Social Insect-Inspired Adaptive Hardware

Matthew Rowlings

Ph.D.

University of York

Electronic Engineering

September 2018

Abstract

Modern VLSI transistor densities allow large systems to be implemented within a single chip. As technologies get smaller, fundamental limits of silicon devices are reached resulting in lower design yields and post-deployment failures. Many-core systems provide a platform for leveraging the computing resource on offer by deep sub-micron technologies and also offer high-level capabilities for mitigating the issues with small feature sizes. However, designing for many-core systems that can adapt to in-field failures and operation variability requires an extremely large multi-objective optimisation space. When a many-core reaches the size supported by the densities of modern technologies (thousands of processing cores), finding design solutions in this problem space becomes extremely difficult.

Many biological systems show properties that are adaptive and scalable. This thesis proposes a self-optimising and adaptive, yet scalable, design approach for many-core based on the emergent behaviours of social-insect colonies. In these colonies there are many thousands of individuals with low intelligence who contribute, without any centralised control, to complete a wide range of tasks to build and maintain the colony. The experiments presented translate biological models of social-insect intelligence into simple embedded intelligence circuits. These circuits sense low-level system events and use this manage the parameters of the many-core's Network-on-Chip (NoC) during runtime.

Centurion, a 128-node many-core, was created to investigate these models at large scale in hardware. The results show that, by monitoring a small number of signals within each NoC router, task allocation emerges from the social-insect intelligence models that can self-configure to support representative applications. It is demonstrated that emergent task allocation supports fault tolerance with no extra hardware overhead. The response-threshold decision making circuitry uses a negligible amount of hardware resources relative to the size of the many-core and is an ideal technology for implementing embedded intelligence for system runtime management of large-complexity single-chip systems.

Contents

List of Figures

List of Tables

Acknowledgements

Many thanks to my supervisors Dr. Martin Trefzer and Prof. Andy Tyrrell for all of their support, guidance and many interesting discussions throughout the duration of the PhD process. I'd also like to thank all of my family for their understanding and patience when I kept talking about ants through all of these years. Likewise to the friends in the Department who have provided many enjoyable pub and climbing sessions during my time at York.

This work would not have been possible without the continuous dedication, assistance and encouragement provided by the SCR coffee machine. And also by Sarah. Thank you to you both.

Declaration

I declare that this thesis is a presentation of original work and I am the sole author. This work has not previously been presented for an award at this, or any other, University. All sources are acknowledged as References. The research presented in this thesis features in a number of the author's publications listed below:

- Matthew Rowlings, Andy M. Tyrrell, Martin A. Trefzer. *Embedded Social Insect-Inspired Intelligence Networks for System-level Runtime Management* in: *Design Automation and Test in Europe (DATE)*, March 2020
- Matthew Rowlings, Andy M. Tyrrell, Martin A. Trefzer. *Hardware Implementation of Social-Insect-Inspired Adaptive Many-Core Task Allocation* in: *IEEE Symposium Series on Computational Intelligence (SSCI)*, December 2016
- Matthew Rowlings, Andy M. Tyrrell, Martin A. Trefzer. *Hardware Implementation of Social-Insect-Inspired Adaptive Many-Core Task Allocation* in: *IEEE Symposium Series on Computational Intelligence (SSCI)*, December 2016
- Matthew Rowlings, Andy M. Tyrrell, Martin A. Trefzer. *Social-Insect-Inspired Adaptive Task Allocation for Many-Core Systems* in: *IEEE World Congress on Computational Intelligence (WCCI 2016)*, July 2016
- Matthew Rowlings, Andy M. Tyrrell, Martin A. Trefzer. *Social-Insect-Inspired Networking for Autonomous Fault Tolerance* in: *IEEE Symposium Series on Computational Intelligence (SSCI)*, December 2015
- Matthew Rowlings, Andy M. Tyrrell, Martin A. Trefzer. *Social-insect-inspired networking for autonomous load optimisation* in: *Procedia CIRP 38*, pages 259-264

Chapter 1

Introduction

1.1. Overview

The advancement of silicon implementation technologies has recently reached a stage in its development where integrated circuit performance and cost has revolutionised the use of, capabilities of and number of electronic devices that we use during our day-to-day lives. The improvements in transistor densities, switching frequencies and lower power consumption of integrated circuits over the last decade now allows us to use complex embedded systems in more and more applications; from advanced medical devices to the high-power embedded computing requirements of autonomous vehicles.

However, as state-of-the-art device technologies come ever-closer to the atomic scale, challenges intrinsic to deep sub-micron fabrication emerge [1][2]. Engineers are required to include large design tolerances to ensure that their designs will not fail after it leaves the manufacturing line, confounded by the device variability challenges that mean a design that works comfortably within specifications on one silicon die may not work on the die created on the next wafer [3]. Unfortunately as feature sizes get smaller, these problems are only going to be of more importance as design margins tighten and very small scale variations of the silicon wafer become the overriding concern [4][5]. Thus the fundamental unit of the digital designer's toolkit, the transistor, is becoming less and less reliable with more variation in its key behavioural properties.

Whilst engineering suffers from such unreliable and varied fundamental blocks, some of the great feats of natural systems have been found to be due to inherent variability and unreliability within the building-blocks of biological systems [6] [7] [8] [9] [10]. There are clearly some value for engineering systems in such processes that biology has developed to overcome and exploit these low level problems. With a suitable model extracted and translated from particular biological processes, there is no reason that engineered systems could not also adapt to variation and cope with faults in their constituent parts.

1.1.1. Many-Core Systems

Hardware co-processors typically provide a performance improvement as well as reduced power (and thus thermal) load compared to a general propose processor implementation. The hardware co-processor model also scales well with parallel problems and so has been used heavily in high performance, embarrassingly parallel applications such as graphics pipelines. Many-core systems [11][12] aim to take this approach to an architectural level whereby lots of heterogeneous hardware accelerators (or processing cores with hardware accelerators attached) are connected together with levels of specialism such that the resulting task architectures distributes thermal and power load across the die and reducing the occurrence of processing “hotspots”.

As well as Dark Silicon [13][14], many-cores can also be used to tackle the issue of device variability and post-manufacture failures through the use of the classic fault-tolerance technique N-modular redundancy. With several copies of the same co-processor within the many-core architecture, it is easily seen that the tasks undertaken by a under-performing or failed core (due to device variation or a post-manufacture fault due to device degradation) can be undertaken by another co-processor of the same type located at a different geometric area of the die that may not suffer from the same inherent silicon characteristics.

1.1.2. Many-Core Design Challenges

Whilst many-core architectures seem well poised to tackle the previously discussed problems, they also bring a large number of different complex issues when designing systems for such an architecture. Many-core architectures tend to be organised in a grid-like fashion on the die with a Network-on-Chip (NoC) used to interconnect the co-processors, also known as Processing Elements (PE). A NoC has many similarities with standard computer networking but tends to be simpler with more deterministic workings. Starting from a grid of PEs, a many-core designer would need to determine:
1) The task (or co-processor) that each PE is specialised for

- The thermal/power budget of this task, which may limit what tasks the adjacent

PEs can undertake

- The expected data throughput of this task, including the thermal requirements on the NoC local to the PE
- A list of tasks/PEs that this node will need to communicate with before, during or after data processing, including specific throughput requirements

2) A routing strategy that connect tasks that need to communicate together, with throughput and thermal requirements on the NoC paths met

- Analysis of alternative/redundant routing paths and how these affect the calculation of each data-transfer pathway in the task graph
- Choice of routing protocol. This will impact the routing paths, some protocols require significant buffering or less deterministic routing strategies
- Analysis, prediction and elimination of any deadlock or livelock scenarios within the chosen NoC routing strategy

3) Consider fault scenarios that will change the balance of tasks in the many-core due to PE failure or the performance of the NoC (e.g. a link that may fail, or a link that has to reduce throughput due to unexpected thermal environment)

- This will require re-analysis of all the previous parameters for the different fault scenarios or environmental changes

This is a huge amount of analysis that scales exponentially with not only the number of nodes in the many-core, but also with the number of tasks mapped to it. If many-core is going to be the platform that allows us to fully utilise future silicon devices then not only must it support a huge number of processing elements, but we must also be able to use it in a multi-application environment. Despite these problems, we see some aspects of many-core design already being used in embedded systems [15], and it is a active research field with lots of outstanding challenges [16][12].

1.1.3. Bio-Inspired Hardware

The huge number of interacting design parameters of a many-core system has many similarities with complex systems found within biology. Natural optimisation methods such as evolution and development have emerged as biology's way of navigating very high-dimensional pareto-front tradeoffs. Using biological inspiration in electronic systems design has been a mature research area for many years now. Despite this, we have not seen an uptake of using biologically inspired models to solve problems in hardware design for the commercial electronics world. There are many reasons for this gap, but the following are some of the main factors:

- Cost of implementing the biological model in hardware is high and non-scalable [17]
- An advance in technology has rendered the problem the biological inspiration solves as redundant [18]
- The biological model is difficult to validate against a set of requirements [19]
- Difficulty integrating the systems application with the biological model [20] [21]

As designing for many-core systems allows a degree of abstraction away from the digital substrate with the design challenges at a much higher level, this opens the possibility of a different bio-inspired approach than the traditional ”building-blocks” up approach seen in many bio-inspired research projects. The model of a processing element communicating with other processing elements via the NoC opens up a link with many highly-scalable biological models.

1.1.4. Social Insect Intelligence

Large social insect colonies also require a wide range of important tasks to be undertaken to build and maintain the colony and in most nests there are many thousands of workers available to offer their assistance to ensure the expansion and survival of the colony. However, there is a crucial equilibrium between the number of workers performing each task that must not only be maintained but must also continuously adapt to sudden changes in environment and colony need. What is most fascinating is that social insects can sustain this balance without any centralised control and with colony members that have relatively little intelligence when considered on their own. Due to this simplicity and evident scalability it would seem that social insects have evolved an interesting scalable approach to task allocation that could be applied to very large many-core systems.

When evaluating a bio-inspired hardware system there are several implementation possibilities and abstractions to chose from. In this work, relevancy to near-future hardware platforms was considered a priority. Device densities have reached a stage where large amounts of computing power are available per chip, this is reflected in the recent rise of high-performance consumer embedded systems (e.g. smartphones [22]). Therefore, it is felt that applicability to modern systems is highly important. This motivates the creation of a large-scale many-core system as part of this work. The emphasis on applicability to hardware systems also drives the implementation and evaluation of the biological models. These models could be evaluated using simulation models, starting from biologically "true" models and refining them such that they can be applied to the hardware system. However there are two reasons for not following this route:

(a) The biological models for social-insect intelligence are relatively new, a consolidated model is not available [23] and the research into these models is active. Modelling techniques used for the biological models that this work takes inspiration from are not coherent across the models, with Kauffman networks [24], oscillatory dynamics models [25] and Markov chains [26] being used. Further work would be required in this field to use a direct implementation of the biological models as a starting point for the embedded intelligence.

(b) Simulation of the biological models would also require simulation of the many-core system. One millisecond of a behavioural simulation of a 128-core Centurion takes around 1 hour to complete and so running an experiment of one second (as used in the experiments in this thesis) would have taken a large number of days to complete. As each experiment consists of averaging from a large number (100 in most experiments) of initial parameters, experiments would take an infeasible amount of time. Therefore an abstraction of the many-core would also be required for simulation. Architecture of efficient large-scale processor simulations abstract the cores to tasks running on a CPU or cluster (for example [27]). This requires precise modelling of communication structures between nodes (critical due to the use of wormhole routing between nodes) and also a modelling of time to allow frequency scaling of cores to be supported. The intended low level hardware signals that the embedded intelligence hooks into will need to be extracted, abstracted and then integrated into the biological simulation model. Understanding how to achieve this becomes more difficult as the many-core abstraction level increases, which is a direct counter to the need of a high node abstraction level for simulation scalability purposes. Modelling of the NoC could also help with gaining accuracy in the simulation model, at the cost of adding overhead to the simulation time. There are many NoC developments and simulation models [28]. When the constraints of proven large scalability and a hardware implementation are applied to a survey of existing NoC developments [28], only three NoCs out of the 60 evaluated had both been implemented for FPGA or ASIC (to ensure applicability to near-future technologies) and supported over 50 processing cores (to prove scalability).

Therefore, in this work simulation is not used as the primary evaluation method. All inputs and outputs of the embedded intelligence interact directly with hardware signals. This has the benefit of:

- A shorter experiment runtime, an experiment of one second only takes one second to run and around 250ms to collect the data and set up the next experiment run.
- No issues with simulation-reality gaps or need to prove correctness of a simulation model to the target hardware platform that it is simulating.
- A hardware-suitable implementation of the biological model is enforced that does not require complex operations that are hardware-resource expensive (e.g. large matrices, solving of differential equations, floating point arithmetic).
- Direct manipulation of low-level hardware signals. This is inherently a hardware-resource efficient approach, allowing support of the largest number of cores pos-

sible.

- A hardware platform with unique access to low-level many-core signals that can be used be other many-core research experiments without the effort of configuring, deploying and debugging a NoC that is not designed for low-level manipulation of its control signals.

Smaller simulations were used during development of the models using behavioural simulation of a 24 core Centurion, however the results of these not presented as the intelligence models developed for them are used in the hardware experiments that are presented in the thesis.

1.2. Hypotheses

The work presented in this thesis establishes how many-core systems can be made more robust by using emergent self-organisation based on the decision pathways of individuals of social-insect colonies. The intelligence is ensured to be hardware efficient by taking inspiration from a simple biological model and through research study using a large-scale hardware platform typical of near-future many-core systems.

In this thesis I propose and evaluate a new source of biological inspiration for digital electronic systems based on the following hypothesis:
Hypothesis: *Embedded social insect intelligence models derived from studies of the social insects can exhibit highly-scalable adaptive behaviours suitable for managing complex digital electronic systems.*

With the following supporting sub-hypotheses:
Sub Hypothesis 1: *Models of task allocation in social insect colonies provide appropriate inspiration for enabling self-organising task mappings for many-core systems.*
Sub Hypothesis 2: *The decision making processes of individual social insects can inspire embedded intelligence circuitry that allow run-time self-optimisation of digital sub-systems to be achieved.*

1.3. Research Contributions

During the undertaking of this work the following research contributions have been made to the field of digital electronics:

- The creation of *Centurion* many-core system a 128-node many-core platform for experimenting with large scale many-core systems in hardware.
- The design of the *Configurable Intelligence Array*, a novel artificial intelligence hardware platform based on simplified aspects of spiking neural networks but highly optimised for efficient FPGA and digital circuit implementations.
- Selection and Evaluation of two reference task allocation models from the biological literature and understanding of their strengths and weaknesses when applied in an artificial emergent environment.
- Methodologies for converting response-threshold based artificial intelligent behaviours into forms that are suitable for embedding in modern digital systems.

1.4. Thesis Structure

This thesis consists of eight chapters and is structured as follows:

- Chapter 2 gives an outline of the trends of modern VLSI design and summarises the problems that are expected to arise in the medium term that could spell the end of the device size and performance scaling that we have been accustomed to over the last few decades. It also introduces many-core systems and autonomic computing as potential design solutions to these problems.
- Chapter 3 explores the biological literature to gain understanding on how social insect colonies are organised and describes the key biological models created to capture the adaptive properties of large social insect colonies.
- Chapter 4 introduces the *Centurion* 128-node many-core platform designed for the experiments presented in this thesis and describes its FPGA implementation.

- Chapter 5 is a second implementation chapter that covers the design of the *Configurable Intelligence Array*, which is then embedded within Centurion and used to implement the social insect-inspired models.
- Chapter 6 presents the results of experiments that validate the emergent behaviour of the social insect intelligence when used to control aspects of the Centurion platform.
- Chapter 7 expands on the adaptive behaviour goals of investigated in Chapter 6 with a fault injection experiment and shows how adaptive behaviours can be implemented using the Configurable Intelligence Array.
- Chapter 8 reviews the social-insects as a source of bio-inspiration for adaptive systems and suggests improvements and further work that could be undertaken with the models, Centurion and the Configurable Intelligence Array.

Chapter 2

Challenges of Large Scale VLSI Design

2.1. Overview

The continuous advancement of semiconductor based technologies has enabled many digital electronic engineering paradigms as transistor densities and power consumption improved over time with each new technology. Today, this allows digital engineers to build complex *Systems-on-Chips (SoCs)*, comprised of hundreds of digital sub-systems, all integrated together into one silicon die. It has been a relatively fast-paced transformation period, driven by the unrelenting Moore's law [29], that has seen large shifts in digital system design methodology as each step forward allowed designers to use a higher level of complexity and abstraction. This has resulted in engineers advancing from a transistor-to-transistor design based on hand drawn circuit diagrams, to the specification description based EDA tool flows that we enjoy today, within only a few decades.

However, this advancement of digital hardware has been focused sharply on silicon-based transistors and these have fundamental physical properties (e.g. leakage current [30], heat dissipation) that at certain technology sizes will start to affect how many transistors can be integrated into a chip. In general the smaller the transistor the more such properties become a fundamental limitation to the density and performance of silicon-based transistors, resulting in our classical route of technological advancements eventually grinding to a halt. Dennard Scaling [31] was the first of these properties to fail and has forced a change in the design of complex VLSI systems, with many other limitations likely to be hit in the near future if digital hardware with silicon-based transistors prevails . Modern design paradigms such as *Many-Core Systems* built on *Networks-on-Chip (NoCs)* promise new ways of overcoming imminent limitations, but with the penalty of an extremely costly (indeed almost prohibitive) increase in design complexity.

This chapter will introduce the problems that current and near-future deep sub-micron VLSI technologies will need to overcome to realise the next generation of high-performance digital electronic systems. Hardware implementation device trends and challenges are explored in Section 2.2. Many-Core Systems and their design challenges are then introduced in Section 2.3. Finally, Section 2.4 will introduce the *Autonomic Computing* paradigm as a means of tackling these challenges.

2.2. Present and Upcoming Challenges in VLSI Devices

2.2.1. Future Technology Trends

As we start to reach silicon's fundamental physical and performance limits, research into the next-generation digital device technologies has focussed on either finding a replacement for silicon as an implementation technology or using the silicon fabric in novel ways to extract more performance per device. This is a broad and speculative field and so the 2015 reporting of the *International Technology Roadmap for Semiconductors (ITRS)*[32] and the 2018 reporting of the follow up group *IEEE International Roadmap For Devices and Systems (IRDS)* [33] provide a useful summary of the medium-term challenges that the technology presented in this thesis aims to address.

The ITRS and IRDS reports are produced to detail upcoming challenges in the semiconductor industry and are prepared by representative experts from several industrial bodies including: European Semiconductor Industry Association, Japan Electronics and Information Technology Industries Association, Korean Semiconductor Industry Association, Semiconductor Industry Association and Taiwan Semiconductor Industry Association [32]. Therefore it can be considered a reliable indicator of the problems that the semiconductor industry feel are most important to tackle. The ITRS/IRDS reports consider a wide range of consumer application uses: from high-performance data centre use, low-power IoT applications and also requirements of future mobile devices that strike a tough balance between power and embedded high-performance [34] [1]. Most of the sub-reportings summarise that in the medium term many electronics fields will require either a replacement for CMOS technologies or use of CMOS technologies in advanced ways. They predict that the growth of use cases and demand for ever more computation power will continue for at least the next 20 years and so these medium term challenges are likely to be required and will drive research directions for the foreseeable future.

The following summaries explore some of the most pressing challenges for digital VLSI systems.

2.2.2. Dennard Scaling and Dark-Silicon

Recently we have reached a fundamental transistor size where Dennard scaling starts to break down [13]. Previously the threshold voltage of the transistors on the die would scale together with the supply voltage, allowing the supply voltage to drop with each process generation. However, now the supply or threshold voltage cannot be dropped without simultaneously increasing either the transistor delay or leakage [11], forcing the supply voltage through each recent process generation to be fixed. This has dire consequences when combined with the scaling in transistor density as the power required by the chip will increase exponentially with each generation, to the point that we cannot provide the power required to switch all the transistors on a chip at their maximum frequency or even remove the thermal energy produced by this switching [35].

This limitation has been dubbed *Dark Silicon* and has been highlighted as a crucial problem for the semiconductor industries, with predictions claiming that at an 8nm process over 50% of a chip may need to be powered off to act as thermal buffering [13]. Further analysis in [36] suggests that this figure can be reduced with DVFS management. Their experiments show that between features sizes and, even with a 40% Dark Silicon overhead at 8nm, the performance benefits of a smaller feature size outweigh the Dark Silicon overhead when overall system performance is considered. These results however are dependant on a thermal analysis and use of DFVS, this will be application specific and requires extra design analysis and runtime management which may not scale to very large systems. Designs are starting to use 7nm processes and the evaluation presented in [37] implemented a 64-core design using a 7nm FinFET technology. The results aligned with the predictions of [13], using high-performance FinFETs (Super-Threshold) and a 15W power budget it was found that the system had a Dark Silicon overhead of 64% (i.e. only 36% of the design cores could be run at full speed at once). If lower performance, but more power-efficient, Near Threshold FinFETs were used then this overhead improved to 19%.

Consequently, several research directions have emerged to tackle the Dark Silicon problem, of which many-core systems are one of them [11][12]. Four possible directions for Dark Silicon mitigation have been outlined in [14]:

1. *Smaller dies:* By reducing the number of transistors per die the overall power and thermal requirements of the die can be reduced. However this approach neglects the fact that the area of the die is a fundamental requirement for heat dissipation performance and so the frequency of operation is then limited, increasingly an issue as transistor density (and therefore the intensity of hotspots) increases. This approach is also restricted in cost savings as the design costs will remain largely the same, if not increasing due to the need to fit the design into a smaller area. This could even require several chips to achieve the required functionality, practically reversing the trend to fit complex systems within on chip with the effect of increasing system power consumption and unnecessary overheads such as greater off-chip communication channels (requiring more IO pads and more complex PCB layouts).

2. *Part-time logic:* Whilst Dark Silicon is important for maintaining power and thermal constraints, it should be emphasised that the silicon does not have to be completely unused all the time and can instead be used for circuitry that is of either low frequency operation or of infrequent use. Through the use of techniques such as clock gating and Dynamic Voltage and Frequency Scaling (DVFS), the energy consumption of specific parts of the chip can be controlled. This has seen some application in modern multi-cores as a “turbo” function where the frequency of some cores is scaled back to allow another core to run at a higher frequency. By careful scheduling of the application or scheduling the use of DVFS it is also possible to move some of the problem to the temporal domain; parts of the system are run at full performance until the thermal capacity of the chip is reached at which point the performance is reduced until the chip has had time to cool. This technique also favours assigning large parts of the chip to memories such as caches, as only a small subset of the memory is accessed (and therefore active) at any one time giving a high “darkness” per square millimetre when compared to computational logic despite being operated at its maximum

frequency. However few applications will benefit from just a de-facto increase in cache resources so a vast amount of silicon will have to be left dark and is thus wasted.

3. *Specialised co-processors:* An elaboration of the previous technique is to utilise the dark silicon for specialised co-processors. The co-processors consist of specialised hardware accelerators that perform a specific operation with far greater performance and energy efficiency than the general purpose processor. They are also clock gated and may support DVFS, allowing them to be disabled when not in use. Execution switches between a general purpose processor and the co-processors (possibly many copies of the same for a parallel application) in such a way that the most efficient core for a particular operation is always used. Unused cores are switched off resulting in a reduction in total switching capacitance for a particular operation, this is in contrast with the previous approach where the aim is to minimise energy use through frequency and voltage management. This approach makes full use of unused dark silicon and it can be envisioned that systems will consist of hundreds or thousands of tiny co-processors residing in otherwise wasted silicon resources. Some care will be required at the design stage to ensure that energy savings are not compromised by inefficient communication between the cores and new programming tools will be required to shield the underlying complexity of the many-core away from the programmer without jeopardising the potentially great improvements offered by co-processor driven architectures.

4. *New material or silicon breakthrough:* The final hope for combating dark silicon comes from changes in the silicon platform itself, a possible move is away from MOSFET technologies due to their fundamental limits on leakage current at room temperature. Two candidate technologies highlighted in [14] are tunnel-FETs (TFETs) and Nanoelectromechanical system (NEMS) switches, both of which offer significantly better leakage performance. However these processes also have their limitations: TFETs are unsuited to be applied to high performance circuitry due to lower on-currents and NEMS technology suffers from slow switching times. This limits their immediate integration into chips, but

they are a key focus of future semiconductor fabrication research and may indeed find niche applications.

From this survey it is clear that Dark Silicon is going to be a significant problem that we must overcome if we are still going to reap the benefits of increasing transistor density. Unless new materials or breakthroughs at the physical layer are made, it is almost certainly going to fundamentally change the way we design complex digital systems; indeed with the adaptation of many-core it could be said that this change has already started to happen. Out of the above directions, it seems that the first is not only rather bleak in terms of future innovation but it is also not particularly scalable or future proof. It will raise the cost of systems, introduce longer engineering times and limit the amount of complexity we can achieve within a system.

In contrast, the fourth direction is the most encouraging in terms of a more holistic solution of the issues regarding dark silicon and indeed the ITRS highlights novel fabrication substrates such as graphene and spin materials as of significant research interest in their *Emerging Research Materials* [38] and *Emerging Research Devices* [39] summaries from 2013. Alternative silicon based technologies are explored in the *Process Integration, Devices and Structures* summary (also from 2013) [40], with a focus on sub 10nm technologies and three-dimensional architectures; but with also a strong focus on improving reliability and other fabrication scaling issues. The 2018 IRDS Metrology report [41] shows that the field has evolved slightly with the challenges focussing on both sub-7nm technology and also larger feature sizes. Implementation of 3D structures (e.g. finFETs) is a key element of the report and so a large focus of this report covers device measurement and feature implementation verification for 3D structures. This shows that whilst building these structures at feature sizes of 10nm is feasible, managing and verifying yield is still a large challenge despite introduction of new fundamental structures. The report also introduces the idea that research into AI predictions of yield using models of a process combined with process parameters and measurement points could be used to predict die measurements that cannot be taken for scale reasons.

The broad range of research directions suggested by the ITRS (shown in the *Executive Summary* of all ITRS assessments [34] [1]) suggests that it is not clear what silicon's replacement will be or indeed when it will disrupt silicon's dominance. Therefore, the principal method for a medium-term approach to tackling dark silicon should be through the second and third research directions [42].

2.2.3. Device Variability

This problem is exaggerated when another major problem affecting modern semiconductors is considered, namely fabrication related issues such as *process variability* and *manufacturing defects*. During fabrication it is impossible to keep the physical characteristics of all transistors and interconnections consistent across the chip and so some local variation in performance occurs. As the size of transistors continuously shrinks these margins become more significant to the point that it becomes inevitable that some parts of a chip will be out of stable operating range for a given operating environment whilst other parts may be well within their margins. No two dies are the same and so this variability is seen not only across the billions of transistors on the chip, but then again across the millions of chips produced in a manufacturing run. Therefore a detailed characterisation of the maximum operating parameters of each individual chip is not possible as this would require exhaustive testing (of all potentially critical paths) under a huge range of operating conditions to expose all possible faults.

Consequently the technique of *binning* [43] is now a common practice of the semiconductor manufacturers; each device is evaluated with a simple test depending on the operating feature required and then device guarantees are based on the performance of this test. For example a microprocessor manufacturer may bin their products based on the maximum frequency at which they can operate without faults, this may be well below the design maximum frequency of the device but it would allow a device to still be sold in spite of a lower performance; this is far better from a yield perspective than disposing of all under-performing chips. However, binning is clearly limited by the binning metric used to evaluate it and may still require very large reductions in performance of many devices in a production run to get a reasonable yield. Variability issues are only going to get worse with smaller fabrication processes; and so the reduction of performance may be so severely limiting that little of the advantage offered by a new process may actually be exploited by the device. Also the binning metric does not scale well with the many-core approach taken to combat dark silicon, in both terms of testing time and effectiveness of binning; with hundreds of cores on the chip it only takes one under-performing core to render the whole device destined for a poor device binning even if all other cores work perfectly well at their maximum performance, drastically limiting the yield of potential for high performing devices [3]. Thus it does not seem that binning is a sustainable approach to tackling variation for very large scale many-cores with thousands of processing elements.

2.2.4. Device Degradation and Ageing

A further challenge that has only relatively recently started to affect device yield is faults induced post-manufacture by *transistor ageing* and *electromigration*. These defects are surveyed in [44] (presented in an IEEE magazine), where three mechanisms for transistor ageing are identified: *hot-carrier injection* which increases the threshold voltage of the transistor (and thus limits the maximum switching speed); *bias temperature instability* which also increases the threshold voltage and *oxide breakdown* which will break down the transistor dielectric over time, eventually leading to a complete catastrophic failure of the transistor. Electromigration however occurs in the tracks between transistors: the accidental drifting of atoms with electrons causes the tracks to thin, increasing the resistance to the point that an open track may occur. As these permanent faults are impossible to mitigate through manipulation of controllable parameters (i.e. voltage, temperature), we will also need to mitigate ageing effects geometrically at runtime; analysis at time of manufacture is not longer sufficient. This has been done in the past with *N-modular redundancy*, where *N* copies of each processing unit exist on the device and their outputs are used to vote and thus can both detect faulty modules and provide a correct output. However the voter can still be subject to the permanent faults introduced here and if the processing unit is of high performance then the total power required by the redundancy scheme will be multiplied by *N*, with severe dark silicon implications. Thus, we shall either need to rely on cold spares which would be advantageous for dark silicon but could potentially be quite wasteful depending on system granularity, or utilise the adaptive nature of reconfigurable device technologies such as *Field Programmable Gate Arrays (FPGAs)*.

2.3. Many-Core Systems

To tackle these issues when designing for high-density silicon devices, we can change our design paradigm to manage the complexity of having so much computation on-chip whilst also managing co-dependent design constraints to ensure as much as the circuity on the device as possible is operating within design limits. As highlighted by the ITRS reports, a level of abstraction of interest is at the data-processor level and is encapsulated by the field of *Many-Core Systems*. These are large SoCs consisting of a high number of processing cores but implemented with a more generic architectures and processing cores than specialist high-core-count SoCs such as graphics cards or dedicated algorithm accelerators. The *More than Moore* 2018 IRDS report [2] predicts many-core systems as the only way to achieve performance targets assuming incremental improvement of the thermal and power density of dies. The report predicts a 4.0GHz maximum mobile processor speed by 2025 with this speed getting worse as technologies reduce in size (due to increased impact of parasitics), to the point where the average frequency of a CPU in a mobile SoC will be 700MHz by 2034. To counteract this the number of processing cores is increased, reaching 194 GPU cores and 170 CPU cores for a mobile SoC by 2034. A modern mobile SoC, for example the Apple A13 [22], consists of six CPU cores (two at 2.65GHz, four running significantly slower) and four GPU cores. The expansion of this SoC to the IRDS processing core densities will require a large shift in the design of software, hardware and run-time management to use the large number of processors effectively.

Many-core computing has been of research interest for many years within the parallel computation research field, a field that has steadily become more integrated with hardware research as the increasing transistor density allows more and more parallel computation to take place within a single chip. In an attempt to tackle engineering problems such as scalability, many parallel architectures have been proposed and evolved over the years including: systolic arrays[45], wavefront arrays [46], hypercubes [47] and stream processors [48]. However the prominent architecture for modern *on-chip* many-core systems is the *Network on Chip (NoC)* [49][50], an interconnection scheme based on conventional networking where routers and channels are provided for communication between nodes. A large number of node topologies, interconnect options and constraint optimisations are possible [16] (IEEE transactions on Computer-Aided Design of Integrated Circuits and Systems), giving the hardware engineer a powerful platform for implementing systems that could be adapted to meet various Dark Silicon criteria or tolerance to ageing effects via redundancy.

This flexibility comes with its own engineering caveats however, as the large number of parameters will require problem and system analysis to ensure that systems implemented within NoCs fit their requirements and may necessitate the need for heuristical approaches [51][52][53] to optimise the design space. This approach also suffers as such analysis is traditionally done at design time and so cannot be adapted should the operating conditions or properties of the chip change during operation (e.g. due to ageing effects). Whilst it would be possible to generate either a design that can operate over a range of operating environments or several different designs that can be switched in/out at run time, the multi-objective optimisation required for the system analysis tends to be computationally expensive. This would make it impossible to undertake this analysis online at runtime and would require a extensive amount of offline analysis; with the potential of storing a huge number of different configurations and parameters for all possible operating conditions that the system may be expected to work correctly for. The following sub-sections will explore this design space and show why managing these parameters is important for effective, scalable many-core design. Factors the influence the physical hardware implementation of a many-core (only Network-on-Chip implementations are considered in this thesis due to their ease of scalability) are discussed first and then the higher-level algorithmic design parameters and processes.

2.3.1. Network-on-Chip

Network-on-Chip (NoC) is a modern digital design paradigm enabled by the recent generation of high-density VLSI fabrics that supports a large number of data processing resources, using a network to interconnect the processing resources [49][50]. Whilst taking a large amount of inspiration from traditional networking, there are some significant design differences due to the fact that NoCs are implemented within a single chip. Network topology for example is more focused towards layouts that are better suited to tiling across a die using protocols and channels that require less storage overhead than current computer networking protocols and physical implementations. The authors in [16] (IEEE transactions on Computer-Aided Design of Integrated Circuits and Systems) have conducted a large survey of NoC design techniques and highlighted problems that need attention, whilst the authors of [54] (ACM Computing Surveys) and also in [55] summarise key design points in NoC design and discuss some example designs. The following summaries use these papers to provide a brief explanation of the role of each component of NoC design and tradeoffs that can be made with each component.

NoC Packets and Routing

The majority of data transmission methods for NoCs fall into either *circuit switched* or *packet switched* schemes. These relate to the amount of data that is moved in a transaction and whether the routing information is located within the packet. Packet switched networks will contain the required routing control information within the packet and the packet can only exist at one router at a time (aside from during a transmission transaction), requiring ample buffer storage to allow each router to be able to store an entire packet. Circuit switching on the other hand reserves a path for the packet, with the routing information typically leading the data. This requires minimal buffering per router channel (just the routing data), but it does mean a packet can span several routers and data channels. This clearly results in a tradeoff between amount of memory required for buffering at each router and the amount of interference on a routing path on the networking; indeed *wormhole routing*, the most common form of circuit switched routing for NoCs, is very susceptible to deadlock [56]. The authors of [57] claim that, generally, for larger packets in lower-load applications that circuit switched communication should be used for NoCs; with packet switching performing better in high-load, small packet applications. They also show the extra area cost of packet switched routing, with their synthesised packet switched router requiring nearly five times the area than their circuited switched router.

NoC Routers

The router tends to be the most complex element in the NoC. Much like a conventional network router its role is to connect routers to each other, connect the processing node(s) to a router and to support the communication protocol(s) that run on top of the NoC hardware. At the heart of the router is the switch that connects input data channels to output data channels. The router requires a control unit that uses an arbitration policy and connection rules (sometimes encoded within a *routing table*) to determine which input channels should be connected to which output channels via the switch. As exemplified in [58], the design of the router will reflect heavily on the system in which the NoC is to be used and the communication demands of the applications running on the NoC, especially with systems requiring support for both "guaranteed" and best-effort traffic. The AETHEREAL NoC [59] for example supports a sophisticated timeslot arbitration and so has a complex switch controller within its design, resulting in a large amount of control logic required for the router and memory resources required for the various timeslot queues.

Data Channels

Physical wires that carry the data connect the routers to each other. This may seem a trivial part of the NoC design, but the number, width and length of these wires can contribute significantly to the operating envelope of the chip, with [54] suggesting that as technologies shrink, the influence of crosstalk, power supply noise and wire capacitance will prove a real challenge to NoC interconnect, likely requiring the use of error protection methods on the interconnections with future technologies. This problem is even more applicable to any global signals or other lengthy signals with high-fanout due to the large number of signal repeaters on the wire, so designs suitable for high scalability will require minimal use of such signals. High-speed serial communication has been successful in conventional networking to alleviate these problems, although their use within NoCs is limited due to the extra area required by the fast serialiser/deserialiser pairs [60] and the increased thermal load of the fast serial clock. NoCs will help in this regard as the grid structure ensures that long wires will not be needed in the design (the longest performance limiting wires will be those between

NoC routers) and the regular grid structure can give an efficient cross-talk and thermal analysis as once one tile is analysed the resultant model could be tiled across the die.

2.3.2. The Many-Core Design Space

The NoC provides a means for the many-core system to communicate between individual cores as well as interfaces external to the chip. There are more design decisions to be taken at the application level that will affect the performance of both the NoC and the application running on the many-core system. There are a large number of high-level design aspects that are still open research problems such as task partitioning, communication partitioning and scheduling [61]. These problems depend on both the higher-level application and the operating environment (the application could be spread across several processing cores, several chips, or even part of a more distributed system) but also on the interfaces and capabilities of the lower-level support components of the embedded system.

In a many-core system these low level components are the mechanisms that support the applications running on the cores. This includes communication between cores, loading and execution of the software running on the cores and also the run-time management of cores (e.g. nodes being enabled, frequency scaling, configuration of hardware accelerators). There is clearly scope for merging of these layers but in order to support many different applications, an architectural divide and abstraction needs to be made at some level in the system. As these problems are highly application specific, use of high-level modelling is becoming popular to design the system. Tools such as Ptolemy [62] allow high-level modelling and simulation of aspects of the tasks of a complex embedded application before it is implemented on a hardware platform.

Given the complexity of the higher-level design aspects, they are not considered any further so this section can focus on the challenges of the lower-level design challenges. The experiments presented in later chapters use abstract task models where task and communication partitioning has already been done and schedulability is not considered an issue. For a more holistic many-core support platform these factors would need to be addressed.

Task Allocation

Deciding which task each node should be doing in a many-core system is a fundamental part of the multi-objective design space exploration involved in many-core system design. The node-to-task mapping will affect many key constraints of the system design, even for homogeneous many-cores. For example, a poor mapping may result in excessive communication overhead through longer communication paths between nodes in the data processing flow, or increased thermal load if busy nodes are clustered together and even limited system throughput if not enough nodes are assigned to tasks on the critical path. Thus an ideal task allocation needs to optimise topologies with both a physical (relative location) and a logical (application) focus. This becomes an even harder problem once adaptation is supported within the task allocation model as changing the task of one node will have both an upstream and downstream effect on other nodes in the data-flow. If heuristical approaches that are used to optimise the design space are then used in an adaptive context, a huge number of different task mapping scenarios must be modelled which is clearly not very scalable and would be a limiting factor as many-cores reach the size of hundreds and thousands of nodes.

The authors in [63] suggest that the problem should be split into hard-realtime and soft-realtime allocations due to the differences in objectives and strategies for the two problem areas. A common feature amongst many of the approaches from both types of allocations is a database driven approach where the application needs, constraints and other heuristics (possible measured from previous executions) are combined with the attributes of the many-core fabric. This database is then used to generate both offline mappings and mappings at run-time. The authors highlight this centralised approach as effective for small many-cores, but the problem space becomes far too large when many different applications are mapped or a large scale many-core is used. A hybrid approach is suggested whereby offline analytical solutions computed from a reduced design space set are combined with less rigorous run-time adaptations that then provide a degree of management once the applications are running on the many-core.

NoC Routing

Very closely tied in with the many-core task mapping is the allocation of NoC routing resources between the communicating nodes. The large amount of network resources allow many options for connecting two nodes and so methods must be in place for determining a packet's path through the network and also how other factors such as traffic priority of Quality of Service (QoS) affect it. As highlighted in [54] the ideal routing algorithm needs to take the following factors into account when deciding on routing paths between nodes:

- The type of data switching that the NoC supports and the router and channel level (i.e. packet switching or circuit switching).
- Whether a deterministic or adaptive routing approach is used. Deterministic routing will use preplanned paths that can not change once the packet has left its source, whilst adaptive routing will look up the next direction from each router's routing table on arrival. Hence with adaptive routing the source node does not know which path its packet will take.
- Whether the routing algorithm always needs to select the shortest path, and if not under which cases this should be relaxed.
- Whether the application allows "dropped" packets, in which they are destroyed and the data lost if they do not reach their destination node by their latest arrival time.
- Whether the routing decisions are being made in a centralised or decentralised way. Decentralised routing decisions will need a method of interaction between routing decisions to allow interplaying factors to be exchanged.

This is a very complex decision space and, as with the task allocation design space, it is unlikely that a centralised decision making process will be able to scale up elegantly to hundreds and thousands of cores.

Fault-Tolerance

An appealing aspect of many-core systems, and a key element of very large scale many-core, is the ease of support for node-level fault detection and mitigation. If a node is failed then it can be swapped for a spare core within the many-core. As the size of the many-core increases, more spare cores will be available due to core provision outstripping application demand or non-optimal task mappings leaving spare cores across the system. Unfortunately most approaches seen so far require online remapping and re-routing of the application requiring significant computation overhead, however some approaches based on local redundancy have been proposed [64] and such approaches are more likely to be scalable to large many-core systems, with the fault tolerance being a property of the fabric that the mapping step can exploit.

2.4. Autonomic Adaptive Systems

Systems that self-organise and self-optimise have regularly been a focus of modern research and can be seen as a promising way to handling the ever-increasing complexity of systems in an efficient and scalable fashion. The field of *Autonomic Computing* has emerged as an approach to manage complex systems autonomously without human input. First introduced by IBM in 2001 [65] by taking inspiration from the *autonomous nervous system*, a system within the human body that maintains body homeostasis through management of many bodily functions in a way that we would consider autonomous; from internal body temperature control to respiratory function and even managing our heat rate to allow us to make “fight or flight” decisions on the spot [66]. IBM envisaged large business systems that are “capable of running themselves, adjusting to varying circumstances, and preparing their resources to handle most efficiently the workloads put on them” [65]; a brief that could easily apply to the many-core systems introduced in the previous section. Further to this and in the same paper, IBM also suggest eight key elements for a system to be considered autonomic:

1. An autonomic system needs to “know itself” and comprise of components that also posses a system identity

2. An autonomic system must configure and reconfigure itself under varying and unpredictable conditions

3. An autonomic system never settles for the status quo - it always looks for ways to optimise its workings

4. An autonomic system must perform something akin to healing - it must be able to recover from routine and extraordinary events that might cause some of its parts to malfunction

5. A virtual world is no less dangerous than a physical one, so an autonomic system must be an expert in self-protection

6. An autonomic system knows its environment and the context surrounding its activity and acts accordingly

7. An autonomic system cannot exist in a hermetic environment

8. An autonomic system will anticipate the optimised resources needed while keeping its complexity hidden

Whilst this list is intended for larger systems than a typical many-core, there is no reason to exclude the possibility that the complexity of a large many-core system could hit an equivalent size as the enterprise systems considered for these rules. Indeed, at least six of the above points (with point 1 possibly excluded as the scalability of large many-core systems means that it may not be possible for the whole system to "know itself" or conversely smaller systems can be managed centrally and will not need system-identity of the sub-components) can easily be seen to be directly relevant to many-core systems. This idea was then consolidated further in 2003 with a follow up paper [67], whereby this set of elements are reduced to four: *self-configuration*, *self-optimisation*, *self-healing* and *self-protection*. This has became known as the *self-** set of goals for autonomous systems. This paper is also where engineering aspects are starting to be considered, including how autonomic elements should be functionally represented via high-level specifications and how system monitoring is crucial to achieving the self-* goals of the system. The biological metaphor is changed slightly to that of the distributed intelligence of an ant colony. However, this change is a good indicator as it shows the autonomous behaviour of the system is more important to the researchers than blindly following a biological metaphor. Finally outstanding research challenges are suggested including:

- Behavioural abstractions: how can behaviours designed local to an autonomic element interact with other autonomic elements to produce the higher level system goals desired?
- How can robustness of autonomic systems be described and analysed?
- How should learning and self-optimisation be implemented in such a dynamic environment where there is no guarantee of convergence; indeed is this actually a desirable feature of autonomous systems?
- How should individuals in the system interact with each other and how should the communication over the system as a whole be structured?
- Can we abstract statistical behaviour over system life and let the system feed this into its learning processes? At what point does over-fitting of such a statistical model actually become detrimental to correct, adaptive behaviour?

The intelligence aspect of autonomic systems is of interest as it is this part that is providing the adaptive, understanding behaviour. However it could be argued that artificial intelligence is not yet at a stage where it can simply be given high-level information and expect to integrate with a system. Instead the design flow for autonomic systems consists of specialising an intelligence system to the system with the autonomy then occuring at run time. This will be crucial for adopting autonomic systems in the near future as hard-AI driven systems would be far too resource intensive and make the verification of autonomous systems even more difficult. Indeed in 2004 the Defense Advanced Research Projects Agency (DARPA) started to show an interest in IBM's work with the ambition of applying the concept to large-scale military command and control systems [68]. Of key interest here was the self-healing and self-protection elements with the following goals from [68]:

- Operate through cyber attacks and provide continued, correct and timely services to the user
- Adapt security posture to changing threat conditions and adjust performance and

functionality

- Always know how much reserve capability and attack margin are available
- Restore system capabilities to full functionality following an event
- Autonomously reassess success and failure of all actions before, during and after an even
- Autonomously incorporate lessons learned into all system aspects

What can be seen here is a more system-focussed refinement of the definition of an autonomic system for DARPA's application. Arguably these are still high-level system requirements and so still omit the details of how the "intelligence" should behave, but it is easier to comprehend how these objectives could start to be engineered for their intended application. Indeed it seems that such behavioural descriptions are an effective way of describing the requirements of an intelligent system (e.g. requirements of system sensors, actuators, methods of analysis and learning schemes), but instead of the typical engineering flow of iteratively refining these requirements into system requirements, component requirements etc., our intelligence model should hook into the system at some point to maintain adaptivity. Where this intelligence hooks in is an important tradeoff in terms of complexity of AI model and adaptivity required across the entire system.

A recent (2010) review of autonomic computing is found in [69] where a focus is given on using biological inspiration to achieve the self-* properties and also some discussion on system theory. It is claimed that our systems will need to move away from being "point-correct", with the system state determinable at any given time, to being "process correct" where they react correctly to changing situations with predictable impact on system performance and/or accuracy. The authors suggest that this includes how we can describe such systems' behaviour as a range of acceptable behaviours given a certain task and then indicate a preference behaviour at a given instant. However they also point out that no compatible systems engineering paradigm exists for describing this kind of system behaviour. They give the example of an autonomic power management system that can optimise power requirements: "we must be able to state the bounds within which the power demand will vary, its impact on response times, and its interaction with subsystems that may affect load or communications, as these factors influence other choices within the management system." However if we are to introduce more determinism then we will still need a mechanism to be able to guarantee that systems will still exhibit self-* properties.

So there still seems to be many challenges regarding autonomic systems; from how we can describe and define the behaviour of the system to guaranteeing effective emergent behaviours. This is still a relatively new research field however and so an experimental approach is certainly justified to allow possible solutions to be discovered. Some give and take will be required, it should not be expected that we can achieve self-adaptive systems for free. One approach may be to "promise" a worst case performance of the emergence behaviours. What is also clear is that there are some commonalities between different autonomic systems: ideas applied to a 1000 spacecraft swarm [70] are rather analogous to a 1000 core many-core or even 1000 sensors in an ubiquitous computing system.

The embedded intelligence created in this work aims to provide a system with several autonomous properties, online adaptation to poor initial conditions and online adaptation to faults for example. Criteria from this section are used when evaluating the experiments in the context of fully autonomous systems. As only a sub-set of system and application parameters are put in control of the embedded intelligence, the capabilities of the system will be considered in the context of the goals suggested by [67], namely: *self-configuration*, *self-optimisation*, *self-healing* and *self-protection*.

2.5. Existing Adaptive Hardware Platforms

The application of biological models to hardware platforms has been a key research area in the field of adaptive systems and many systems are implemented on FPGA due to their reconfiguration possibilities. Many previous approaches have suffered from a complex translation of the biological model to the hardware platform and smaller FPGA densities in the past were a serious limitation. However, FPGA densities have increased dramatically over recent years allowing more logic to be dedicated to supporting the biological model; suggesting that some of the previous work in this field may be worth re-visiting with significantly larger devices. Table 2.1 shows a summary of recent developments in this field for systems that target a single chip or a small number of chips that could now be possible to merge into a single device. Bio-inspired systems that span networks or a large number of large-CPUs have not been considered as it is felt that these focus more towards management of distributed computing systems than autonomous management of an embedded system and have access to computing resources that are beyond scope for a large many-core on a single device (despite the large scale of the many-core the capability of the single nodes will still be relatively simple to achieve the scalability) and face challenges that are not applicable to single-chip many-core systems.

Project	**Biological inspiration**	**Year Developed**	**Description**
Teramac [71]	N/A but conclusion highlights need for autonomously managed system	1997	Teramac was a large machine built out of failed FPGA devices. Diagnostic tests determined where the faults occurred and record them in a defect database. Faulty regions are isolated on a device basis and designs are mapped around these faults. Redundancy on critical inter FPGA and memory tracks.
Bionode [18]	Cellular, endocrine signalling, hormone systems	2004	Endocrine system inspired cellular architecture. Built structure of 30 cells (each with Virtex-2 and C) connected in 3D-toriod shape with simple asynchronous interconnect between nodes. Processing, communication and fault tolerance is based on hormone production and suppression. Grouping of functionally similar cells produces organs depending on process demand. Experimented with simple calculations and removal of nodes. No ability to dynamically change cell function.

Project	**Biological inspiration**	**Year Developed**	**Description**
POEtic [72]	Cellular evolution, artificial development	2004	Based on the biological development principles of Phylogenesis (the evolution of the genome), Ontogenesis (the growth of the ogranism) and Epigenesis (the development of a individual through its learning systems). This is implemented on a cellular hardware fabric of basic functions that can be interconnected to build more complex functions. Evolution and artificial development is used to configure routing, molecule interconnect and molecule function to evolve the system function. ASIC of 12 molecules was developed.
MOVE [20]	Cellular, self-replication, differentiation	2006	Processor implemented on POEtic with a Transport Trigger Architecture, processor moves data from one functional unit to another. This lends well to a cellar architecture with the functional units located in cells. Routing is predetermined where each cell 'announces' its inputs and outputs, then the routing layer of POEtic is configured to build the organism's structure. Self-replication allows the system to be built from one cell. Replication is built from self-inspection.
PERPLEXUS [73]	Cellular, self-replication, artificial culture application	2006-2009	Ubidules aimed to produce neural friendly architectures by means of self-contained multi-cellular organisms. Ubichip supports this by providing cells consisting of a 4 bit registers and LUTs, usually configured as an ALU for neural applications. Self-replication is via self-inspection and a configuration interface allows cells to be dynamically reconfigured without impacting the overall system (and without frame limited reconfiguration). A neural robot controller of 40 cells was implemented on simulated platform. The project also showed neural adaptivity as older, unused neurons die off when perception changes (i.e. new environment).

Project	**Biological inspiration**	**Year Developed**	**Description**
SABRE [17]	Cellular, based on prokaryote cells	2010	Captures the behaviours of high level embryonic systems based on unicellular prokaryotic organisms by creating a programmable "cell" that could perform an arithmetic function and also contained routing capabilities to allow cells to be connected to form clusters and colonies analogous to social bacteria. The routing and cell overhead to support a 4-bit multiplier required 40 cells to implement.
DodOrg [19]	Cellular, Artificial Hormone System	2011	A hierarchical approach of: Brain (application, consisting of tasks), Organ (mapping tasks to processing units (OPCs) and Organic Processing Cell (task processor, reconfigurable). Organ formation was via an Artificial Hormone System to groups tasks together to produce organs. Organic monitoring system monitors and evaluates the system performance across all three levels including data generation and data analysis. The AHS claimed to support self-organisation, self-configuration, self-optimisation (at reallocation time) and self-healing. Simulated as part of a robot controller consisting of 6 tasks on a 2x2 OPC grid, each OPC a Virtex-2.

Table 2.1: Survey of existing adaptive hardware platforms

From Table 2.1 it is clear that most of these recent efforts have targeted cellular biological systems. This is not without good reason as all life is built upon these fundamental life processes. The ecosystem however is extremely complex. For example, POEtic's plan to combine evolution and developmental processes was extremely ambitious but such complexity is required when developing systems modelled this close to their biological inspiration. Cellular dynamics rely heavily on chemical interactions and reactions which are extremely expensive to model in hardware. It is therefore unsurprising to see that most of these projects produced very basic functionality relative to the large amount of hardware the model required to implement them. This is a shame as these models would benefit highly from a large implementation and should scale well.

Therefore it is imperative that, for a bio-inspired hardware model to be successful, the underlying organisation method used by biology must map well to the targeted hardware platform. Any chemical or molecular based systems will require numerical computation for the model and so will be expensive to implement in hardware or may suffer from not exhibiting the desired biological properties if the numerical precision is too low.

2.6. Summary

This chapter has explored some of the challenges facing the future of electronic systems and has also illustrated how the trend of large many-core systems will require multi-objective analysis and optimisation of a huge number of constraints. These are traditionally tackled with offline modelling and analysis but these algorithms tend not to scale well to the size of systems predicted by ever increasing transistor densities. Further still, device variability means one solution cannot be applied to all devices without sacrificing the achievable performance to build in margins for variability. The effect of transistor ageing requires even more extensive modelling and so design analysis will become more complex and more performance will be lost to the added ageing margins. Future systems will need to be designed not to avoid faults at all costs, but instead embrace faults as a normal part of device life.

Thus systems will need to be designed that can swap single core performance with a larger number of smaller cores, leverage the unused resources from Dark Silicon and adapt to faults and variations in the filed at runtime. Many-core provides us with an adaptable hardware platform, their adaptivity can be exploited with the use of an intelligent autonomic control system “hidden” in the dark areas of the chip that manages our design space and detects and handles variation at runtime. Autonomic computing has given some inspiration for how the high-level behaviours of such systems could look, however many open problems remain concerning bounding the emergence and how such complex adaptive systems can be engineered.

The next chapter explores the autonomous adaptive biological system that this thesis is inspired by: *the social insects*. Chapter 4 then discusses the many-core experiment platform, with the design choices for some of the many-core components based on some of the surveys introduced in this chapter.

Chapter 3

Large Scale Social Insect Systems

3.1. Overview

There are many examples of large complex systems in Nature that exhibit both the scalability and collective co-ordination that are required for large many-core systems. From the molecular interactions of Gene Regulatory Networks driving development of multi-cellular structures [7], to the chemical signalling between bacteria in a *Protozoa* society [8] and complex systematics of clonal organisms in a *Hydrozoa* consisting of many sub-organisms [9]; many of these natural systems exhibit highly-scalable development and maintenance abilities for solving a particular set of actions for survival in a number of challenging environments.

In this chapter the *Social Insects* are presented as a suitable model for enabling autonomic many-core systems. The individual workers have limited memory and decision making capabilities [6] yet when working together in colonies at huge scale exhibit many non-centralised features that are desirable for large many-core systems including self-organisation, self-optimisation and fault tolerance. This chapter starts by discussing the required intelligence abilities of a single colony member versus the abilities of the entire colony and motivates our choice to focus on social insect colonies. Intelligence models of distributed task allocation are then introduced as these models are crucial for determining most of an individuals behavioural decisions. Finally, other selected behaviours of individuals and colonies that map well to the engineering challenges of many-core systems are described.

3.2. Intelligence Capabilities of an Individual

3.2.1. Colony Size versus Individual Capabilities

The decentralised nature of social-insect colonies requires a compromise to be established between individual complexity and inter-agent communication complexity. In general, the more capable an individual is of understanding and interacting with their environment, the more complex the communication methods need to be. This is in order to support the transfer of more detailed information to their peers (with potential for loss of scalability), allowing an individual to make decisions with richer information acquired from a smaller number of their peers. All social insect colonies of differing organisms (e.g. ants, bees, termites, wasps) and each organism's sub-species rely heavily on joint decision making processes, although the degree of social decision making is driven by the size of the colony as much as by organism and species differences.

The analysis in [74] shows there is a clear relationship between the intellectual capabilities and specialisation of an individual and the typical size of the colony. This applies equally across colonies of different species of social wasps, social bees, termites and ant colonies. For small colonies (tens to hundreds of workers), individuals exhibit highly adaptive properties in both physical form and behavioural capabilities. Workers in small colonies have not evolved to be reproductively sterile allowing them to produce more worker eggs if required. In some species workers can even replace queens as the primary breeding members if the queens die or are cast out from the colony. Genetic variation between individuals is also low which results in a lack of "specialists" as seen in larger colonies. This leads to a more adaptive class of workers, which is required for a smaller colony as the loss of only a handful of individuals will have a high impact on the dynamics of the colony.

However, a more adaptive set of workers requires a higher level of worker intelligence to ensure it is making the correct decision; the impact of an individual's choice of role within the small colony will have a far greater effect on the colony as a whole. This had led to hypotheses given in the paper *Individual versus Social Complexity, with particular reference to ant colonies* [75] of *"individuals of highly social ant species are less complex than individuals from simple ant societies"* and more generally *"as social complexity increases, there is indeed a correlated decrease in individual complexity"* [75]. This is supported by Bourke's survey [74] where large colonies (hundreds to millions of colony members) are considered. These colony sizes exhibit social specialisms unique to managing the large-scale of the nest, such as sterile workers and more genetic variability between individuals. This results in some individuals being more suited to certain tasks than others (*polymorphism*).

The concluding remarks of [75] complement the observations in the survery, the authors make the following (selected and paraphrased) generalisations of large scale ant colonies:

1. Social complexity is positively correlated with differentiation between individuals. Individuals differ from each other in three main ways: polymorphism (genetic variation), physiological specialisation (developmental variation) and behavioural specialisation.
2. Individuals in complex societies tend to be restricted from exhibiting the full behaviour repertoire that an individual could support (i.e. tend not to be generalists).
3. Complex societies tend to have relatively little intra-colony conflict and often tackle tasks in a highly cooperative manner, requiring working as groups and teams.
4. Complex societies tend to be "high tempo", comprising very active and fast moving individuals.
5. Complex societies are highly integrated, involving a sophisticated and heterogeneous communication network of signals and cues.

3.2.2. Neural Complexity

Despite the variation in behavioural roles of workers due to colony type and size, it is evident that all workers have a common set of basic behaviours that allows them to perform basic functions such as moving, eating and interacting with their nestmates. It is important to understand how the individual worker's intelligence is structured to allow us to extract the sensory handling and decision making processes we are interested in. The review paper *"Are bigger brains better?"* [76] explores the insect brain in relation to larger organisms. The authors highlight that there is a direct relationship between the body mass of an animal and its neural mass, predicting a larger brain for larger organisms which is fairly non-surprising. However, the authors also explore why this is the case when many complex features such as locomotion control, visual processing and movement planning are fairly common in many organisms. They conclude that most of the extra neural processing power capabilities of larger organisms is related to the fact that larger muscles will require more complex control circuitry. This will also increase the amount or neural mass required for ancillary functions such as movement planning and larger fields of vision that larger organisms will possess. And so the authors claim that, despite their relatively low neural mass to body mass, insects are still capable of a large repertoire of complex, cognitive behaviours; the low neural mass attributed to being highly optimised to their operating environment and being less concerned about survival of the individual.

The paper also introduces the role of memory as a neural resource [76]. Whilst comparative testing of memories of vertebrates and invertebrates is a difficult task, it is generally accepted that invertebrates have minimal amount of long-term memory. In the case of social insects this is a small number of sensory cues used in navigation. However the authors highlight that this is in agreement with the evolutionary neural optimisation of the individuals as an advantage of a large memory in vertebrates is the ability to apply remembered situations as a form of adaptivity when facing new problems, with social insect colonies this level of adaptivity is achieved via colony dynamics and so the neural cost of memory has been minimised. This viewpoint is expanded in another aspect in [77] where the energy cost of learning is also considered. By reducing the amount of cognitive behaviour reliant on memories the amount of learning required before an individual can perform the action is also reduced, saving the energy and time that would be "wasted" undertaking the learning process. Burns and Foucaud explore intelligence as a trade-off, and argue that, due to the extra time, energy and neural substance required to support learning and memory, learning imposes a fitness penalty when compared to a member where the same behaviour is purely instinctive. Again the distinction is made between small and large brains in terms of quantitative improvements (e.g. sensor accuracy) vs qualitative improvements (larger behaviour repertoires). They also identify that insects can exhibit some degree of complex learning (maze traversal for example), with the suggestion that higher-order neural circuits exist in insects that allow them to integrate sensor and memory information so that they can adapt various learned behavioural outputs. This leads to an indication that there is a also trade-off between energy and neuronal matter dedicated to sensor analysis and energy and neuronal matter dedicated to higher-order decision units.

This minimal learning approach results in most of a social insect worker's behaviours being innate and allows an individual to perform actions as soon as they hatch. Again, the authors in [76] suggest that this is a key part of the high neural efficiency of social insects but does not hinder their cognitive capabilities. They have produced a survey of honey bee activities and found that there are 59 distinct behaviours that a honey bee worker can perform. It is impressive to note that in similar behavioural studies with dolphins have shown that they exhibit 123 distinct behaviours [78], despite a neural mass 10,000 times that of a honey bee. The authors of [76] once again link this back to evolved neural efficiency with many of the insect's basic behaviours being created by simple local reflex circuitry and more complex behaviours being then made up of combinations of these simple reflex circuits by the insect's cognitive abilities. The authors point out that achieving this cognition is not that neurally costly as circuits for visual categorisation, landmark learning and basic numerical abilities have been achieved with circuits comprising 10s of neurons.

3.2.3. Hardware System Implications

In this section we have seen that (*a.*) large colony sizes can reduce the intellectual requirements of each individual and (*b.*) intelligence pathways in insects are highly optimised for the most computational power into minimal space but, thanks to this high level of optimisation, this does not necessarily mean that the behavioural abilities of an individual are very limited. This has positive implications for the hardware system because it means that the embedded intelligence in the many-core fabric will benefit from high node-counts. It also means that it should be possible to exhibit more complex behaviours with a low number of embedded neural pathways, provided that the pathways rely on minimal learning to create and maintain the circuits. It is envisioned that most of the implemented pathways will be analogous to the many innate behaviours of insects with a few pathways providing the cognitive aspects which enable, disable or modify the innate behaviours of the embedded intelligence.

3.3. Social Insect Intelligence Models for Task Allocation

3.3.1. Overview of Intelligence Models for Task Allocation

The decentralised yet highly-scalable task allocation of social insect colonies is one of the key dynamics that needs to be captured for large scale many-core systems. Having seen the limited cognitive capabilities of an individual, task allocation capability needs to emerge from colony dynamics rather than from a coordinator or highly-informed decisions by individuals. For this analysis, ant colonies have been selected as a focus point. This is because their tasks are very well defined and their communication methods are extremely simple compared to other insects; honey bees for example use visual cues as their communication method between individuals [79], requiring neural pathways for image processing that would be difficult to interpret and implement as a communication method in the hardware system.

Tasks that individual ants can undertake are either primarily internal to the nest or require leaving the nest for large amounts of time. Internal tasks include: brood rearing, nest maintenance and expansion, removing dead and ill ants, food distribution, food storage and processing, tending to fungus farms (some species) and queen care. External tasks include foraging for food, sharing locations of food sources, patrolling the nest and aphid farming (some species). This diverse set of tasks requires different sets of skills and sensory inputs for many of the tasks and so switching tasks will require a cognitive decision by an individual and the ability to decide what sensory inputs are used. The number of individual ants performing each task is also important and needs to be constantly within certain bounds to ensure short-term and long-term colony survival. For example, if the number of brood-rearing individuals is high enough relative to the number of foragers (or amount of food each forager manages to bring back), then a larger number of new ants will be created than the food coming into the nest to maintain them and the colony may starve.

Beshers and Fewell provide a comprehensive review of different task allocation models under consideration in *Models of Division of Labor in Social Insects* [23]. They consider six classes of models: response threshold, integrated information transfer, self-reinforcement, foraging for work, social inhibition and network task allocation models. These schemes are represented in Figure 3.1 with illustration of which factors are present in each model. Each model differs in what information source is used by individuals to determine which task they should be undertaking and so a brief summary of each model is given:

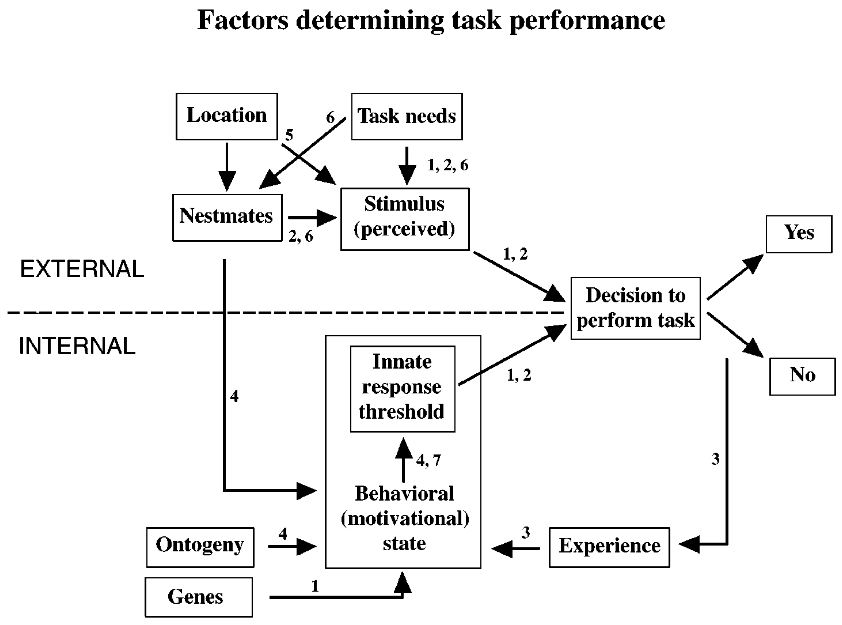


Figure 3.1: Illustration of factors influencing an individual's choice to undertake a particular task. Numbers on the arrows indicate effects that are included in each type of model: 1 response thresholds; 2 information transfer; 3 self-reinforcement; 4 social inhibition; 5 foraging for work; 6 network task allocation. This diagram shows how no model relies on a single factor for determining the task to undertake aside from Foraging For Work. The external and internal split is also of interest as it shows that individual ants rely heavily on the conditions of their environment when deciding which task to undertake. Only the self-reinforcement model relies on learnt memory of task performance, the only other form of storage in the other models comes from genetic memory. Figure from [23]

1. *Response Threshold:* In this model the assumption is made that workers are exposed to task-specific stimuli (e.g. dirty chambers, untended larvae, hunger) and each worker has an internal threshold the dictates whether an individual decides to undertake a task depending on if a task stimuli exceeds this threshold, with a default behaviour of a "rest state" i.e. doing no task. The thresholds can vary between individuals and when a worker starts a particular task before other workers (it may have a lower stimulus threshold) it also starts reducing the task stimulus for other workers, providing a lot of negative feedback into the task

allocation system.

2. *Integrated Information Transfer:* This is an extension to the prior threshold model, whereby social information transfer is also integrated into the threshold. Workers could inform each other information on what tasks they perceive need to be undertaken, yielding a more step-wise distribution of task allocation from the positive feedback nature of the social-communication.

3. *Self-Reinforcement:* In an attempt to model the occurrences of specialists and generalists in the colony, experience based models have been proposed. In such models the decision to undertake each task is considered a probability, a successful undertaking of a task increases the probability that this task is performed again whilst an unsuccessful task or a lack of opportunity to undertake it will reduce the probability of the task being performed. This results in a self-reinforced system and by adding a notion of "forgetting" it allows specialists to revert back to generalists should the balance of tasks in the colony change.

4. *Foraging For Work:* This model uses a production line analogy such that there are a series of tasks to be done, geometrically spread. On contact with a particular task an individual will perform this task until it is no longer required (see task stimuli in the Response Threshold model), at which point it will them roam the nest until a new task to be done is found. This model predicts *temporal polyethism*: the observation that a worker's set of tasks will depend on their age. Foraging for Work uses the fact that a worker's location in the nest has a strong correlation to the age of the worker and so the set of stimuli an individual is exposed to will change as they age and move to different areas of the nest. For example newly hatched workers will start their lives in the brood chambers at the back of the nest and so will find brood tending tasks to perform. However once this reaches a critical limit of workers then there will be a time where no brood tasks need to be performed and they will slowly work their way to the front of the nest as they forage for new tasks.

