Technical Report

UCAM-CL-TR-828
ISSN 1476-2986

Number 828

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

Exploring networks-on-chip for FPGAs

Rosemary M. Francis

January 2013

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

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

© 2013 Rosemary M. Francis

This technical report is based on a dissertation submitted July 2009 by the author for the degree of Doctor of Philosophy to the University of Cambridge, Darwin 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

Exploring Networks-on-Chip for FPGAs

Rosemary M. Francis

Summary

Developments in fabrication processes have shifted the cost ratio between wires and transistors to allow new trade-offs between computation and communication. Rising clock speeds have lead to multi-cycle cross-chip communication and pipelined buses. It is then a small step from pipelining to switching and the development of multi-core networked systems-on-chip. Modern FPGAs are also now home to complex systems-on-chip. A change in the way we structure the computation demands a change in the way we structure the communication on-chip.

This thesis looks at Network-on-Chip design for FPGAs beyond the trade-offs between hard (silicon) and soft (configurable) designs. FPGAs are capable of extremely flexible statically-routed bit-based wiring, but this flexibility comes at a high area, latency and power cost. Soft NoCs are able to maintain this flexibility, but do not necessarily make good use of the computation-communication trade-off. Hard NoCs are more efficient when used, but are forced to operate below capacity by the soft IP cores. It is also difficult to design hard NoCs with the flexibility needed without wasting silicon when the network is not used.

In the first part of this thesis I explore the capability of Time-Division Multiplexed (TDM) wiring to bridge the gap between the fine-grain static FPGA wiring and the bus-based dynamic routing of a NoC. By replacing some of the static FPGA wiring with TDM wiring I am able to time division multiplex hard routers and make better use of the non-configurable area. The cost of a hard network is reduced by moving some of the area cost from the routers into reusable TDM wiring components. The TDM wiring improves the interface between the hard routers and soft IP blocks which leads to higher logic density overall. I show that TDM wiring makes hard routers a flexible and efficient alternative to soft interconnect.

The second part of this thesis looks at the feasibility of replacing all static wiring on the FPGA with TDM wiring. The aim was to increase the routing capacity of the FPGA whilst decreasing the area used to implement it. An ECAD flow was developed to explore the extend to which the amount of wiring can be reduced. The results were then used to design the TDM circuitry.

My results show that an 80% reduction in the amount of wiring is possible though time-division multiplexing. This reduction is sufficient to increase the routing capacity of the FPGA whilst maintaining similar or better logic density. This TDM wiring can be used to implement area- and power-efficient hard networks-on-chip with good flexibility, as well as improving the performance of other hard IP blocks.

Acknowledgements

This thesis is as much the result of my experiments, as the combination of a vast array of differing opinions and suggestions. I believe that I learnt the most about the way in which to address my work when I was presented with conflicting advice and assumptions. This thesis has taught me the value of exchanging ideas. I very much hope that the resultant conclusions are greater than the sum of each part.

I chose to work in the Computer Architecture Group because of their support for each other and their drive to share expertise generously. I especially need to thank Robert Mullins, my second supervisor, for always listening, Arnab for asking the right questions and Andrew for explaining so many things patiently. Most of all I want to thank my supervisor, Simon Moore, for giving me his fully attention when I needed help, his excellent advice and for keeping me fed and accommodated throughout the last three years. It was also though him that I did an Altera Sponsored internship, without which I may not have decided to to a PhD.

I would also like to thank Microsoft for providing me with the opportunity to go to FPT'07 and get to know the greater FPGA community. Without this I would not have been able to spend time with Guy Lemieux, to whom I am grateful for taking the time to work with me over the summer of 2008 and introducing me to his team. Thank you to everyone who made me feel so welcome. I owe a lot to the FPGA groups at UBC, SFU and those at Imperial for their challenges and suggestions. My work would have suffered were it not for them.

The Windsurfing club have been fantastic for keeping me injured and beered up. It was largely due to the wind, rain and beer that I am still sane and to the injuries that I am finishing on time. Thank you to my friends Beck, Xanda, Hannah, Wendy and Jo for hugs, tea and cakes and to my family, Mum, Dad and Felicity for continued support and encouragement. You all now know more about FPGAs than is healthy and for that I am grateful. Thank you to Ruan for always being there and making me smile.

Darwin college has been my second home for my PhD, but I really have to thank Newnham college for getting me there. In particular, Katy Edgecombe for believing in me from the start. Without her I would never have had the courage to come so far.

Funding was provided by EPSRC through grant EP/D036895. It was though them that much of this was possible.

Contents

Chapter 1

Introduction

In this dissertation I explore a wide range of providing Field Programmable Gate Arrays (FPGAs) with Networks-on-Chip (NoCs). Investigation into existing methods show that, until now, no solution has been presented which takes full advantage of the unique platform of the FPGA or the demands and possibilities of NoCs.

I show that soft (configurable) NoCs do not offer a good trade off between computation and communication, but that hard (silicon) NoCs can be very area inefficient when unused routers are taken into account. I present a system of reconfigurable Time-Division Multiplexed (TDM) wiring which reduces the size of hard routers and provides similar flexibility as soft routers.

1.1 Motivation

The increasing fault rates of future technologies and the exponentially rising mask costs [54] are pushing up the cost of ASIC production. FPGAs have high volume production to serve low volume markets. The regularity of their design allows for efficient fault tolerance and redundancy. This means that the market for FPGAs is only going to expand as it becomes less economically feasible to produce a hard-core custom solution. FPGAs need to adapt to meet this demand while maintaining the levels of abstraction needed for platform independence and ease of programming.

Increase in FPGA capacity combined with CMOS technology scaling has resulted in increased demands on the configurable wiring architecture. As technology scaling favours transistors over wires [83], so the relative communication cost has increased. I have been informed by provate conversation that the silicon area given over to configurable wiring has remained at around 50% and the architecture relatively unchanged for the last few generations, but the complexity of the programmable logic elements has increased and the way we use FPGAs is changing.

FPGAs now support multi-core Systems-on-Chip (SoCs). Inter-core communication needs to be switched and pipelined in and out of these cores: a number of bus-based solutions exist [2, 7], but none offer a scalable solution for future SoCs; as the system increases in complexity bus performance drops off as more cores request bandwidth. Although buses can be pipelined and hierarchical structures implemented, but at some point the difference between a custom NoC and a hierarchical bus is difficult to distinguish and buses start to have the same design issues as NoCs.To cope with increasingly partitioned systems-on-chip with heterogeneous hard blocks within the FPGA, an adaptable, intelligent Network-on-Chip (NoC) will need to be available.

Many NoCs have been developed for ASICs and some have been adapted for use on an FPGA [66, 59, 71, 16, 23, 11]. These designs have all been expensive in terms of area and therefore power consumption and fail to exploit the advantages offered by the unique design flow in FPGAs. The goal of this thesis is to apply techniques learnt from ASIC NoC design, and use them to design an FPGA NoC with the following characteristics in mind:

- Low development cost
- System-specific customisation
- Flexibility
- Performance
- Area and power efficiency
- Regularity

These characterises have been selected because they describe the ways in which FPGAs differ from or are similar to ASICs. The low development costs are counter intuitive to the way in which FPGAs have been used in the past. Their capacity for custom configuration now has to be weighed against the cost of designing a complex system-on-chip so code reuse and parametrisable ready-made solutions are as much for FPGAs as for ASICs.

Homogeneous platforms such as Multi-Processor Systems-on-Chip (MPSoCs) are often used for applications that are not known at the time of chip design. For this reason they require homogeneous NoCs with a great deal of run-time flexibility. It is this area which has attracted most of the research into networks on-chip because it presents the most interesting challenges.

Heterogeneous Application-Specific Integrated Circuits (ASIC) have different requirements. The application, and therefore traffic pattern, is usually known at the time that the chip is designed. Flexibility in the NoC can be discarded at design time in favour of a heterogeneous custom design.

Systems-on-FPGAs are different again. They must have the homogeneity and flexibility of a MPSoC when the chip is make, but that flexibility need not be reflected in the NoC at run time. For this reason FPGAs present a unique design platform for NoCs. The main implication of this observation is that soft (configurable) NoCs need not be as flexible as one design for a MPSoC. They can be simplified and made more efficient. Conversely, hard (silicon) NoCs would need to have all the flexibility of a multi-processor NoC and so would have to be proportionally more complicated. Many of the benefits of hard structures would therefore be lost.

Migrating commonly used structures such as processor cores and digital signal processing modules to hard (silicon) blocks is a technique employed by most FPGA manufacturers. This serves not only to cut down on power consumption, but also makes more effective use of the chip area. While hard blocks go a long way to accelerate the performance of mapped systems, the reconfigurable FPGA fabric still lags behind in performance, power and area efficiency [52]. The decision to implement the NoC in hard silicon or soft reconfigurable fabric is one addressed in this thesis. A hard FPGA NoC will be more area and power efficient, but limited by the data injection rate of the slower soft (reconfigurable) cores.

Hard networks are more efficient than soft networks, but only if they serve the needs of the system. The cost of unused hard blocks has to be taken into account when calculating the area of any network. Restrictions on the hard-soft interface can also limit the performance and area efficiency of a hard network; hard networks can switch data quickly and within a small area, but the reconfigurable fabric may lack the capability to produce and route this data to and from the router.

Conversely, a soft network may have very different communication-computation trade-offs, which will lead to different network architectures being economically feasible. They have a great deal of flexibility, but pay a high price for that in power, performance and area.

1.2 Contribution Summary

I have been looking at a more effective way of bridging the gap between fine-grain configurable static routing and coarse grain NoC for FPGAs. In this thesis I propose the development of an FPGA architecture with time-division multiplexed (TDM) wiring to meet the demand of future applications on FPGAs. To reduce the amount of wiring on the FPGA, I have been working on sharing the wires effectively without requiring significantly more configuration logic or significantly extending the critical path. I have developed a scheme whereby configurable interconnect wires are shared by TDM signals in a pre-scheduled network. These TDM wires are then used to route global wiring structures and funnel data in and out of the hard routers efficiently. This scheme is compared to a soft network-on-chip and the differences evaluated.

1.2.1 Publications

Many of the observations, results, and conclusions in this thesis have been published following peer review.

Conference Papers

A Network of Time-Division Multiplexed Wiring for FPGAs [38]
Full paper published at the Network-on-Chip Symposium 2008.

Exploring Hard and Soft Networks-on-Chip for FPGAs [36]
Short paper published at the International Conference on Field Programmable Technology 2008.

Talks

Fine-Grain Time Division Multiplexing in FPGAs
Represented at the HiPEAC Interconnection Network Architectures workshop 2008.

On-chip Interconnect Design for the Past, Present and Future
Invited guest lecture given to graduate students at Simon Fraser University 2008.

Poster Presentations

FPGAs with Time-Division Multiplexed Wiring: an architectural exploration and area analysis [37]
Abstract to be published at the International Symposium on FPGAs 2009

On-chip Networks for FPGAs
Presented at the Microsoft Summer School 2007. Won first prize for best presentation.

1.3 Overview

Chapter 1: Introduction

An introduction to FPGAs and the evolution of on-chip networks. The motivation for the research and how the contributions will make a difference to the industry and the world at large.

Chapter 2: FPGA Architecture

The history and development of FPGAs. This chapter looks at academic and commercial ECAD tools and architectures.

Chapter 3: Networks-on-Chip

Off-chip networking and how it has been adapted for on-chip interconnect. A look at existing hard and soft networks for FPGAs.

Chapter 4: Time-Division Multiplexed Wiring

A look at multi-context, pipelining and time-division multiplexing for FPGAs. This chapter describes the details of the TDM wiring under investigation in this thesis.

Chapter 5: Networks-on-Chip for FPGAs

A comparison between a selection of hard and soft networks for FPGAs. This chapter shows how the two compare for area, power and flexibility. Networks with TDM wiring are compared with those using conventional static FPGA wiring.

Chapter 6: Scheduling of Time-Division Multiplexed Wiring

A detailed description of the scheduling tool used to map benchmarks to FPGAs with TDM wiring. The tool is designed to sweep the design space of TDM wiring to find a practical solution.

Chapter 7: Scheduling Results

The results of using the scheduling tool to map benchmarks to an FPGA with TDM wiring. The results show how far the algorithm can reasonably trade off wire count for latency.

Chapter 8: Conclusion

A summary the conclusions drawn from the results. This chapter also looks at future work and research questions raised by this study.

Chapter 2

FPGA Architecture

2.1 Introduction

Field Programmable Gates Arrays (FPGAs) comprise arrays of Look-Up Tables (LUTs) and registers connected via a mesh of configurable wires. Logic functions are implemented in the lookup tables and large circuits can be built up by configuring the SRAM controlled switches in the wiring. As transistors have got cheaper these components have increased in complexity. The days when you could think of FPGAs as simple arrays of lookup tables (LUTs), registers and configurable wiring are long gone. These LUT-register pairs, known as basic logic blocks, now comprise a tiny proportion of the configurable resources, even in very simple devices. The complexity of the devices has grown over the years and FPGAs are now home to complex systems-on-chip.

2.1.1 History

It is impossible to talk about the development of FPGAs without looking at the history of Xilinx and Altera, the two leading FPGA manufacturers. FPGAs were invented by Ross Freeman [24], co-founder of Xilinx, in 1984. He was a forward thinking man who realised that if Moore's Law [61] continued to hold true then transistor counts would double every 18 months and it would become viable to implement logic on arrays of LUTs.

At this time Altera was a manufacturer of CPLDs, a similar, but less flexible device. Altera entered the FPGA market in 1992 soon after FPGAs because large enough to be widely adopted. Since then academic FPGA models have developed from a technology mapping tool VT-Pack and a placement and routing tool VPR (Versatile Place and Route) [19]. These tools were originally developed in 1998 by Right Track CAD Corporation in partnership with Altera. When Right Track CAD Corporation was bought out by Altera, the tools developed on two fronts.

Altera started work on developing the tools for advanced commercial devices and the original version of the tools was released for academic use as VPR.

2.2 ECAD Flow

Modern FPGA Electronic Computer Aided Design (ECAD) flows strive to take a Hardware Description Language (HDL) netlist and produce a bit stream, with which to program an FPGA, at the touch of a button. To some extent they have achieved this, but to get really good results it is generally necessary to understand a little about the FPGA architecture and the tool flow.

A typical flow from an HDL description to a bit stream for programming an FPGA will comprise of the following steps:-

1. **Synthesis** begins by converting the high level description into a series of lower level components. Several academic tools are available for synthesis [48, 72].

2. **Optimisation** then continues by iterating over a series of algorithms. Although the target is eventually k-input lookup tables the tools often target 2-input gates for intermediate optimisation. The tools will try to limit delay by minimising depth of the logic between pipeline stages.

3. **Technology mapping** converts the 2-input gate representation of the circuit to k-input lookup tables and a variety of tool are available to do this [39, 30].

4. **Packing** into clusters of lookup tables follows technology mapping. The lookup tables might be in homogeneous groups such as the Altera LABs or in heterogeneous pairs within a cluster like the Xilinx Configurable Logic Blocks (CLBs). At this point routing within the cluster is often performed. Clusters rarely have the same number of input pins as the total number of inputs to the lookup tables. There is usually a lot of internal routing. Versatile Technology-Pack (VT-pack) is the main free tool available to do this.

5. **Placement** (also known as fitting) of the clusters tries to find the optimal positioning of the clusters. This is generally timing driven, but is sometimes motivated by minimising routing resource requirements.

6. **Routing** allocates wires in the design (called nets) to wires of the FPGA. Some wires and buffers are optimised for speed, others for distance and others for power. Versatile Place and Router (VPR) is the main academic tool used for placement, routing, and timing analysis [19].

Algorithms and heuristics for performing these tasks within sensible computational limits eventually dominate the performance of the system on the FPGA. The tool choice is important as designs can been extremely sensitive to the algorithms used for optimisation and mapping [84].

2.3 Academic FPGA models

Academic FPGA models represent good models of early FPGAs, but are understandably slow to adopt new features. Most of FPGA academic research has been influenced by the versatility of VPR, which is unique in exposing the architectural description. The architectural description is a list of parameter settings that describe the FPGA. It is configurable within the parameters available. From the architectural description VPR computes the minimum number of wires (channel width) needed to route the circuit.

2.3.1 VPR Model

The FPGA is modelled as an array of logic clusters [20] surrounded by configurable wiring segments, known as an island-style architecture. This is following a commercial move from the array of basic logic blocks to clusters of basic logic blocks. The size of the cluster and the size of the lookup table was investigated using VPR [12].

With the move to logic clusters, the wiring was decoupled from the logic and the number of inputs to a cluster depopulated [18], forcing LUTs to share inputs within a cluster.

The FPGA wiring consists of a series of wire segments spanning one or more logic clusters. The wire segments are staggered so that the architecture is truly homogeneous. The wire segments are joined via a series of switches. The switches can be pass transistors or tri-state buffers. A mix of pass transistors and tri-state buffers can be used to drive wire segments spanning multiple logic blocks [22].

Wires in VPR are bidirectional and can be driven by local wiring from logic clusters or by other wire segments. Switches in which wire segments connect to other wire segments are arranged in a *switchbox*. There is one switchbox per logic cluster. Switches connecting the logic clusters to the global wiring are arranged into a *connection box*. Switchboxes cannot afford to be fully connected and various switchbox architectures have been presented such as the Universal or Wilton designs [35, 60]. VPR has a selection to choose from, the Wilton switchbox being the default. The optimisations rely on the trade-off between resources and flexibility. Academic and commercial design indicated that it is better to have more wire segments than necessary and less flexibility in the number of connections available to those components [68].

2.3.2 Further Academic Developments

Over the years the original VPR models has been built upon and improved; sometimes independently, and sometimes in conjunction with the original designers. A new version of VPR has recently been released and contains many of the following features found on modern FPGAs.

Single Driver Wiring In the early VPR model, each wire segment could be driven by multiple switches and was truly bidirectional. Since then, single-driver, directional wiring has been shown to be more effective [55]. Single-driver wiring is driven by a single mux constructed from nMOS pass transistors and is used in modern FPGAs. It removes the need for a separate connection box and switchbox.

Embedded Hard Blocks Hard blocks are defined as those implemented in silicon alongside the arrays of configurable logic. Embedded memory blocks [82] are needed to implement most systems on an FPGA. Modern FPGAs contain multi-ported RAM blocks [46] and hard DSP blocks. Larger hard functions include hard processors or floating point units [45].

Hard Wired Routing Patterns A detailed investigation into Hard Wired routing Patterns (HARP) [75] showed that L-shaped wire segments can bypass switch boxes, reducing power consumption and giving symmetric routing patterns. This L-shaped wiring has been used in the latest Virtex device from Xilinx.

Clock Networks Academic models generally assume a single clock source for simplicity, but larger systems on FPGAs need different clock frequencies and phases to cope with complex off-chip communication and varying IP core performance. Research into trade-offs between area, power and flexibility [53] have considered designs mirroring those found in industry. The clock networks investigated all consisted of a number of global clocks driving a hierarchy of local clocks. Logic blocks can generally select from two or more clock sources which in turn can be selected from a number of high level sources. This is not necessarily a fully static configuration. Some high level control is available at runtime, so globally distributed clock signals can change sources though user controlled logic.

2.4 Stratix Family Architecture

Altera is one of the world's leading FPGA manufacturers with their high performance Stratix devices. After purchasing Right Track CAD Corporation, Altera released VT-pack and VPR for research use and went on to refine them for the design of the Stratix. It is not surprising therefore that the highly optimised architectures produced by Altera confirmed results published by academics using the same tools.

Altera's FPGAs [56] continue to resemble those designed in academia, but have been optimised and developed beyond the modelling capabilities of the free tools. The island-style routing has been extended to include longer wires with hierarchical connections. This means that long wires can connect to shorter wire segments, but are unable to connect to the logic clusters directly.

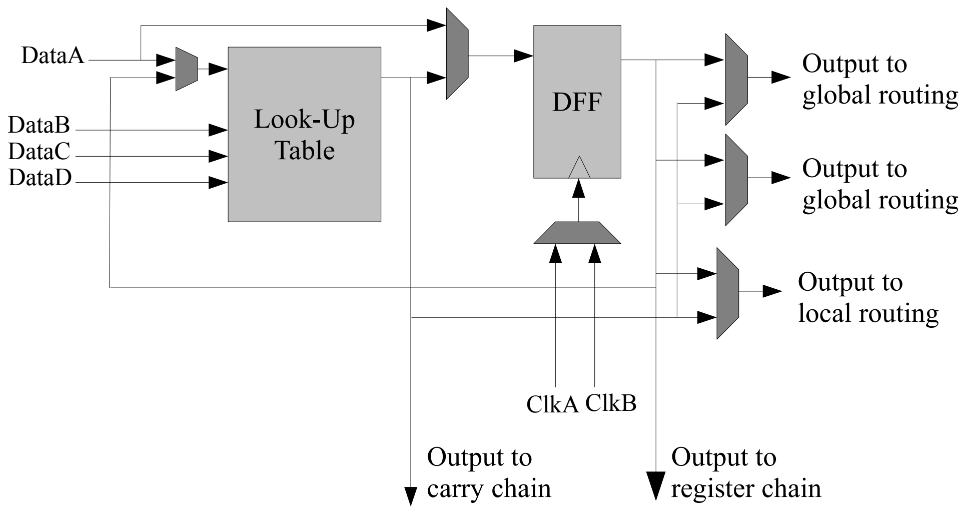


Figure 2.1: Stratix Logic Element.

The family now includes Stratix II, III, and IV and each member can be purchased in a variety of sizes and hard features. Several of the FPGA models presented in this thesis are based on Stratix and StratixII devices and their main features are outlined below.

2.4.1 Stratix

Details of the Stratix Architecture have been taken from the Stratix Device Handbook [8]. They are much smaller than more modern devices, but for military applications are still the preferred choice because of their reliability. They are implemented in $130nm$ technology.

Logic Block Clusters

The Stratix clusters are called Logic Array Blocks (LABs) and contain ten logic elements. Each logic element has a 4-input LUT and a register. Both components have bypass and feedback circuits to allow them to be interchanged within a circuit or used independently. Figure 2.1 shows the internal structure of a logic element.

The LUTs are joined via an arithmetic carry chain and the registers can also be chained. The registers can select from two clk sources and asynchronous and synchronous resets. The LAB is served by local routing. This connects the LAB to neighbouring LABs and the global routing and drives the logic element.

Clusters of Logic Elements are served by local routing as well as horizontal and vertical global routing *channels*. The logic elements may drive the local or global wiring directly. Each logic element has multiple outputs to allow both the LUT and register to drive the global wiring independently.

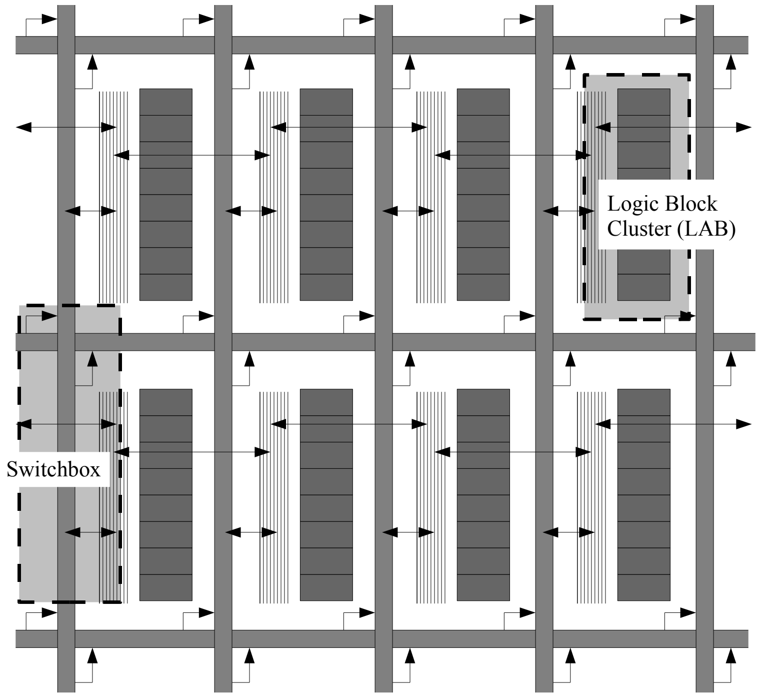


Figure 2.2: Stratix Global and Local Routing.

Routing Architecture

The Stratix has unidirectional single-driver wiring. Each global wiring segment is controlled by a single mux. This mux is driven by a mixture of local and global routing. There is no logical division between local-global and global-global connections so the notion of a connection box is abandoned in favour of a switchbox containing all global wire drivers. This is shown in Figure 2.2.

Horizontal (row) wires segments span 4, 8, or 24 clusters. Vertical (column) wire segments span 4, 8, or 16 clusters. All wires have a single driver with the same number of wires being driven in each direction. Wires are staggered so that there is a constant channel width. The distribution of wire lengths is summarised in Table 2.1.

The number of wires per channel can be computed from the number of wires driven at each switchbox. For example, Each R4 wire spans 4 switchboxes and so the number of R4 wires per channel is:

$$\begin{aligned} \text{Wires driven} \times \text{Length} = &\quad \text{Wires per Channel} \\ 40 \times 4 = &\quad 160 \end{aligned}$$

The total channel width is the sum of all wires passing though a given switchbox. For the Stratix this is 160 + 48 + 24 = 232 wires running horizontally and 80 + 32 + 16 = 128 wires running vertically. There are more horizontal wires than vertical in each channel despite evidence that uniform routing patterns in both horizontal and vertical directions across the FPGA is more

Table 2.1: Distribution of Wiring in the Stratix device

Wire ID	Length	Direction	Wires per channel	Wires driven per switchbox
R4	4	Horizontal	160	40
R8	8	Horizontal	48	6
R24	24	Horizontal	24	1
C4	4	Vertical	80	20
C8	8	Vertical	32	4
C16	16	Vertical	16	1

efficient [17]. This is because the aspect ratio of the tiles comprising of a logic block cluster and a switchbox is approximately 2:1 and so there are more vertical channels than horizontal channels. Overall the number of horizontal wires and vertical wires in a given square section is approximately equal.

Channels running around the boundaries of the FPGA are wider to give better pin access. Near the edge of the chip the number of wires driven and their length is adjusted to given uniform channel width rather than uniform switchboxes. This uniformity is critical to achieving high yield in the device. It allows extra columns to be included for fault tolerance, without which the cost of each working device would rise sharply with increased fault rates.

The wires area arranged so that both the vertical and horizontal channels have the same number of wires running in each direction. This is why there has to be an even number of wires of each type driven per switchbox. The exceptions to this are the long wires which alternate their direction between neighbouring switchboxes.

The switchboxes and logic clusters are hand placed for highly optimal performance. The wires are individually optimised for area, power or speed so wires of the same type may have different performance. This allows the tools to trade-off between area, power and performance and get the best from the device for each circuit [21].

2.4.2 Stratix II

The StratixII was developed in 2004 [56]. The driver behind the architectural modifications was the move to $90nm$ technology. The details presented here are taken from the Stratix II Device Handbook [9] or derived from the tools.

Logic Block Clusters

The number of logic elements was reduced from ten in the Stratix to eight in the Stratix II but the capabilities and complexity of these logic elements was increased. The single 4-input LUT and

Table 2.2: Distribution of Wiring in the Stratix II device.

Wire ID	Length	Direction	Wires per channel	Wires driven per switchbox
Local	2	Horizontal	132	44
R4	4	Horizontal	208	52
R24	24	Horizontal	24	1
C4	4	Vertical	128	32
C16	16	Vertical	16	1

register pair was replaced by an 8-input adaptive LUT, two full adders and two registers. The 8-input adaptive LUT is capable of operating in different modes to implement two functions of between 3 and 6 inputs. Where the total number of inputs is more than 8, inputs can be shared. In this way two 5-input functions can be implemented. The adaptive LUT is backwards compatible with two 4-LUTs. This move is likely to be in response to academic pressure to move to heterogeneous LUT designs [43]. In other aspects the logic clusters remain the same.

Routing Architecture

The routing architecture changed materially little between the Stratix and Stratix II. The wires spanning eight clusters were abolished in favour of more wires spanning four clusters and more local wiring. The aspect ratio of the switchbox-cluster tile remains the same. The wiring counts are shown in table 2.2.

2.5 Virtex Family Architecture

Xilinx brought out the Virtex Family in 1998, four years before the Stratix. Many believe Xilinx to be the architectural leader [70], but there is little to choose between the vendors. They offer equivalent devices to fill every corner of their shared market in SRAM-based FPGAs [7, 2].

2.5.1 Virtex-4

The Virtex-4 was fabricated in $90nm$ technology and is the equivalent of the Stratix II. Details presented here have been taken from the Virtex-4 FPGA User Guide [7] or derived from the tools.

