n CR2=0.222n CR3=0.223n CR4=0.219n CR5=0.224n

** gnd inductors - (however, not used with LLEV=0)
+ LR1=1.21u LR2=1.23u LR3=1.21u LR4=1.20u LR5=1.22u

** line 1
+ R11=221.43k L11=1.649u C12=88.4p L12=1.423u C13=2.17p L13=1.292u
+ C14=0.238p L14=1.231u C15=0.003p L15=1.163u

** line 2
+ R22=221.43k L22=1.662u C23=88.3p L23=1.41u C24=2.26p L24=1.303u
+ C25=0.258p L25=1.158u

** line 3
+ R33=221.43k L33=1.652u C34=97.8p L34=1.421u C35=5.7p L35=1.292u

** line 4
+ R44=221.43k L44=1.645u C45=87.5p L45=1.421u

** line 5
+ R55=221.43k L55=1.644u

** set conductance to zero and gnd plance resistance to 10 Ohms/m
+ RRR=10 GR1=0 GR2=0 GR3=0 GR4=0 GR5=0
+ G12=0 G13=0 G14=0 G15=0 G23=0 G24=0 G25=0 G34=0 G35=0 G45=0
```