5. *Social Inhibition:* Another explanation of temporal polyethism can be obtained by considering the effect of older workers as an inhibitor for younger workers taking up new tasks. If a number of foragers is lost then the number of mature

workers in the nest is reduced, resulting in less inhibition of the potential tasks a young worker can perform and so making up for the loss of foragers. However such a model for task allocation assumes that all tasks decisions are polyethism and inhibition driven, which has been shown not to be the case with task stimuli.

6. *Network Task Allocation Models:* The final modelling method considers the interactions between workers and their environment as a series of differential-equations or network models. The resultant models show similarity to real colonies to an extent that it can be concluded that division of labour can be generated and maintained purely from the local information encountered by an individual worker.

The review then compares these models with with empirical evidence and identifies specific similarities and differences to try and determine how well each model fits to the real emergent task allocation seen in colonies. Finally, the authors conclude that these models should be considered "exploratory" and that in reality some hypothesies of these models will eventually be refined and merged to produce a final "explanatory" model (for example the merged threshold and reinforcement models in [80]). Thus it is likely that Figure 3.1 represents the general model well with all elements in the figure contributing some information to the task allocation process at both the colony and individual ant level. Assuming that this is the case then stimuli from the environment and nest mates have a key role on the decision made and so by implication the spatial arrangement of individuals within a nest is a fundamental property of what task is chosen; indeed this has been studied and shown in real colonies in [81].

A final consideration comes from the management of the dynamics and more recent biological thinking suggests that these models are likely to be combined with different systems providing inhibitory or excitatory feedback for expressing certain task allocation behaviours. For example, the model proposed in [25] refines the original network interaction model with learning from some individuals and feedback based on environmental stimuli. This produced a model that is more real to the behaviours of foragers deciding to leave the nest depending on the time of day and the humidity and temperature of that particular day. This required modelling of the dynamics of foragers outside of the nest depending on the environmental conditions and providing a distribution of nest return rate depending on the environmental conditions and what the individual had learnt about the environment that day. The interaction model then used this updated information to elaborate the interaction between individuals such that individuals with a bad experience of foraging at that day or time will be less likely to pass its information on to interacting nest-mates.

3.3.2. Hardware System Implications

The task models being related to an individual's location and local stimuli has a direct translation to the many-core system, where a node will have a different set of task, thermal and NoC traffic stimuli depending on its location in the system and the task currently being performed in that region of the many-core. So the embedded intelligence will need to be located at each node to capture the locality aspects of the sensing and to also capture the local effects of the decision making. Stimuli sensed from the many-core will need to include stimuli that affect the ability of an individual node to complete a task (fulfilling the *response threshold* model), stimuli that signify work that needs to be done (fulfilling the *Foraging for Work* model) and stimuli that communicate what work neighbouring nodes are undertaking (fulfilling the *Network Task Allocation* model). A fundamental difference between the dynamics of the many-core and social insect colonies is that the nodes of the many-core cannot move and so, instead of the individuals moving to exchange information as with ant colonies, the information must move using packets sent via the NoC and the nodes will infer information from the properties of these packets.

Figure 3.1 is a useful point to propose translation points. The internal/external split can respectively be seen as factors that are internal to an individual node and factors that require either direct or indirect input from either neighbouring nodes or from the local on-chip environment. More specifically:

External:

- *Location:* Factors that are determined by a node's location. This could be physical properties such as poor clock performance due to distance from a clock source or a bad thermal environment caused by potential other items on the chip other than the many-core (e.g. SERDES, clock generation).

- *Nestmates:* Factors that are determined by other nodes in the system. This could be application related (packets of data to process) or environment related (thermal hotspots, crosstalk).
- *Task needs:* Factors that are determined by the ability to run the application on a node e.g. processor capabilities, memory capabilities.

Internal:

- *Genes:* Settings/design of a node that give it a preference for a task (e.g. hardware accelerators)
- *Ontogeny:* Self-optimisation towards aspects of a task (e.g. reconfigurable hardware accelerators)
- *Experience:* Self-learning of a preference towards certain tasks by learning from previous performance when these tasks are tried.
- *Behavioural state:* The current mode of the node or other limitations affecting its ability to run tasks.

Both the *response threshold* and *information transfer* models rely directly on a stimulus-threshold decision making intelligence i.e. when a stimulus exceeds a threshold then a decision is made. However the experimental implementations of *Foraging for Work*[26] and *Network Task Allocation* [24] both also use stimulus-threshold structures to make the decisions within the models. This motivates the use of a stimulus-threshold intelligence architecture for the implementation of the embedded intelligence.

3.4. Other Aspects of Social Insect Task Performance

3.4.1. Ratio of Working versus Resting Ants

A recent focus of the task allocation models of ant colonies is to understand the observation that a surprisingly high number of individuals are inactive at any one time. Whilst there are opportunities in all of the task allocation models seen in the previous section for workers to be inactive, the observed number of inactive workers is extremely high. For example, in [82] the authors observed the number of active workers performing a nest migration task in the lab. They used several colonies of the same species of ant but split the colonies into two groups depending on their size; either a small size (median of 57 workers) or a large size (median of 165 workers). They found that even in the migration task (the ants moving themselves, the queen, eggs and young brood to a new nest), a set of inactive ants did not even walk to the new nest, but had to be carried by other workers. In large colonies the number of inactive ants that had to be carried was averaged at 42% of the colony's workers, but most surprising to the authors was that with small colonies 69% of the workers were inactive for the migration. The individuals that were active remained active for the entirety of the migration task and no inactive workers switched to an inactive state; suggesting an emergence of "key workers". The authors claim that this could be advantageous if the energy cost to the colony of carrying a worker versus the energy cost of worker walking is low. Thus key individuals can learn the route and will optimally ship workers and other nest material from the old nest to the new one using their learned trails in a more efficient manner than all workers finding their own way once; this approach is also less risky in terms of number of colony members getting lost during the migration and could explain why the ratio of active workers is lower for smaller colonies (loss of a small number of workers is more detrimental to a smaller colony).

Further work from a different team has shown that inactivity is not a side-effect of the experiment being performed in laboratory conditions [83], and also not due to a "shift work" organisation where one ant replaces another as it gets tired [84]. The authors here found that an average of 45.8% of their colonies was inactive and observed that when an ant is inactive it tends to then be more likely to be inactive at future points; leading to the suggestion that inactivity is a task speciality of certain individuals. Activity patterns for workers outside of the nest are driven by sunlight [6] but deeper within the nest the work rhythm is more constant so ants continuously at rest deep in the nest must be due to a side-effect (or feature) of the task allocation.

The same authors explored as to what causes this in two papers that examined a large number of hypotheses [85] [86]:

- *Resting:* The inactive individuals were resting having performed an action. Observations of the inactive ants showed that they rest for longer periods of time between either performing a task or nest wanderings when compared to active ants. Some ants never left the inactive state during the experiment (over 4 days).
- *Inexperience:* Younger individuals may have entered a new area (possibly due to over-crowding of the brood chambers) where their age polyethism means that they are not yet sensitive to any of the task stimuli in the area.
- *Cost of task switching:* Some tasks will have an energy cost of undertaking it, a learning period for example. If this is costly then the more energy efficient approach for the colony as a whole may be to only allow individuals already trained on the task to undertake it unless need is very high (e.g. in an emergency). This could be represented by a long delay period in the FFW model for individuals not specialised to that task.
- *Response threshold side-effect:* If an inactive worker has a higher response threshold to all of the available task stimuli when compared to its neighbours then another worker is likely to take up the task before its decision threshold is exceeded. This also suggests that inactive workers can act as a "reserve task force", whereby their decision threshold is only exceeded in cases of extreme survivability.
- *Workforce scaled to workload peaks, not average workload:* Brood development takes a long time and so cannot be used to adapt quickly to severe upsets in the colony dynamics. Maintaining workers in a low metabolic state may have evolved to be worth the energy cost for quick adaptation.
- *Food storage:* Ants share food by passing it to each other and can store relatively large amounts before passing it on to a nest mate. Having a large number of spare workers suspended in an inactive state allows them to be used as food storage buffer. Indeed some species have evolved "larder" individuals specialised for food storage by swelling their abdomens to share food later [6].
- *Communication nodes:* In a similar vein as the use of inactive workers as food storage, inactive workers can be used to enhance colony communication. Most

ant-to-ant communication is through pheromone emission during an antenna interaction period. This requires two ants to interact for a small amount of time, using inactive workers as an intermediate communication resource would allow active workers to offload their communication interactions to in inactive workers from a single interaction. This would also allow information to spread quicker amongst the colony, possibly providing increased colony-wide adaptivity abilities.

The authors do not conclude that any one of the above factors is the main reason for the high-levels of inactivity in the colony, or indeed if it has a net advantage to the colony or not. However, the simulations performed within [87] show that variable response thresholds between individuals is a key reason for the inactivity emerging. They used a model of random walks through nest with 75 workers and variable thresholds that decrease depending on the time since the last task was undertaken. These experiments found that certain ratios of worker fatigue to task stimuli resulted in emergent inactive workers that never work as despite their stimulus thresholds dropping very low there was always an active worker that would pick up the task first.

This area of research is currently highly active and so far no biologists have concluded as to whether repeated inactivity is a key component of the social insect's task allocation dynamics, if it is a detrimental side-effect or indeed if it has little effect on the behaviour of the colony.

3.4.2. Polymorphism

Polymorphism leads to size and strength differences between workers, dubbed *castes*, typically induced by workers tending to the brood who feed the larva differently to exhibit the polymorphism when they hatch and grow to adult size. This affects the optimisation dynamics of the colony as larger workers require more food and tend to move more slowly (and so interact with other workers less) despite being able to carry larger objects and have better fighting abilities. Tradeoffs have to be made in terms of the ratio of each caste and also at the task allocation stage as the allocation of workers to tasks will need to take the worker's caste into account.

One observed mechanism of controlling the recruitment of caste of workers to tasks is given in [88]. In these experiments the authors observed smaller workers carrying a variety of prey. For small flies the smaller workers could carry the prey between themselves, but for larger prey such as cockroaches or a bundle of flies a larger worker would be recruited to help carry the prey. The authors observed that a stronger food recruitment pheromone was laid in the cockroach case than as for in the fruit fly case. Assuming this couples with a larger response threshold in the larger worker then the larger worker will ignore the fruit fly recruitment pheromone trail but respond to the trail laid for the cockroach. Whether this higher threshold is applied to all decision thresholds in the larger worker or just specifically to this particular decision pathway at some point in the larger worker's development is not explored by the authors.

Another source of polyethism based specialism of individuals can be seen in Figure 3.1 and the task allocation model of the previous section. In the figure the learnt experience of performing a task feeds into the decision via the self-reinforcement model. This feedback would allow workers to attempt a task and increase their chance of performing it again if they deemed themselves good at it; this does imply an individual has the ability to review their performance, but this could be as simple as a measured reduced task stimuli within a time period. This would promote specialists to repeat tasks that their specialisms mean they are good at and also deter workers of a caste that is a poor match for the task. This model would also support adaptivity as a task stimulus could get to the point such that it overrides the deter inhibitor response sourced from the worker's experience, useful for tasks that would be better carried out by a non-expert than not at all.

3.4.3. Hardware System Implications

The role of the distribution of active versus inactive workers will be a interesting investigation with the many-core as inactive nodes can be used for managing Dark Silicon, network congestion and as standby spare processing capacity. The hardware will need to support disabling and re-enabling of the nodes from the embedded intelligence. Polymorphism also fits the many-core paradigm well as one of the key advantages of many-core systems is the ability to attached specialist hardware accelerators to a subset of the nodes. This is analogous to the castes within social insect colonies, with the different types of hardware accelerated nodes representing the individuals size/strength specialisation of the caste.

3.5. Hardware System Translation

As seen in the previous chapter, many-core systems and Network-on-Chips are a promising technology for the future of high-processing power embedded systems. Whilst the ideas and models presented in this chapter could be applied to a large range of technologies, their potential for low-overhead implementation and their relatively small operating environments (for example direct contact between individuals rather than interacting via the environment) lends these models well to implementation in single-chip systems. To bridge the gap between biological model and hardware system, Table 3.1 details how it is envisaged that these models can be used to solve problems faced by large scale many-core systems.

Capability Supported	**Social Insect Inspiration**	**Translation**
Task Allocation	Network Interaction Model	This model requires the individuals to move around the colony and interact with each other. In the hardware system it is not possible for the nodes to move. However as this movement is to generate information transfers with neighbours (via the individuals' antennae) this can be captured through the information transfer system between nodes i.e. by sending packets. This can be refined one step further as the goal is to ascertain what task the individual was undertaken. This can be captured indirectly by monitoring the type of packets emitted by an individual.

Capability Supported	**Social Insect Inspiration**	**Translation**
Task Allocation	Foraging For Work	This model also requires individuals to walk around the colony. Once their stimulus for a task is removed and it has not reappeared within a time period, the individual will go on a walk and will switch task to the next task stimulus that presents itself. The translation requires a task stimulus to be presented to the node and a notion of time to regulate for how long this stimulus should be considered valid. The stimulus in the many-core system is the packets that need to be processed. Once again, the node does not need to move to get this stimulus as it will be passing through the node in the form of packets. A timer can be provided from a global timing source.
Heterogeneity	Polymorphism	Hardware accelerators unique to a set of nodes would allow specialism to be provided, knowledge of the presence of such accelerators could be fed into the intelligence (a genetic or ontogenetic input to the intelligence) or memory could be added to the embedded intelligence which would allow a node to self-learn good and bad out through trial and error (this is the “experience” feedback loop from Figure 3.1).
Fault Tolerance	Task allocation Model	Individual level fault-tolerance is a key part of the dynamics of the task allocation methods discussed. Therefore both emergent task allocation schemes described would be able to support node-level fault tolerance without any additional changes to the model.
Thermal Management	Worker Activity and Resting	Thermal management will be in the form of disabling nodes or reducing their activity levels. This can be provided through an external environmental monitor that feeds the temperature of the node into the intelligence model. This could feed into any of the hypothesises presented in Section 3.4.1. Out of these hypothesises it is likely that the *resting*, *inexperience*, *cost of task switching* and *response threshold side effect* are the models most implementable into the hardware system.

Capability Supported	**Social Insect Inspiration**	**Translation**
Quality of Service	Worker Activity and Resting	There is no direct translation for QoS, however many of the objectives are tied heavily into the resting hypothesises and so a solution would look similar to these pathways. Higher level stimuli would need to be added to report the status of QoS goals. Motivation could also be taken from the self-regulation patterns introduced in [25]

Table 3.1: Translation between Bio Inspired Model and HW Model

3.6. Summary

In this section we have seen some of the complex high-level behaviours that social insect colonies have achieved in order to successfully survive in a vast number of different and ever changing environments. The colony is always striving to maintain a balance between energy efficiency, adaptivity and survivability and many millennia of evolution have led to a emergent system relying on decisions being made by individuals based on their immediate environments. The task allocation models we have seen use simple response threshold structures that can readily be mapped to a embedded hardware circuitry. An abstraction gap exists between the task allocation models and the neural models but this is acceptable for our intended investigations as we are not planning to implement an "ant-correct" neural controller for task allocation in our many-core; in fact, as shown by the interaction network models, simpler controllers can still provide capable adaptive behaviour. Thus, supporting the *Foraging for Work* and *Network Task Allocation* models through stimulus-threshold decision making units will motivate the design choices of the implementation of the embedded intelligence. These two models will also provide the main intelligence base of our experiments, with extra capabilities added to the model in the form of stimulus-threshold decision circuity "bolted on" to these fundamental task allocation models.

The next two chapters describe the hardware developed for the experimental many-core platform and also the hardware to implement these embedded intelligence models. Chapter 6 then presents the results of running an interpretation of the models described in this chapter on the many-core, including the capabilities of the emergent higher-level behaviours and how they compare to what is expected from the behaviours predicted in this chapter.

Chapter 4

The Centurion Many-Core System

4.1. Overview

The previous two chapters have shown that, firstly the problem domain considered in this thesis is intrinsically linked to the low-level hardware characteristics of a device and secondly that the proposed biological inspiration will need to interact with the hardware system via sensing and actuating parts of the hardware system. This chapter describes the high-level and technical details of the custom many-core system which is used for the experiments: the *Centurion* many-core system.

Centurion was primarily designed as a research platform for enabling experiments in both large scale many-core systems on a single chip and for Network-on-Chip (NoC) evaluation. Thus the main design goals reflected achieving the highest core count possible (for large-scale SoC research) whilst maintaining enough NoC reconfigurability to allow different NoC algorithms and settings to be supported.

At the time of development the community lacks such a platform. Existing many-core platforms implemented in hardware have so far tended to be on a small scale, for example the Intel Core i9-7980XE with 18 cores [89], the Intel Xeon Phi 3110X with 61 cores and the TILE-Gx72 with 72 cores [90]. Some larger scale many-core hardware platforms have emerged in recent years (e.g. GRVI Phalanx: 1680 cores [91], Epiphany-V: 1024 cores), but such platforms have many general-purpose aspects of the NoC removed or capabilities of the processing cores sacrificed to achieve the highest core count possible making them too limited to be suitable for NoC research. Centurion has to strike a balance between a high core count and providing representative NoC functionality that next-generation many-core systems will require.

Centurion also needs to be suitable for FPGA implementation as a VLSI design was not within the scope or nature of such a research focussed platform as the ability to slightly modify the system to fulfil a research objective is a important capability. The established soft-core processors provided by the FPGA vendors *Microblaze* [92] and *Nios* [93] as well as the often FPGA implemented *Leon3* processor [94] all have multicore processor support built into their design. However, when implementations of multicore systems using these processors are explored, it is seen that either a large shared interconnect or crossbar is used (Microblaze [95], Leon [96]) or a mixture of global interconnect and daisy chained local bridges are used (Nios [97]). The shared interconnect and the global memory space it supports gives lots of options for node communication protocols but is poorly scalable with lots of global signalling meaning that the maximum speed of the interconnect will drop quickly as more nodes are added to the system. The node-to-node daisy chaining of the Nios multicore solves the speed requirement but does not provide the node topology or network flexibility needed for large scale NoC research.

This chapter introduces the Centurion platform and describes the design of the components required for the investigations into large scale NoC. Firstly, the design of the NoC router and then the capabilities of the processing core attached to each router are discussed. Following this, the first of two implementations of the embedded intelligence capability is described (the second approach is more complex and is detailed in Chapter 5). The design of some of the advanced hardware features of the platform is then covered. Finally the chapter is completed with discussion of how the FPGA design process was undertaken for this complex design and includes design metrics such as hardware resource requirements.

4.2. Centurion Overview

The Centurion platform developed for the experiments in this thesis consists of 128 processing elements connected in a 8x16 grid and implemented on a Xilinx Virtex-6 LX760 FPGA. Each node in the many-core consists of the custom 5-port NoC router and a Xilinx MicroBlaze Micro Controller System (MCS) [98], this arrangement is shown in Figure 4.1.

Embedded within each router is either the Picoblaze microcontroller based intelligence [99] or the *Configurable Intelligence Array* module where our biological models are implemented in hardware. This module (described in detail in Section 4.5 and Chapter 5) has access to many of the internal signals of the router and processor, called Monitors in our system. These include signals such as the task-ids of packets routed through the router, the current clock frequency of the node and the current thermal state of the processor. The intelligence module can also affect several aspects of the router and

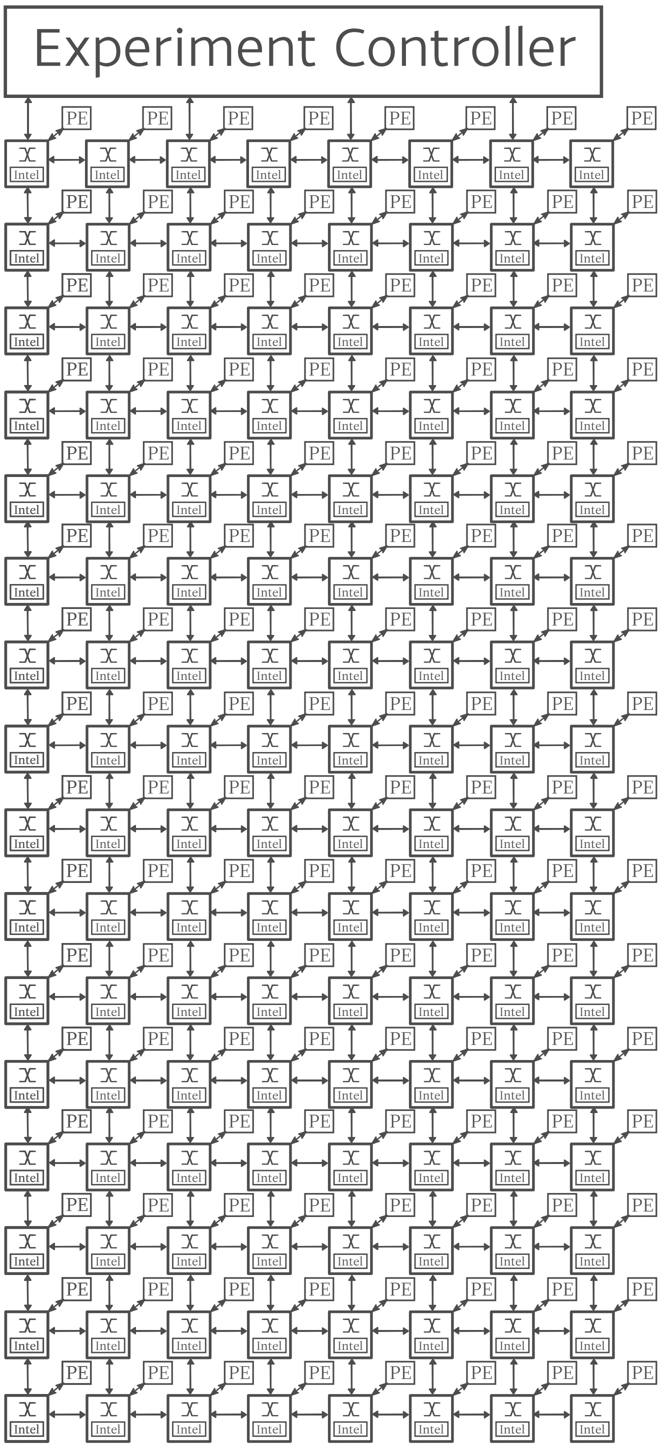


Figure 4.1: General overview of the Centurion 16x8 many-core. The 128 nodes, each consisting of a router and attached processing element, are arranged in a grid with cardinal connections using a Network-on-Chip (NoC) to connect the nodes together. A larger processor, dubbed the *Experiment Controller*, is connected to the NoC via the North port of four of the routers in the top row.

processor, dubbed Knobs, for example: the current task the processor should be running, the clock frequency of the processor and the routing direction of packets through the router.

Tying the many-core together is also a processor dedicated to running experiments called the *Experiment Controller*. This is a larger AXI-based Microblaze that manages the LVDS connection between the NoC and the PC used to manage experiment data. This allows experiment parameters to be sent from the PC and experiment runtime data to be sent from the NoC to the PC. The experiment controller can inject and receive packets from the NoC via four NoC interfaces connected to the (otherwise unconnected) North channel for four routers on the top row of the NoC. The experiment controller can also access the nodes separately to the NoC via a specialist debug interface. This allows experiment data to be downloaded and parameters to be set (e.g. for fault injection) whilst the experiments are running without interfering with the NoC traffic during experiment run time.

4.3. Router Design

The Centurion router, shown in Figure 4.2, is a five port NoC router that includes a *Router Configuration Access Port (RCAP)* to allow router settings to be changed remotely via the NoC. It also includes several performance monitoring signals and signals for modifying the router's behaviour, used by the intelligence model. The router supports two packet routing modes and is wormhole routing based to reduce device resources spent on packet buffers. A basic deadlock recovery mechanism is included within the router to enable experiments to survive deadlock conditions, however this is not as comprehensive as other NoC deadlock avoidance schemes as there is no design-time analysis of routing paths and their sensitivity to deadlock. Whilst Centurion will resolve all deadlock situations within a configurable timespan, it will affect application performance by removing deadlocked packets from the system. These packets can then be logged as a system performance metric or the packet can be resent from the place of deadlock resolution.

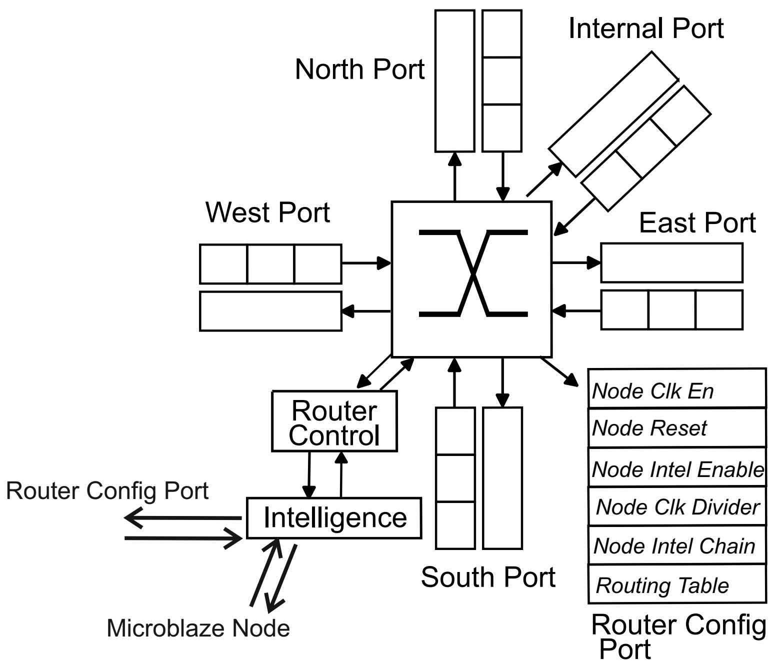


Figure 4.2: The Centurion 5-channel NoC router. The main ports consist of the cardinal directions and an internal port connected to the processing element. A sixth-port, the *Router Configuration Access Port (RCAP)*, allows the router to be configured remotely. Up to five concurrent connections can be set up between these six ports and independence between their input and output interfaces allows full-duplex communication across the five channels.

4.3.1. NoC Packet and Routing Modes

Centurion uses 9-bit words for communicating data between nodes, consisting of an 8-bit data word and a 1-bit flag that indicates if this word should be interpreted as a control (‘1’) or data word (‘0’). The following words are control tokens that Centurion supports and so have the control bit set to ‘1’:

Table 4.1: A list of Centurion control tokens. The first seven entries are used in the packet header and are routing instructions as described in this section. The final token is placed at the end of the packet and marks the tail of the wormhole packet, closing connections as it passes through the NoC.

Control Bit	**Data Token**	**Description**
‘1’	x80	Task Packet Header
‘1’	xC0	Route North
‘1’	xC1	Route East
‘1’	xC2	Route South
‘1’	xC3	Route West
‘1’	xC4	Route Internal
‘1’	xC5	Route to Config Port
‘1’	x7F	EOP (end of packet)

Unlike state of the art NoCs, Centurion does not inherently support a range of advanced routing capabilities such as virtual channels, flits and packet interleaving. This allows the Centurion data packet to be very simple, Figure 4.3 shows the layout of a packet and its two routing variations. The only essential data is a packet header (at least 1 word, this could be larger depending on routing mode used) and an EOP at the end of the packet. These are all “control words” i.e. have their control/data flag set to ‘1’. At the NoC level there is no limit to packet size but, as discussed in subsection 4.4.2, the buffers on the PEs meant that a packet size limit of 2048 words was enforced for the experiments reported in this thesis.

Centurion supports two packet switching methods: *a) turn-by-turn with header deletion* for packets where the location of the destination node is known, and *b) task-id based with routing tables* for packets that do not have a specific destination node but instead carry data for a certain type of task. From this point forward, packets of type *a)* will be dubbed **System Packets** and packets of type *b)* will be dubbed **Task Packets**; this distinction arising from experiments where system packets are used to set-up the experiments and task packets are used within the experiments to carry application data.

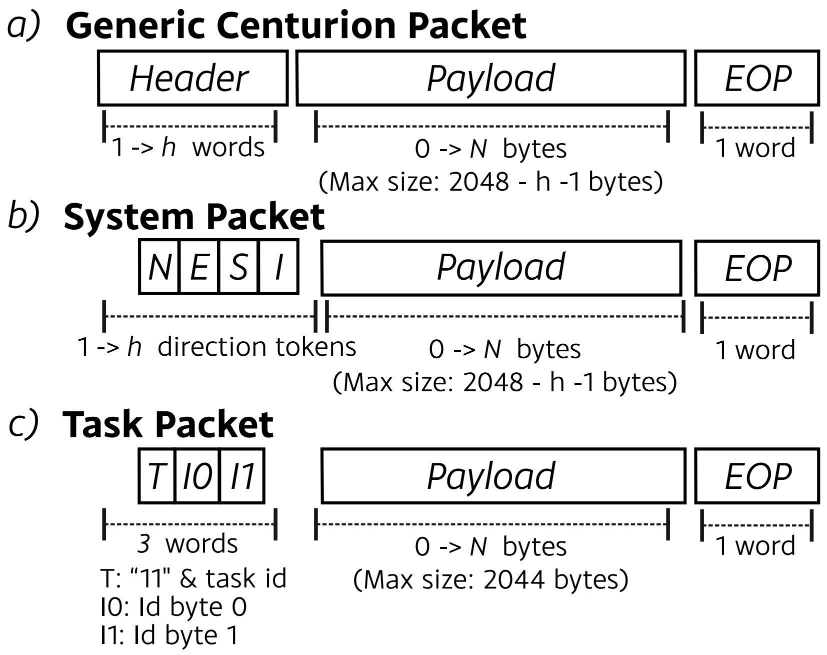


Figure 4.3: Structure of valid Centurion packets: *a)* shows the generic structure of Centurion packets, consisting of: a header (compulsory, at least one 9-bit control word), data (0 bytes or more) and a tail marker consisting of the EOP control token. *b)* System packets have deterministic routing and so the header contains a list of routing instructions. This tells the packet which path to take through the NoC, turn-by-turn. At each node the router reads the control token at the start of the header, deletes it from the packet and then forwards the rest of the packet on in the specified direction. Thus a system packet must always have an "Internal" or "RCAP" routing token as its final word in the header. *c)* A task packet has non-deterministic routing in that the routing direction is looked up from the router's routing table. With the two MSBs set of the 9-bit word (bit 8 and bit 7), this signifies the task packet and the ID of the task is encoded in the lower 7 bits of the word. This task ID is extracted and used to look up the routing direction from the routing table.

Routing System Packets

The routing of system packets is based on the header deletion routing of the Spacewire standard [100]. A path between the source and destination are calculated by the source node consisting of “North”, “East”, “South” and “West” tokens for each router that the packet will visit on its journey to the destination node. At the final router an “Internal” token is added to forward the data on to the router’s attached PE. These tokens are then prepended to the start of the packet to form the routing header. When the packet arrives at a router, the router will interpret the first token as its routing instruction and will also remove it from the packet header. It then ignores the rest of the routing instructions and these are forwarded on to the next router in the direction of the first token; along with all other packet data on that input port until the EOP packet has been forwarded, at which point the routing connection is closed. When the “Internal” token is encountered first at a router then the router knows the packet is destined for this node and the packet is forwarded on the Internal channel to the attached PE. An example of this routing over four nodes is shown in Figure 4.4.

Routing Task Packets

Routing of task packets allows routing-table based routing of application packets. Now the packet header is a single word: the “task packet token” using the upper two bits of the word, and the task id is inserted into the lower 6 bits of the word; allowing for 64 unique tasks. Deadlock avoidance is required as routing tables are often randomly initialised in the experiments. Together with dynamic task switching this may lead to non-deterministic behaviour. The scheme Centurion employs to achieve this requires a weak-unique identifier for each packet, which is stored in word two and three of the packet as data words (i.e. control select bit kept ’0’). When a task packet arrives at a router, the router uses the task id as a look-up for the router’s internal routing tables. This look-up will return the routing direction for the packet and all of the packet’s data, *including* the packet header word, is forwarded in the requested output direction until the packets EOP token is encountered. An example of task routing over four nodes is shown in Figure 4.5.

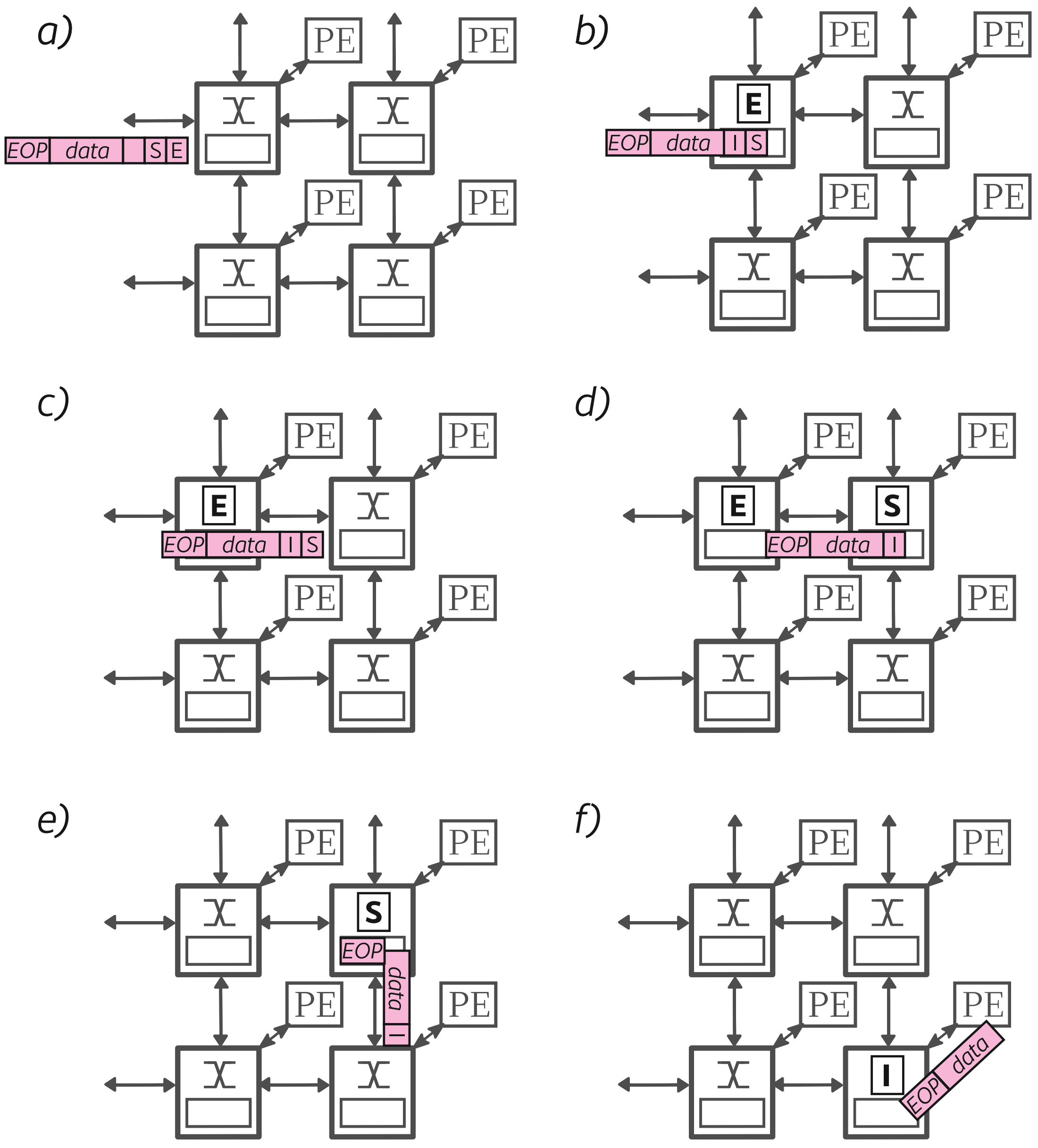


Figure 4.4: Example of routing a system packet: a) The packet arrives at the West input port and fills the three word FIFO. The first word in the header is the `EAST` control token b) The router controller sets the switch up to connect the East output port to the West input port. The EAST control token is deleted from the packet. c) The header and data flow through the router and arrive at the West input port of node 2. d) Node 2 decodes the `SOUTH` control token and sets the switch up accordingly The `EOP` token passes through node 1 and so the connection is to be closed down. e) The packet is routed south, on arrival at node 4 the `INTERNAL` control token is encountered and a channel is set up from North to Internal f) The `EOP` causes node 2 to close the channel. Data is flowing into the node 4's attached processing element via the Interal output port. This continues until the `EOP` token is encountered.

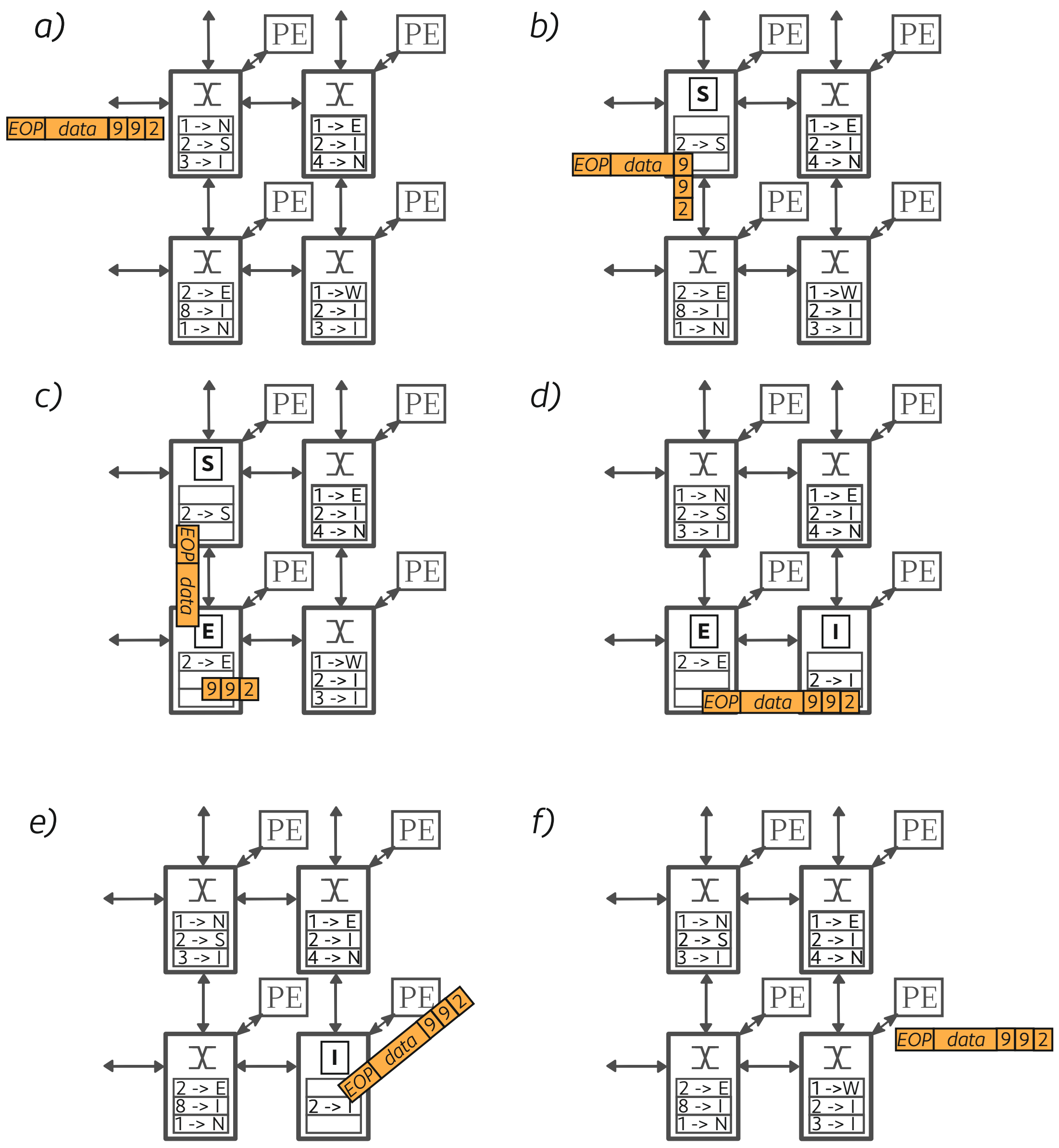


Figure 4.5: Example of routing a Task packet: a) The packet arrives at the West input port and fills the three word FIFO. The first word in the header is a task packet control token with a task ID of 2. b) The router controller looks up the task ID of 2 in the routing table and finds that the packet should be routed south. It sets up the required channel and does not delete the packet header word. c) The header and data flow through the router of node 1 and arrive at the North input port of node 3. Node 3 decodes the packet header and looks up task 2 within its routing table. Its instruction is to route task 2 packets East and so it sets up the channel. d) The `EOP` token passing through Node 1 causes it to close the channel. The task 2 packet arrives at Node 4 and its routing table entry for task 2 packets is to route the packet internally. This is how a task packet is eventually sunk by a node. e) As the packet is routed internally on node 4, this causes the `EOP` token to pass through node 3 and close the channel. f) The task 2 packet arrives complete at the internal processing element of node 4. Note that the task header is still complete with no tokens removed (unlike with system packet routing).

4.3.2. Data Channels

As seen earlier in this section, the grid layout of Centurion requires each router at a node to have a connection to its four cardinal neighbours and also a fifth connection to its internal processor. To minimise the resources needed for each router, virtual channels are not supported by the design. This reduces the amount of packet buffering required at each node to a minimum as virtual channel support would increase the amount of input buffering by the number of virtual channels that are supported. They would also significantly increase the complexity of the router control logic. Thus even support for one or two virtual channels would lead to a large increase in required hardware resources, of which it is likely that a large amount will remain unused (as it is unlikely that an application will constantly use its virtual channel allocation across the entire NoC). The lack of virtual channels does make deadlock a challenging issue, thus a three word buffer for each data channel is required for the deadlock avoidance scheme described in Section 4.3.4.

Each of the router's five data channels support full-duplex operation and are connected to either side of the five-channel switch, allowing up to five packets to be routed concurrently. The channel interface consists of the previously seen 8-bit data word and 1-bit control/data select, complemented by handshake signals *data_valid* and *read_enable*. As seen in Figure 4.6, a word transaction requires at least three clock cycles to complete giving Centurion a net max-throughput of just over 250 MBit/s.

To reduce the length of signal paths between routers for performance and to ease a grid-style implementation footprint, the signals from the input and output ports are registered. This register is extended to a three word FIFO on the input channels as the "task packet" routing scheme requires both the task packet header and the two word anti-deadlock header when making a routing decision. The input channels output several status signals for the router controller: when a start-of-packet (SOP) token is detected, the two words of the anti-deadlock header of the current packet and also a timeout notify signal when a packet SOP header has been waiting in the input FIFO for longer than a programmable threshold (required as part of the deadlock avoidance scheme, Section 4.3.4). The output port emits one status signal: when a EOP token

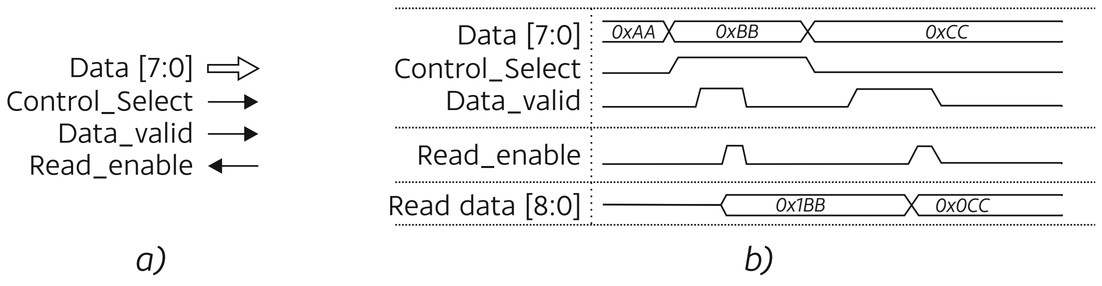


Figure 4.6: Data transmission signals for the NoC data channels. *a)* The signals required to send one NoC word of 9-bits. The data byte is contained in the `data` bus and the `control_select` distinguishes between data and control tokens. The `data_valid` signal indicates to the remote end when the control and data signals are valid. The remote end drives the `read_enable` signal to indicate that it has read the valid data.
b) An example of two words being transmitted. The values 0x1BB and 0x0CC are transmitted to the remote end. Note that with the 0x0CC transmission the remote end does not respond immediately to the data valid signal rising and so the data valid signal is held high, also resulting in more than three clock cycles being required to send the data.

is detected indicating that the packet has been sent, this is used by the router control logic to close down the connection between input and output channels. The output port also has an input signal that controls the output enable of the channel. This allows the router controller to delete the header of a packet which is required for the "system packet" routing scheme.

These signals are shown in Figure 4.7 and the general layout of input and output channels can be seen in Figure 4.8.

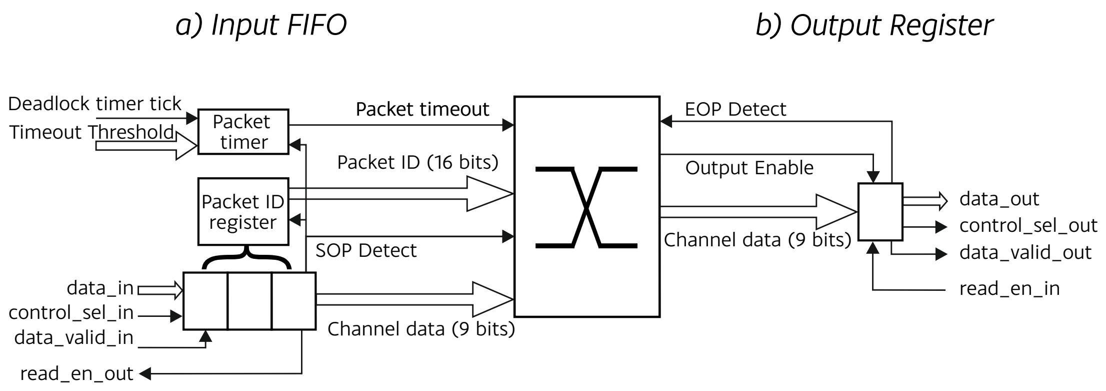


Figure 4.7: The layout of a data channel consisting of *a)* an input port and *b)* an output port, with the switch in between. *a)* The input port combines the data handshaking signals with a three-word FIFO. Once this FIFO fills up with a valid control token in the first position, the `SOP Detect` signal is set to indicate to the input controller state machine that a new packet has arrived. This SOP signal also triggers the loading of words 2 and 3 into the `Packet ID register`, storing the packet ID in the case that this is a task packet and deadlock mitigation is required. The `SOP detect` also triggers the packet timer that records the number of `Deadlock timer ticks` that elapse between the SOP detection and the packet being routed to an output port. *b)* shows the output register, a simple 9-bit register that holds the data until the `read_enable` signal is pulsed from the remote end (an input port). The `Output Enable` signal can be used to skip words from transmission and is used for header deletion when routing system packets.

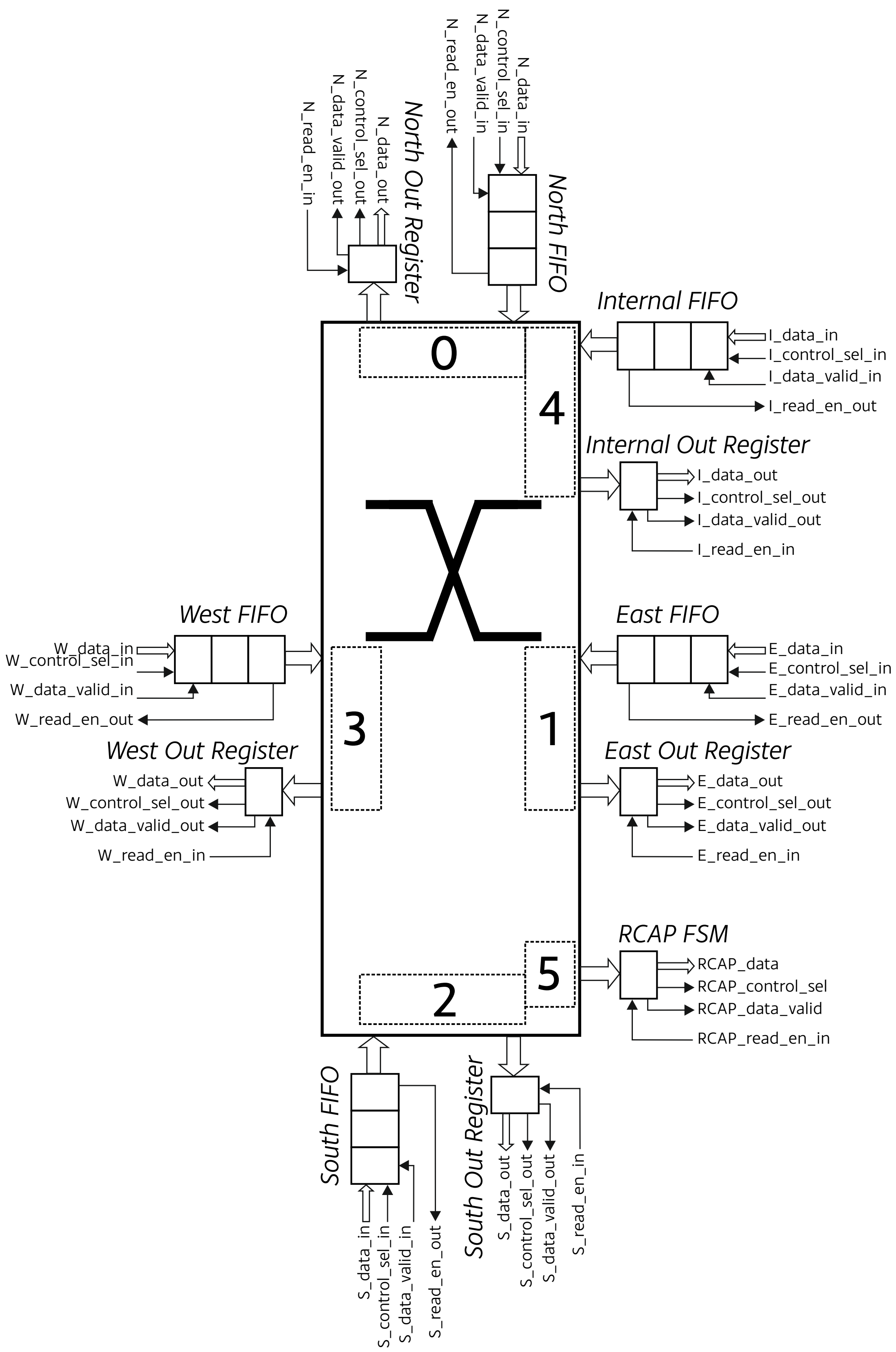


Figure 4.8: An overview of all of the data channels in the Centurion router. Note that the RCAP is a write-only endpoint and so does not have an input port. The numbers represent the index to the switch for each port i.e. a switch configuration of $3 \rightarrow 1$ will connect the West input port to the East output port.

4.3.3. Router Configuration Port

As seen in Figure 4.2, there is a sixth data channel on the router: the *Router Configuration Port (RCAP)*. This channel only acts as a data sink (i.e. cannot output data) and is used to set the router's configuration registers, routing tables and to configure the *Configurable Intelligence Block*. A special packet header is sent to the RCAP that selects which register or data sink the data words following the header are loaded into. A state machine decodes the format depending on which part of the router is being configured. The format of these packets is shown in Figure 4.9.

The routing table is stored in the RCAP entity and top level ports provide access to the routing table for the main router control FSM, whilst the RCAP provides a convenient access point for programming the routing tables. The table is stored as a linear list consisting of a task ID and a routing direction. The control FSM can then search this table for matching tasks and fetch directions by simply providing an address. This also allows non-primary entries for a task by simply skipping over the first or early entries of a particular task in the routing table. Figure 4.10 shows the layout of the table and its access points. The table is implemented using the LUTRAM available in the Virtex-6 SLICE-M primitive and is currently 32 entries long; a larger table may require the table moving to a block-RAM for a more efficient implementation.

Table 4.2 summarises the settings that the RCAP can set and the registers used to store these settings.

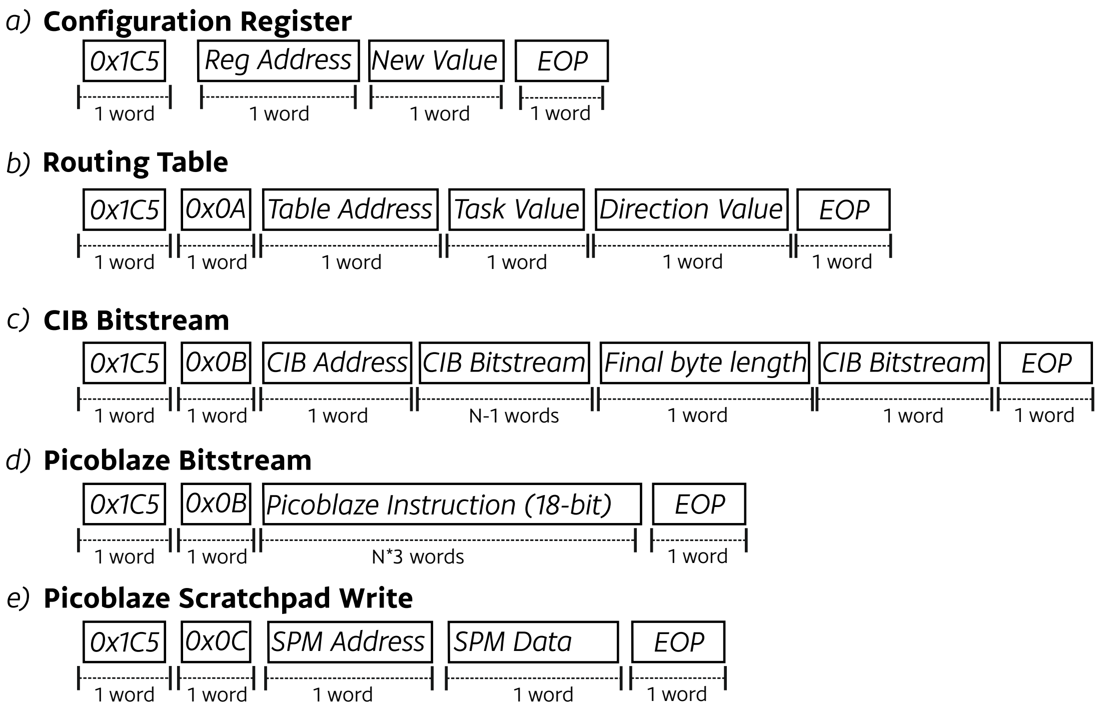


Figure 4.9: Structure of valid RCAP packets: *(all RCAP packets start with the RCAP control token: 0x1C5).*
a) a packet that writes to a single register in the RCAP. The register address is given in the first word and the new value in the second word.
b) a packet that write a routing table entry to the routing table. The 0x0A token signifies this is a routing table packet. The first word after is the index of the routing table that is to be written to. The next word is the new task value, followed by the routing direction.
c) a packet that writes a configuration frame to a *Configurable Intelligence Block* within the router intelligence. The packet structure contains the 0x0B token that indicates an intelligence packet followed by a variable length configuration bitstream for the CIB.
d) a packet that uploads software into the Picoblaze's attached memory. The write address is set to zero when the identification token (0x0B) is received and is auto-incremented as each new instrcution is read (i.e. every three bytes). The identification token is the same as the CIA token as both intelligence modules will not be present in the same design.
e) a packet that writes a byte into the Picoblaze's scratch-pad memory. This allows parameters to be set without updating the Picoblaze's software.

Table 4.2: Registers stored within the RCAP and how the bits are mapped to router behaviour. The address column relates to the address field shown of RCAP packets, as shown in Figure 4.9

Address	Bit/byte map	Description
0x00	Bit 0: Node Reset	*Controls the reset signal on the attached processing node.*
	Bit 1: Node Clock Enable	*Controls the clock enable for the node's clock divider.*
	Bit 2: Intelligence Enable	*Takes the CIA out of reset and enables its outputs to the router/node.*
	Bit 3: Intelligence Clock Select	*When set allows the intelligence to set the clock divider value and enable signal.*
0x01	Bit 4 $\rightarrow$ 0: Clock divider Value	*Division value of the 600MHz node clock.*
0x02	Bit 3 $\rightarrow$ 0: Deadlock timeout value	*Value used to determine if a packet has deadlocked, in clock ticks of the deadlock clock.*
Routing Table Access		
0x0A	Byte 0: Table address	*Routing table entry that is to be modified.*
	Byte 1: Task value New	*Task value for the routing table entry.*
	Byte 2: Routing Direction	*New routing direction for the routing table entry.*
Configurable Intelligence Array Access		
0x0B	Byte 0: CIB Id	*Selects the CIB that the configuration bitstream is loaded into.*
	Byte 1 $\rightarrow$ (N-3): Configuration bits for CIB	*CIB bitstream, penultimate byte is signalled by setting the control bit.*
	Byte (N-2): Number of valid bits in last byte	*Dictates how many bits in the last byte are to be shifted into the CIB.*
	Byte (N-1): Final configuration bits for CIB	*Final byte of CIB bitstream.*
Picoblaze Upload Program		
0x0B	Byte n: program data LSB	*The Picoblaze instruction to write (bytes 7:0)*
	Byte n: program data	*The Picoblaze instruction to write (bytes 15:8)*
	Byte n: program data MSB	*The Picoblaze instruction to write (bytes 17:16)*
	The Picoblaze instruction write address register is incremented per three bytes that are written until the EOP arrives	
Picoblaze Upload Byte to Scratchpad		
0x0C	Byte 0: address	*Picoblaze SPM address*
	Byte 1: data	*Value to write to Picoblaze SPM address*

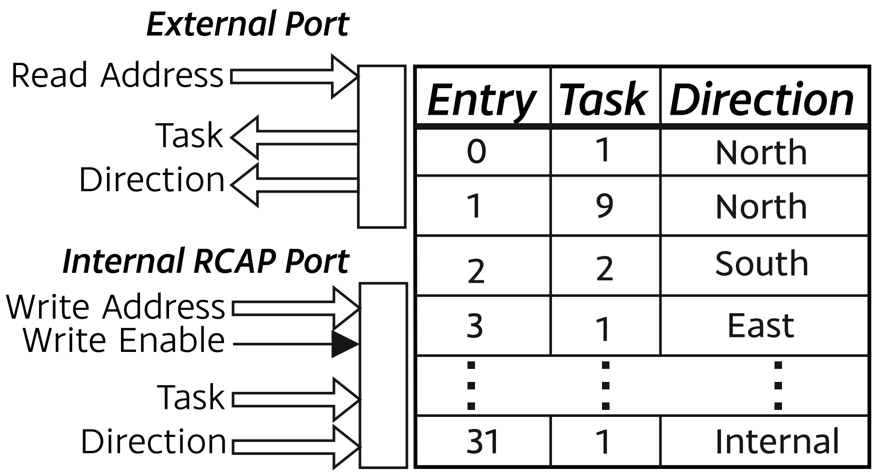


Figure 4.10: The routing table structure. Implementation of the routing table in the FPGA's LUTRAM gives a independent dual-port RAM that is easily used for the routing lookup table. The first port is read only and is used by the router controller for routing table lookup operations. The second port is write only and is connected to the RCAP FSM that decodes incoming packets. Routing table packets activate the write enable and the task and direction are loaded in from the packet as seen in Figure 4.9.

4.3.4. Deadlock Avoidance

Wormhole routing is susceptible to deadlock [56] and must be expected as part of the dynamic nature of the adaptive many-core experiments; as each task switch may invalidate the routing tables. Therefore the system needs to detect and handle deadlocks whilst also still fulfilling the large scalability requirements. Centurion should only use techniques that rely on local information and so cannot employ deadlock techniques that require global analysis of the entire network or constrain the routing decisions that can be made based on routing path information (such as restricting the turns that can be made at a certain node). For scalability, techniques that use a large amount of hardware resources, such as the extra buffering required by virtual channels, are also not considered. The deadlock handling capability is provided by the platform in all experiments and so any enhanced deadlock handling based on global information or information from other nodes should be included as part of the intelligence model. This isolates the transfer of network information so it can be controlled by the intelligence model and not unintentionally enhanced by the capabilities of the platform. Turn-based restrictions for example need either knowledge of the routing path of the packet (the destination node) or what turns have been encountered so far in the packet's journey. Both of these information sets (global system information, packet specific travel information) will need to be integrated with the intelligence model. The social insects for example do not have the ability to transfer global information or "chaining" specific information from one individual to the other to reach a further individual. By only providing a minimal deadlock handling mechanism, Centurion is suited better for researching generic intelligence models. Knobs and monitors specific to deadlock could be added (packet turn history stamping for example) if an experimental intelligence model required it.

As previously mentioned, the input FIFOs on the router ports read and store the weakly-unique packet ID (the second and third words in the packet header) before passing the packet to be routed by the control logic. If this ID already exists on one of the other ports then this means that the packet has looped around and would deadlock if attempted to be routed in the same direction. Thus it is routed out on a different port and so will take a different path through the network. This is repeated should the packet return as the ID is not cleared until its tail End of Packet marker (EOP) passes through the deadlocked ports. This means that eventually the packet is (potentially incorrectly) routed to the internal port, ultimately relieving the deadlock but requiring the packet to be resent by the node that temporarily accepted the packet or for the node to perform a temporary task switch to the task of the deadlocked packet. This not only provides a decentralised, low overhead manner of handling deadlock but also provides several potential monitors and controls that could be used to attach a router intelligence module. Figure 4.11 demonstrates this approach taken which aims to exploit the adaptivity of Centurion.

As seen in Chapter 6, deadlocks are present in many of the experiments. Experiments starting with random routing tables or with large numbers of task switches encounter packets that are terminated by the deadlock mechanism. However, this is to be expected as there is not any analysis of the routing paths used in the experiments as the self-adaptive nature of the task allocations is being explored; the number of deadlocks is used as one representation of the efficiency of a task allocation method. It is acknowledged that the simple deadlock handling combined with the task packet routing is a poor design choice for NoC applications that are designed following traditional design flows, however it is felt that imposing all the restrictions that such a design process entails could have an effect on the emergent behaviours that the experiments aim to exhibit. In the future, design analysis based deadlock handling and avoidance could always be added on top of Centurion's deadlock avoidance system when the routing tables are programmed.

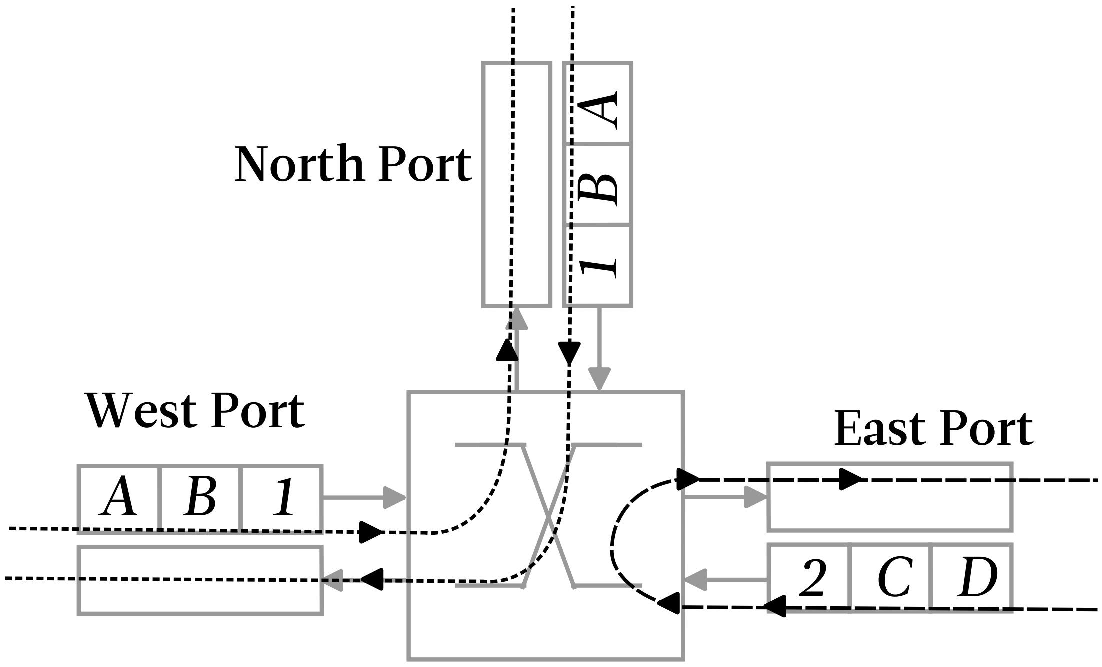


Figure 4.11: Example of deadlock detection and handling, only three ports are shown for clarity. There are two packets in the router, packet 1 is destined for a task 1 node and has a header of 1AB; packet 2 is for task 2 and its header is 2CD. Packet 1 has arrived at the West port and the routing table dictates that it is to be routed North. Packet 2 has arrived on the East port and the routing table says it should be routed back out on the East port. During packet 1's journey it ends up being routed back to this node and so arrives back on the North input. The routing table returns that the packet should be routed North, but it is currently in use. This would lead to a deadlock as the packet cannot progress, however it the ID 1AB matches in both the North and West inputs and so the router now automatically take the second option, routing out on the West port. This is free and so the packet is routed this way and alleviates the deadlock. If the packet ID did not match then the packet would wait for the North port to be free to send, until the deadlock time out time has elapsed at which point the router will try a different direction.

A further layer of decentralised deadlock handling comes from a timeout on each input channel. If the routing for the packet has not been set up before this timeout elapses then the requested routing output direction is deemed deadlocked (either due to this packet or a remote one) and the next option in the routing table for the task is taken. Eventually this index will increase to the point where all options are exhausted and the last option is to take an internal routing path and sink the packet internally for deadlock statistic measurement or further handling and retransmission.

4.3.5. Router Switch and Controller

The data pathways through the router are provided by the routing switch, which is managed by the router controller. The switch connects the five input channels to the five output channels (plus the RCAP channel) and allows five concurrent connections to be set up. The switch takes five inputs, each one setting the input channel multiplexer for each output port. This connects the channel signals together required to forward data across the router.

The settings for the switch originate from the router controller. This module contains seven state machines: one for each input channel (N, E, S, W, I), one that manages routing table look-up requests and the final one that handles the requests for input-to-output channel connections. Both of the shared state machines (routing table requests, output port requests) use a round-robin of all input channels to handle routing table and output port requests. This architecture allows channels to manage themselves whilst avoiding complex synchronisation issues that a non-FSM based design would suffer from. It also supports easier extension of the complexity of the channel setup for supporting additional knobs and monitors.

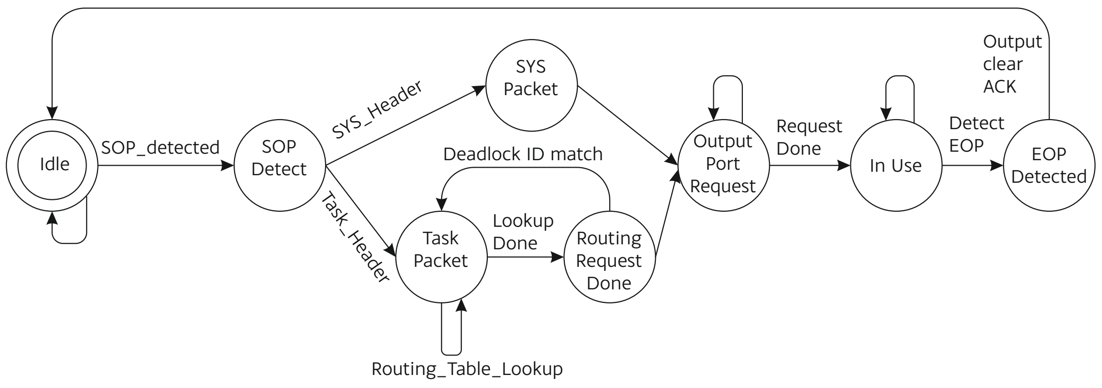


Figure 4.12: The state diagram of the input channel FSM, there is a FSM for each of the five input ports. Once a packet fills the input FIFO on a channel the SoP signal is raised and the flow of the FSM depends on the type of packet detected. Request and Acknowlege signals are used to interact with the shared Routing Request FSM and Output Port Request FSM.