Logic Block Clusters

Logic Block Clusters are known as Configurable Logic Blocks (CLBs) on Virtex devices. They consist of four *slices*, each slice containing two 4-input LUTs and two registers. They are

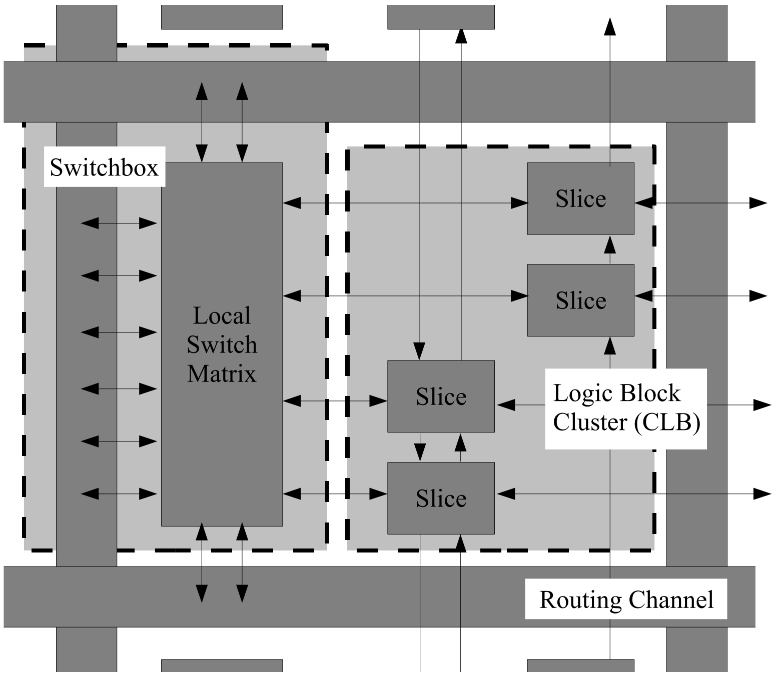


Figure 2.3: Virtex Global and Local Routing.

functionally very similar to the Stratix II logic elements. The slices are grouped into two pairs with two separate carry chains. This allows the logic to be better interleaved with the switchbox and maintains a 1:1 aspect ratio of the switchbox-cluster tile.

The Virtex-4 CLBs are half the size of the Stratix II LABs with four Virtex-4 slices in a CLB, compared to the eight StratixII adaptive LUTs in a LAB. The local routing is very similar however. Direct connections to neighbouring logic clusters exist for local communication. The logic elements connect to the global routing via a switch matrix.

Routing Architecture

The routing architecture is similar to the Stratix routing architecture. It differs mainly because of the smaller logic clusters and 1:1 aspect ratio of the tiles. The most commonly available wires span 6 logic clusters, but are similar length to the R4 and C4 wires on the Stratix II. The longest wires are bidirectional. The wiring distribution is summarised in table 2.3.

2.6 Summary

FPGAs have increased in complexity over the years, but the underlying island-style architecture has remained the same. FPGA designers have exploited the changing trade-offs between the cost of wires and the cost of transistors by increasing the complexity of the logic clusters, but the interconnect has remained reasonably unchanged. It is time that this was reviewed in light of

Wire ID	Length	Direction	Wires per channel	Wires driven per switchbox
Double	2	Horizontal	132	44
H Hex	6	Horizontal	60	20
H Long	24	Horizontal	24	2
V Hex	6	Vertical	60	20
V Long	24	Vertical	24	2

Table 2.3: Distribution of Wiring in the Virtex-4 device.

the interconnect revolution in the ASIC world [33]. As FPGAs host complex systems-on-chip they must provide modern interconnect architectures. The interconnect infrastructure must be redesigned to do this.

Chapter 3

Networks-on-Chip

3.1 Introduction

This chapter gives an overview of networking terms and practices, and a survey of existing Networks-on-Chip (NoCs). *Principles and Practices of Interconnection Networks* [34] has provided a great deal of the networking background, but here the emphasis is on on-chip networks and the techniques relevant to FPGA NoC design.

3.2 Topology

The topology of a network is the arrangement and connectivity of *nodes* and *channels* within a network. Network nodes may have local connections to allow data on and off the network, these are known as *terminal* nodes and form a subset of the set of nodes. Non-terminal nodes are often referred to as *switch* nodes. The *degree* of the node is the number of channels connecting to it. Out degree and in degree can differ, but these are assumed to be equal unless otherwise stated. Channels connect to precisely two nodes and are directional. They should not be confused with routing channels on an FPGA. The suitability of a topology for a network depends on a number of trade-offs. In general the performance of a network is defined by the throughput, latency and path diversity.

Latency The latency of a network depends on the number of *hops* a data must travel between its source and destination and the latency of each hop. A hop is usually taken to be a router or switch and a section of interconnection wire. The number of hops can be optimised for the average case or the worst case. The latency of each hop will depend on the complexity of the router. The number of hops between two terminals in a network may vary depending on the path chosen. The *minimum hop count* between two nodes in the network is the minimum number of nodes data must pass though to get from the input node to the the output node over all possible paths between the two nodes. The *diameter* of a network is the highest minimum hop count over all the possible input-output node pairs in the network.

Bandwidth The bandwidth of a channel is the maximum rate of data transfer in that channel. It can be calculated by multiplying the channel bit width with the maximum operating frequency.The bandwidth of channels need not be uniform across the network. Bandwidth is often given is a measure of network performance, but should not be confused with throughput. The throughput of a network is the rate of data transfer at run-time. The throughput will depend on the node architecture and the traffic patterns, as well as the topology.

Path Diversity The path diversity is the number of paths between any two terminal nodes in a network. Path diversity in NoCs is used for fault tolerance and congestion management by dynamic routing algorithms [13].

A *circuit* can be set up through a network by allocating a set of nodes and channels in a path between an input node and an output node. These resources should be sufficient to allow data to be transferred between the input node and output node and should not be used for any other communication whilst the circuit is in place.

3.2.1 Butterfly Networks

Butterfly networks [32] have very low hop count, but are rarely used in NoCs because there is no path diversity; there is exactly one path between any two terminal nodes. The hop count is low on average, but it is also constant and so forces other aspects of the network to scale poorly.

3.2.2 Clos Networks

Clos Networks [27] are visually similar to butterfly networks, but differ in significant ways. Importantly, there are multiple paths from any one terminal node to another. Further more, a clos network is *incrementally non-blocking*. A non-blocking network allows any set of inputs to connect to any set of outputs at any given time assuming that the set of input-output pairs are known before the circuits are set up. An incrementally non-blocking network allows a circuit to be set up between any free input and output node regardless of the existing state of the network. That is to say that no communication can be blocked on another circuit and that the circuits for each possible communication pattern can be set up incrementally.

Clos networks are have three stages described by a tuple (m, n, r). This means the middle stage has m switch nodes, the first and third stages have n switch nodes and each of those $2n$ nodes have r terminal nodes connected to them. Clos networks with more stages can be built by recursively replacing some of the switches with clos networks.

3.2.3 Torus Networks

Torus and mesh networks have a variety of attractive features. They have good path diversity and connection redundancy without being totally connected. This allows adaptive routing to manage traffic congestion and dynamic faults, but without requiring excessive on-chip resources. Torus and mesh networks generally combine switch and terminal nodes into a single node type. On-chip these can be evenly distributed across the chip.

In a mesh network the hop count and latency increases roughly linearly with on-chip distance. To some extent this is true of torus networks, but due to the cyclic nature of the channel patterns, terminals located adjacently on-chip can be many hops apart on the network. This is undesirable in some cases, but is traded-off against lowered hop counts for medium-distance communication.

3.3 Data Packets

Network data is grouped into *packets*. Packets can be a fraction of the channel width or be very much larger. A network may have a fixed or variable packet length. Typically a packet will be prefixed with information about its destination. In a network with variable length packets the length must be included or a footer marker appended to the end of the packet.

Packets are usually divided into *flits*. A flit is usually the same size as the channel width and is the largest amount of data that can be transferred in parallel. It is therefore the smallest amount of data to which resources can be allocated.

3.4 Routing

The method by which packets are directed from source to destination can be deterministic, oblivious or adaptive. Networks with no path diversity are forced to used deterministic routing methods; other topologies may use *oblivious* or *adaptive* algorithms.

Oblivious algorithms route packets without any knowledge of the state of the network. Deterministic algorithms are a subset of oblivious algorithms. Oblivious algorithms are easy to implement and make deadlock free.

Adaptive algorithms may use information about faults or congestion to route packets or circuits.

3.4.1 Source Routing

Source routing requires the path through switching nodes to be pre-computed at source. This requires a global knowledge of the network state and the state of the network, but can reduce latency though switch nodes.

3.4.2 Table-Based Routing

Table based routing can be implemented at the source or at switch nodes. Only the section of the table pertaining to a particular node needs to be stored at that node. The table may store multiple paths for a particular destination from a given node.

3.4.3 Algorithmic Routing

Algorithmic routing can be used to compute the output port from the destination ID. For example, in a mesh network the node needs only determine the relative direction of the destination from the destination ID and the current node ID if absolute addressing is used. Otherwise the direction can be computed from a relative address and the header packet updated for the next node.

3.4.4 Deadlock Avoidance

In order to avoid network deadlock all cyclic dependencies must be eliminated. The West-First, North-Last algorithm is commonly used to impose a total order on the the network channels of mesh networks. This is implemented by forcing packets to travel west before travelling north or south and east before north. This is a simple solution, but removes some path diversity from the network.

3.5 Flow Control

Flow control can be defined as the allocation of resources and the resolution of conflicts.

3.5.1 Switching

Switching is usually performed at the circuit or packet level, but can also be done at lower levels.

Circuit switching is the simpler of the two and involves the allocation of resources to an entire circuit. Circuits are set up by sending a header packet. An end-to-end circuit is set up and maintained for the duration of the data transfer. The circuit resources are only deallocated once the transfer is complete and the last packet sent. The setup cost can be high, but streaming transfer have very low latency overall.

Networks employing packet switching have to route and switch every packet. Resources are deallocated as soon as the packet has been forwarded to the next node. The arbitration costs are higher than with circuit switched networks, but the resource allocation is more efficient.

3.5.2 Buffering

In a bufferless network packets are dropped when the next resources in the path are not available. Reliability has to implemented at a higher level. Bufferless networks are usually unsuitable for NoCs because few on-chip applications can tolerate packet loss and to implement reliability at a higher level requires significant additional latency and switching.

Circuit switching requires only that the header packet is buffered. Once resources have been allocated to the circuit, subsequent packets are forwarded without buffering.

When packet switching all packets are buffered pending resource allocation. Complex flow control must be employed to ensure efficient use of buffering resources.

Store and Forward

Store and forward buffer control requires that the entire packet is buffered at a network node before it can be forwarded. Before a buffer can be forwarded it must have been fully received and been allocated access to the switch. The output channel must be available and a packet sized buffer must be free at the next node.

Cut Through

Cut though buffer control is similar to store and forward, but the packet can be forwarded as soon as resources become available. The header flit can be forwarded as soon as it has been granted access to the switch, the channel, and a flit sized buffer on the next node.

Wormhole Flow Control

Wormhole flow control uses *virtual channels* to share physical channels more effectively. Many virtual channels represent one physical channel. A virtual channel maintains information about the output port and the buffer status. These physical resources are allocated at a flit level rather than a packet level. This means that a packet allocated to a virtual channel can begin to be forwarded as soon as there is sufficient buffer space to receive one flit rather that a whole packet. This is more efficient on buffer space than cut though.

3.5.3 Buffer Management

There are three main ways in which buffers can be managed.

Credit Based Flow Control

Buffer space at the channel sink is counted at both ends of the channel. At the sink the channel count is increased when a flit is sent and decreased when a flit is received. When a flit is sent a buffer credit is sent back down the channel to increase the buffer count at the channel source. The source buffer count is decreased when a flit is sent. No flits are sent unless the buffer count is non-zero.

If the buffer becomes full, no more flit will be sent until there is space again. The latency to restart the channel is twice the channel latency because a credit must be sent and acted upon.

On-off Flow Control

On-off flow control is simpler to implement than a credit based system. There are no counters; instead the sink node signals flow to be on or off. The off signal must be sent before the buffer is full in order for there to be space for the flits sent while the off signal reaches the channel source and is acted upon.

As with the credit based system there is twice the channel latency to restart flit transmission once it has been stopped.

Ack-nack Flow Control

The ack-nack protocol reduces the restart time of the channel because flits are sent regardless of buffer capacity. Upon receipt of a flit the sink node issued an acknowledge (ack) or buffer-full (nack) signal to indicate successful traversal of the channel or otherwise.

There is no gain overall for using this protocol because the flit must be stored at the source until an ack signal has been received. Buffer space is wasted if the network is minimally loaded. Packets may have to be sent many times in a congested network, which increases the power consumption at times of high activity.

3.6 Router Architecture

The router architecture determines how the network functionality is implemented.

3.6.1 Pipeline

The number of pipeline stages affects the throughput and latency of the router. Most virtual channel routers use the following pipeline stages:

1. Routing
2. Virtual channel allocation
3. Switch allocation
4. Switch traversal
5. Channel (interconnect) traversal

In many cases the switch traversal and channel traversal are one stage, known as the data path [65]. The other stages are known as the control path. Only virtual channel routers require the second pipeline stage.

Depending on the switching granularity, only the header flit or head packet needs to pass though the control path. Once switching has been performed subsequent data flits or packets can bypass the control path and be forwarded directly to the data path.

Speculation

The pipeline can be reduced to a single stage though the use of speculative routing and arbitration [63]. The data path is pre allocated based on speculative control signals. The virtual channel and switch allocation are performed in parallel to the switch traversal and abort signals are sent in the case of an incorrect speculative decision.

3.7 Networks-on-Chip for FPGAs

Designers of NoCs for FPGAs have to decide whether to implement a hard (silicon) network of limited configurability or a soft (configurable) network using the precious reconfigurable resources. Many hard and soft networks have been proposed and they trade off between the two modes of implementation in different ways.

3.7.1 Hard Routers

Hard (silicon) routers for FPGAs are seen to be more area-efficient, power-efficient and offer higher performance. This is has been the motivation behind the addition of hard IP blocks such as multipliers, DSP blocks, embedded RAM and hard processors to high-end commercial FPGAs. Despite this, relatively few designs for hard NoCs have been proposed.

A high-level simulation of a hard network has been proposed [44]. This was designed specifically to enable operating system support of dynamic reconfiguration. In this context the restriction in the granularity and positioning of the network does not need addressed; this is determined by the granularity and positioning of the dynamic reconfiguration blocks. No other design issues are addressed.

An area comparison between hard and soft networks has been made [41], however no details of the network architecture was given. A high granularity network may have a router for every 1400 LUTs, a reasonable size considering that a NiosII soft processor uses 1800 LUTs. In network tiles of this size the hard router would use around 10% of the silicon area.

3.7.2 Soft Routers

Soft routers are implemented using configurable resources on an FPGA. They are larger, slower, but more flexible than those implemented in silicon. Custom interconnect solutions with custom topologies are often used on FPGAs. Application specific designs benefit from knowing the application and the communication patterns and real time deadlines at design time. This flexibility often makes up for the lack of performance, though not without significant design costs. Commercial soft-core IP solutions are a popular way of cutting System-on-FPGA design costs. Parametrisable cores compromise between design costs and flexibility. Unused resources are not implemented and so no resources are wasted.

Various soft routers have been proposed, including a study between packet-switched and time-division multiplexed soft routers [51]. Both designs performed well in different circumstances, but the packet-switched network needed large buffering area and the TDM network required large context memory and compile-time scheduling.

Another soft network study looks into the routability of large networks with different topologies [69]. The network is unable to forward packets between nodes, packet forwarding and routing has to be performed at a higher level within each processing core attached to the network. Each router can transmit only one packet at a time. The packet is broadcast on all output ports along with the routing information instruction. The receiving routers ignore the packet unless it is destined for them. This study therefore says nothing about the implementation of the routers, but it does show that highly connected and complex topologies route well on modern FPGAs. Rings, stars, hypercubes, meshes, and point-to-point connections were all found to route with up to 20 cores.

Table 3.1: A summary of soft NoCs implemented on Xilinx Virtex2 devices.

Network	Frequency	Approx. router size	Width	Notes
RMBoC [11]	94 MHz	2400 LUTs	32 bit	1D Circuit switched
Dynoc [23]	77 MHz	2150 LUTs	32 bit	1D Circuit switched
Packet [16]	50 MHz	700 LUTs	16 bit	Virtual cut though
OS [59]	50 MHz	1222 LUTs	16 bit	Control and data networks for OS, partially bus-based
NoCem [71]	150 MHz	1400 LUTs	32 bit	Packet switched
CoNoChi [66]	88 MHz	820 LUTs	32 bit	Design for partial reconfiguration

Table 3.1 summarises various NoCs developed for implementation on FPGAs. They are common in achieving low clock rates and requiring a large number of LUTs. The Virtex2 device used for these networks will be a little slower than the Virtex4 used in my experiments later, but only fractionally; the maximum frequency of the Virtex2 is 420 MHz compared to 500 MHz on the Virtex4 [7]. The LUT count, on the other hand, is fabrication process independent. All LUT counts quoted are estimates based on 4-input LUT utilisation. The best LUT count achieved was 700 LUTs. If combined with a 1800 LUT NIOS II soft processor [2] the router would take up 28% of the component area.

3.8 Investigative conclusions

Complex virtual channel NoCs for ASICs consume 5-10% of the silicon area [14]. This will vary according to the complexity of the router and the size of the computational core it serves, but in general the complexity of the router increases with that of the core so should be true of most networked SoCs.

Soft routers can be configured immediately prior to run time and can be designed to meet the specific requirements of the system traffic pattern. Flexibility in the network is not as important because there is great flexibility in the reconfigurable fabric. We can therefore expect a soft router to be simpler and smaller that a virtual channel NoC and so the soft network should not consume more than 5% of the FPGA.

A soft core can be between 10 and 40 times larger than its equivalent hard implementation [52]. This means that the soft computational core served by an FPGA NoC router will be 10 to 40 times larger than the equivalent hard computational core. A hard router on an FPGA should use proportionally less of an FPGA network tile than an ASIC network tile. As there is a ten-fold increase in tile area between an ASIC to an FPGA, there should be a ten-fold decrease in router area as a proportion of the tile. The hard router should consume less than 1% of the silicon area, a tenth of than consumed on an ASIC NoC.

3.9 Summary

Networks have been developed for off-chip communication for decades and recently efforts have been made to adapt these techniques for on-chip communication. The change in trade-offs and the ways to deal with these changes are reasonably well understood for ASICs, but have not been investigated for FPGAs in much detail. Some attempts have been made, but reasonable inspection of the problem indicates that there are many improvements to be made. Few hard NoCs for FPGAs have been proposed. Those in existence try to provide a great deal of functionality, but at a high area cost.

The soft core router proposed has extremely high LUT use: a minimum of 28% of a NIOS II processor and router combined tile. NoCs exploit the change in computation-communication trade-off, but the high cost of implementing a complex soft core router does not seem to exploit that trade-off efficiently. This indicates that on FPGAs more wiring must be used to reduce the size of the soft router area.

Investigation into existing NoC designs for ASICs and FPGAs has led me to believe that the following goals are possible:

1. It should be possible to implement a hard network on an FPGA at high granularity while using less than 1% of the silicon area
2. It should be possible to implement a soft network on the FPGA at high granularity while using less than 5% of the reconfigurable area.

I aim to exploit the inherent flexibility of dynamic switching to implement a suitable NoC for an FPGA.

Chapter 4

Time-Division Multiplexed Wiring

4.1 Introduction

Time-Division Multiplexing (TDM) has been used to improve resource usage on FPGAs in areas of research such as multi-context FPGAs, but I aim to adapt the FPGA architecture by combining multi-context interconnect with built-in pipelining to allow time-division multiplexing over individual wire segments within the FPGA.

4.2 Multi-Context FPGAs

Multi-context FPGAs [77] are built by duplicating the configuration SRAM and cycling though each configuration in turn. This is performed in order to increase the capacity of the FPGA and permit large designs to fit on one chip that would otherwise have required more than one.

One problem associated with multi-context FPGAs is the partitioning of designs between contexts [50], particularly when the number of contexts is increased to eight or more [76]. Eight contexts reduces the overall resources on chip per context, but allows the chip to emulate up to eight devices. Off-chip communication costs are eliminated at the cost of slow operating frequencies. The extra configuration RAM can be used for user memory when it is not used to hold configuration information, but the cost of extra configuration SRAM remains high.

Another study has exploited redundancy and regularity between contexts to reduce context memory [26]. Maintaining configuration between contexts not only reduces the configuration SRAM, but also reduces the power consumption associated with reconfiguration.

To further this, it is possible to have multi-context wiring and single context Look-Up Tables (LUTs). Such an architecture has been proposed [58], which uses two-context wiring to exploit permutation equivalent LUTs. The wiring is not pipelined and so the latency of the circuit is doubled in order to increase the capacity of the FPGA.

4.3 Pipelined Architectures

FPGA throughput is hampered by the connection of relativity low-speed components in series. There have been several attempts to improve this via the introductions of pipelining components. This becomes of particular interest when across-chip communication involves multi-cycle transfers.

One proposed architecture used pipelined interconnect to help schedule data transfers between wave steered logic blocks [74]. The traditional LUTs are replaced by wave pipelined binary trees. The interconnect is pipelined and stalled using a two phase clock in order to satisfy the timing constraints of the logic. In my architecture we use a similar technique to stall data in order to satisfy resource constraints.

More commonly pipelining is used to share resources. The RaPiD architecture [73] seeks to pipeline the interconnect to improve clock rates. Registers are inserted at regular intervals between wire segments. Mapping relies on annotated Verilog. Fine-grain pipelining can be performed every time a signal passes though a logic block. Each LUT has a register and so pipelining almost comes for free at the local level. Globally, more sophisticated interconnect is needed than statically-configured pipelined interconnect.

The Regular Distributed Register Architecture [28] divides chips into tiles of single-cycle latency. These tiles are then connected via channels of multi-cycle signals. An extension to this architecture, AutoPipe [29], pipelines the multi-cycle interconnect to improve throughput and allow sharing of wires though statically scheduled time-division multiplexing. This work assumes clock speeds which far exceed those achievable on an FPGA. In order for this architecture to be extended for use on an FPGA, the interconnect must be run faster than the cores.

Recently, Achronix Semiconductor Corporation have released the Speedster FPGA [31], which they claim to be the “fastest FPGA in the world”. They use fine grain *picoPIPE* pipeline elements and asynchronous pipelining techniques to achieve clocks speeds of 1.5 GHz.

4.4 Time-Division Multiplexed Wiring

Multi-context wiring can be combined with pipelined interconnect in order to time-multiplex wires and share them at a very fine-grained level. Time-Division Multiplexed (TDM) wiring allows wires to be used by multiple signals within a design clock cycle. Previously static paths are pipelined over TDM wiring to achieve high data rates using fewer wires. The interconnect latches are clocked at a much higher rate than the design latches. Signals are serially scheduled at compile time onto shared wires by allocating them in time as well as space.

Similar architecture have been patented recently [79, 78] and outline additional components that must be incorporated into programmable logic devices to implement TDM wiring. Both describe TDM wiring that runs alongside the conventional static wiring, but do not consider the feasibility of designing or using such an architecture.

4.5 My Time-Multiplexed Wiring

To implement time-division multiplexing over global interconnect wiring, interconnect latches are added to each global wire segment and extra configuration bits are added to the switch boxes. The latches are clocked at a higher rate than the design clock to allow data to pass between logic block registers in one design clock cycle.

The TDM wiring is used by dividing each design clock cycle into a series of time slots, representing the cycles of the interconnect clock. There is a set of configuration bits for each time slot, allowing a new configuration of the wiring each time slot. Wires are shared by scheduling signals with different time slots onto the same wire. Each interconnect latch may have to hold its data for many time slots because the delay to the destination latches can vary between configurations.

The inputs to any given LUT must hold valid data simultaneously in order for the function in the LUT to be evaluated. With static wiring this is not a problem because each of the k inputs will be driven by k distinct wires. In the TDM wiring architecture it is important to share as many wires as possible. The inputs to the LUTs must be latched as well as the wires so that all the inputs to a LUT can be driven by a single wire. This is necessary to allow efficient sharing of the TDM wires.

4.5.1 Configuration SRAM

Static FPGA wiring is driven by a mux controlled by 8 bits of configuration SRAM [55] If the TDM wiring is capable of cycling though n configurations then the number of bits required to implement a single TDM wire is $(8+1)n$; for every cycle of the TDM wiring 8 bits are needed to control the mux, as for the static wiring, and one bit controls the latch.

The TDM wiring is statically scheduled at compile time. With n configurations it seems to the soft core that there are n wires. The TDM router sees the data flits from the soft cores broken down into n pieces sent serially over the TDM wiring.

This extra routing capability comes at a cost. If the NoC runs n times faster than the soft cores then n times as many SRAM bits are needed to control the input mux and a further n bits are needed to control the latch. This increases the area and power footprint of the NoC.

TDM wiring allows data to be pipelined and wires shared without using valuable configurable logic element resources.

4.5.2 Shared Configuration SRAM

Some work has been done in the past on shared-configuration interconnect (bus based wiring) [86]. With conventional wiring the area gains from sharing configuration have not been enough to make up for the loss in flexibility, but with TDM wiring there is great potential. TDM wiring needs to duplicate the configuration SRAM in order to reconfigure the mux driving the wiring multiple times during the soft core design clock cycle. With approximately half of the wiring overhead given over to SRAM already this is a potentially a large area overhead.

If $(8+1)n$ bits are needed to control a single TDM wire, but the configuration SRAM is shared amongst k wires, then $(8+1)n$ SRAM bits can configure k wires. If $n = k$ then the number of bits used to configure a single TDM wire is 9: just one more than that needed to configure a static wire.

Static wiring with shared configuration SRAM is less flexible because wires must be grouped together. The tools are unable to place and route the design as efficiently and the overall area gains from sharing configuration are low. NoCs with TDM wiring naturally group wires with identical routing patterns and so the disadvantages of sharing configuration SRAM are reduced.

The algorithms presented later in this dissertation are not able to group wires to share configuration SRAM because efforts have to be made pre-synthesis and thorough the CAD flow in order for the benefits to be evident. The wiring sharing algorithms in this dissertation are all applied after place-and-route where most of the high-level information used to compute efficient SRAM sharing has been lost. The idea is presented here as an obvious extension one would consider for commercial development of TDM architectures.

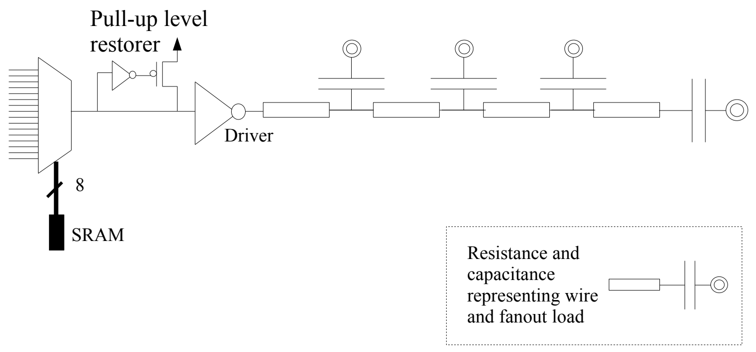
4.6 Wire Model

The wire model and component layouts were based on implementation assumptions made in previous work and generally accepted by the FPGA community [55]. They were constructed at the transistor level in HSpice [4].

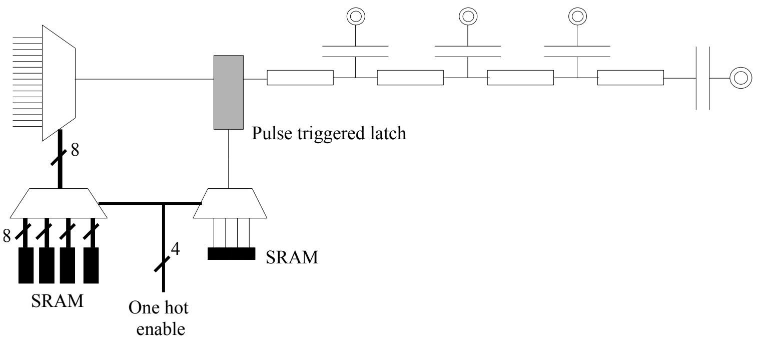
The muxes were modelled as pass transistors built from minimum width nMOS FETs. The wires are modelled as staggered resistors and load capacitance, as shown in figure 4.1. I have constructed models representing Altera's R4 and C4 wiring and Xilinx's Hex wiring. This is medium length ($< 1mm$), flexible wiring and the most commonly used on modern FPGAs.

The TDM wiring routes n different configurations, depending on the amount of extra SRAM. I have chosen n to be four because the hard routers are capable of running at over 800MHz. This is four times the frequency of a fast soft-core design [70]. The router with 8-bit TDM wiring operates four times as quickly as the 32-bit cores.

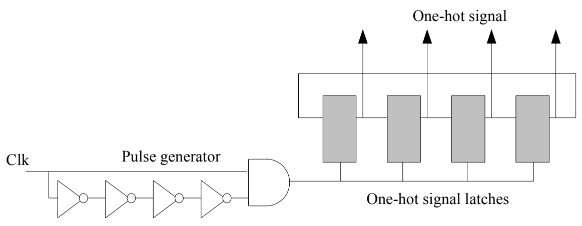
The components constructed are shown in figure 4.1. The TDM wiring circuitry is compared with that of the static wiring. The TDM control overhead is required once for every switch box.


Pull-up level restorer
Driver
8
SRAM
Resistance and capacitance representing wire and fanout load

(a) Conventional static wiring.



(b) Time-Division Multiplexed wiring with four sets of configuration SRAM.



(c) TDM control overhead.

Figure 4.1: Static and TDM wiring.

Using the HSpice models, I verified timing and measured the power consumption of the wire models.

I chose to use a pulse triggered latch for the interconnect latch because it is smaller than more complicated edge-triggered flip-flops and has a shorter setup and hold time. It is implimented two simple R-S flip flops with a clock pulse as the enable. This is shown in figure 4.2.