The input channel state machine is shown in Figure 4.12. The state machine flow differs depending on if a system packet or task packet is to be routed. A system packet can skip the routing table lookup step and use the direction given in the first token. In both cases the channel state machine starts by waiting for the SoP token detect signal to go high and then decodes the input word to decide if a system packet or task packet is to be processed. If it is a system packet then the requested direction register is set and the header deletion output for the output register is enabled (removing the routing instruction from the packet header as shown in the example in Figure 4.4). It then can request the output port from the output port FSM.

In the case of a task packet, the task ID is extracted from the first token in the packet and a request is signalled to the routing table state machine to fetch the first routing direction from the routing table that matches this task ID. Once the routing table request is accepted and the direction returned, the state machine then checks if the input channel of this routing direction is currently routing a packet and if so compares the packet ID of this channel with the packet ID of the current channel. If there is a match then routing the packet to that output would cause deadlock (as it is the same packet) and so the state machine re-enters the routing request state but with an increased "option count" telling the router to select the second task entry in the routing table for that task. Once an output port is selected that does not have the same packet ID then that port is requested from the output port FSM.

Once the output port FSM reserves the requested channel for the input port then, for both task and system packets, the input port FSM allows the switch to pass data and waits for an EOP to be detected on the output register. When this happens it means that the packet has passed through the router and the channel between the input port and output port can be closed. The input port FSM signifies this by raising a signal and waits for an acknowledge signal from the output port FSM signalling that it has cleared the output port usage from this input port. The input port FSM can then enter the idle state, waiting for the next packet to arrive.

The routing table request FSM and the output port FSM are shared by all of the input channel state machines and so need some form of arbitration. As seen in Figure 4.13 and Figure 4.14, both use a round robin of all input port FSM request signals when in the idle state to give all ports a fair chance of getting a request. When a routing request is issued, the routing request FSM takes the task and choice offset (for deadlock

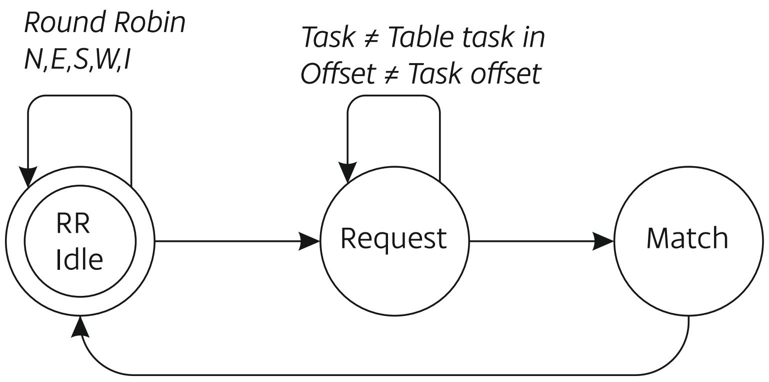


Figure 4.13: The state diagram for the routing request FSM. The idle state continuously re-enters on the next input port request in a round-robin fashion. If the currently selected port request is high then the further states operate on the signals from that port and issue the acknowledge to that input port.

avoidance) from the chosen input channel FSM. It then iterates through the routing table (via the address interface provided by the RCAP entity) until it finds a task entry that matches the task for the input port request. If the offset is not zero then it will keep matching tasks up until the offset count. Once it has a matching task and offset then it loads the routing table direction output into the output direction register of the input port, signals to the relevant input port FSM that the request was handled and then returns to the idle state.

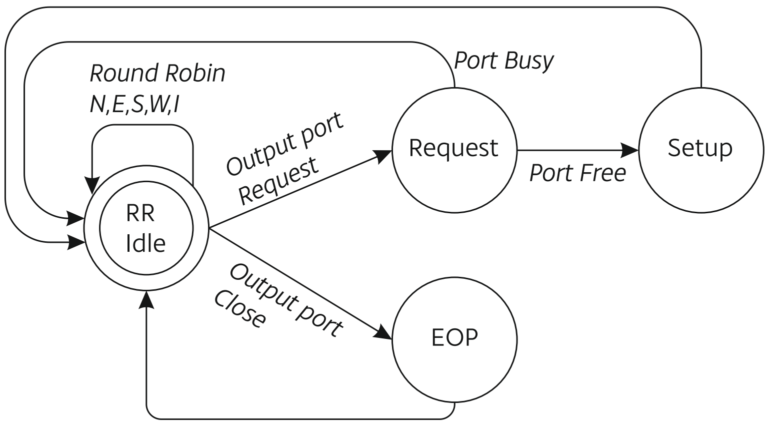


Figure 4.14: The state diagram for the Output Port request FSM. The idle state continuously re-enters on the next input port request in a round-robin fashion. There are two types of request, one is to open a channel between the input and output port; this may fail due to the output port being busy. The second request is to close an output port, in which case the channel between two previously connected ports is disabled and the switch reconfigured to close the connection.

The output port FSM has a similar structure. There are two types of request that an input port can issue: a request to create a connection to an output port, or a request to close a connection to an output port. The round robin checks the request signals for either type of request and responds appropriately. In the case of a connection request, it will check if the output port is currently in use by another input port. If this is the case then the request is ignored until the round-robin revisits the port. If the port is free then the output port marks this port as in-use by setting a bit in the output port status register. It then returns an acknowledge signal to the relevant input port FSM indicating that its output port request has been successful and it can move to the "in use" state. If the request is to close a connection, then the in-use bit for the relevant port in the output port status register is cleared and a different acknowledge signal is sent to the requesting input port.

4.4. Processing Element Design

The processing elements (PE) in many-core systems can vary from simple hardware accelerators to complex, high performance CPUs. All that is required is for the element to include enough interface logic to support communication with the internal data channel of the router. However, due to the research-enabling nature of Centurion we have three main requirements of the processing node:

1. General purpose enough to support various models of computation as deemed by experimental need, with a configuration interface that allows easy and quick development of application models.

2. As low hardware resource overhead as possible to allow as many nodes to be implemented on the same chip as possible, to achieve Centurion's high-scalability requirement.

3. Performance typical of FPGA implemented processors (i.e. 50 - 200MHz, pipelined for an instruction per clock cycle) to enable realistic and accurate modelling of application scenarios.

These requirements suggest a simple CPU as a suitable tradeoff between resource requirements and software flexibility. This would allow both computation models for experiments and real world applications to be developed in software, with a shorter development cycle for adding new features or repairing bugs in the experiment setup. Section 4.4.1 discusses the capabilities, development workflow and implementation of the PE CPU. Section 4.4.2 explains how this is connected to the NoC router channel, so that interfaces for other PEs can be developed.

4.4.1. Processing Core

The PE of Centurion consists of a Microblaze MicroController System (MCS), a hardware resource optimised microcontroller version of Xilinx's Microblaze soft core [98]. The Microblaze MCS shares the same Harvard architecture of its sister processor but has a smaller three-stage pipeline (vs. five stages) and a simple IO peripheral bus instead of an AXI compliant interface. Application code and data share the same memory space (but with separate dedicated access ports to fulfil the Harvard architecture) and are stored using the FPGA's Block RAM resources. The size of this memory space can be configured in power of two's boundaries. Supported processor peripherals include a UART, GPIOs, timers and an interrupt controller. The processor hardware is configured using a Xilinx provided TCL tool, which will output a netlist that can be integrated with the rest of Centurion's HDL. Software development is fully supported by the standard Microblaze C/C++ development flow, a MCS-specific processor definition XML file is generated by the hardware TCL tool which instructs the Xilinx SDK to target the Microblaze MCS instead [101].

The Microblaze MCS used in Centurion is generated with the following hardware parameters:

This results in the following hardware resource requirements for each processing core:

Figure 4.15 shows the general arrangement of the processing element. To support the Globally-Synchronous-Locally-Asynchronous nature of Centurion, each PE resides within its own clock domain and so clock domain crossing (CDC) registers are required for each signal entering the PE that is synchronous to the NoC clock domain. These signals (excluding the NoC signals as these are described in the next sub-section) are as follows and perform the following functionalities:

Parameter	**Value**
Memory size:	8KB
IO Bus:	True
Debug Support:	False
UART:	False
GPO0:	True
GPO0 bits:	9
GPI0:	True
GPI0 bits:	32
GPI1:	True
GPI1 bits:	9
Interrupt Controller:	True
Number of IO Interrupts:	2
Interrupt Type:	edge, edge
Interrupt Polarity:	rising, rising

Table 4.3: List of parameters for generation of the Microblaze MCS using the Xilinx TCL tool.

Resource	**Use**
LUTs:	875
Registers:	669
Total Slices:	194
BRAM:	4

Table 4.4: Hardware resource requirements of the processing element.

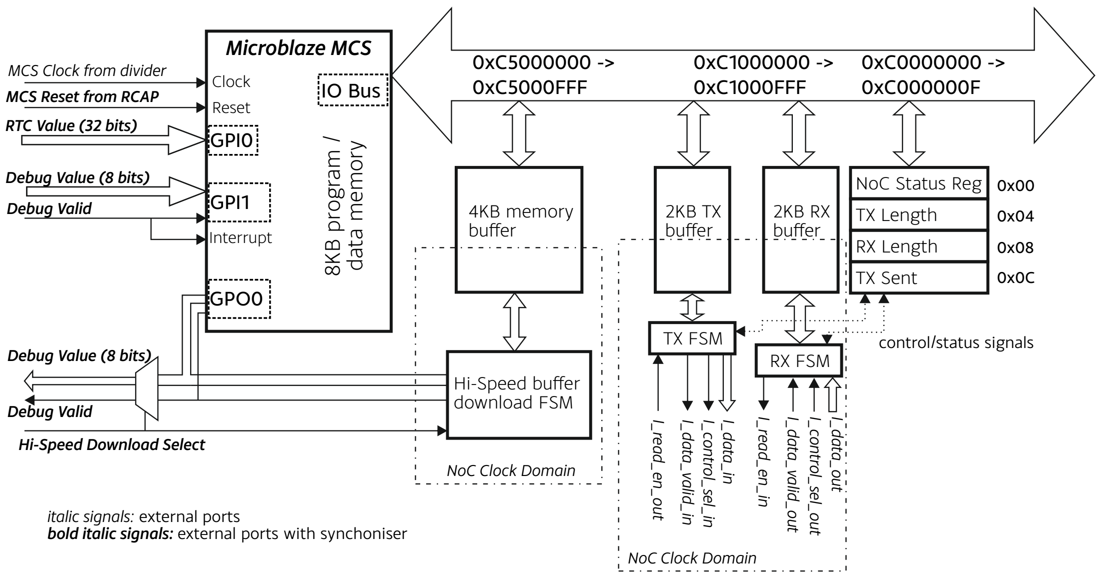


Figure 4.15: A Centurion Processing Element. The Microblaze MCS is generated from the Xilinx TCL tool using the parameters given in Table 4.3. There are two clock domains within the processing node: the `MCS Clock` generated from the clock divider and the globally synchronous `NoC Clock` generated by the experiment controller. The global debug signals to the left of the diagram and the Hi-Speed buffer and NoC interface signals at the bottom of the diagram are synchronous to the `NoC Clock` and so CDC synchronisation techniques are used to cross from the `MCS Clock` clock domain.

Node Reset

As the reset for the processing element is controlled by a register in the router RCAP port, this signal must be synchronised with the PE clock domain. This also helps with timing as it allows the reset paths for the microcontroller's sequential elements to be registered nearer to the flip flops it sets and reduces the need for the placer to place all of the microcontroller logic near to the RCAP register.

Real Time Clock

The Real Time Clock (RTC) is a 32-bit signal provided from outside of the many-core (in the case of the experiments for this thesis: a fully-featured Microblaze on the north-west corner) that provides a timing source and value that is global to all of the PEs. This signal is also synchronous to the NoC clock and so must be registered into the MCS clock domain. This also has the advantage of easing the routing of this high-fanout signal (32-bits distributed all nodes), making it easier for this signal to meet timing requirements. The synchronised version is then connected to the MCS node's 32-bit GPI0 port to allow the node to read the RTC value as required.

Node Debug Interface

Due to the focus of Centurion as a platform supporting NoC research, a link between each node and the experiment control processor is seen as an essential feature for tracking experiment progress and debugging without consuming any of the NoC bandwidth. It can be used by the node to output continuous status information for the experiment controller to monitor. The experiment controller can use this interface to hook into a single node and perform tasks such as setting experiment parameters, fetching experimental results or change the flow of an experiment dynamically via events such as fault injection. By bypassing the NoC the influence of these data transfers on the traffic balance of the NoC will not need to be mitigated, although this link does need to be considered asynchronous as the intelligence in each node can change the node's clock frequency at any time. An overhead of software synchronisation exists to cover this that would not be required if NoC packets were to be used (as all NoC interfaces have asynchronous hardware FIFOs).

This link consists of two channels (one node $\rightarrow$ experiment controller, the other experiment controller $\rightarrow$ node) consisting of a data byte and the necessary synchronisation signals to perform the asynchronous clock domain crossing. On the node side these channels are connected to the GPO0 and GPI1 ports of the Microblaze with the data valid flag connected to the interrupt controller, as shown in Figure 4.15. To be able to run the NoC at 100MHz the paths between the multiplexer and the nodes are registered twice and as such the debug interface should be treated as a multi-cycle path. There is also a direct connection on this interface to the node's external storage buffer. On assertion of the node's *Hi-Speed Download Select* signal the debug output is switched to the output of address 0 of this buffer and the FSM cycles through the buffer's address space for each clock cycle that *Hi-Speed Download Select* is held high.

For the experiment controller this arrangement is similar aside from being connected to the AXI bus of the experiment controller processor and also including a 128-1 multiplexer that connects the signal pair of channels of the experiment controller processor to a node as selected by writing to this multiplexer. This multiplexer has several registered paths in it to help meet a 100MHz NoC frequency and is split into several multiplexers to allow a register between stages in the multiplexer and help scalability. Each row of 8 nodes has an 8-1 multiplexer to select the debug node from the row, the output of which is registered and the lower 3 bits of the node address are used to select the node from the row. Each registered row is then fed into a 16-1 multiplexer, using the upper 4 bits of the node select address to finally select the node to be connected to the debug interface. This split multiplexer design allows the debug interface to be run at 100MHz for the 128-node setup used in this thesis.

Due to the experiment controller only being able to set the debug node select address, a master-slave communication scheme is used whereby the nodes passively output information on their channel to be read by the experiment controller as part of monitoring an experiment without any formal communication between the experiment controller and the nodes. When communication is required, the experiment controller writes the required node to the debug node select address and then writes a command to its output channel. This will cause the debug interrupt to be fired on the selected node whereby it then responds to the command in the software interrupt handler. Supported commands are currently:

1. *Fetch Logs:* the node uploads the contents of its experiment log to the experiment controller via the debug bus and resets its log counter.
2. *Fetch Logs Hi-Speed:* the node sends the number of log entries stored in its storage buffer and the experiment controller then uses this value to read this many entries from the buffer using the Hi-Speed method (i.e. by setting *Hi-Speed Download Select* high).
3. *Stop Experiment:* forces the node out of experiment mode and into idle mode.
4. *Get TX busy:* returns the status of the NoC TX interface, used for detecting deadlock at the end of the experiment.
5. *Set Node faulty:* will cause the node to stop sending or receiving packets and performing CPU operations. This simulates a node-level fault.
6. *Clear Node faulty:* clears the fault state set by the previous command.

As the current clock speed of the processing node can not be known, data transfers over this link need to perform clock-domain-crossing synchronisation. Standard three-stage register synchronisers are used at the hardware level to capture bits across the asynchronous clock domains but such a strategy cannot mitigate the difference in clock speeds between the node and experiment controller. This data-rate synchronisation is carried out in software to minimise hardware resources required at the node and to ease design complexity. The software implementation is currently not optimised for efficiency but as the debug link is primarily for transferring small control bytes and status information (aside from log download via the hi-speed link, which is clocked directly by the NoC clock), this should not cause any issues. The software synchroniser works by requiring each value written to the debug connection to be mirrored by the remote end, once both values match then the end sending the data knows that the word was correctly received by the remote end and clears the valid flag. It then waits for the remote end to also clear its valid flag, guaranteeing that the remote end knows that the value is now no longer valid. The near end can then send the next data word. Figure 4.16 illustrates this flow for the *Stop Experiment* command.

This approach of software synchronisation does not meet the full potential throughput of the NoC debug interface but is a suitable compromise as the clock frequency and clock enable of the node could be changed at any time by the intelligence. Approaches that do not use the valid flag as a handshake either need to enter wait cycles or will need to use global timeslots. To use wait cycles the experiment controller will need to write/sample data at a known rate and the node will need to constantly check what speed it is being clocked at to ensure that it is entering the correct number of cycles between data writing/sampling. This sample rate will need to be several times lower than the slowest clock speed of the node (9.2MHz as described in Section 4.6.1) as the overhead of the node checking its frequency is several cycles (at least three cycles for an IO bus transaction, which will need to be read twice) and so this software synchronisation will have a maximum transfer frequency of less than 1MHz (even if the node is clocked at 100MHz). Timeslots using the global RTC would update the data at a fixed time point after a global RTC transition, at which point the node could sample the data and know that it would be valid. However, the RTC is clocked at 1μs and so again a best case throughput of sub-1MHz could only be achieved without changing the hardware. Finally, for both of these schemes the intelligence resetting the node clock enable would lead to data loss as there are no transaction acknowledgements between the node and experiment controller, whilst in the handshaking method used the experiment controller would at least wait for a response. This could be used to detect that the node has been turned off and the transaction can resume when the node's clock enable is set again.

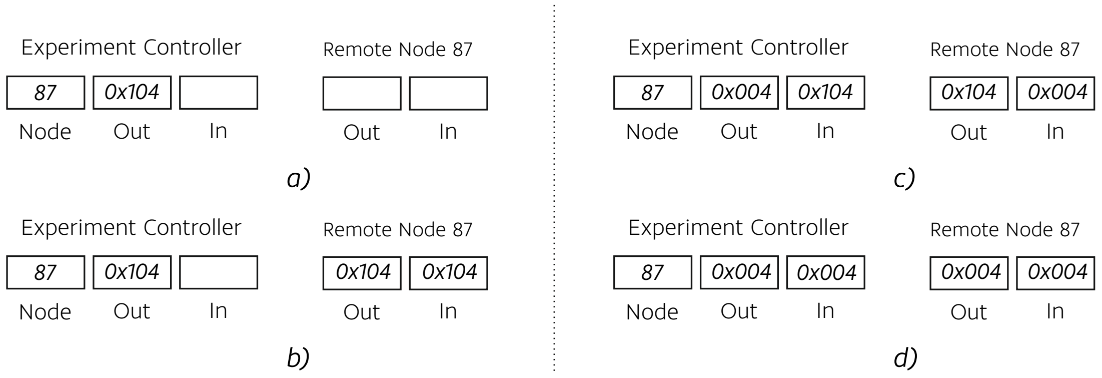


Figure 4.16: Example of data synchronisation over the node debug interface.
a) as the master on this interface the experiment controller initiates the transaction by writing the node ID of the node that it is fetching the data from to the Node ID select register; in this case node 87. It then writes the data (0x04 in this example, the `Stop Experiment` command) it is sending with the valid bit set (MSB in this example).
b) Raising the valid bit will send an interrupt to the node and the debug command handler is entered. The node reads the debug port and extracts the command. It replies with the same command to show that it has received the correct data.
c) Once the experiment controller receives the same value that it sent out then it knows that the remote end has received the data correctly. It can remove the valid signal from the output port to indicate that it is ending the transaction of this word.
d) The remote node notices the valid signal has become low and the node mirrors this to acknowledge the end of the word transaction. Once the experiment controller detects the valid signal going low on the remote end then it can either send the next word or command, send data to another node or stop using the debug interface.

4.4.2. Network Interface

The NoC interface between the node and the Internal port of the NoC router consists of a TX and RX channel and associated control logic. To decouple network delays from the processing unit, packets are sent and received via the 2KB TX and RX buffers (as shown in Figure 4.15). These buffers also act as asynchronous FIFOs and (coupled with a few control signals that cross the clock domains using three stage synchronisers) allow the TX and RX control state machines to reside in the NoC clock domain, resulting in a complete decoupling of the node clock domain and the NoC clock domain; a Globally-Synchronous-Locally-Asynchronous paradigm.

To send a packet the node reads the NoC interface status register and waits for the `TX_Busy` bit to be cleared. Once it is cleared then the node is free to modify the data in the TX buffer and so loads the packet data to be sent into the buffer starting from address 0, including the required packet header and EOP at the end of the packet. The node can then send the packet by writing the number of words to be sent to the TX length register. At this point the TX FSM will take control over the buffer and NoC interface, sets the `TX_Busy` bit and then sends the packet word-by-word into the Internal port on the router. Once the number of words specified in the TX length register have been sent, then the TX FSM clears the `TX_Busy` bit and a new packet can be sent.

For received packets the RX state machine assumes an empty RX buffer on reset. Once valid data starts arriving on the router's Internal port then this data is stored in the RX buffer from address 0. All words are stored until the EOP token is encountered, at which point the RX FSM loads the RX length register with the length of the received packet and sets the `RX_Valid` bit in the NoC interface status register; signalling to the processor that new data is available in the buffer. The RX interface is now blocked and can not receive any new data. This allows the processor to react to the set `RX_Valid` bit and read the packet out of the RX buffer. Once the processor has finished with the data in the RX buffer, it sets the `RX_Ack` bit in the NoC control register and then clears this bit. This signals to the RX FSM that the data in the buffer has been used and it can now resume receiving new packets.

4.5. Embedded Intelligence Using PicoBlaze

The embedded intelligence models need low-level access to the signals of the router to sense and change its behaviour. The decision pathways are extracted from the biological models and translated into a form that is suitable for implementation with Centurion. Early experiments, one of which is presented in [102], used dedicated hardware circuits for implementing the models. Developing and debugging these models became quite a challenge as they interact directly with the internals of the router control state machines; issues when developing could lead to undefined behaviours of the router control logic. Therefore, it was decided to implement the embedded intelligence using a programmable method, with a knobs and monitors interface standardised for all intelligence models even if they do not make use of the signals. One of these implementation modules is the *Configurable Intelligence Array* (described in the next chapter) which allows dedicated hardware pathways to be developed with configurable interconnects and threshold parameters. The second module is described in this section and uses a microcontroller. The pathways are programmed using software to abstract away from the development of dedicated hardware pathways.

The microcontroller used is a Xilinx *Picoblaze* microcontroller [99] which can be embedded inside each router as the intelligence model provider. The Picoblaze can be programmed in assembler language which gives the designer maximum flexibility and simplicity with a small hardware footprint when translating and developing the bio-inspired social insect-intelligence model in hardware.

The program code is uploaded by the Experiment Controller via the node's RCAP interface. As with the CIA, the Picoblaze has access to the internal signals of the router and processor via the monitors and can also affect several settings and behaviours of the router and processor through the knobs. This is shown in the general layout of the Picoblaze system shown in Figure 4.17. To facilitate the implementation of the response threshold models, the Picoblaze software platform provides functions for: interfacing to convert between impulse sequences (spike trains) and binary number representation, logical comparators that generate impulses when vector inputs match, and threshold circuits that act as final decision makers. The intelligence models can then be implemented by tying these functions together to produce a response-threshold decision pathway from the monitors through to the knobs.

The knobs and monitors are accessed through the Picoblaze's IO ports and the following knobs and monitors are available:

Monitors:

- Routing event: This is signalled from the router controller whenever a packet

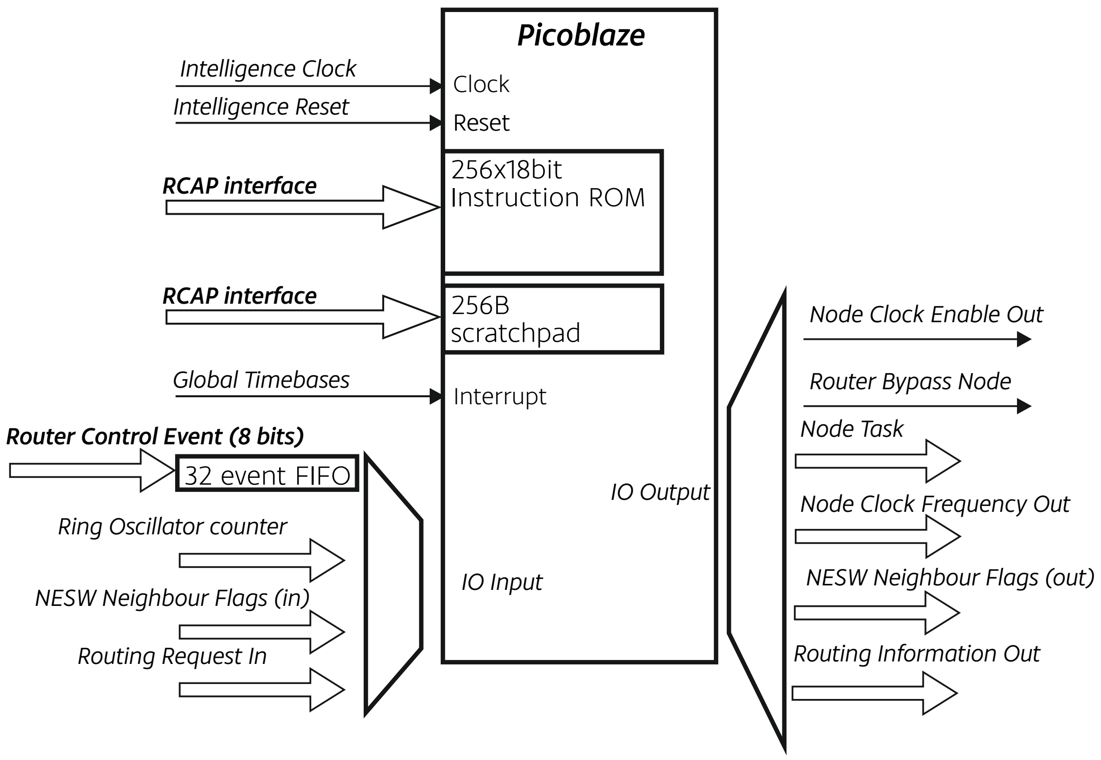


Figure 4.17: Layout of the Picoblaze-based embedded intelligence

SOP or EOP is routed. A 32-event FIFO ensures that events are not missed if the Picoblaze is busy at the time of the event being issued. To minimise hardware resources for this FIFO, the distributed LUTRAM is used. Each LUT in the SLICEM [103] primitive has a 32-bit deep RAM, the slice has four of these LUTRAMs. Thus the 8-bit wide event FIFO can be implemented using only two SLICEM slices.

- Ring oscillator counter: The node's ring oscillator (Section 4.6.1) is fed into this counter to give an indication of the node's current temperature.
- NESW Neighbour Flags: These consist of a 1-bit signal from each neighbouring cardinal node which can be used to communicate directly between intelligence units or indicate a status of the neighbour intelligence.
- Routing Request in: This allows the router controller to fetch its routing directions from the Picoblaze instead of the routing table. This can be used to provide adaptive routing via the intelligence.
- Global Timebases in: Two programmable impulse signals are provided from the Experiment Controller to all nodes. Connected to the interrupt input, this allows

counters and thresholds to change when a time impulse arrives.

Knobs:

- Node Clock Enable: This signal allows the clock for the Microblaze node to be turned off and on.
- Node Clock Frequency: This sets the operating speed of the Microblaze node (Section 4.6.2).
- Router Bypass: When set this signal instructs the router to bypass the internal node port.
- NESW Neighbour flags: a four-bit output that is connected to each of the NESW neighbouring Picoblazes
- Routing information out: A routing direction outputted when the router controller asks for routing information from the Picoblaze instead of the routing table

4.6. Advanced Features of the NoC

The processing element described so far in this chapter could be used to build a basic NoC, however the embedded intelligence block proposed by our social insect-inspired approach (as introduced in Chapter 3) allows the introduction of advanced control options that basic many-core techniques would struggle to utilise effectively. These are a collection of knobs and monitors that are integrated with both the router's RCAP and the intelligence block that is presented in the next chapter.

4.6.1. Dynamic Node Clock Rates

A key element of Dark Silicon, variability and ageing mitigation is the modification of the characteristics of the clock driving the digital logic. For example, a hotspot on the die could be intelligently reduced by lowering the clock frequency of processing cores local to it. As our agent-based approach deals with intelligence embedded at the node-level, our system will also require adaptive clocking at the node level. An

ASIC implementation would have many options available here as clock manipulation circuitry could be added with the node floorplan in the same fashion as other digital circuitry. On an FPGA implementation however, the clock resources are limited and generally not very adaptive post-configuration. Therefore the dynamic node clock rate generator on this FPGA implementation of Centurion may seem convoluted compared to an ASIC equivalent method of achieving this functionality. Currently the clock speed of each node can be configured and the NoC is clocked at a fixed 100MHz clock shared between all routers. The Block-RAM FIFO on the node's network interface provides the required asynchronous clock domain crossing between the NoC and the node clock domain.

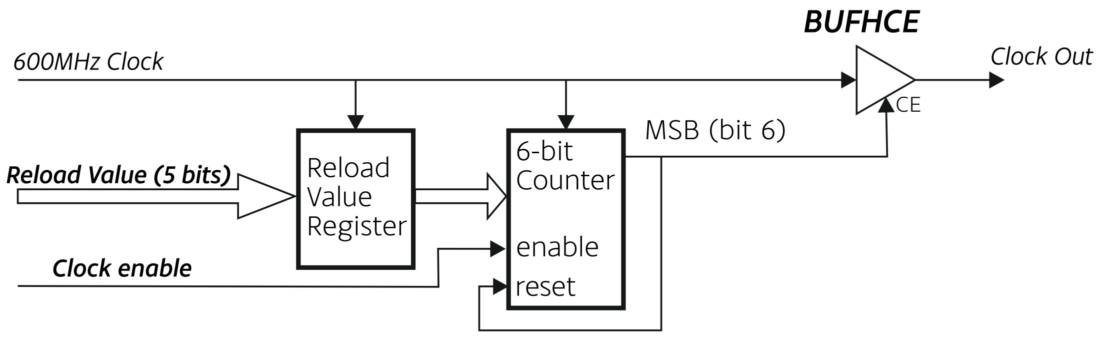


Figure 4.18: Design of the dynamic clock divider. The reload value is provided from one of the router's RCAP registers, this means the signal is far away from the clock divider and so will not be able to meet a 600MHz timing constraint, hence the register within the clock divider. The counter counts up until its MSB is set, at which point a reload of the counter value from the register is triggered. This allows a 600MHz counter to be built without the slow comparison logic that using the divider value as a compare instead of a load would require. The MSB also enables the transmission of the 600MHz clock through the BUFHCE FPGA element to produce the divided clock.

The dynamic node clocks are generated by dividing down a fast global reference clock derived from a PLL within the FPGA, using a loadable count-up counter and the glitch-less output enable function of Xilinx horizontal clock buffers [104] to perform the division. This allows the reference clock to be divided down by integers to get a desired frequency. The counter has a 5-bit loadable value but the counter itself is 6-bits, using the MSB overflow to enable the output of the clock buffer and reset the counter as shown in Figure 4.18. A downside of this scheme is that it provides a "decimated clock" output that heavily skews the duty cycle, each clock "pulse" will be a single high half-cycle of the fast reference clock with the disabled clock buffer output providing a '0' for the low part of the clock-cycle, an example waveform is given in the timing diagram in Figure 4.19. Crucially this means that the hold requirements for all logic driven by the dynamic clock is defined by the hold requirements of the reference clock. The setup time is exempt from this rule as the divider means it is not possible to drive the divided clock at the reference clock frequency; instead the setup requirement is driven by what frequency the tools are set to optimise the node for, in the experiments presented in this thesis this is set to 100MHz.

As the counter has to be clocked from the reference clock, the upper frequency that can be used by the reference clock is limited. By using the MSB overflow to trigger the counter reset helps in this respect by removing the slow combinatorial logic required by a comparison circuit. A higher reference clock frequency gives more dividing options at higher frequencies and so an optimal value is application dependant. As our nodes are synthesised for a nominal 100MHz ideally the clock divider should provide a suitable range of frequencies around this base. By using a 600MHz reference clock the following frequencies are possible (Table 4.5):

Div.	**F (MHz)**	**Div.**	**F (MHz)**	**Div.**	**F (MHz).**	**Div.**	**F (MHz).**
63	300	55	60.0	47	33.3	39	23.1
62	200	54	54.5	46	31.6	38	22.2
61	150	53	50.0	45	30.0	37	21.4
60	120	52	46.2	44	28.9	36	20.7
59	100	51	42.9	43	27.3	35	20.0
58	85.7	50	40.0	42	26.1	34	19.4
57	75.0	49	37.5	41	25.0	33	18.8
56	66.7	48	35.3	40	24.0	0	9.2

Table 4.5: A list of divider values and their associated output frequencies. Only the top 31 values are shown due to the large number of values between a divider value of 30 and 0.

The value for the divider can be set by either writing the associated RCAP port (as discussed in Section 4.3.3) or as an output from the intelligence module. The source is set by writing to the *Intelligence Clock Select* bit in the RCAP.

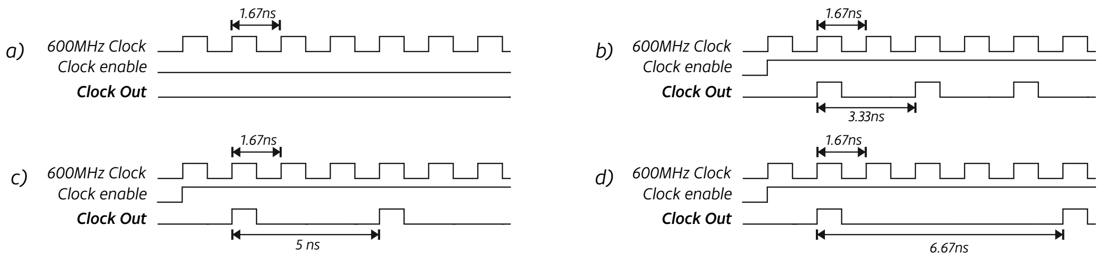


Figure 4.19: Examples of the decimated clock produced by the clock divider.
a) The clock enable signal sourced from the node's RCAP has been cleared and so the counter is disabled. This never enables the CE pin on the BUFHCE and no clock transmission happens.
b) The clock enable signal is now high and the divider value register has been loaded with a value of 0x2F (all values to 1). The counter therefore counts once which sets the MSB and the BUFHCE is high for one clock cycle. The counter is then reloaded to 0x2F and the BUFHCE is disabled again. This results in a divide by two and a frequency of 300MHz is output. Note the "decimated clock" with a duty cycle of 25%.
c) The divider value register has now been loaded with a value of 0x2D. The counter therefore counts three times before being reloaded. This results in a divide by 3 and a frequency of 200MHz is output. The "decimated clock" duty cycle has decreased to 16.7%.
d) The divider value register has now been loaded with a value of 0x2B The counter therefore counts five times before being reloaded. This results in a divide by 5 and a frequency of 120MHz is output. The "decimated clock" duty cycle has decreased to 12.5%.

4.6.2. Local Thermal Sensing

Due to the large effect of the thermal state of the die on the timing performance and power requirements of the application, the local temperature of the node is an important input to the adaptive many-core controller. This allows the intelligence to make informed decisions on the running of the node (clock speed, current task, clock enable) that take the thermal envelope of the application node into account resulting in a self-organisation that can thermally balance local regions to ensure the nodes do not overheat. Unused dark silicon between nodes can be removed by balancing the arrangement of nodes running tasks that make them run "hot" with nodes that may have a lower execution speed or a cooler thermal profile, allowing maximal silicon resource use.

This work only requires a coarse grain view of the node's temperature and so a simple digital thermal sensor will suffice. A *Ring Oscillator* was used due to its simplicity and small hardware resource footprint. This implementation, based on [105], uses 15 buffer stages and a single inverting stage to form and maintain the oscillation as shown in Figure 4.20. The theory of ring oscillators dictates that the frequency of operation is defined by the total signal propagation delay through the delay elements, inverting element and routing delay between these elements. As the temperature of the silicon implementing these elements increases, so will the total delay of the oscillation path and the output frequency will drop. So by reading the output frequency of the oscillator the temperature of the die can be inferred.

The Xilinx timing model does not allow the constraining of combinatorial loops (as the timing engine only analyses register-register paths [106]. Therefore it is difficult to implement homogeneous ring oscillators within each node (user hard macro support for Virtex-6 is essentially defunct due to poor support in FPGA editor). As a compromise, the physical location of the delay and inverter elements are constrained to the same shape of co-located LUTs using RLOC attributes. This gives the tools a good chance of implementing similar routing paths as there is minimal routing outside of hops to the switch-box and back to the same CLB. By using RLOC attributes the ring oscillator can be placed in the same position within each node. Whilst this does not guarantee that they will be measuring the same parts of the node circuitry (due to the placer placing components in different locations within each node floorplan), it does mean that the die's thermal state is evenly sampled reducing the chance of a very local hotspot having an affect on neighbouring ring oscillators.

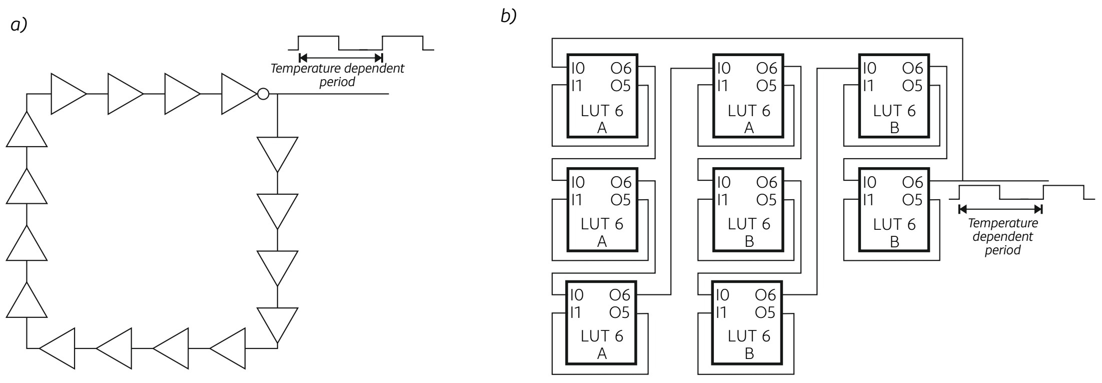


Figure 4.20: The 16-stage Ring Oscillator used for Temperature Sensing and its LUT implementation.
a) The oscillation can be seen from this diagram. On startup the buffers will output a zero (defined by the FPGA LUT), causing an inversion at the inverter's output. This inversion will propagate through the ring of buffers until it reaches the inverter again and is inverted, causing the oscillation.
b) The FPGA implementation using dual-output LUTs configured as buffers, aside from the final inverting stage that is configured as an inverter. As there are four dual-output LUTs in each Virtex-6 slice, two slices are required (marked A and B). RLOCS are used to keep the slices adjacent and in the centre of the node's partition. If the slices where located apart or in different locations within the partitions then we would see a large amount of variation between nodes.

4.7. FPGA Implementation

The 128-node Centurion developed for the experiments in this thesis has been customised to suit the Xilinx XC6VLX760 Virtex-6 FPGA [107] and some degree of design specialisation has been undertaken to efficiently map such a large design to the FPGA device.

4.7.1. Node Implementation

As noted in Section 4.4, the processing element requires 4 RAMs: one 8KB for the processing node memory, one 4KB for the attached storage memory and two 2KB

RAMS for the NoC interface asynchronous FIFO. This can be achieved using 4 dual-port 4KB BRAMs embedded within the Virtex-6 fabric. The router only requires a few small RAMs (for storing the RCAP settings and routing lookup tables) and these can be implemented within the 32-bit RAMs of the SLICE-M Virtex-6 resources. This gives the hardware resource requirements for a single Centurion node implemented on a Virtex-6 specified in Table 4.6:

Resource	**Use**
LUTs:	2808
Registers:	2692
Total Slices:	734
BRAM:	4

Table 4.6: Resource Requirements of a Single Centurion Node. The tools were run with just the Centurion node included and ran until the end of the MAP phase (resources allocated and placed on the FPGA fabric). This means at least 730 slices and four BRAMs for each node are required to be allocated when floorplanning.

These figures are important for floorplanning the design to give Centurion a realistic ASIC physical layout, as they show how many resources need to be reserved for each node (with some margin as it is not guaranteed that the design tools will implement each node in the same way).

Of interest are the resources required by each sub-entity as this gives an indication of the implementation efficiency relative to the design and can highlight problems such as device primitives not being inferred correctly. These are shown in Table 4.7 and suggest that the design has successfully been translated to the implementation.

It can be seen from Table 4.7 that the Configurable Intelligence Array (CIA, described in the next chapter) has quite a high overhead requiring 37% of the total slices for the router. However, the XC6VLX760 architecture means that the design is BRAM limited. Thus, reducing the overhead of the CIA would not allow a larger NoC as the CIA does not require any BRAMs.

Table 4.7 also shows the ratio of NoC resources to processing resources, with 26% of the total slices used for the processing elements when the CIA is included or 43% of the total slices if the CIA is excluded. Whilst this may seem a low percentage as the processing elements provide the application support, it is worth considering that the MicroBlaze MCS processors used are as simple as possible to ease the implementation flow for the tools. A NoC used in a general processing application environment will require processors with features allowing higher performance (e.g. 5 stage pipeline, hardware multiplier and dividers, 64-bit architecture, floating point accelerators) which would significantly rebalance the allocation of hardware resources towards the processing side.

Entity	**Slices**	**Registers**	**LUTs**	**LUTRAM**	**BRAM**
Router	*517*	*2001*	*1187*	*60*	*0*
North FIFO	19	42	21	0	0
East FIFO	16	42	18	0	0
South FIFO	11	42	18	0	0
West FIFO	13	42	18	0	0
Internal FIFO	8	42	18	0	0
North Output Reg	9	12	6	0	0
East Output Reg	7	12	6	0	0
South Output Reg	6	12	6	0	0
West Output Reg	6	12	6	0	0
Internal Output Reg	7	12	6	0	0
RCAP	18	37	63	10	0
Router Controller	97	115	350	0	0
Switch	2	0	130	10	0
Ring Oscillator + Counter	20	55	71	2	0
Intelligence	278	1524	1150	48	0
PE	*194*	*669*	*875*	*218*	*4*
Microblaze MCS	150	449	635	155	2
IO Bus and NoC Interface	44	220	240	63	1
Attached Buffer	0	0	0	0	1
Global Signal Treatment	*23*	*22*	*46*	*0*	*0*

Table 4.7: Breakdown of Resource Requirements for a Centurion Node. Some variation between identical entities (e.g. the input FIFOs) is expected due to variation in the tool optimisations and the degree of LUT packing and slice sharing that occurs at a local level when the design is placed by the placer. The clock divider is not included within the node and so no clock resources are required.

4.7.2. NoC Layout

The FPGA architecture of the xc6vlx760 is generally arranged in a vertical fashion with programmable elements placed in columns running down the device. Primarily, these consist of slices (programmable logic), DSP slices (programmable MAC units) and BRAM blocks (embedded memory). As seen in [107], the xc6vlx760 has 118,560 slices, 864 DSP slices and 1440 BRAM blocks. The Centurion node does not use and DSP slices and so BRAMs are the limiting resource. These are placed in 10 columns across the device that are 144 blocks long. This determines the NoC layout as exceeding more than 10 nodes will mean that BRAMs need to be routed from further below; leading to longer paths, potential timing problems and breaking the tileable architecture of the NoC.

It has been calculated that eight horizontal nodes provide the most efficient use of hardware resources: a single row would result in a best-case efficiency of 91% use of slices and 80% of BRAM resources. The NoC can also be tiled vertically, the consideration here is how many resources will be required for the experiment controller. As the experiment controller complexity increases the number of rows decreases until the design is settled at 16 rows, providing a 16x8 NoC, or 128-node many-core, as shown in Figure 4.1. Also shown in Figure 4.1 are the four interfaces on the North edge of the NoC that connect the experiment controller to the NoC. Each one of these is an independent interface to the experiment controller with its own transmit and receive buffer; allowing packets to be inserted and removed from the NoC at x4 the throughput of a single link. All NoC control signals on other edge nodes are pulled to logic ’0’, ensuring that data cannot be sent off the edge of the NoC and lost or that false data can not be inserted on the edges.

4.7.3. Experiment Controller

Whilst not strictly part of Centurion, the experiment controller plays a vital role in using the many-core. For this thesis it is the primary connection with the external world and so is responsible for setting up experiments, collecting results from the nodes and communicating these results back to a PC for analysis. The debug interface was introduced in Section 4.4.1 and the four NoC interfaces have been introduced within this section, however there is also a large amount of support hardware connected to this processor as seen in Figure 4.21.

The four-lane DDR LVDS interface is used to connect to an external board which in

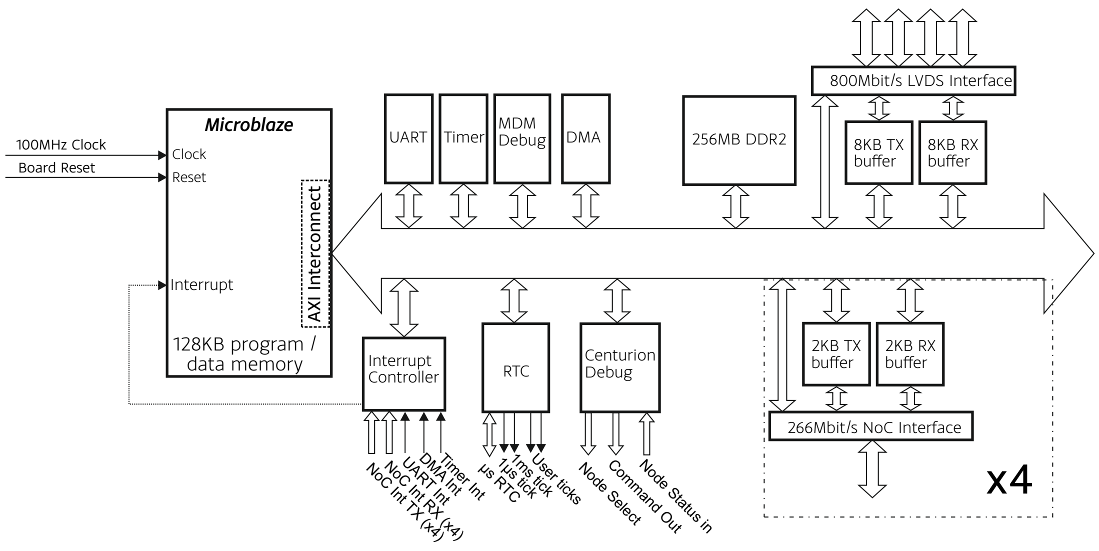


Figure 4.21: Architecture of the experiment controller. A selection if IP cores are connected to the AXI interconnect, some of these are provided by Xilinx (UART, timer, MDM debug, DMA, DDR2 MIG, interrupt controller) and the rest have been developed for use with Centurion (the RTC, debug interface and the NoC interface) and the LVDS interface has been developed for high-speed upload of experiment results and application data to a lab PC.

turn connects to a host PC via a Gigabit Ethernet connection. This is the high-speed link for uploading experimental results and so is an important factor when designing the experiment process. Technical discussion of this interface is beyond the scope of this chapter but the link operates at full-duplex with a total throughput of 800MBit/s in one direction using up to 8KB data frames. The frame overhead is very small and so, with the use of automated DMA descriptor chains, it is possible to nearly achieve 800MBit/s experimental data upload. The interface requires a 100MHz data clock for both transmit and receive, requiring the use of regional clocking resources for capture and re-assembly of high-speed input data.

The custom RTC IP core provides programmable real-time timing ticks to the many-core. It has four output ticks: 1μs, 1ms and two programmable ticks variable from 1μs to 4,294s (floored to the nearest second). These are used by the deadlock mechanism and intelligence units of the nodes. The RTC IP also provides a global “wall time” to all nodes which is a 32-bit count of the number of 1μs ticks that have elapsed since reset, it is possible to reset this count from software by writing to the RTC’s 0x00 register.

4.7.4. Clocking Structure

The main clocking concern when implementing Centurion is the dynamic clock rates introduced in Section 4.6.1. Each of these dynamic clock scalers for each node requires a gated clock buffer to decimate the clock forwarded to the node's processor. In the Virtex-6 the BUFHCE clocking primitive [104] can do this and is more resource efficient than a BUFGCE as it is a local clock buffer that can only clock logic within one clock region; unlike a BUFGCE which can drive all logic elements within the device. Indeed, the BUFHCE is used to forward global clock signals on the BUFGCE network into the local logic.

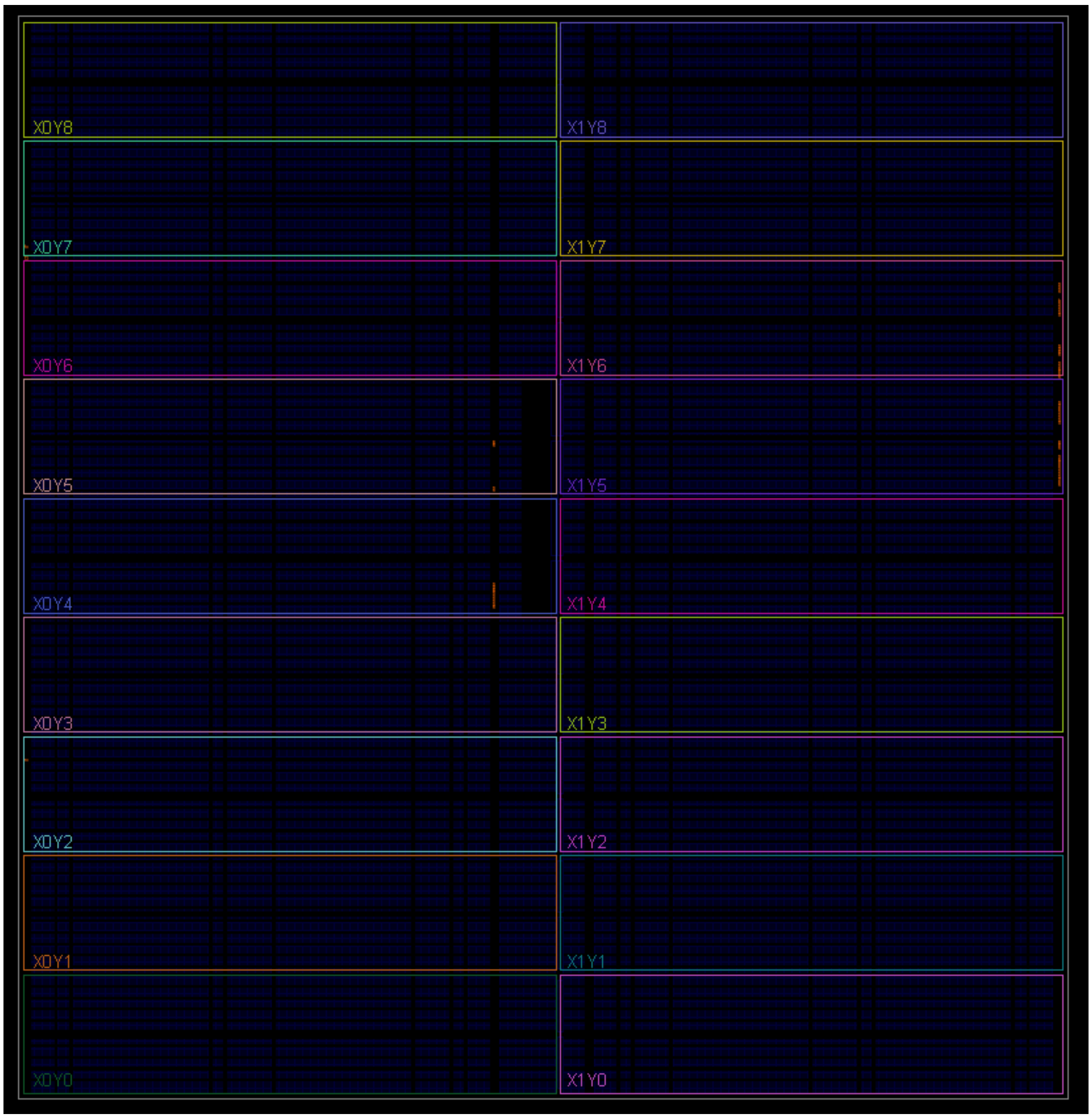


Figure 4.22: A screenshot from the *Planahead* tool showing the 18 clock regions of the xc6vlx760. Blue blocks indicate configurable logic whilst the vertical dark stripes indicate the location of embedded BRAMs, DSPs and IO components.

Each Virtex-6 clock region supports 12 BUFCHEs. One of these is used to forward the global 100MHz NoC clock (distributed via a BUFG) to the NoC sequential logic, another is used to forward the global 600MHz clock used as the reference for the node clock divider and also drives all of the divider logic. The other 10 BUFHCEs can be used for divided clocks for the nodes; although this is not true for all clock regions as if the clock region includes clocked IO ports within the region (i.e. DDR output flip-flops, used on the LVDS and DDR2 RAM interfaces). Figure 4.22 shows the layout of the 18 clock regions within the xc6vlx760 FPGA. Without the experiment controller or BRAM limitations, Centurion's clocking scheme could support 180 nodes all with uniquely controlled clock dividers.

The full scale could not be used due to the clocking requirements of the experiment controller. Care has been taken with the IO planning to group together pins that are clocked by the same output clock but also to maximise the amount of clocked IO within the clock regions dedicated to the experiment controller. The experiment controller takes the input clock (forwarded on an IBUFGDS primitive) and generates several clocks for the DDR RAM controller, two clocks for the LVDS link and also the 100MHz NoC clock. This allows most clocks associated with the experiment controller peripherals to be located in the clock regions in the top row. Some DDR clocks are required in the second row and some LVDS clocks in the third row but, as there are only four extra clocks in both cases, eight Centurion nodes are still supported in these regions. All other clock regions consist of the NoC clock, 600MHz clock and the eight node clocks.
Table A.1 in Appendix A gives a full overview of how the BUFHCE resources within each clock region are allocated.

4.7.5. Floorplanning

To aid an "ASIC style" capture of properties of high density VLSI design that rely on low-level device characteristics (e.g. thermal properties), an ASIC approach to floorplanning the many-core is followed. Whilst the device layout model given by the tools does not guarantee the architecture of the device, it will be a general representation due to the timing model (i.e. two nodes floorplanned near each other will be close, even if the mapping between the tool and the real FPGA layout is not correct at the transistor level). A tiled design will also aid the implementation tool as it does not have to keep reshuffling component mappings until the timing between nodes is met.

As seen earlier, each node requires at least 734 slices and 4 4KB BRAM blocks. If some margin is included to give the tools implementation options and a ratio that tiles efficiently is considered then a single node resource usage requirement of 760 slices (38 slices wide, by 20 slices high) is a good compromise to get a high node density. This is the size of the implementation partition for each node. Figure 4.23 shows a visualisation of the UCF *AREA_GROUP*[104] constraint that is used to partition the FPGA for a single node. The entire device is then floorplanned with 128 of these 760 slice blocks and an area at the top reserved for the experiment controller (the top location was chosen due to fixed IO ports for the DDR RAM attached to the experiment controller process). The pre-implementation partition map for the entire device is shown in Figure 4.24. This partitioning scheme also obeys the clocking capabilities of the device with respect to the number of BUFHCE buffers available per clock region (12 total, with 8 nodes per clock region, the 600MHz global reference clock for the clock dividers and the global NoC clock: 10/12 BUFHCE used per clock region).

The floor planning is achieved with physical constraints set within the UCF file. Each node instance is extracted from the design hierarchy and assigned to an *AREA_GROUP*, for example for node 100:

```
INST "PE_gen[100].many_core_PE_inst/*" AREA_GROUP = "pblock_P_gn[100].mny_cr_P_inst";
```

The *AREA_GROUP* is then constrained to the FPGA resource region requried using the *RANGE* physical constraint:

```
AREA_GROUP "pblock_P_gn[100].mny_cr_P_inst" RANGE=SLICE_X180Y60:SLICE_X217Y79;
AREA_GROUP "pblock_P_gn[100].mny_cr_P_inst" RANGE=RAMB18_X5Y24:RAMB18_X5Y31;
AREA_GROUP "pblock_P_gn[100].mny_cr_P_inst" RANGE=RAMB36_X5Y12:RAMB36_X5Y15;
```

Placing and routing of other components through the *AREA_GROUP* is then restricted:

```
AREA_GROUP "pblock_P_gn[100].mny_cr_P_inst" GROUP=CLOSED;
AREA_GROUP "pblock_P_gn[100].mny_cr_P_inst" PLACE=CLOSED;
```

Due to the 600MHz requirement for the counter within the clock dividers for each node, these have been hand mapped to individual slices in the spare slice resources down the centre of the device (the unused region can be seen in Figure 4.24). This

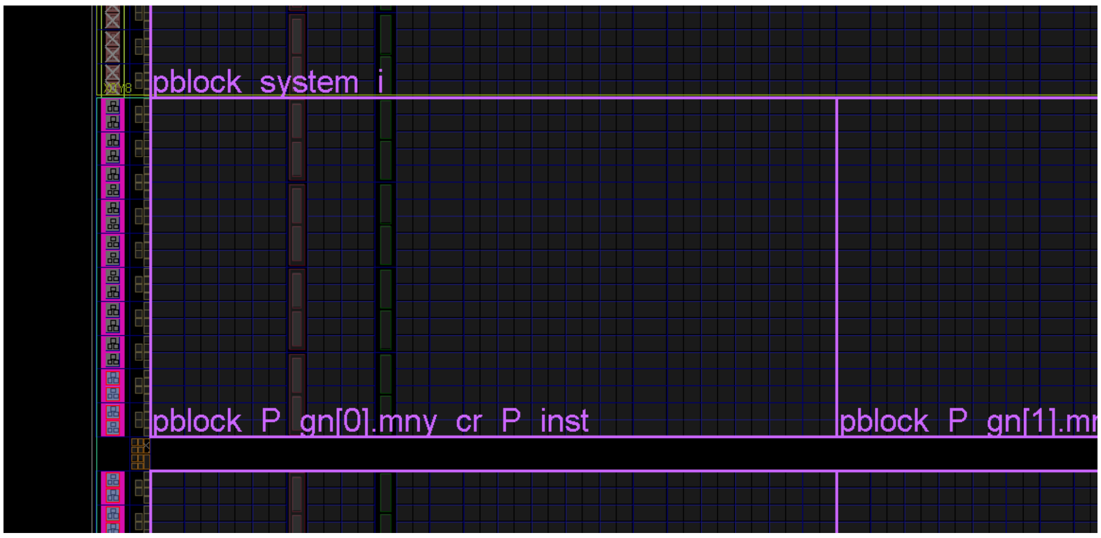


Figure 4.23: The layout the hardware resource partition for node 0 (top left node). Programmable slices have a blue border and have a geometery of 38 slices wide by 20 slices high to give a total of 760 slices within the partition. The four BRAM components can be seen in the column with the purple border to the left of the partition. The green column contains DSP slices which are currently unused in the Centurion node. The location of BRAM columns is not heterogeneous and so some nodes have different layouts.

also helps alleviate timing errors in the mapping phase as these are the fastest paths for the tools to meet. These are placed using the same flow as used with the node floorplanning:

```
INST "clk_div_gen[100].clk_inst/BUFHCE_inst" LOC = "BUFHCE_X1Y16";
INST "clk_div_gen[100].clk_inst/*" AREA_GROUP = "AG_clk_div_gen[100]";
AREA_GROUP "AG_clk_div_gen[100]" RANGE=SLICE_X172Y48:SLICE_X173Y49;
```

As can be seen in the post-implementation net mapping in Figure 4.25 (akin to viewing the ASIC metal layers), the tools successfully generate our floorplan with a runtime of $\approx$ 18 hours, using 50 concurrent MAP and PAR processes running with different mapping strategies and starting cost tables. Further post-implementation figures of the floorplanned design are available in Appendix B.

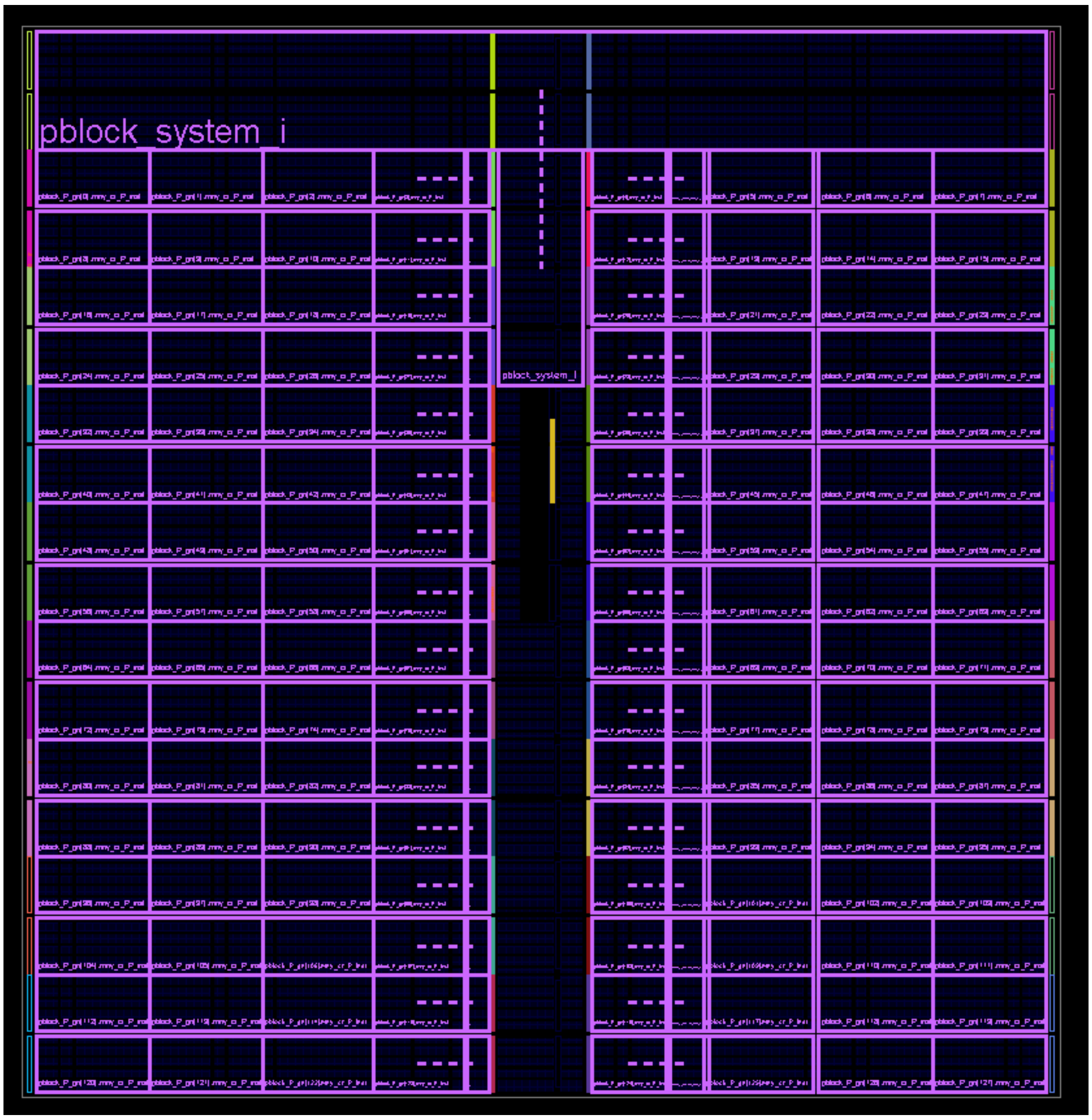


Figure 4.24: All partitions of the 128-node Centurion. The single node partition seen in Figure 4.23 is tiled 128 times to provide a floorplanning of the many-core. At the top of the device is the hardware resources dedicated to the experiment controller, with a small tongue down the centre to ease routing of DDR2 signals and the four NoC interfaces.

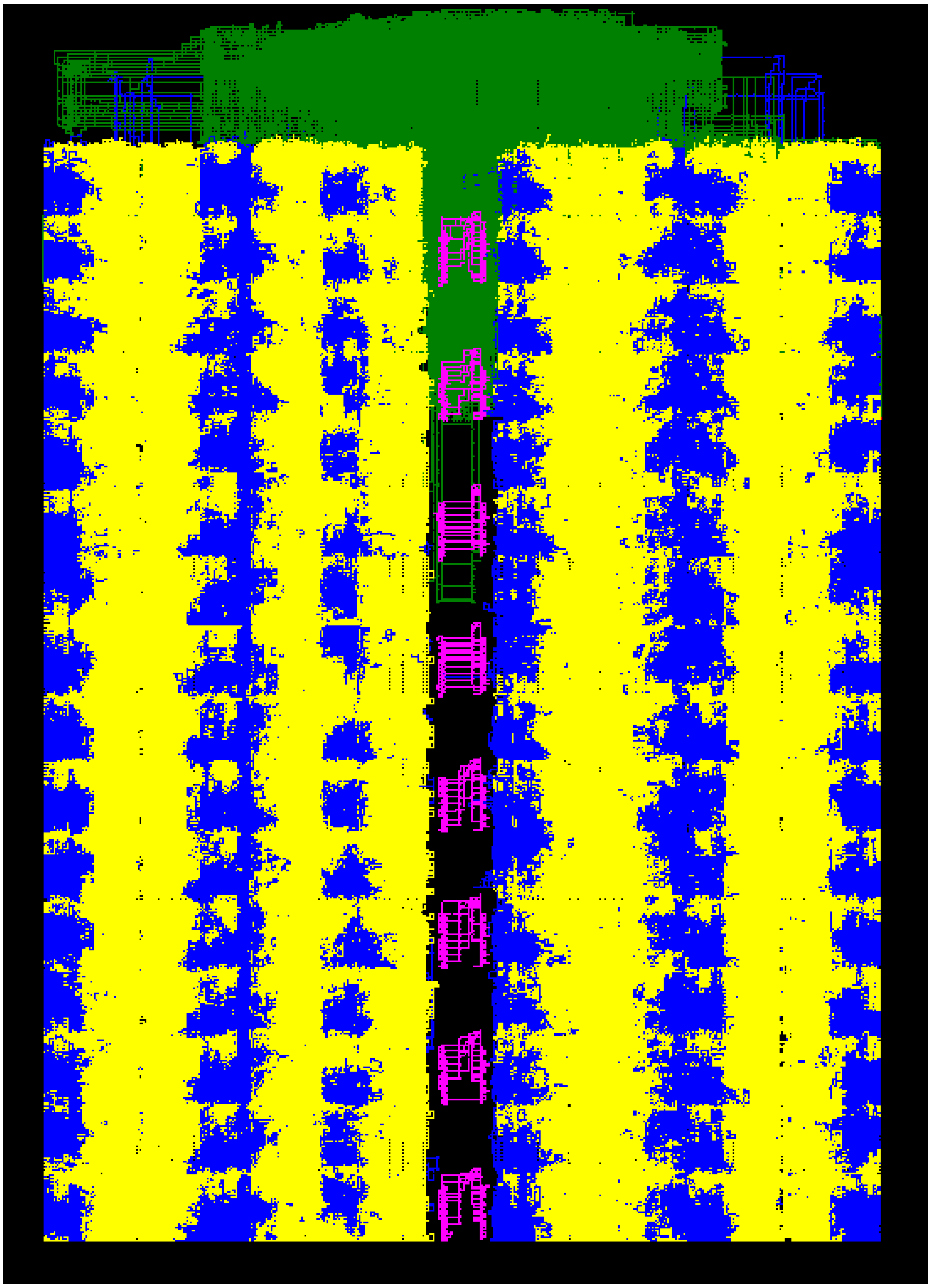


Figure 4.25: A post-implementation view of the hardware resources (nets in this image) used by Centurion. Blue wires indicate wires related to the processing elements (including debug signals seen in the top of the image). Yellow wires are NoC and router signals. Green wires are signals of the Experiment Controller. Purple wires are signals for the clock-divider circuity (but not the clock resources). It shows the floorplanning is successful but also shows the design variation in implementation by the FPGA tools.

4.7.6. Hardware Resources

The results from the floorplanning give some confidence that the many-core has been implemented in such a way that allows the experiments to be representative of a how a future high-density many-core would behave on an ASIC. The total resource usages figures for the top level entities is given in Table 4.8 and provides another verification of the implementation of the design. As expected, the many-core requires the majority of the slice and BRAM resources. The average slice resource use of the nodes is well within the 760 that is allocated for the partition (slice packing and design optimisation will reduce the slice usage, especially for edge nodes that have at least one interface disabled). The BUFHCE resources are not reported as part of Xilinx's design resource utilisation report.

Entity	**Slices**	**Registers**	**LUTs**	**LUTRAM**	**BRAM**
Experiment controller	6,650	12,563	12,805	1,788	56
Centurion	91,470	341,480	331,875	28,377	512
Node average	*714.61*	*2,667.81*	*2,592.77*	*221.7*	*4*
Clock Dividers	391	1664	1149	0	0
Divider average	*3.05*	*13*	*8.98*	*0*	*0*
Total	***99,343***	***356,323***	***346,931***	***30,174***	***568***

Table 4.8: The total hardware resource requirements for the top-level hardware instances. The node averages and clock divider averages match well within expectations from the post-MAP resource requirements for a single node

4.7.7. Software Development

As all of the processors in the system are Microblazes, the Xilinx software development flow can be used albeit with a few customisations to support the large core count. The implementation flow creates a *system.xml* file for both the full Microblaze used in the experiment controller and for the Microblaze MCS used in the nodes. This file can then be used with the Xilinx SDK to generate software projects for both experiment controller development and for node development. Due to the small amount of shared code and data memory in the node (8KB) it is suggested to customise the standard GCC start-up libraries: removing the C runtime library cleanup call found after the call to `main()` in *crtinit.s* for example can save nearly 1KB of code. Aside from this the nodes can be programmed using the standard Microblaze C port, with the following address space for the processor peripherals:

Address	**Size**	**Peripheral**
0x00000000	8KB	Program/Data Memory
0x80000010	4B	Debug Out Register
0x80000020	4B	RTC In Register
0x80000024	4B	Debug In Register
0xC0000000	16B	NoC Control/Status Registers
0xC1000000	2KB	NoC TX Buffer
0xC1000800	2KB	NoC RX Buffer
0xC5000000	4KB	Node Data/Log Buffer

Table 4.9: Address map of the Microblaze MCS contained on each node.

Address	**Size**	**Peripheral**
0x00000000	128KB	Program/Data Memory
0x40600000	64KB	UART
0x41200000	64KB	Interrupt Controller
0x41400000	64KB	MDM Debug
0x41C00000	64KB	Timer
0x50000000	256M	DDR2 RAM
0x60000000	20B	NoC Control/Status Registers
0x61000000	4KB	NoC IF0 TX/RX
0x62000000	4KB	NoC IF1 TX/RX
0x63000000	4KB	NoC IF2 TX/RX
0x64000000	4KB	NoC IF3 TX/RX
0x76000000	16B	Centurion Debug Registers
0x77000000	4KB	Centurion Debug Fast Upload buffer
0x7BE00000	20B	RTC
0x7E200000	64KB	DMA Controller
0xC0000000	12B	LVDS Control/Status Registers
0xC1000000	8KB	LVDS TX Buffer
0xC2000000	8KB	LVDS RX Buffer

Table 4.10: Address map of the experiment controller. The TX and RX side of each NoC interface share an address space, writing to this address space will access the TX buffer and reading from it will access the RX buffer.

Once the software is developed it must then be programmed onto the device. The Xilinx SDK flow does not natively support such a large core count and so an extra step is required at device programming. The Xilinx provided *data2mem* program [101] is used to copy the developed `.elf` files into the embedded block RAMs for each node via the *centurion_bd.bmm* file produced by the tools at the MAP process. This *.bmm* file contains a list of BRAMs that make up the program/data memory space for each Microblaze in the system; thus there are 129 instances within this file for this implementation of Centurion. Specific *.elf* files can then be loaded into the configuration bitstream by the *data2mem* program. The bitstream is then programmed onto the FPGA in the standard fashion using either the *impact* tool or via a prepared flash device. For the experiments in this thesis, the *data2mem* program is configured to load the same *.elf* file into all of the nodes; however this is not a requirement and any *.elf* could be loaded into any node as long as it is compiled for Microblaze MCS and has a memory requirement of under 8KB.

4.8. Summary

This chapter has described the design and implementation of the main novel design element of the work undertaken for this thesis. Centurion provides everything that is required for the many-core aspect of this research platform and the configurable intelligence modules described in the next chapter will provide the hardware required to implement the embedded bio-inspired social-insect models for control of Centurion. The in depth description of Centurion is also crucial for understanding and exploring the adaptive many-core behaviours that are observed in the experiments presented in Chapters 6 and 7.

Chapter 5

The Configurable Intelligence Array

5.1. Overview

The *Configurable Intelligence Array (CIA)* is a hardware module embedded within each router of the Centurion system that implements the biological models described in Chapter 3. This module interacts with sensory and actuator hardware, dubbed *monitors* and *knobs* respectively in this thesis, and makes decisions informed by both current and trend information collected by the monitors. The fundamental decision making part of the intelligence is the *threshold-response* model. This model requires a stimulus and a threshold that states how much the stimulus contributes to the decision. As these experiments target a digital system the stimulus are in the form of digital impulses and the threshold integer values. The goal of the CIA is to provide these threshold-response units with both programmable thresholds and also programmable routing between monitors, knobs and other threshold units. The CIA can then change the behaviour of the attached router (and thus the packet-level dynamics of the many-core) via the knobs.

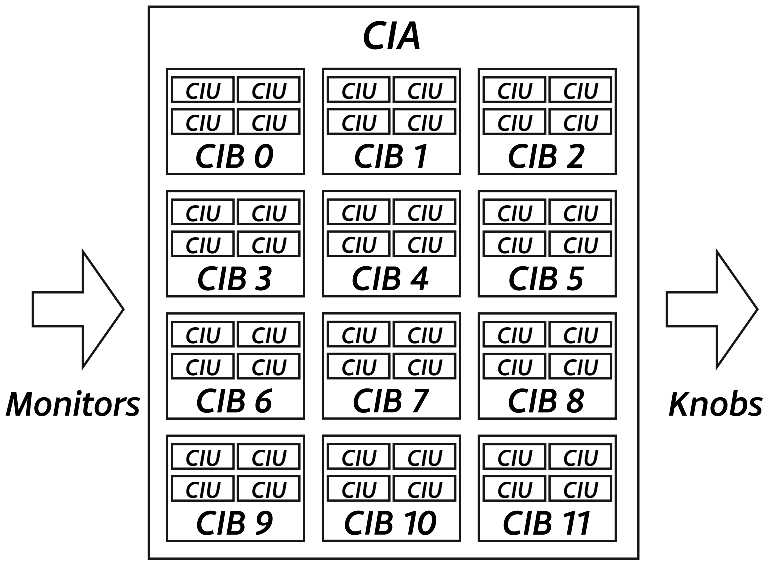


Figure 5.1: The *Configurable Intelligence Array (CIA)*. The array used in this thesis comprises of 12 *Configurable Intelligence Blocks (CIBs)*, which in turn contain four *Configurable Intelligence Units (CIU)* each. The CIUs are the fundamental decision units of the CIA, and the CIB manages routing connections between the CIUs to build decision making pathways.

This chapter describes the design of the CIA, built from the: the *Configurable Intelligence Block (CIB)* and the *Configurable Intelligence Unit (CIU)*. Figure 5.1 shows the general overview of these component blocks within the CIA. The implementation used for the experiments described in this thesis uses 12 CIB blocks arranged as a three by four grid. Each block contains four CIUs and each CIU contains four *thresholder units* which make the stimulus-response threshold decisions. This results in 48 thresholder units available for use to implement the bio-inspired intelligence. This figure is largely driven by spare resources available within the node partitions; indeed this chapter is heavily focussed on hardware resource minimisation as a CIA is present within all nodes and uses a significant percentage of the total node resources: 278 out of 711 slices $\approx 40\%$. This percentage overhead is high, but there are two factors that make this figure more reasonable. Firstly, 60% of this overhead is due to the configurable aspects (routing, programmable thresholds) of the CIA which would be removed in a production system leaving only the decision making elements present; this is discussed in Section 5.4.2. Secondly, both the NoC router and the processing element used in Centurion have been optimised for low hardware overhead. In a system with a more capable NoC router and a more capable processor then these hardware resource requirements will increase and the CIA would drop to a smaller percentage of total node size.

5.1.1. Role of the CIA

As highlighted in Table 3.1 the translation of the biological models revolves primarily on the extraction and translation of stimulus-response pathways in the biological model. Both the Picoblaze (Section 4.5) intelligence software implementation and the CIA hardware implementation will capture these pathways with response-threshold decision making units. Both implementations use impulses extracted from the router control and status signals to feed into the excitatory and inhibitory inputs of the response-threshold units of each embedded intelligence platform. The CIA showcases how these signals can be taken directly from the hardware and processed in a hardware resource and power efficient manner through slow clock speeds and simple digital computing elements (counters, comparators, multiplexors). The CIA also allows many decisions to be made in parallel as the implementation hardware being so cheap (2 FPGA slices per threshold unit, Table 5.3), mimicking the neural structure of nervous systems of small creatures with specific neural pathways dedicated to specific decisions [108].

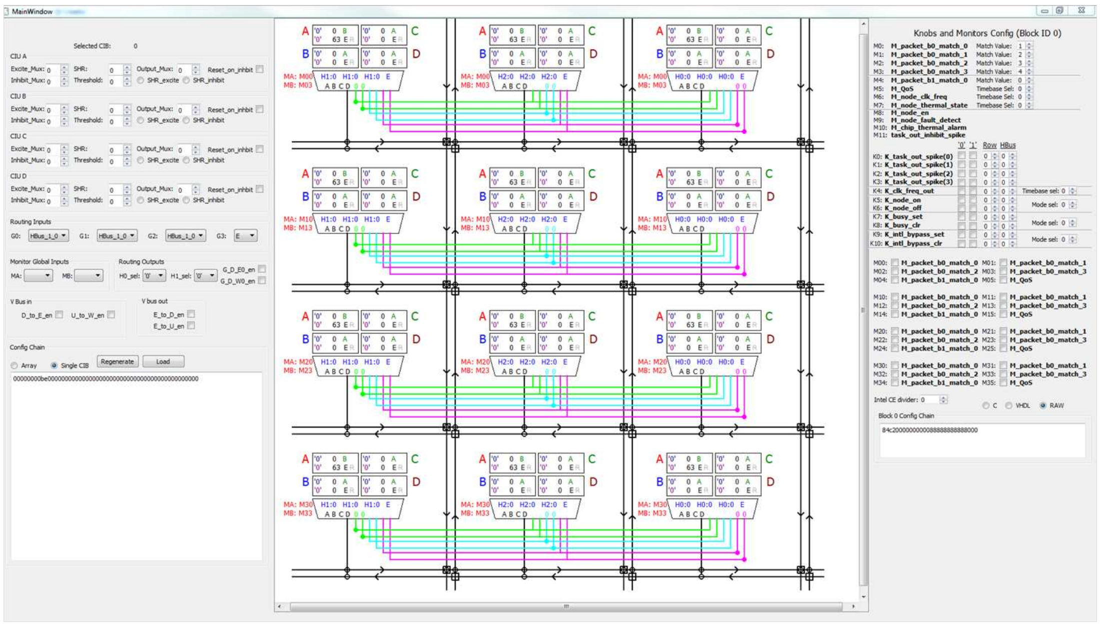
This is a significant abstraction away from the biological models whose implementations typically requires linear systems or differential equations to capture the dynamics of their models. However these modelling approaches require capturing many scale and environmental aspects (physical location of an individual and nestmates, rates of interaction, chemical signalling), some of which will be provided by the hardware system through the fact that the experiments are not modelling the behaviour for the use in models of social-insect colonies, but instead trying to emerge the same high-level properties (e.g. self-organisation, task allocation). As the decision processes of individuals at the stimulus-response level are relatively simple to define, with the complexity coming from the scale of large number of individuals making their own decisions, it is anticipated that the embedded intelligence not having complex decision making capabilities (i.e. its not non-linear) should not be an issue.

5.1.2. Designing Decision Pathways with the CIA

The configurable nature of the CIA requires some strategy for implementing the decision pathways. Ideally, a decision pathway or model presented by the biological sources could be implemented by copying the neural decision pathway of the social-insect directly into the CIA with a translation between the sensory input of the insect to the desired CIA input monitor, and likewise for the actuators or knobs. Unfortunately, neither these pathways are known in that level of neural detail yet nor is the CIA a good representation of the neural structures of the insects. Therefore abstractions are made on both sides of the modelling, the process described being influenced by the modelling processes in [108]. Splitting decision pathways into excitatory and inhibitory factors is a key abstraction when looking at the social insect models and this is reflected in the CIA. Once the excitatory and inhibitory factors are determined, their effect on the decision can be scaled through the response-threshold mechanism with other factors providing extra inhibitory or excitatory input; one of these factors possibly being driven by oscillatory neural pathways or temporal chemical aspects to add a temporal dimension to the decision. The thresholder in the CIA provides this scale and time-based inputs are available as either input impulses to the CIA or as a sampling signal on monitor signals.

Once these pathways are designed they can be transferred into the CIA programmable bitstring (details follow in each of the upcoming sections). To aid this translation, a tool was written to allow routing and threshold parameters to be set and a CIA bitstring generated, as shown in Figure 5.2.

This process is based on the notion of “copying” biology: transplanting decision pathways, translating their input and outputs and expecting the same emergent properties to emerge directly from the hardware implementation. The natural world that biological agents interact with, however, ia very different to the many-core applications that the CIA will be interacting with. Therefore it is also feasible that building decision pathways with the CIA may be more effective with other optimisation tools that are used pre-deployment. Genetic Algorithms manipulating the CIA bitstring could be used for finding both the routing and threshold values of a decision pathways, with a measure of the desired emergent property (decentralised task allocation for example) being used as the fitness function. Other numerical methods could be used to find optimal threshold values for pre-defined decision pathways suited to a particular application task graph or class of task graphs. Simulation of the many-core application could also be used to determine critical knobs and monitors and then pathways could be simulated to provide the required autonomous management behaviours. The CIA would then implement these pathways in the deployed application. This has the advantage that the behaviour of the decision pathways would be known in the context of the actual application and guarantees could be made about the application’s performance. The drawback of this approach is the need to simulate the application and with the CIA working at such a low level (directly interfacing with router control signals, sensing local thermal effects), the simulation would need to be extremely fine grained (at digital circuit level) at huge computation cost per deployed application, or details abstracted away from which would loosen the behavioural guarantee benefits that a simulation approach provides.



```
Selected CIB: 0
CIU A
Excite_Mux: 0   SHR: 0        Output_Mux: 0   Reset_on_inhbit [ ]
Inhibit_Mux: 0  Threshold: 0  ( ) SHR_excite ( ) SHR_inhibit
CIU B
Excite_Mux: 0   SHR: 0        Output_Mux: 0   Reset_on_inhbit [ ]
Inhibit_Mux: 0  Threshold: 0  ( ) SHR_excite ( ) SHR_inhibit
CIU C
Excite_Mux: 0   SHR: 0        Output_Mux: 0   Reset_on_inhbit [ ]
Inhibit_Mux: 0  Threshold: 0  ( ) SHR_excite ( ) SHR_inhibit
CIU D
Excite_Mux: 0   SHR: 0        Output_Mux: 0   Reset_on_inhbit [ ]
Inhibit_Mux: 0  Threshold: 0  ( ) SHR_excite ( ) SHR_inhibit
Routing Inputs
G0: HBus_1_0  G1: HBus_1_0  G2: HBus_1_0  G3: E
Monitor Global Inputs          Routing Outputs
MA:      MB:                   H0_sel: '0'  H1_sel: '0'  G_D_E0_en [ ]  G_D_W0_en [ ]
V Bus in                       V bus out
D_to_E_en [ ]  U_to_W_en [ ]   E_to_D_en [ ]  E_to_U_en [ ]
Config Chain
( ) Array  (•) Single CIB
00000000be000000000000000000000000000000000000000000

Knobs and Monitors Config (Block ID 0)
M0:  M_packet_b0_match_0    Match Value: 1
M1:  M_packet_b0_match_1    Match Value: 2
M2:  M_packet_b0_match_2    Match Value: 3
M3:  M_packet_b0_match_3    Match Value: 4
M4:  M_packet_b1_match_0    Match Value: 0
M5:  M_QoS                  Timebase Sel: 0
M6:  M_node_clk_freq        Timebase Sel: 0
M7:  M_node_thermal_state   Timebase Sel: 0
M8:  M_node_en
M9:  M_node_fault_detect
M10: M_chip_thermal_alarm
M11: task_out_inhibit_spike
                            '0' '1' Row HBus
K0:  K_task_out_spike(0)    [ ] [ ]  0   0
K1:  K_task_out_spike(1)    [ ] [ ]  0   0
K2:  K_task_out_spike(2)    [ ] [ ]  0   0
K3:  K_task_out_spike(3)    [ ] [ ]  0   0
K4:  K_clk_freq_out         [ ] [ ]  0   0   Timebase sel: 0
K5:  K_node_on              [ ] [ ]  0   0
K6:  K_node_off             [ ] [ ]  0   0   Mode sel: 0
K7:  K_busy_set             [ ] [ ]  0   0
K8:  K_busy_clr             [ ] [ ]  0   0   Mode sel: 0
K9:  K_intl_bypass_set      [ ] [ ]  0   0
K10: K_intl_bypass_clr      [ ] [ ]  0   0   Mode sel: 0

M00: [ ] M_packet_b0_match_0  M01: [ ] M_packet_b0_match_1
M02: [ ] M_packet_b0_match_2  M03: [ ] M_packet_b0_match_3
M04: [ ] M_packet_b1_match_0  M05: [ ] M_QoS

M10: [ ] M_packet_b0_match_0  M11: [ ] M_packet_b0_match_1
M12: [ ] M_packet_b0_match_2  M13: [ ] M_packet_b0_match_3
M14: [ ] M_packet_b1_match_0  M15: [ ] M_QoS

M20: [ ] M_packet_b0_match_0  M21: [ ] M_packet_b0_match_1
M22: [ ] M_packet_b0_match_2  M23: [ ] M_packet_b0_match_3
M24: [ ] M_packet_b1_match_0  M25: [ ] M_QoS

M30: [ ] M_packet_b0_match_0  M31: [ ] M_packet_b0_match_1
M32: [ ] M_packet_b0_match_2  M33: [ ] M_packet_b0_match_3
M34: [ ] M_packet_b1_match_0  M35: [ ] M_QoS

Intel CE divider: 0     ( ) C  ( ) VHDL  (•) RAW
Block 0 Config Chain
84c200000000008888888888888000
```

Figure 5.2: A screenshot of the software created to program the decision units of the CIA and connect them together.

5.2. Configurable Intelligence Unit

The key part of the decision making process are the threshold functions within the intelligence models. The *Configurable Intelligence Unit*(CIU) contains the multiple entities within the intelligence array that realise this functionality. An up-down counter combined with a programmable comparison register provides a basic excitatory/inhibitory thresholder in a form that is suitable for effective digital implementation. Programmable multiplexors allow the input excitatory and inhibitory impulses to be selected from a range of signals, the source of these signals is chosen by the higher layers of configuration described in Sections 5.3 and 5.4. The general overview of a CIU can be seen in Figure 5.3.

Due to the CIU being the fundamental entity of the CIA, there are extreme resource constraints on the hardware implementation of this kind of decision pathway. A target of four Virtex-6 slices is set as this provides a good balance between the number of thresholders embedded in each router and the spare resources available in the partition. Each Virtex-6 slice consists of four LUTs, eight flip-flops and extra routing resources such as a carry chain and wide multiplexor support [103]. Each fourth slice in the FPGA fabric is a SLICEM resource [107] and so a CIU can use one of these and still

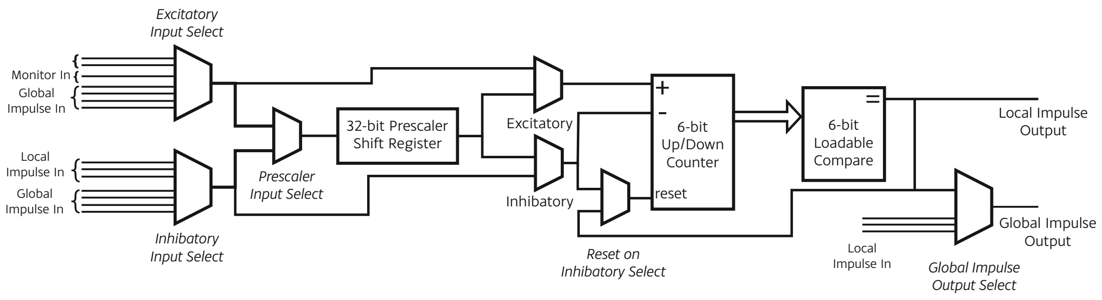


Figure 5.3: A *Configurable Intelligence Unit (CIU)*. The CIU first selects the excitatory and inhibitory inputs and routes one of them through the prescaler shift-register depending on the relevant configuration bit. The excitatory and inhibitory signals are then fed into a 6-bit up/down counter. The output of this counter is compared to the value stored in the comparison unit and an impulse issued if the counter matches this value. The issue of the impulse will also reset the counter, as can an inhibitory impulse if the relevant configuration bit is set.

exhibit good tiling properties. Each component of the CIU is now discussed to show how it was implemented to meet this target slice usage.

5.2.1. Thresholder

The bit-width of counter has an obvious effect on the hardware requirements of the thresholder. A larger counter will offer more flexibility with regards to handling fast impulse trains as the counter will not overflow without the need to cascade thresholders. It is also worth considering the performance of the counter, as faster counters typically need more hardware resources. Due to the tight packing of the CIU in slices it can be assumed that routing delay between LUTs and Flip-Flops will be very low and so a slow counter can be used. There is no minimum frequency requirement for the intelligence, but it will not be faster than the NoC as one of the sub-hypotheses is that longer term behaviours can be managed by low performance resources embedded in spare silicon. Therefore the CIU can be designed for a maximum frequency of 100MHz (the NoC clock frequency), and thus the smallest counter that can achieve 100MHz is required. The counter needs to be able to count in both directions (to support excitatory and inhibitory impulses), have a count enable (so the count is only affected by incoming impulses) and be resettable (to allow strong inhibitory impulses that clear the excitatory response so far).

As discussed in this counter design for Xilinx FPGAs [109], a FPGA hardware efficient counter that fits these requirements is a prescaler based *tri-bit block counter*. This counter design groups the counting stages into threes and when each triplet rolls over the count enable for the next MSB triplet is enabled for one clock cycle. This makes each triplet behave as a prescaler. Thus for each count enable a four input AND-gate is required. Due to the two output LUTs in the Virtex-6, this results in two counter flip-flops being able to share a single LUT and the final flip-flop in the trio sharing the LUT with the triplet count enable ripple. This allows for a counter that packs efficiently into the slice LUT resources. By using two triplets, a 6-bit counter can be implemented using only four LUTs, six flip-flops and thus a single slice as shown in Figure 5.4. This gives an impulse threshold value of up to 64.

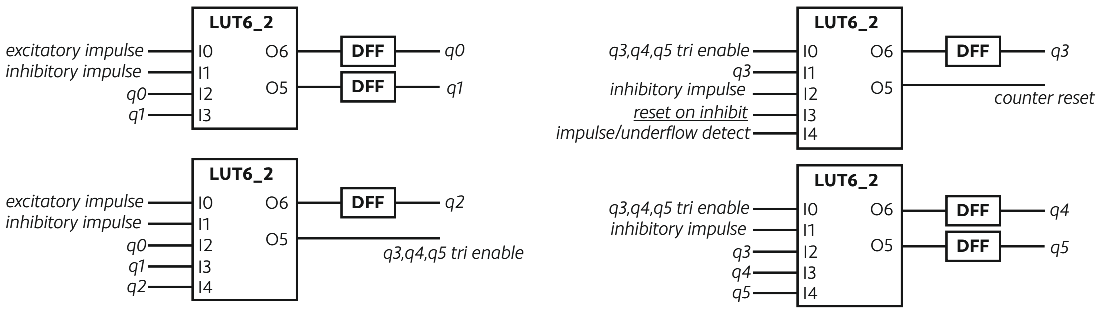


Figure 5.4: LUT implementation for the CIU counter. Two output LUTs are used for the most effective packing. The first LUT determines the next value for bits q0 and q1 of the counter by examining the count up (excitatory) and the count down (inhibitory). In the case of a collision the inhibitory (count down) takes priority. The second LUT generates the q2 bit in the same fashion but also takes advantage of the use of q2 to also generate the tri-block count enable signal for the q3,q4,q5 tri block, n.b. this signal is also driven in the case of count down roll-over. The q3 LUT uses this tri-block count enable signal to know when to increment/decrement q3, by using the inhibitory input to determine the direction that it should count in (if the count enable is set to ‘1’ and the inhibitory input is set to ‘0’ then the count direction should be up). It also uses its spare output to generate the counter reset signal based on the “*reset on inhibit*” configuration bit. The value of q4 and q5 are determined in the next LUT, once again using the tri-block count enable and inhibitory input to determine the direction of the count. These are the only four LUTs that the counter requires and can be contained within a single slice.

The thresholder also consists of a comparison unit of the same width. Six flip-flops are required to hold the comparison value and 3 LUTs are used to perform the AND operation between the stored threshold value and the current counter value. If they match then the counter is reset and an impulse released for one clock cycle. The comparison flip-flops are connected to the CIU configuration chain that is used to shift the configuration bitstring into the CIA.

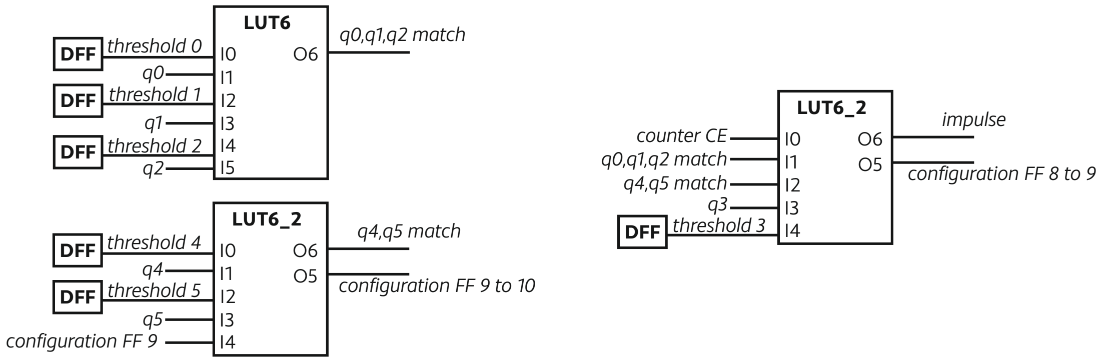


Figure 5.5: LUT implementation for the CIU thresholder. The thresholder performs a 6-bit logical XNOR between the counter output value and the threshold value stored in configuration flip-flops. The XNOR operation is split into three parts q0,q1,q2; q4,q5 and q3. The q3 match operation also aggregates the outputs from the other sub-match units and performs an AND operation to determine if all match units output ’1’. This results in an impulse being raised on the impulse output, however the counter clock enable signal is also included in the AND operation to ensure that the impulse is only valid for a single enabled clock cycle (for when the CIA is being run at a sub-clock frequency via the CE). The O5 outputs on the second and third LUTs show one of the routing tricks employed to allow maximal LUT packing; a configuration chain signal (configuration FF 9) uses the LUT to route directly to the inner FF of the slice. This would be unavailable if the “nX” slice input is already used, in this case by another configuration chain signal using the second FF in the sub-slice. See [103] for more information on the routing of the X LUT bypass path.

The final part of the thresholder is an input prescaler that can divide down either the excitatory or the inhibitory impulse trains. This extends the effective threshold value that the thresholder emits its impulse at. It is implemented as a 32-bit shift register loop with the value loaded in as part of the configuration chain. By loading specific binary patterns into the shift register, power-of-two impulse frequency divisions from 1 to 32 can be emulated, for example the pattern *0101...* will divide by two whilst the pattern *0001...* will divide by four. This gives the entire thresholder unit a range of possible impulse values ranging from 1 to 2048. To save resources there is only one shift register within the thresholder unit and whether it is applied to the excitatory or inhibitory impulse input is determined by a control bit in the CIU bitstring. This allows the prescaler to be implemented with three LUTs and one MUXF7 resource as shown in Figure 5.6. Due to the use of the 32-bit shift register these resources have to be located within a SLICEM slice [103].

This gives an overall dataflow through the CIU thresholder as shown in Figure 5.16,

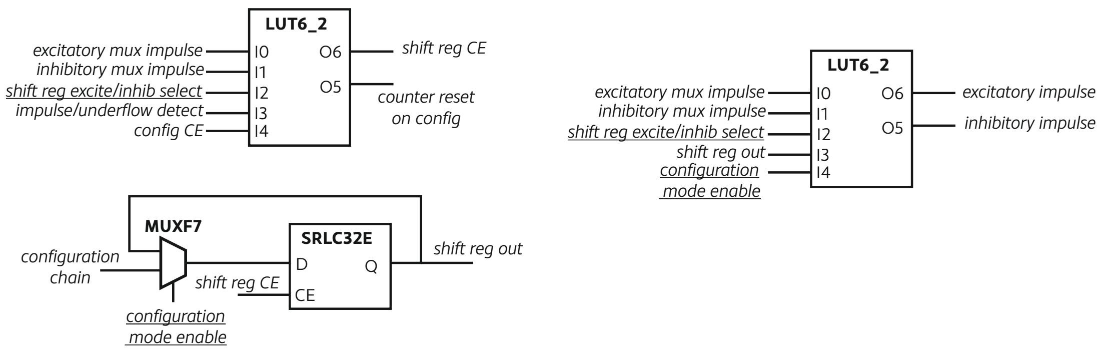


Figure 5.6: The CIU prescaler can be applied to either the excitatory or inhibitory input, determined by the *shift reg excitatory/inhibitory select* configuration bit. The first LUT determines which impulse controls the shift register clock enable. This moves the shift register sequence on by one element and so provides the pre-scaling functionality. It also has to account for when the CIA is in configuration mode and the shift register is part of the configuration chain and so must be shifted with the incoming configuration data. The second output is a helper function for resetting the counter when configuration mode is enabled (and so not related to the shift register functionality). The shift-register simply shifts its output back into the input when the clock enable is high, unless it is in configuration mode in which case the configuration chain is connected to the input of the shift register via a MUXF7 resource to save a LUT. The final LUT selects the excitatory and inhibitory impulses to be sent to the counter. Either the output from the excitatory/inhibitory input multiplexors are used or else the shift register output is used, depending on the value of *shift reg excitatory/inhibitory select.*

with a total resource utilisation of 10 LUTs and 12 flip-flops.

5.2.2. Internal Routing

The second part of the CIU consists of the selection of input and output thresholder. Eight sources of input spikes can be selected from: two of these are local connections from the other CIU units within the CIB block (consisting of four CIUs), four are global connections from other CIB blocks, one is a global monitor input and the final option is a constant ‘0’ to suppress the input. These sources can be selected for both excitatory and inhibitory inputs. Extra support for wide multiplexors are supported by the Virtex-6 slice through the addition of multiplexors between the A/B and C/D LUTs (as seen in [103]), meaning that only 2 LUTs are required per input mux (so four in total). This layout and implementation using the MUXF7A/B resources are shown in Figure 5.7.

There is also a multiplexor on the output impulse. This selects which impulse is forwarded to the global impulse routing, allowing the CIU to be bypassed to assist global routing; the impulse from each CIU is also routing locally to the other CIUs within the CIB separate to this path through the multiplexor. This is implemented as simple 4-to-1 multiplexor, requiring one LUT.

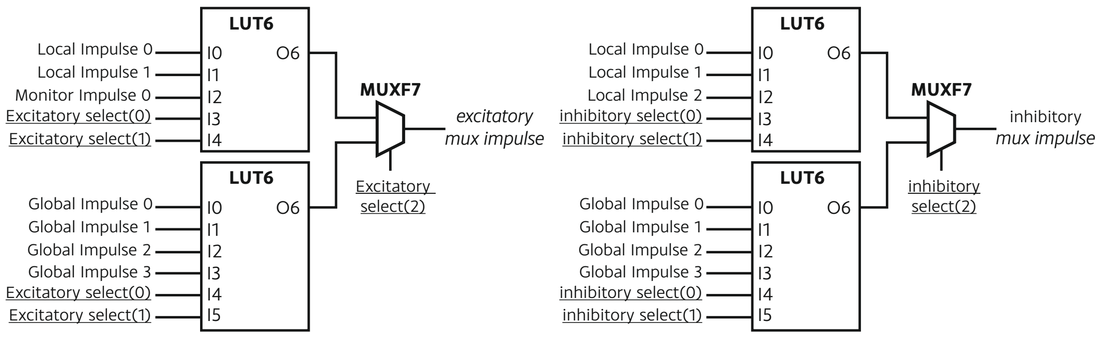


Figure 5.7: The excitatory and inhibitory input multiplexors. To achieve the most efficient LUT resource usage the wide multiplexor resources of the slice are used; namely the MUXF7 primitive. The global impulse signals are the same for both excitatory and inhibitory inputs. The excitatory input only has two local inputs and a dedicated monitor input, whilst the inhibitory input has three local inputs and no monitor input. When the multiplexor address is set to “000” the select muxes will output ‘0’ to disable all the impulse inputs.

5.2.3. Level Output Option

A feature of natural low level decision units such as nervous systems [108] is that one part of circuitry can disable the response of another circuit. This allows organisms to “switch modes” through a single pathway. Indeed this is present on some of the knobs in our system: the clock enable for example does not work by impulse and must alternate between two steady states (on or off).

To allow for this type of thresholder in the CIA every fourth CIU in the CIB has a slightly different structure to the other three. As detailed in Figure 5.8 this consists of the same input and output muxes but without the shift-register prescaler attached to the input to the counter. The extra LUT resources freed up is used to add overflow protection to the counter and to introduce another configuration mode for the CIU. Instead of outputting an impulse when the threshold is exceeded and resetting the counter back to zero, the comparator instead can test for values greater than the threshold and will output a ‘1’ whenever this is the case. This constant signal is intended to be used as a signal that will keep other units inhibited until the inhibitory input to the *level CIU* drops the counter down below the threshold and the constant output is then replaced by a ‘0’. This functionality is present in every fourth CIU within a CIB and this CIU can also still be configured in impulse mode, just without the prescaler shift register option available.

5.2.4. Placement and Configuration

The 6-bit thresholder and dataflow routing described in this section requires 16 Virtex-6 LUTs for implementation. With careful attention to where each of these LUTs are placed it is possible to implement this within the optimal 4 slices (each slice contains four LUTs). To ease the pressure on the tools (this highly-packed logic is implemented 48 times within each node and so 6,144 times across the NoC), as well as the logic being hand mapped to individual LUTs, the LUTs are also assigned to slices using RLOC constraints to place the slices within a CIU. These are placed in horizontal strips for later tiling by the CIB implementation, as seen in Figure 5.9.

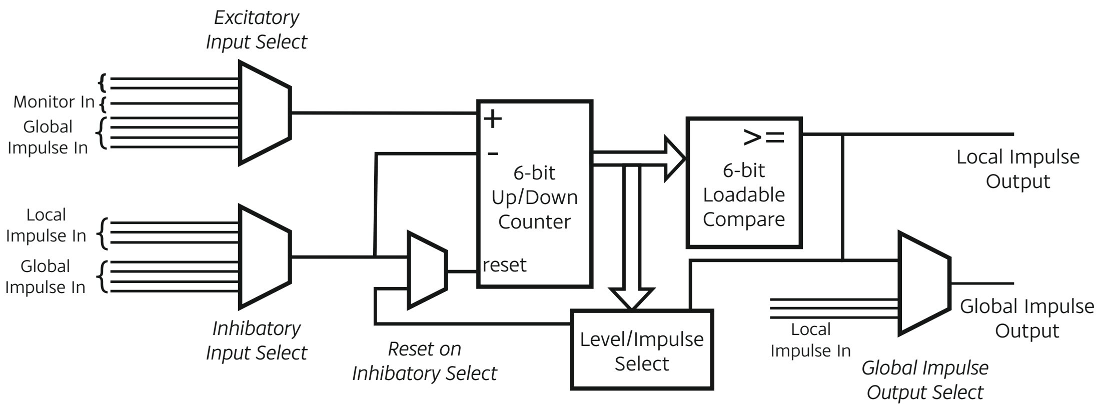


Figure 5.8: Every fourth CIU in a CIB has the option to be used in "Level output" mode. This mode emits a constant '0' or '1', instead of an impulse, depending on if the value of the counter is less than or greater than the threshold value. This allows other parts of the intelligence to be enabled or disabled through the decision made at the level output unit. The CIU can still support standard impulse mode, albeit without the prescaler shift register attached. The CIU is still contained with four Virtex-6 slices.

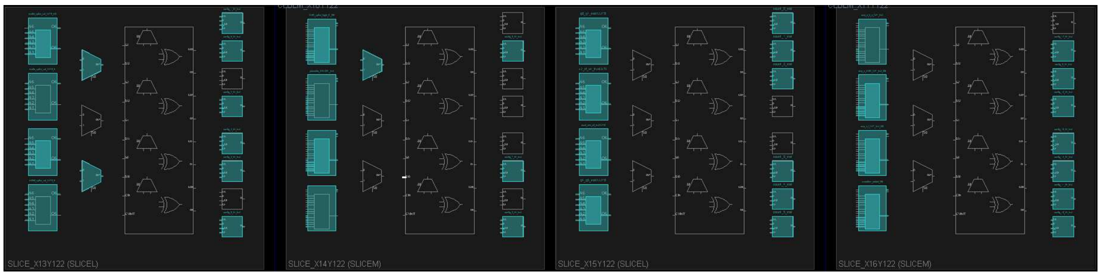


Figure 5.9: A screen-shot from the *PlanAhead* software showing the implementation of the CIU RLOC'd to four adjacent slices. In the first slice the four LUTs and two FMUX7s of the input multiplexors can be seen. The second slice contains the shift register logic (including another FMUX7) and counter reset logic, the third slice the 6-bit counter and the final slice the comparison and global output select logic. Configuration flip-flops are seen towards the right of each slice, aside from slice 3 where the flip-flops are used for storing the counter value.

The final aspect of the CIU to consider is the configuration of the programmable elements. These are stored in spare flip-flops in the four slices, chained together into a configuration chain and their outputs feed into the various configuration options around the CIU. By manipulating their clock enable inputs, the outputs can be fixed once an experimental configuration is completed. However, this adds another constraint due to the control-set limitations of Virtex-6 (only one set of flip-flop control signals can be present per slice) and so the counter registers, which use the global intelligence clock enable, are packed into one slice and the other 24 flip-flops present in the CIU are used for the configuration chain. Table 5.1 shows the bitstring of the configuration chain for a single CIU, with the configuration input for loading the prescalar shift-register attached to the end of the chain.

Bit	**Use**
0	Excite Select (0)
1	Excite Select (2)
2	Excite Select (1)
3	Inhibit Select (0)
4	Inhibit Select (2)
5	Inhibit Select (1)
6	Reset on Inhibit Impulse
7	Output Impulse Select (0)
8	Threshold Value (3)
9	Output Impulse Select (1)
10	Threshold Value (0)
11	Threshold Value (1)
12	Threshold Value (2)
13	Threshold Value (4)
14	Threshold Value (5)
15	Shift Register Excite/Inhibit Select
16 $\rightarrow$ 47	Shift Register Load Pattern

Table 5.1: Bit mapping of the CIU configuration bitstring. The non-contiguous nature of some of the select vectors is due to limitations on the flip-flop routing: some LUT outputs are used to forward the configuration chain and so forces certain configuration settings to lie at certain points in the configuration chain. When in configuration mode the prescaler shift register is connected to the end of the configuration chain to allow its divider pattern to be uploaded via the configuration chain loading process.

5.3. Configurable Intelligence Block

Whilst the fundamental decision making process is done by a single CIU, it is the combination of many such units that gives a more powerful decision making intelligence. Connections between many (arbitrary) CIUs are required to facilitate this and the CIB provides this by interfacing four local CIUs with global routing connections, as seen in Figure 5.10. The hardware resources available for the routing are restricted to 4 slices to allow maximum tessellation with the four CIUs; thus the global routing resources are a tradeoff between routing options and the resources required.

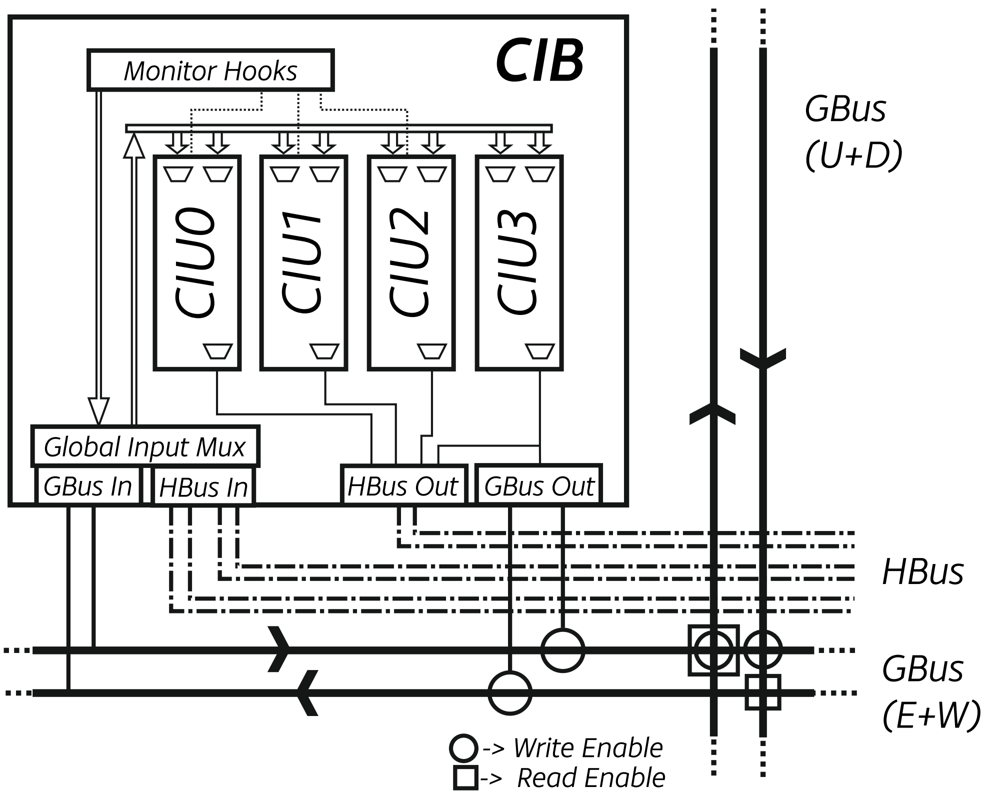


Figure 5.10: The *Configurable Intelligence Block (CIB)*. Each CIB contains four CIUs and global routing resources. There are three sources of global routing resource: 1. the monitor hooks: global inputs from the CIA monitors, 2. the HBus: each CIB outputs two signals that are shared amongst the other CIBs in a row and 3. the GBus: one signal in each cardinal direction allows Manhattan-style routing of longer-distance connections.

5.3.1. Inter-CIB Routing

The role of the global routing is to allow impulses to flow between CIUs that are not in the same block. Its design is based on the assumption that a CIB is used to take several impulses in and the local connections are used to transform these into decisions, using multiple local CIUs if required, then outputting a single impulse. Clusters of these will form the decision making blocks for other knobs in the system, relying on very little (one or two impulses) inter-CIB communication. Therefore configurability (routing choices) was chosen as the focus and with a direct hook from monitor inputs to reduce global routing resources dedicated to getting monitor inputs into the CIBs.

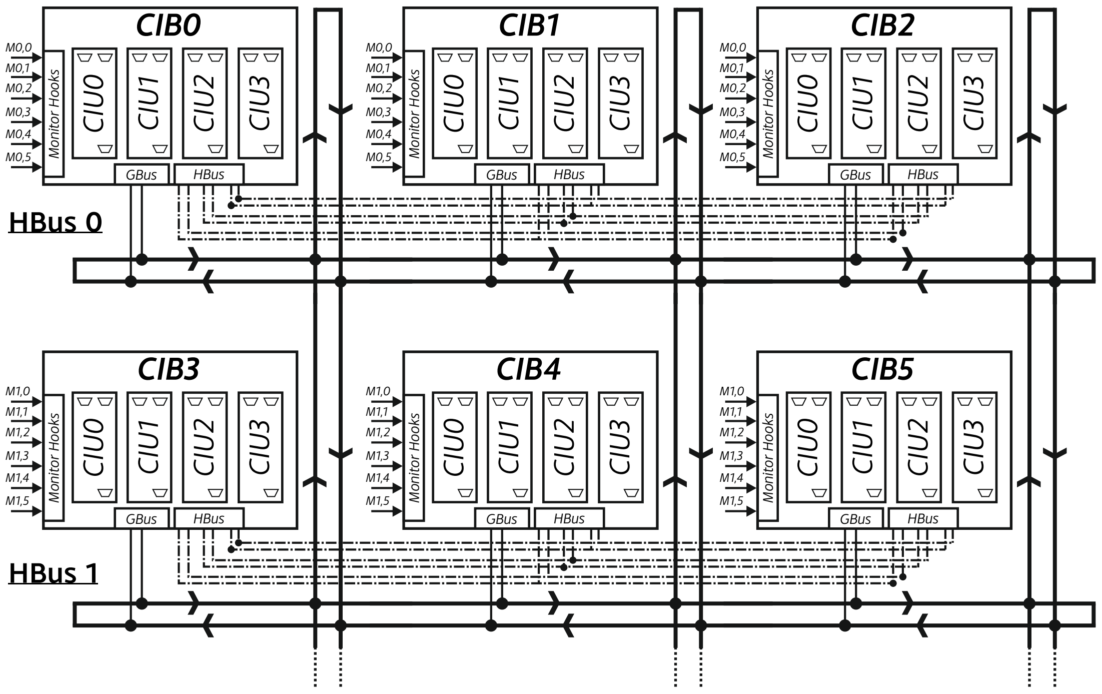


Figure 5.11: A scaled up implementation of six CIBs showing the global routing structures in use. Black dots signify connection points where a bus can be driven.

Figure 5.11 shows the global routing resources available. There are three global routing resources: *Horizontal bus (HBus):* connections between CIBs connected on the same horizontal plane, *Global bus (GBus):* a single horizontal and vertical bus in each direction connecting CIBs vertically and *Monitor hooks:* direct connections between monitors and the global input multiplexors in each CIB.

HBus

The HBus interconnects CIBs horizontally by providing each CIB with two fixed dedicated lines that are connected to each CIB in the horizontal plane. The CIA used in this thesis is three CIBs wide and so there are six HBus wires at each row in the CIA resulting in each CIB having two HBus outputs and four HBus inputs. The output of the six HBus lines are provided to the knob-mapping multiplexors to allow outputs from the CIBs to control knobs within the router.

GBus

The GBus is used for communicating between the horizontal layers and supports longer routing of signals. There are three vertical interconnects (each consisting of a separate “Up” and “Down” bus) and four horizontal interconnects (each consisting of an “East” and “West” bus). These are placed as a grid overlaying the locations of the CIBs as shown in Figure 5.11 but, unlike the HBuses, each CIB does not have a fixed dedicated connection. Instead the CIB has to be programmed to output to the bus. At this point the bus is split, allowing multiple CIBs to reuse a single bus. For example if a CIB is configured to output onto a “East” bus then all CIBs east of the configured CIB receive the signal outputted by the configured CIB. CIBs west of this CIB however are not aware of this connection and so can reuse the bus up until the output port of the configured CIB. This is why the GBus elements are directional.

The CIBs always output onto horizontal buses and at the intersections of the horizontal and vertical buses the buses can be configured to output onto each other. There are limited options here however to save resources: Down buses can output onto East buses, Up buses can output onto West buses, East buses can output onto both Up and Down buses and finally West buses cannot drive any of the other GBuses. This limitation was required to fit the CIB routing muxes and configuration bits into the four slices available.

The pairs of directional buses are connected together at the edges of the CIA to provide a useful opportunity for signals to move from one bus to the other, however care must be taken when configuring as this can lead to combinatorial loops within the routing logic.

Monitor Hooks

The final route of global signal propagation reduces the amount of global routing logic required to get monitor signals, the main source of excitatory/inhibitory impulse inputs, into the CIUs. Each CIB has two global monitor multiplexors that can each select from three monitor inputs. The three monitors that can be selected from vary between the two multiplexors and also across the different CIBs in the CIA. This provides a high variation of possible monitor input options across the CIA without using the global routing to setup paths from the input of the CIA to the relevant CIU. This is complementary to the monitor hooks that exist in each CIU (one CIU input is connected directly to a monitor. The monitor connected, once again, varies across the CIB and CIU).

5.3.2. Intra-CIB Routing

The CIB is responsible for managing the connections between the global routing resources and the individual CIUs. As seen in Section 5.2, there are four global inputs for use by the CIUs. The multiplexors in the CIB select a subset of the global routing signals to be routed to these four inputs. To reduce hardware resources required, the multiplexors take their input from different locations:

Global 0: HBus[1,0], HBus[1,1], HBus[2,0], HBus[2,1], GBus[E], GBus[W], M[A], M[B]

Global 1: HBus[1,0], HBus[1,1], HBus[2,0], HBus[2,1], GBus[E], GBus[W], M[A], M[B]

Global 2: HBus[1,0], HBus[1,1], HBus[2,0], HBus[2,1]

Global 3: GBus[E], GBus[W], M[A], M[B]

where M is the monitor hook interconnect and has the following choices for an monitor index offset of 0:

M[A]: *monitor hook[0]*, *monitor hook[1]*, *monitor hook[2]*

M[B]: *monitor hook[3]*, *monitor hook[4]*, *monitor hook[5]*

These four global inputs can then be selected by the four CIUs within the CIB. Global 0 and Global 1 both require 8-to-1 multiplexors and so require two LUTs each and the MUXF7 primitive to multiplex between the LUTs. Global 2 and Global 3 are only 4-to-1 multiplexors and so only require a single LUT each. Likewise the monitor hook multiplexors only require a single LUT each due to their 3-to-1 nature. This gives a total of eight LUTs required for global input routing within a CIB, as can seen in Figure 5.12.

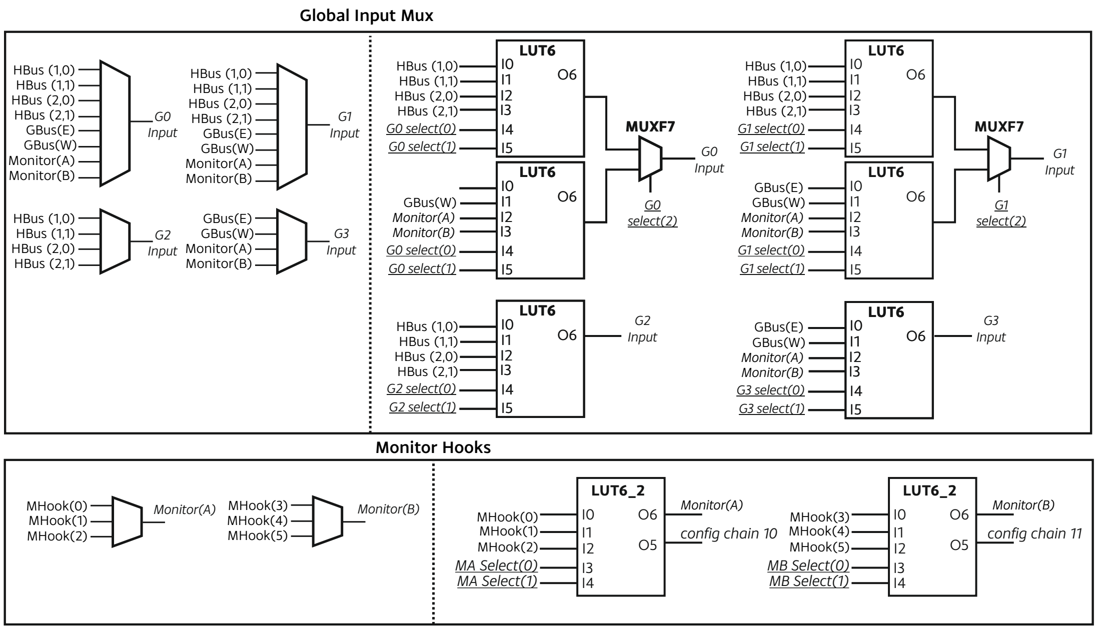


Figure 5.12: Design of the CIB input multiplexors and the resultant LUT efficient implementation. Once again MUXF7 resources are used for the two wide multiplexors.

The CIB also handles routing of data from the CIUs to the global routing logic. There are two HBus outputs that the CIB can drive and a 4-to-1 multiplexor is used for each. The first can select the impulse output from the “Global out” port of CIU0, CIU1, CIU2 and a ‘0’ logic level (to disable the output completely) and the second can select from CIU1, CIU2, CIU3 and logic ‘0’. This gives complete coverage of the CIUs onto the two dedicated HBus signals whilst only requiring two LUTs.

The CIB can also output onto the GBus. This however is limited to only the output of CIU3. The CIB can be configured to either allow the output of the East bus to continue to propagate down the bus or to output CIU3’s impulse onto the bus at this point. The same applies for the West bus and requires one 2-to-1 multiplexor each. There is also a setting that enables the GBus' vertical Up and Down buses to output onto the East and West horizontal buses. Finally the corresponding setting is also supported, but only the E bus can output onto the Up and Down buses. Six 2-to-1 multiplexors are required for this functionality requiring three LUTs. An overview of this implementation architecture for the output routing is shown in Figure 5.13.

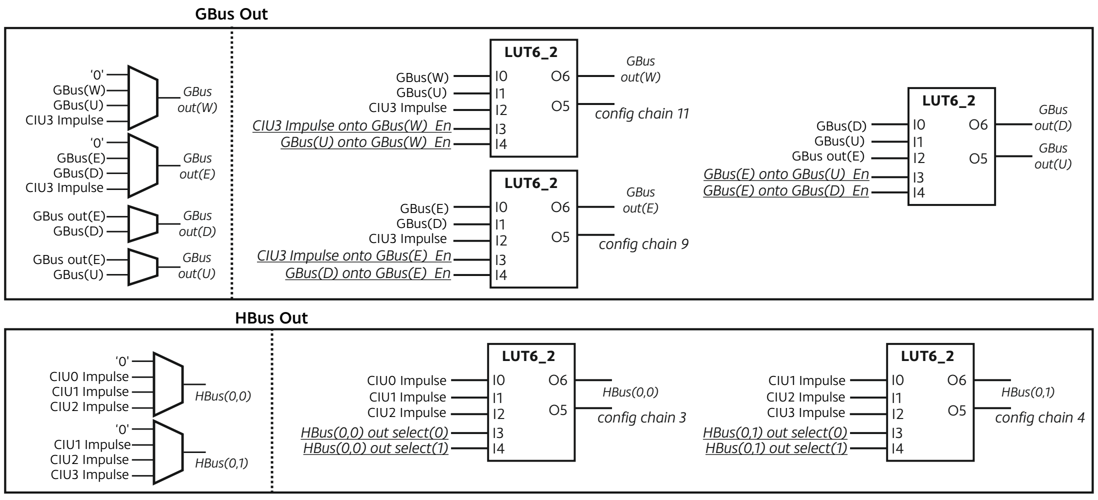


Figure 5.13: Design of the CIB output multiplexors and the resultant LUT efficient implementation. Included in this set of multiplexors is the logic to either propagate or drive the GBuses in all cardinal directions.

This gives an overall total of 13 LUTs required for the CIB routing. An extra LUT is used to decode the configuration chain data to enable loading of the CIB (and its contained CIUs) by driving the `Config_CE` signal when the configuration ID matches the hardware ID of the CIB, this ID is generated depending on the location of the CIB within the CIA. This results in a total resource requirement of 14 LUTs and so fitting our implementation constraint of using a maximum of four slices (each slice contains four LUTS).

5.3.3. Placement and Configuration

As the router can be packed into four slices, it is able to be tiled with the four CIUs into a single 20-slice component. Once again RLOC constraints allow tight manual packing and relative placement of CIBs, this also eases the tool's job of finding CIU-CIU and CIU-CIB routing connections as all units are local to each other and so long-distance routing resources are not required. The four slices of the placed router can be seen in Figure 5.14, whilst a placed CIB is shown in Figure 5.15; the tiling can clearly be seen with each CIU and the CIB placed horizontally (to ensure each CIU can use a SLICEM resource, which are striped vertically across the device) and stacked vertically. By applying relevant *HU_set* attributes [110] to the CIUs and CIB routing block within the CIB, the RLOC placer can then place the entire CIB using a single set of co-ordinates when tiled up to the full CIA.

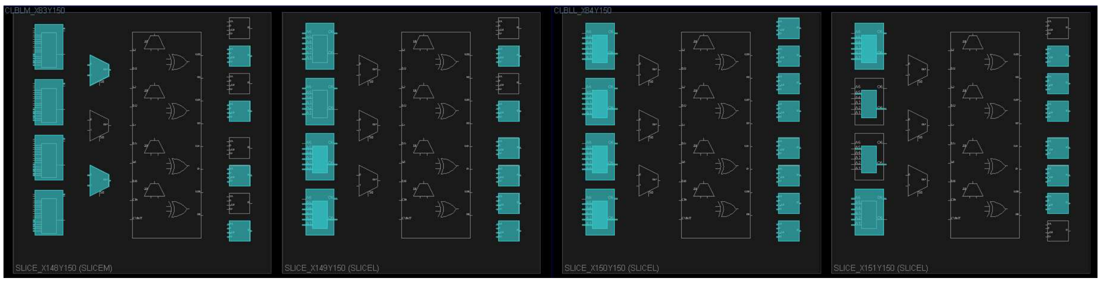


Figure 5.14: A *PlanAhead* screenshot showing the implementation of the CIB router within the four slices required for tiling. Once again the use of two FMUX7s of the wide input multiplexors can be seen. Also of note in the final slice is that two LUT5s are available for future use, indeed the CIB is constrained by the configuration chain flip-flops available (the unused flip-flops seen in this figure are not accessible due to LUT input signal congestion using the “nX” input).

All of the configuration settings for the CIB concern the setup of the routing resources. Table 5.2 details these settings within the bitstring. The CIA’s configuration chain is broken up into a granularity of the information need to configure one CIB (i.e. the smallest unit that can be configured at a time is a CIB). This allows changes to be made to the setup of a CIB without having to re-configure the entire CIA and also means a long, unwieldy, bug-prone bitstring doesn’t need to be manipulated! This has two implications: 1. the configuration data for the four CIUs within a CIB is appended to the CIB bitstring and 2. the CIB contains decode logic that only enables the clock enables on configuration flip-flops when the “configuration ID” matches the CIB’s pre-programmed ID. This is contained within a LUT in the routing slices.

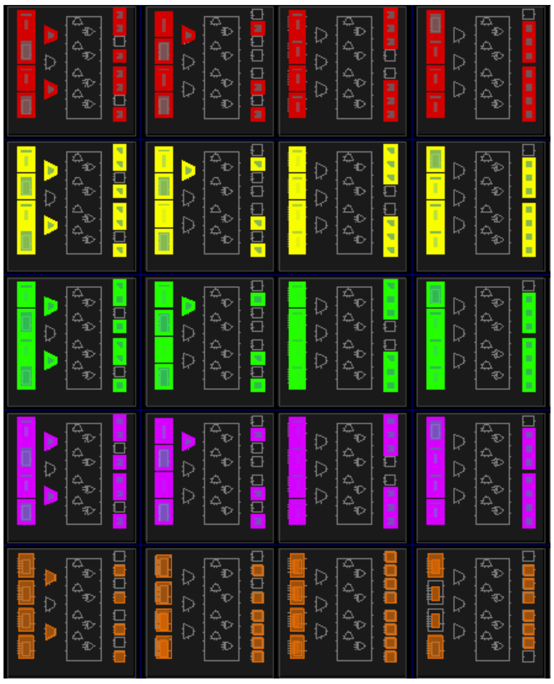


Figure 5.15: A *PlanAhead* screenshot showing the implementation floorplan of the 20-slice CIB tile. RLOC constraints allow a regular placement of the CIUs with the routing slices: Red, Yellow, Green, Purple primitives are for CIU0, CIU1, CIU2 and CIU 3 respectively. The orange primitives are the CIB routing primitives.

Bit	Use
0	Global Input 0 Select (2)
1	Global Input 0 Select (0)
2	Global Input 1 Select (2)
3	Output HBus 0 Select (0)
4	Output HBus 1 Select (0)
5	Global Input 1 Select (1)
6	Global Input 1 Select (0)
7	Global Input 0 Select (1)
8	Output HBus(0,0) Select (1)
9	Output CIU3 onto GBus(E) Select
10	Monitor A Select (0)
11	Output CIU3 onto GBus(W) Select
12	Monitor B Select (0)
13	Monitor A Select (1)
14	Monitor B Select (1)
15	Output HBus(0,1) Select (1)
16	Global Input 2 Select (0)
17	Global Input 2 Select (1)
18	Global Input 3 Select (0)
19	Global Input 3 Select (1)
20	GBus(D) output to GBus(E) Enable
21	GBus(U) output to GBus(W) Enable
22	GBus(E) output to GBus(U) Enable
23	GBus(E) output to GBus(D) Enable
$24 \rightarrow 71$	CIU0 Bitstring
$72 \rightarrow 119$	CIU1 Bitstring
$120 \rightarrow 167$	CIU2 Bitstring
$168 \rightarrow 215$	CIU3 Bitstring

Table 5.2: Bit mapping of the CIB configuration bitstring. As with the CIU, some LUT outputs are used to forward the configuration chain and so some of the bits of the select vectors are non-contiguous. The CIUs are connected to the end of the chain with the output of the shift register in each CIU connected to the configuration chain input of the next CIU to build the chain.

5.4. Configurable Intelligence Array

The final requirement for the embedded intelligence is scaling up of the CIBs with the global routing resources and the translation and integration of routing sensory signals *(monitors)* and actuator signals *(knobs)*. For the experiments in this thesis a 3x4 array of CIBs was used with 12 monitors and five knobs.

5.4.1. Array Layout

Figure 5.16 shows the scaled-up 3x4 array of CIB units. All CIB in a row are connected via the HBus and the East and West global buses. Rows of CIBs are connected via the vertical Up and Down GBuses. A total of 12 CIBs are instantiated requiring 240 slices. Ideally these would be tiled again but it proved too difficult for the tools to implement the processing elements within their dedicated floorplan with the tiling of the entire CIA enabled. Instead the relative placement within a CIB (the CIUs and routing slices i.e. the tile of Figure 5.15) is maintained and the tool can then place these tiles where it wishes within the processing element floorplan partition.

As also seen in Figure 5.16, the CIA has the role of providing the input and output interface with the Centurion router. The monitor hooks have been seen in the previous section and at this level they are connected to input signals via a large crossbar. Each row of CIBs has six monitor hook inputs: M(r,0) to M(r,5) where r is the row number. As there are four rows of CIBs, the crossbar has four 6-bit outputs. There are currently 12 monitor inputs translated into a impulse-based form and so the crossbar simply selects one of two possible outputs for each output signal; giving each row of CIBs the possibility of having each monitor as an input and using the internal routing logic to route the required monitor signals from the row's crossbar output to the required CIU.

A crossbar is also used for connecting outputs from the CIA to the router knobs. To save on global routing resources, the CIA output is captured directly from the HBuses. The six bits of each HBus is connected into the crossbar and the signals for the five knobs that are currently included are routed from outputs from all of the HBuses. These are then forwarded to the knob translation modules for integration within the

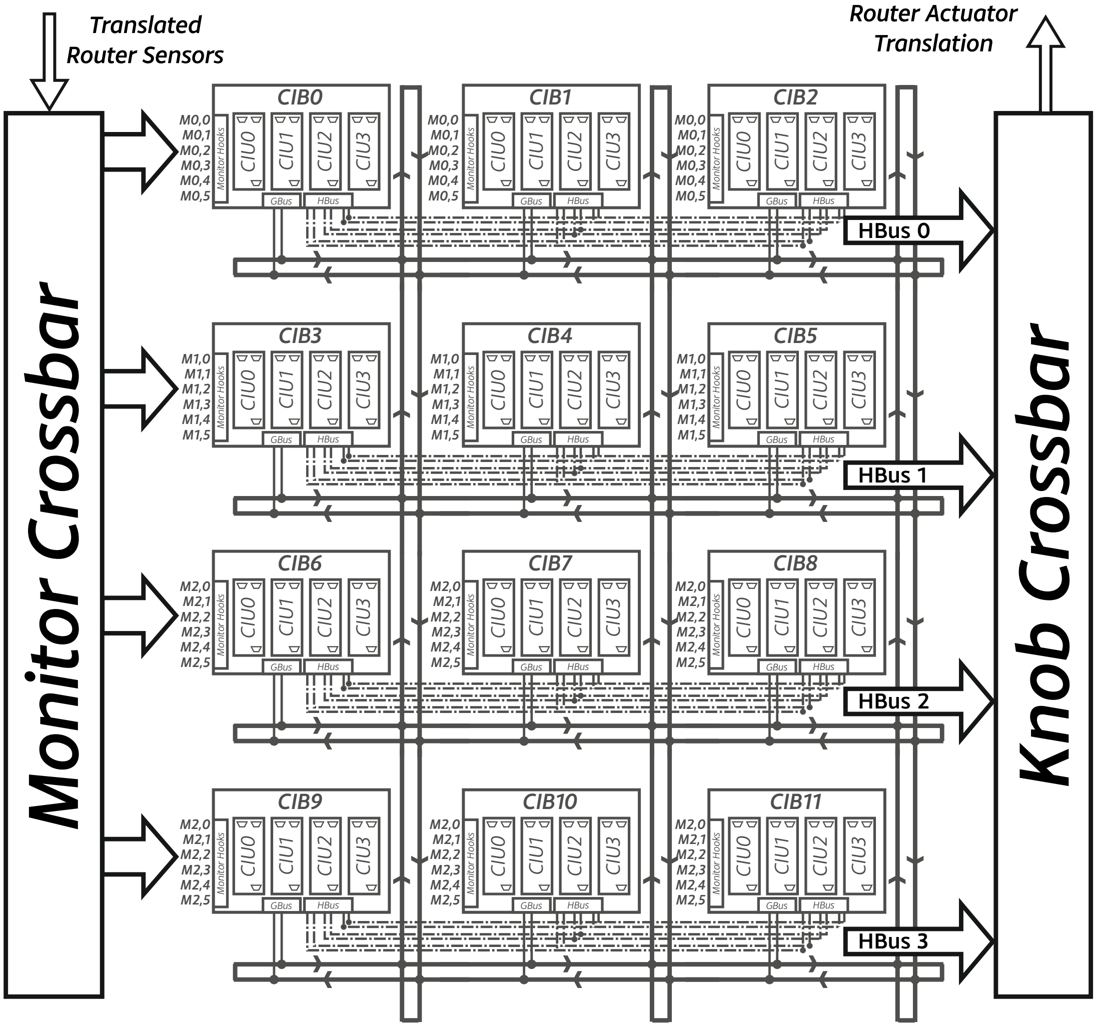


Figure 5.16: The *Configurable Intelligence Array (CIA)* used for the experiments presented in this thesis. This is a 4x3 array of CIBs and so consists of 48 thresholders. The diagram shows how the monitors and knobs are passed into and out of the array using the large configurable crossbars. There are six unique monitor inputs per row of CIBs (the same six monitors go to all CIBs within a row) and the knobs are selected from the outputs of the HBuses.