4.6.1 Alternative Circuit Implementation

Static FPGA wiring commonly uses eight SRAM bits to control a 16:1 mux [55]. This increases performance by decreasing the mux depth. It is implemented as five 4:1 muxes. While this is area efficient for static wiring, the increased SRAM cost of TDM wiring makes this less effec-

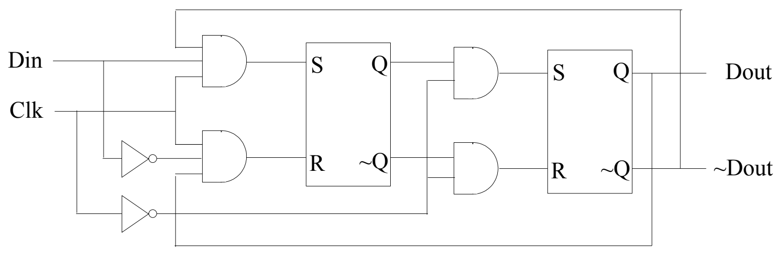
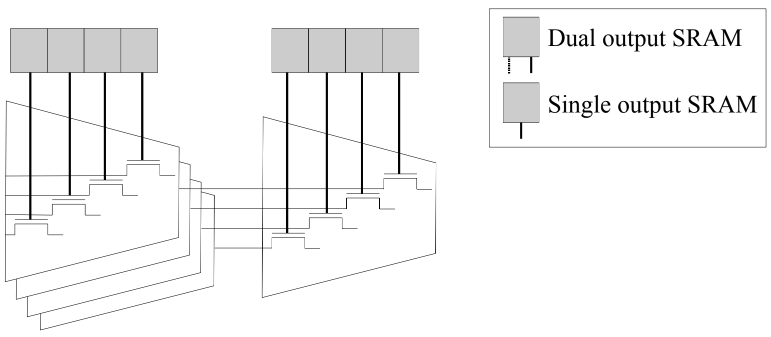
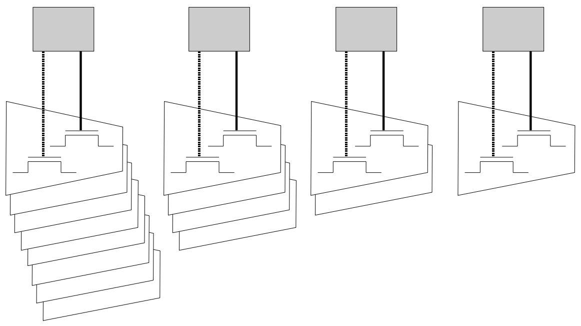


Figure 4.2: Pulse-triggered latch.



(a) 8-bit SRAM controlling five 4:1 muxes.



(b) 4-bit SRAM controlling fifteen 2:1 muxes.

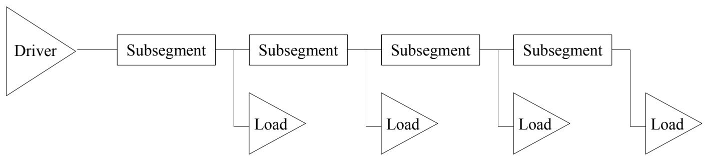
Figure 4.3: Two 16:1 muxes implemented for 4- and 8-bit SRAM.

tive. Four SRAM bits can still control a 16:1 mux, but that requires that the mux is implemented as fifteen 2:1 muxes or provide two 2-to-4 bit one-hot encoders. This is shown in Figure 4.3.

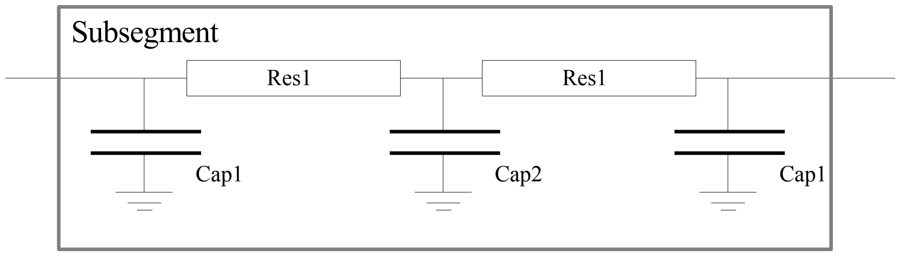
In later experiments, both circuit level implementations are used to evaluate the area of the TDM wiring.

4.6.2 Wire RC Model

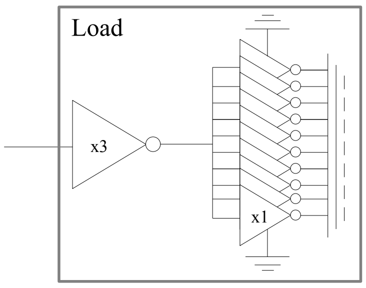
I created a model for Altera's R4 and C4 wire and for Xilinx' Hex wire. These are the most common wires found on the StratixII and Virtex-4 and are similar in design. Each wire is divided into four subsegments onto which a resistance and load model is applied. I use the term subsegment because the term subsegment is often used to describe an FPGA wire as part



(a) Wire model with four subsegments and four loads.



(b) Resistance and coupling capacitance for wire subsegments modelled as two resistors and three capacitors coupling with a ground plane.



(c) Load modelled as a $\times 3$ inverter driving ten minimum width inverters. The $\times 3$ inverter has the source and drain wired to power and ground, but the min inverters have power, ground and output wired to ground.

Figure 4.4: Diagram of resistance and capacitance model for FPGA wires.

of a larger configurable path. The subsegmented model is designed to model the coupling capacitance and fanout load of real wires in a device.

Resistance The subsegment resistance is modelled as two identical resistors in series. A coupling capacitance with the ground substrate is applied at three points. This is shown in figure 4.4.

Load The load on each subsegment is modelled as a single $\times 3$ inverter driving ten minimum width inverters. These minimum width inverters have all other terminals wired to ground. This is shown in figure 4.4.

The values used for each of the wire models are given in table 4.1. The lengths of each subsegment is the width or height of the StratixII switchbox and logic cluster for the StratixII wires because this is a quarter of the wire length. The subsegments of the hex wires is $6/4$ of the width or height of a Virtex-4 logic cluster and switchbox because the wire spans six clusters.

Table 4.1: Values of resistance and capacitance used for modelling the wires. Please refer to figure 4.4 for definitions of the values given.

Wire type	Metal layer	Metal width (μm)	Subsegment length (μm)	Res1 (Ω)	Cap1 (fF)	Cap2 (fF)
Altera R4	5	0.14	135	39.50	2.54	5.08
Altera C4	6	0.14	218	63.06	4.16	8.32
Xilinx Hex	7	0.28	180	52.07	8.28	16.55

The metal layers were chosen to reflect probable metal layers chosen by the FPGA vendors. SRAM cells typically will use metal layers 1-4 [67] so the lowest layer available for use by the medium length wires is metal 5. It is likely that this will be used for the shortest wires so I have modelled the R4 wires in this layer. The longer wires will want to take advantage of the high layers and the corresponding lower capacitance between them and the ground plane so I have put the hex wires on metal 7 and the C4 wires in between. FPGAs may have as many as nine metal layers, but these will be used for longer wires, power grids and shielding [] so it is unlikely that any medium-length wires will be higher than metal 7.

The wires were all minimum width for minimum power consumption, but placed with double spacing to minimise crosstalk. Given the width and height of the switchboxes it is likely that this method is employed.

The resistance and capacitance of the segments was calculated using Quickcap [3]. The tools was given the metal specifications and calculated the resistance and coupling capacitance with ground given that the wire was surrounded by two identical wires on each size with a ground plane above and below.

The specification generated by Quickcap was converted into an HSpice model and simulated with a number of different driver sizes to measure timing and approximate power. The simulations showed that for each of the wires, drivers of size $\times 5$ to $\times 16$ gave the best delay and energy-delay product results. The wires in high metal layers generally needed larger drivers. In real FPGA devices a range of drivers are used to allow the CAD tools to trade off between power and delay and to make the most of the silicon area. The experiments indicate that a reasonable approximation to real devices would be to use a $\times 12$ driver; this assumption has been backed up by private conversation. The static wires had a standard buffer chain to drive the wires. The TDM wires had the drive strength built into the latch design.

4.6.3 HSpice Models

Each individual component shown in figure 4.1 was modelled using HSpice and combined to model each individual wire type in full. An example *spice subckt* (pronounced ‘sub-circuit’) is

```
.include "MUX16.spi"
.include "R4_WIRE.spi"
.include "DRIVER.spi"

.SUBCKT STATIC_R4 MIN15 MIN14 MIN13 MIN12 MIN11
+ MIN10 MIN9 MIN8 MIN7 MIN6 MIN5 MIN4
+ MIN3 MIN2 MIN1 MIN0 S7 S6
+ S5 S4 S3 S2 S1 S0
+ DOUT

# MUX16 with data inputs MIN0-MIN15 and SRAM inputs S0-S7
XUMUX16 S0 S1 S2 S3 S4 S5 S6 S7
+MIN0 MIN1 MIN2 MIN3 MIN4 MIN5 MIN6 MIN7
+MIN8 MIN9 MIN10 MIN11 MIN12 MIN13 MIN14 MIN15 DM MUX16

#Wire driver
XUDR DM DDR DRIVER

#R4 wire with RC model
XUR4D DDR DOUT R4_WIRE

.ENDS
```

Figure 4.5: HSpice model for a full R4 wire including the 16:1 mux, wire driver, and wire model.

shown in figure 4.6 for an R4 wire and its driving circuit. The details of each component are contained in the subckt imported by the `.include` statement.

In order to measure the power consumption of the circuits test wrappers were written with suitable measurement statements. Individual power measurements can be taken if a separate supply voltage source is used for the subckt of interest.

The power measurements were taken for an ambient temperature of $25°C$ and are shown in table 4.2. Operating frequencies of 800 MHz and 200 MHz are shown for TDM and static wiring respectively. It is these power measurements which are used to perform power estimations for different NoC designs in later chapters.

```
*Four Data inputs
*Four SRAM inputs
.subckt MUX4 S0 S1 S2 S3
+D0 D1 D2 D3 D

.param invn = "1"

Mn0 D0 S0 D VSS n L="lmin" W="wmin*invn"
Mn1 D1 S1 D VSS n L="lmin" W="wmin*invn"
Mn2 D2 S2 D VSS n L="lmin" W="wmin*invn"
Mn3 D3 S3 D VSS n L="lmin" W="wmin*invn"

.ends MUX4
```

Figure 4.6: HSpice model for a 4:1 mux with four minimum width transistors.

Table 4.2: Power measurements for static and TDM wiring components. The TDM SRAM control represents the extra muxes needed by the TDM wiring to control the SRAM, shown in figure 4.1(b). The TDM control overhead is shown in figure 4.1(c).

	Operating	Power consumption at $25°C$ (μW)		
Circuit	frequency	Held high	Held low	Switching
C4 wire	200 MHz	2.38	0.26	25
C4 wire	800 MHz	0.36	0.21	138
R4 wire	200 MHz	2.37	0.26	67
R4 wire	800 MHz	0.35	0.22	116
Hex wire	200 MHz	2.39	0.26	137
Hex wire	800 MHz	0.36	0.22	195
8-bit SRAM	-	-	-	0.38
TDM SRAM control	800 MHz	-	-	1.43
TDM control overhead	800 MHz	-	-	134

4.6.4 Area Calculation

Figure 4.1 compares the TDM wiring circuitry with that of the static wiring. In order to estimate the area cost of the TDM wiring in comparison with the static wiring I laid out individual components using the custom layout tool *electric* [1] and UMC $90nm$ 1P9M2T1F technology [6]. The component of most note is the 16:1 mux depicted in figure 4.7. This depicts the style of component designed. It is much higher density than a standard cell implementation and very

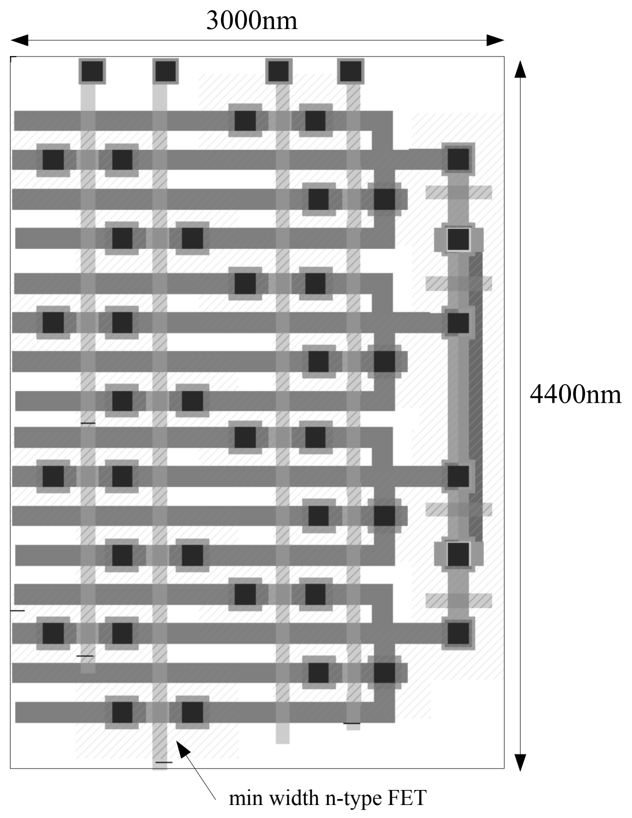


Figure 4.7: Custom layout of a 16:1 mux using 8 bits of configuration SRAM to control five 4:1 muxes.

labour intensive to produce.

Table 4.3 gives a component-by-component break down of each component used to model the static and TDM wiring and its control overhead. The area of such wires are computed by summing the component areas. The mux only uses nMOS FETs because pass transistors and so the static wire needs a level restorer to achieve a full voltage swing. This is not needed by the TDM wire because the latch is able to deal with the problem.

For example, assuming the area of each SRAM cell is $5\mu m^2$, the area of a static R4 wire would be:

level restorer $\times$ 1 =	2.0
8bit mux16 $\times$ 1 =	13.2
load $\times$ 4 =	32.0
driver $\times$ 1 =	6.5
SRAM bits $\times$ 8 =	40.0
total	$91.7\mu m^2$

The area of a TDM R4 wire with 8 contexts would be:

Table 4.3: Breakdown of area by component. The circuit diagrams for these components can be found in figures 4.1, 4.3 and 4.4.

Component	Area (μm^2)	Description
8bit_Mux16	13.2	16:1 mux constructed from five 4:1 muxes with 8-bit control as in figure 4.7
4bit_Mux16	26.3	16:1 mux constructed from fifteen 2:1 muxes with 4-bit control as in figure 4.3
Interconnect latch	20	latch driving TDM wire
Load	8	buffer driving fanout
Driver	6.5	Wire driver needed only by static wire
Level restorer	2	passive pull up need only by static wire
One-hot signal latch	12	used to generate one hot signal for TDM wiring
Pulse generator	18.28	Converts the 50% duty cycle of the clock into a pulse for the TDM wiring
SRAM cell	2.5-5	a high density SRAM cell and its read and write overhead [10]

interconnect latch × 1 =	20.0
4bit mux16 × 1 =	26.3
load × 4 =	32.0
SRAM bits × 32 =	160.0
mux8 × 8 =	55.2
total	293.5μm^2

The size of the TDM wire is much less than 8 times the size of the static wire, but it has the routing capacity of 8 wires. This is how the TDM control overhead is justified. The TDM control is only needed once per switch box. There are four configurations contexts in figure 4.1, but some architectures are likely to need eight or more. The area of the TDM wiring and its control will vary with the number of contexts.

4.6.5 Scheduling Model

In addition to the capabilities described above, the scheduling algorithm requires that the interconnect latches have a bypass function. This is to allow longer combinatorial paths to be built and reduce the timing overhead of the interconnect latch. The logic to implement such a function is likely to be simple and built into the latch itself. The latch design presented here is extremely crude. To develop this further the design of the interconnect latch would be a subject of much research. It is considered to be beyond the scope of this thesis and so the circuitry to bypass the latch is not considered here.

4.7 Summary

In this chapter I have surveyed multi-context and pipelined FPGAs. Combining the ideas of both, I have presented a model for TDM wiring derived from the implementation of FPGA wiring.

Chapter 5

Networks-on-Chip for FPGAs

5.1 Introduction

This chapter explores the implementation of Networks-on-Chip (NoCs) for FPGAs and the communication vs. computation trade-off for hard and soft designs. A simple router is proposed and hard and soft implementations are compared. In addition to this comparison, an architecture is proposed with hard Time-Division Multiplexed (TDM) wiring. The TDM wiring is inserted into the FPGA architecture alongside the statically configured wiring. Two TDM architectures are investigated: bit-level TDM wiring and bus-based TDM wiring which shares configuration SRAM between neighbouring wires.

The architectures under investigation can be summarised as follows:

- a soft router with statically configured wiring
- a hard router with statically configured wiring
- a hard router with bit-wise TDM wiring
- a hard router with bus-based TDM wiring

I configure each architecture to provide identical performance and soft-core interface. I then compare the networks for area, power and flexibility. An important part of this analysis is the calculation of the area cost of the hard network. I take into account the difference in area between the hard and soft routers when all hard routers are needed and when only a few are needed. This gives a more accurate view of the real cost of hard blocks on FPGAs, but one which is usually neglected.

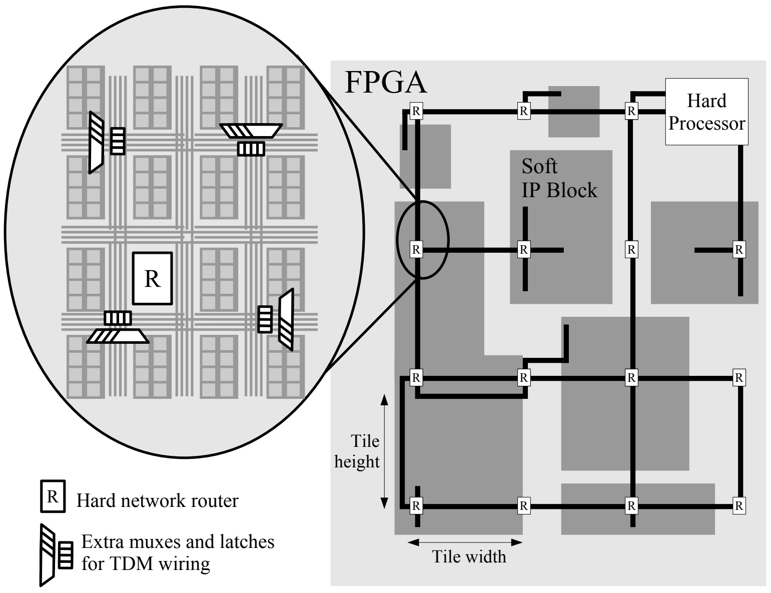


Figure 5.1: A hierarchy of static, time-multiplexed and dynamically-switched routing on a reconfigurable substrate

Table 5.1: Networks evaluated.

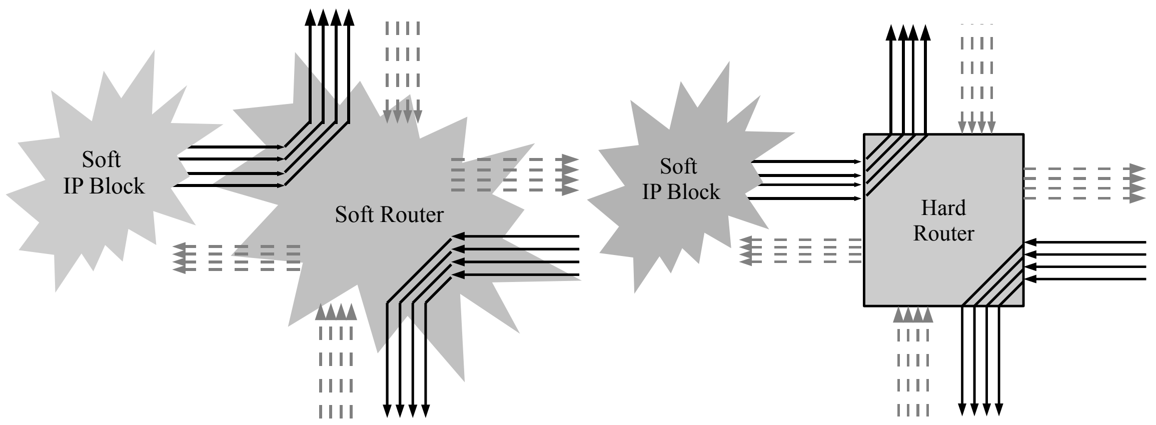
Router	Port width	Freq	Wiring
Soft StratixII	32 bit	200 MHz	Static
Soft Virtex4	32 bit	200 MHz	Static
Hard	32 bit	200 MHz	Static
Hard	8 bit	800 MHz	TDM
Hard	8 bit	800 MHz	TDM (shared configuration SRAM)

5.2 Networks under comparison

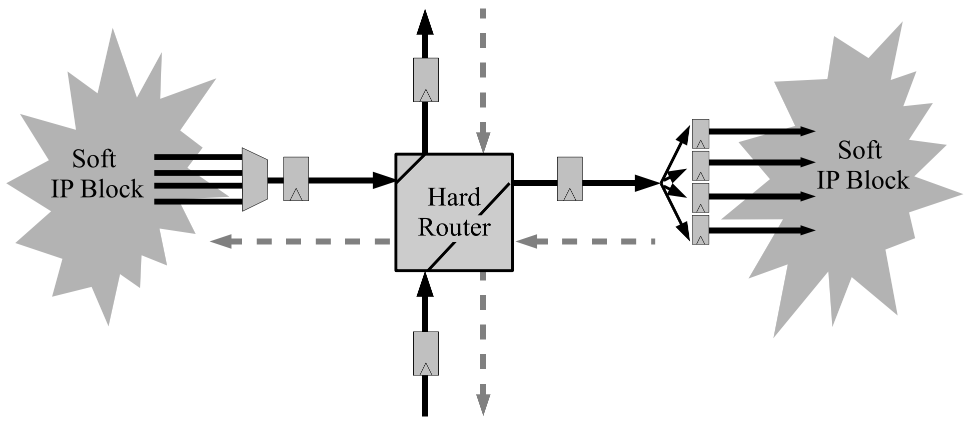
Details of the networks evaluated are listed in Table 5.1. The soft network is implemented on both Xilinx and Altera devices. The hard network is implemented with static FPGA wiring and with both types of TDM wiring. Figure 5.1 show how the routers are arranged into tiles, but that the soft cores have no such restrictions. The wiring is configurable so any topology can be implemented.

The TDM wiring transports data from highly distributed and slow soft-core IP blocks into the high-speed hard-core routers. For routers with TDM wiring the packets are broken down into n flits, where n is the number of different configurations of TDM wiring. This means that the hard routers using TDM wiring can run n times faster than those routers with static wiring and they require narrower ports.

Figure 5.2 represents the three different network architectures modelled in this study. Part(c) shows the way in which TDM wiring is used to funnel data to and from a hard router from



(a) Soft network router. (b) Hard network router with static wiring.



(c) Hard network router with TDM wiring.

Figure 5.2: Network routers and wiring configurations.

soft cores. The fully TDM wiring is used to select data from different parts of the soft core at different times and send it at a higher rate to the hard router.

5.3 Router Architecture

I have chosen to implement a circuit switched router because of the modest input buffer resources needed compared to a packet switched router [16]. Buffers are extremely area-inefficient on FPGAs because look-up table clusters contain few state holding elements for the area, and on-chip memory blocks are valuable resources, not necessarily very well placed for NoC implementation.

Circuit switched routers are poor at sharing busy resources, but on an FPGA with flexible wiring and known traffic patterns this can be overcome by the system designer. Congested areas can be avoided by keeping hop counts low and maintaining good path diversity.

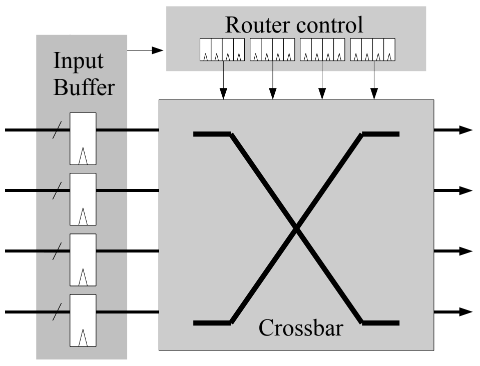


Figure 5.3: Router architecture.

Figure 5.3 shows the main components of the router. The number of ports is parameterisable in the HDL, but for this study all routers have four input ports and four output ports. To allow a custom topology there is no difference between local and inter-router ports. There is also no notion of direction so any port can be connected to any other component on the FPGA via configurable wiring. Within the router, the crossbar is fully connected so any input port can connect to any free output port.

5.3.1 Routing

A small amount of control logic is needed to support the TDM routing; in all other respects all the routers are identical. The routing is performed at the source node. This is a good choice for regular and irregular networks. For regular networks (e.g. 2D mesh) the route calculation is straightforward and can be performed efficiently by either the source or each router - there is no difference in performance. If the network is irregular then it is likely that table based routing is needed. By storing the table at the nodes the traffic can be managed differently for each node. Similar nodes with similar communication patterns can share tables.

Static Routing

The data is sent as a series of 32-bit data packets. The first packet contains the circuit addressing information. The top two bits denote the output port. Before the address packet is forwarded the top two bits are discarded and the data shifted. With four ports encoded in two bits, the circuit is able to make sixteen hops before reaching its destination. This means that such an architecture is able to support meshes of 8$\times$8 routers, but only small adaptations would be needed for much larger systems.

Local	Bit pattern received	Address	Bit pattern transmitted
Yes	`aa bb cc dd`	`aa`	`11 bb cc dd`
No	`11 bb cc dd`	`bb`	`10 cc dd 00`
No	`10 cc dd 00`	`cc`	`01 dd 00 00`
No	`01 dd 00 00`	`dd`	`00 00 00 00`
No	`00 00 00 00`	NA	`00 00 00 00`

Table 5.2: Address encoding scheme for TDM flits

TDM routing

The networks using TDM routing employ a similar addressing scheme, but each 32-bit packet is broken down into four 8-bit flits and so some extra control logic is needed to keep track of the address. The encoding scheme is summarised in table 5.2.

The router has a bit indicating whether an input port is from a local core or otherwise. If the input is local then the top two bits contain the address. If the input is not local then the top two bits contain the number of address entries left in the flit. A non-zero number indicates that the address is in the next two bits. Zeros in the top two bits indicates that this is an empty flit. The next non-empty flit to be received is treated as a local input and the top two bits taken as the address accordingly. It is in this way that the router is able to use all four flits in the address packet, but without buffering more than one at a time.

The control flow signals are synchronised with the first flit so that they operate on a per-packet bases.

Router control

The input buffer reads the header information containing the routing information. This make take more than one TDM clock cycle in the TDM design. This routing information is translated into a routing request and passed to the *router control* block. This block arbitrates between requests, computes the new configuration for the crossbar. The head has access to the crossbar as soon as the control registers are set.

5.3.2 Flow Control

The circuit is set up using a 3-bit request-acknowledge protocol. The *circuit_request* bit goes high with the address packet. The *circuit_acknowledge* bit signals that the address packet has reached its destination and the target core is ready to receive data. The data packets are sent along with a *data_valid* bit. The transfer is complete when the *circuit_request* bit goes low. This is very simple and easily implemented in soft IP. Each router is unaware of whether it is communicating with a soft IP block, a hard block or another router. This makes hybrid systems possible where there is a mix of hard and soft routers.

5.3.3 Soft Implementation

Both StratixII and Virtex4 FPGAs are used. These are implemented using TSMC 90nm GT (high performace) technology [5], allowing me to translate the configurable resource use into silicon area. Being of similar performance, but different architectures, vendor bias is eliminated from the results.

Both FPGAs have an island style architecture. This means they have clusters of basic logic elements connected via a mesh of configurable wiring. The basic logic elements are Look-Up Tables (LUTs) and register pairs. The clusters also contain carry chain logic and features for implementing functions in addition to those implementable in the LUTs.

The interconnect is arranged in a hierarchy of short, medium and long wiring and is staggered to create a truly homogeneous architecture. Short wires connect neighbouring clusters and long wires span 16 or 24 clusters. Each wire is driven by a single mux controlled by configuration SRAM. Each wire in turn drives many muxes allowing complex wiring patterns to be formed.

The medium length wires are the most commonly used and the most flexible. They have been developed to be the optimum trade-off between range and configurability. The length and fanout are not necessarily optimal and should be the subject of re-examination in the development of this archtecture for commercial markets.

The StratixII has clusters of eight 8-input LUTs. Medium length wires span four clusters with a distance of 876 μm vertically and 540 μm in the horizontal direction. These wires are called C4 and R4 wires respectively.

The Virtex4 has clusters of eight 4-input LUTs. The medium length wires span six clusters, a distance of about 720 μm in each direction. These are known as Hex wires.

The router was mapped to both the Virtex4 and StratixII using the appropriate tools from the FPGA vendors and were analysed using these tools.

5.3.4 Hard Implementation

The hard routers were placed and routed in 90nm technology using the Synopsys design flow. The silicon area results are likely to be an overestimation because commercial FPGAs use custom layout techniques. I assumed that the hard routers replace LUT resources and therefore have access to the same wiring as LUT clusters.