Of course this functionality could be implemented within the Picoblaze or even a dedicated neuromorphic processor. However the role of the CIA is to find, optimise threshold values of and experiment with relevant pathways to produce the autonomous behaviours desired. Once these pathways are found then the routing elements of the CIA, CIB and CIU can be removed and a highly efficient (only a counter and comparator per thresholder) set of autonomous intelligence pathways will remain.

router.

5.4.2. CIA Hardware Resource Requirements and Scalability

The CIA design is fully parametrisable with some design restrictions that reflect the implementation specific to the needs of the experiments in this thesis. These restrictions are found in the inter-CIB routing, with widths for the HBus fixed for this design. Redesign of the CIB router would be required to support arrays that are wider than three CIBs with a shared HBus. However, given the information reducing nature of many biological intelligence models (e.g. sensory fusion in the nervous system [108]), the need to route a large number of signals further between CIBs may not necessarily be required for more complex intelligence pathways; an expansion of the GBus's may be more beneficial. Larger areas would also require registers in the inter-CIB routing signals if high processing rates are required; the current critical path for 25MHz operation comes from the length of the GBus paths.

Knobs and monitors have not been discussed yet, for this implementation a large cross bar is used for both the monitor inputs and the knob outputs. Crossbars are not inherently scalable and so in a very large array it is likely that the attachment of knobs and monitors will need to be to a subset of nodes. This will affect how general purpose the CIA is, but will reflect the architectures of natural decision pathways where sensory inputs are connected to specific neural structures such as ganglia [108].

Another aspect of CIA scaling will be improving the local connectivity by expanding the number of CIUs. The CIU design is heavily tied to the layout of the Virtex-6 LUT and this leads to the maximum number of four CIUs per CIB. Increasing the local connectivity may allow more complex decision to be made without requiring inter-CIB or global routing. However it is likely that this tradeoff will require some understanding of the structure of the desired intelligence pathways to be known, maybe relying on other design methods to approximate the intelligence pathway layouts as introduced in Section 5.1.2.

An overview of the hardware resource requirements for each aspect of the CIB structure is given in Table 5.3. As the CIU and CIBs are already directly mapped to FPGA primitives, scaling up these elements will simply require multiplications of the number of CIBs desired. The table also details the amount of hardware resources dedicated for configuring the intelligence. As mentioned in Section 5.1.2 once a design has been prototyped for a system with the routing structure, connectivity settings and threshold parameters set and tested, the intelligence pathways can be extracted with the configurable aspects removed. This results in a large hardware resource saving as the configurable aspects of a CIB require 60% of the CIB's total FPGA primitives. As the current design of the CIA does not have any capability for self-learning, no part of the configurable intelligence parameters can be updated once implemented and so removing the configurable parts of the CIA will not have an effect on the underlying flow of impulses through the configured (now fixed) datapaths and the dynamics of the decisions made by the CIA will not change.

Component	**Total LUTs**	**Total FF**	**Total Slice**	**Config. LUTS**	**Config. FF**	**Config. Slice ***	**Config. Overhead**
CIU (A-C)	16	23	4	7	16	2	50%
CIU (D)	16	22	4	8	15	2	50%
CIB router	16	24	4	16	24	4	100%
Total for CIB	80	115	20	24	87	12	60%
Total for CIA	960	1380	240	540	1044	144	60%

Table 5.3: Hardware Resources required by CIU, CIB and the CIA. The configuration overhead is resources dedicated to impulse routing between CIUs and between CIBs, resources for setting the behaviour of the CIU (e.g. reset on impulse, level or impulse mode) and for storing the threshold. In a manufactured implementation, these resources can be removed as the settings and routing architecture of the CIA can be fixed. This offers significant savings of hardware resources. **slices estimated as savings could be more or less depending on LUT/FF packing of non-configurable design. CIA has not been implemented with the configuration resources removed.*

All of the above factors have considered enhancing the CIA functionality by expanding the size or improving the CIU or inter-CIB routing. The data path could also be expanded. The 6-bit tri-counter implemented is the most resource efficient counter for the Virtex-6 LUT (as the DSP blocks are unavailable due to their layout on the device) and so this determined the thresholder size. In a less resource constrained implementation, increasing the size of this counter will not bring any immediate benefit aside from more efficient CIU use as in the current implementation the shift register can be used as a pre-scaler or CIUs can be chained together to form wider thresholds. The inclusion of the level-output CIU adds benefits for designing intelligence pathways that require parts of the decision pathway to be enabled/disabled based on another decision. Therefore improvements on the datapath should consider improving the capability of the thresholder. Potential expansions include thresholders whose threshold value could be changed by other CIUs; this would enable online self-learning. Another improvement would be to add a controllable source of randomness to the decision making element as random elements are a large aspect of many artificial design making elements; for example the Network Interaction model assumes random walks throughout the nest by colony members [23]. Improved computation within the thresholder would also allow the CIA to move away from an impulse based approach. However, this would start to lose the power and low resource overhead (due to increased wires between CIUs and CIBs) benefits that impulse-based decision making units provide.

5.4.3. Monitors

The sensory signals embedded within the router may require translation to be compatible with the intelligence array. Some of these signals are vectors that need to be converted into impulse trains, some are slow signals originating from the node that need converting into an impulse form. Whilst the intelligence blocks could be used for this translation, it would unnecessarily take valuable resources away from the bio-inspired intelligence models as there are circuits that are far more efficient at achieving this. A toolkit of monitor translation blocks was created for this.

With many of these signals being determined by the performance and nature of the application running on the many-core, there is a possibility that a dependency exists between the characteristics of these signals and the performance of the decision making units. Therefore care must be taken when using time-domain based signals, such as time-dependent inhibitory or excitatory inputs to the CIUs. As time plays a vital role in many NoC performance goals (application throughput, QoS, packet latency) the relationship between periodic impulses and router driven signals must be understood for each application.

For example, a QoS decision pathway may trade off clock frequency of a node against the number of packets moving through the router. As the number of packets seen increases, the clock frequency of a node will be increased as the implication is that the local area of the network is busy and so there are more tasks to undertake. A time reference as the inhibitory input would allow the clock speed to decrease if no packets are seen, the relationship of the decision would be the ratio of packets seen (excitatory) to the number of periodic impulses (inhibitory). If this period is set too short then the system will seem to be too eager to reduce the clock frequency and will result in poor QoS as the clock frequency of the node will be kept too low. On the other hand, if this period is set too long then the system will be quick to raise the clock frequency and will take a long decay period to reduce it back down, leading to a waste in power by leaving the processor at a high speed for too long.

Managing time-based periodic impulses can be done by using a CIU to divide down the input periodic impulses, using the threshold to either set the division or allowing an optimiser to find a good value for the threshold value. An alternative is to set the impulse period of the periodic input at the point where it enters the CIA, this would effect all of the decision pathways that use this period reference as an input. Both of these approaches to tuning the period would require some system analysis when defining monitors to the embedded intelligence, and also when defining knobs as the translation from CIA impulse train to router actuator may require a sampling period or other time-based translation. Expected rates of input monitors can be defined by looking at the application design and decision pathways that rely on any periodic impulses need to be inspected and updated to ensure that the period is suitable for the expected frequency of occurrence of the other input monitors in the decision pathway.

Table 5.4 lists the monitors available and their translation module used for the experiments carried out as part of this thesis. There are four knob outputs as listed in Table 5.5.

The monitor modules provided in the CIA knob and monitors toolbox are as follows:

Vector to Impulse Train

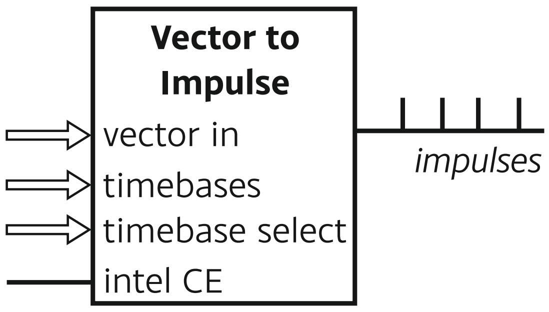


Figure 5.17: The Vector to Impulse monitor translator. An input vector is sampled and an impulse train is generated, its frequency dependant on the value of the vector.

This module converts a vector into an impulse train with the value of the vector determining the frequency of the impulse chain, shown in Figure 5.17. This is done by using one of the four system timebases (originating from the experiment controller's RTC, as introduced in section 4.7) to increment a counter. The used timebase is selectable by the configuration. Once the counter reaches the value of the input vector then the counter is reset and an impulse generated. By continuing in this fashion it can be seen that the rate that impulses are generated will be dependant on the vector value. At this stage the output impulse is also synchronised with the intelligence clock enable (for when the intelligence is run at a slower frequency than the clock) to ensure that the impulse is not missed due to the clock enable being low.

Match to Impulse

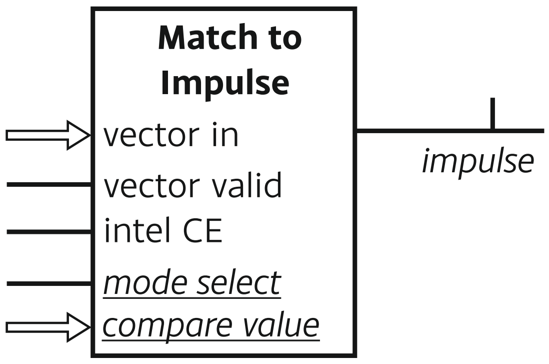


Figure 5.18: The Match to Impulse monitor translator. An input vector is sampled and if it matches the programmable compare vector then an impulse is generated.

Another monitor conversion module working with vectors, this module compares an input vector to a programmable value and issues an impulse if the input value matches the internal value. Once again the output impulse is synchronised to the intelligence clock enable to ensure that an impulse is not missed. A data valid input means that only valid data is compared against and is used for one of the operating modes of this module. These operating modes are *a.)* match unit generates an impulse when the input vector matches and data valid = '1' (possibly leading to multiple impulses per data valid) *b.)* match unit only generates an impulse once per match and data valid = '1', requiring data valid = '0' before the next impulse can be issued. This is shown in Figure 5.18.

Monitor	**Translator**	**Description**
Packet "A"	Comparison to Impulse	*Detects when a packet with programmable header "A" has been routed by the router*
Packet "B"	Comparison to Impulse	*Detects when a packet with programmable header "B" has been routed by the router*
Packet "C"	Comparison to Impulse	*Detects when a packet with programmable header "C" has been routed by the router*
Packet "D"	Comparison to Impulse	*Detects when a packet with programmable header "D" has been routed by the router*
Packet ID "A"	Comparison to Impulse	*Detects when a packet with programmable ID of "A" has been routed by the router*
QoS	Vector to Impulse Train	*A quality of service metric output from the node*
Node Clock Frequency	Vector to Impulse Train	*The current value of the node clock divider*
Node Temperature	Vector to Impulse Train	*The current value of the node's ring oscillator counter*
Node Enable	Raw Signal	*The current value of the node's clock enable signal*
Node Faulty	Raw Signal	*A signal output by the node that can be used as a watchdog or fault detection flag*
Knob Task Impulse	Raw Signal	*Output by the* Impulse Array to Vector *knob when new impulse is detected*

Table 5.4: List of CIA monitors and the translation units used to translate the input from the router/node into a form that is suitable for the impulse-based intelligence.

5.4.4. Knobs

The inverse transformation is required for the outputs from the CIA, individual and trains of impulses need to be converted into signals usable by the router and node. Once again the functionality can be done by the intelligence (conversion of many impulses into a frequency for conversion to a vector for example) but due to the focus on low resource usage, the following dedicated translation circuits are used:

Impulse Train to Vector

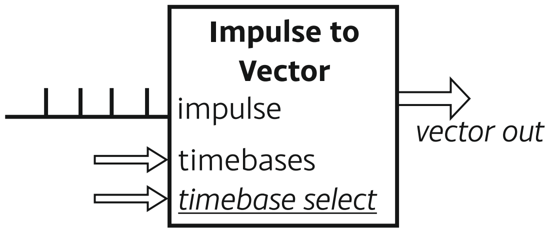


Figure 5.19: The Impulse to Vector knob translator. An impulse train is sampled and a vector is output, its value dependant on the frequency of the incoming impulse train.

This module converts a impulse train to an output vector and is shown in Figure 5.19. The impulse train drives an internal counter which is then sampled and reset once the selected timebase emits a ‘1’. The counter will then start counting again and a constant impulse frequency will result in the counter reaching the same value before being sampled and reset, a lower frequency will result in the count being lower at the sample point and vice versa. The sampling is used to ensure a constant value is output whilst the counter is counting. There is no support for translating the impulses relative to an input timebase and so any required prescaling must be done with the intelligence blocks.

Impulse Array to Vector

The second output translation module takes several impulse outputs and constructs a vector of the same width as the number of inputs, the signals of which are set depending on which input impulse was the last to fire. Thus only one signal can be set to ‘1’ at a time but it signifies that this signal was the last impulse to equal logic ‘1’. When a

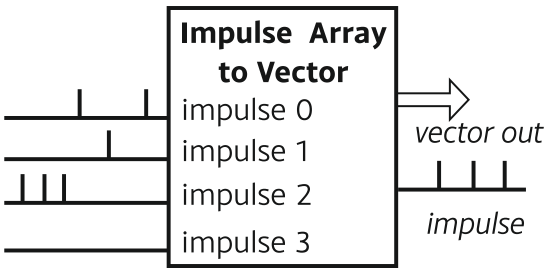


Figure 5.20: The Impulse Array to Vector knob translator. An array of impulses is sampled and when an impulse is detected on an input then the output vector is set to reflect which impulse line was last detected. An output impulse is generated when a change is detected.

new impulse arrives then the output vector is set and the module also emits an impulse that can be used as a monitor input (as seen in Table 5.4) to influence other parts of the intelligence. Figure 5.20 shows the ports of this module.

Impulse to Switch

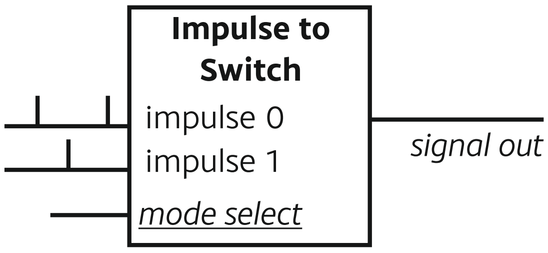


Figure 5.21: The Impulse to Switch knob translator. An impulse on input impulse 0 will cause the output to set to '1', whilst an impulse on input impulse 0 will cause the output to reset. A second toggle mode uses input impulse 0 to toggle the output on arrival of a impulse.

The final output translation is designed to drive control signals within the router and is shown in Figure 5.20. It has two modes, one that requires a single input and simply toggles the value of the output (reset to '0') when an impulse arrives on input 0. The second mode has two inputs. It uses a rising edge on the impulse 0 input to set the output to '1' and a rising edge the on impulse 1 input to clear the output back to '0'.

Knob	**Translator**	**Description**
Task Out Suggest	Impulse Array to Vector	*Suggests the task that the node should process and the router should route internally. Impulse array is four signals wide*
Clock Frequency Out	Impulse Train to Vector	*Loads the node's clock divider register with this value*
Node Clock Enable	Impulse to Switch	*Sets the value of the node's clock enable signal*
Router Bypass Internal	Impulse to Switch	*Disables the router from allowing any packets to be routed to the node (i.e. the Internal port)*

Table 5.5: List of CIA Knobs and the translation units used to integrate the knobs with the internal router/node control logic.

5.4.5. Configuration

The monitor crossbar, knob crossbar and the parameters for the translation units all need configuring and this is done through the CIA configuration chain in the same fashion as the CIBs. The CIB ID of 0 is reserved for the CIA configuration and so any bitstrings sent to this CIB address will be loaded into the CIA chain. Included in this configuration chain is also the loaded value for the intelligence clock enable clock divider. This is a 13-bit wide count-down counter that drives the CE signal for the translation units and also for the counter flip-flops within the thresholders in the CIUs. This allows the intelligence to be run at a range of divided frequencies from 25MHz (the maximum speed of the CIA circuitry) down to 12.2KHz, supporting experiments where the intelligence is run at a significantly lower speed than the router and node it is controlling.

Table 5.6 shows the bitstring mapping for the CIA settings.

Bit	**Use**
$0 \rightarrow 3$	Monitor: Match Unit 0
$4 \rightarrow 7$	Monitor: Match Unit 1
$8 \rightarrow 11$	Monitor: Match Unit 2
$12 \rightarrow 15$	Monitor: Match Unit 3
$16 \rightarrow 19$	Monitor: Match Unit 4
$20 \rightarrow 21$	Monitor: QoS Timebase Select
$22 \rightarrow 23$	Monitor: Clock Frequency Timebase Select
$24 \rightarrow 25$	Monitor: Thermal State (i.e. Ring Oscillator) Timebase Select
$26 \rightarrow 27$	Knob: Clock Frequency Out Timebase Select
28	Knob: Node Clock Enable Mode Select
29	Reserved
30	Knob: Internal Bypass Mode Select
$31 \rightarrow 36$	Monitor Crossbar Row 0 selects
$37 \rightarrow 42$	Monitor Crossbar Row 1 selects
$43 \rightarrow 48$	Monitor Crossbar Row 2 selects
$49 \rightarrow 54$	Monitor Crossbar Row 3 selects
$55 \rightarrow 58$	Knob Crossbar: Task Suggest (0) input select
$59 \rightarrow 62$	Knob Crossbar: Task Suggest (1) input select
$63 \rightarrow 66$	Knob Crossbar: Task Suggest (2) input select
$67 \rightarrow 70$	Knob Crossbar: Task Suggest (3) input select
$71 \rightarrow 74$	Knob Crossbar: Clock frequency input select
$75 \rightarrow 78$	Knob Crossbar: Node Clock Enable impulse 0 select
$79 \rightarrow 82$	Knob Crossbar: Node Clock Enable impulse 1 select
$83 \rightarrow 90$	Reserved
$91 \rightarrow 94$	Knob Crossbar: Router Bypass Internal impulse 0 select
$95 \rightarrow 98$	Knob Crossbar: Router Bypass Internal impulse 1 select
$96 \rightarrow 112$	Intelligence Clock Divider Value

Table 5.6: Bit mapping of the CIA configuration bitstring. The first set of bits are used to configure the monitor translation modules. The second set of bits configures the knob translation modules. The rest of the bits are used for programming the monitor and knob crossbars, aside from the final 13 bits which determine the value of the intelligence clock enable divider.

5.5. Summary

The Configurable Intelligence Array described in this chapter is the second novel design contribution of this thesis. The embedded intelligence hardware allows quick prototyping and flexible intelligence implementations for the bio-inspired social-insect models used in the experiments presented in the next chapter. Whilst the configurable elements of the CIA strikes a balance between configurabilty and hardware resource usage, if the decision making parts of the intelligence are considered separately, i.e. the thresholder, then each CIU would only require two slices to implement. Thus fixed intelligence pathways would be extremely cheap to implement in hardware (once the CIA has been used to prototype these intelligence pathways).

The upcoming chapters will use the Centurion many-core from Chapter 4 and, in Chapter 7, combine it with circuits implemented on the CIA for the purpose of exploring the CIA implementation of the bio-inspired models.

Chapter 6

Social Insect Inspired Many-Core

6.1. Overview

This chapter presents the experiments undertaken on the Centurion platform (described in Chapter 4) for the purpose of demonstrating that the social-insect intelligence models (Chapter 3) can be translated to a form suitable for embedding within the many-core using the PicoBlaze (Section 4.5) or the CIA (Chapter 5) and evaluating that they then exhibit the expected emergent behaviours. The first experiment introduces the test cases and explores the performance without any embedded intelligence. The following experiments implement elements of the social insect models and build upon them using the PicoBlaze. The next chapter will use these models to achieve fault tolerance and also implement the models using the CIA.

6.2. Experimental Benchmark Applications

A many-core application, like all computing applications, can have an unbounded level of complexity of interaction between the sub-tasks that achieve the application's goal. Therefore a compromise is required to capture the needs of typical many-core applications. The authors in [111] provide a set of seven task graphs that they claim represent the main types of parallel program. The first three are general purpose shapes that the remaining four applications are made up of and so these three base graphs will be used in the following experiments. A fourth basic linear task graph is also used as this represents parallel applications with embarrassingly parallel sequential segments. These four test graphs are shown in Figure 6.1.

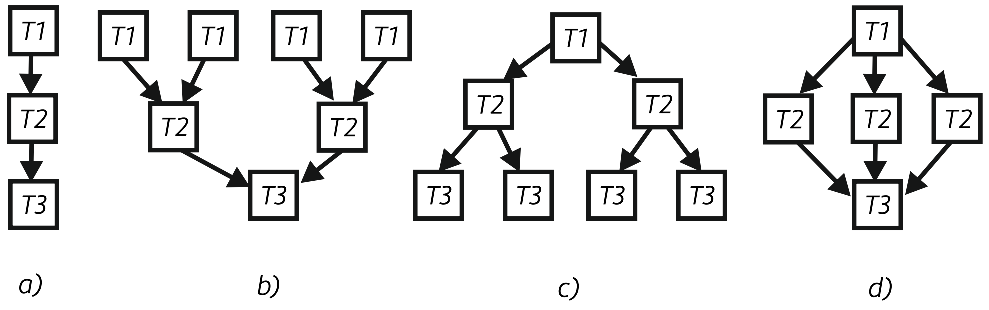


Figure 6.1: The test task graphs. *a)* linear, *b)* in-tree, *c)* out-tree, *d)* fork-join

All experiments use the same application model settings. Packets from Task 1 nodes are generated at a fixed rate of one every 4ms, whilst Task 2 and 3 nodes do not send a packet out until they have received a packet with their task as the destination. All packets are the same size and each node also has a "processing time" whereby on receipt of a packet the node cannot send or receive any new packets whilst in this processing phase (to simulate the task being performed). These and the other experiment parameters are summarised in Table 6.1.

In a real-world setting, these parameters would be provided by the application running on the many-core. In absence of a real-world application, these settings reflect what is felt to be the outcome of a typical application architecture and design analysis process. Specifically, the notation of a "producer task" (Task 1) producing packets at a fixed rate reflects a periodic input source (camera, sensor, remote system). An application analysis will ensure that the producer task does not produce more data than the consumer tasks can handle (i.e. the task graph is schedulable), however too much margin here would make the system inefficient.

A requirement for experimental use is that the experiment can be run in a short period of time to allow for parameter sweeps, another requirement is that the data produced by an experiment is not too large to allow a large number of parameter sweeps. Naturally, this will depend on the setup of the experiment system and with Centurion the limiting factor is the DDR memory in the experiment controller (256MB), the speed that the node experiment logs (2KB) can be read by the experiment controller, the speed of the experiment controller to PC link (800 MBit/s) and the rate that the PC application can absorb experiment data (unknown analytically, tested through use). After some performance testing, the critical bottleneck was determined to be the rate at which the experiment controller can read the logs of all nodes in the worst case.

An experiment run-time of 1 second was desired. This allows a large number of random starting conditions to be trialled (e.g. 1000 will take just over 15 minutes) and experiments with parameter sweeps can be undertaken in a matter of hours. Given an experiment runtime of 1 second, a maximum packet rate of 1.5ms was determined:

- The experiment log buffer at each node is 2048 bytes, each event requires 8 bytes.

- The experiment controller requires approximately 25μs to read a whole experiment log of 2048 bytes (the buffer interface is clocked at 100MHz and data is read out on every clock cycle once set-up, see Section 4.4.1).
- As there are 128 nodes, it will take 54ms to read all of the logs of all of the nodes. Therefore 18 full log sets can be fetched within one second, generating 4.5MB of data.
- Given 18 log flushes per experiment, a node can store 4,608 events per experiment.
- Depending on the experiment, sending and receiving a packet requires roughly 4 events (packet start transmit, packet end transmit, packet received, packet metadata).
- Therefore, the maximum experiment packet rate is 1,152 packets per node per second, or 147,456 total packets per experiment.
- However, this assumes a well balanced distribution of packet events. This is unlikely, with some nodes receiving more packets (and thus sending more packets if they are task 2 and 3 nodes) than they should. Hence the relaxation of the packet sending rate (driven by the producer Task 1 nodes) to 4ms.

6.2.1. Experiment Settings

The initial condition parameters for the experiments in this thesis consist of the following initial settings loaded into Centurion:

1. *Task mapping:* The assignment of tasks to the nodes of the many-core. All task graphs in these experiments use three tasks. The task mapping concerns both the total number of each tasks (dubbed the *task ratio* for the experiments in this thesis) and also the geometric assignment of tasks across the many-core nodes. In all experiments the initial geometric mapping is done randomly (i.e. the total set of tasks is distributed at random across the nodes). The ratio varies between even (i.e. 1-1-1, a third of the many-core nodes per task) and skewed heavily to one of the task: 4-2-1 in the in-tree example (where there will be 4 times as

Table 6.1: Application Model Settings. These determine when a packet is generated for each of three tasks in the application graphs shown in Figure 6.1.

1. *Ratio* relates to how many instances of each task should be created for the initial task mapping.
2. *Rate* determines how much time must elapse between packets being sent.
3. *Packets Required* defines how many packets a node needs to receive before it can enter the "CPU processing" state.
4. *CPU Time* is the amount of time that a packet simulates task execution on receipt of the correct number of packets.
5. *Packet size is the length of the payload within the task packets.*
6. *Packets to Send* determines how many packets a node sends out after finishing the "CPU processing" phase.

Task:	**1**	**2**	**3**
Ratio:	depends on task graph		
Rate:	4ms	0	0
Packets Required:	0	1	1
CPU Time:	1ms	1ms	1ms
Packet Size:	1KB	1KB	1KB
Packets to Send:	depends on task graph		

many task 1 nodes as there are task 3 nodes).

2. *Routing:* Once the tasks are mapped to Centurion, the routing tables need to be set up to reflect the paths between nodes. This is a non-trivial task especially due to the adaptive nature of the social insect inspired intelligence: a mapping generated at the start of the experiment will not be valid once a node has changed task/failed. Therefore only two basic routing schemes will be used for these experiments: *Manhattan routing* and *Random routing*. Manhattan routing takes a task mapping and calculates the XY distance [112] between two nodes. The direction that leads to the node with the target task assigned and the shortest XY distance is chosen as the routing table direction. This is repeated three more times for the other cardinal directions and the result will be a list of each direction (N, E, S, W) sorted by shortest XY distance to the target task. This is repeated for all tasks and then for all nodes. Random routing is simply a random initialisation of the routing tables. For each task the direction list (N, E, S, W, unless the node is at an edge of the NoC) is shuffled 1000 times and then loaded into the routing table. This is repeated for all tasks and all nodes. This does not give valid routing paths (unless by coincident) and so this may be

seen as unrealistic. However due to our experimentation with decentralised task switching, a random routing state may be the case once a series of task switches have happened without updating the routing tables.

3. *Intelligence Enable:* This condition dictates if the embedded intelligence (picoblaze, CIA) is used to control the knobs in the experiment.
4. *Task Switch Enable:* When this condition is set the node will read the task suggestion from the intelligence and switch its task when a new task is suggested.
5. *Intelligent Routing Enable:* When this condition is set the router will fetch the task directions from the Picoblaze instead of the routing table.
6. *Fault Injection Enable:* If this parameter has a non-zero value then this number of nodes will be failed halfway through the experiment (t=500ms).

All experiment runs last 1000ms after which time the experiment logs are downloaded from the nodes by the experiment controller (also any log buffers that fill up during experiment runtime are fetched via the node debug link whilst the experiment is running). These are then sent over the LVDS link for storage on the PC.

6.3. Experiment 1: Performance of System without Embedded Intelligence

6.3.1. Description

The purpose of this experiment is to introduce the test cases and benchmark the performance of the system without any intelligence models applied. This experiment also uses a much larger set of test initial conditions to explore the distribution of the starting states the effect the has on the distributions of the performance of each experiment. This will allow a suitable size set of test initial conditions to be used for the rest of the experiments. The in-tree, out-tree and fork-join task graphs will be evaluated using both Manhattan and random routing. This should verify that random routing is a difficult starting condition for the many-core despite the small number of tasks. These experiments will also look at the effect of the task ratio topology against the task graph, to qualify the need for optimising not only the topology of tasks but also the ratio of the number of each task. All intelligence models are unused for this experiment.

6.3.2. Experiment 1.1 In-Tree

The In-Tree task graph consists of a large number of Task 1 nodes, a smaller number of Task 2 nodes (which require two Task 1 packets to be received before sending a packet out) and an even smaller number of Task 3 packets (again which require two Task 2 packets before entering the compute state). Ideally this should benefit from a task ratio of twice as many Task 1 “producer” nodes as Task 2 “consumer” nodes, repeated for Task 2 “producer” for Task 3 “consumer”. Therefore a task ratio of 4:2:1 is expected to be more productive than a 1:1:1 task ratio.

Figure 6.2 shows the results of running the four experiment cases (ratio 1-1-1: Manhattan routing, random routing; ratio 4-2-1: Manhattan routing, random routing) with 1000 runs each with a different initial conditions. As there is no adaptivity provided by the intelligence the runtime of the experiment should have little effect on the performance. Application throughput is the most important metric as it represents how effective the task mapping is at getting the required data to Task 3 nodes. As expected it is clear that there is a significant penalty when random routing is applied, with many packets needing to be cleared by the “last resort” deadlock mechanism. There is also a large benefit for optimising the task mapping to the task graph. Interestingly the packet latency increases for this case, this is likely due to the many-to-one nature of the in-tree coupled with the routing schemes not guaranteeing an equal allocation of packets to nodes, leaving some Task 2 and Task 3 nodes being unused.

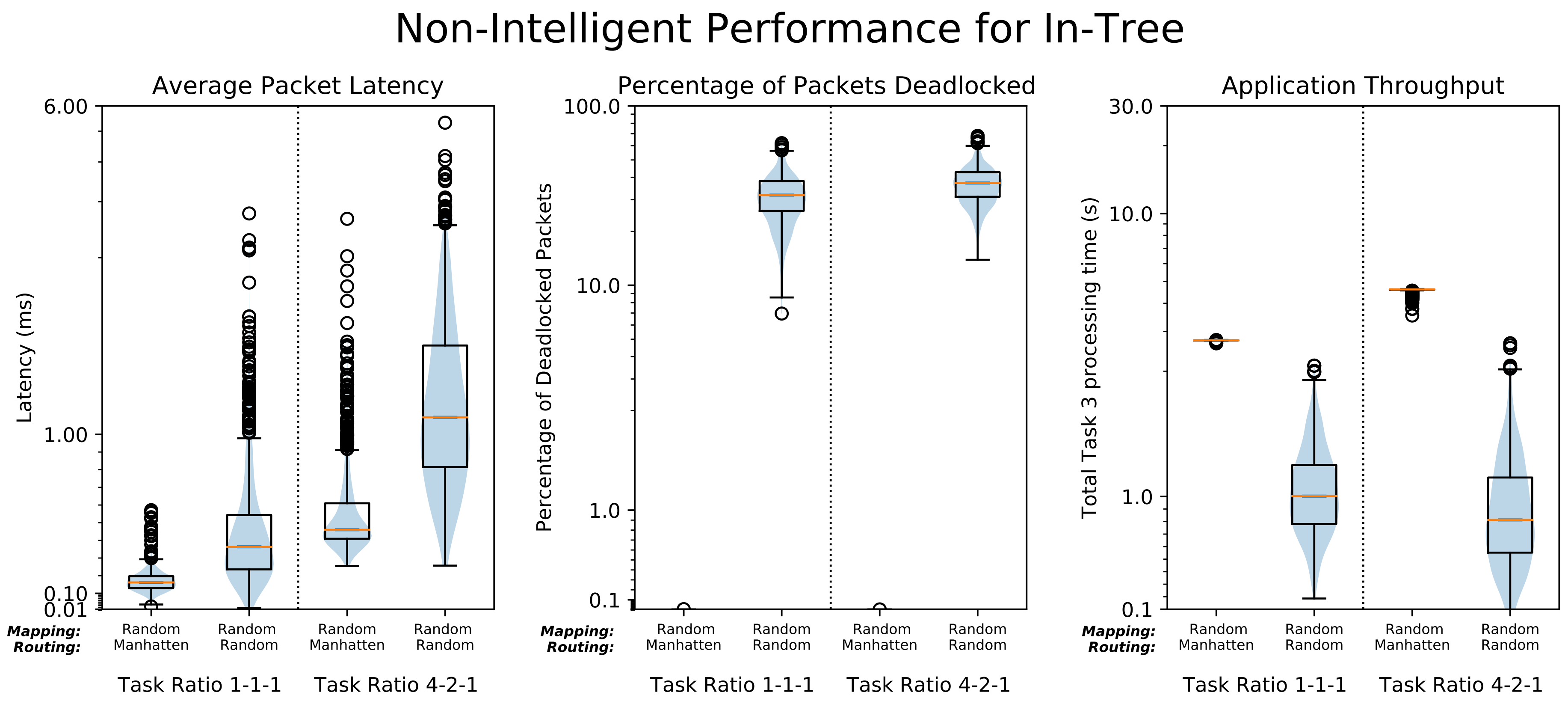


Figure 6.2: Performance metrics for Experiment 1.1

6.3.3. Experiment 1.2 Out-Tree

The next experiment investigates the out-tree task graph. This has the same task ratio as the in-tree graph but reversed i.e. 1:2:4. Each Task 1 “producer” node produces two Task 2 packets, which in turn produce two Task 3 packets to be consumed by the Task 3 nodes, resulting in 4 times as many Task 3 packets being produced for each Task 1 packet. As there is currently no way of dealing with parallelism of packets in the network, all packets will be sent on the same routing path to the task node with no spreading of load to other nodes of the same task. Therefore lots of local congestion is expected as the packets cannot be distributed further across the network from the first node that has the correct task.

As can be seen in the results shown in Figure 6.3 the advantage of Manhattan routing over random routing for a packet’s latency is less marked. This is evidence of the loading effect discussed above. The minimal XY distance that Manhattan routing promotes will mean that in certain topologies nodes will be overloaded. Due to the blocking nature of wormhole routing this means such hotspots will delay packets in the local neighbourhood as the node is trying to process the backlog of packets. Random routing will promote more routing diversity, but at the cost of many deadlocked packets as can also be seen in the plots. It is interesting to note that when the task graph ratio is also considered this, unlike the previous experiment, does not improve the application throughput. This is also driven by the lack of parallelism of routing paths as the throughput is determined by the number of packets reaching Task 3 nodes, which in turn require Task 2 nodes to receive packets. Due to the lack of routing diversity a smaller number of Task 2 nodes are receiving packets and so a number of Task 2 nodes are not producing any packets at all. To compound this effect, the routing paths for Task 3 packets will also suffer from a lack of diversity so somne of the Task 3 nodes that do receive packets will be overloaded.

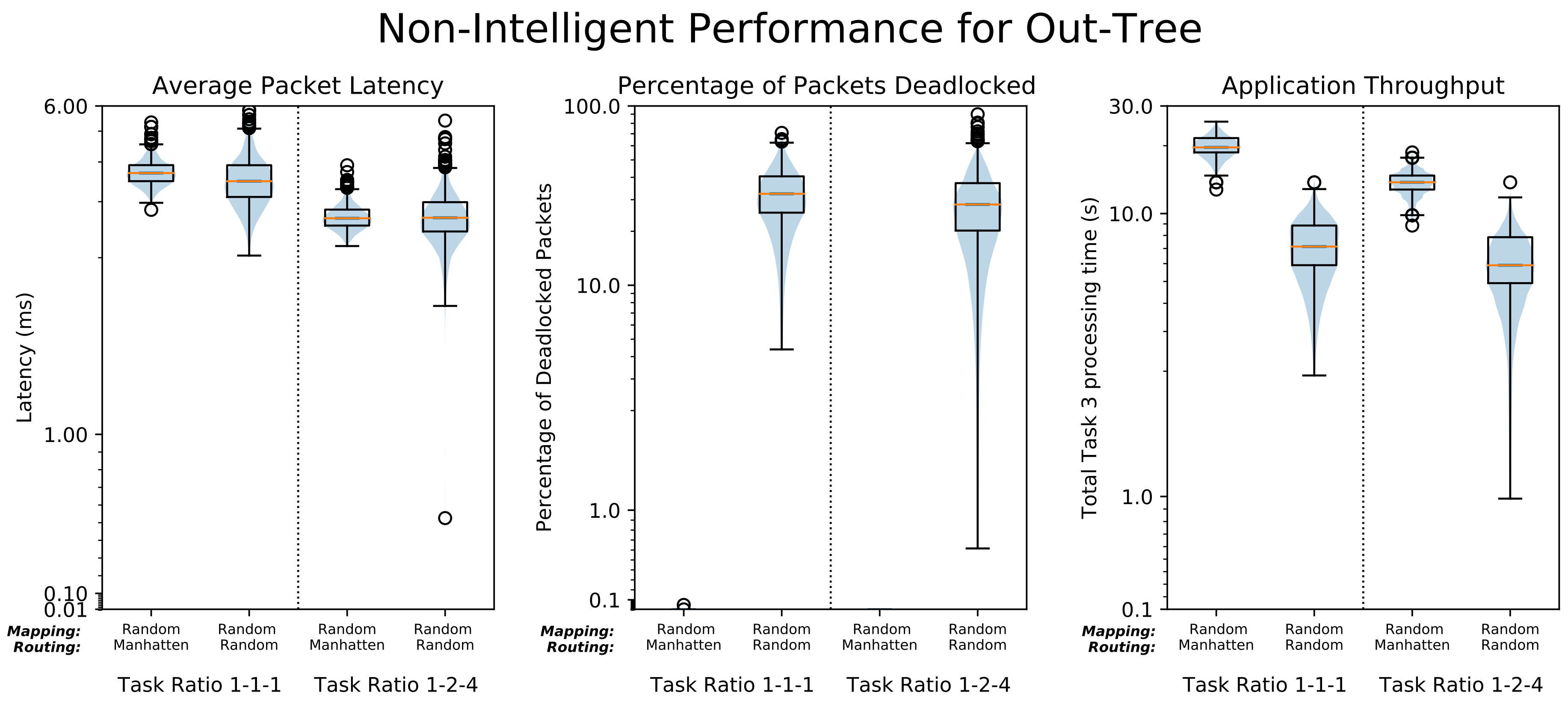


Figure 6.3: Performance metrics for Experiment 1.2

6.3.4. Experiment 1.3 Fork-Join

The final task graph to be experimented with is the fork-join. This has an out-tree and an in-tree phase and so it is expected that one of these sub-graphs may dominate the other's performance.

Figure 6.4 shows that this is the case and from the application throughput it would seem that the Out-tree restricts the improvement that a optimal task ratio would provide. However when the packet latency is considered it is clear that Manhattan routing has a noticeable improvement in latency distribution. This is likely to be contributed from the In-tree: the large number of Task 2 packets generated will need an efficient route to Task 3 nodes for reduction which Manhattan routing will provide. Once again, random routing produces a large number of deadlocked packets in both cases which will severely hamper the Task 3 throughput performance.

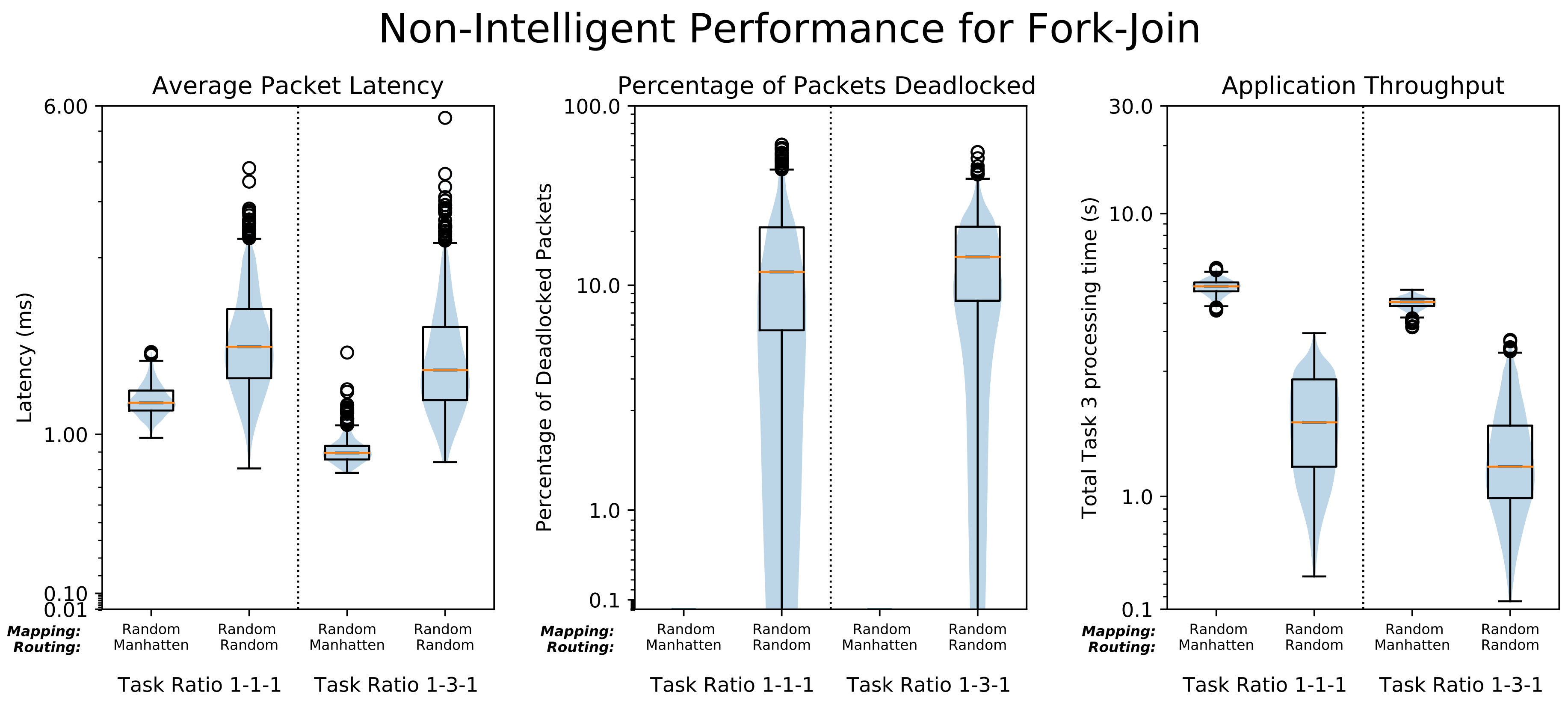


Figure 6.4: Performance metrics for Experiment 1.3

6.3.5. Review

From these three experiments the relationship between random routing, deadlocks and reduced throughput is clear. The adaptive cases will need to adapt with the random case to have a good chance of improving on the Manhattan cases. It is also seen that the lack of parallelism or diversity in the routing mechanism means that it is unlikely that a near to full use of all the many-core nodes will be seen in any experiments. Finally, from the underlying distributions shown by the violin plots it can be seen that the all distributions are centred around their medians with several large outliers responsible for when the distributions are widened. Therefore the rest of the experiments for this thesis will use 100 runs (i.e. 100 random starting conditions) for each individual investigation.

6.4. Experiment 2: Adaptive Task Allocation using the Interaction Network Intelligence Model

6.4.1. Biological Inspiration

The first emergent task allocation scheme that we experiment with is Gordon's Interaction Networks Model [113][24][6]. As discussed in Section 3.3, the key aspect of decision making within this task allocation model is the patterns of interactions between colony members. In the parallel distributed model Gordon proposes in [24], threshold decision functions were used by each agent (member of the colony) to determine which of eight states the agent should currently be fulfilling depending on its interactions with other agents. It was found that not only did this model exhibit several characteristics of colony dynamics, it also allowed perturbations in tasks to be introduced and the agents in the system would then adapt their states until it would eventually return to a normal, stable state.

When the ants interact they use their antennae to detect the chemicals that their colleague is covered in. From this they can infer which task their colleague has been undertaking (e.g. an ant covered in soil was probably doing nest maintenance). This is then used to inform their choice of task with information from other interactions and other stimuli taken into account [6]. To capture these dynamics in the many-core, this experiment uses the arrival of packets at the router as an interpretation of the interactions that the ants perform, the pattern and amount of packets arriving will infer which tasks a node's neighbourhood are performing. Congestion, faults and application changes will alter these information patterns in a comparable fashion to the ant's information environment.

6.4.2. Experiments Overview

The aim of these experiments is to start from an initial random task mapping and emerge a new task mapping that minimises the distances and congestion between the three task nodes of the applications described in Section 6.2. An example of the required transformation is shown in Figure 6.5. The experiments will also start from random routing table entries and so will emerge a task mapping to suit this random routing pattern. The use of random routing entries is important for large scale many-core systems as a node changing task would make a existing routing mapping invalid and in a decentralised system this would require difficult online analysis to keep all routing paths updated and valid.

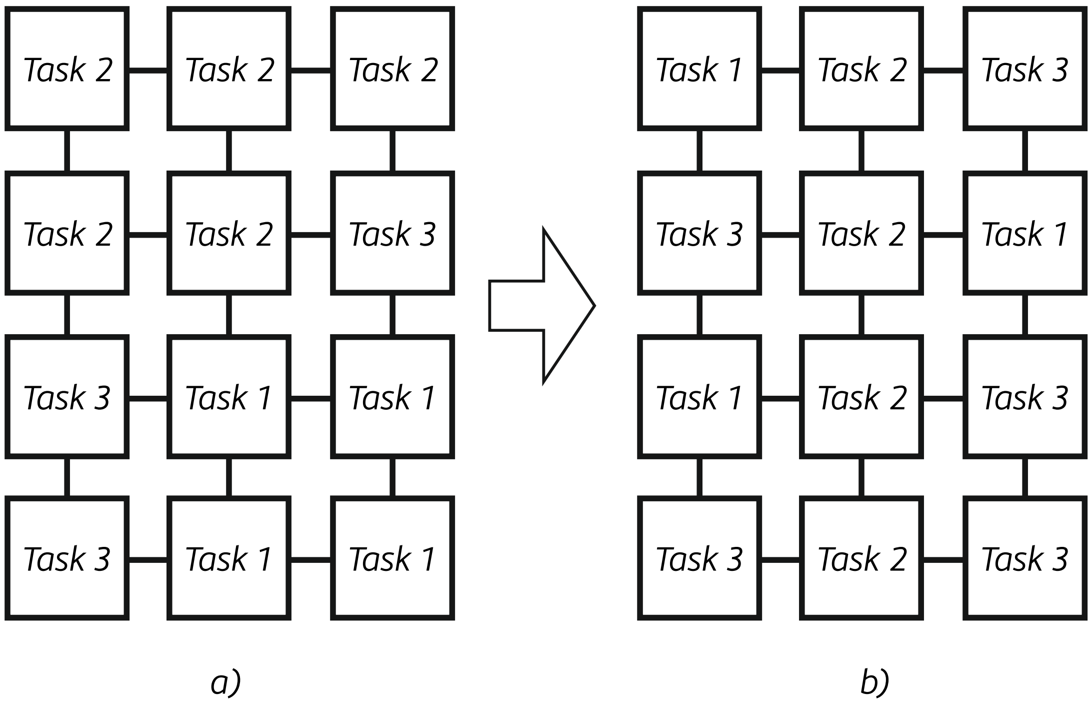


Figure 6.5: The concept of the task allocation experiments used for the following sections. Most experiments will start from a random task mapping *a)* and the role of the intelligence is to emerge a more efficient mapping such as *b)*.

The intelligence model will sense the state of local packet patterns, feed this into the decision making threshold units and then output the suggested task to the attached Microblaze MCS processor. The processor will then switch task and use the parameters from Table 6.1 for the new task it is undertaking. As we have seen in the previous experiment (Section 6.2), the test task graphs provide different optimisation requirements and so these experiments will evaluate how the interaction model can cope with the different packet patterns.

For the linear task graph the experiment will look at the effect of the Manhattan and random routing schemes on the ability of the intelligence. For the rest of the task graphs, the random routing is used and the task mapping ratio starting condition is modified. 100 runs for each set of stating experiment settings are used as the previous experiment showed that the typical experiment set has a tight distribution for the 1000 runs used in that experiment.

6.4.3. Intelligence Implementation

The Picoblaze is used to implement these experiments.

Knobs and Monitors

Monitor: The router event FIFO, to detect SOPs of each task passing through the NESW ports of the router. The I port is discounted to stop a positive feedback loop whereby a task's own output packets would cause a switch away from the current task.

Knob: The suggested node task. This is fed into the Microblaze node which will change task to reflect this value.

Picoblaze Software

The flowchart in Figure 6.6 shows the flow of the assembler code written in the Picoblaze to implement the interaction model. The threshold values are loaded from the Experiment controller with a default value of 5 tasks. These are stored in the Picoblaze's scratchpad memory alongside the current count value of the threshold. When a SOP is detected the task is extracted and the count and threshold value retrieved. The count is increased and compared against the threshold value. If the count value is greater than or equal to the threshold then this task is chosen, the other counters are cleared and the task switch process starts again.

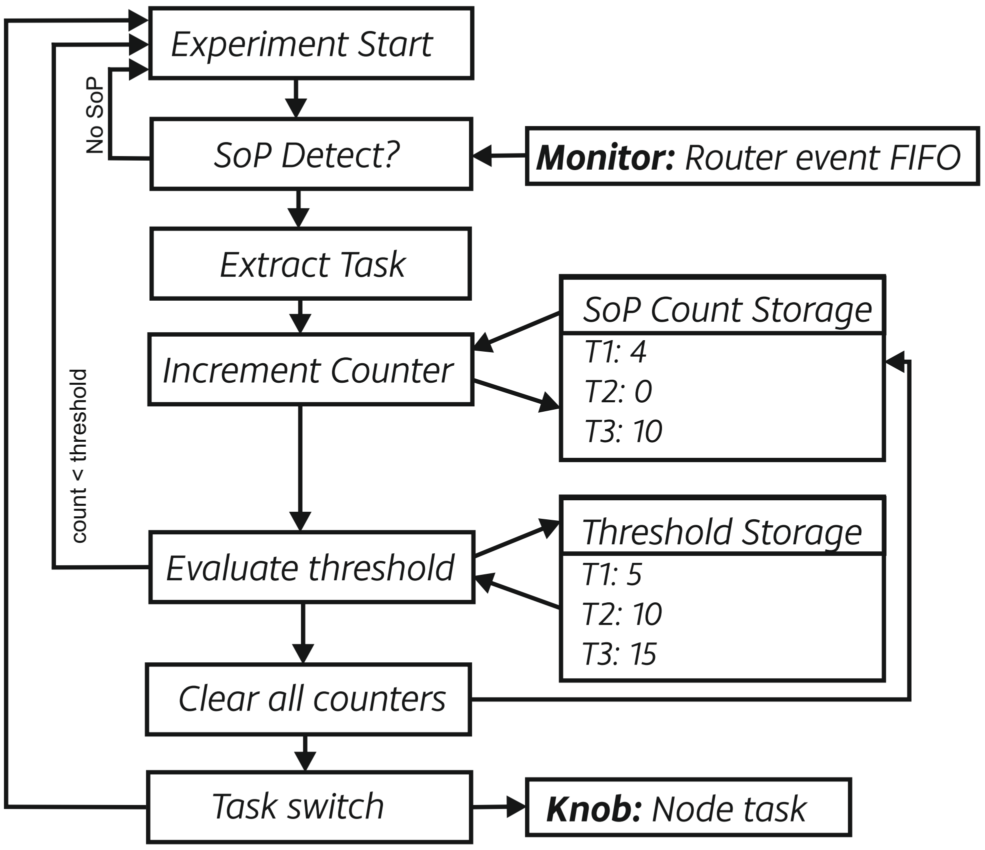


Figure 6.6: Design of Picoblaze software for running the Network Interaction Intelligence Model

6.4.4. Experiment 2.1: Interaction Network Model - Linear Task Graph

For the linear task graph it is expected that the difference between Manhattan and random routing will be reduced as, due to the fact that each node has 4 neighbouring nodes and has only two potential target tasks, it is likely that a target task can be reached in a small number of hops. The intelligence should be able to adapt to this random routing pattern relatively easily as any task switches will only affect one upstream or downstream node at a time (as there is no parallelism).

Figure 6.7 shows the performance metrics for the linear task graph. The ability of the intelligence model to adapt to the random routing case is clear with the reduction in the number of deadlocked packets a clear indicator that nodes are being switched to take up better task allocations in routing paths that are poor enough to require the deadlock mechanism to step in a sink the packet. The total throughput is also improved from the random case, the median is still a small way away from the Manhattan performance so therefore the intelligence has not managed to emerge a better mapping than the mapping heuristic. It is interesting to note that the Manhattan case is worse for the emergent task mapping. This will be due to the limited routing path variability that Manhattan routing will give. The paths are always to the closest node of the correct task and so a neighbourhood of nodes will likely have similar paths for each task; leading to little routing variability which Gordon's model would require to disrupt the system into more optimal task mappings.

Figures 6.8 and 6.9 show the time domain view of these metrics and the task switch characteristics for the experiment run that produced the median throughput for each experiment setting. From the throughput and deadlock metrics for the adaptive, random routing case it can be seen that a very quick adaptive phase takes place and then the system settles down. This is also seen in the adaptive, Manhattan routing case but there appears to be less settling on the packet latency. This will be caused by hotspots created by the lack of local routing variety that is a side effect of Manhattan routing. The quick settling time is also seen in the task switch metrics. Of much interest here though is the task distribution which, for the random routing adaptive case, shows the

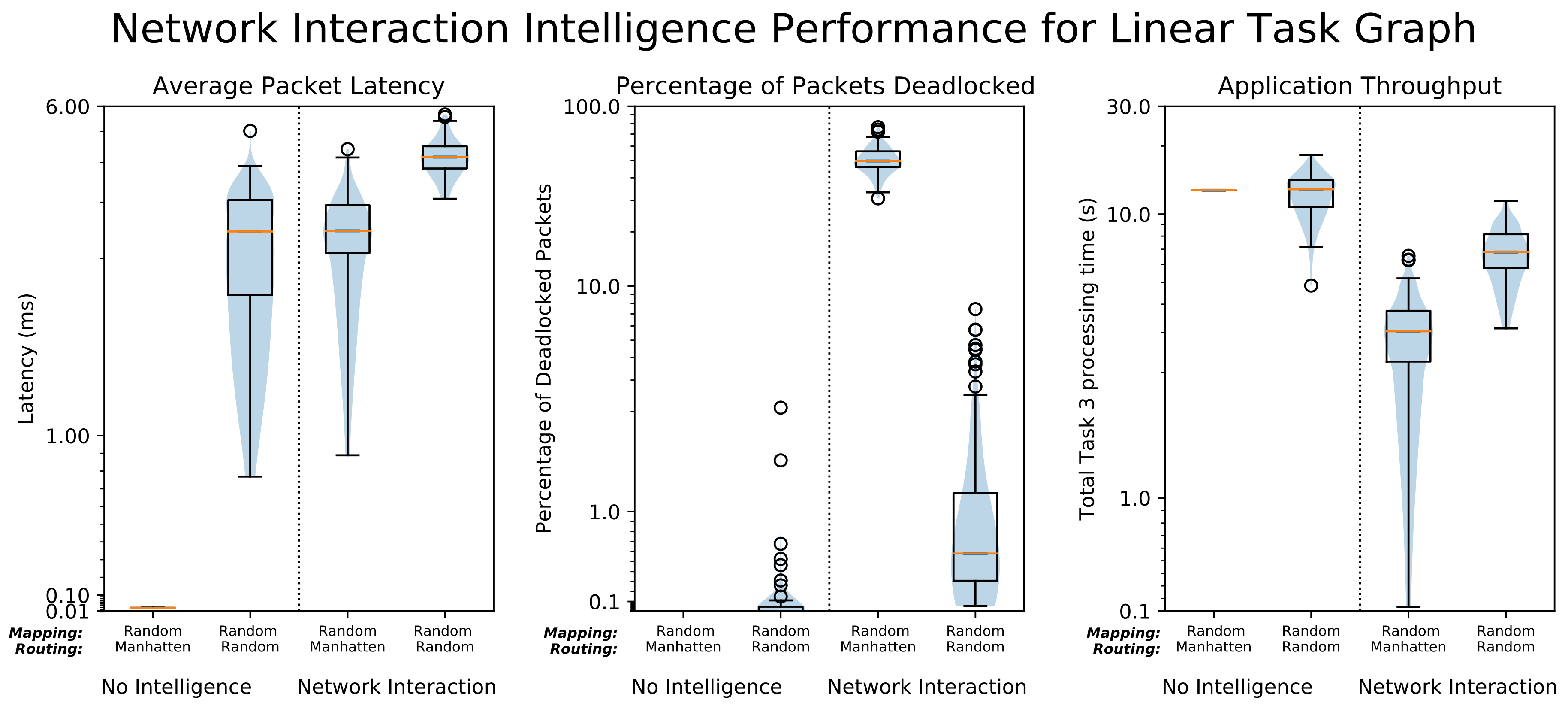


Figure 6.7: Performance metrics of Experiment 2.1

system has reduced the number of task 1 nodes (for the linear task graph it is expected that all ratios are the same, 42 nodes). As there is no Task 3 to Task 1 edge on the task graph there is no incentive for nodes to switch to Task 1 nodes. This has the effect of choking the maximum use of the 128 nodes to only 60 nodes as 20 Task 1 nodes can only produce data to keep 20 Task 2 nodes and so 20 Task 3 nodes busy.

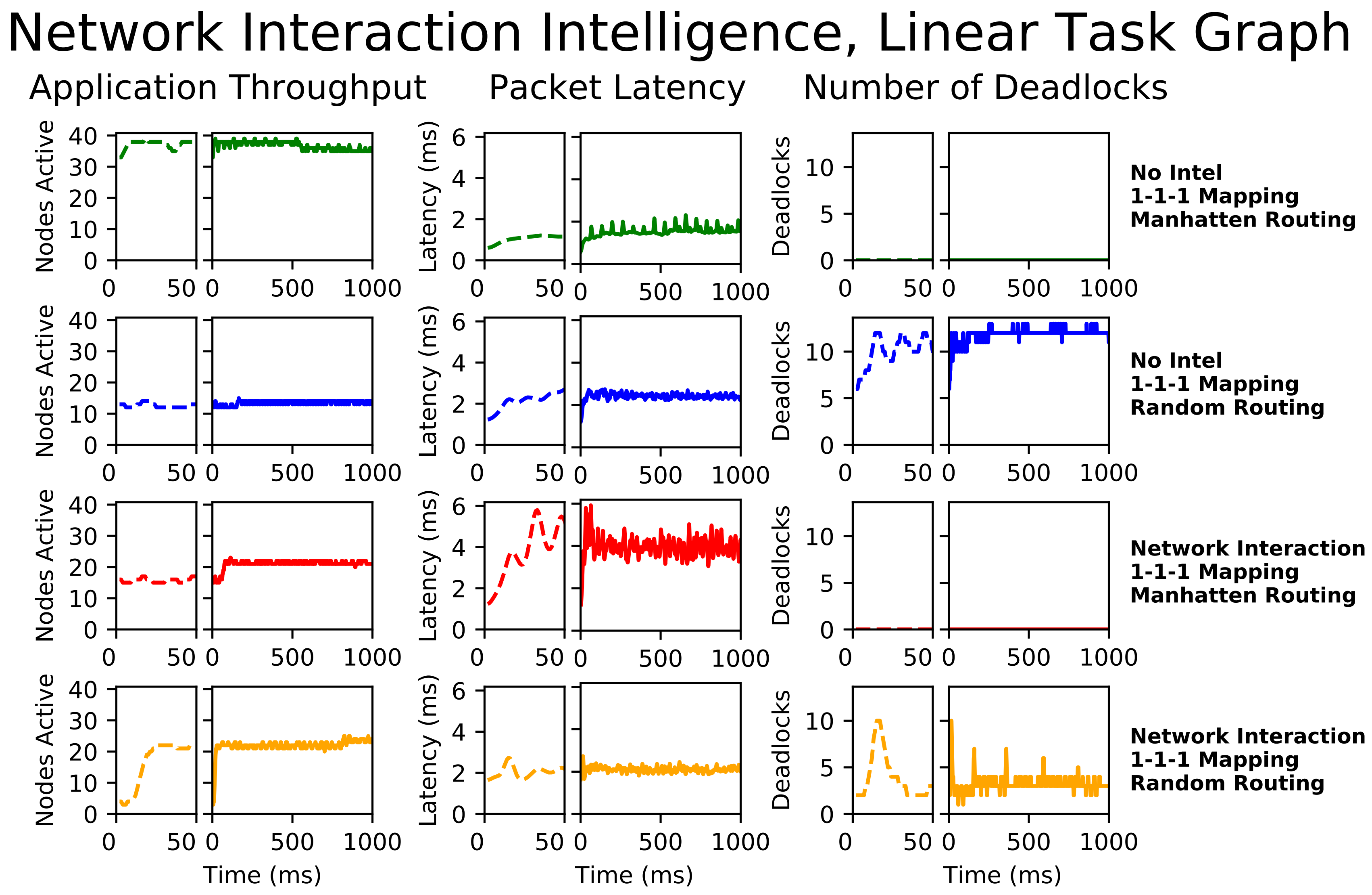


Figure 6.8: Packet Analysis for Experiment 2.1. The first section of each graph (dashed plot line) is an focus on the first 50ms of the experiment.

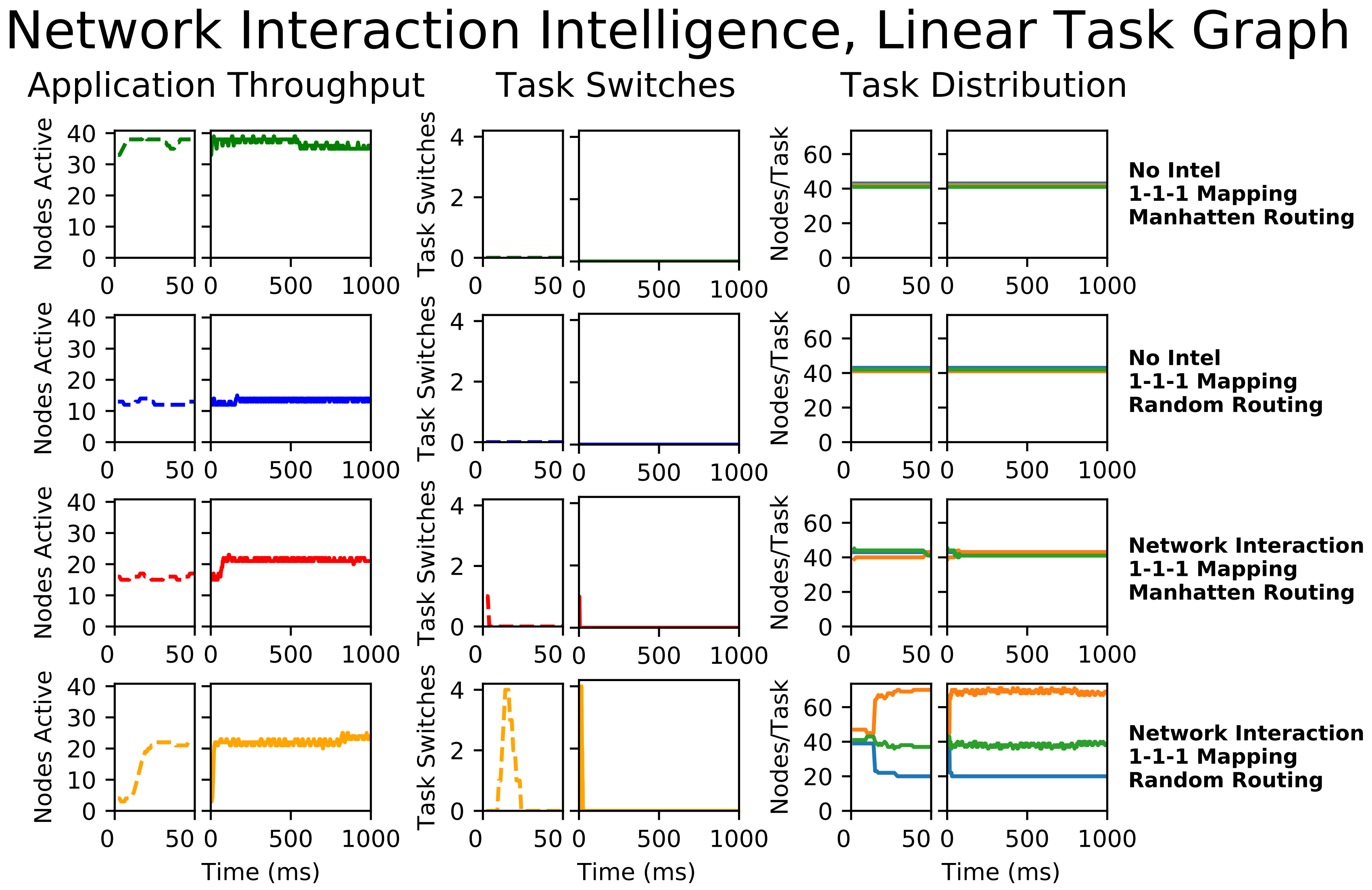


Figure 6.9: Task Switching Analysis for Experiment 2.1 **blue:** Task 1, **orange:** Task 2 **green:** Task 3. The first section of each graph (dashed plot line) is an focus on the first 50ms of the experiment.

6.4.5. Experiment 2.2: Interaction Network Model - In tree

For the in-tree case we would expect the interaction model to capture the 4:2:1 mapping which we saw was a more effective mapping in the previous experiment. Figure 6.10 shows the performance metrics for the in-tree with the network interaction model. When compared to the linear model it can be seen that the starting ratio has an positive effect on the throughput but a negative effect on the number of deadlocks. The packet latency stays similar regardless of the starting task topology ratio which may indicate that the intelligence is reducing Task 1 nodes to provide more Task 2s for sinking the larger number of Task 1 packets (although without parallel routing paths this effect will be limited. When the time domain graphs (Figure 6.11 and 6.12) are taken into account there is confirmation of this effect: the number of Task 1 nodes are swiftly reduced from their higher starting ratio and the number of Task 2 nodes increases rapidly, followed later by Task 3 nodes.