Table 5.3: Maximum operating frequency of each router

Router	Maximum operating frequency
StratixII	256 MHz
Virtex4	307 MHz
Hard 32-bit	977 MHz
Hard TDM 8-bit	1041 MHz

5.3.5 Operating Frequency

Table 5.3 shows the maxium operating frequency of each router. The 8-bit and 32-bit hard routers are both capable of running at 800 MHz, but the 32-bit router connects directly to soft cores and so must be run at the same clock speed as the soft cores. Surveys [70] suggest that 200 MHz is a fast, but not uncommon design speed and so presents a realistic point for comparison. The soft routers can run faster than 200 MHz, but are also limited by other soft cores. The TDM routers interface with the soft logic using TDM wiring and have none of these limitations. By running the (8-bit) TDM routers at 800 MHz and the other (32-bit) routers at 200 MHz all networks switch data at the same rate and run at a realistic clock speed for their system. In this way I present an equivalent performance comparison, representative of real systems-on-FPGAs.

5.3.6 Wire Implementation

The TDM wiring routes n different configurations, depending on the amount of extra SRAM. I have chosen n to be four because the hard routers are capable of running at four times the frequency of a fast soft-core design. The router with 8-bit TDM wiring operates four times as quickly as the 32-bit cores.

I have assumed that the wiring uses the same mux design as the static wiring which uses 8 bits of configurations SRAM to control a 16:1 mux. With only four sets of configuration SRAM the benefits of reducing the number of SRAM bits in favour of a more complex mux design will be limited.

Network topology

Any topology supported by the four port routers can be constructed. Without loss of generality I assume that the routers will be connected in a regular mesh.

Tile area calculation

The area results are given as a percentage of tile area. A tile is defined as the area contained by the bottom left hand corner of four neighbouring routers showing in figure 5.1. Each tile

Table 5.4: Router Silicon area results.

Router	Area (μm^2)	Number of Logic Clusters
Soft 32 bit StratixII	125 077	4.25 (StratixII)
Soft 32 bit Virtex4	126 000	8.75 (Virtex4)
Hard 32 bit Static	9 549	0.65 (StratixII)
Hard 8 bit TDM	5 757	0.39 (StratixII)

therefore contains one router and a rectangular area of reconfigurable fabric. I also assume that the network wiring running though each tile is the same and consists of four uni-directional channels. In this way the network will be constructed in a mesh from identical components.

The internal dimensions of the StratixII archtecture have been provided via private conversation. The area of a StratixII logic block cluster and switchbox is $218\mu m \times 135\mu m = 29\ 430\mu m^2$. The most common wires span four clusters in each direction. A tile with just one of these wires on each side would have an area of $47\ 0880\mu m^2$.

The area of a Virtex4 logic block cluster and switchbox is estimated to be $120\mu m \times 120\mu m = 14\ 400\mu m^2$. The most common wires span six clusters in each direction. A tile with just one of these wires on each side would have an area of $518\ 400\mu m^2$.

For the rest of this chapter the tile size is measured in terms of the number of wire segments spanning each tile side.

5.4 Area Results

The number of LUT clusters used by the soft routers is translated into silicon area and the hard routers are given as a proportion of a StratixII LUT cluster. Table 5.4 summarises these figures. Note that the Virtex4 cluster has fewer LUTs and is smaller than the StratixII logic cluster. The area of each cluster and its corresponding switchbox is included for the soft designs to take into account the internal wiring area. The internal wiring area is already included in the area figures for the hard routers and so the number of logic clusters is taken as a proportion of the logic cluster area with no switchbox.

5.4.1 Network Area Comparison

The ratio of router area to wiring area varies with the spacing of the routers. The networks with TDM wiring have been designed so that there is a minimum of 16 blocks of four TDM wires which are evenly distributed. This amounts to $16 \times 4 = 64$ TDM wires. This allows $8 \times 4 = 32$ input wires and 32 output wires to be time-division mulitplexed so each of the four 8-bit ports can use TDM wiring. This is more than sufficient because only wires passing to and from soft cores will need to do so.

There is a TDM block every two wire lengths to allow regular pipelining; any more would lead to excessive ammounts of TDM wiring, but any less would lead to too few TDM wires. This means that larger tiles have more than 16 blocks of four TDM wires. Figure 5.4 shows the arrangement The optimum arrangement of wires is an area of further investigation.

All TDM wiring in the tile is counted towards the total area as well as extra static wiring used. This leads to an over estimation in TDM wiring costs because in a real system TDM wiring not used in the network can be used for other purposes.

Figure 5.5 shows the percentage of tile area given over to the network for different tile sizes. The hard networks have been modelled with both StratixII and Virtex4. The differences between the hard NoC implementations on the Virtex4 and StratixII were very small and so only the hard NoCs using Virtex4 wiring are shown. The soft NoCs are shown for both architectures.

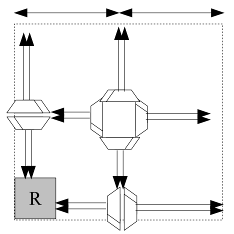
At small tile sizes the router size uses a larger proportion of the network area and the hard routers are clearly smaller than the soft routers. The hard 8-bit router with TDM wiring using shared configuration is the smallest. The soft implementations only start to catch up with the hard routers at low granularity.

Even at very fine network granularity the soft and hard routers fulfil the area limits I imposed on the system of 1% and 10% for hard and soft networks respectively. With tiles just 4 wire segments (around 2880 μm) across both hard and soft routers are very small at less than 0.5% and 5% of tile (and therefore chip) area. This means that there is great scope for implementing more complex circuit switched routers than those presented here.

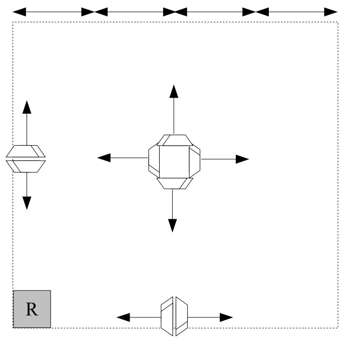
5.4.2 Area Cost of Unused Hard Routers

Consider a multi-processor system implemented on an FPGA in which a router is placed every 4 wire segments. This gives a tile size of $16 \times 16 = 256$ LUT clusters. This is twice the size of a NiosII [2] soft processor and so allows for the processor and some custom logic to be implemented on each tile. In this scenario, the hard NoC with TDM wiring and shared configuration is the obvious choice: it is by far the smallest and makes up for flexibility by high availability.

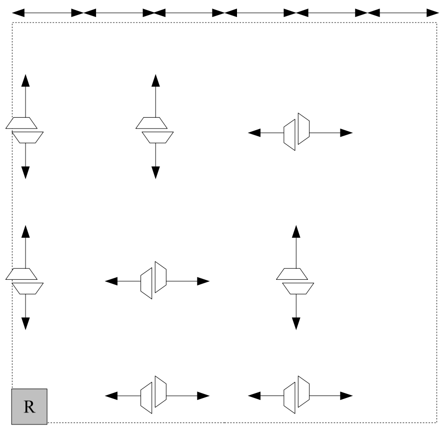
The results in figure 5.5 assume 100% network use and ideal core and router spacing across the device. In reality the core sizes will vary and the router spacing will not be optimal. Large cores could lead to an over population of routers. For example, if only 1 in every 16 routers was used, the relative network areas would be very different. The area of such a senario is shown in figure 5.6. The unused wiring can be used for other purposes, but the large non-configurable area of the static router means that the 16 hard routers consume more area than the single soft router



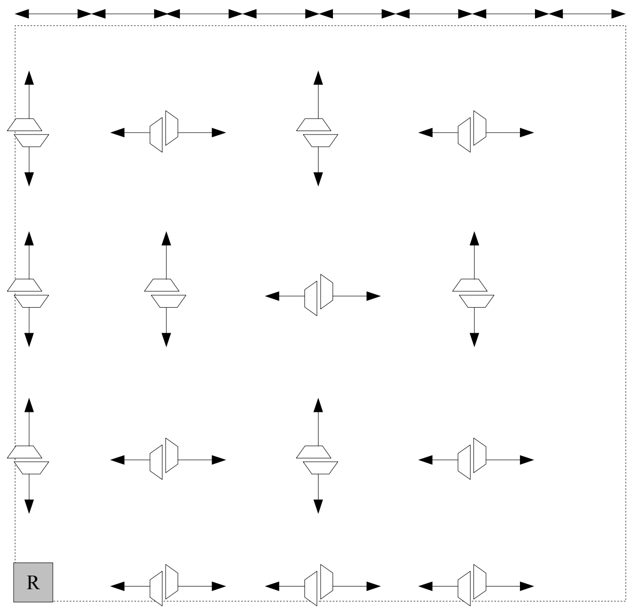
(a) Network tile two wire segments wide with 16 blocks of four TDM wires.



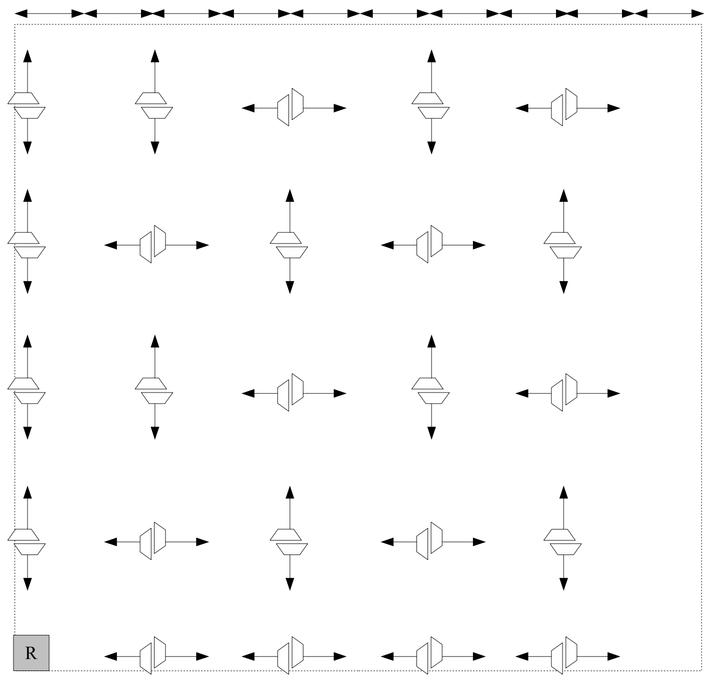
(b) Network tile four wire segments wide with 16 blocks of four TDM wires.



(c) Network tile six wire segments wide with 16 blocks of four TDM wires.



(d) Network tile eight wire segments wide with 30 blocks of four TDM wires.



(e) Network tile ten wire segments wide with 48 blocks of four TDM wires.

Figure 5.4: Diagrams showing the placement of TDM wiring blocks in networks with different granularities of routers.

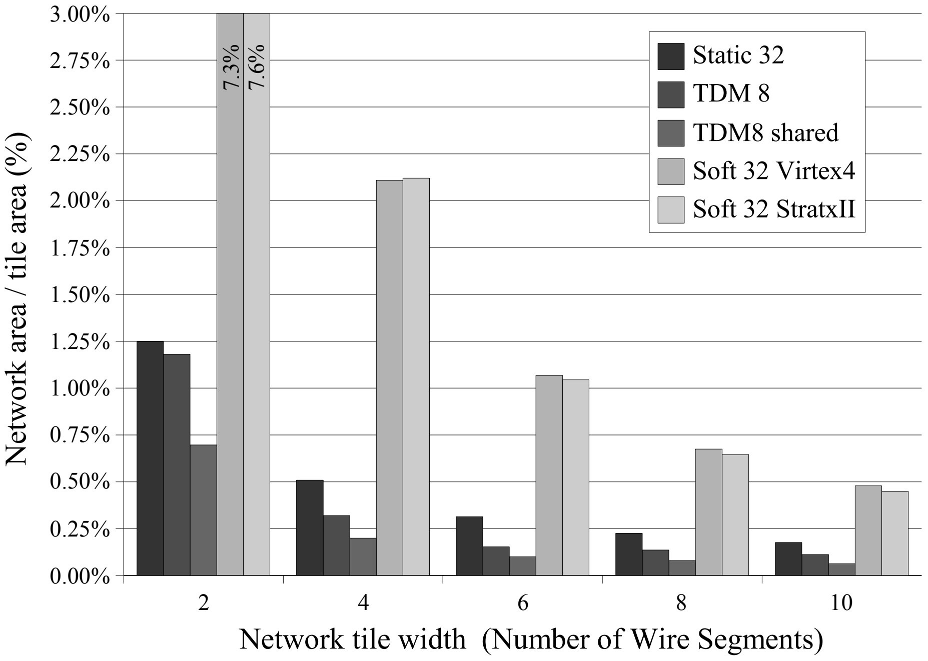


Figure 5.5: Network node area as percentage of tile area.

needed. Where the TDM wiring has allowed a smaller hard router to be implemented, this cost is reduced and the hard router with TDM wiring is still better than the soft router.

5.5 Wiring Area vs. Router Area

The graph in figure 5.7 shows the ratio of wiring area to router area. The networks assumed tiles of 4 wire segments across. The wire to router area ratio is shown for a network in which all routers are used. The proportion is also shown for an ASIC router. This was calculated by using the static 32-bit router and combining it with dedicated non-configurable wiring. The dedicated non-configurable wiring area was computed by summing only the buffering area used for the reconfigurable wiring.

There are two ways of looking at figure 5.7:

- ASIC NoC designers will notice a similar trade-off between computation and communication in the soft routers as with ASIC designs [15]. They may also criticize the low amount (20%) of resources given over to computation in the hard routers compared to the ASIC network.
- FPGA designers will notice the high area proportion (20%) given over to non-configurable resources. They may also note that at low network use (eg only 1 in every 16 routers used) the proportion of non-configurable router area will be much higher.

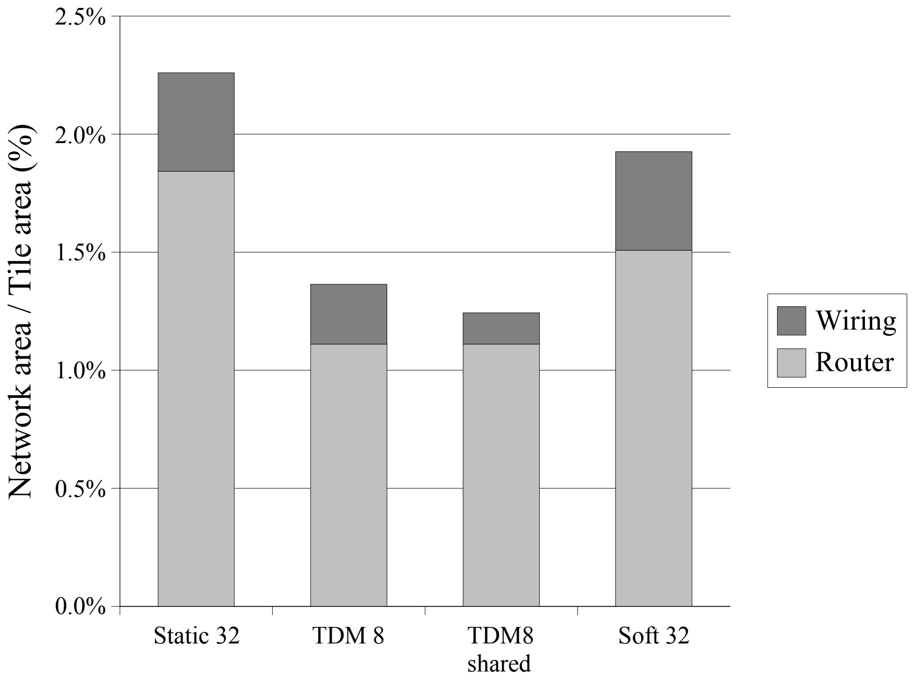


Figure 5.6: Network area for 16 tile network where each tile is 4 wire segments wide and only 1 of the 16 routers is used.

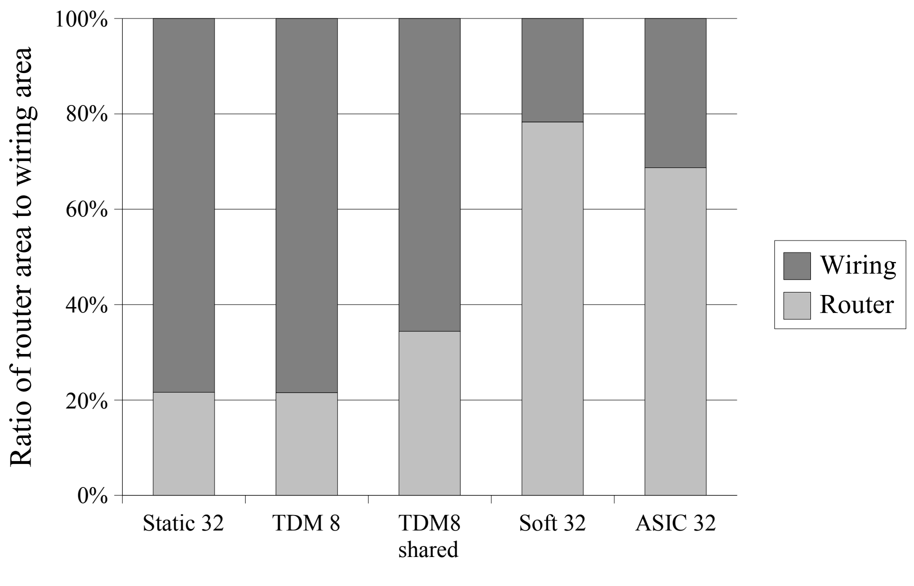


Figure 5.7: Ratio of wiring area to router area for a network with tile widths of 4 wire segments.

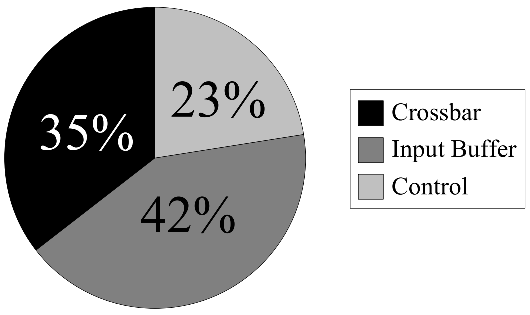


Figure 5.8: Proportion of combinatorial LUTs used in soft router components on the Stratix II.

This understanding of the differences in designing for FPGAs rather than ASICs is critical to the understanding of the implications of the TDM wiring.

5.5.1 Computation vs. Communication

Figure 5.7 also highlights the difference in the computation-communication trade-off. For ASICs this trade-off has been well examined to offer the best compromise between large routers and costly connections. At first glance the soft router seems to be closer to the ASIC trade-off. This counts in favour of the soft router; the larger overall area is caused by the flexibility of implementing the the network in the configurable fabric.

The hard routers have a large proportion of wiring to router area. This is assuming 100% network use. If the network use is reduced, the ratio of wiring to router area will be reduced as routers are unconnected and unused. This would make the relative router cost much higher in comparison with the amount of wiring.

Another way of looking at the ratio of wiring to router area is to consider that the wiring area also contains the configuration logic which enables the custom topology. Considering the wiring alone would give the same trade-offs as the ASIC network. The additional wiring area could be counted a reconfiguration logic, a cost justified by the design of the FPGA.

5.5.2 Soft Router Area Breakdown

The soft router is an attractive choice for those clients unlikely to need a high granularity NoC in every application of the device. In order to investigate the possibility of improvements to the soft router I extracted the breakdown of the router area into its modular components, shown in figure 5.8.

Broadly speaking, the three components have comparable areas to those in an ASIC design [15]; no one component is implimented particularly well or particulary poorly on the FPGA so there would be little benefit to increasing the complexity (and therefore area) of one to decrease the area of the other.

5.5.3 Hard vs. Soft Area

The router and wiring areas both increase with the conversion from hard to reconfigurable implementations. However, they do not scale by the same amount and this affects the way in which the logic and wiring trade-off in an FPGA NoC.

FPGAs need a high redundancy in the routing in order to implement designs. The logic block use will typically be much higher than the wiring use, but the ratio of wiring to logic has been carefully calibrated to make the best use of the silicon area for designs in the past. This means that the difference in area between dedicated ASIC-style wiring and FPGA configurable wiring is low, but the difference in logic area between hard and soft routers is high. This arrises from the difference in the way the flexibility is implimented for logic and wiring. It is possible for a k-input LUT ot impliment and k-input function, so although they are large, compared to a dedicated logic gate, it is possible to use all the LUTs in an FPGA. Wiring flexibilty is provided in part by a high degree of configurable conectivity, and in part by an over provision of wires [22].

Wiring Area

The silicon cost of a 2mm wire designed for an ASIC could be implemented using a $\times 50$ driver with three $\times 50$ repeaters [63]. This gives an area of approximately $72\mu m^2$ in 90nm. This is approximately the same length as four StratixII R4 wires spanning a total of 16 logic clusters.

By custom layout, I estimated the area of four StratixII R4 wires to be 287 to $367\mu m^2$ using the model described in Section 4.6, assuming a single SRAM cell is between 2.5 and $5\mu m^2$.

The combined horizontal and vertical channel width on the StratixII is 496 including wires joining neighbouring logic clusters. The mean cost per wire spanning 16 clusters can be calculated as:

$$\frac{\text{Switchbox area} \times 16}{496} = \frac{14715 \times 16}{496} = 475\mu m^2 \tag{5.1}$$

This value is necessarily too large because there are components for clock distribution, fault tolerance and testing that I cannot estimate. This mean value is therefore an upper bound for the area cost of one wire. It supports my estimations by being larger, but fractionally so.

These calculations show that the area cost of transporting a single bit 2mm increases by approximately 4.0 to 5.1 times with a strict upper bound of 6.6 times.

Router Area

Table 5.4 lists the router area in terms of silicon area and the equivalent number of logic clusters. On the StratixII the 32-bit soft router used 4.25 clusters. The hard router of the same design used only 0.65 clusters by area equivalence. The reconfigurable router is therefore larger by 13.1 times.

Trade-off

The area increase of the logic is at least twice that of the wiring. The efficiency of routing wires in parallel mean that the unused redundant wiring is still very much usable and accessible in the configurable architecture. This means that away from a hard router there is more than enough wiring to support a network. The channels can support up to 475 bits, but a 32 bit mesh network needs only 256. The problem with the configurable wiring is the bottleneck at the hard router; the interface between a logic cluster and a switchbox is not 256 bits wide.

What is a good ratio of logic to wiring in an ASIC design will lead to a poor trade-off in a soft-core NoC design. The high cost of configurable routers means that for soft NoCs there is scope for using more wiring to reduce the computation cost.

5.6 Power Results

Router power was estimated by setting up two data circuits though the router. Random data packets were generated and sent over the network. As with with the area results there are two input ports and two output ports, each running the length of the tile. The system power was then estimated by combining the router power consumption with the power models of the interconnect. It is assumed that the network tile is four segments across, an area of 8.3 mm^2 on the Virtex4 and 7.5 mm^2 on the StratixII. The TDM architecture routes the channels using equal amounts of static and TDM wiring.

The soft router power consumption was calculated by assuming that the power consumption consisted equally of leakage and dynamic power. This is reasonable and in line with published trends [2]. The average leakage power of a Xilinx logic cluster is $4.2\mu W$ [80]. The soft router on the Virtex 4 used 8.75 clusters and the soft router on the StratixII used the area equivalent of 8.81 Virtex logic clusters. This was used to estimate the average power consumption of the soft routers. This could be much lower than peak power consumption and so my power estimations of the soft network are likely to be too low.

The power consumption of the hard routers was measured using VCS and PrimePower [4]. I assumed a transition rate of 50%; that is to say that in the imput data to the network there was a 50% probability of a bit flipping between one flit and the next in the packet. I used a behavioural test bench wrapper to generate the input data for the router. I used VCS to perform the timing simulation so that each net could be annotated with transistion and parasitic data. I then used PrimePower to perform a power simulation using the annotated netlists. The routers had no

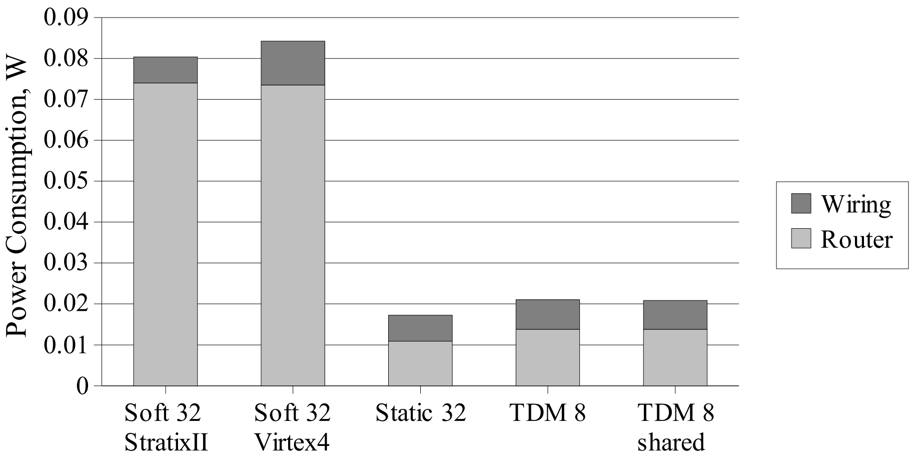


Figure 5.9: Power consumption for hard and soft routers, with and without TDM wiring.

Table 5.5: Power consumption of hard routers.

Router	Power consumption (mW)			
	Cell Leakage Power	Cell Dynamic Power	Net Switching Power	Total
Static 32-bit	0.105	8.1	2.69	10.9
TDM 8-bit	0.108	10.3	3.45	13.8

clock gating capabilities and so their power consumption is likely to be lower in reality. The power consumption results for the hard routers can be found in table 5.5.

The wiring power consumption was estimated using transistor-level HSPICE models as decibed in detail in section 4.6.3. Low-to-high and high-to-low transitions were measured and the average power consumption measured for each wire. The power used to clock the TDM wiring was not taken into account as it is hard to quantify and likely to be insignificant compared to the power consumption of the router clock.

Figure 5.9 shows the results broken down into router power and wiring power. The soft NoC on the Virtex4 consumes more than four times the power of the Hard NoC with TDM wiring. This is not as high as indicated in other hard-soft comparisons [52], but our assumptions are quite conservative and the gap may well be higher than estimated.

The TDM network consumes more power than the hard static network, but the difference is small compared to the difference between the hard and soft designs. If a hard network is made practical through the use of TDM wiring then the overall power savings will be great. The small increase for TDM wiring will be worth paying.

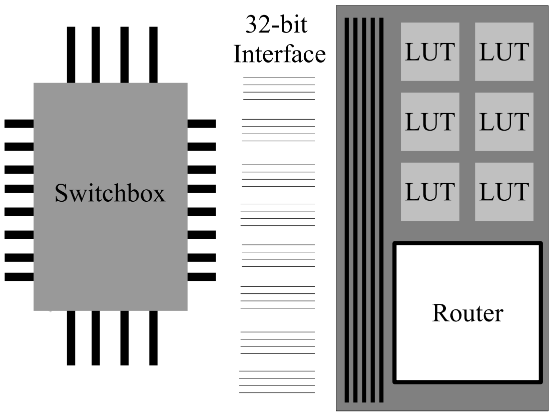


Figure 5.10: Position of a hard router with four ports sharing the logic cluster interface.

5.7 Router Interface

The regularity of the FPGA is what allows for such high yield and quick adoption of new technologies. Any hard block must interface with the FPGA using the existing wiring in order to maintain that regularity.

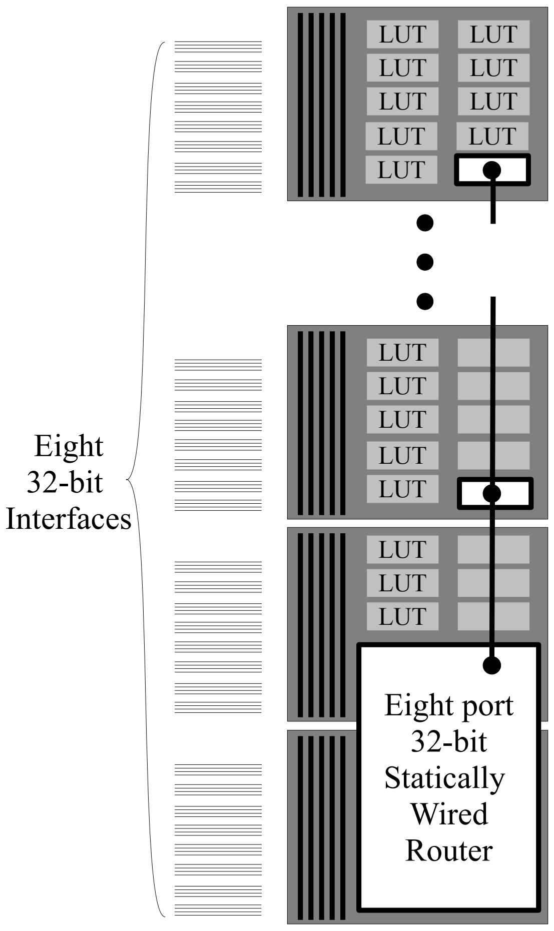
In order to preserve the homogeneity of the FPGA the router must replace logic elements within a cluster and share the common interface. To do otherwise would create a hole in the fabric with irregular wiring. This technique is know as a shadow clustering [49] and is shown in figure 5.10. The remaining LUTs in the cluster can only be used when the router is not.