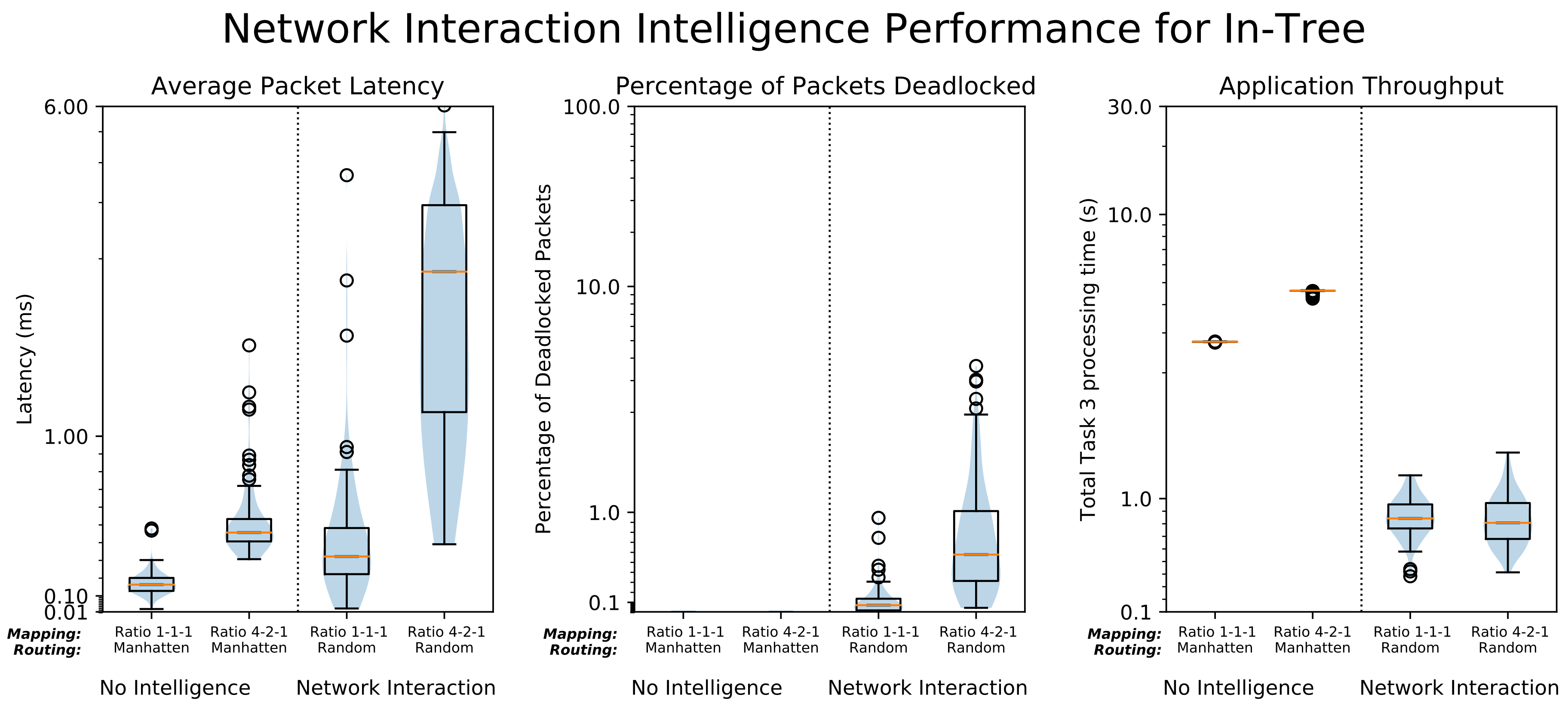


Figure 6.10: Performance metrics for Experiment 2.2

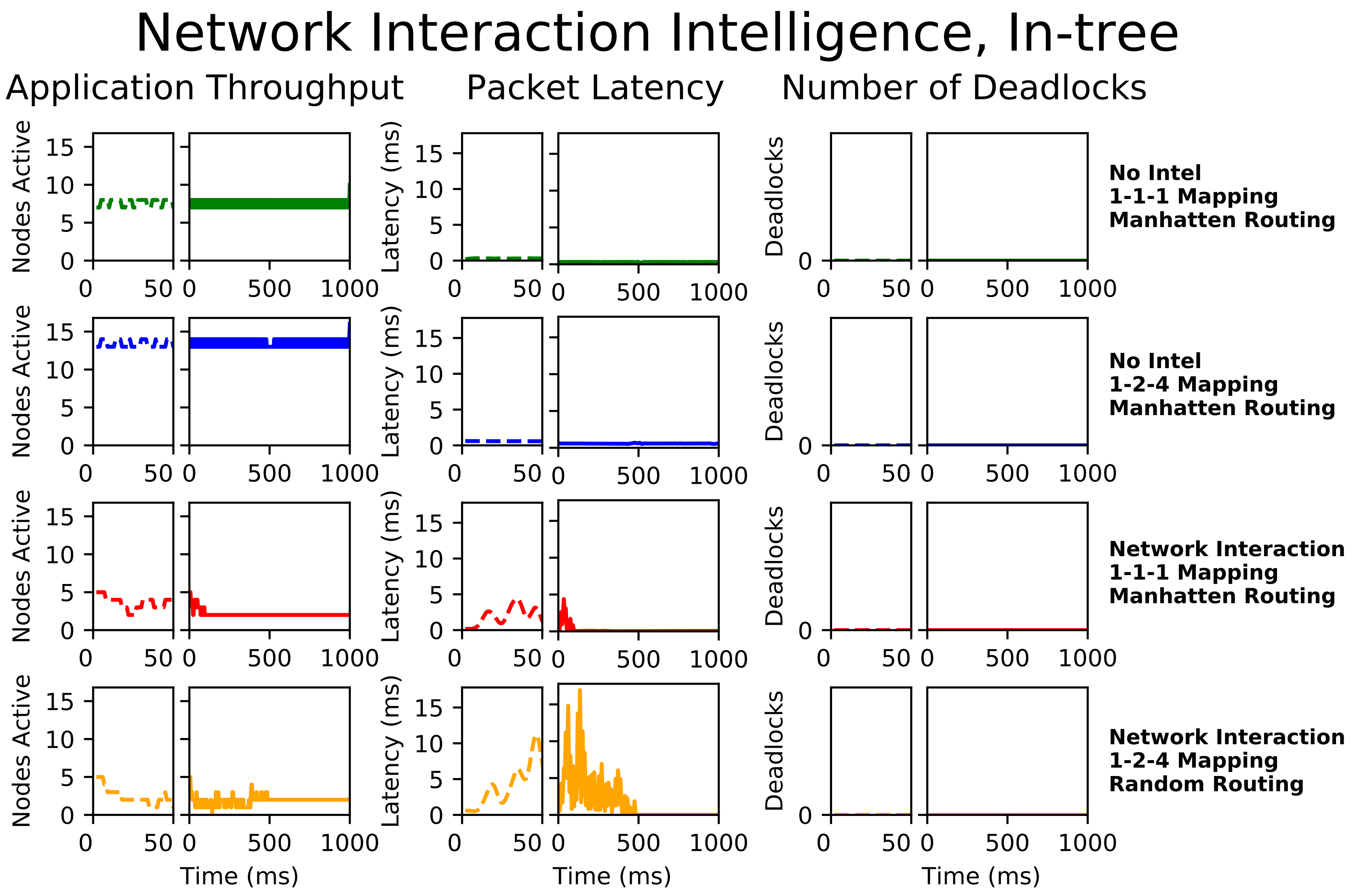


Figure 6.11: Packet Analysis for Experiment 2.2. The first section of each graph (dashed plot line) is an focus on the first 50ms of the experiment.

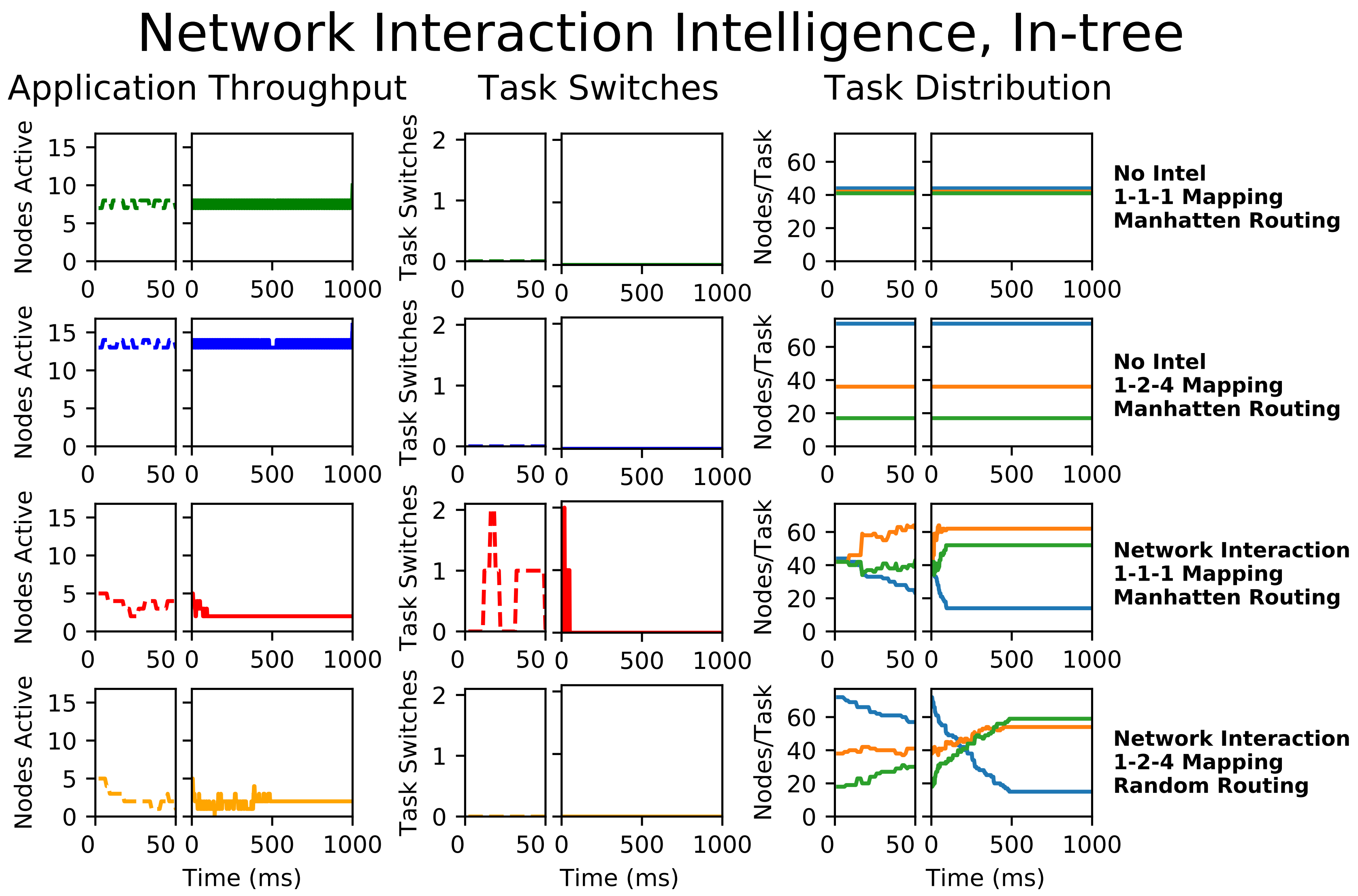


Figure 6.12: Task Switching Analysis for Experiment 2.2 **blue:** Task 1, **orange:** Task 2 **green:** Task 3. The first section of each graph (dashed plot line) is an focus on the first 50ms of the experiment.

6.4.6. Experiment 2.3: Interaction Network Model - Out tree

The out-tree performance metrics shown in Figure 6.13 show that the intelligence struggles to improve the overall topology for this task graph, as seen by the large range in packet latencies and a high number of deadlocks. However, the reasonable throughput median shows that there are some configurations where it does manage to find good topologies. When the time domain data is considered (Figures 6.14 and 6.15) only a slight dropping of Task 1 nodes is seen. This is a smaller degree of task switch for this task graph when compared to the larger number of Task 2 packets in the previous task graph. This shows that the number of packets generated by a task will have a large influence on the switching of the nodes to that task. This is expected as the interaction model is based purely on the interactions occurring around the node. This time, however, the settling time of the experiment is longer and is related to the longer time that is has taken to remove the Task 1 nodes from the system.

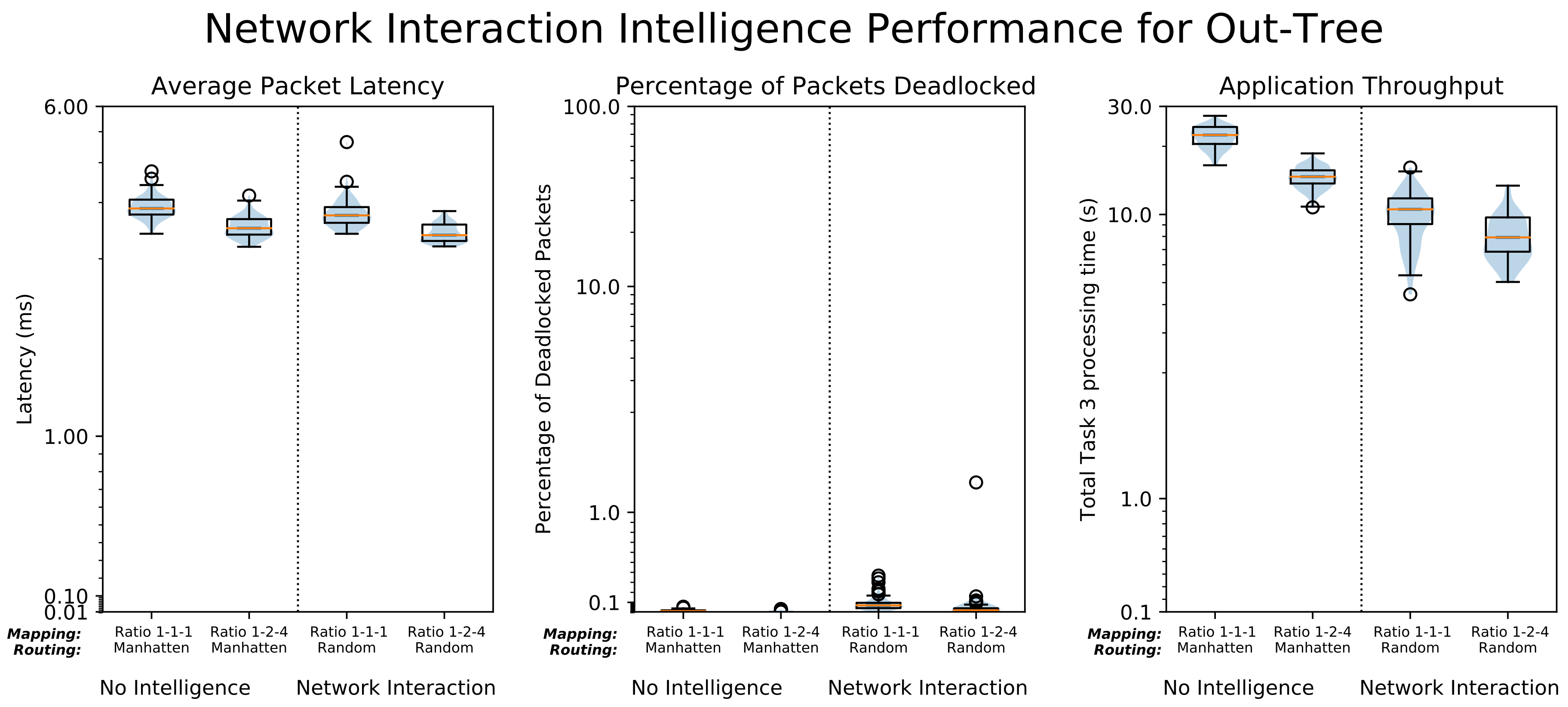


Figure 6.13: Performance metrics for Experiment 2.3

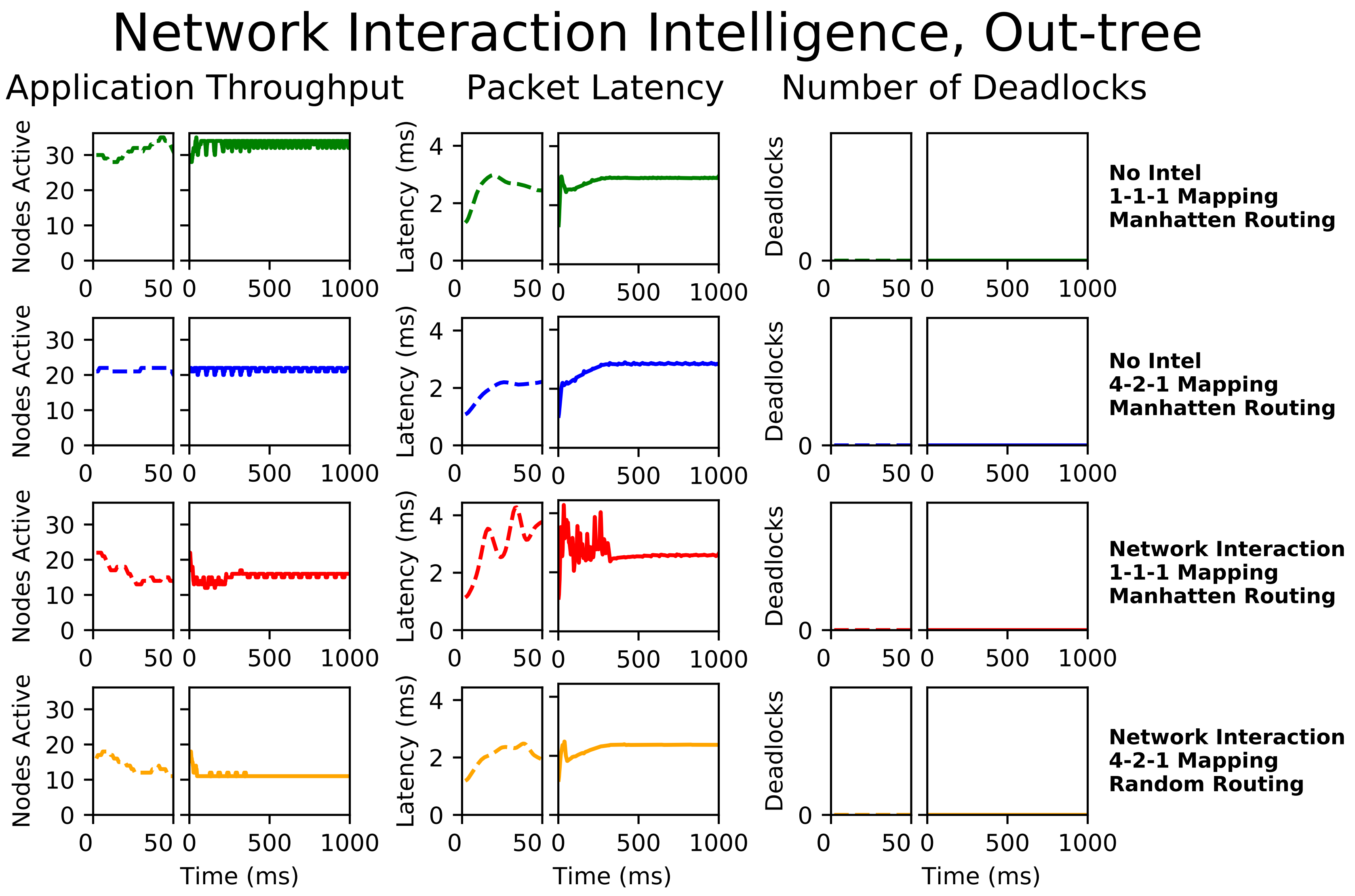


Figure 6.14: Packet Analysis for Experiment 2.3. The first section of each graph (dashed plot line) is an focus on the first 50ms of the experiment.

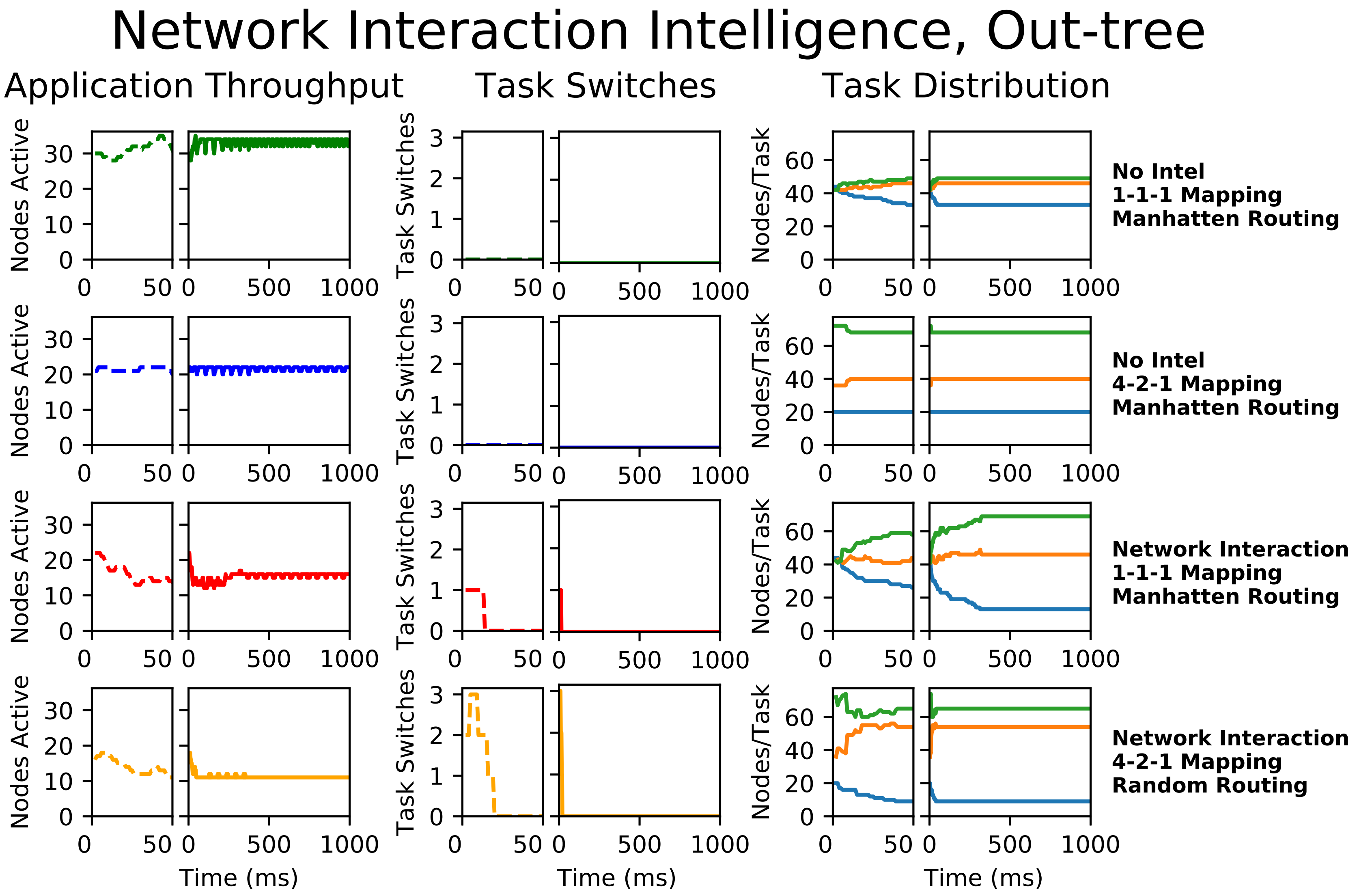


Figure 6.15: Task Switching Analysis for Experiment 2.3 **blue:** Task 1, **orange:** Task 2 **green:** Task 3. The first section of each graph (dashed plot line) is an focus on the first 50ms of the experiment.

6.4.7. Experiment 2.4: Interaction Network Model - Fork Join

With the poor throughput performance of the interaction model with both the In-tree and Out-tree experiments, it is unlikely that Fork-Join will show any improvement as it combines these two graphs. Indeed as can be seen in Figure 6.16 this is the case and Figure 6.18 shows that once again Task 1 nodes are being optimised away for both the 1:1:1 case and the 1:2:4 case, the lower starting number of Task 1 nodes in the 1:2:4 mapping case means that the number of Task 1 nodes is reduced quickly and thus the overall throughput is reduced. This is clear in Figure 6.17 whereby the number of active nodes does not reach over 10 at any one time.

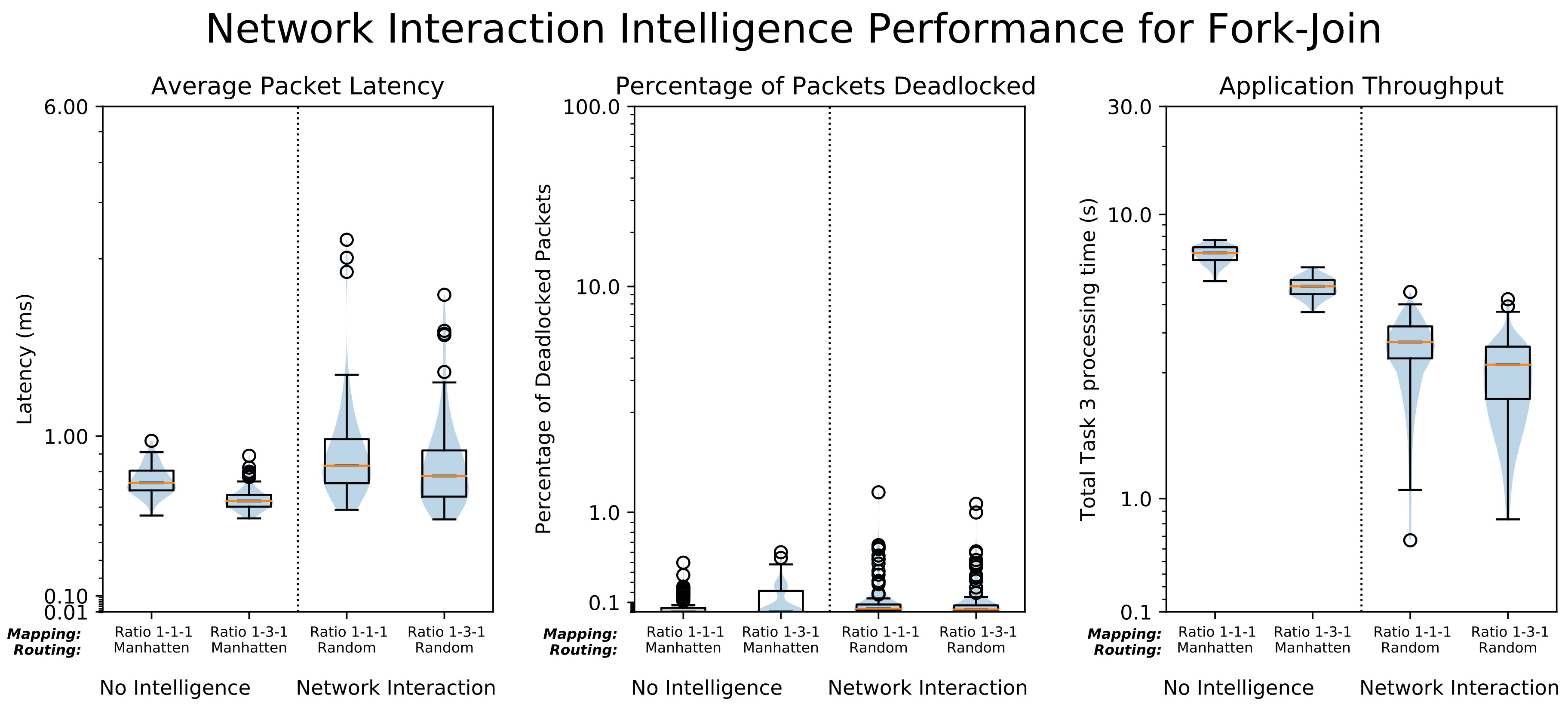


Figure 6.16: Performance metrics for Experiment 2.4

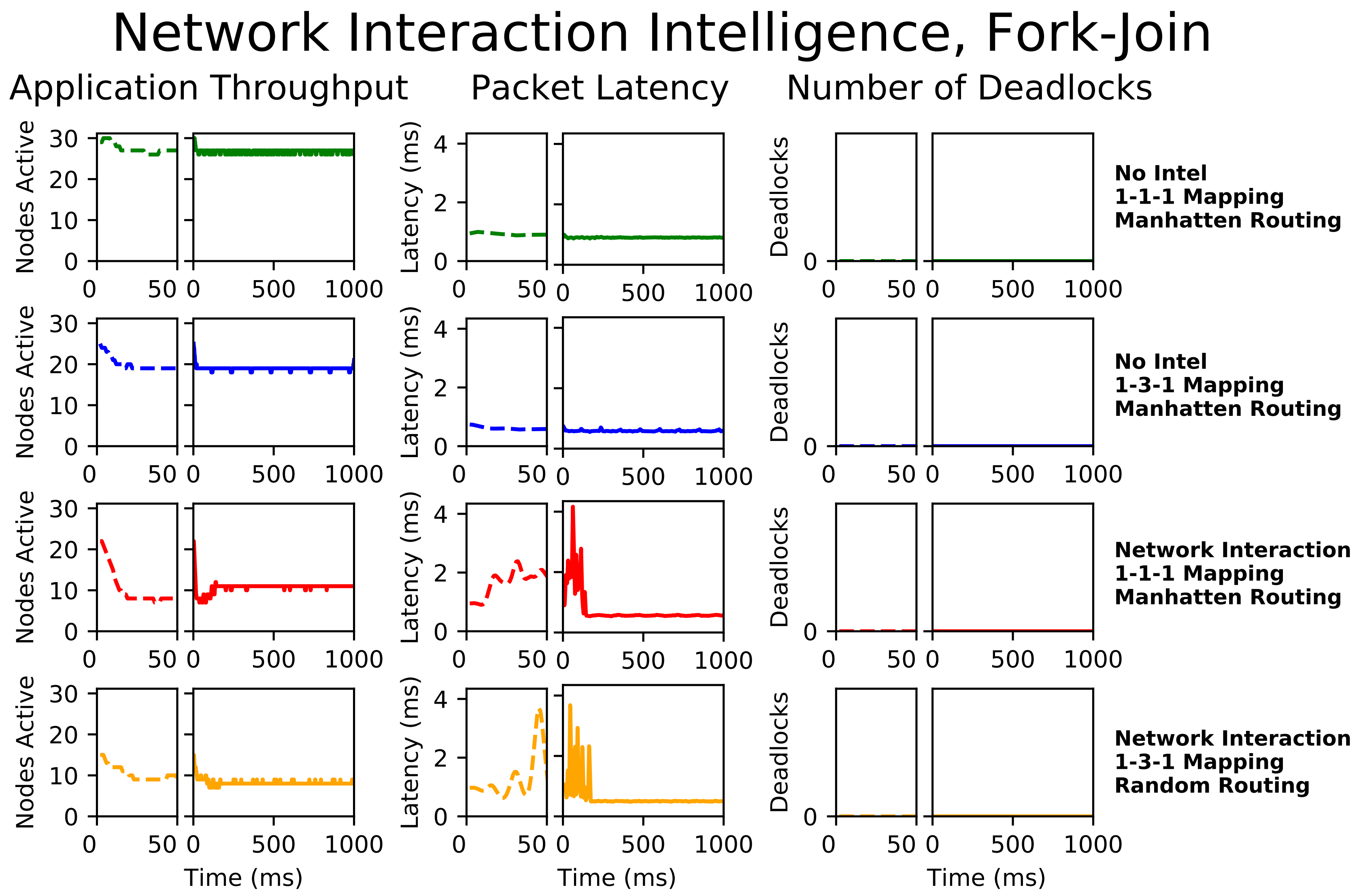


Figure 6.17: Packet Analysis for Experiment 2.4. The first section of each graph (dashed plot line) is an focus on the first 50ms of the experiment.

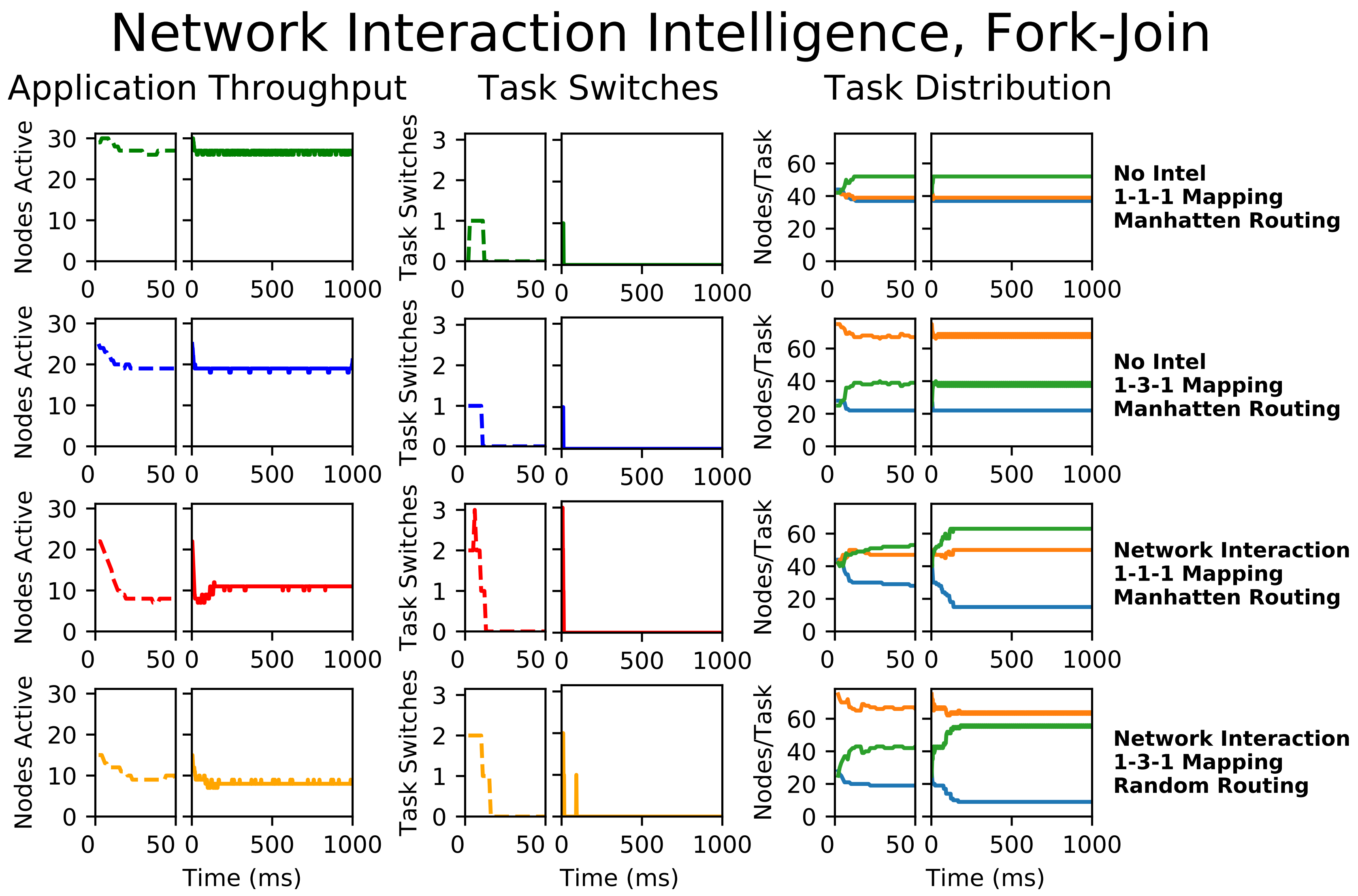


Figure 6.18: Task Switching Analysis for Experiment 2.4 **blue:** Task 1, **orange:** Task 2 **green:** Task 3. The first section of each graph (dashed plot line) is an focus on the first 50ms of the experiment.

6.4.8. Review

The interaction model has shown some adaptivity and settles down relatively fast to topology that are stable with respect to packet latency and the number of deadlocks. However, the prime method of achieving this settledness seems to come from reducing the number of Task 1 nodes to a level that means the task graph is very poorly supported. This was the case for all task graphs with a parallel element. For the linear case the interaction network improved on random and successfully reduced deadlocks, however the ratio of Task 1 nodes to 2 and 3 was still not satisfactory to reach the performance levels of the Manhattan routed case.

6.5. Experiment 3: Adaptive Task Allocation with the Foraging For Work (FFW) Model

6.5.1. Biological Inspiration

The second task allocation experiments uses the Foraging For Work (FFW) model [26] introduced in Section 3.3. The basis of Foraging For Work relies on workers being able to wander the nest once a task stimulus expires and their next task is determined by the nest task they encounter that exceeds its stimulus threshold. Once again this requires implementation using stimulus response-threshold decision making units, but further to Gordon's model also has a time element: the period of time that elapses since the last task was performed to a task switch is accepted. In this set of experiments we shall refer to this as the *task decay time*.

6.5.2. Experiments Overview

As with the previous experiment, this set of experiments aims to adapt to a more efficient task mapping of the test application from a random task mapping and a random routing pattern (see Figure 6.5). The intelligence model will again need to sense the state of packets moving through the router as this represents the nest walks that a colony worker may do; albeit instead of the router moving to the packets in the case of the current task stimulus dropping, the new packets are moving to the router.

The experiment structure takes the same form as the previous set of experiments. The first experiment will compare the FFW intelligence with optimal mappings and random mappings with no intelligence.

6.5.3. Intelligence Implementation

The Picoblaze is used to implement these experiments.

Knobs and Monitors

Monitor: The router event FIFO, to detect SOPs of each task passing through the NESW ports of the router. The I port is discounted to stop a positive feedback loop whereby a task's own output packets would cause a switch away from the current task. *Monitor:* A fixed time reference, set here to 1ms and used with a threshold to gain 20ms as Task 1s output at a period of 4ms. This means that 5 missed packets will open up the "forage" window whereby the intelligence can switch the node's task to the next packet that arrives.

Knob: The suggested node task. This is fed into the Microblaze node which will change task to reflect this value.

Picoblaze Software

The flowchart in Figure 6.19 shows the flow of the assembler code written in the Picoblaze to implement the FFW model. The timer tick happens every 1ms and the threshold value is set to 20 to result in a 20ms expiry window. As can be seen by the flowchart, any packet received for the correct task (same task as the node is executing) will result in the counter being cleared and the task switch inhibited for another 20ms. If the forage window is open then the task of the next packet to pass through the router, regardless of its task, is extracted and set as the new task for the node.

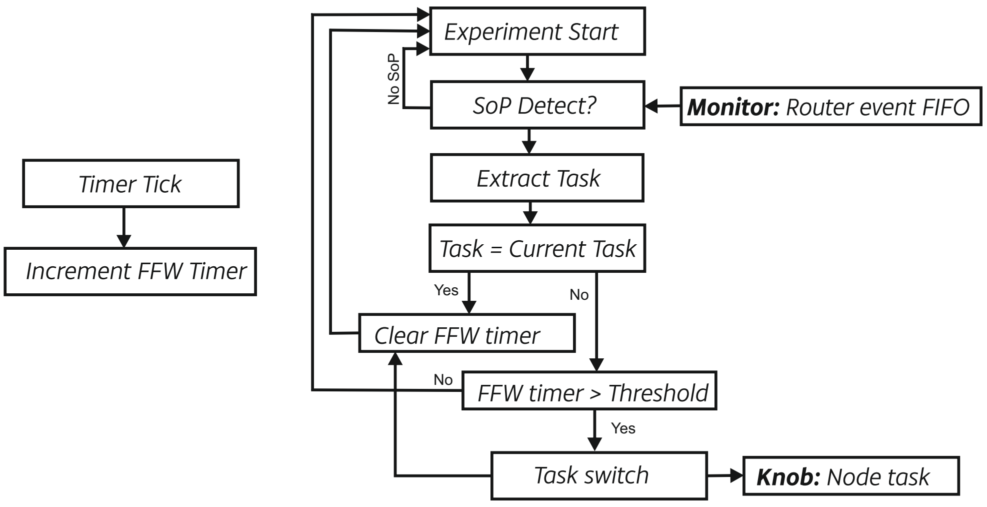


Figure 6.19: Picoblaze software design for implementing the Foraging for Work Task Allocation

6.5.4. Experiment 3.1: Foraging For Work - Linear Task Graph

As with the network interaction model, for the linear task graph it is expected that the difference between Manhattan and random routing will be reduced due to the fact that each node has 4 neighbouring nodes and has only two potential target tasks. A poor random routing set will result in packets not reaching their destination node in time and so the forage window will open and the node switch to the task. If this does not improve the situation then the node will end up switching task again and so it could be expected that FFW will be less stable than the interaction model.

Figure 6.20 shows the performance metrics for the linear task graph. As expected the distributions for random and Manhattan routing are quite similar. A promising reduction in packet latency is seen, however this comes with the cost of more deadlocked packets. However when the time domain of the medians are considered in Figures 6.21 and 6.22 it can be seen that, as with the network interaction model, there is a severe reduction in the number of nodes performing Task 1 and that the throughput of the nodes is very poor post adaptation. The thicker line on the task distribution shows that there is some degree of oscillation occurring between Task 2 and Task 3 nodes.

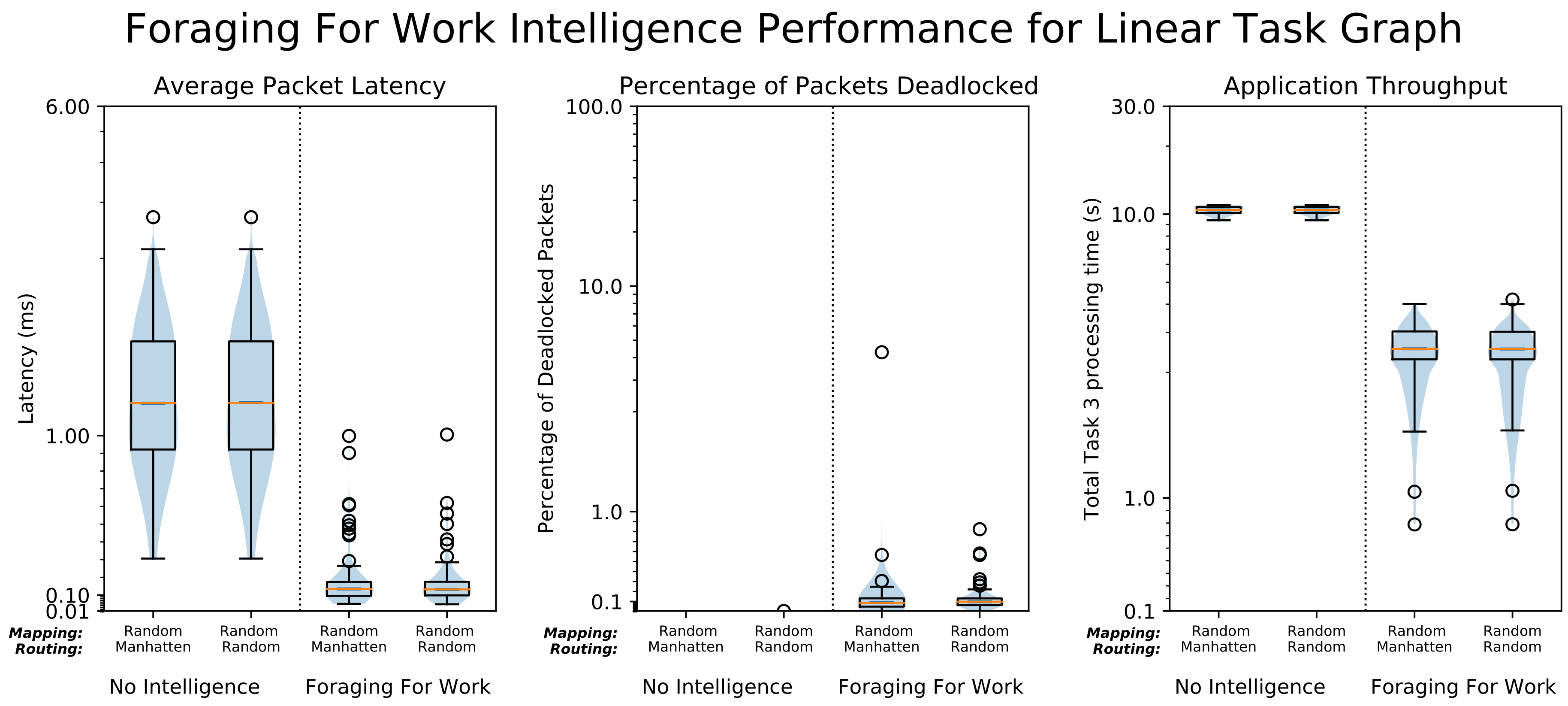


Figure 6.20: Performance metrics for Experiment 3.1

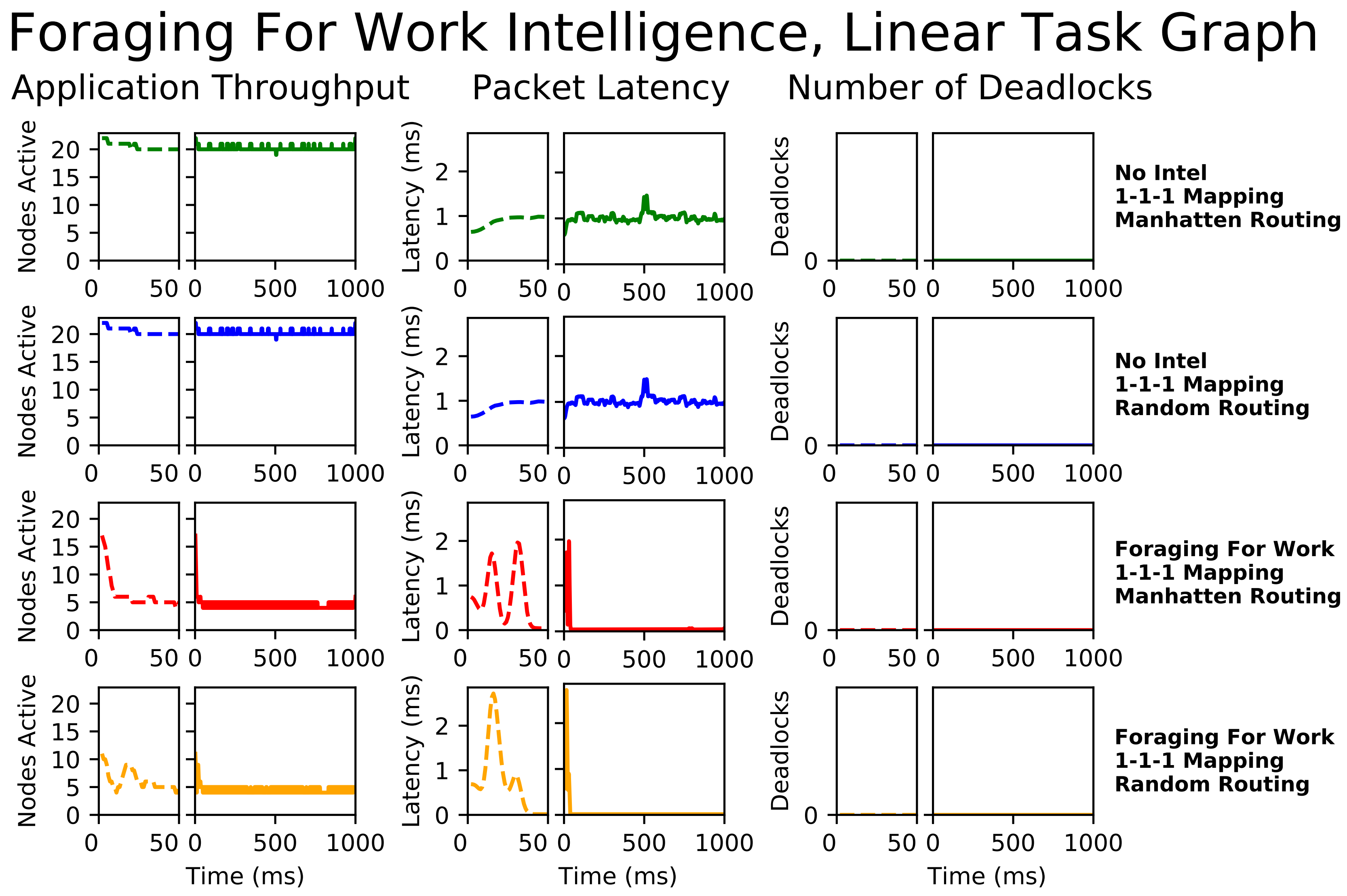


Figure 6.21: Packet Analysis for Experiment 3.1. The first section of each graph (dashed plot line) is an focus on the first 50ms of the experiment.

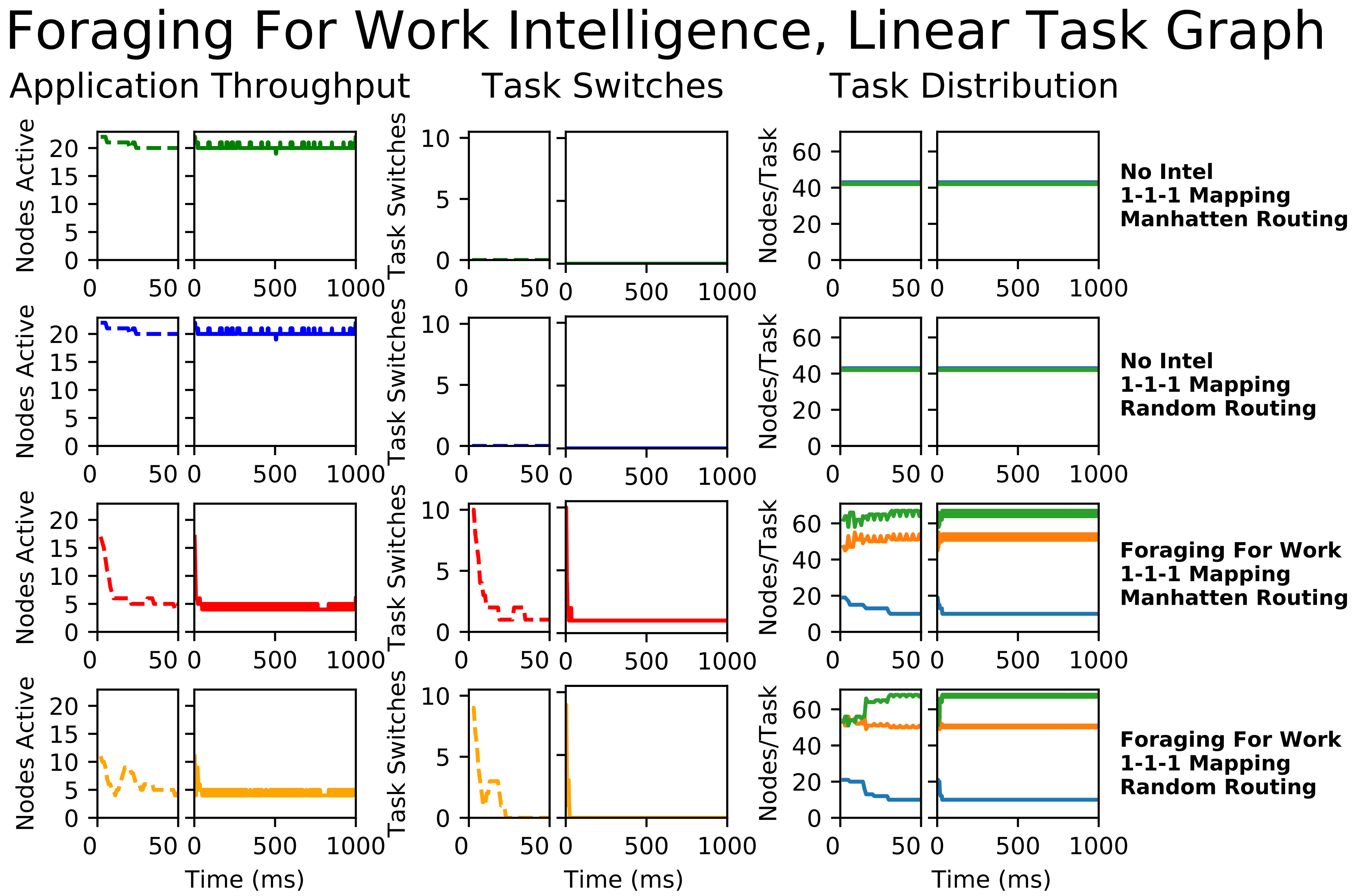


Figure 6.22: Task Switching Analysis for Experiment 3.1 **blue:** Task 1, **orange:** Task 2 **green:** Task 3. The first section of each graph (dashed plot line) is an focus on the first 50ms of the experiment.

6.5.5. Experiment 3.2: Foraging For Work - In tree

Figure 6.23 shows that the FFW performance for the In-Tree is very similar to the interaction model with very little difference between the 1:1:1 and 4:2:1 ratios and poor performance when compared to the Manhattan routed models. Once again looking at the time domain (Figure 6.24 and 6.25) shows the now familiar pattern of the Task 1 nodes dropping off and the network settling down into a very poor performance with only a few nodes active at once. As there is no Task 3 to Task 1 packets, there is no incentive for FFW nodes to move to the Task 1 state once the Foraging mode window is entered. This chokes the rest of the application as the Task 1 nodes are the producer task for the task graph. For the In-Tree, the choking of Task 1 packets has a huge impact on the performance of the application due to the reduction nature of the task graph: two Task 2 packets need to be emitted by Task 1 nodes for a Task 3 packet to be emitted.

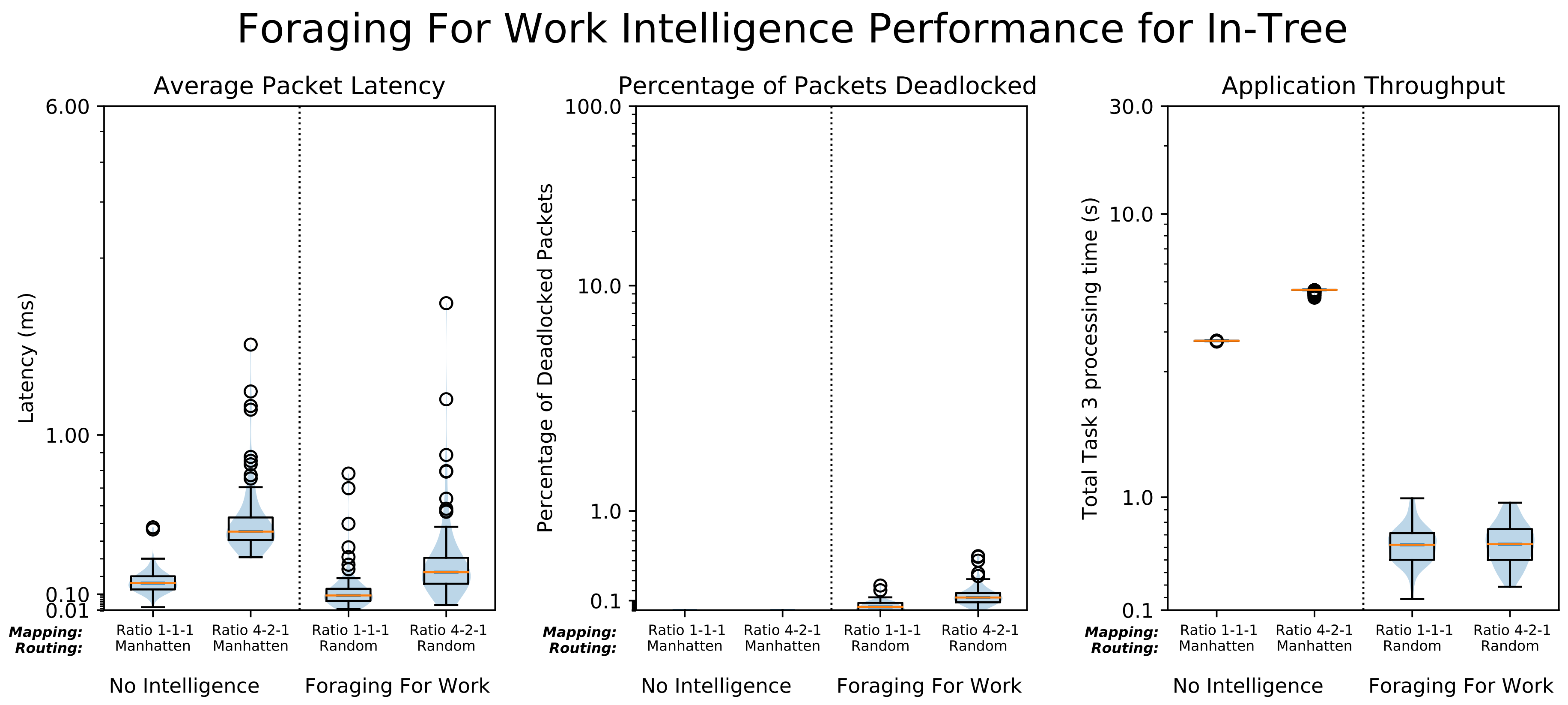


Figure 6.23: Performance metrics for Experiment 3.2

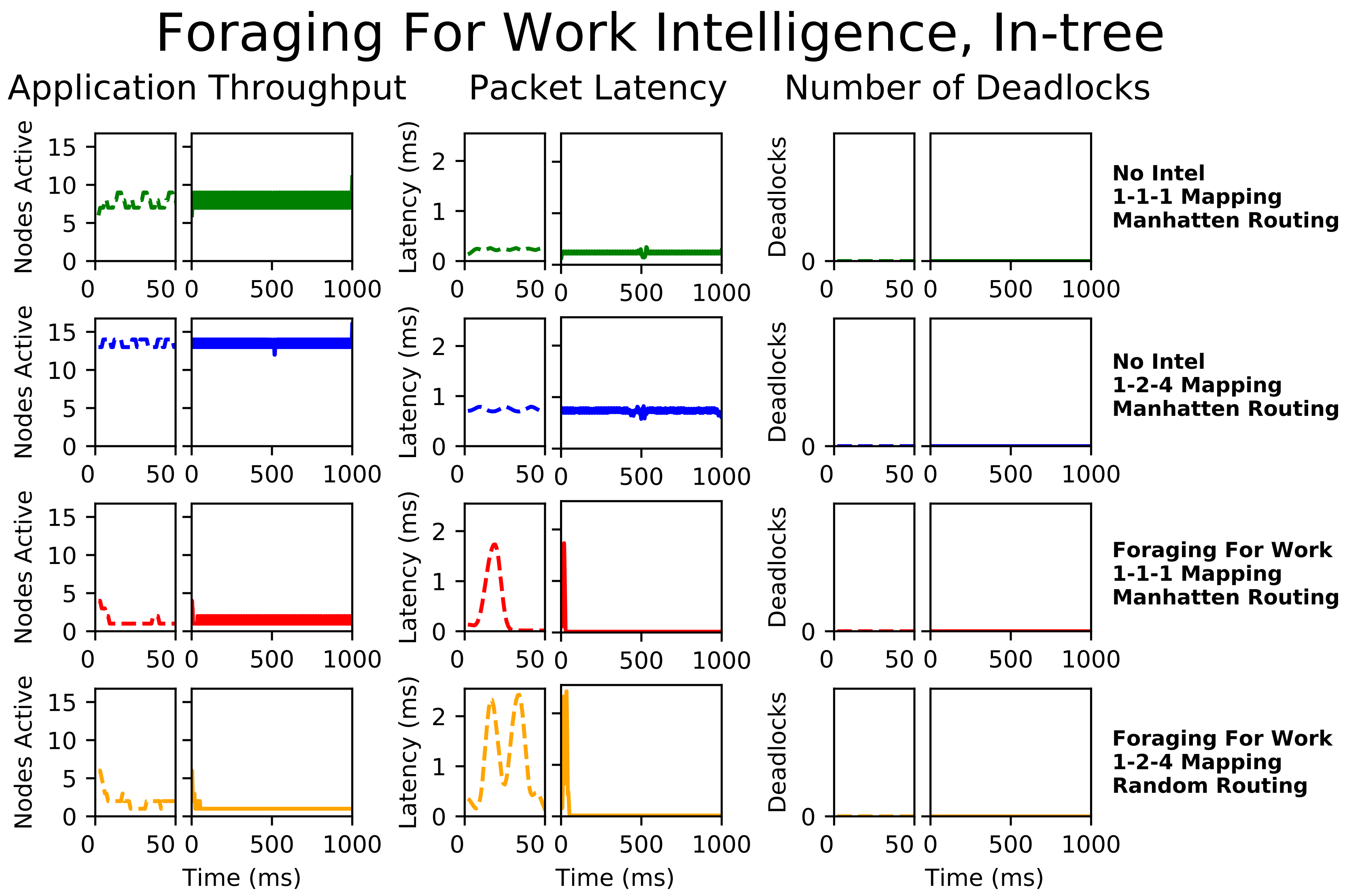


Figure 6.24: Packet Analysis for Experiment 3.2. The first section of each graph (dashed plot line) is an focus on the first 50ms of the experiment.

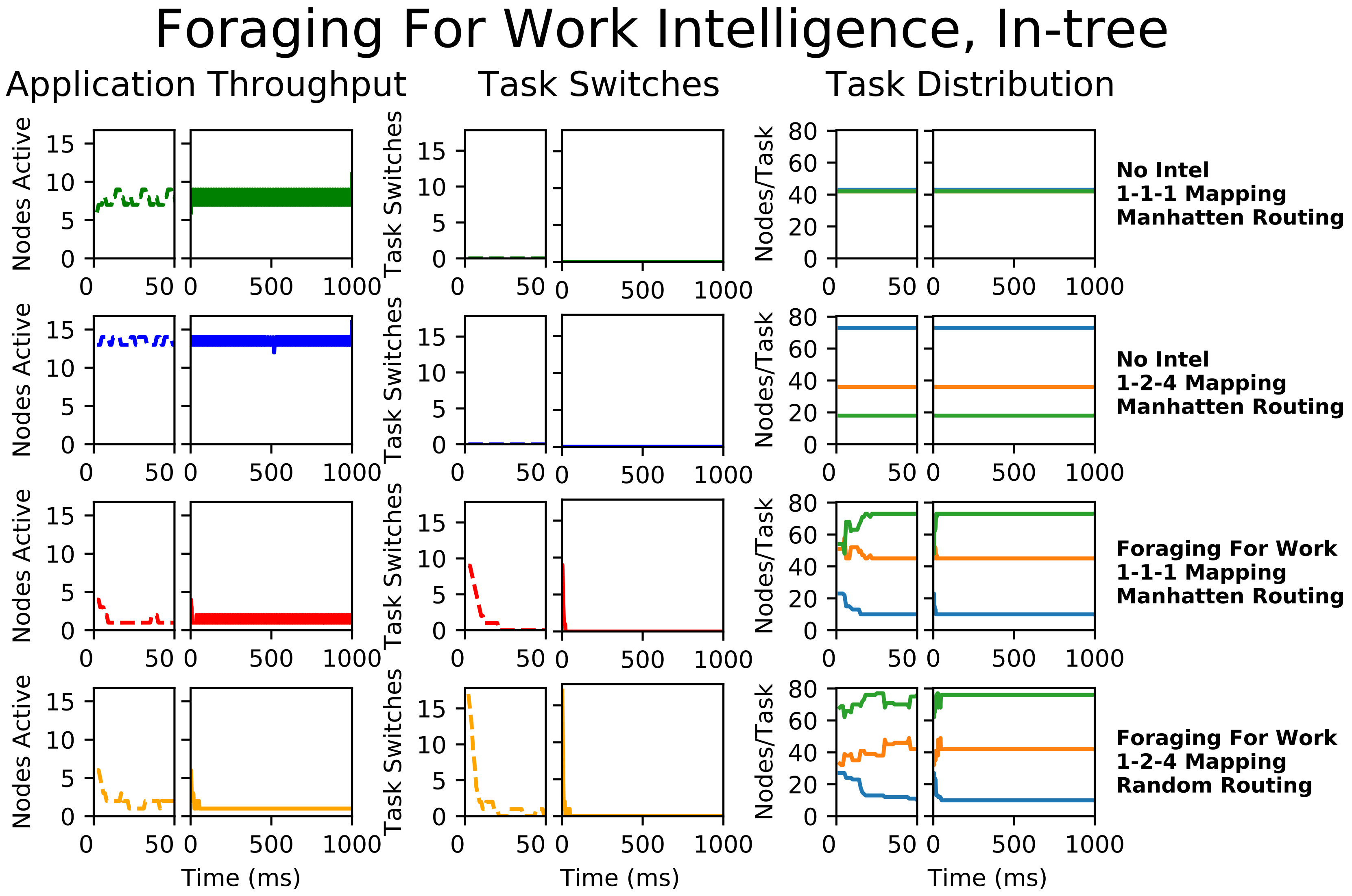


Figure 6.25: Task Switching Analysis for Experiment 3.2 **blue:** Task 1, **orange:** Task 2 **green:** Task 3. The first section of each graph (dashed plot line) is an focus on the first 50ms of the experiment.

6.5.6. Experiment 3.3: Foraging For Work - Out tree

As is seen in Figure 6.26, FFW copes better with the Out-Tree. The starting ratio of tasks has little effect, unlike the non-adaptive case where the hotspots generated by Manhattan routing in the 1:2:4 case reduce the performance relative to the 1:1:1 mapping. The time domain graphs, Figure 6.27 and 6.28, show an interesting effect for all nodes where a "ramping up" of packet latency is observed. As this is also seen in the non-adaptive cases its presence in the FFW cases can be discounted. The settling time during this ramp up can be seen and at less than 100ms can still be considered fast relative to the experiment length of 1000ms. The task distribution for Task 1 nodes drops off at this time as well as seen in all other adaptive experiments.