The advantage of this design is that the router causes least disruption to the regular architecture and puts the least amount of strain on the ECAD flow. The disadvantage is the extremely limiting interface. The logic cluster typically has a 32-bit interface to the chip-wide routing architecture [7]. This is sufficient for the TDM router which has four 8-bit ports, but the 32-bit router needs a 32-bit interface for every port. This means that a 32-bit hard router will need the same connectivity as four logic clusters; this is nearly half the size of the soft router. By using the connectivity of neighbouring logic clusters, the router must disable the logic and increase the cost of a used router.

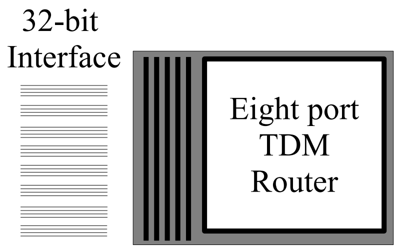
5.7.1 Number of Router Ports

The router design in this chapter has four ports. This is insufficient for the majority of popular topologies because they require four or more connections to neighbouring switches, leaving no ports to connect to local computational cores. NoCs with four-ported routers are only able to impliment more exotic topologies with more than one router per computational core.

Eight router ports allows for four ports in a regular topology and two local connections. More local ports mean that fewer routers are needed to implement an efficient network. The two remaining ports could be used to bypass the regular topology. Increasing the number of ports on



(a) Statically wired router.



(b) TDM router.

Figure 5.11: Placement of hard routers with eight ports.

the routers is necessary to exploit the reconfigurability of the wiring and implement a network with low congestion and low hop count, but it exacerbates the problem with the interface.

Another problem with increasing the number of ports is the increase to the size of the hard router. The router will necessary increase by more than 100% in size. The statically wired 32-bit router with eight ports would have to replace two logic clusters in area and use eight cluster interfaces. This is shown in figure 5.11.

The four port TDM router uses less than 40% of the logic cluster area in the StratixII architecture so it possible to increase the number of ports and still keep the router within one cluster. This would reduce the cost of unused logic clusters when the interface is monopolised by the router. The extra ports give more points of access to the NOC and more flexibility and so fewer would be needed. This reduces the number of unused routers. The cost of a used router remains the same because a used router still represents one logic cluster.

The wiring would have to be time multiplexed more than four times though in order to maintain the 32-bit interface with eight ports. Higher rate TDM wiring is explored in later chapters.

5.8 Scalability

A small system with eight cores would be trivial to design with a hard TDM routing architecture. One router with eight ports would suffice. The wiring would be run faster than the router and very low latency data transport with no congestion would be possible.

A larger system with 64 cores may want to have as many as 32 8-ported routers. Each router would have two local connections on average leaving four ports for a mesh, and two more for congestion reduction. In general, routers with eight ports would be able to support many custom topologies with six router-router ports and two local ports. This means that only one router is needed for every two cores. Six ports is more than sufficient for most popular topologies so it may be possible that even fewer routers would be needed. It would be difficult to impliment a network with fewer than four router-router connections so there should be at least one 8-ported router for every four cores.

As systems start to become very large a different approach might be necessary. A system with 1024 cores would have a 32×32 mesh and a longest hop count of 64. At this point circuit switching starts to become impractical for long hops as high latency circuits may occupy many network resources for a long time even for small amounts of data.

Soft packet switching routers could be attached to the hard circuit switched network at regular intervals to handle long hop communications. Although large, these packet switching routers would be few in number compared to the number of cores in the system and their cost would be proportionally acceptable. The circuit switched network would transport the packet to a packet router. The router would then dynamically route the packet to its final destination either though dedicated channels or by using circuit switched dynamic links between neighbouring packet routers. In this way complex custom networks could be built from parameterisable routers with very low design and implementation costs.

5.9 Summary

This chapter has provided a fresh look at the trade-offs between hard and soft IP blocks on FPGAs and applied this technique to the design of Networks-on-Chip (NoCs). I have shown that for low network usage, soft networks can be more efficient than hard networks.

In order to reduce the cost of unused hard routers I introduced some Time-Division Multiplexed (TDM) wiring onto the FPGA. This allowed the router to be reduced in size and run at a high frequency. The smaller router was able to provide the same performance as the larger router with static wiring, but at a fraction of the silicon area. The area cost of the network was moved into the more flexible, reusable TDM wiring components. Even when the router is not used, these TDM wiring components can be, so they do not add to the cost of unused network components.

If the system requires efficient utilisation of resources then an FPGA with a hard NoC and TDM wiring would be an excellent design choice.

The TDM wiring comes at a small power cost, but if it makes a hard network more practical than a soft network then the overall power savings will be great. The TDM routers are so small they can be placed in abundance. Custom topology and large number of ports can reduce congestion.

The soft routers allow for a more sophisticated network design to be implemented and changed, but this is unlikely to be of more benefit than using the abundant wiring available in FPGAs to reduce congestion.

Chapter 6

Scheduling of Time-Division Multiplexed Wiring

6.1 Introduction

The approach taken to evaluate an FPGA with Time-Division Multiplexed (TDM) was to follow a conventional flow to place and route the benchmark circuits and then schedule the routed nets onto shared TDM wiring. This did not give a fully developed commercial solution, but it would allow me to assess the viability of TDM wiring on FPGAs. In this chapter I present the algorithm behind the scheduler used to evaluate my TDM architecture.

6.2 ECAD flow

There were three ECAD flows which could be used to place and route the benchmarks. These were the academic flow (VPR) [19], the Altera tools [2] or the Xilinx tools [7]. A comparison between academic tools highlights the sensitivity of experimental results on the tools chosen [84]. All existing ECAD flows map designs to statically wired FPGAs and so all must be expected to give sub-optimal place-and-route results for TDM wiring. This is because the clustering and timing optimisations performed by modern tools will be assuming a very different timing model for the static wiring than what is needed for the TDM wiring. This cannot be avoided so is taken into account at a later stage.

In the interests of maintaining modern, commercially applicable models, I chose not to use VPR. The tool has now undergone some major updates, but when I was writing the scheduler, VPR was unable to model modern heterogeneous FPGAs with embedded hard blocks and complex logic elements.

The synthesis and technology mapping stages for VPR were also limited to older tools developed for academic use [72, 30]. These tools have shown innovative techniques which have driven the development of FPGA ECAD tools, but this will have allowed the commercial tools to develop those ideas further. Xilinx and Altera both have the resources to benchmark their tools against the academic flow and learn from them.

The Xilinx and Altera tools both presented a challenge. Detailed timing information for each wire segment in the design with precise connectivity data was needed by the scheduling algorithm, but neither vendor make public the post-place-and-route timing information needed to schedule signals on individual wire segments. I was able to get access to the detailed timing information within Altera's QuartusII flow through the generosity of their research department and this made the decision for me.

6.3 Scheduling Algorithm

There are many resource-constrained scheduling problems and many more algorithms presented as solutions. The algorithm in this disseration bears much resemblence to the problem of *register colouring* [25] due to the high level of dependedncy between the choosen scheduling of one wire on the constraints of another. This method is a form of *list scheduling* [62] where a list of possible candidates for a resource is made and one selected.

6.3.1 Algorithm Details

The scheduler takes a graph of each benchmark, as placed and routed onto a conventional Stratix FPGA, and re-maps it onto a TDM FPGA. The details of the algorithm are depicted in figure 6.1. The graph is in the form of a list of every combinatorial path in the benchmark. After parsing the lists, the scheduler constructs the graph as a set of linked wire objects. The scheduler serializes neighbouring signals so that they may share a single wire. The Stratix [57] device was chosen for the study because it is a good compromise between modern high-performance and modelling complexity.

The benchmark graph is constructed from a custom timing and routing file generated by a TCL script. The TCL script calls Quartus II executables to place and route the design, perform timing analysis and then output tables of placement, routing and timing information for every wire mapped onto the FPGA. The scheduler is architecture independent and can be configured to investigate any homogeneous FPGA architecture provided there is sufficient architectural and timing data available.

Wires in the benchmark graph are scheduled by assigning a first and last time slot to each one. These time slots represent the interconnect clock cycles in which this logical wire will be allocated to a physical TDM wire on the device. A logical wire may be allocated to a physical TDM wire for multiple consecutive time slots. Assignment of a time slot to a virtual wire represents allocation to a physical resource for that duration. Each virtual wire must remain assigned to a physical wire until all destination latches in the signal fanout have received the data. TDM wires may drive long combinatorial paths and so this delay may vary significantly between signals. In order to reduce the length of these combinatorial paths and promote the sharing of TDM wiring, additional latches are added to the inputs of the Look-Up Tables (LUTs).

In the first 'naive' scheduling stage it is assumed that there are an infinite number of wires and that each wire in the graph will be assigned to a physical wire on the FPGA for its required time slots. The required time slots are determined on a "as soon as possible" basis using the Quartus II worst-case delay model. This is very similar to many forms of *force-directed* scheduling [64], which has been combined with list scheduling in the past to solve similar problems [81].

The second scheduling stage maps the signals in the graph onto physical wires on the device. To do this the scheduler resolves conflicts caused by too many signals requesting the same time slots in the same channel. It is assumed that the switch boxes are internally totally connected; for conventional FPGAs this is a vast over simplification. It has long been known that reducing the population of connections within switch boxes is highly effective [35, 60], however TDM wiring allows us to reduce the number of wires and so reduce the cost of internally totally connected switches.

After the scheduler has attempted to schedule the benchmark the vector constraining the maximum number of each type of wire is varied and the schedule re-run. If the schedule is successful then the number of wires in decreased. If it is unsuccessful then the number of wires will increase. This increase will only be performed four times. The schedule is deemed to have failed if the scheduler has tried to remove conflicts one hundred times without success. These number were picked after a number of experiments trading off execution time against quality of results.

6.4 Architectural Parameters

The following parameters can be varied:

Number of Time Slots This is the ratio of design clock frequency to interconnect clock frequency and is determined by the number of configuration bits at each configurable switch. A time slot represents a single interconnect clock cycle and therefore the number of time slots is the same as the number of configuration bits.

Length of Time Slot Combined with the number of time slots, this determines the latency of the design. The length of the time slot can be varied from design to design. The minimum is determined by the maximum frequency of the interconnect clock. I have assumed that the

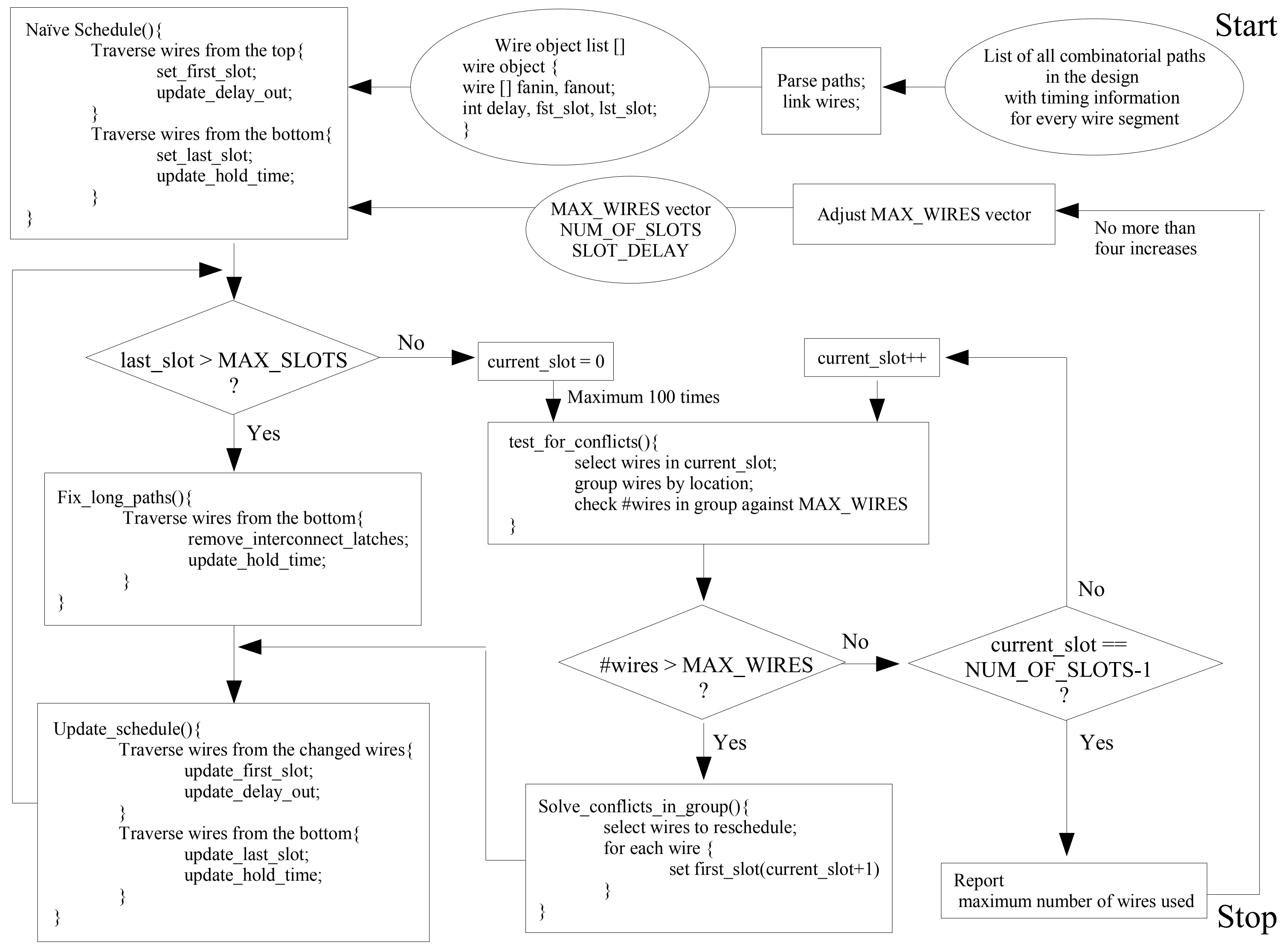

Start
List of all combinatorial paths in the design with timing information for every wire segment
Parse paths; link wires;
Wire object list [] wire object { wire [] fanin, fanout; int delay, fst_slot, lst_slot; }
Naïve Schedule(){ Traverse wires from the top{ set_first_slot; update_delay_out; } Traverse wires from the bottom{ set_last_slot; update_hold_time; } }
MAX_WIRES vector NUM_OF_SLOTS SLOT_DELAY
Adjust MAX_WIRES vector
No more than four increases
last_slot > MAX_SLOTS ?
No
Yes
current_slot = 0
Maximum 100 times
current_slot++
test_for_conflicts(){ select wires in current_slot; group wires by location; check #wires in group against MAX_WIRES }
Fix_long_paths(){ Traverse wires from the bottom{ remove_interconnect_latches; update_hold_time; } }
#wires > MAX_WIRES ?
No
Yes
current_slot == NUM_OF_SLOTS-1 ?
No
Yes
Update_schedule(){ Traverse wires from the changed wires{ update_first_slot; update_delay_out; } Traverse wires from the bottom{ update_last_slot; update_hold_time; } }
Solve_conflicts_in_group(){ select wires to reschedule; for each wire { set first_slot(current_slot+1) } }
Report maximum number of wires used
Stop

Figure 6.1: Flow diagram showing the operation of the scheduling algorithm. The MAX_WIRES vector, NUM_OF_SLOTS, and SLOT_DELAY are input parameters. *Top* and *bottom wires* are those driven by or driving a design latch repectivly.

interconnect clock can support up to 2 GHz, which limits the time slot length to 500ps. Many designs perform better with longer time slots. It is worth noting that it is unlikely that the router can be designed to run at 2 GHz so for some designs the interconnect would have to run faster than the network routers.

Number of Wires per Switch Box The shared wires are differentiated by their orientation (row or column), by their direction, and by the number of logic clusters they span. The Stratix has six global wire types with many in each channel so there must be a minimum of twelve shared wires at each switch box to maintain wire counts in each direction. The scheduler is unable to move signals between channels or wire types. The ratio of each wire type to another is heavily restricted by the Stratix architecture and the QuartusII placement and routing tool.

A wire contributes to the wire count of a switch box if that switch box contains the driver for that wire. A wire may span multiple switch boxes, but it is only counted once.

If the scheduler is unable to schedule the circuit on the requested number of wires then the scheduler returns a *fail* flag and the number of wires is increased before starting over.

6.5 The Critical Path

The critical path is determined by the number and length of time slots. The length can be varied between scheduling attempts, but must remain fixed for the duration of the algorithm execution. Differences in the latency of a signal and the next available latch time causes increased delay in the critical path of each benchmark. To some extent this is expected and forms part of this investigation. However, the critical path extension can increase to the point where the critical path takes more time slots to propagate than there are available. For this reason, latches can be bypassed to allow longer combinatorial paths. This reduces the increase to the critical path, but makes efficient sharing of TDM wires more difficult.

This technique is similar to *operation chaining*, where dependent operations are always scheduled together on a machine inorder to improve performance. By removing the interconnect latches, the un-latched wires have to be scheduled immediatly after the driving wire.

6.5.1 Trading-off Between Delay and Wire Count

Consider a signal passing though three wire segments, the delay of each coming to 0.6 of a time slot. The combinatorial delay is 1.8 time slots and the scheduler has two choices. These choices are depicted in Figure 6.2 and are as follows:

1. The scheduler can latch at every segment. This only uses a single time slot on each segment, but increases the path delay to 3.0 time slots.

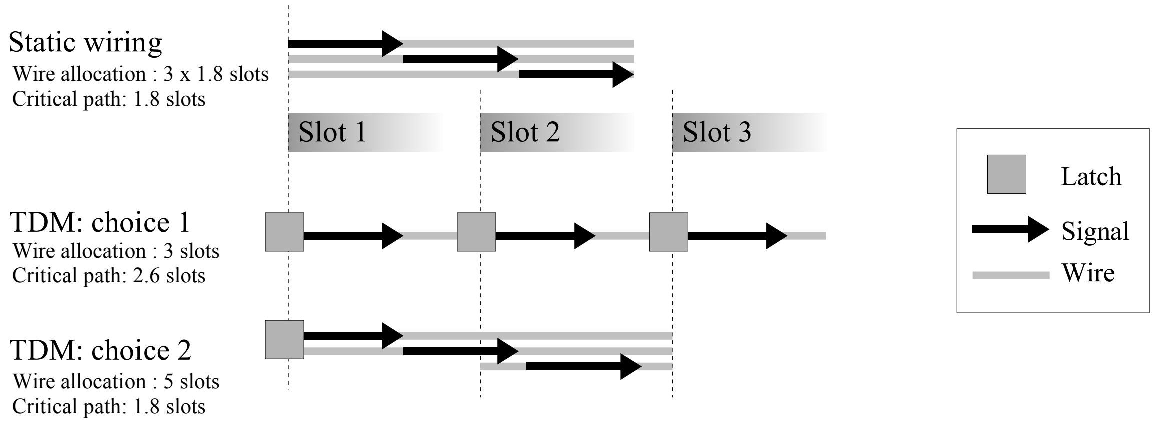


Figure 6.2: Comparing static and TDM scheduling choices. Wire allocation is in grey. The datapath is in black with arrows showing the passage of time.

2. Alternatively the scheduler can latch only on the first segment. This reduces the delay to 2.0 time slots, but the first two segments in the signal require allocation to a wire for two time slots.

Choice one has a longer path delay, but the total wire uses only one time slot per wire. Choice two has a shorter path delay, but uses a total of five wire time slots. The scheduler must trade off between critical path delay and wire count by making choices such as these.

6.6 Algorithm Performance

The scheduling algorithm was developed to demonstrate the potential of TDM wiring. It is not fully optimised for performance. Ideally one would start from the design source at the RTL level and rewrite the ECAD tool chain to make full use of the shared wiring. The scheduler has been developed to demonstrate that designs can be mapped to FPGAs with TDM wiring and that this leads to a reduction in the number of wires. Therefore, when looking at the results section it is necessary to bear in mind that the reduction in the number of wires required is itself sub-optimal.

In solving conflicts the scheduler often introduces more conflicts elsewhere. These must be removed, possibly at the expense of more conflicts. The algorithm iterates through the benchmark graph solving conflicts until an iteration limit has been hit or all conflicts have been resolved. If the iteration limit is hit then the number of wires is increased and the algorithm tries again.

It is probable that the scheduler sometimes iterates between poor scheduling choices, unable to break the cycle and make a better choice. An intelligent algorithm with choice history would give better results in less time, but would require longer to develop.

Another flaw in the algorithm is the need to update all subsequent paths and schedules following a conflict resolution. A more efficient implementation would solve all conflicts for a given time slot and only update the schedules as far as they are needed to test and solve for the next time slot. This would save considerably in the checking and redoing of scheduling information. This inefficiency of the algorithm was introduced in order to allow fine-grain testing and verification of intermediate results. The correctness of the results is valued above performance and so the scheduler was written in order to maintain a valid schedule state with deterministic results.

6.7 Summary

The scheduler maps benchmark circuits to an FPGA with TDM wiring. It aims to minimise the wire count as far as possible, while limiting the increase to the critical path. It is able to solve scheduling conflicts by buffering signals on earlier wire segments, but is unable to re-route wires or re-place logic.

Chapter 7

Scheduling Results

7.1 Introduction

I now present the results of scheduling benchmark circuits onto a Stratix FPGA with all of its wiring replaced by Time-Division Multiplexed (TDM) wiring.

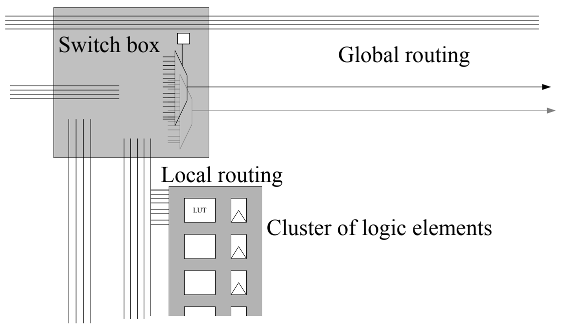
7.2 FPGA Architecture

The scheduler takes designs routed by Altera's Quartus II so the benchmarks were mapped to an Altera device. A great deal of time had to be spent building an accurate model of the device and so the simplest design that would still yield meaningful results was chosen. In order for the results to be widely applicable, the Stratix [57] was chosen as a starting point for the scheduling experiment. The Stratix architecture is a modern high-performance FPGA, but simple enough architecture to model accurately. Figure 7.1 shows the changes made to the architecture to test TDM wiring.

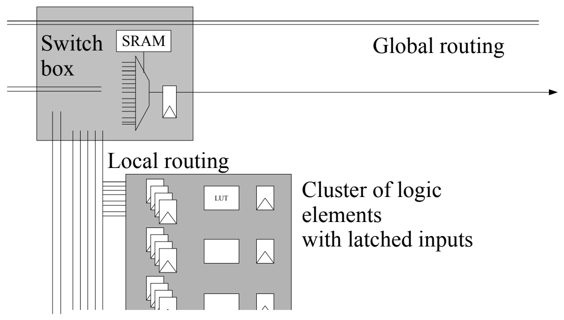
The Stratix II [56] is a higher performance FPGA that is built on a more modern $90nm$ process. The scheduling algorithms are tested on the Stratix, but silicon area results are calculated for the Stratix II. The Stratix II differs from the Stratix sufficiently to make it worth scheduling for the simpler device, but not so much the results are not applicable to both.

7.3 Benchmarks

In the past the MCNC [85] benchmarks have been popular, but they no longer represent the larger, more complex cores used on FPGAs today. Some work [47] has been done to automatically generate benchmarks, but it is no substitute for real circuits.



(a) Stratix FPGA with static wiring.



(b) Stratix FPGA with Time-Division Multiplexed Wiring.

Figure 7.1: Stratix FPGA switchbox and logic cluster.

Altera offer a large set of benchmark circuits with the *Quartus University Interface Program*, (QUIP) [2]. These are constructed in house or adapted from on-line sources. Whilst they are not high performance systems, they are, however, representative of the types of cores one would find in a System-on-Chip (SoC). They give a good indication of how individual cores would perform on TDM wiring. I have not included a system-level interconnect structure to any of the benchmarks because I am assuming that the system-level interconnect will be designed with TDM wiring in mind.

The NiosII soft embedded processor uses 1800 LUTs when configured for high performance, but fewer than 700 when configured for economy [2]. In order to reflect this range of core sizes, the benchmarks with more than 500 LUTs were selected. These were then placed and routed onto a Stratix FPGA and their detailed routing and timing information extracted using the Quartus TCL interface. These benchmarks are summarised in table 7.1.

Due to the constraints placed on the routing information format, this was often very verbose and I was forced to discard benchmarks which had more than 500MB of routing information. These benchmarks were too large to perform full parameter sweeps on, but some will be discussed later in this chapter.

Table 7.1: QuartusII University Interface Program Benchmarks

Benchmark		LUTs	I/O bits	Mem blocks	Clock frequency (MHz)	Max. wires per switchbox
A	ata_ocidec1	540	125	0	257	28
B	des_area_opt	691	189	0	201	30
C	des3area	1135	304	0	190	38
D	ata_ocidec2	588	125	0	257	40
E	ata_ocidec3	1045	130	224	190	42
F	ata_vhd	1040	130	224	198	42
G	aes_core	1680	388	32768	179	46
H	video_huff_enc	613	23	0	113	46
I	hdlc	640	82	0	196	50
J	mux32_16bit	853	54	0	153	50
K	barrel64	882	136	0	98	52
L	des_perf_opt	5336	185	0	187	52
M	mux64_16bit	1702	87	0	154	54
N	blowfish	1527	585	67168	114	60
O	aes_core_inv	1947	389	34176	93	62
P	vga_lcd	2207	585	32640	122	62
Q	pci	2439	367	1720	104	64
R	cfft_1024x12	1655	68	24576	187	68

7.4 Feasibility

TDM wiring is designed to create a trade-off between the number of wires and the number of configuration bits. The aim of this work is to exploit that trade-off and find a point at which TDM wiring is better than static wiring. Before exploring the results of the scheduling algorithm I would like to define a *good* result. This *good* result will be a point at which it is likely that the silicon cost of the TDM wiring is the same as the static wiring and the bandwidth of the TDM wiring is higher. I will estimate by how much the wire count in each switchbox must be reduced in order for the silicon area of the channel to remain the same. This gives a good basis for quantitatively accessing the scheduling results. A more detailed look at the silicon area of the TDM wiring is given at the end of this chapter.

I estimate the silicon area by assuming that the static wiring is configured using 8-bit SRAMs and that the wiring and configuration use approximately the same area. This is a reasonable assumption, confirmed by private communication. I take this area to be w therefore a static wire would take up (wire + SRAM) = $2w$ units of silicon area. The silicon cost of the wire includes the mux and all other components which are not SRAM bits.

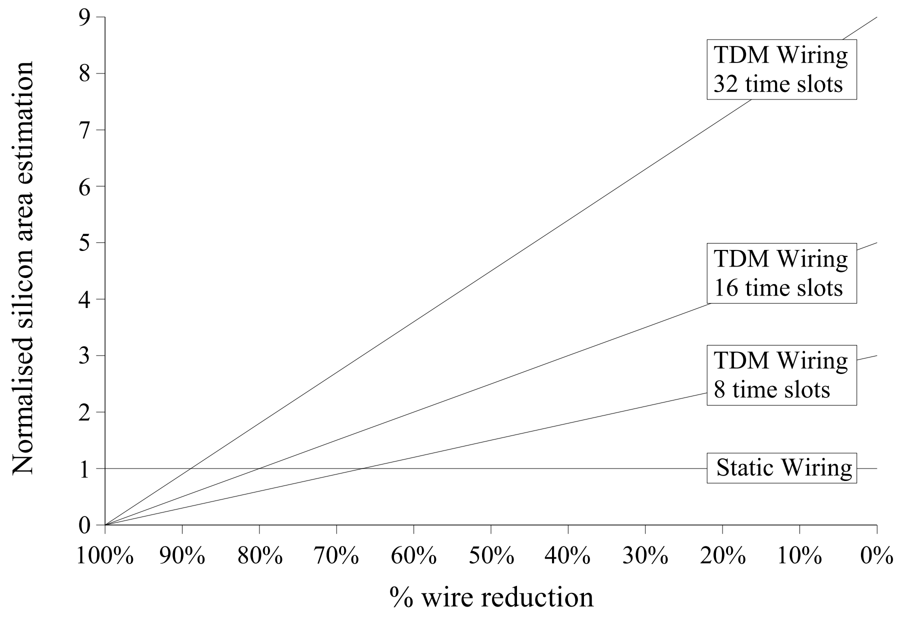


Figure 7.2: Silicon area comparison between the static architecture and TDM architectures with wire reduction.

The mux can be controlled using 4-bit SRAM instead of 8-bit SRAM at a cost of increasing the mux size. This has no effect on the size of the static wiring, but is beneficial to the size of the TDM wiring. The cost of a TDM wire with n time slots would therefore be (wire + SRAM) = $2w + (n/2)w$. These calculations do not attempt to take into account extra logic needed to implement TDM wiring.

Using these rough area calculations the graph in figure 7.2 was constructed. The graph shows the silicon area normalised to that of the static wiring for a variety of TDM architectures. The points at which the lines cross represent architectures with channel area equivalent to that of statically wired FPGAs.

With 8 time slots wire reduction of between 65% and 70% is needed to maintain the same channel silicon area. This rises to 80% for 16 time slots and 90% for 32 time slots. This means with 8 time slots channels would have 8 $\times$ 30% = 2.4 times the capacity of the static wiring. The 16 and 32 time slots both have a channel capacity of 3.2 times after the wire count has been reduced accordingly.

The architecture with 8 time slots would give the best power performance out of the three TDM designs considered because it is clocked at a lower speed and so will consume less dynamic power. The 32 time slot model will have the most flexibility, but all rely on the scheduler being able to exploit timing slack in the benchmarks and schedule them onto the device.

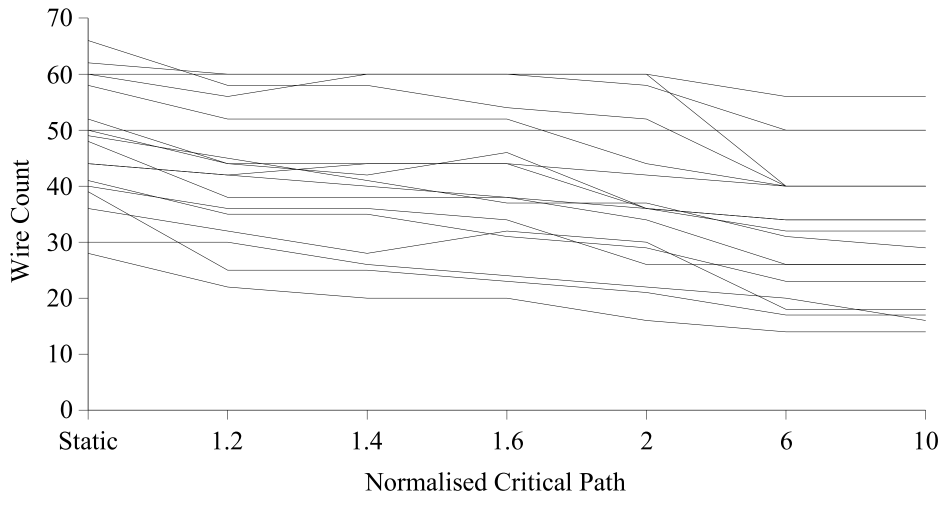


Figure 7.3: Scheduling results for 8 time slots. Each line represents a different benchmark.

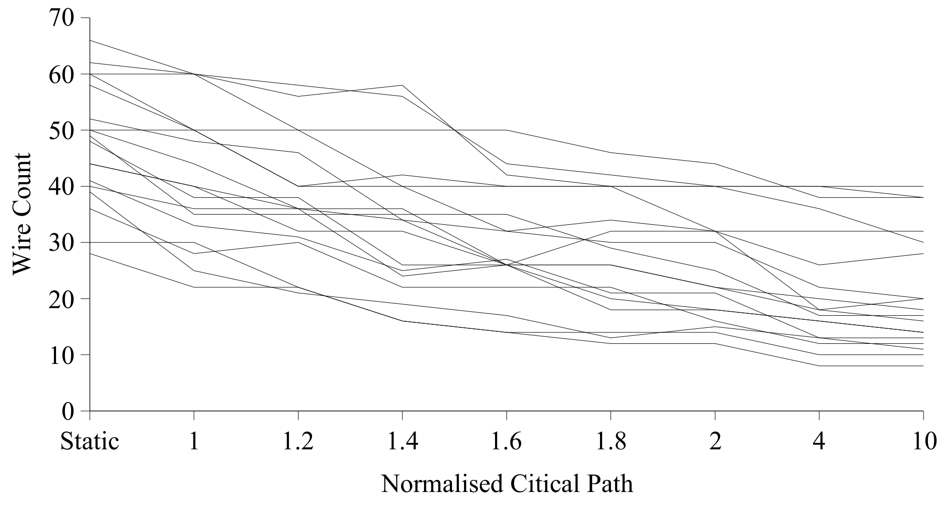


Figure 7.4: Scheduling results for 16 time slots.

7.5 Scheduling Results

The results of scheduling with 8, 16 and 32 time slots are presented in figures 7.3, 7.4 and 7.5. The length of a time slot was varied so that the new critical path was between one and ten times the length of the critical path on the static device. This is shown on the x-axis with a non-linear scale. The values were chosen to reflect points of interest in the architectural space. The results for 8 time slots were not very good so fewer points have been plotted. The results for 16 and 32 time slots are more promising and so were investigated with finer granularity. The normalised critical path delay was used to determine the length of time slot using the following equation:

$$\begin{array}{c}\text{Normalised}\\\text{critical path}\end{array} = \frac{\text{length of time slot} \times \text{number of time slots}}{\text{static critical path}} \tag{7.1}$$

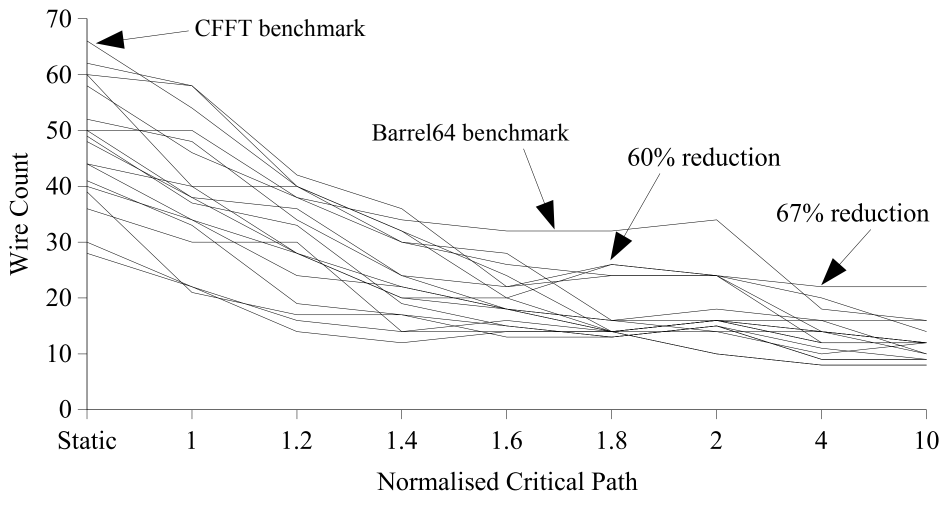


Figure 7.5: Scheduling results for 32 time slots.

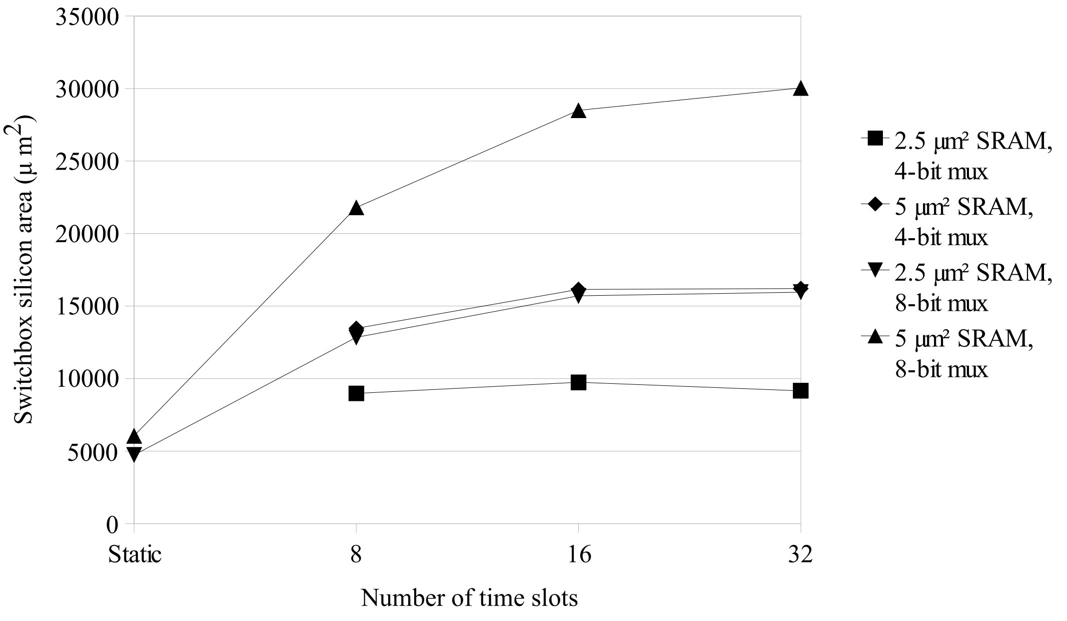


Figure 7.6: Graph showing switchbox silicon area against number of time slots. The wire count for each switchbox is the minimum number needed to route all the benchmarks with a normalised critical path of four.

For example, this means that a normalised critical path of 1.2 will be 20% slower in real time. A normalised critical path of 1.0 has the same critical path as the statically routed design. By bypassing all the latches on the critical path, it is possible for a TDM design to have the same critical path as a static design so this point has been included in the study, but the benefits are minimal.

The y-axis shows the number of wires needed to be driven per switch box in order to schedule the benchmarks on such architectures. Each line on the graph represents an individual benchmark and those of interest are labelled.

All the graphs clearly show the trade-off between wire count and critical path latency. As the critical path constraints were relaxed, the scheduler was able to reduce the wire count further.

I discovered that for my timing model, increasing the critical path by just a factor of 2 was enough to achieve dramatic reductions in the number of wires needed to route the critical path. This is particularly noticeable in the results for 32 time slots.

The maximum number of wires available on the Stratix is 72, but the maximum used by the benchmarks is 66. I therefore compare all new wire counts with 66 in order to take into account additional flexibility in the static architecture.

The best results are of course achieved by the architecture with the most time slots. This gives the finest granularity of control, but at the cost of increased configuration SRAM. I have to look to the architecture with 32 time slots to see a reduction of over 60% in the wire count. The majority of the benchmarks easily achieve more than a 60% reduction with small increases to the critical path. This reduction is good, but not good enough to justify the large amount of extra configuration SRAM needed to implement the TDM wiring. One benchmark, Barrel64, requires a longer critical path to achieve a lower wire count. This is looked at in more detail in the following sections.

Figure 7.6 shows the trade-off between number of slots and silicon area. The silcon area does not increase linearly with the number of time slots, but the switchbox is still much bigger than the statically wired switchbox.

In a real statically wired device, the wire count is higher than shown as I have plotted the wire count needed for the set of benchmarks. This flexibility is needed for the static device, but there is already a lot of redundancy in the TDM device. It is likely, therefore, that the gap is not quite as wide as shown on the graph, but I have choosen to plot pessimistic results throughout in order that the results are meaningful.

7.6 Clock Selection Optimisation

The number of time slots is critical to the architecture. It determines the flexibility of the wiring, the capacity of the channels and the amount of extra configuration SRAM required. I have devised a scheme whereby each switchbox is able to select either the positive or negative edge of the interconnect clock. Time slots are restricted to odd or even values for each switchbox. This takes advantage of the slack available in the schedule and halves the number of time slots needed. The details of this algorithm can be found in figure 7.7.

The optimisation is applied after the first scheduling stage. Either odd or even slots are chosen per switchbox based on what is best for the existing schedule. The scheduler is then rerun to

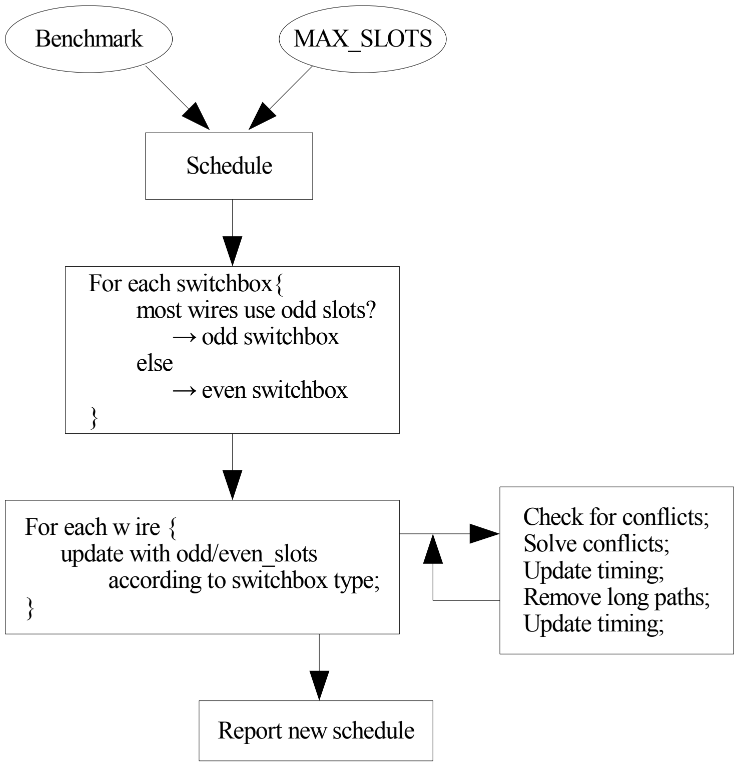


Figure 7.7: Flow diagram detailing the clock section optimisation.

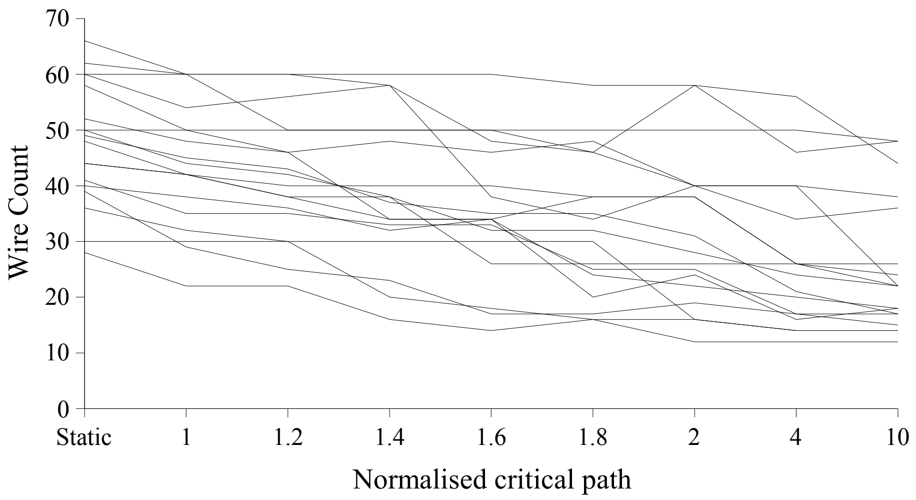


Figure 7.8: Scheduling results for 8 time slots with clock selection optimisation for a range of benchmarks.

remove any new conflicts. With better wire design for a TDM architecture, the time slots would fit more naturally to the wire delays and such an optimisation would not be needed.

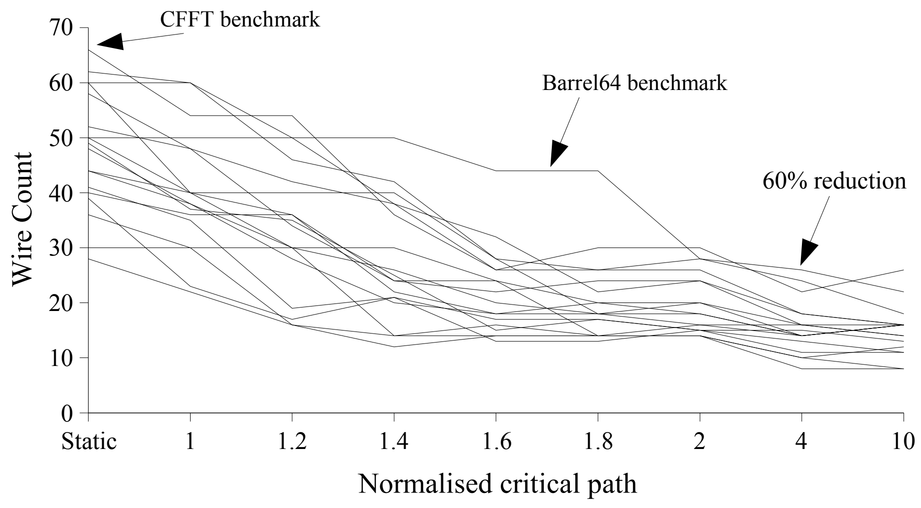


Figure 7.9: Scheduling results for 16 time slots with clock selection optimisation for a range of benchmarks.

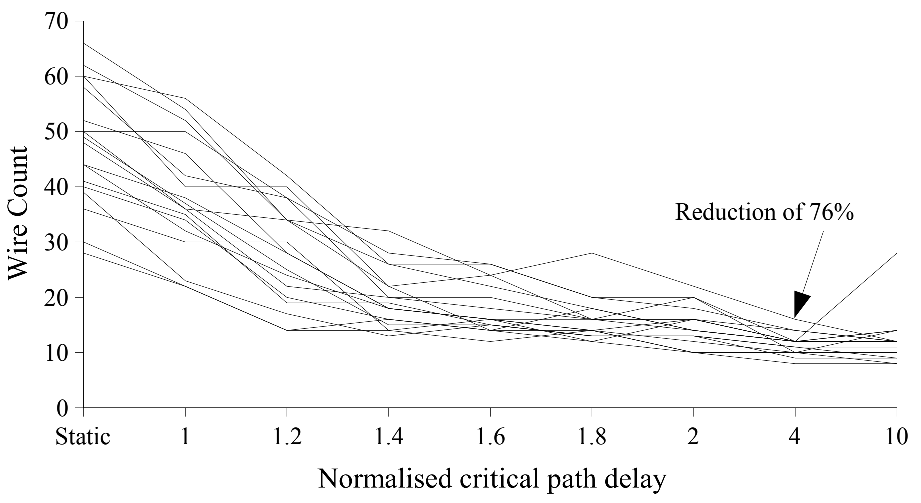


Figure 7.10: Scheduling results for 32 time slots with clock selection optimisation for a range of benchmarks.

7.6.1 Clock Selection Optimisation Results

Figures 7.8, 7.9 and 7.10 show the results after the clock selection has taken place. For n time slots in the optimised TDM architecture the results are similar to wire counts achieved with $2n$ time slots in the unoptimised TDM architecture. I increased the critical path as before and recorded the wire count required by the scheduler. For 32 time slots it is possible reduce the wiring by 76% with a critical path only four times that of the static configuration.

The most notable improvement is with 16 time slots: there is a similar pattern as with the results for 32 time slots in Figure 7.5. This is a marked improvement on the results shown in Figure 7.4. With the critical path extended four times, the wiring can be reduced by 60%. This is still

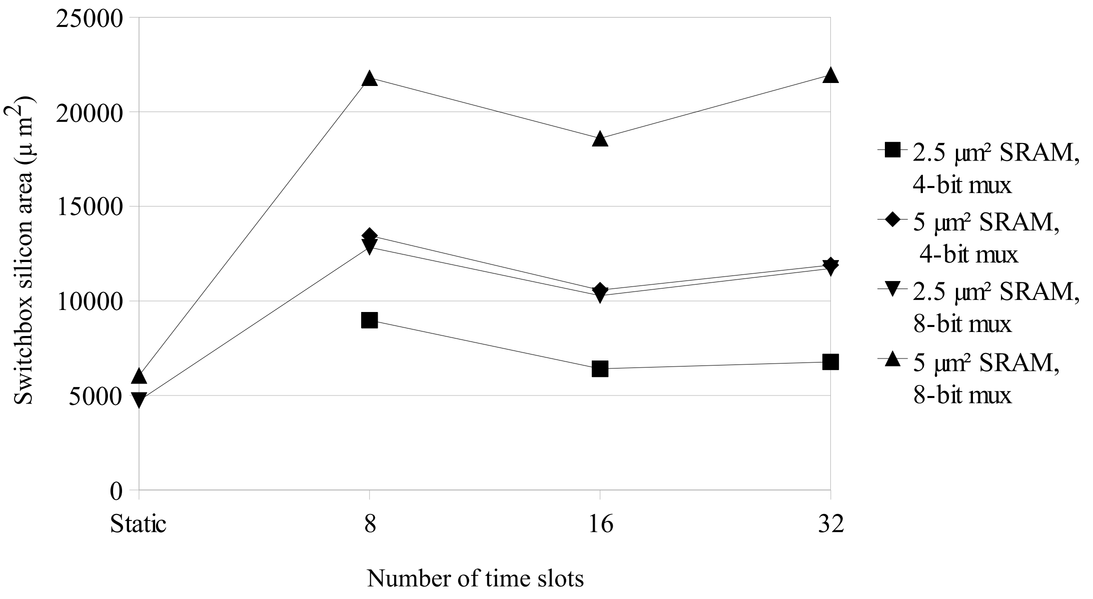


Figure 7.11: Graph showing switchbox silicon area against number of time slots. The wire count for each switchbox is the minimum number needed to route all the benchmarks with a normalised critical path of four and the clock selection optimisation.

not enough to justify the extra configuration SRAM, but it is a massive improvement on the first set of results.

Figure 7.11 shows the trade-off between number of slots and silicon area with the clock selection optimisation. There was not enough redundant flexibilty with 8 time slots to see much of an improvement, but the improvement is obvious with 16 and 32 time slots. The improvement with 8 time slots is not good enough to indicate that further wire reductions are possible. For the following further optimisations only 16 and 32 time slots are considered.

7.7 Wire Type Optimisation

The Stratix FPGA has 6 wires types running in two directions both horizontally and vertically. These are detailed in Table 7.2.

Decisions made at the place and route stage cannot be undone by the scheduler, which means that a TDM wire type must be provided for each static wire type. For symmetry in the routing there must also be the same number of wires in each direction. As a result, we could end up requiring a number of wires of different types in the system, where in an ideal system the wire types would be unified and the wire count reduced.

In the Stratix the two longest wire types are only driven in one direction in each switchbox. This means that with a minimum of one wire of each type the wiring cannot be reduced beyond

Table 7.2: Distribution of wire types in the Stratix FPGA

Wire type	R4	C4	R8	C8	R24	C16	Total
Wire count	40	20	6	4	1	1	72

10 wires (unless the benchmark does not use all the wire types). The ability to select the wire types is beyond the control of the scheduler and so a reduction beyond ten wires is not possible. This restricts the wire reduction using the scheduler to a maximum of $(66 - 10)/66 = 85\%$. The architecture with 32 time slots needs a wiring reduction of over 90% to keep the silicon cost low and so this architecture is not possible.

It is possible to contrive an example in which only one wire is used per switchbox, but each of a different type. The scheduler would be forced to include ten wires in every switchbox to accommodate. This happens to some extent in the scheduling experiments, leading to higher than necessary wire counts. With unified wire types, we would see a lower wire count.

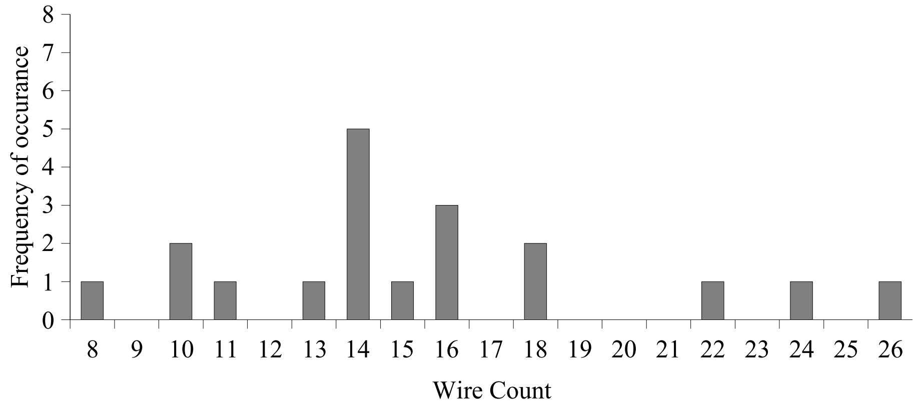
The most commonly used wires are those spanning four logic clusters horizontally and vertically. By removing the longer four wire types in each direction it is possible that we could reduce the wire count by up to 6 wires (four spanning 4 clusters, one spanning 16 and one spanning 24). The removal of these wires will lead to a higher count in the shorter wires, but the increase will not be as high as the number of wires removed because few benchmarks have a switchbox which uses every single wire available.

To reduce the wire count by 8 is optimistic, but I would expect to see a reduction in the wire count of at least 2 if the architecture was designed with fewer wire types because the two longest wires driven in each switchbox are only used for very long connections. These connections are likely to be replaced by the global interconnect scheme and so the longer wires can reasonably be removed at little cost to the performance of the system.

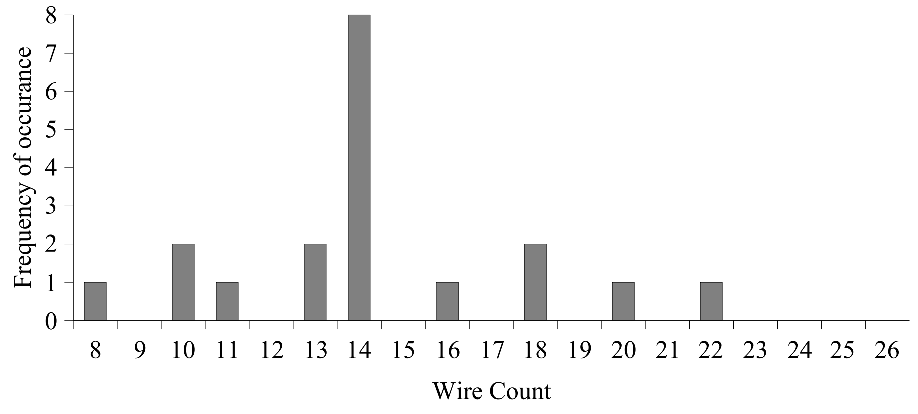
In order for the architecture with 16 time slots to be practical a reduction of around 80% is needed. This means a reduction from 66 to 14 wires, assuming even number of wires for symmetric driving in each direction. Assuming the reasonable removal of the two longest wires this means that the scheduler must reduce the wire count to 16 for each of the benchmarks with 16 time slots.

7.7.1 Wire Type Selection

The scheduling results are sensitive to the number of each wire type. The different wire counts are given to the scheduling algorithm as a vector. The tool searches for the optimum arrangement of wires by starting with a large number of each type and reducing this for the second attempt depending on the results of the first attempt. If the attempt fails then the number is increased until the schedule can complete successfully. Four iterations of this process was enough



(a) Wiring vector selected by algorithm.



(b) Optimum wiring vector selected by a complete sweep.

Figure 7.12: Wire count frequency over the 18 benchmarks for an architecture with 16 time slots and a critical path extension of four times.

to achieve a good result in general, but there is no guarantee that this algorithm is able to find the optimum wiring configuration for the benchmark. The starting vector for all experiments was set to (10,5,3,2,1,1) representing 10 R4 wires in each direction, 5 C4, 3 R8, and 2 C8. The R24 and C16 wire types are not duplicated and run in one direction only. This reflects a vector of wire counts close to the maximum number of wires available on the statically wired device. Brief experiments with different starting vectors resulted in variations in the set of vectors tried and so resulted in variations in the wire count for a particular architecture.

In order to measure the difference between the best wiring vector chosen by this algorithm and the optimum vector, a sweep of all possible vectors was used. The architecture had 16 time slots and the critical path could be extended up to 4 times.

The results of this comparison are shown in figure 7.12. The complete sweep of all possible vectors gives tighter clusters around 14 wires. The highest wire count with the optimised scheduler input is 22 wires. Removing the two longest wires in each switchbox give a reduction of $(66 - 20)/66 = 70\%$. This is an improvement on 60%, but still far from the 80% needed to justify the extra configuration SRAM for 16 time slots.

All but five of the benchmarks achieved wire count of 14. This is a $(66 - 12)/66 = 82\%$ reduction after the two longest wires have been removed. This would be enough to justify the extra configuration SRAM needed for the 16 time slots. The next few sections focus on why five benchmarks have a higher wire count and how that can be improved upon.

7.8 Congestion Analysis

The QuartusII ECAD flow, used to place and route the designs prior to scheduling, is optimised for static wiring. It is expected that this creates areas of high congestion in the TDM wiring unnecessarily. Eight of the benchmarks were chosen for congestion analysis. This group was selected by taking the five benchmarks that needed more than 14 wires in order to be scheduled and comparing them with those benchmarks which had the same or higher static wire count. This is the set of benchmarks for which the scheduler must work the hardest. The group allows a comparison between those for which the scheduler performed poorly, and those for which it performed well.

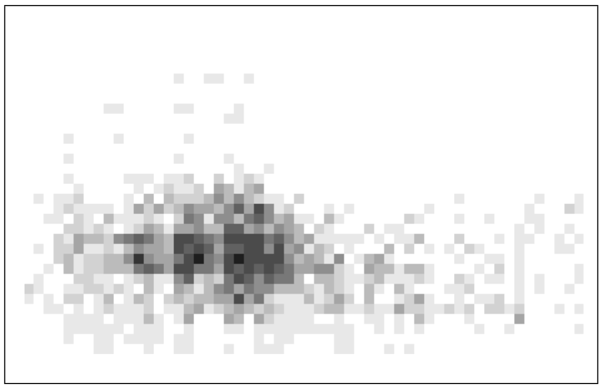
Figure 7.13 shows the congestion patterns given 4 times the original critical path and 16 time slots. The clock optimisation was used. Each spot represents a switch box with darker shades indicating a higher wire count. All of the benchmarks have been fit around the large block of on-chip RAM. This is more obvious in some than in others. The paler stripes show the arrangement of embedded DSP blocks and smaller memory blocks [2].