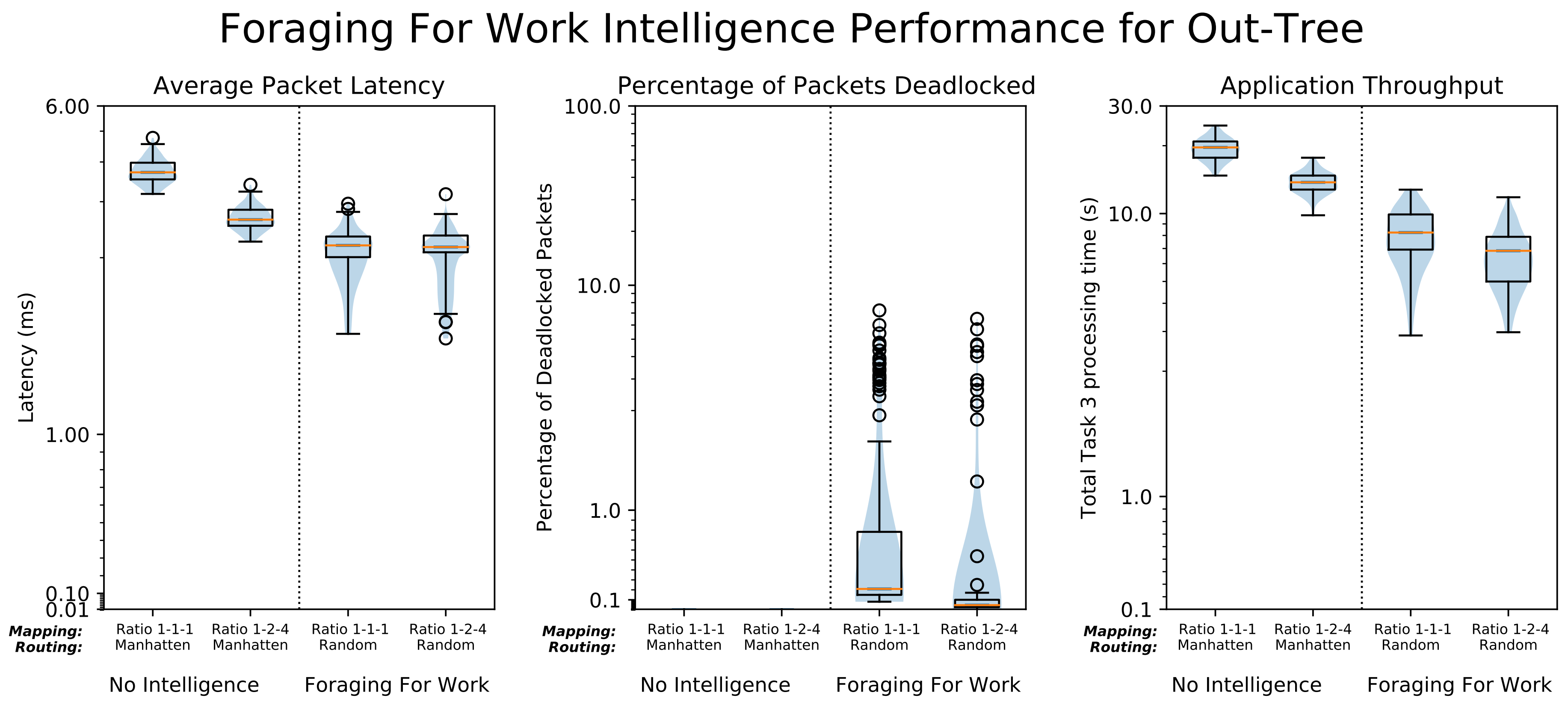


Figure 6.26: Performance metrics for Experiment 3.3

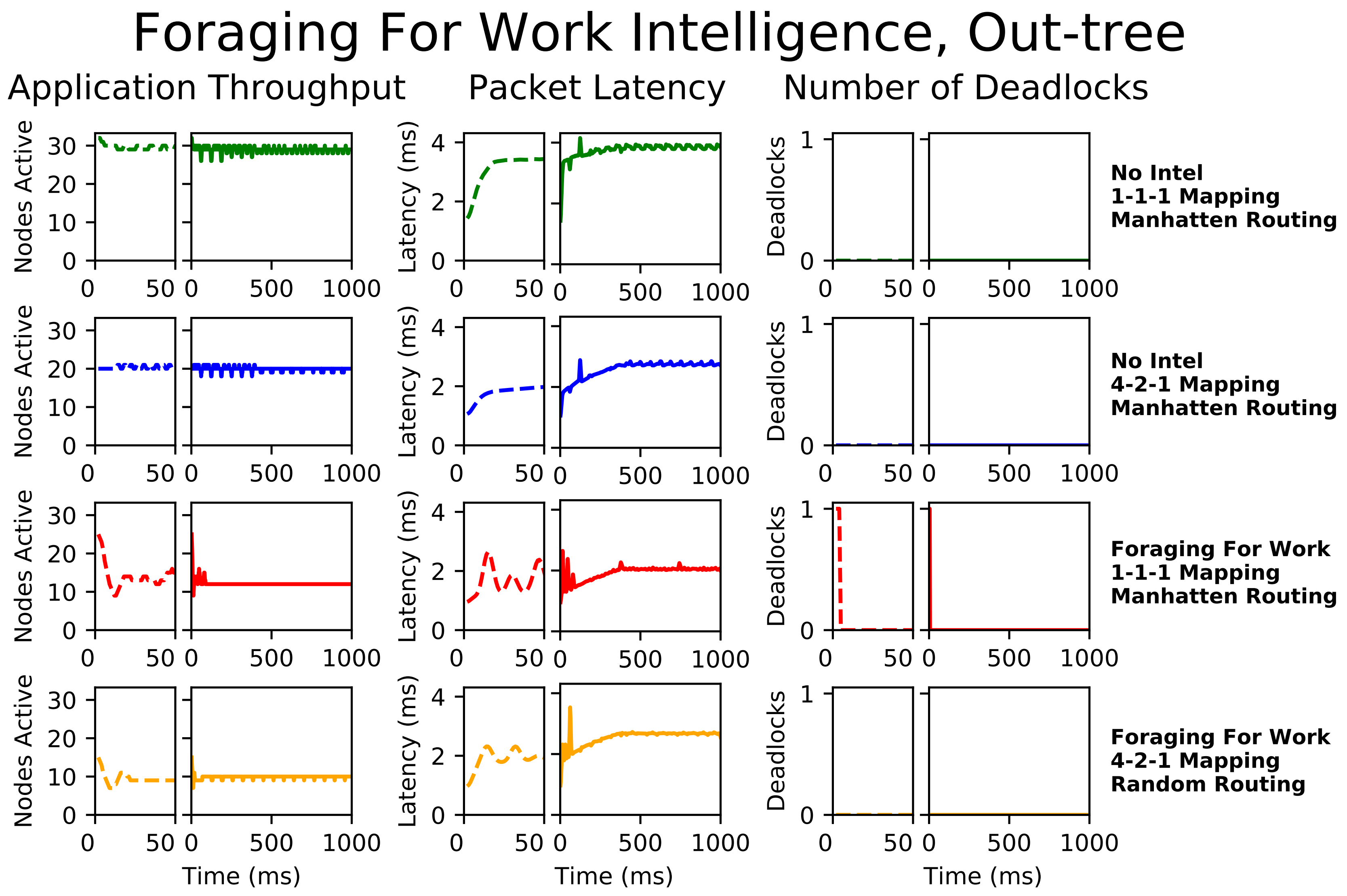


Figure 6.27: Packet Analysis for Experiment 3.3. The first section of each graph (dashed plot line) is an focus on the first 50ms of the experiment.

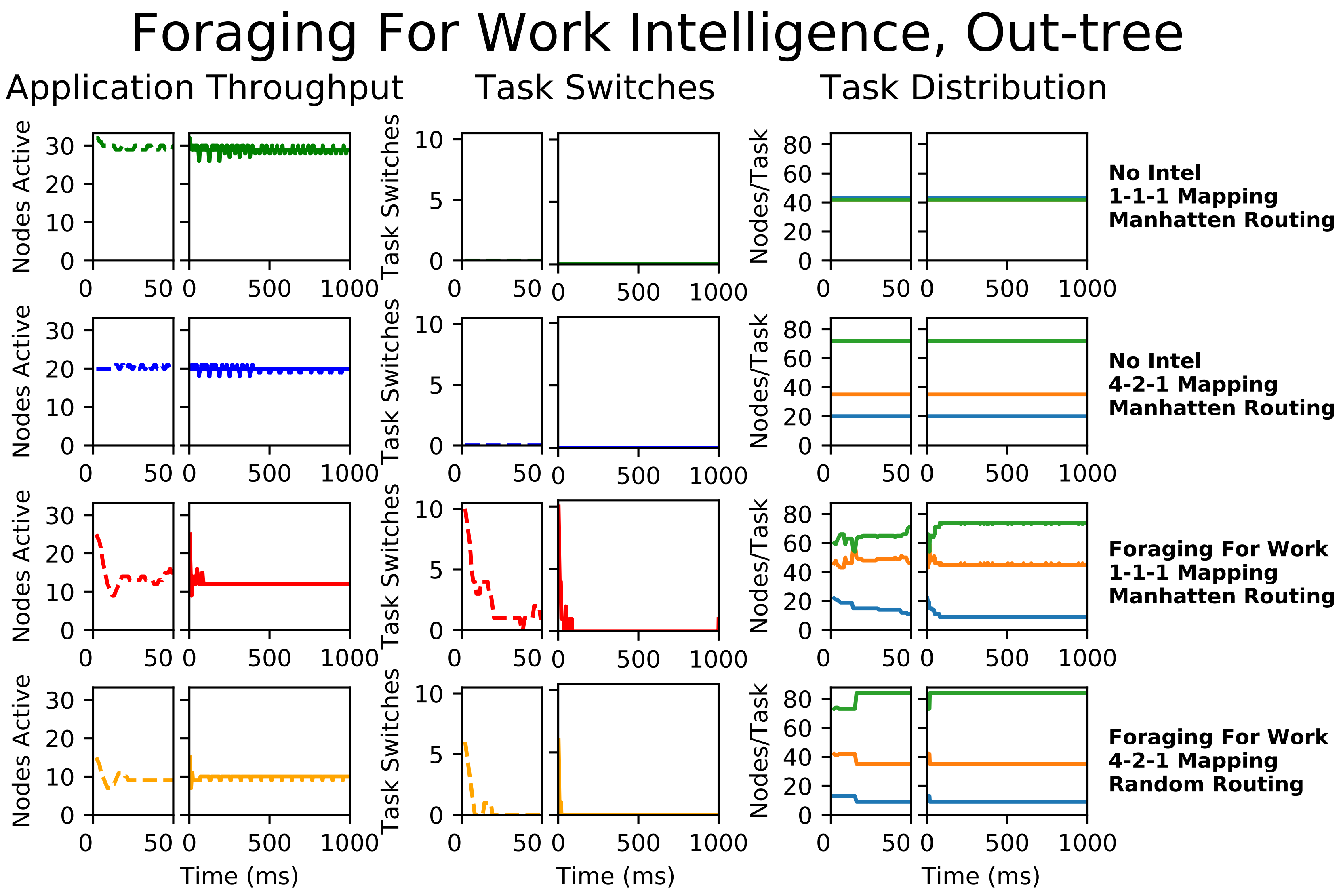


Figure 6.28: Task Switching Analysis for Experiment 3.3 **blue:** Task 1, **orange:** Task 2 **green:** Task 3.

6.5.7. Experiment 3.4: Foraging For Work - Fork Join

Unsurprisingly, Fork-Join sees its performance (Figure 6.29) dominated by the poor performance of the In-tree. Packet latency is improved but a large number of deadlocks cancels out this advantage. The time domain (Figure 6.30 and 6.31) shows that these deadlocks are removed quickly but, as expected, at the cost of loosing Task 1 nodes and therefore application throughput. In the 1:3:1 adaptive case a small oscillation is present on the task distribution, which is interesting as the ratio of Task 2 and 3 is identical, suggesting that nodes may be “trading tasks” as their foraging for work windows both open up simultaneously.

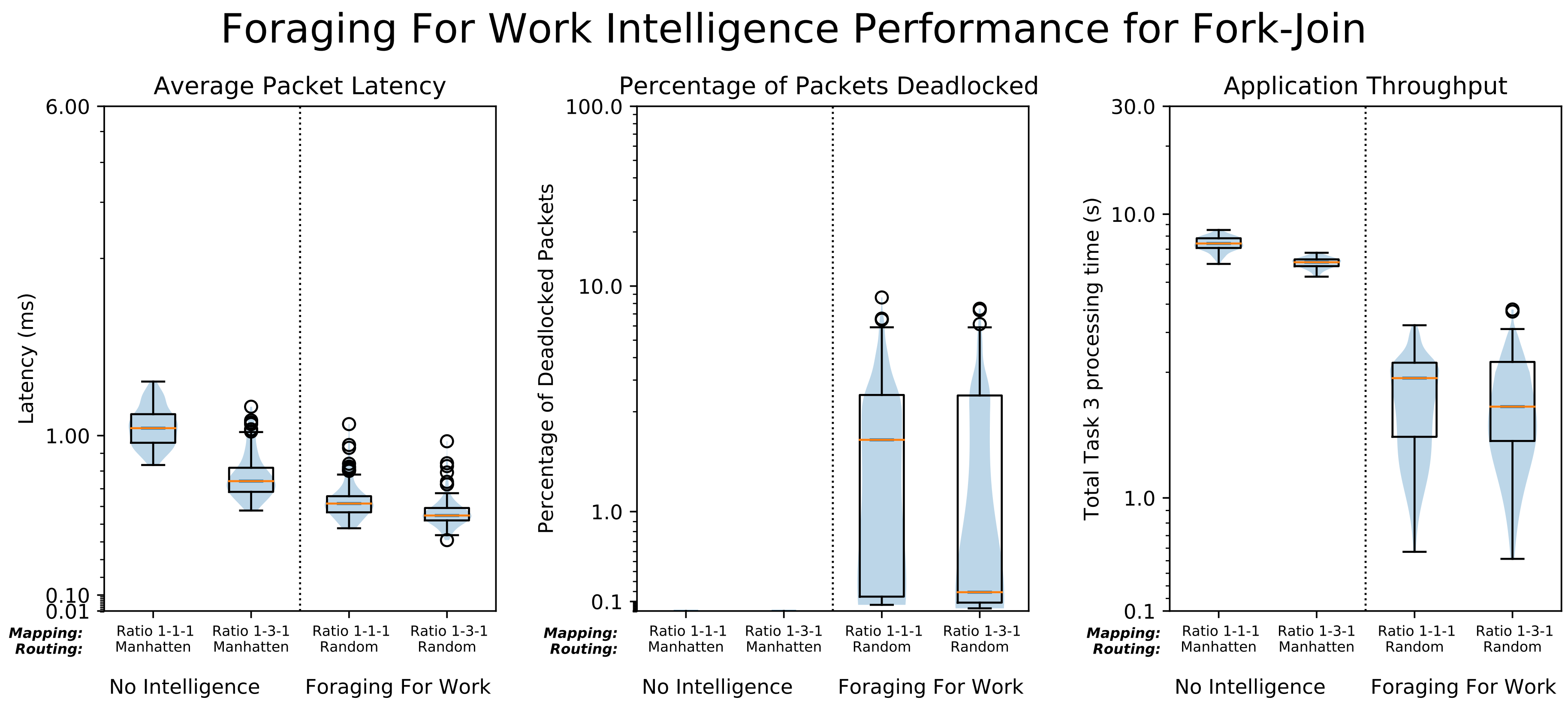


Figure 6.29: Performance metrics for Experiment 3.4

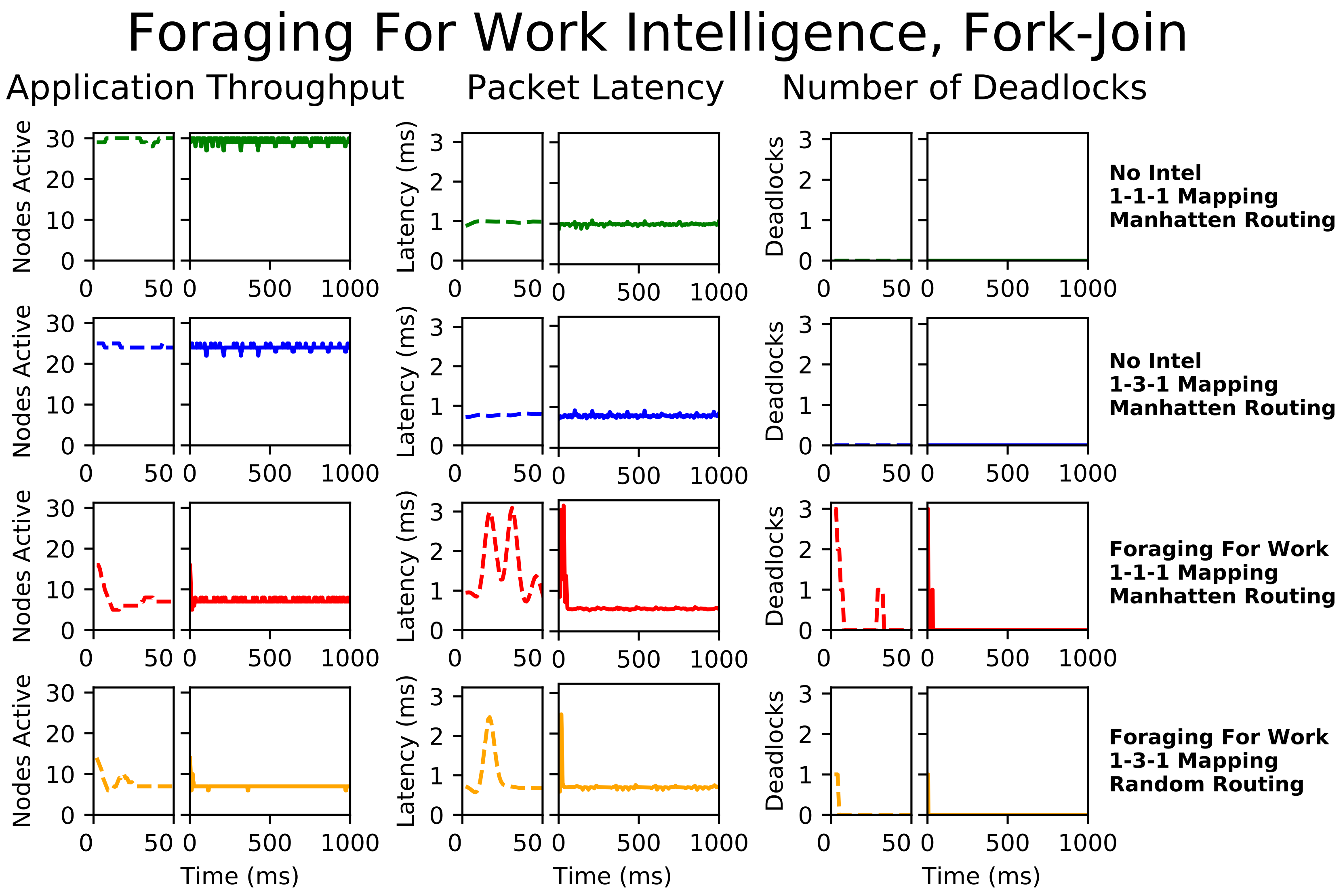


Figure 6.30: Packet Analysis for Experiment 3.4. The first section of each graph (dashed plot line) is an focus on the first 50ms of the experiment.

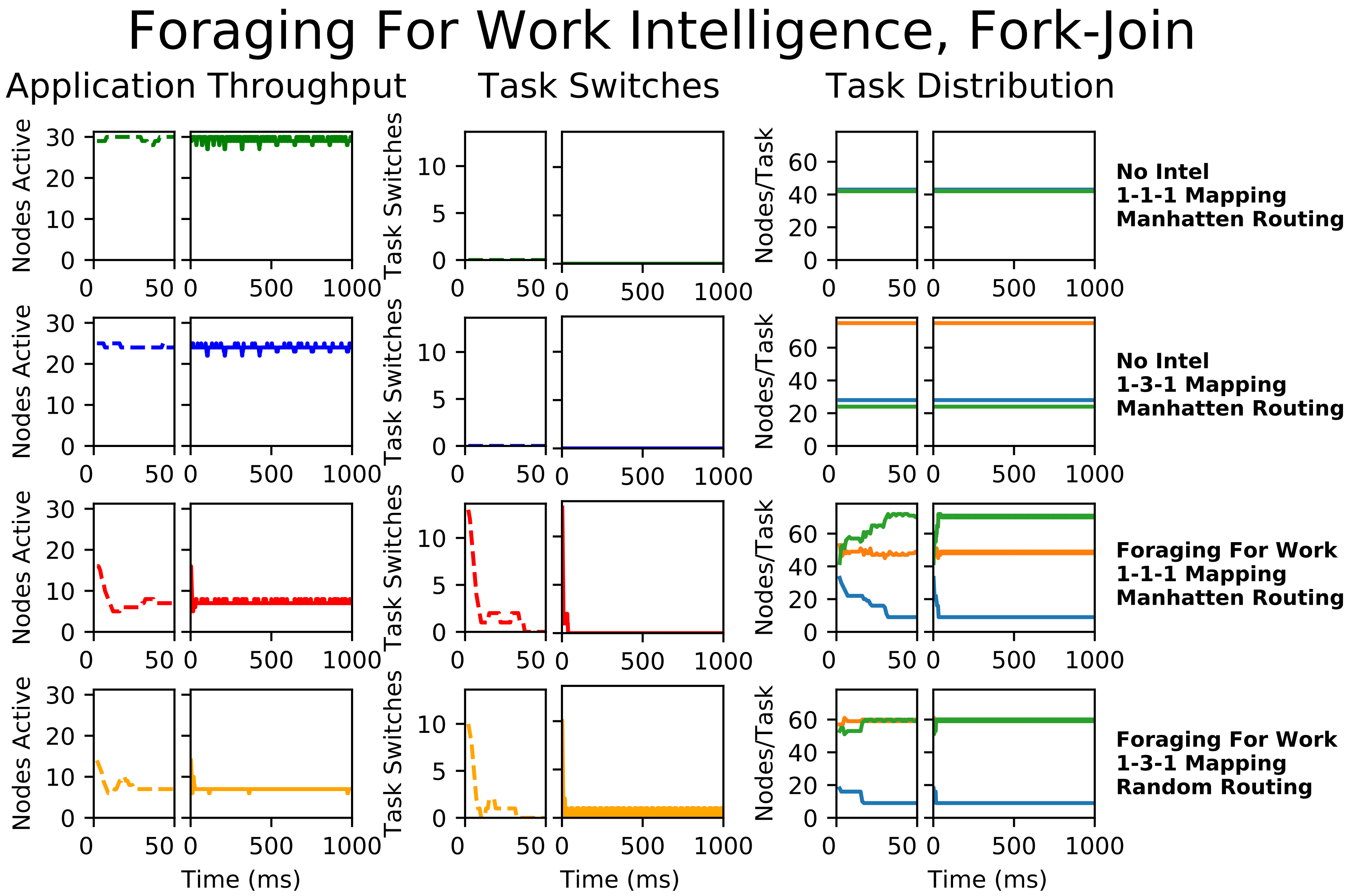


Figure 6.31: Task Switching Analysis for Experiment 3.4 **blue:** Task 1, **orange:** Task 2 **green:** Task 3. The first section of each graph (dashed plot line) is an focus on the first 50ms of the experiment.

6.5.8. Review

These experiments show that Foraging For Work performs very similarly to the interaction model and suffers from the same pitfalls regarding premature settledness coming from reducing the number of Task 1 nodes to a level where the task graph cannot be sustained. This is a clear disadvantage of stimulus based embedded intelligence: the high-level designer must ensure that there are stimuli for all cases in the system. Despite this the performance was not disastrous as in all experiments a small number of Task 1 nodes remained active, although this is likely to be accidental due to the large scale of the many-core and never receiving a packet whilst the foraging for work window is open, rather than a feature of the intelligence model.

6.6. Experiment 4: Self-Regulation of Task 1 Nodes

6.6.1. Biological Inspiration

The removal of Task 1 nodes by both intelligence models is clearly a fundamental limitation of these social-insect models. Whilst some emergent properties are apparent, they are obscured by this issue. This experiment will look at extending these models in the same vein as biologists have done by adding feedback loops into the original task allocation models (i.e. in [25]). Therefore this experiment will add a self-regulation loop that is temporal in a similar fashion to Foraging For Work. If node does not receive a packet within a given amount of time (this can be QoS or application driven) then the task will switch to the producer task. This forces Task 1 nodes back into the system from idle nodes and gives the two models extra stimulus to interact the model dynamics with.

6.6.2. Experiments Overview

This experiment will extend both task graphs with the self-regulation of Task 1 nodes and run them through the same set of application graphs used for the previous experiments. When a node spends 50ms idle then the self-regulation kicks in and the node is switched to Task 1 by the intelligence. Otherwise the task allocation uses exactly the same Picoblaze code as the previous two experiments. The performance metrics for the interaction model and FFW will be compared directly with their self-regulation equivalents.

6.6.3. Intelligence Implementation

The Picoblaze is used to implement these experiments.

Knobs and Monitors

This is the same for the models and each of their previous knobs and monitors. Both models have an addition of:

Monitor: A fixed time reference, set here to 1ms with a threshold set to 50 to gain a self-regulation event every 50ms.

Picoblaze Software

The flowchart in Figure 6.32 shows an abstraction of the assembler code with the existing models attached. Every time the router sees a packet that is the same as the active task for that node then its self-regulation is suppressed by resetting the regulation count to zero. When the timer tick event happens this increases the count of the self-regulation counter. Once this value exceeds its threshold (20ms) then a task switch is ordered and the node switches to Task 1.

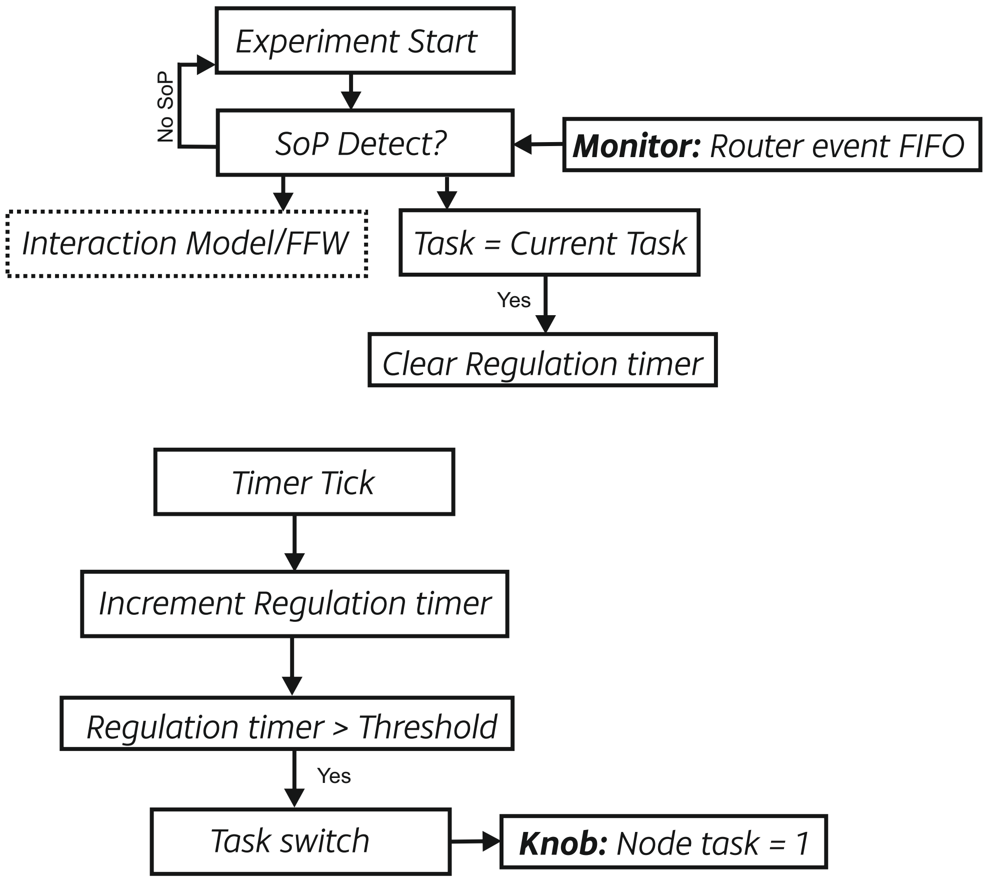


Figure 6.32: Picoblaze software design for implementing the Self-Regulation functionality

6.6.4. Experiment 4.1: Self-Regulation - Linear Task Graph

The linear task graph suffered the least from the lack of Task 1 nodes out of all the approaches. Adding more Task 1 nodes will have a large impact on the network as the task-graph will require a corresponding Task 2 and 3. Therefore if many nodes switch to Task 1 at once it could be expected that the network struggles to route these packets until some other tasks switch to Task 2 and then 3. Therefore a greater settling time is expected.

As the results in Figure 6.33 show, it is clear that the self-regulation has significantly improved the performance of the application throughput, the median Task 3 throughput is around 3 times greater for FFW. It is also clear that the network interaction starts to suffer severely from deadlocked packets, indicating that it cannot adapt to the large changes that the self-regulation introduces fast enough. The time domain results (Figure 6.34) confirm this observation as the number of active nodes drops, the latency increases and the number of deadlocked packets starts to rise at the end of the experiment. Foraging For Work on the other hand has a similar settling time to the non-regulated case and then improves on both the active nodes and the packet latency. The task switch characteristics (Figure 6.35) give more insight. The network interaction model sees an inverse behaviour compared to earlier experiments where the number of Task 1 nodes grows steadily throughout the runtime of the experiment. Foraging For Work on the other hand manages to maintain the 1:1:1 ratio well despite what seems to be quite an unstable network at times with up to 5 task switches per ms happening at times later on in the experiment.

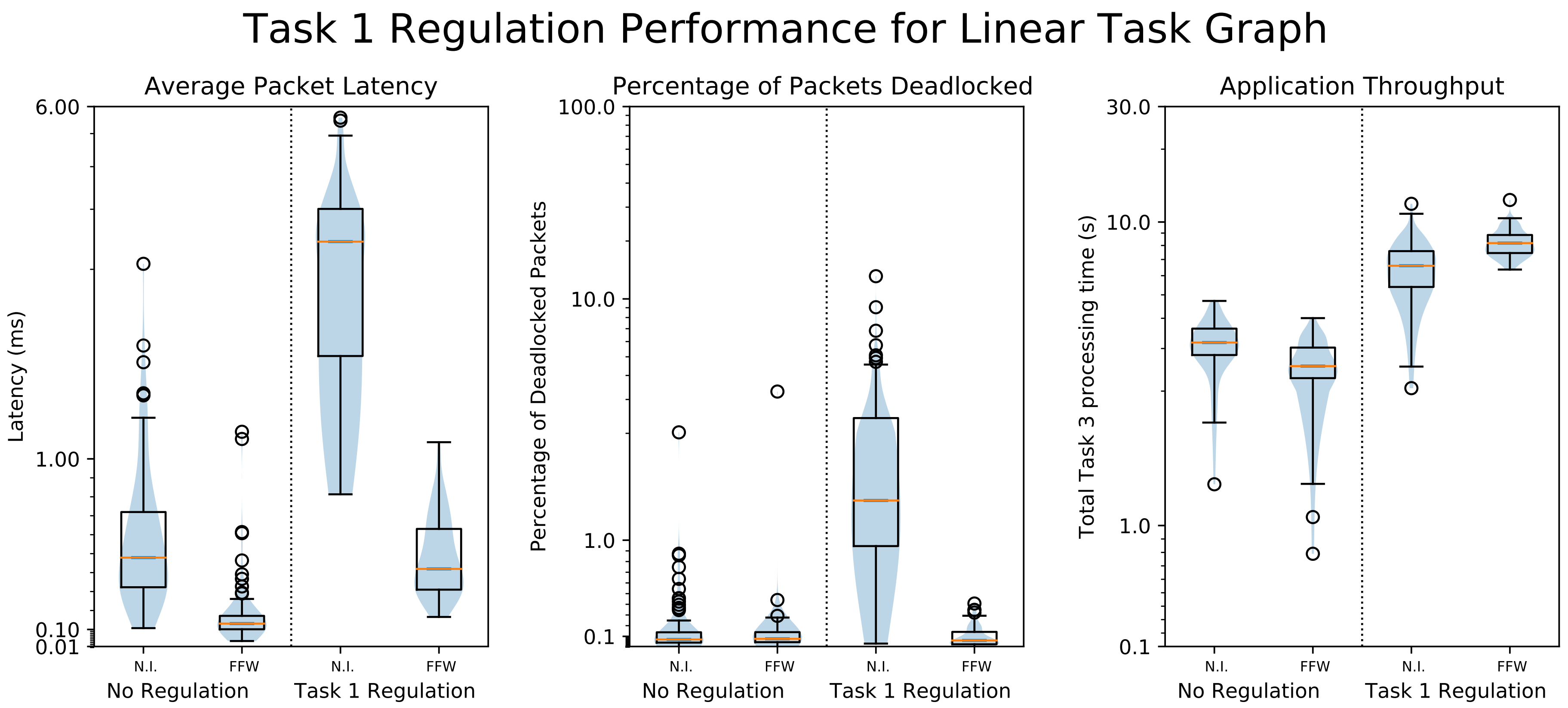


Figure 6.33: Performance metrics for experiment 4.1

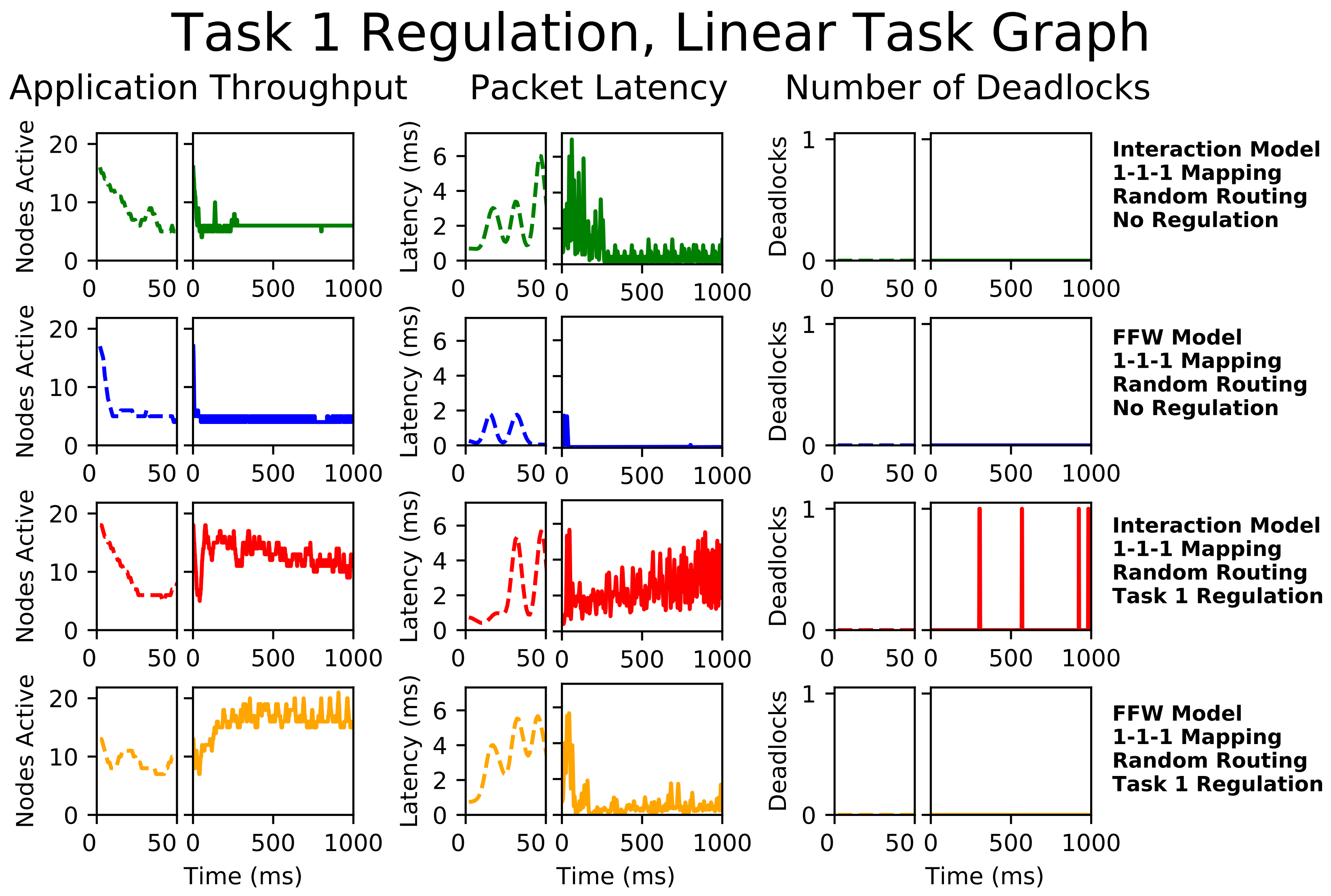


Figure 6.34: Packet Analysis for Experiment 4.1. The first section of each graph (dashed plot line) is an focus on the first 50ms of the experiment.

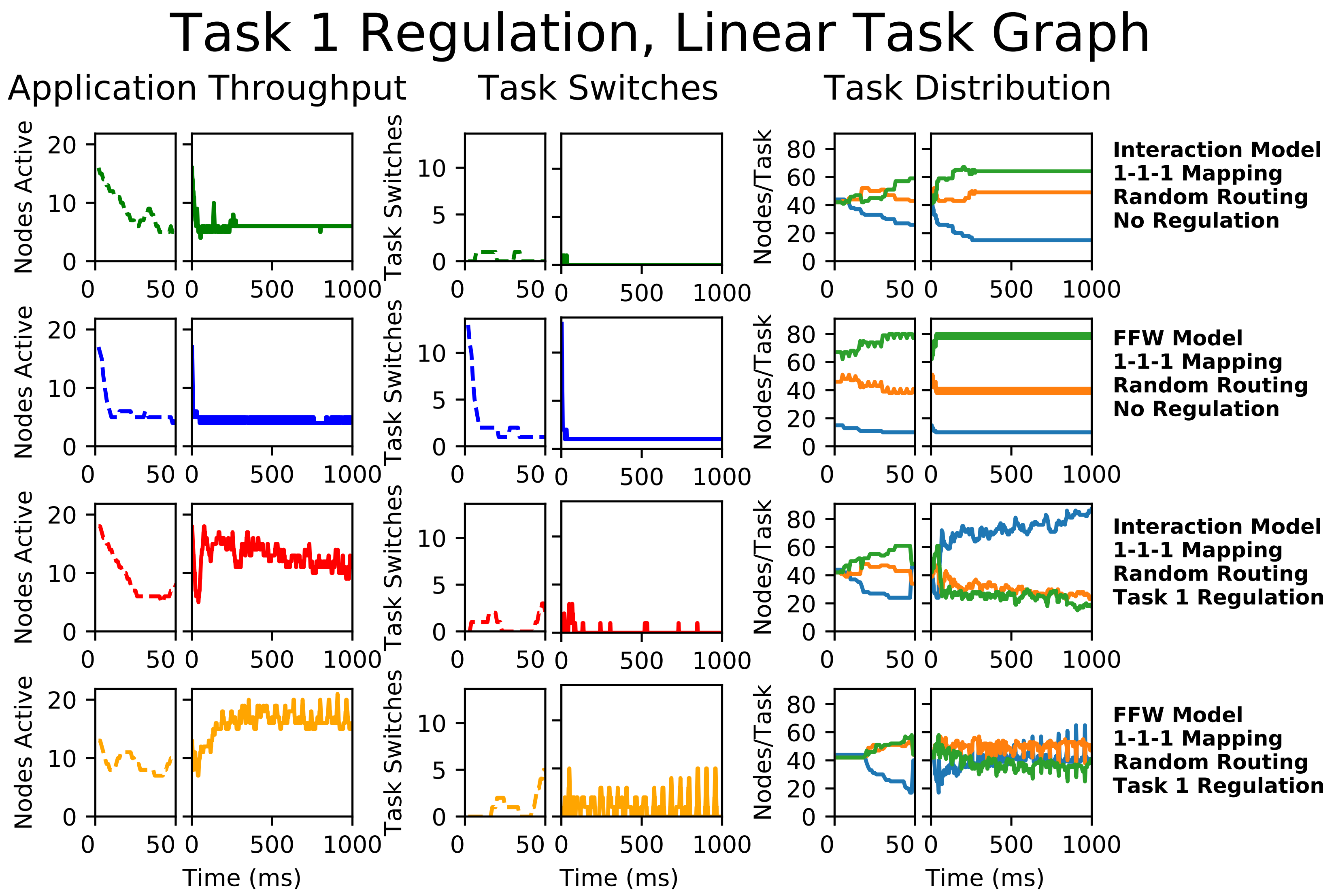


Figure 6.35: Task Switching Analysis for Experiment 4.1 **blue:** Task 1, **orange:** Task 2 **green:** Task 3. The first section of each graph (dashed plot line) is an focus on the first 50ms of the experiment.

6.6.5. Experiment 4.2: Self-Regulation - In tree

The In-Tree performance, Figure 6.36, shows a similar performance distribution to the linear case. A large number of deadlocked packets are present with the network interaction case leading to a large spread in application throughput. The time domain figures (Figure 6.37 and Figure 6.38) at first seem to confirm this is the same as the linear case, but in reality the throughput (number of active nodes) is much lower for both cases and the rise in latency and deadlocks for the network interaction model as the experiment progresses is not seen. The task distribution suggests why this may be the case for the network interaction model: a large number of Task 1s is generated once again but the corresponding number of Task 2 nodes is not the correct ratio when compared to Task 3 (there should be twice as many). A similar issue is seen with Foraging For Work where the ratio seems to want to trend to 1:1:1 but in reality still has an excess of Task 2 nodes compared to Task 1 nodes.

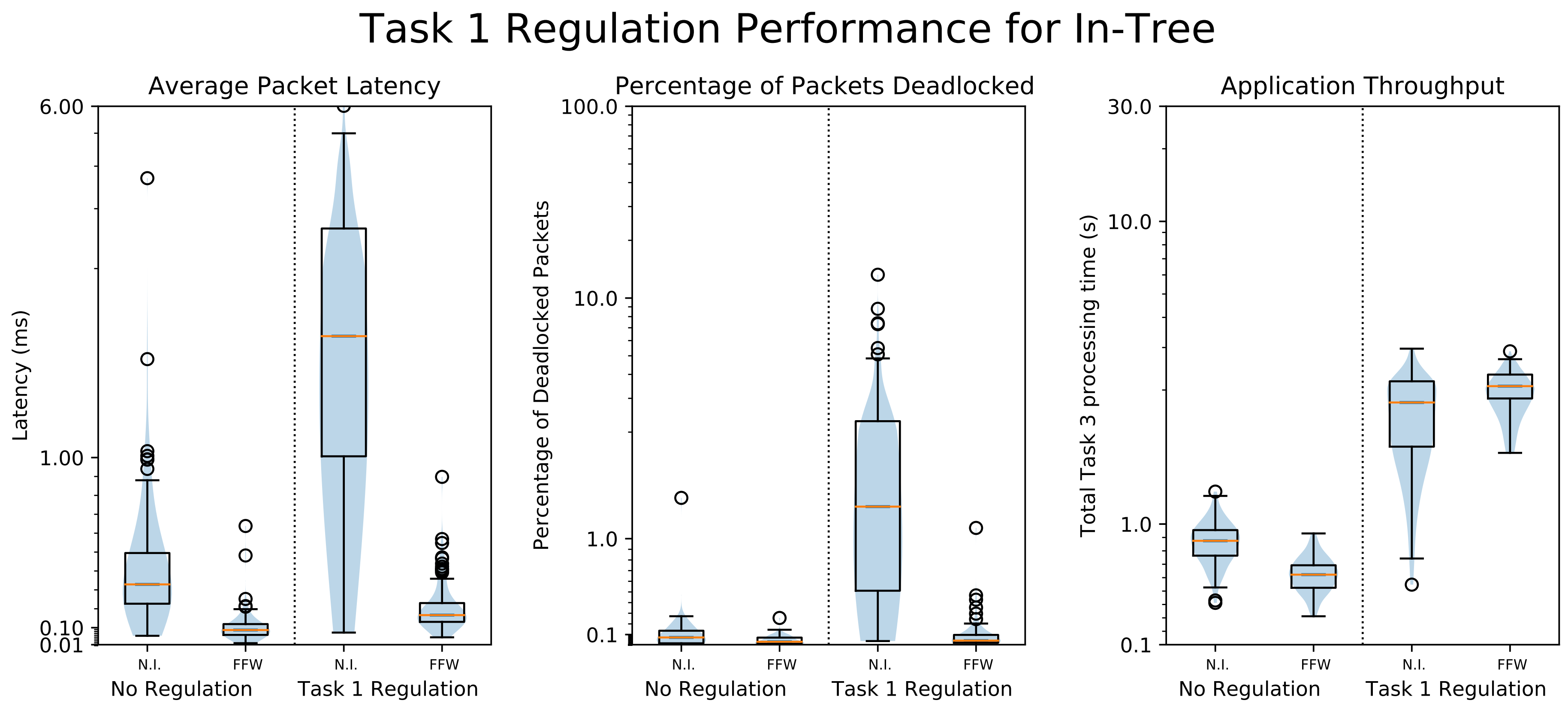


Figure 6.36: Performance metrics for Experiment 4.2

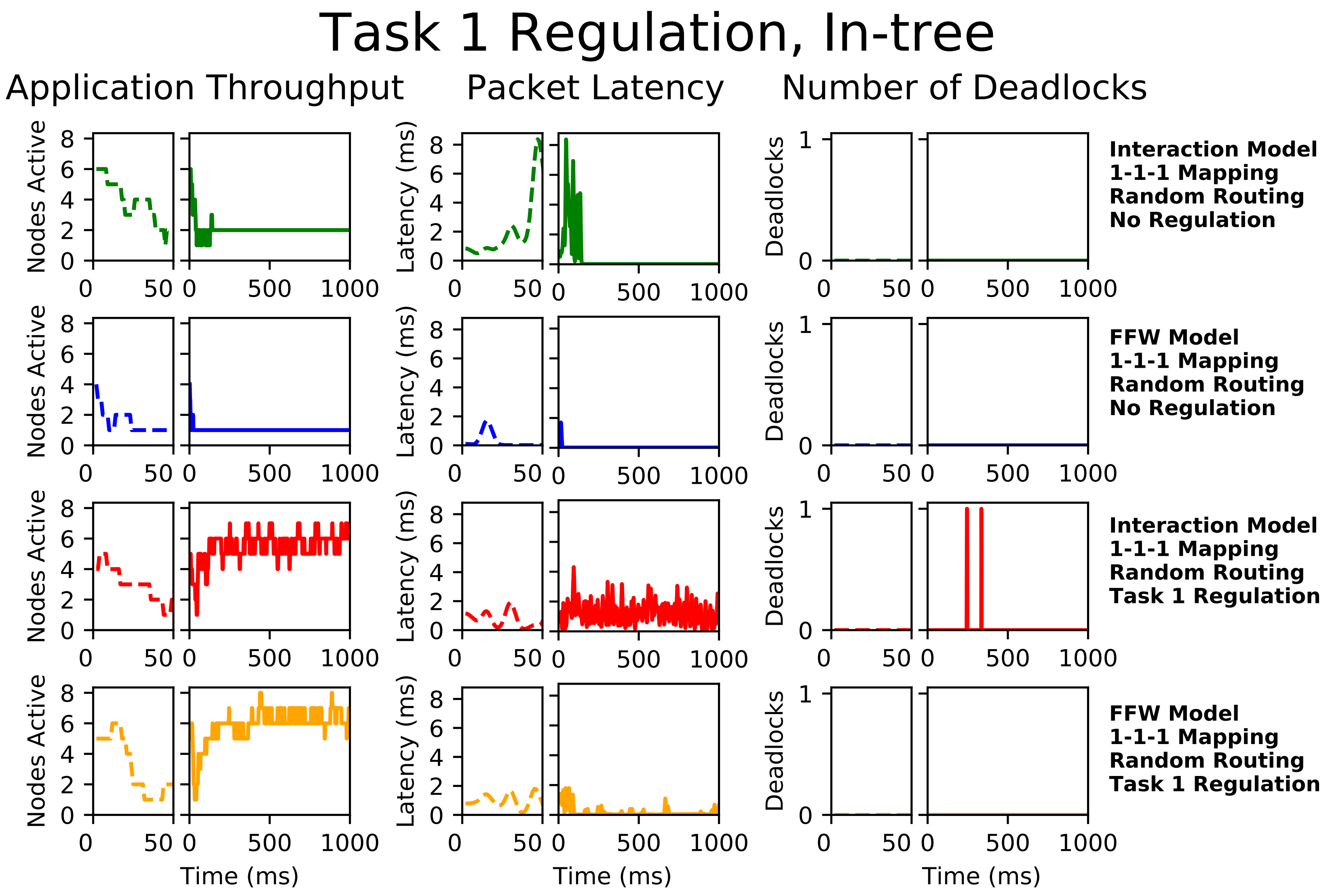


Figure 6.37: Packet Analysis for Experiment 4.2. The first section of each graph (dashed plot line) is an focus on the first 50ms of the experiment.

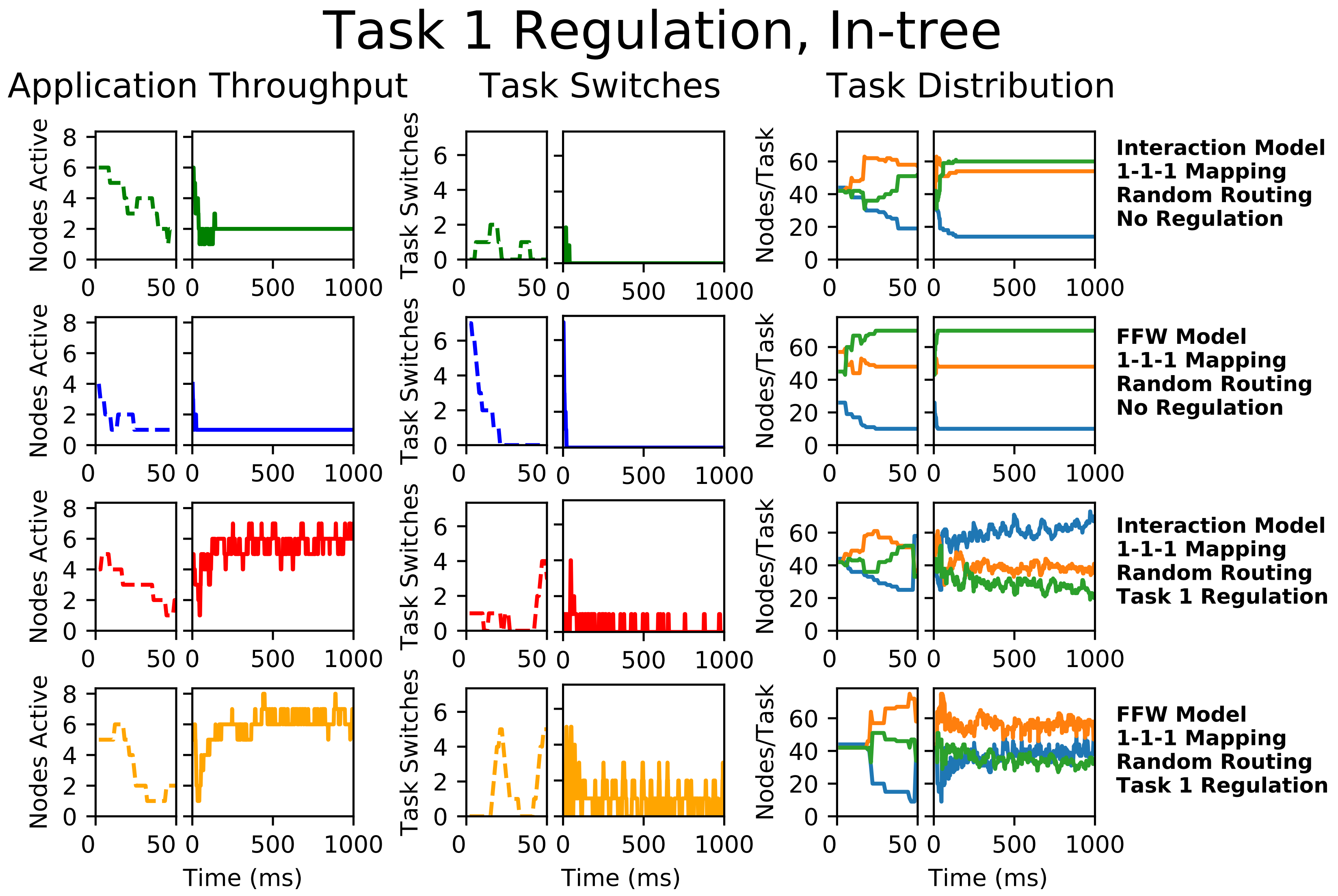


Figure 6.38: Task Switching Analysis for Experiment 4.2 **blue:** Task 1, **orange:** Task 2 **green:** Task 3. The first section of each graph (dashed plot line) is an focus on the first 50ms of the experiment.

6.6.6. Experiment 4.3: Self-Regulation - Out tree

The out-tree performance metrics shown in Figure 6.39 show a clear preference for Foraging for Work. Network interaction still suffers from high deadlock but whilst Foraging For Work does in the non self-regulating case, it manages to overcome this when self-regulation is enabled. The time domain analysis in Figure 6.40 and Figure 6.41 show a similar pattern to the linear case but with the interaction model becoming swamped with Task 1 nodes in less time than in the linear case. Foraging For Work settles to a lower number of task switches (2 per ms) but still emerges a 1:1:1 ratio. Once again this is probably due to a lack of parallel routing allowing the task allocation to reach its full potential with adapting to the task graph.

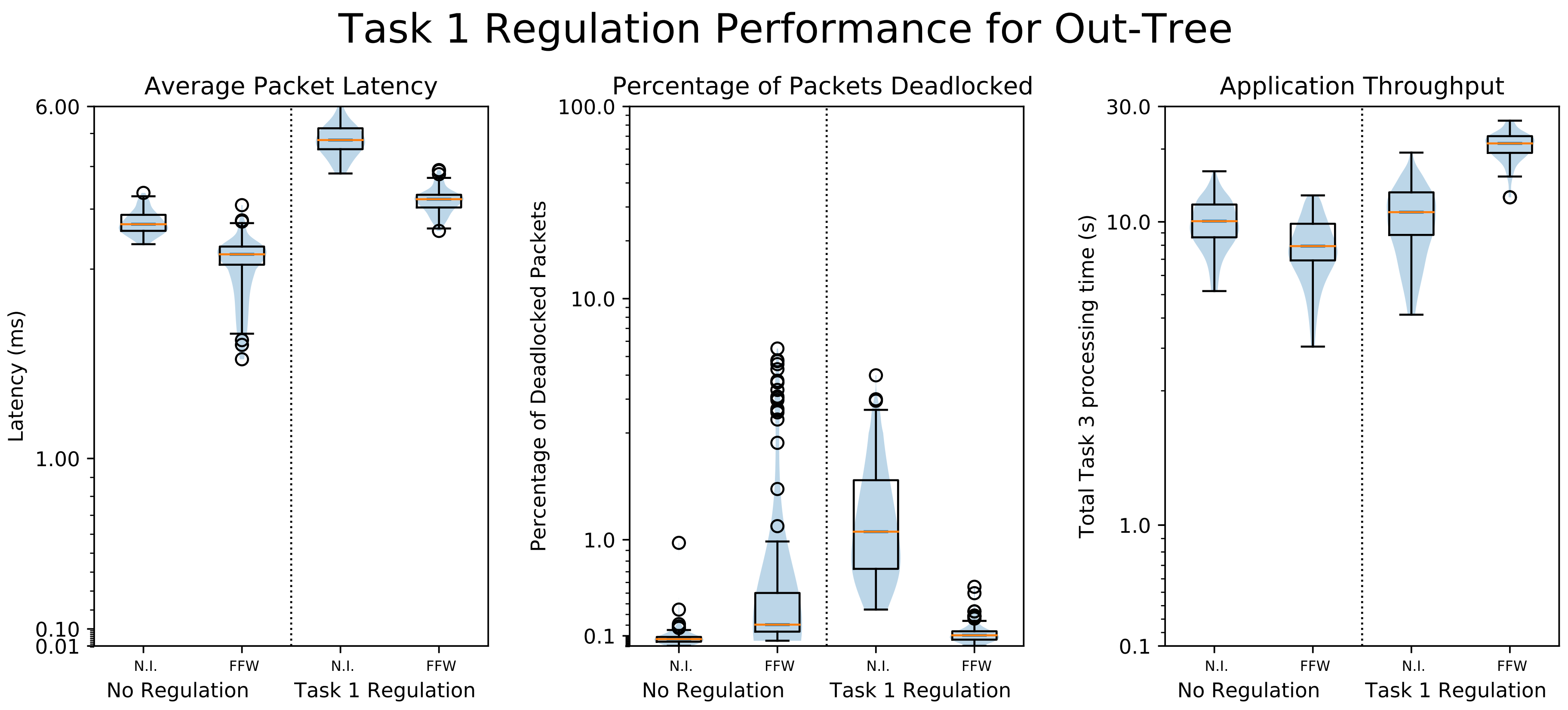


Figure 6.39: Performance metrics for Experiment 4.3

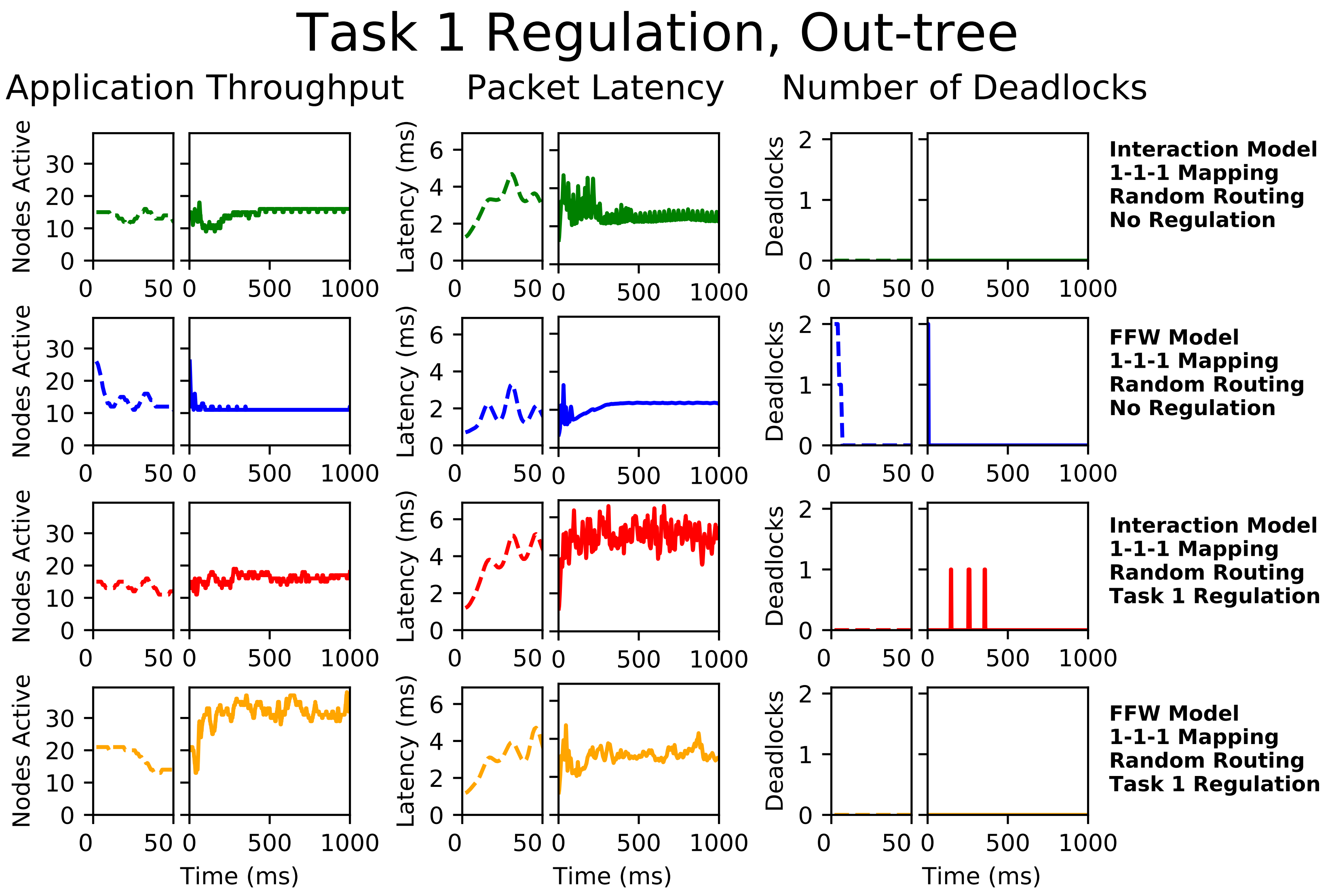


Figure 6.40: Packet Analysis for Experiment 4.3. The first section of each graph (dashed plot line) is an focus on the first 50ms of the experiment.

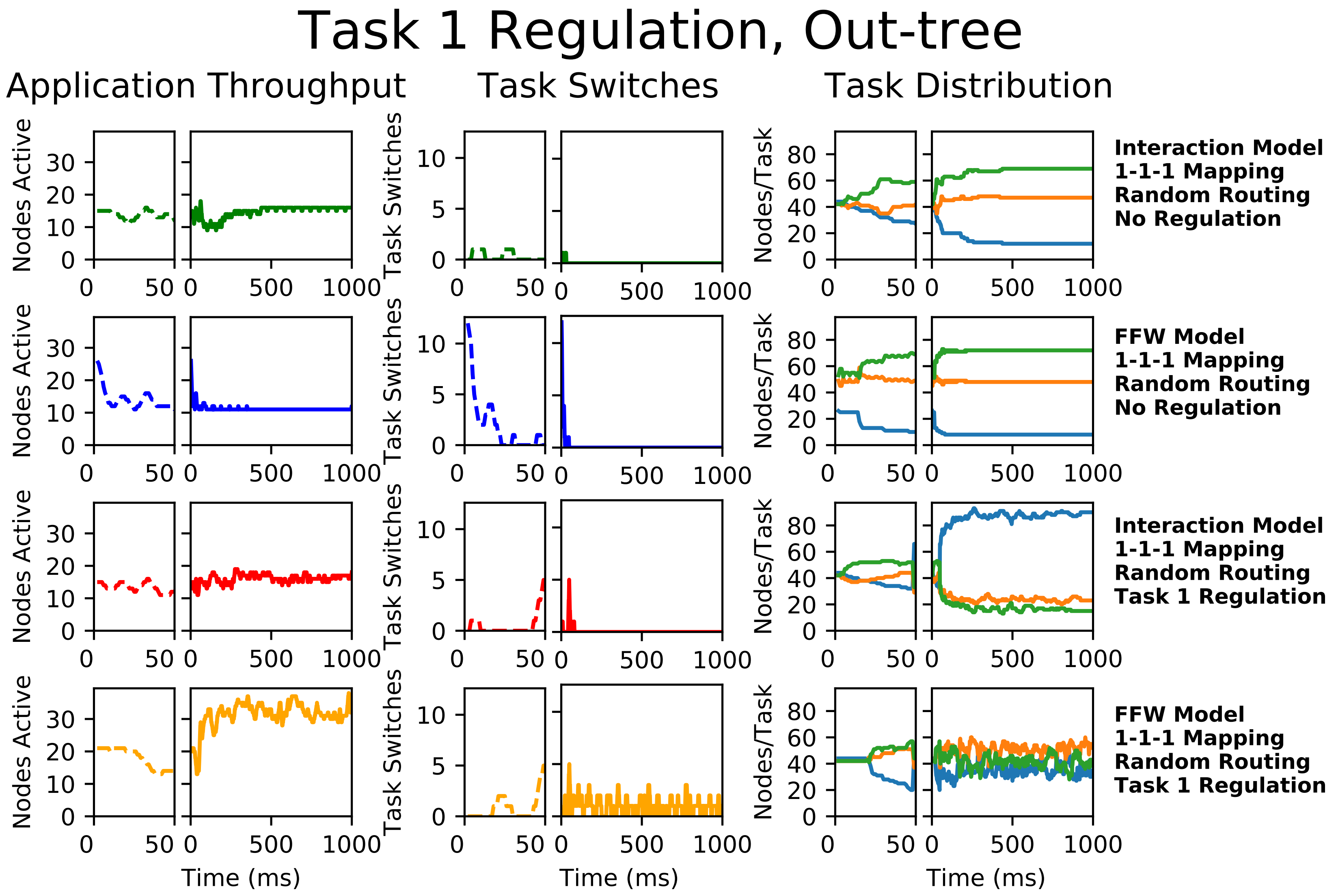


Figure 6.41: Task Switching Analysis for Experiment 4.3 **blue:** Task 1, **orange:** Task 2 **green:** Task 3. The first section of each graph (dashed plot line) is an focus on the first 50ms of the experiment.

6.6.7. Experiment 4.4: Self-Regulation - Fork Join

Given the differences in the In-Tree and Out-tree performance, the performance for Fork Join will be of interest to see if one half of the task graph dominates the other. Figure 6.42 shows a similar distribution to the previous experiments with a larger number of deadlocks for the interaction model than Foraging for Work. The application throughput for Foraging For Work is also better and the network interaction is lacking the spread in its performance that earlier investigations in this experiment showed.

Figures 6.43 and 6.44 show the time domain analysis for the median throughput cases. These results are very favourable for Foraging For Work which, after a long settling time, achieves a high performance and a non-balanced distribution that favours Task 1 and 2 (the Out-Tree) with a lower number of Task 3 nodes (which is desirable being the reduction part of the In-Tree). Network interaction once again sees the network being dominated by Task 1 nodes which increases its packet latency to poor levels by the end of the experiment.

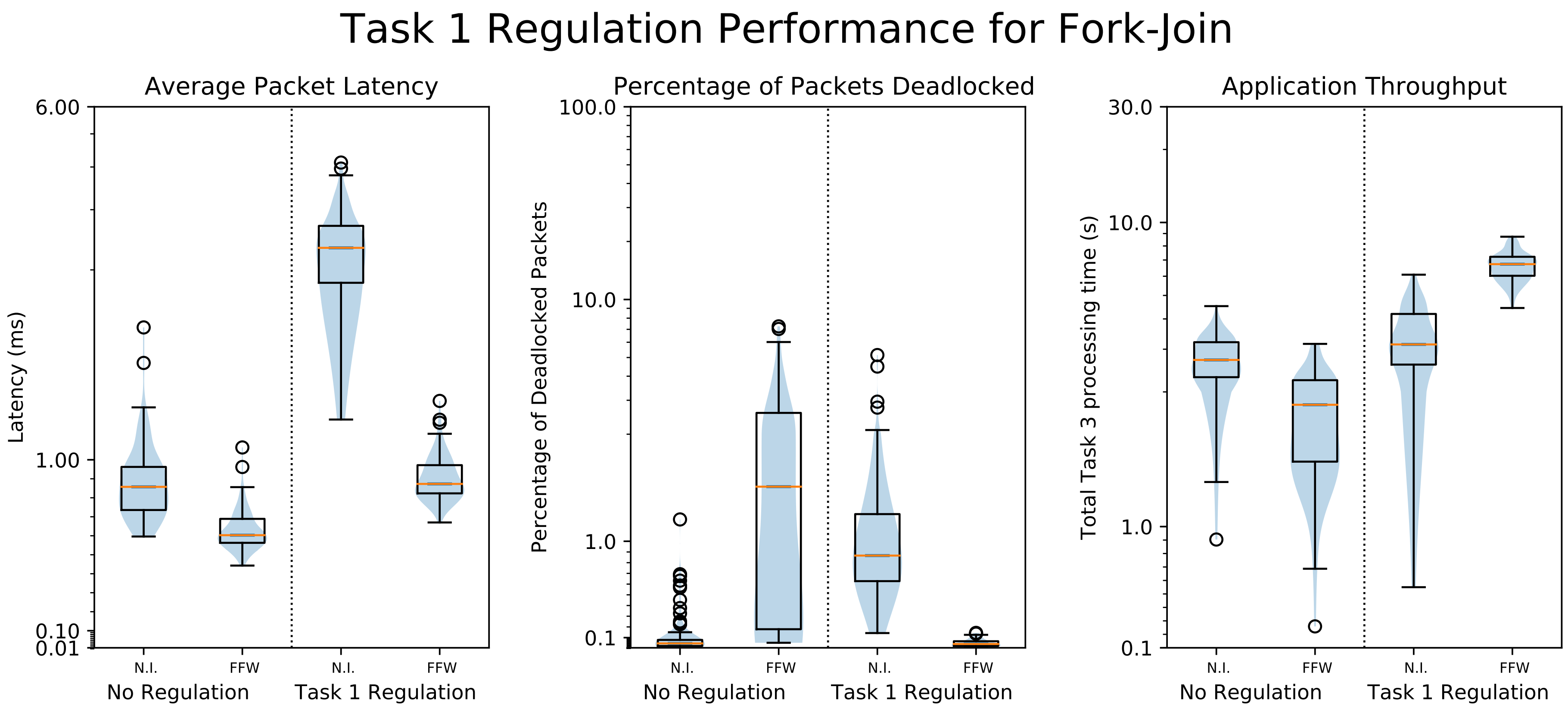


Figure 6.42: Performance metrics for Experiment 4.4

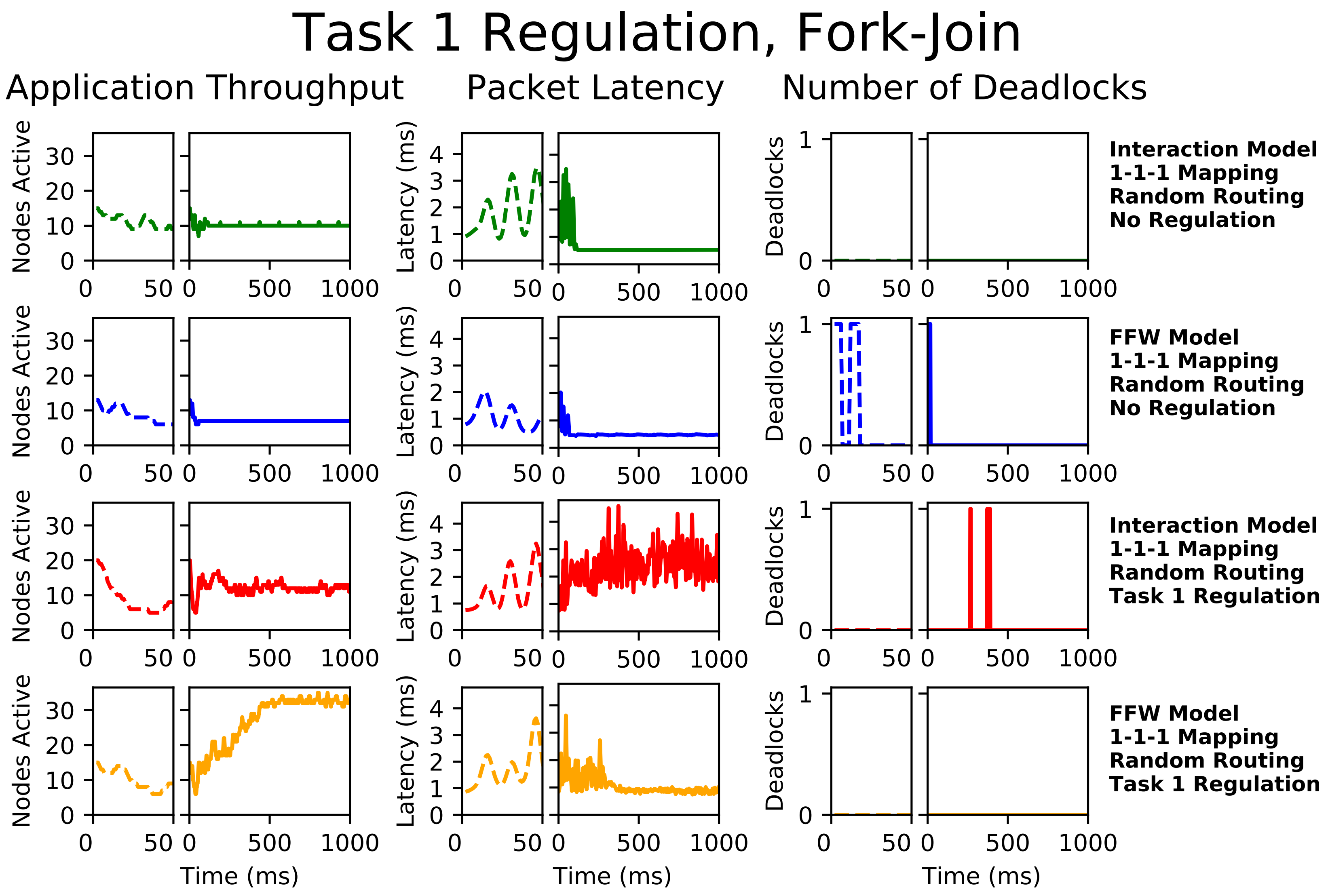


Figure 6.43: Packet Analysis for Experiment 4.4. The first section of each graph (dashed plot line) is an focus on the first 50ms of the experiment.