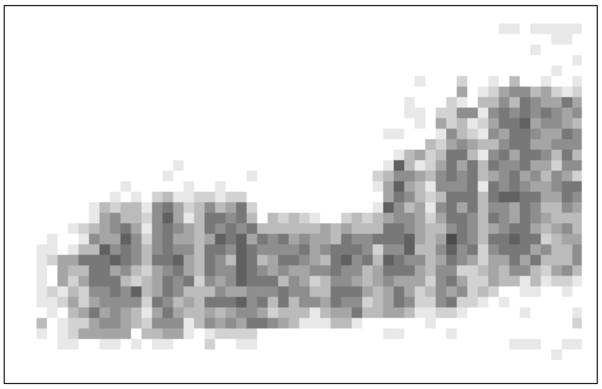
The benchmarks with the higher wire count have a few dark spots. Those with a wire count of 14 have a much more uniform distribution of wire count. This indicates that the switch boxes with a higher wire count are few in number and distributed independently. It is likely that a combined routing and scheduling tool would be able to exploit the under used wiring channels surrounding the congested switch boxes. With so few switch boxes affected this could be performed at the expense of a higher LUT clusters usage without making a significant difference to the logic density.

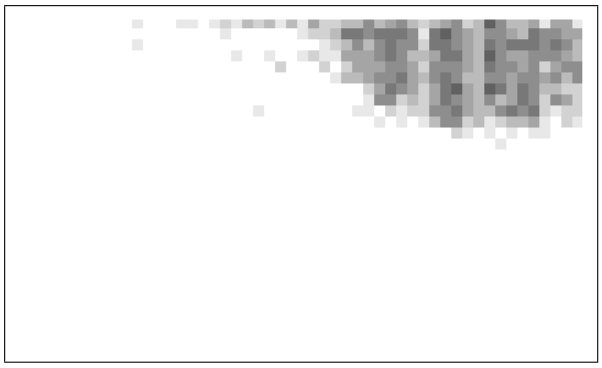
7.9 Random Seed Sensitivity

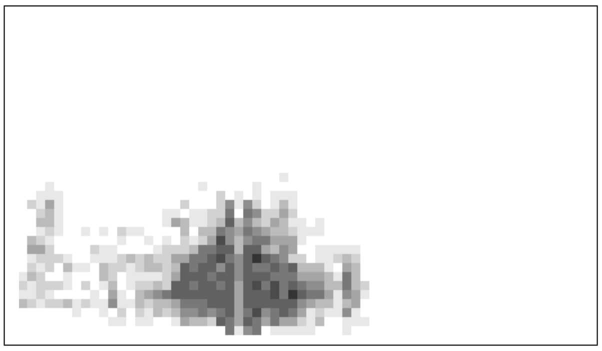
The resultant placement and routing pattern from Quartus depends on the random seed used at the fitter stage [2]. All the results so far have been generated from a placement with the default seed. To test the sensitivity of the results on this seed I have re-placed the five benchmarks which needed more than 14 wires per switchbox.

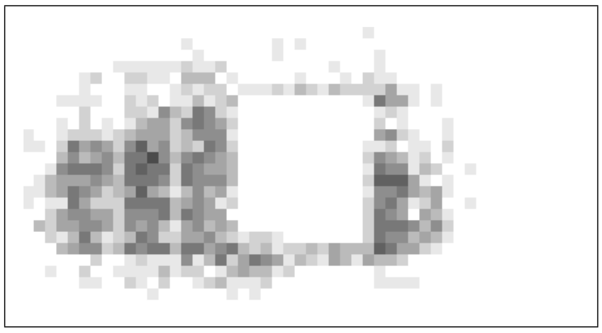
Table 7.3 shows the results of this re-placement. Each of the benchmarks was re-placed ten

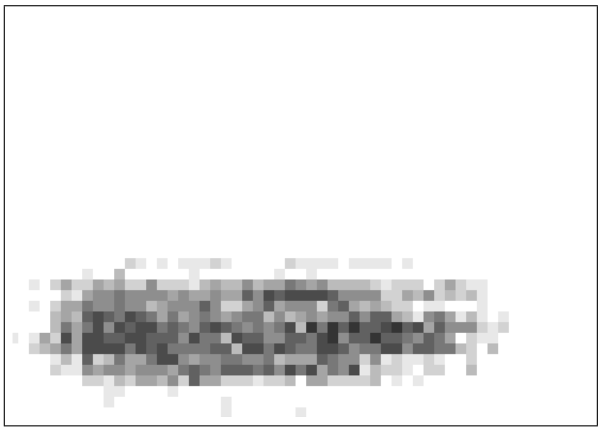

(a) Barrel64: wire count reduced to 20 from 50.

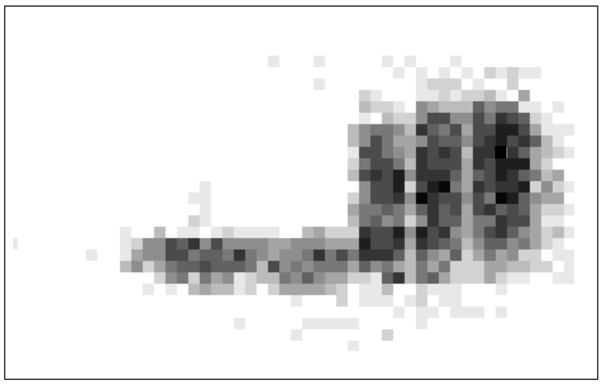

(b) Des_perf_opt: wire count reduced to 14 from 50.


(c) Mux64_16: wire count reduced to 14 from 52.


(d) Blowfish: wire count reduced to 16 from 58.


(e) Vga_lcd: wire count reduced to 14 from 60.


(f) Aes_core_inv: wire count reduced to 18 from 60.


(g) Pci: wire count reduced to 22 from 62.

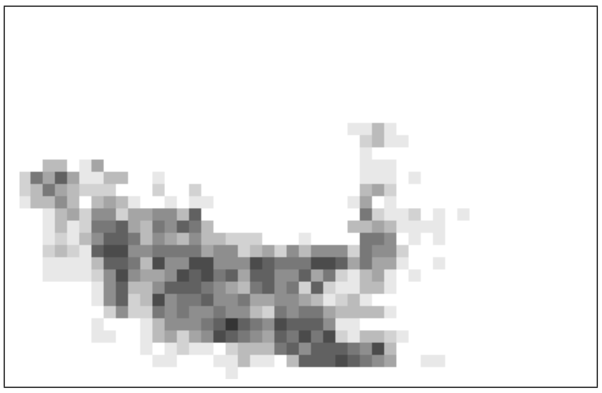

(h) CFFT_1024x12: wire count reduced to 18 from 66.

Figure 7.13: Congestion patterns

Table 7.3: Improved wire count using an alternative random seed for placement.

Benchmark		Wire count (default seed)	Wire count (selected seed)
K	Barrel64	20	16
R	Cfft_1024x12	18	16
Q	Pci	22	16
O	Aes_core_inv	18	16
N	Blowfish	16	16

Table 7.4: QuartusII University Interface Program Benchmarks

Benchmark		LUTs	I/O	Mem	Clock frequency (MHz)	Wires per switchbox
S	des3perf	1680	298	32768	124	68
T	Ethernet	2607	211	9216	77	66
U	fpu	6967	110	0	49	21
V	minirisc	635	389	1024	81	48
W	huffman_dec	649	23	0	105	55
X	wb_dma	3479	444	0	135	66

times using random seeds in the range of 0-9. Each placement was then scheduled with 16 time slots and four times normalised critical path. The best result for each benchmark was selected.

All the benchmarks achieved a wire count of 16. This is within the goals of the feasibility study and show that 80% wire reduction with 16 time slots is possible.

7.10 Large Benchmarks

Some of the benchmarks had to be represented by a very large amount of data. Others took a very long time to be processed by the scheduler. The large amount of data is a reflection on the way the routing data had to be extracted from the tools; it is not a reflection on the size of the benchmark, but it did prevent many experiments from being conducted due to timing constraints. The benchmarks which were too large or slow for the full set of experiments, but with less than 1GB of routing data, are listed in table 7.4. It was impractical to run the full parameter sweep on those benchmarks, but it was possible to run the algorithm with a single architecture. These benchmarks also represent an unseen set. Many techniques were tried on the original set during the development of the scheduling algorithm. To test the algorithm on this second set goes some way to demonstrate that scheduling is possible in the general case.

The results of scheduling these larger benchmarks are shown in table 7.5. As before, the wire count is the number of wire drivers needed per switchbox. The critical path was extended up to

Table 7.5: Results from scheduling with 16 time slots and normalised critical path.

Benchmark		Static wire count	TDM wire count
S	des3perf	68	18
T	ethernet	66	20
U	fpu	21	11
V	minirisc	48	14
W	huffman_dec	55	17
X	wb_dma	66	18

four times its original and the benchmarks could use up to 16 time slots.

Two of the benchmarks reach the targeted 16 wires (or fewer) and three more are not far off with 17 and 18 wires. The ethernet benchmark has a little further to go, but all use fewer wires than the PCI benchmark before it was re-placed with a different random seed. I am confident that these unseen benchmarks support the earlier findings. It is reasonable to expect the wire counts to reduce with a schedule-aware ECAD flow.

7.11 Clock Frequency

The first set of benchmarks has a range of clock frequencies on the Stratix of 93-257 MHz. The results of the scheduler indicate that a slow down of two-to-four times is necessary and sufficient for scheduling onto TDM wiring.

With a two times slow down the benchmarks running at 257 MHz would be reduced to 128.5 MHz. This is still a respectable clock speed for a soft core as the average clock speed was 162 MHz when the Stratix was released [70] and is an interesting performance/area trade-off. In the future hybrid FPGAs with TDM areas of high data throughput could be mixed with statically wired tiles for timing-critical computational cores.

An architecture with 16 time slots would need to run at 2056 MHz. This is high, but not unachievable [31]. Architectures with 32 time slots would be limited to running slower cores because it is unlikely that the interconnect could be designed to run at over 4 GHz.

7.11.1 Clock Frequency and Wire Reduction

The Barrel64 benchmark needed a longer critical path than some of the other benchmarks before the wire count was reduced. It also has one of the lowest clock frequencies of the group. Figure 7.14 shows the correlation between clock frequency and wire count. Each point on the graph represents a benchmark. The architecture has 16 time slots and the benchmarks were allowed a normalised critical path of four. The clock frequencies quoted are those of the benchmarks

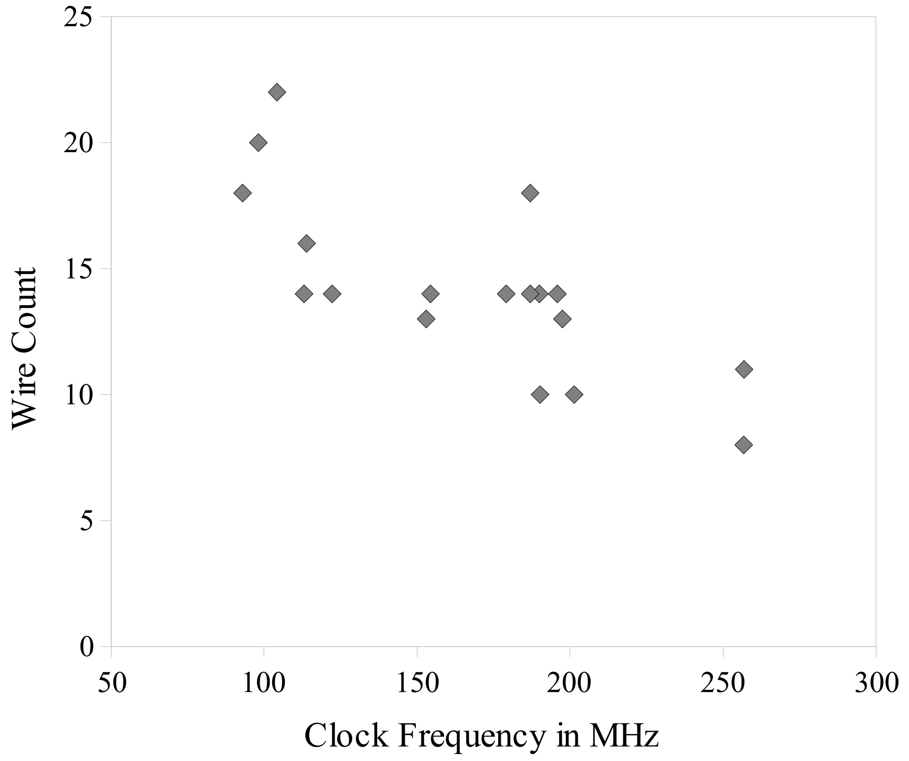


Figure 7.14: Correlation between clock frequency and wire count for architectures with 16 time slots and normalised critical path of four.

running on the Stratix. The wire counts are after clock selection optimisation, but with no other optimisations.

There is a clear inverse correlation between the clock frequency of the benchmark and the wire count. This is an unexpected result, but one which can be explained. The benchmarks with long critical paths will be more likely to have many long paths. These long paths will pass though more interconnect latches between user latches. Each interconnect latch incurs a timing penalty caused by the miss match between the time slot and the wire delay. The more mismatched, the larger the total delay on that path. Long paths will have longer delays added by the TDM wiring and be more likely to exceed the new specified critical path limit. When the new critical path delay is exceeded interconnect latches must by bypassed to reduce the timing penalty of the TDM wiring. Bypassing the interconnect latches allows long paths to meet their timing constraints, but makes it harder for the scheduler to share the wires efficiently. This leads to the higher wire count. Benchmarks with shorter critical paths have more flexibility and the scheduler is able to trade-off in favour of a lower wire count.

This trade-off would be improved by redesigning the wiring to better fit the uniform delays needed for optimal scheduling of TDM wiring. It is also possible to extend the critical path further to accommodate slower benchmarks.

7.12 Routing Capacity

The area benefits are small when just the channel width reduction is taken into account, but the increased overall capacity of the routing channels can be used to improve logic density in many other ways. With 16 time slots up to 16 bits can be sent down one wire segment in a single user clock cycle. This increases the bandwidth between hard blocks, allows resources to be automatically shared, and facilitates cross-chip and off-chip communication.

I have shown that an 80% reduction in wire count is achievable with TDM wiring capable of switching though 16 configurations (time slots). The wire count reduced to a fifth of its former value gives us a channel capacity of $16 \times 20\% = 3.2$ times the static channel capacity.

The initial look at the feasibility of TDM wiring with 32 time slots indicated that a wiring reduction of 90% would be needed if the silicon area of the wiring was to remain the same. This also gives a channel capacity of $32 \times 10\% = 3.2$ times the static channel capacity. This means that there are unlikely to be any benefits to using 32 time slots over 16 if the wire reduction has to be this high.

Given a routing capacity increase of 3.2 times, the scheduler can increase the critical path length 3.2 times before the bandwidth of the TDM channel is the same as that of a statically wired channel. For the majority of the experiments I have allowed a critical path increase of up to 4 times, but inspection of the data in figure 7.9 shows little difference between the results with normalised critical paths of 2 or 4.

7.13 Conflict Resolution Heuristics

Conflicts occur when more wires need to be mapped to a given location with the same time slot than there are wires on the device. Conflicts are removed by delaying signals until a free slot is available. Earlier schedules are extended to allow the signal to take more time to propagate. This keeps the number of wires required low, but at the expense of delaying signals and possibly causing additional conflicts elsewhere or extending the critical path delay.

For example, a simple signal with two logical wires could be scheduled with the first using time slot one and second using time slot two. A conflict at time slot two for the second wire would be solved by rescheduling the first wire at time slot one and two and the second at slot three. The conflict at the location of wire two was solved at the expense of increasing the signal latency by one slot.

The scheduler must decide which logical wires to reschedule when a conflict occurs. Decisions are made depending on the cost of rescheduling. Some wires can be rescheduled without affecting wires later in the path because there is slack in the timing. Those which do not affect

Table 7.6: Measures used by the rate function to determine which wires should have their schedules moved.

E	denotes the number of wires in the fanin that will need to have their schedule extended
C	denotes the number of new conflicts cause by extending those schedules
P	denotes the maximum delay between the input to the wire and a user defined latch
U	denotes the number of user defined latches which will have their schedule extended beyond the maximum number of slots
D	denotes the number of user defined latches which will have their schedule extended, but not beyond the maximum number of slots
M	denotes the number of slots by which the wire schedule needs to be moved

```
fun choose_wires_to_move (wires[], current_slot) {
  foreach wire in wires[] {
    wire.rate = rate_function(wire,current_slot,*E,*C,*P,*U,*D,*M)
  }
  wires[].sortby(rate); #ascending
  num_to_move = size(wires[]) - MAX_WIRES;
  return pop(num_to_move,wires[]);
}
```

Figure 7.15: Pseudo code describing the rating function for selecting which wires to reschedule in order to remove a scheduling conflict.

the critical path and do not cause further conflicts are selected for rescheduling over those with higher cost. After a conflict is removed the surrounding schedules are updated.

A number of measures are taken and used to determine which wires should be moved. These measures (U,D,M,E,C and P) are described in detail in table 7.6. Pseudo code for computing these measures is given in figure 7.16. The cost of rescheduling a wire to a later slot is determined using a simple function. The function is used to rate each wire and is described by the pseudo code in figure 7.15 and the equation 7.2.

$$Cost = U \times D \times M \times (E \times C + P) \tag{7.2}$$

Cost function 7.2 was chosen to be a starting point for further improvement. All experiments up

```
fun C,E (wire,current_slot) {
  c,e = 0,0;
  foreach f in wire.fanin_wires {
    if (f.last_slot <= current_slot){
      e++;
      g = f.schedule_group[];
      f.last_slot = current_slot;
      if (conflict_fount(g)) c++;
      restore(f.last_slot);
    }
  }
  return c,e;
}
fun P (wire,current_slot) {
  bottom_wires[] = get_bottom_wires_from(wire);
  max = 0;
  foreach b in bottom_wires{
    max = maximum((b.delay_in - wire_delay_in), max);
  }
  return max;
}
fun U,D (wire,current_slot) {
  new_delay_in = current_slot * SLOT_DELAY;
  w = wire;  fanout_list = [];
  while (w.new_delay_in > w.old_delay_in){ #difference greater than slack
    if (w.fanout_list == empty_list) {
      bottom_wires.append(w);
    } else {
      update_delay_in(w.fanout_wires[]);
      fanout_list.append(w.fanout_wires[]);
    }
    w = pop(fanout_list);
  }
  u,d = 0,0;
  foreach b in bottom_wires {
    new_slot = get_new_slot(b.new_delay_in);
    if (new_slot > MAX_SLOT)  u++;
    else                      d++;
  }
  return u,d
}
fun M (wire,current_slot) { return current_slot - wire.first_slot + 1; }
```

Figure 7.16: Pseudo code describing the measures used by the rating function.

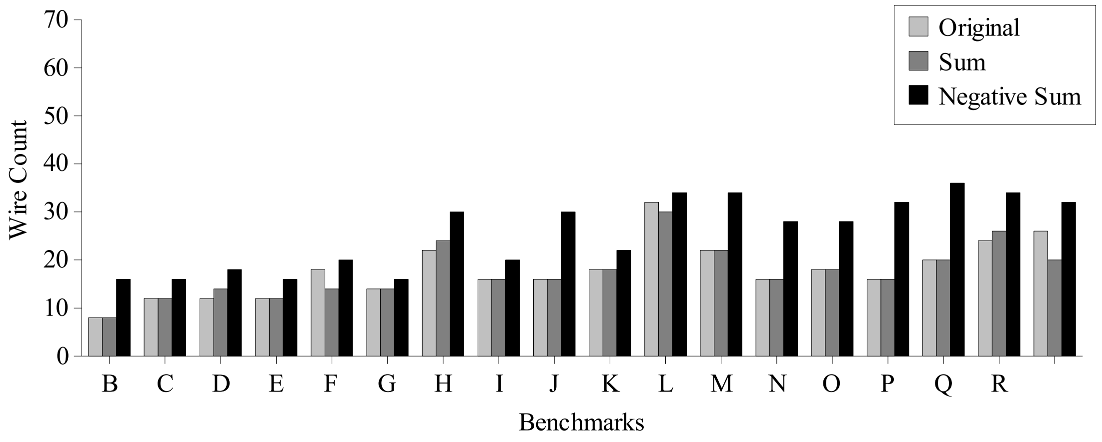


Figure 7.17: Sensitivity of conflict resolution heuristics.

to this point have been computed using this cost function. It is designed to give a large positive value when the delay of this schedule affects several user defined latches (U and D) and zero otherwise. This value is largely determined by the length of the path to those latches (P). It gives little emphasis to local problems such as extra conflicts introduced to the fanin wires (C). I assumed that the scheduler was better able to solve the local problems than problems caused by increases to the critical path and the removal of interconnect latches.

After inspecting the function values for a selection of the benchmarks it because apparent that this function is zero for the majority of cases and so the majority of conflict resolution decisions are made obliviously. It is almost equivalent to using no cost function. The order in which the wires are consistent depends on the ordering in which they were presented to the scheduler and this determines the order in which they are chosen for rescheduling.

$$Cost = U + D + M + E + C \tag{7.3}$$

An alternative function which gives equal weight to local conflicts and problems with the critical paths is given by the sum function 7.3. This gives a small non-negative value. The fanout path length is discarded to give more emphasis to the other heuristic parameters.

To test the sensitivity of the cost function, the two cost functions were compared with the negative of the sum function. If the sum function always makes the best choice then its negative will always make the worst choice.

The results for this comparison are shown in figure 7.17. The sum function gives almost identical results to the original function. The negative sum function performs

badly. This shows that the function is not random, but that in general the scheduler is limited by other factors, such as the routing configuration, rather than its ability to select wires for rescheduling. As long as a poor choice is not made every time, the scheduler is able to achieve good wire reduction.

7.14 Wire Area Analysis

In order to evaluate the area cost of the TDM architecture, the area of the static wiring is compared to TDM wiring using the model described in section 4.6.

Without access to commercial FPGA layout there is likely to be a certain amount of error in these estimations. This effect is most prominent in estimating the area of the SRAM. In order to complete a more accurate comparison, I have calculated the area for the maximum and minimum SRAM area costs. A study puts the cost of a 90nm SRAM cell as low as $1\mu m^2$, but with the control overhead the average area cost per bit is $2.5\mu m^2$ [10]. FPGA SRAM has to constantly drive the muxes and so is likely to have a higher area cost. Calculating the total number of configuration bits in a switchbox and comparing this to the area of the switchbox indicates that the cost of one SRAM bit on the FPGA is less than $5\mu m^2$. Area results for both $2.5\mu m^2$ and $5\mu m^2$ SRAM are presented.

The scheduling and optimisation results indicated that an 80% reduction in wire count should be possible with 16 time slots. The crude feasibility study indicated that a 90% reduction in wiring would be necessary to implement an architecture with 32 time slots. The scheduler was not able to reduce the wire count beyond one of each type and direction (a reduction of 85%), but given the large reduction possible with 16 time slots it is not unreasonable to expect a 90% reduction with 32 time slots. The area results for 32 time slots are given with both 80% and 90% wire reduction.

The wiring was based on the horizontal and vertical wires spanning four switchboxes in the Stratix II [56] because these are the most commonly used wires. Those familiar with the Stratix II will note that the estimations are smaller than the actual switchbox area. This is because they contain many components such as local wiring and testing circuitry, which are of no interest to this study.

I assume that there are 80 wires per switchbox because this corresponds to the architecture of the Cyclone II FPGA [2]. The Cyclone II is a simplified version of the Stratix II with fewer wire types. It was not used for any of the previous experiments because it is designed for low performance.

The TDM area includes the wiring cost and the cost of adding a control mechanism to each switchbox. Many clocks of varying performance are already distributed throughout a modern FPGA [53]. The TDM wiring requires that a one-hot counter and pulse generator is added to the clock network. I have included these in the area calculation of each switchbox, although they could be shared between neighbouring switchboxes.

All architectures included in the area study are summarised in table 7.7. This table also gives the architecture abbreviation needed to understand the the results. The results are shown in figure 7.18.

Table 7.7: Summary of architectures included in the area comparison.

Abbreviation	Schedule	Time Slots	SRAM bits per mux	Wire count reduction
Static	Static	NA	8×1	0%
16TS 8bit 80%	TDM	16	8×16	80%
16TS 4bit 80%	TDM	16	4×16	80%
32TS 8bit 80%	TDM	32	8×32	80%
32TS 4bit 80%	TDM	32	4×32	80%
32TS 8bit 90%	TDM	32	8×32	90%
32TS 4bit 90%	TDM	32	4×32	90%

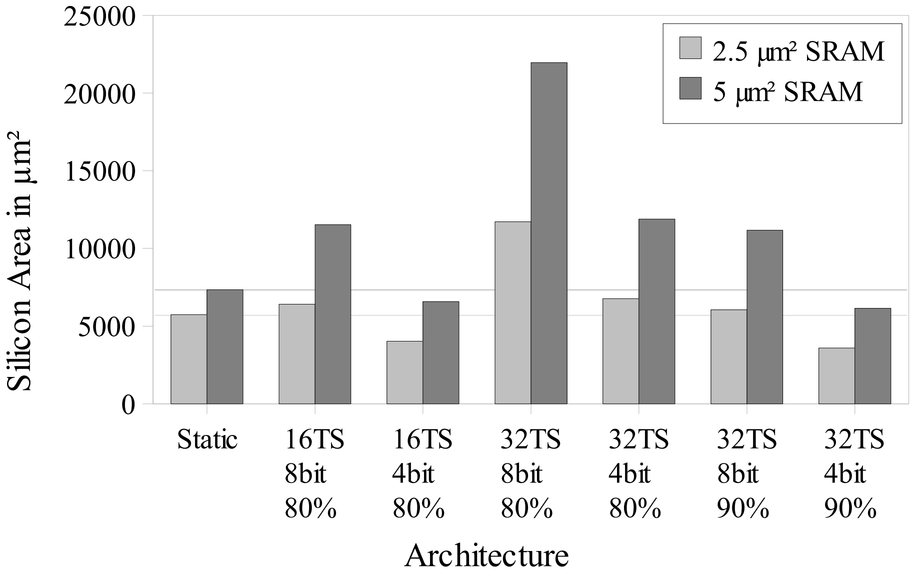


Figure 7.18: Switchbox area for static and TDM wiring.

As predicted by the feasibility study, only the architectures with 16 or 32 time slots and an 80% or 90% wire reduction respectively have a smaller silicon area than the static wiring. Other architectures come close, however. This is particularly true of those with small SRAM sizes.

The area measurements are not accurate enough to definitely demonstrate that the switchbox area can be reduced by using TDM wiring, but it is possible to conclude that for approximately the same area cost, the channel capacity can be significantly increased. The architecture abbreviated to “32TS 4bit 80%” could increase the channel capacity from 80 to 512 signals for a fractional increase in silicon area. This is an increase in routing capacity of 6.4 times.

It is likely that with more SRAM a more compact design is practical and the SRAM size can be reduced. This will decrease the area cost of all the TDM architectures.

Table 7.8: Stratix II wiring information (estimated).

Wire type	Muxes per switchbox	Load points	Mux size	Fanout
R4	52	4	16	7
C4	32	4	16	10
R24	1	6	32	15
C16	1	4	32	10

7.15 Switchbox Connectivity

Until this point the scheduling algorithm has assumed that the switch box is totally connected. In modern FPGAs a trade-off is made between flexibility, mux inputs and fanout [35, 60]. Table 7.8 gives estimates for the mux sizes and wire fanout. These values have been estimated using the publicly available wire counts for the device and the publicly available switchbox architecture of the Virtex4. The design is complex and non-uniform due to the mismatch between the number of each wire type.

Based on the previous experiments, a TDM architecture may have wires spanning four switchboxes with loads in the middle and at the end. Each switchbox would drive 80/5 = 16 wires. These would be arranged with four in each direction. Each wire should be able to drive other wires at their end and mid-point. This model is based on the Stratix architecture, but is simplified to allow clear connectivity estimation.

If the switchbox drives four wires in each direction, each wire would have to be driven by four terminating wires from each of the three other directions with another four from each direction driving from their mid-points. This gives a fanin of 24 wires from the global wires, figure 7.19 illustrates this. Driving muxes make most efficient use when they have binary-valued fanin so the smallest driving mux that would allow 24 global connections and a reasonable number of local connections would be a 32:1 mux.

In the area estimations I have assumed 16:1 muxes as used by the static R4 and C4 wires. A 32:1 mux may be too expensive. Some reduction in connectivity is necessary to arrive at a realisable design of TDM wiring. This reduction in connectivity is beyond the scope of the tools at my disposal and would form part of a detailed study into the precise wiring architecture needed to implement TDM wiring. The ease with which connectivity has been reduced in FPGAs in the past indicates that this would not be a problem [68].

7.16 Summary

In this chapter I have explored FPGA architectures in which all the global routing wires have been replaced with Time-Division Multiplexed (TDM) wiring. These wires have been modelled

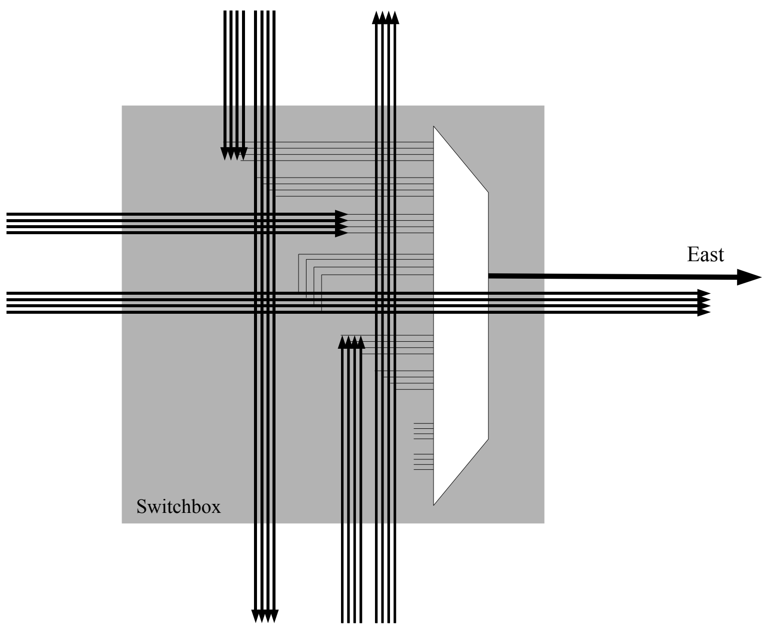


Figure 7.19: A switchbox capable of driving 16 wires; only one wire driven East is shown. It is driven by 8 wires from the North, 8 from the West, 8 from the South and 8 local connections. Half the global wires are driving the mux from their end points and half from their mid points. No other wires present in the switchbox are shown.

with 8, 16 and 32 time slots. I have used the scheduler to map a set of benchmarks to each of these models and explored the trade-off between the number of wires needed to schedule the benchmarks (wire count) and the increase in critical path required. The goal was to reduce the number of wires to the point where the TDM architecture uses the same silicon area as the static design, at minimum timing cost.

The architecture with 8 time slots did not allow enough of a reduction in the wire count to be practical. The additional configuration SRAM used more silicon area than the area of the static wires removed.

The architecture with 16 time slots was more promising. The results from the scheduler indicate that an 80% reduction in wiring is possible. Subsequence area analysis indicated that this was sufficient to be able to implement TDM wiring using the same silicon area as the static wiring architecture.

The architecture with 32 time slots still holds some interest. The scheduler was restricted to including a single wire of each type in the switchbox and was therefore unable to show much more than an 80% reduction in wire count. The area analysis showed the possibility that it may be possible to have 32 time slots with only an 80% reduction in wiring, but there is likely to be an increase in the silicon area used by the wiring. A 90% reduction is more likely to be area efficient, but gives the same channel capacity as the 16 time slot architecture. The 32 time slot architecture is only beneficial if the finer granularity of scheduling is used.

Overall, the results indicated that the architecture with 16 time slots gives the best trade-off between the number of wires and the number of configuration bits. This may change as the wiring and the tools are redesigned, but I believe I have identified a wide enough design space in which this architecture is practical.

Chapter 8

Conclusion

The contribution of this thesis is in three parts: (1) an investigative contribution, (2) an innovative contribution, and (3) a proof of concept.

Firstly, I have examined the challenges faced in the design of Networks-on-Chip (NoCs) for FPGAs. I have highlighted problems never before identified and drawn attention to the differences between NoC design for ASICs and NoC design for FPGAs.

The second contribution is a NoC design which takes advantage of hard and soft characteristics to fully exploit the FPGA platform and implement the NoC in the most efficient way. The design is a circuit-switched high-speed hard router which uses reconfigurable Timing-Division Multiplexed (TDM) wiring to funnel data in and out of slower soft cores.

Thirdly, I investigate the potential of replacing all static wiring on the FPGA with TDM wiring. The feasibility of the new architecture is tested by mapping open-source soft IP blocks to this architecture using a custom scheduler to convert the static wiring into TDM wiring.

By sharing the TDM wiring, the number of wires needed in each channel is reduced. Subsequent area analysis has shown this to be sufficient to implement the TDM wiring at no area cost. The channel silicon area remains the same but the bandwidth is increased. This extra bandwidth can be used to implement efficient NoCs and time multiplex hard blocks and I/O automatically. The logic density will be increased at no cost to the design time of the system.

8.1 Investigative Contribution: NoCs for FPGAs

An important motivator for the development of NoCs for ASICs is the changing trade-offs between transistors and wiring. However, this trade-off is not the same on an FPGA. This observation is enumerated in Section 5.5.3. I determined that the area of my router increased by around thirteen times when converted from an ASIC implementation to an FPGA implementation. The wiring increased by only five times. This does not mean that wiring flexibility is more efficient than logic flexibility. Wiring flexibility is implemented using redundancy, so the components used are more area efficient than the logic components, which are implemented with high utilisation in mind, but in any design there will be a high proportion of unused wires. The amount of wiring required is determined by the bottlenecks in the system rather than the overall usage. For NoCs this means that on-chip communication vs. computation does not trade-off in the same way as for ASICs.

Another difference between ASICs and FPGAs is the cost evaluation of a component. This is important in the comparison of hard and soft IP blocks. The cost of a single hard block is much lower in terms of area and therefore power than a soft core, but the area cost of an unused hard block can be the same as when it is used. An unused soft core never incurs a penalty by comparison. In Section 5.4.2 I showed that the cost of a hard network with static wiring could be higher than that of a soft network if only one in sixteen hard routers was used. This problem is rarely taken into account when evaluating hard and soft cores.

In order to be usable, FPGAs need to put flexibility over efficiency, but flexibility comes at a high cost [52]. When designing a network for comparison, I enumerated all the choices a NoC design must make and tried to eliminate as many as possible without reducing the performance of the NoC.

NoCs for FPGAs can be designed at the same time as the soft System-on-FPGA, which is after the silicon design has been finalised. NoC designers can exploit the fact that the application is known and therefore the traffic pattern. In this way the network router can have reduced flexibility because a custom topology for the known traffic pattern can ensure low congestion over the whole network. This helps the network remain small. I designed a circuit-switched network router with this in mind.

The size of the network was of specific importance. A survey of previous NoCs for FPGAs in Section 3.7.2 revealed a variety of networks with very large area figures. Many soft routers were as large as the cores they served. ASIC routers use 5-10% of the silicon area. There is little to suggest that soft routers cannot do the same. Hard routers are many times smaller so they should use many times less reconfigurable area. This observation was use to form the following hypotheses:

1. The hard network can be implemented on an FPGA at high granularity while using less than 1% of the silicon area.

2. The soft network can be implemented on the FPGA at high granularity using less than 5% of the reconfigurable area.

8.2 Innovative Contribution: TDM Wiring

Both hard and soft versions of a circuit switched router were implemented and compared. I was able to implement the hard and soft routers using less than 1% and 5% of the chip area respectively.

Two of the hard networks in the comparison used TDM wiring. This allows an exchange of bandwidth for clock frequency. The TDM routers were therefore smaller and no longer limited by the clock frequency of the soft cores. The TDM wiring adds reconfigurable flexibility to hard network routers thereby making them a viable alternative to soft routers. The silicon area is moved from non-configurable routers into reusable TDM wiring components.

I also considered the interface between the hard routers and the configurable routing. Insertion of the hard routers into the reconfigurable fabric posses a problem because loss of regularity is costly and produces low yields. FPGAs are designed with redundant wiring resources, but this was not enough to connect the hard routers to the reconfigurable wiring. TDM wiring was needed to reduce the bottleneck.

My results show that TDM wiring is needed to ensure that the hard router is still an efficient choice. Huge power reductions are made by using hard routers instead of soft.

8.3 Proof of Concept: FPGAs with TDM Wiring

Statically-scheduled Time-Division Multiplexed (TDM) wiring forms a compromise between the high-level, dynamically-routed bus-based wiring of a NoC and the low-level, statically routed FPGA wiring. Although statically scheduled, the TDM wiring changes configuration between interconnect cycles to use wires more efficiently that the conventional FPGA wiring, but without the overhead of dynamic switching.

The scheduling tool is able to share wires effectively and reduce the amount of wiring required on the FPGA. The wiring is clocked many times faster than the user clock so the signals can be pipelined and the wires shared without changing the functionality of the circuit.

All wires are shared, but not necessarily pipelined. The scheduling tool trades-off between effective wire sharing and critical path delay. The wiring was redesigned to encode configuration bits more efficiently and reduce the silicon area. This transistor-level model of the wiring was used to estimate the silicon area of my TDM architecture.

My results in Chapter 7 indicate that the amount of wiring required can be reduced by as much as 82% whilst running the interconnect clock sixteen times faster than the user clock. The interconnect cycles through sixteen wiring configurations every user clock cycle. Area estimations indicate that this wire reduction is sufficient to reduce the silicon area, despite the extra configuration SRAM. This reduction in silicon area comes with an increase in channel routing capacity.

The benefits of an architecture with TDM wiring are:

- the non-reconfigurable hard router area is reduced by moving the silicon cost of the network into reusable TDM wiring components.
- the high-frequency hard router is no longer performance limited by the lower frequency of the soft IP cores.
- the TDM wiring can be used to time multiplex other hard components and improve the hard-soft interface for them.
- the TDM wiring can be used to time multiplex soft components at a fine grain level and exploit permutation equivalent LUT reuse.

More efficient use of hard IP blocks is made possible by time multiplexing them in conjunction with TDM data transport. Look-up table utilization can be reduced through automatic serialization of system-on-FPGA interconnect and off-chip communication.

8.4 Future Work

This thesis has described two new FPGA architectures and demonstrated their feasibility. Both bridge the gap between statically wired FPGAs and dynamically switched NoCs, but there remains a gap between FPGAs with only TDM wiring and those with a little TDM wiring for the implementation of NoCs.

8.4.1 CAD Flow

It is likely that with a fully custom ECAD tool chain a compromise can be reached between the TDM wiring and static wiring such that critical paths can still be routed on TDM wiring, but there is still enough TDM wiring to route more than just the NoC.

The following CAD flow would need to be developed for commercialisation of the TDM architecture:

1. System Design and NoC specification
2. Synthesis
3. Clustering and Technology Mapping

4. Floorplanning

5. Placement, Routing and Scheduling

System Design and NoC Specification

The system design would need little change; both Xilinx and Altera provide high level system construction as part of their CAD flows. Software allowing hock-up to the available soft bus IP could be extended to allow connections to one or more NoCs.

The NoC specification would have to be explicit. Connections to the NoC could be made at the click of a button, but a set of constraints with all possible connections between nodes and the bandwidth of these connections would need to be provided for each node by the system designer so that a suitable NoC topology could be generated.

The system designer would need to explicitly connect nodes onto the hard circuit switched network or generate soft packet-switched routers, but at this stage the arrangement of nodes in relation to the routers would not be known.

For example, a specification could be that nodes A,B and C connect to network N: inside network N, A can communicate with B and C with equal bandwidth, but there is never a connection between B and C.

Synthesis

Each computational core would need to be synthesised separately to maintain the system hierarchy. At this stage information about design could be gleaned from the high-level description and annotated onto the synthesised netlist. This technique was used to map designs to FPGAs with bus-based connections [86]. Simple timing constraints could be generated to aid scheduling in later stages.

Clustering and Technology Mapping

Clustering and technology mapping should be schedule-aware. Clustering and technology mapping should use the timing constraints to group LUTs with different timing constraints together. Logic with identical fanin and fanout delays should not be placed in the same logic cluster because this puts undue strain on the scheduler to share the wires for that cluster efficiently. Locality can still be maintained by using neighbouring logic clusters.

In more developed architectures LUTs can be time multiplexed as well as the wiring: it would be possible to map multiple permutation-equivalent functions to a single LUT which is time-multiplexed by the wiring. It would be at this stage that potential for an opportunity would be identified. Similar techniques have been recently developed to map circuits to loops of simple assembly instructions and run them on arrays of processing elements [42].

Floorplanning and NoC generation

After clustering the size of each core would be known and so floor planning can take place. This will have to be in conjunction with a NoC generation tool. The tool will strive to generate a NoC that will satisfy the constraints written for the system. For a complicated system there may have to be several rounds of generation and manual adjustment to ensure that locality is exploited and NoC congestion minimised.

Placement, Routing and Scheduling Efficiency

Placement, routing and scheduling is likely to be a looped operation with poor schedules over come with re-placement and re-routing techniques. With good decisions made throughout the CAD flow the scheduler should have little to do.

The scheduling algorithm written for this thesis was designed with good results taking priority over run time. With hindsight and a greater understanding of the problem it is clear that it could be designed to run more efficiently without compromising on results. With reasonable consideration at the place and route stages, conflicts will be few. The scheduler can be designed to work progressively performing all schedules for a given time slot before moving onto the next. This dramatically reduces the amount of computation involved at each stage.

For the current time slot each signal would request a wire. If two signals request the same wire one will be buffered by allocating it to the fanin wire for another time slot. This is the same as the algorithm described in this thesis, but this time there is no updating required because the next time slots have not been considered yet. If the conflict cannot be solved in this way then one of the signals is rerouted. This alteration to the algorithm makes it significantly more efficient. It would grow roughly with the number of wires in the benchmark.

8.5 Long Term Developments

In the short term, I expect a small number of circuit-switched networks should be sufficient for most Systems-on-FPGAs, however as systems grow there will be more demand for more scalable architectures with packet-switched routing such as the one described in section **??** and shown in figure 8.1.

In the long term, the network will only get more important as transistors fail to scale and we have to turn to other measures to scale integrated circuits. Some suggestions involve cutting off-chip costs by combining multiple wafers in a single package [40]. The inter-wafer connections

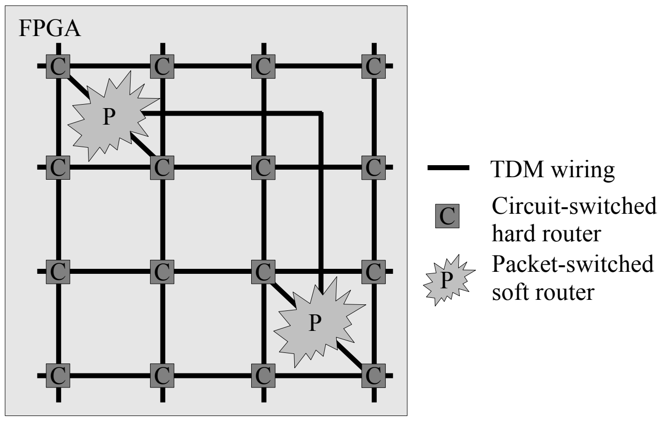


Figure 8.1: A combination of hard circuit-switched routers and soft packet-switched routers.

will be cheaper in terms of area, power and latency than inter-package connections, but they will still be more expensive than intra-wafer wiring so the need to trade-off between computation an computation will increase.

Bibliography

[1] electric.sun.com.

[2] www.altera.com.

[3] www.magma-da.com.

[4] www.synopsys.com.

[5] www.tsmc.com.

[6] www.umc.com.

[7] www.xilinx.com.

[8] Stratix device handbook, 2005.

[9] Stratix II device handbook, 2006.

[10] B. Agrawal and T. Sherwood. Guiding architectural SRAM models. *In proceeding of the International Conference on Computer Design*, 2006.

[11] A. Ahmadinia, C. Bobda, J. Ding, M. Majer, J. Teich, S. P. Fekete, and J. C. Van Der Veen. A practical approach for circuit routing on dynamic reconfigurable devices. *In proceedings of the International Workshop on Rapid System Prototyping*, 2005.

[12] E. Ahmed and J. Rose. The effect of LUT and cluster size on deep-submicron FPGA performance and density. *In proceedings of the International Symposium on Field-Programmable Gate Arrays*, 2000.

[13] G. Ascia, V. Catania, M. Palesi, and D. Patti. A new selection policy for adaptive routing in network on chip. *In proceedings of the International Conference on Electronics, Hardware, Wireless and Optical Communications*, 2006.

[14] A. Banerjee, R. Mullins, and S. Moore. A power and energy exploration of network-on-chip architectures. *In proceedings of the International Symposium on Networks-on-Chips*, 2007.

[15] A. Banerjee, P. T. Wolkotte, R. D. Mullins, S. W. Moore, and G. J. M. Smit. An energy and performance exploration of network-on-chip architectures. *In Transactions on VLSI Systems, volume 17, issue 3*, 2008.

[16] T. A. Bartic, J-Y. Mignolet, V. Nollet, T. Marescaux, D. Verkest, S. Vernalde, and R. Lauwereins. Highly scalable network on chip for reconfigurable systems. *In proceedings of the International Symposium on System-on-Chip*, 2002.

[17] V. Betz and J. Rose. Directional bias and non-uniformity in FPGA global routing architectures. *In proceedings of the International Conference on Computer-Aided Design*, 1996.

[18] V. Betz and J. Rose. Cluster-based logic blocks for FPGAs: Area-efficiency vs. input sharing and size. *In proceedings of the Custom Integrated Circuits Conference*, 1997.

[19] V. Betz and J. Rose. VPR: A new packing, placement and routing tool for FPGA research. *In proceedings of the International Conference on Field Programmable Logic and Applications*, 1997.

[20] V. Betz and J. Rose. How much logic should go in an FPGA logic block? *In Design and Test of Computers, Volume 15, Issue 1*, 1998.

[21] V. Betz and J. Rose. Circuit design, transistor sizing and wire layout of FPGA interconnect. *In proceedings of the Custom Integrated Circuits Conference*, 1999.

[22] V. Betz and J. Rose. FPGA routing architecture: segmentation and buffering to optimize speed and density. *In proceedings of the International Symposium on Field-Programmable Gate Arrays*, 1999.

[23] C. Bobda and A. Ahmadinia. Dynamic interconnection of reconfigurable modules on reconfigurable devices. *In Design for Test, Volume 22, Issue 5,*, 2005.

[24] W. Carter, K. Duong, R. H. Freeman, H. Hsieh, J. Y. Ja, J. E. Mahoney, L. T. Ngo, and S. L. Sze. A user programmable reconfigurable gate array. *In proceedings of the Custom Integrated Circuits Conference*, 1986.

[25] G. J. Chaitin, M. A. Auslander, A. K. Chandra, J. Cocke, M. E. Hopkins, and P. W. Markstein. Register allocation via coloring. *In Computer Languages, volume 6*, 1981.

[26] W. Chong, S. Ogata, M. Hariyama, and M. Kameyama. Architecture of a multi-context FPGA using reconfigurable context memory. *In Proceedings of the International Workshop on Parallel and Distributed Processing*, 2005.

[27] C. Clos. A study of non-blocking switching networks. *In Bell Systems Technology Journal, volume 32*, 1953.

[28] J. Cong, Y. Fan, G. Han, X. Yang, and Z. Zhang. Architecture and synthesis for multi-cycle on-chip communication. *In Proceedings of the International conference on Hardware/software codesign and system synthesis*, 2003.

[29] J. Cong, Y. Fan, and Z. Zhang. Architecture-level synthesis for automatic interconnect pipelining. *In proceedings of the Design Automation Conference*, 2004.

[30] J. Cong, J. Peck, and Y. Ding. RASP: A general logic synthesis system for SRAM-based FPGAs. *In proceedings of the International Symposium of Field-Programmable Gate Arrays*, 1996.

[31] Achronix Semiconductor Corporation. Speedster FPGA family, 2009.

[32] D. Cross, R. Drefenstedt, and J. Keller. Reduction of network cost and wiring in ranade's butterfly routing. *In Information Processing Letters, Volume 45 , Issue 2.*, 1993.

[33] W. J. Dally and B. Towles. Route packets, not wires: On-chip interconnection networks. *In proceedings of the Design Automation Conference*, 2001.

[34] W. J. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. The Morgan Kaufmann Series in Computer Architecture and Design, 2004.

[35] H. Fan, J. Liu, Y.L. Wu, and C.C. Cheung. On optimum designs of universal switch blocks. *In proceedings of the International Conference on Field Programmable Logic and Applications*, 2002.

[36] R. Francis and S. Moore. Exploring hard and soft networks-on-chip for FPGAs. *In proceedings of the International Symposium on Field Programmable Technology*, 2008.

[37] R. Francis and S. Moore. FPGAs with time-division multiplexed wiring: an architectural exploration and area analysis. *In proceedings of the International Symposium on Field-Programmable Gate Arrays*, 2009.

[38] R. Francis, S. Moore, and R. Mullins. A network of time-division multiplexed wiring for FPGAs. *In proceedings of the International Symposium on Networks-on-Chip*, 2008.

[39] R. Francis, J. Rose, and Z. Vranesic. Chortle-crf: Fast technology mapping for lookup table-based FPGAs. *In proceedings of the International Conference on Design Automation*, 1991.

[40] E.A. Garcia and C-P. Chiu. Two-resistor compact modeling for multiple die and multi-chip packages. *In proceedings of the Symposium of Semiconductor Thermal Measurement and Management*, 2005.

[41] K. Goossens, M. Bennebroek, J. Y. Hur, and M. A. Wahlah. Hardwired networks on chip in FPGAs to unify functional and configuration interconnects. *In proceedings of the International Symposium on Networks-on-Chip*, 2008.

[42] D. Grant and G. Lemieux. A spatial computing architecture for implementing computational circuits. *In proceedings of the Conference on Microsystems and Nanoelectronics Research*, 2008.

[43] J. He and J. Rose. Advantages of heterogeneous logic block architectures for FPGAs. *In proceedings of the Custom Integrated Circuits Conference*, 1993.

[44] R. Hecht, S. Kubisch, A. Herrholtz, and D. Timmermann. Dynamic reconfiguration with hardwired networks-on-chip on future FPGAs. *In proceedings of the International Conference on Field Programmable Logic and Applications*, 2005.

[45] C. H. Ho, C. W. Yu, P. H. W. Leong, W. Luk, and S. J. E. Wilton. Domain-specific hybrid FPGA: Architecture and floating point applications. *In proceedings of the International Conference on Field Programmable Logic and Applications*, 2007.

[46] W. K. C. Ho and S. J. E. Wilton. Logical-to-physical memory mapping for FPGAs with dual-port embedded arrays. *In proceedings of the International Workshop on Field Programmable Logic and Applications*, 1999.

[47] M. D. Hutton, J. Rose, and D. G. Corneil. Automatic generation of synthetic sequential benchmark circuits. *In Transactions on CAD of Integrated Circuits and Systems, Volume 21, Issue 8*, 2002.

[48] P. Jamieson and J. Rose. A verilog RTL synthesis tool for heterogeneous FPGAs. *In proceedings of the International Conference on Field Programmable Logic and Applications*, 2005.

[49] P. Jamieson and J. Rose. Architecting hard crossbars on FPGAs and increasing their area-efficiency with shadow clusters. *In proceedings of the International Conference on Field Programmable Technology*, 2007.

[50] N. Kaneko and H. Amano. A general hardware design model for multicontext FPGAs. *In proceedings of the Conference on Field Programmable Logic and Applications*, 2002.

[51] N. Kapre, N. Mehta, M. DeLorimier, R. Rubin, H. Barnor, M. J. Wilson, M. Wrighton, and A. DeHon. Packet switched vs. time multiplexed FPGA overlay networks. *In proceedings of the Symposium on Field-Programmable Custom Computing Machines*, 2006.

[52] I. Kuon and J. Rose. Measuring the gap between FPGAs and ASICs. *In proceedings of the International Symposium on Field-Programmable Gate Arrays*, 2006.

[53] J. Lamoureux and S. J. E. Wilton. FPGA clock network architecture: flexibility vs. area and power. *In proceedings of the International Symposium on Field-Programmable Gate Arrays*, pages 101–108, 2006.

[54] M. LaPedus. Open-silicon to drive down mask costs. *EE Times*, 2007.

[55] G. Lemieux, E. Lee, M. Tom, and A. Yu. Directional and single-driver wires in FPGA interconnect. *In proceedings of the International Conference on Field-Programmable Technology*, 2004.

[56] D. Lewis, E. Ahmed, G. Baeckler, V. Betz, M. Bourgeault, D. Cashman, D. Galloway, M. Hutton, C. Lane, A. Lee, P. Leventis, S. Marquardt, C. McClintock, K. Padalia, B. Pedersen, G. Powell, B. Ratchev, S. Reddy, J. Schleicher, K. Stevens, R. Yuan, R. Cliff, and J. Rose. The stratix II logic and routing architecture. *In proceedings of the International Symposium on Field-Programmable Gate Arrays*, 2005.

[57] D. Lewis, V. Betz, D. Jefferson, A. Lee, C. Lane, P. Leventis, S. Marquardt, C. McClintock, B. Pedersenand G. Powell, S. Reddy, C. Wysocki, R. Cliff, and J. Rose. The Stratix routing and logic architecture,. *In proceedings of the International Symposium on Field-Programmable Gate Arrays*, 2003.

[58] C-C. Lin, D. Chang, Y-L. Wu, , and M. Marek-Sadowska. Time-multiplexed routing resources for FPGA design. *In proceedings of the International Workshop on Field-Programmable Gate Arrays*, 1996.

[59] T. Marescaux, J-Y. Mignolet, A. Bartic, W. Moffat, D. Verkest, S. Vernalde, and R. Lauwereins. Networks on chip as hardware components of an os for reconfigurable systems. *In proceedings of the International Conference on Field Programmable Logic and Applications*, 2003.

[60] M. I. Masud and S. J. E. Wilton. A new switch block for segmented FPGAs. *In Proceedings of the International Workshop on Field-Programmable Logic and Applications*, 1999.

[61] G. Moore. Cramming more components onto integrated circuits. *In Electronics, Volume 38*, 1965.

[62] A. Mtibaa, B. Ouni, and M. Abid. An efficient list scheduling algorithm for time placement problem. *In Computers and Electrical Engineering, Volume 33, Issue 4*, 2007.

[63] R. Mullins, A. West, and S. Moore. The design and implementation of a low-latency on-chip network. *In proceedings of the Asia South Pacific Design Automation Conference*, 2006.

[64] P. G. Paulin and J. P. Knight. Force-directed scheduling for the behavioral synthesis of asics. *In Transactions on Computer-Aided Design, volume 8, issue 6*, 1989.

[65] L-S. Peh and W. J. Dally. A delay model and speculative architecture for pipelined routers. *In proceedings of the International Symposium on High-Performance Computer Architecture*, 2001.

[66] T. Pionteck, R. Koch, and C. Albrecht. Applying partial reconfiguration to networks-on-chips. *In proceedings of the International Conference on Field Programmable Logic and Applications*, 2006.

[67] Hobson R. F. Compact multiport static random access memory cell. *US. Patent no. 5754468*, 1998.

[68] J. Rose and S. Brown. Flexibility of interconnection structures for field programmable gate arrays. *In Solid-State Circuits, Volume 26, Issue 3*, 1991.

[69] M. Salda, L. Shannon, and P. Chow. The routability of multiprocessor network topologies in FPGAs. *In proceedings of the International workshop on System-level interconnect prediction*, 2006.

[70] M. Santarini. Xilinx tops ranks of FPGA and EDA vendors. *EE Times*, 2004.

[71] G. Schelle and D. Grunwald. Onchip interconnect exploration for multicore processors utilizing FPGAs. *In proceedings of the International Workshop on Architecture Research using FPGA Platforms*, 2006.

[72] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli. SIS: A system for sequential circuit synthesis. Technical report, Tech Report, University of California, Berkeley, 1992.

[73] A. Sharma, K. Compton, C. Ebeling, and S. Hauck. Exploration of pipelined FPGA interconnect structures. *In proceedings of the International Symposium on Field-Programmable Gate Arrays*, 2004.

[74] A. Singh, A. Mukherjee, and M. Marek-Sadowska. Interconnect pipelining in a throughput-intensive FPGA architecture. *In proceedings of the International Symposium on Field-Programmable Gate Arrays*, 2001.

[75] S. Sivaswamy, G. Wang, C. Ababei, K. Bazargan, R. Kastner, and E. Bozorgzadeh. HARP: hard-wired routing pattern FPGAs. *In proceedings of the International Symposium on Field-Programmable Gate Arrays*, 2005.

[76] S. Trimberger. Scheduling designs into a time-multiplexed FPGA. *In proceedings of the International Symposium on Field-Programmable Gate Arrays*, 1998.

[77] S. Trimberger, D. Carberry, A. Johnson, and J. Wong. A time-multiplexed FPGA. *In proceedings of the Symposium on Field-Programmable Custom Computing Machines*, 1997.

[78] S. M. Trimberger and A. H. Lesea. Programmable logic device with time-multiplexed interconnect. *US Patent No. 6829756*, 2004.

[79] S. M. Trimberger and A. H. Lesea. FPGA with time-multiplexed interconnect. *US Patent No. 7268581*, 2007.

[80] T. Tuan and B. Lai. Leakage power analysis of a 90nm FPGA. *In proceedings of the Custom Integrated Circuits Conference*, 2002.

[81] W. F. J. Verhaegh, E. H. L. Aarts, J. H. M. Korst, and P. E. R. Lippens. Improved force-directed scheduling. *In proceedings of the conference on European design automation*, 1991.

[82] S. J. E. Wilton. *Architectures and algorithms for field-programmable gate arrays with embedded memory*. PhD thesis, University of Toronto, 1997. Adviser-Jonathan Rose and Adviser-Zvenko Vranesic.

[83] www.ITRS.net, 2007.

[84] A. Yan, R. Cheng, and S. J. E. Wilton. On the sensitivity of FPGA architectural conclusions to experimental assumptions, tools, and techniques. *In proceedings of the International Symposium on Field-Programmable Gate Arrays*, 2002.

[85] S. Yang. Logic synthesis and optimization benchmarks user guide version 3.0. Technical report, Tech. Report, Microelectronics Center of North Carolina, 1991.

[86] A. Ye and J. Rose. Using bus-based connections to improve field-programmable gate array density for implementing datapath circuits. *In proceedings of the International Symposium on Field-Programmable Gate Arrays*, 2005.