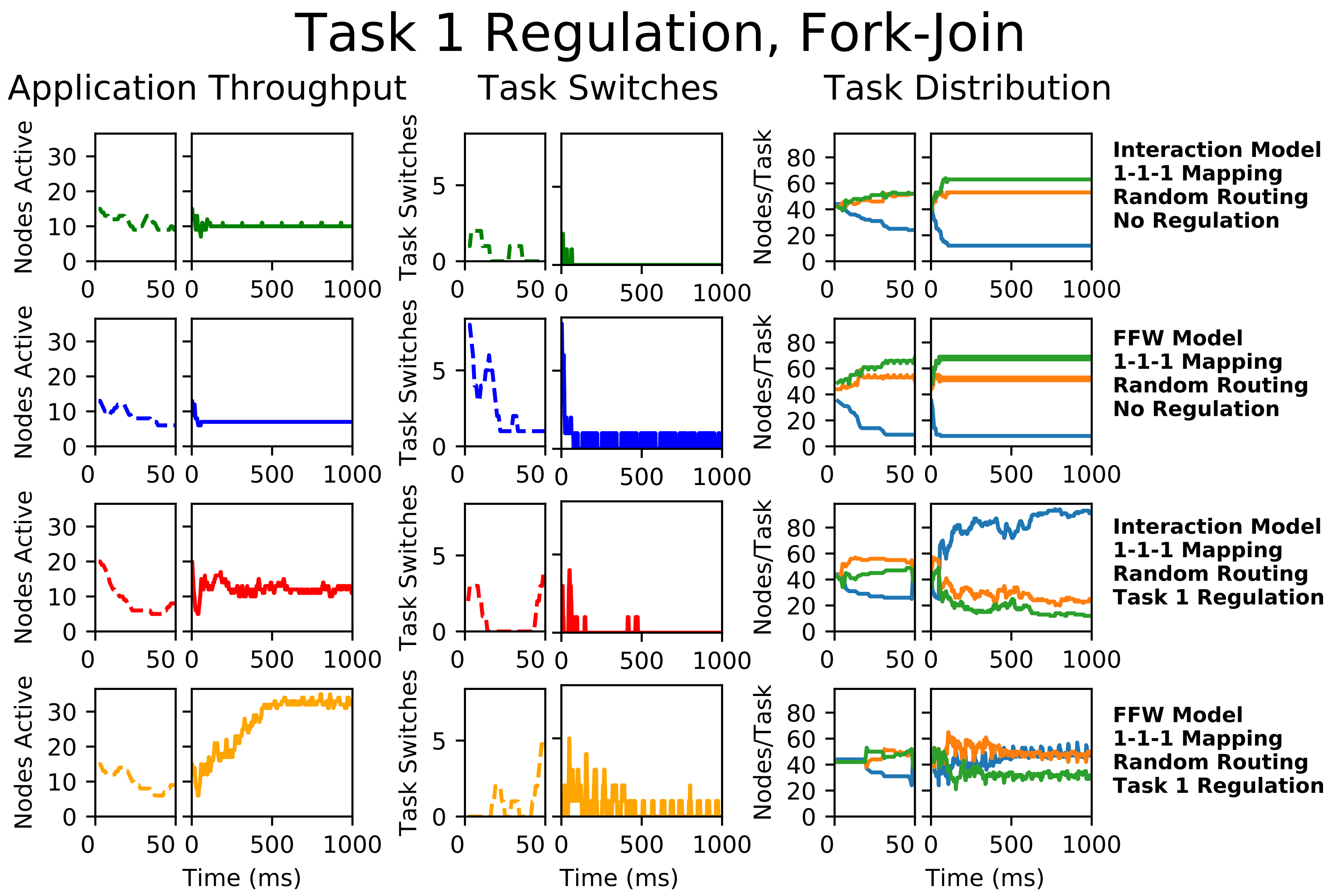


Figure 6.44: Task Switching Analysis for Experiment 4.4 **blue:** Task 1, **orange:** Task 2 **green:** Task 3. The first section of each graph (dashed plot line) is an focus on the first 50ms of the experiment.

6.6.8. Review

It is clear that the self-regulation adds a much needed level of dynamics to the Task 1 aspects of all task graphs. It seems that the effect is too high for the network interaction model. This would be interesting to experiment with by exploring the task switch threshold of the network interaction model and also by modifying the time after which the self-regulation task switch happens. The fact that both the self-regulation and the Foraging For Work are temporally driven could also be a factor in their good compatibility.

The self-regulation in this experiment uses the knowledge that Task 1 nodes are the producer tasks and so an under representation of Task 1 nodes will hamper the achievable performance. These results show that, even without tuning of the existing parameters in both models, this capability can improve total throughput of the system. The increase in deadlocks for the Network Interaction model does show some network stability and so further tuning or parameters or extension of the intelligence model will be required. Indeed this experiment shows that an approach of extracting suitable application properties (i.e. Task 1 nodes being the producer task) to social-insect decision pathways is feasible.

The addition of self-regulation to the models used in biology is to explore and understand the relationship between the environment, individual and nestmates (in [25]). Whilst the weather and the colony needs are obvious factors to be considered for energy-efficient foraging, the model ties them together with very little communication overhead and allows any of the three factors to dominate the decision if needs be; a useful quality when trading difficult and critical factors such as the survival of the colony (at risk from starvation) or danger to the individual (when foraging in poor weather, in a dangerous environment or when food sources are scarce). This experiment has reflected on one aspect of these factors: the need of the application.

To extend this intelligence to the state of the individual then factors such as the node's thermal state, power budget and suitability to the task can be added as factors when deciding to switch to Task 1. The state of the nestmates is inferred from traffic moving through the router in a similar fashion to the Network Interaction model, knowledge about the application could be introduced to help an individual understand what the pattern of network interactions mean.

6.7. Summary

This chapter has shown how two bio-inspired models are translated into a form suitable for hardware realisation using the Picoblaze. It has highlighted some strengths and weaknesses of decentralised adaptive task allocation and routing within a large scale system. It is encouraging to find that the Interaction Network model and the Foraging For Work model can be combined and extended by other stimulus-threshold sensory pathways to change and improve their characteristics. Despite being implemented within a micro-controller, the arithmetic behind the decision pathways are very simple.

It is clear that an adaptive or parallel routing capability would be highly advantageous for spreading workload across the system. The router can be configured to source its routing direction from the Picoblaze and so this could be used to provide a decision that could differ for packets of the same task to provide this variation, at the risk of reducing the stability of the task allocation. A complementary experiment to these experiments with the task allocation adapting to the random routing, could be that the routing adapts to the task allocation. This would not remove the risk of a poor performing task allocation however, so it is likely that experiments combining both adaptive task allocation and adaptive routing would be required.

To qualify the autonomous system attributes of the behaviours seen in this experiment, the resultant behaviours can be considered in line with the goals seen in Section 2.4. The four main *self-** properties seen in this section are used as a criteria to gauge the system autonomy against:
Self-Configuration: In the context of these experiments, self-configuration allows processing to happen as (due to the random topologies) there is the chance that the initial task and routing table allocations do not allow any processing to happen. Stability is also a key factor of self-configuration as, if the system becomes unstable, it could leave a working configuration. Both of the emergent task allocation models exhibit

self-configuration through deciding on the task a node should be performing. The resultant configuration is driven by the random setting of the routing tables, the task graph and also by the dynamics of the task allocation itself as the allocation of tasks to nodes will set the traffic patterns that use the routing tables. Therefore the stability of the configuration is a key criterion of self-configuration that is applicable to emergent task allocation; but may not be of importance in other task allocation schemes. The results of Experiment 4 suggest that there is some performance variation between starting topologies (the distribution of the Application throughput box plots) but the time domain plots suggest that this variation could be bounded as (once the system has settled) it seems that a performance floor can be defined that the system does not drop below.
Self-Optimisation: this ties heavily into the behaviour seen for self-configuration. For the Network Interaction model the self-optimisation is inherent in the way the intelligence will reduce the number of routing hops: a large task packet response is taken up by a node immediately if the stimulus is greater than its current task. This is instead of passing it on to a neighbouring node, resulting in an optimisation where the number of routing hops are reduced. This can be seen in the sharp reduction in deadlocks in the time domain graphs for Network Interaction and the improvement in active nodes as nodes clustered around task 1 nodes switch to task 2 and 3. Foraging for Work only has motivation to optimise once a node becomes inactive enough to trigger the Foraging For Work window where it will accept the next task that arrives. This will have an optimising effect as this packet would have been forwarded on to another node if not accepted by the inactive node. In turn, the original target node may then become less active and so may switch to a more optimal task (again reducing the routing latency).
Self-Healing: Self-healing recovers aspects of the self-configuration in the context of faults or parts of the device becoming derated. As these experiments did not explore these areas, there are no self-healing behaviours exhibited.
Self-Protection: Self-protection alters system parameters to allow the system to ensure that the operating envelope of the device is not exceeded before faults manifest. As these experiments did not explore this area, there are no self-protection behaviours exhibited.

In conclusion, these experiments have shown that both the Network Interaction model and the Foraging for Work model can exhibit self-configuration and self-optimisation behaviours in the realm of task allocation. It can be envisaged that similar decision pathways could also be used for self-configuration and self-optimisation of other NoC aspects such as the routing tables.

Chapter 7 will continue to use the intelligence models from this chapter but apply them to a fault tolerant scenario and also implement the models on the CIA and explore how the emergent behaviours differ to the set-ups used in this chapter.

Chapter 7

Adaptive Many-Core Investigation

7.1. Overview

The previous experimental chapter focussed on the implementation and the properties of the bio-inspired intelligence models. In this chapter these models are tested with fault injection to show that, without any modification to the decision pathways, the models inherently support fault tolerance. The following experiments translate the Picoblaze implementation of the intelligence for implementation on the CIA. The final experiment then demonstrates how the CIA can be used to implement these models with lower power overhead to ensure that the CIA is not contributing to the challenge of Dark Silicon.

7.2. Experiment 5: Fault Tolerance

7.2.1. Biological Inspiration

Both the Network Interaction Model and the Foraging For Work model will support fault tolerance at the individual (i.e. node level) without any additional changes to the model. This experiment will explore this and characterise the response time relative to faults injected into the application. In the network interaction model, removal of individuals of a particular role (for example by a hungry predator at the entrance to the nest removing foragers) will cause a change in the balance of interactions that an individual will encounter. If a hungry individual is suddenly not interacting with many foragers then it is likely to change task to a foraging role, restoring the balance of tasks within the nest. Foraging for work will also exhibit this dynamic, in this example the lack of foragers will create a task stimulus for an individual to undertake foraging (due to individual hunger) and so when the next task selection window arrives, the hunger stimuli may override the stimuli for the task that the individual was doing at the time.

7.2.2. Experiment Overview

In the many-core this translates to node-level faults. When these faults are injected the local task allocation mappings that the intelligence has settled at are upset. This propagates to disrupt the network traffic and so the inputs to the network interaction and FFW models will suddenly change. This will cause the local task allocation to be re-balanced, in a similar fashion to the reaction of the intelligence at the start of an experiment as seen in the previous chapter.

This experiment explores this fault tolerance by injecting node-level faults during the experiment runtime. As seen in Chapter 2, a likely source of faults in future systems is age-induced derating of on-die components, issues driven by device variability and incorrect or inaccurate margins by the design tools to mitigate thermal or power-limitation induced timing faults. This experiment injects a series of faults ranging from 2 faulty nodes to 42 faulty nodes to test the fault tolerance response of the intelligence. Injecting a small number of faults across the running system represents a complex systematic error in the design of the application software or in the EDA tool margin that is bought out by a change in environment or application scenario (e.g. a stack overflow). The cases with larger number of faults (42 nodes, 1/3 of the processors in Centurion) represents a fault in a critical piece of the system's hardware, such as a failure of a global clock buffer or a hardware design error that causes the same fault in several locations at once. Both fault cases fail their respective number of nodes at the same time to allow the impact of the effect to identified in the results easier. In the first fault injection this also represents the notion that the faults are caused by design issues that only surface when a complex set of conditions are met; in the worse case this will affect all nodes that have this design defect at once.

For this experiment the fork-join task graph will be used as this captures properties of both In-Tree and Out-Tree. There are six fault injection scenarios, these correspond to 0, 2, 4, 8, 16, 32 faults injected per run. Each scenario consists of 100 runs which have different initial conditions (random task allocations, random routing tables) but the 100 starting states are consistent across the six scenarios, so the impact of the fault is what differs in each case. For further analysis and visualisation the cases of 5 and 42 faults are also considered as these represent the small series of application faults and the failure of a critical global circuitry respectively. In all runs, all of the faults will be injected halfway through the experiment (at 500ms). This is done by the experiment controller choosing a number of nodes at random, and then distributing a series of debug commands at 500ms (via the node debug interface) to tell the affected nodes to enter the faulty state. Once in the faulty state the nodes set their active task to 0 and no longer send or receive packets. The fact that their task is set to 0 means that the router will then forward any existing packets to them onwards in the network to be consumed by other nodes.

7.2.3. Intelligence Implementation

The Picoblaze is used to implement these experiments. The same intelligence set-up as for Experiment 4 is used (i.e. self-regulation enabled).

7.2.4. Experiment 5.1: Fault Injection

Quantitative results for 100 independent runs of each model without any faults are summarised in Table 7.1 and are used as the baseline for comparison to the faulty cases. Table 7.2 shows quantitative results for different numbers of faults injected. The recovery time after fault injection and the performance achieved after recovery are compared with the pre-fault case. Typical examples are shown in Figure 7.1 where at 500ms a proportion of the nodes develop faults and fail.

The first 500ms of the graphs in Figure 7.1 show the adaptivity of both bio-inspired approaches, *Network Interaction* and *Foraging for Work* in comparison with an implementation using a heuristic fixed routing approach (minimised Manhattan distance). As can be seen from Figure 7.1, both approaches exhibit a settling phase as the network adapts to the initially random task topology. FFW then enters a steady state (settled) phase that is similar to the performance of the heuristic approach. The NI model also settles into a steady state, but does not achieve the same performance as FFW.

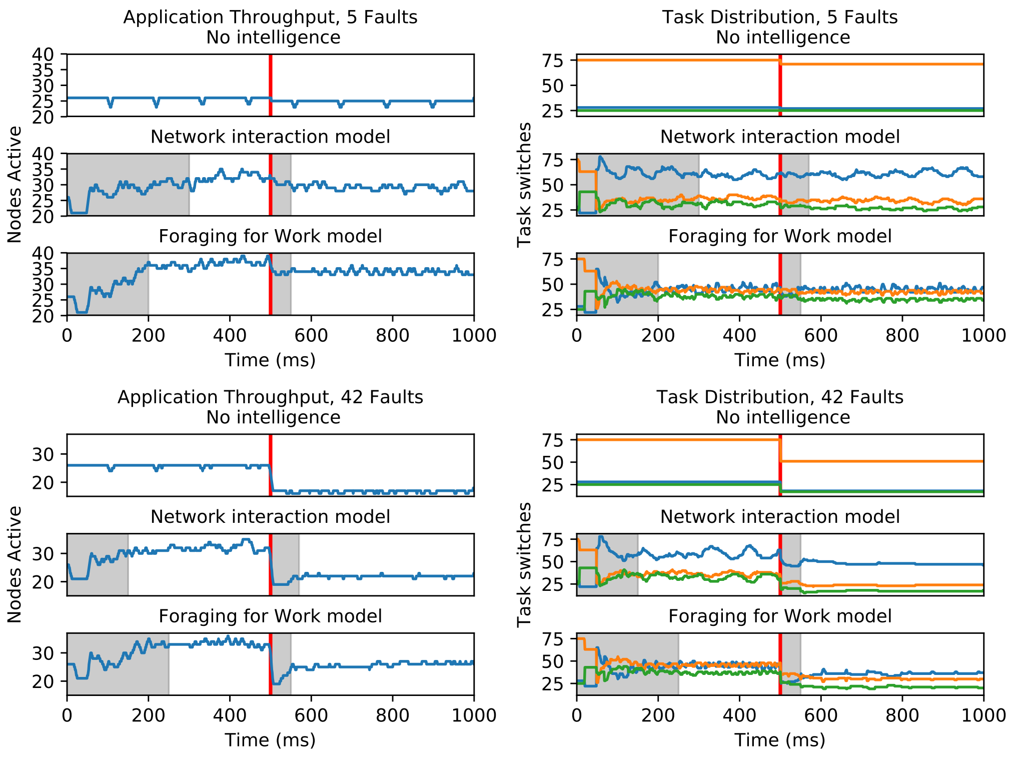


Figure 7.1: Results of fault injection experiments for five faults and 42 faults (1/3 of Centurion). In both experiments the systems were started and then left to self-optimise (the shaded area shows the settling period as the task topology adapts). After 500ms the faults are injected and the system resettles into a new task topology. This recovers some of the performance compared to the pre-fault state by reorganising the task topology to reflect the task graph.

The recovery phase can be seen in Figure 7.1, starting immediately after faults occur as the intelligence adapts to the new task landscape and starts to route around the failed nodes. Once the recovery phase has passed, the system is settled in a steady state where it has recovered to an overall lower performance than before due to the loss of a number of nodes. However, within the limits of reduced resources, performance has recovered and a task structure required for data to effectively reach task 3 nodes has been restored. Again, FFW outperforms the network-interaction model in terms of performance recovery.

Table 7.1: Performance reached—relative to highlighted case—after settling time without fault injection. Shown are median (Q2) and 25th/75th percentiles (Q1/Q3) for 100 independent, randomly initialised runs of each experiment.

	Settling Time			**Relative Performance**		
	Q1	**Q2**	**Q3**	**Q1**	**Q2**	**Q3**
No Intelligence	6	6	7	96%	**100%**	103%
Network Interaction	12	56	58	93%	102%	108%
Foraging For Work	10	86	170	105%	114%	124%

Table 7.2: Performance reached—relative to highlighted case—after recovery time following fault injection at 500ms. Shown are median (Q2) and 25th/75th percentiles (Q1/Q3) for 100 independent, randomly initialised runs of each experiment.

		Recovery Time (ms)			**Relative Performance**		
	Faults	**Q1**	**Q2**	**Q3**	**Q1**	**Q2**	**Q3**
No Intelligence	**0**	–	–	–	96%	**100%**	103%
	2	3	3	19	95%	98%	102%
	4	3	3	17	94%	96%	100%
	8	3	3	5	88%	93%	98%
	16	3	3	3	79%	84%	89%
	32	3	3	3	63%	69%	75%
Network Interaction	**0**	–	–	–	98%	108%	117%
	2	3	30	160	94%	104%	113%
	4	3	30	153	92%	102%	109%
	8	3	19	141	85%	97%	105%
	16	3	3	76	76%	85%	92%
	32	3	3	3	52%	64%	74%
Foraging For Work	**0**	–	–	–	117%	129%	141%
	2	3	29	136	115%	125%	140%
	4	3	36	177	112%	124%	136%
	8	3	53	175	109%	118%	129%
	16	3	81	276	100%	107%	122%
	32	3	3	272	81%	89%	101%

7.2.5. Review

This experiment has shown the node-level fault recovery capabilities of the emergent task allocation. No extra hardware resources are required for the intelligence to automatically adapt to faults in this way. The recovery time in the median case is in the order of tens of milliseconds and a large amount of system performance recovery can be achieved; indeed even with 16 nodes failed the system manages to recover 100% relative to the non-adaptive pre-fault throughput. The recovery is assisted by the nature of the task allocation performance: it has been seen in the previous chapter that most task allocations do not have 100% utilisation of all nodes. Therefore any Task 2 or Task 3 nodes that are not busy will find it easy to switch; especially for the Foraging For Work as the intelligence will be looking to switch as soon as it can if the node is idle.

In general the recovery time will depend on the nature of the application, the threshold settings in the intelligence and the nature of the fault. In this experiment the packet production rate is 4ms. Therefore it can be approximated that the fault recovery time in this experiment is between 6 and 40 times the application packet period (Q2 to Q3, ignoring Q2s of 3ms as this implies the recovered performance was not significant). Given that the Network Interaction threshold is set to 5 SOP tokens and the Foraging For Work expiry threshold is 20ms (5 packet period), these recovery time values fit within the expectations of the response of the intelligence. Regardless of this recovery time, the recovery of performance shows the reorganisation of the task allocation is successful. A real world deployment would need further analysis of the application, fault likelihood analysis of the operating environment and fault recovery requirements would need to be considered to determine how these thresholds should be set.

7.3. Experiment 6: CIA Implementation of the Interaction Network Intelligence Model

7.3.1. Experiment Overview

The previous chapter demonstrated that the Network Interaction model can perform emergent task allocation with the intelligence implemented as response-threshold models implemented in software within the Picoblaze. This experiment will use the CIA to implement this intelligence model to understand if the structure of the CIA decision pathways can also perform the emergent task allocation. This would allow an implementation at much lower overhead that the Picoblaze (memory, sequencer and other microcontroller functions not required) and also allows many decision pathways to be built in parallel.

The use of response-threshold models in the Picoblaze implementation means that the translation to the CIA should be relatively straight-forward, each response-threshold pathway being implemented in a CIU of the CIA. The monitors and knobs, SOP detect and task out, stay the same and interact with the internal router signals in the same fashion. The CIUs can also support the same threshold values. Therefore it is expected that the same high-level behaviour should be emerged as with the Picoblaze implementation.

7.3.2. Knobs and Monitors

A SOP detect is required for each task, implemented using three Match to Impulse monitors, Figure 5.18. Each monitor is configured to match with the SOP header for the task it monitors (i.e. data words `0x181`, `0x182` and `0x183`), generating a impulse when such a header is passed through the router.

One knob is required for this experiment to output the suggested task to the Microblaze MCS. An Impulse to Vector knob, Figure 5.20 is used for this, the output task will reflect the index of the previous impulse issued.

7.3.3. CIA Configuration

The layout of the CIA used for this set of experiments is shown in Figure 7.2. Three CIUs are required, each taking input from one of the task SOP match detectors. Once one of the units reaches its threshold the stimulus for the task it represents is deemed high enough and it issues an impulse. This is sent via the HBus and then received by the output knob which will update the current output task suggestion and also issue an impulse to signify a change in the task state. This impulse is used as an inhibitory input which resets the task stimuli counters, resetting the task allocation process. The CIA is clocked at 25MHz which is 1/4 of the NoC clock speed, this being its maximum speed due to the critical path through the GBus.

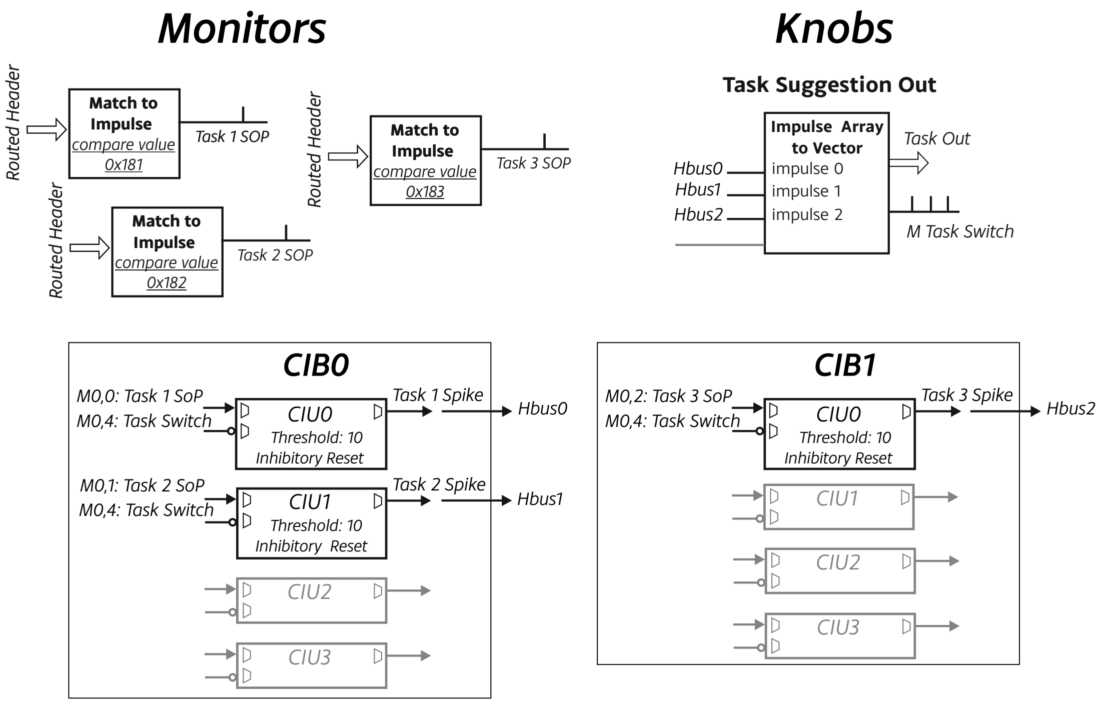


Figure 7.2: The CIA needs to be configured as follows to implement the interaction network intelligence model. The monitors consist of three match-to-impulse detectors that generate an impulse when the SOP header of the relevant task is detected passing through the router. The task suggestion is output by the Impulse-to-Array knob which will latch the value of the last emitted impulse onto the task out vector. It will also generate an output impulse when an impulse is received by the knob. The CIA requires two CIBs and three CIUs to implement the intelligence model, as described in Section 7.3.3.

7.3.4. Experiment 6: Interaction Network Intelligence Model

The fist experiment implements the interaction network intelligence model and compares it various to non-intelligence configurations: *1)* an optimal solution with optimal routing tables, *2)* a random topology with the routing tables preloaded with their direction chosen based on the shortest a Manhattan distance to the node of the correct task, *3)* a random topology with random routing tables. The intelligence is tested in two configurations, the first has a random task mapping and the same Manhattan routing metric as the non-intelligent case and the second has random routing tables and a random task mapping.

Each configuration was loaded into the many-core and the test application run for 1 second, this is then repeated 100 times for each configuration. The random mappings and random routings are consistent between each configuration however, ensuring that each configuration is compared on the same set of random mappings and random routing tables. Figure 7.3 shows the distribution of the average packet latency for each of the configurations.

As expected, the optimal mapping achieves nearly perfect performance with a median of 17 μs and a very tight distribution around this value. This is expected for the short packet hop between optimally mapped nodes with perfect routing paths. The random mapping performs far worse, even with the Manhattan routing applied on the top of the random task mapping. The random mapping combined with the nature of the application means that it is likely that a small percentage of the nodes end up sinking the majority of the packets. This is confirmed in Table 7.3 which shows the average number of unique nodes that processed data during the experiment for each task for each experiment configuration. The load balancing nature of the optimal mapping is clear, but the extra workload sharing of offered by the intelligence could explain why the average latency distributions are less spread for the random mapping with the intelligence model enabled.

Table 7.4 summarises the number of critical deadlocks (i.e. required intervention from the node) encountered by each scheme. As expected the random routing is extremely deadlock prone as there is no coherence in the routing paths. The intelligence offers an

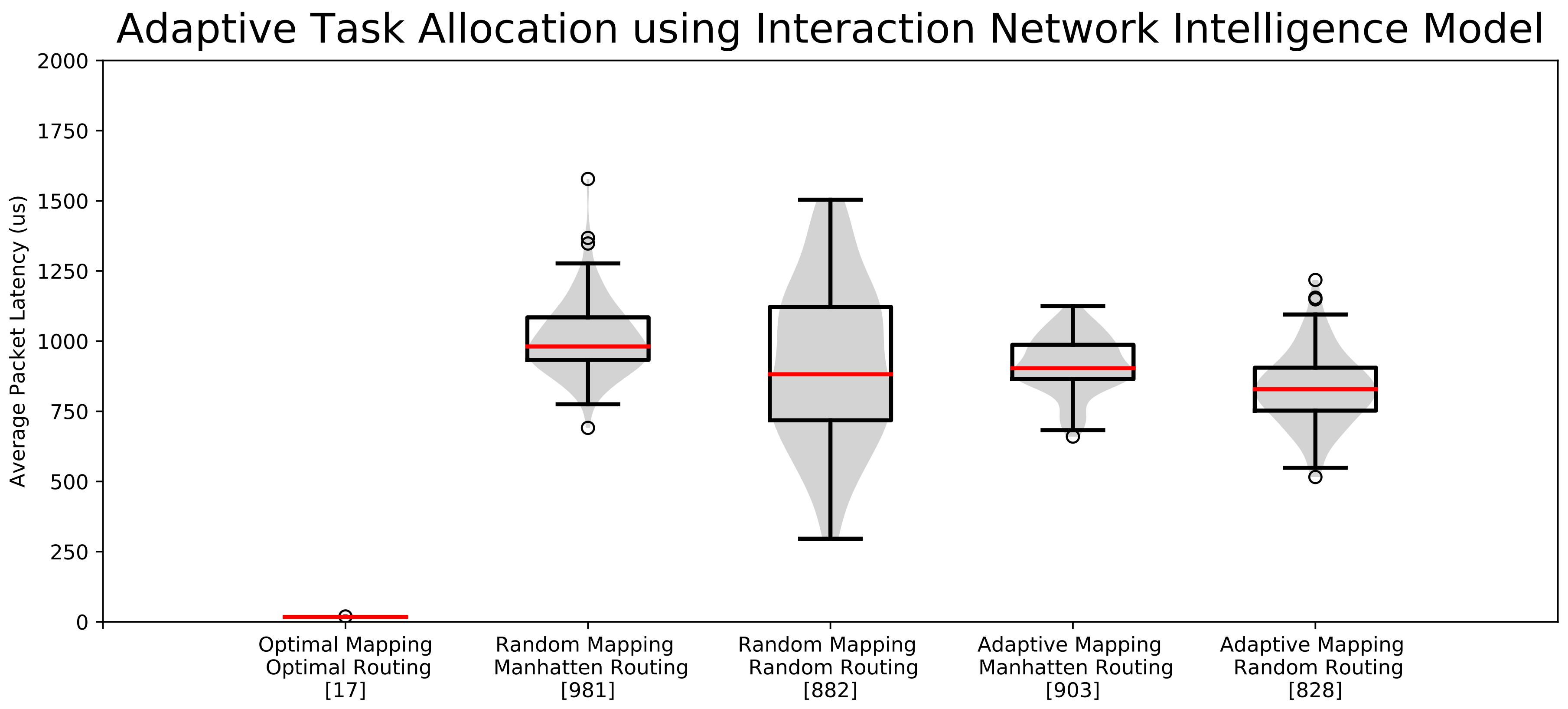


Figure 7.3: This graph shows the distribution of the average packet latency of each of 100 experiment runs of the task allocation configuration used for Experiment 6. The red lines indicate the median and the grey shadow shows the distribution of the actual average values used to create the boxplots. The median is also give in square brackets underneath the x-axis labels. The intelligence model works well at reducing the spread of the random mapping, random routing case.

Configuration	**Task 1**	**Task 2**	**Task 3**
Optimal:	40	40	40
Random + Manhattan:	16	26	20
Random + Random:	4	15	8
Intel + Manhattan:	34	41	36
Intel + Random:	44	58	49

Table 7.3: The average number of individual nodes of each experimental configuration that contribute to the processing load of the application. The low values for the non adaptive approaches show an example of the effect poor mappings can have on the many-core.

improvement on this by removing two thirds of the total deadlock the happened during the random routing configuration. As can be seen in the lower graph of Figure 7.4 a large amount of the critical deadlock occurs during the initial phase of the experiment where the system has not had a chance to adapt to the routing tables and so it operation this number will be much lower (although for the first quartile the level of deadlock stays high for most of the run, despite the mapping's good latency otherwise).

Configuration	**Average Deadlocked Packets per Run**
Optimal:	0
Random + Manhattan:	0
Random + Random:	7,410
Intel + Manhattan:	106
Intel + Random:	2,480

Table 7.4: Critical deadlock figures for each configuration of Experiment 6. Critical deadlock is the case where the node has to accept a packet that is of the incorrect task due to all router ports having timed-out waiting for the deadlock to clear.

Figures 7.4 and 7.5 give us some insight into the performance of the intelligence model. From Figure 7.4 it can be seen that the initial poor latencies are reduced after a small period, however the latencies do not settle completely suggesting that there are still bad mappings introducing congestion and possibly disrupting good mappings that have already settled. Despite this the middle graphs of Figure 7.4 show the rate of packets moving around the system and this is fairly stable for all three quartiles despite the less stable packet latency. This suggests that the network has settled to some extent and the task switching dynamics of Figure 7.5 agree that, for the Q2 and Q3 graphs, the task switching activity has dropped significantly since the dynamic start to each experiment run. Although at $\approx$500 task switches per second it has not settled down to what could be considered a steady state.

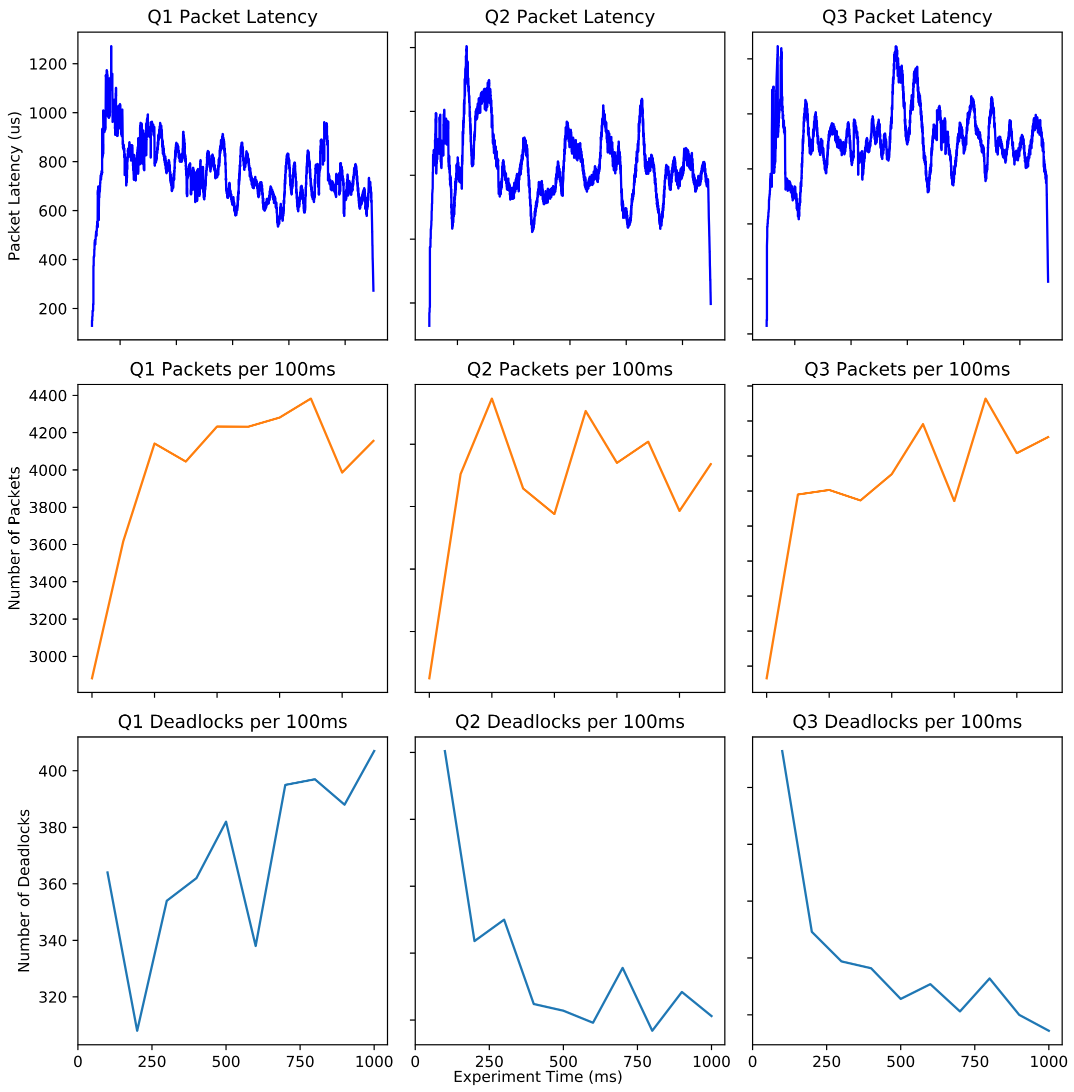


Figure 7.4: Time domain plot of three runs from Experiment 6. All three runs are chosen from the Random Mapping, Random Routing case and represent the Q1, Q2 and Q3 points on the distribution of run average packet latencies. The packet latency records the latency of all packets received and a moving average over 100 packets removes enough variation to see the longer term trend. The packets and deadlock counters take the average number of the events over the last 100ms.

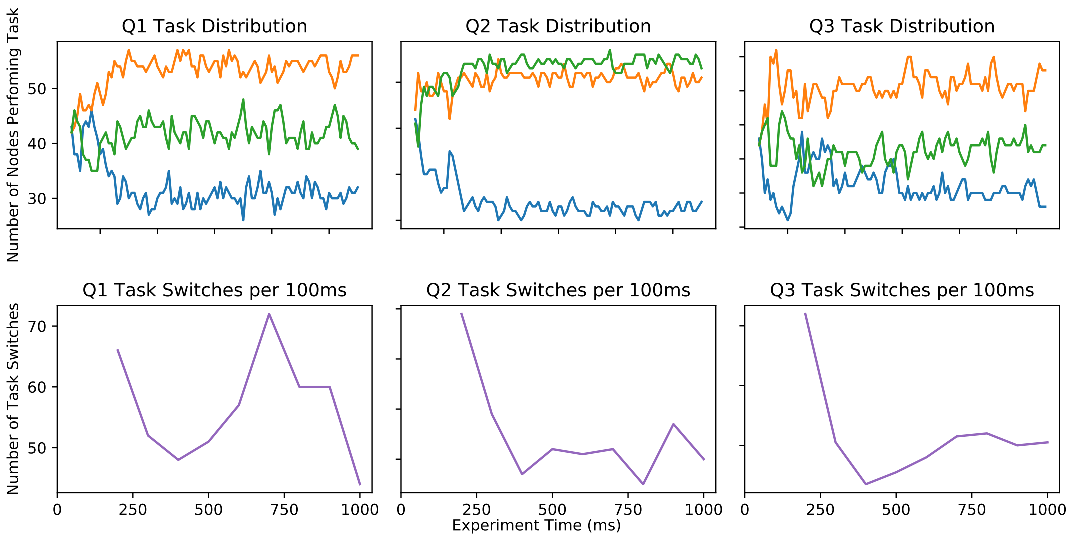


Figure 7.5: Time domain plot of three runs from Experiment 6. As with Figure 7.4, all three runs are chosen from the Random Mapping, Random Routing case. The top graph shows the number of processing elements carrying out each task over time with the following task mappings: **blue:** Task 1, **orange:** Task 2 **green:** Task 3. The lower graph shows the average rate of task switches occurring every 100ms.

7.3.5. Review

This experiment has shown that the Interaction Network intelligence model can be implemented on the CIA and has some self-organisation properties when applied to the many-core. It can improve on packet latencies and deadlock when compared to a random alternative, however it does not reach the capabilities of an optimal mapping. This could be due to the role of packet counters as the monitors in this intelligence and so the mapping will be ultimately be driven by the routing patterns, the random routing tables may not give enough traffic diversity for the more optimal task mappings to be encountered.

7.4. Experiment 7: CIA Implementation of the Foraging For Work (FFW) Model

7.4.1. Experiment Overview

In the same fashion as with the previous experiment, this experiment implements the Foraging For Work intelligence model using the CIA. The CIA implementation of FFW will allow further hardware savings than the Network Interaction model as the FFW timer uses the Picoblaze's interrupt handler, requiring interrupt support from the intelligence microcontroller. This experiment will also show that the CIA is capable of supporting time-domain driven inputs as the FFW decay tick is implemented as a global tick signal.

As with the Network Interaction CIA implementation, the use of response-threshold models in the FFW Picoblaze implementation lends to a straight-forward translation to the CIU architecture. The FFW timer is prescaled by a CIU and a level thresholder is used to disable the task switch mechanism when the FFW model is in "task assigned" state (i.e. when the stimulus for the present task is high enough to keep the individual working on that task). A difference with the Picoblaze implementation is an extra monitor that indicates if the node's Microblaze is performing a processing event for the current task. This monitor is required as the CIA cannot understand what task the node is currently performing, as it does not have memory capabilities, and once a task switch is suggested the threshold counts are reset. The extra monitor takes the place of the SOP to current task ID check that the Picoblaze undertook (as shown in Figure 6.6). Despite this change, it is expected that the same high-level behaviour can be emerged as the Picoblaze implementation.

7.4.2. Knobs and Monitors

The same monitors as the previous experiment are used: three SOP detects using three Match to Impulse monitor. Two additional monitors are required: the 1ms global tick source which is used for implementation of the task decay part of this model and also the *Microblaze Busy* flag from the local Microblaze MCS. This flag is set to '1' when the Microblaze is performing the "CPU Time" processing portion of the task model and '0' when the processor is not performing the simulation application task.

Again the same task suggestion knob is required for this experiment to output the suggested task to the Microblaze MCS, implemented using an Impulse to Vector knob.

7.4.3. CIA Configuration

The layout of the CIA used for this set of experiments is shown in Figure 7.6. This is very similar to the model used for the previous set of experiments, with an extension for the task decay time. Each of the SOP match detectors are connected to an CIU, but this time the CIU's threshold value is set to one. This means one SOP can cause a task switch as the impulse is immediately forwarded to the output task switch knob via the HBus. Each CIU however has an inhibitory input that is driven by the task decay time circuitry. This inhibitory signal will stop the passing of SOP impulses to the task switch knob.

The task decay time is implemented in CIB5. Two CIUs are used: CIU2 is used to scale the 1ms tick for the task decay time: raising this CIU's threshold will result in a larger number of ticks required for the CIU to impulse and will increase the task decay time. It is reset by the MCS busy flag. CIU3 is configured in "Level mode", in this mode the CIU will not impulse but will instead output a constant '0' or '1' depending on if the internal count is below or above the threshold value. As the MCS busy flag is the excitatory input for this CIU, any task activity longer than a few microseconds on the node (i.e. when a packet is received) will result in this CIU's internal count rising above the threshold to its maximum value. This will cause the CIU to output a level '1' and so will inhibit the task switch mechanism implemented in CIB0, CIB1 and CIB2.

The inhibitory input to CIU3 is the scaled 1ms tick (output of CIU2) and so this will reduce the internal count every time this tick time has elapsed. When the count drops below the threshold value then the FFW intelligence has entered Forage mode. This causes the CIU to output a level '0' and so the inhibitory signal to CIU the task switch mechanism in CIB0, CIB1 and CIB2 will be released and so they will suggest a new task depending on the next SOP impulse generated.

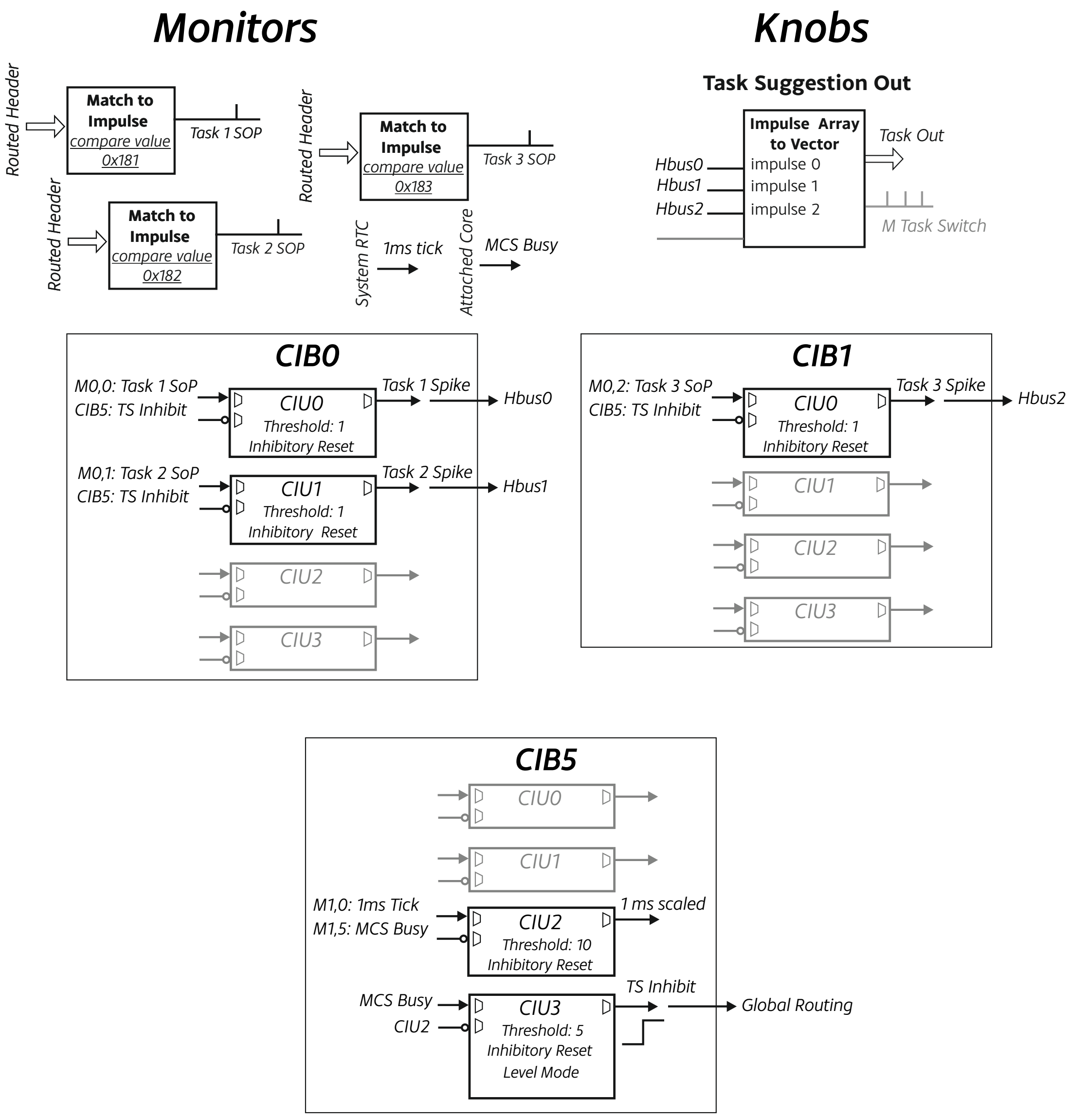


Figure 7.6: The implementation of the FFW intelligence is similar to the interaction network implementation. The three SOP detectors are still used but have their thresholds set to one and so emit an impulse as soon as their respective SOP arrives. Unlike the interaction model however their inhibitory input is controlled from another intelligence circuit implemented in CIB5. This circuit uses the “MCS busy flag” from the processing core and a scaled 1ms tick to disable the task switch mechanism when there is enough work to do for the current task to keep the MCS active.

7.4.4. Experiment 7.1: Foraging For Work Intelligence Model

This experiment is the equivalent of Experiment 6 but for evaluating the Foraging For Work intelligence model against the same non-intelligence configurations (optimal solutions with optimal routing tables, random topologies with Manhattan distance routing tables and a random topology with random routing tables). Once again, the intelligence is tested in two configurations: the first has a random task mapping and the same Manhattan routing metric as the non-intelligent case and the second has random routing tables and a random task mapping.

The distribution of the run average packet latencies is presented in Figure 7.7. As with Experiment 6, both of the adaptive approaches achieve a more optimal distribution then the random mappings, although still far from the optimal mapping case with a median latency of 638μs against 17μs for the optimal mapping. However the FFW intelligence has managed to achieve a significantly better median latency that the Interaction Network model (828μs) for the random mapping and random routing tables.

By exploring these results in the time domain in Figure 7.8 the difference in adaptivity between Q1 and Q3 is clear with Q3 seemingly settled on an average higher latency but exhibiting more stability than Q1 and Q2. Unlike the interaction model, the number of packets appears to fluctuate quite wildly, but with 128 cores sending data roughly every 1ms these variations are within what could be expected. The deadlock performance is poor however at nearly twice that of the interaction model. Figure 6.9 shows the task switching dynamics for the three distributions which is surprisingly high after a settling period. The number of task switches is high but fairly constant, so the network could be undergoing an oscillating effect; possibly caused by the pathways also causing the high levels of deadlock. This higher level of deadlock is confirmed in the critical deadlock numbers in Table 7.6. The workload figures in Table 7.5 show that not as many nodes are be getting packets to work on when compared to the interaction model and so there seems to be less traffic route diversity with the FFW model.

Configuration	**Task 1**	**Task 2**	**Task 3**
Optimal:	40	40	40
Random + Manhattan:	16	26	20
Random + Random:	4	15	8
Intel + Manhattan:	24	32	27
Intel + Random:	19	33	24

Table 7.5: The average number of individual nodes of each experimental configuration that contribute to the processing load of the application.

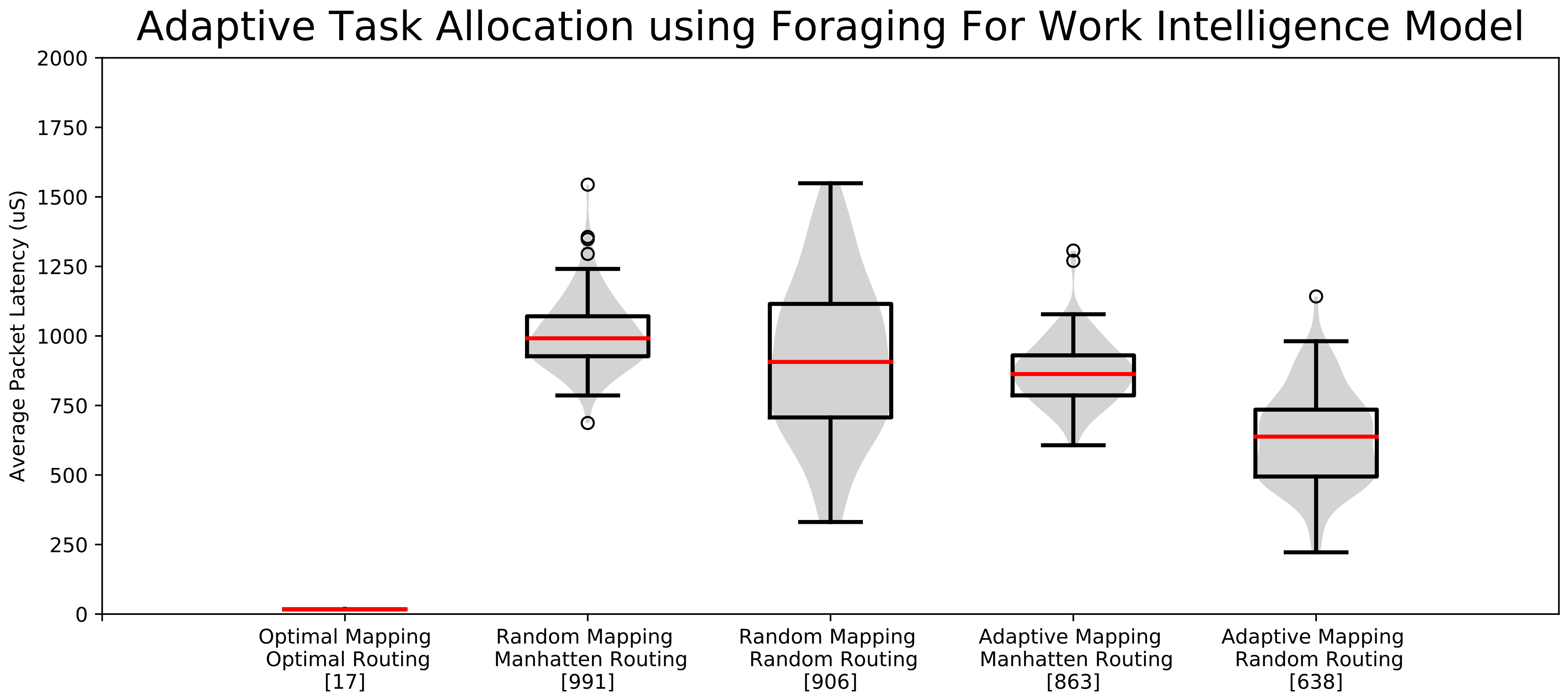


Figure 7.7: The distribution of run averages for the Foraging For Work intelligence model. The model results in more spread than the interaction model but this seems to allow it to achieve a significantly lower median and a better lower point than the random mapping case.

Configuration	**Average Deadlocked Packets per Run**
Optimal:	0
Random + Manhattan:	0
Random + Random:	7,390
Intel + Manhattan:	505
Intel + Random:	4,820

Table 7.6: Critical deadlock figures for each configuration of Experiment 7.

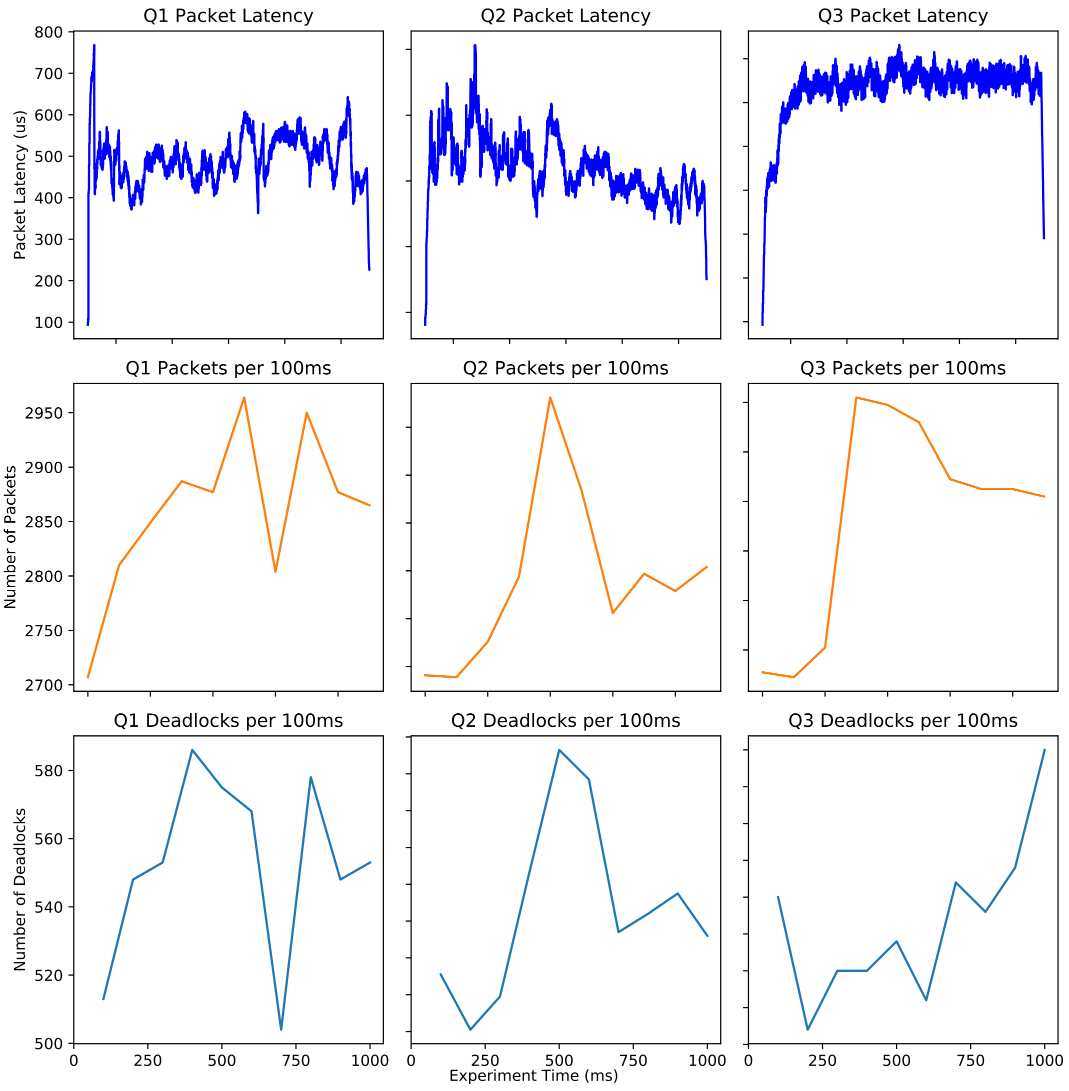


Figure 7.8: Packet characteristics for Experiment 7. The latencies (with a moving average of 100 packets applied) are fairly stable espeically Q3 that seems to have settled on a higher average latency. However the number of deadlocked packets are high (which do not count to the packet latency).

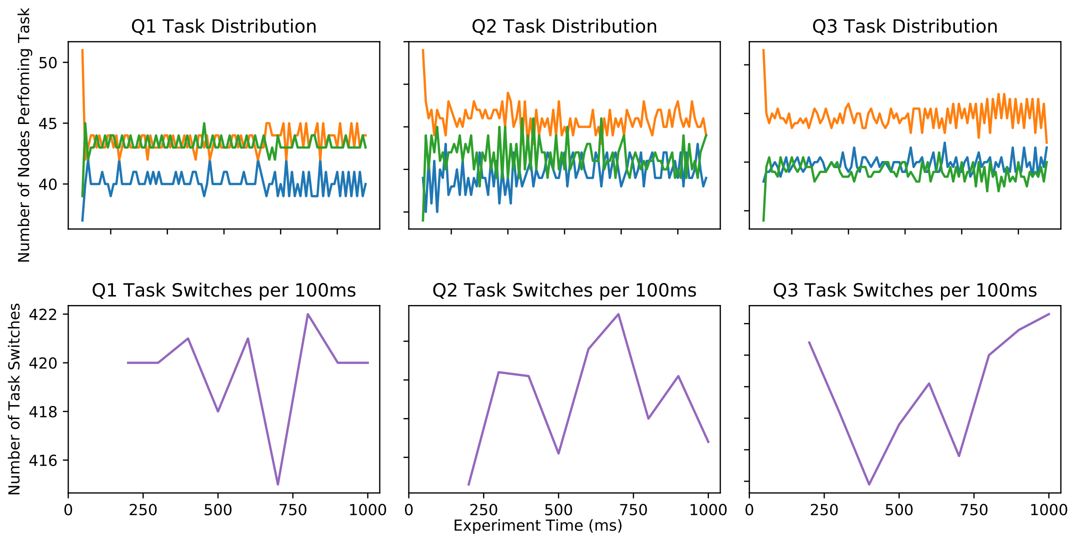


Figure 7.9: Task switching characteristics for Experiment 7. Despite the high number of task switches per second the task distribution seems to settle well for the three tasks, although the shape of the plot suggests some oscillation may be taking place in all three runs. This could suggest why the task switch rate is high but the packet latency has managed to settle.

7.4.5. Review

Foraging For Work translates well to implementation within the CIA and many-core context but seems to provide dynamics that are heavily unsettled, causing many task switches and deadlocks. As with the Interaction Network models, this could be a limitation of the random routing not providing enough diversity for the mappings to self-organise into topologies that are more stable for FFW. The model also seems to be prone to oscillations, although it must be noted that it seems to manage reasonably stable packet latencies despite these oscillations and other instabilities in the topology.

7.5. Experiment 8: Experiment Performance of Intelligence Array Relative to Clock Frequency

7.5.1. Experiment Overview

An appealing aspect of the CIA is the ability to use it at low power and "hide" it in the Dark Silicon that would be left empty as a thermal buffer. This would allow the protective and run-time management aspects of the CIA to run with negligible power and thermal overhead relative to the requirements for the high-performance system. If the system has been partitioned with thermal buffers between high-performance circuitry, then the hardware resources required by the CIA are also negligible as they would be left unused (or dedicated to cache) as part of the Dark Silicon role. Running the CIA at a lower frequency also reduces timing performance requirements of the CIA, allowing longer routing paths for monitoring signals and reducing the design-insertion overhead when synthesising a CIA into an existing design.

This experiment investigates the effectiveness of the CIA when run at a clock frequencies significantly lower than the circuit it is protecting. Running the CIA at a lower frequency will reduce the response time of the intelligence, potentially leaving the intelligence too little time to react to changes in the system, or "keep up", which could result in poor performance of the task allocation models. This approach will also require latching of information captured by monitors at full-system speed and so some loss of information into the intelligence will occur; e.g. if a number of SOP events occur between CIA clock cycles then these events will be reduced into only one event. However, due to the periodic nature of the application richer sets of information can be gathered through sampling over a longer time period. Therefore, the intelligence could be expected to gather the same information if it is run slowly; although over a longer time frame and likely with a sacrifice of increased settling time or slower adaptation rates to perturbations. There is also the expectation that at a certain reduced sensory speed the intelligence would not keep up with the effect that it has on the system when the knobs are updated, thus it could be that at a certain frequency the intelligence would not settle on a task allocation.

7.5.2. Experiment 8.1: Underclocking the Intelligence to 50kHz

The maximum operating speed of the CIA is 25MHz. This is due to a long combinatorial path in the routing of the GBus signals which are not registered, resulting in a critical path which is the longest possible FPGA routing path between CIU outputs (where the signal is last registered) and the input of all CIUs in the column. Due to the programmable input routing multiplexors in the CIUs and CIBs, this path has several points of signal delay present. This bus could be registered in future CIA implementation to allow higher operating frequency of the embedded intelligence. The NoC is running at a fixed 100MHz and the Microblaze is able to switch from 20MHz to 300MHz as controlled by either the intelligence or the RCAP port. Therefore the CIA is designed with clock domain crossing circuitry on the relevant knobs and monitors. The actual operation speed of the CIA is set by a field in the configuration chain and can be set from 25MHz down to 50kHz. This is the range this experiment tests with the Interaction Network intelligence and also the Foraging For Work model.

Figure 7.10 shows the results of the frequency sweep and it can be seen that the performance of the Interaction Network task allocation intelligence is not affected by the reduction in frequency, even down to 50kHz. As this model relies on task detection impulses which are latched as part of the clock domain crossing procedure, it is likely that reducing the clock speed will just extend the time frame required to adapt over (i.e. decrease the adaptivity response time). Some packets will be missed (only one packet SOP can be latched for each task at a time) but this does not appear to affect the dynamics. The results for the Foraging For Work model is given in Figure 7.11. This model also does not suffer from the significantly reduced clock frequency, despite the FFW model having a time based element to it. However, the clock period at 50kHz is 20μs and so there are still 50 clock cycles per 1ms tick of the FFW inhibitory input, so in theory 50 latched SOP impulses could still be captured by the CIA without losing information whilst running at 50kHz.

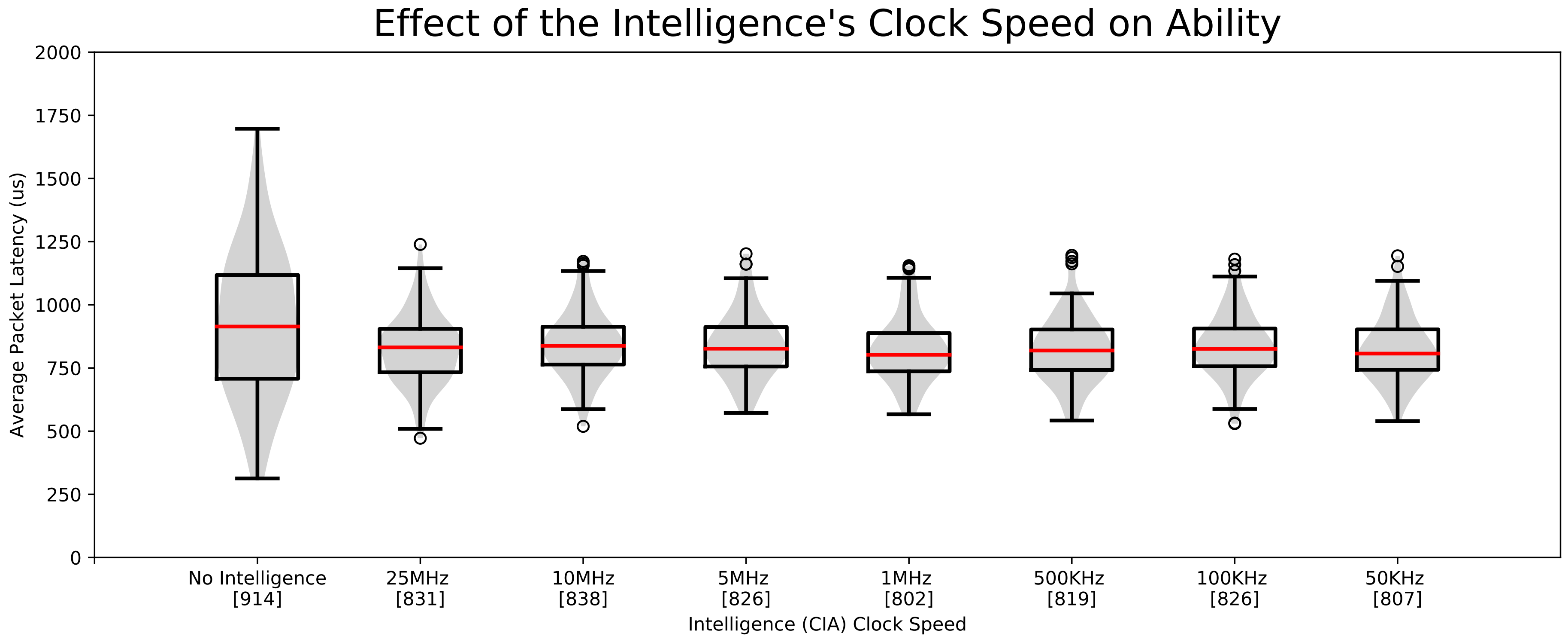


Figure 7.10: CIA Frequency Sweep with Interaction Network model. This graph shows the distribution of the average packet latency of each of 100 experiment runs with the intelligence clocked at the indicated frequency.

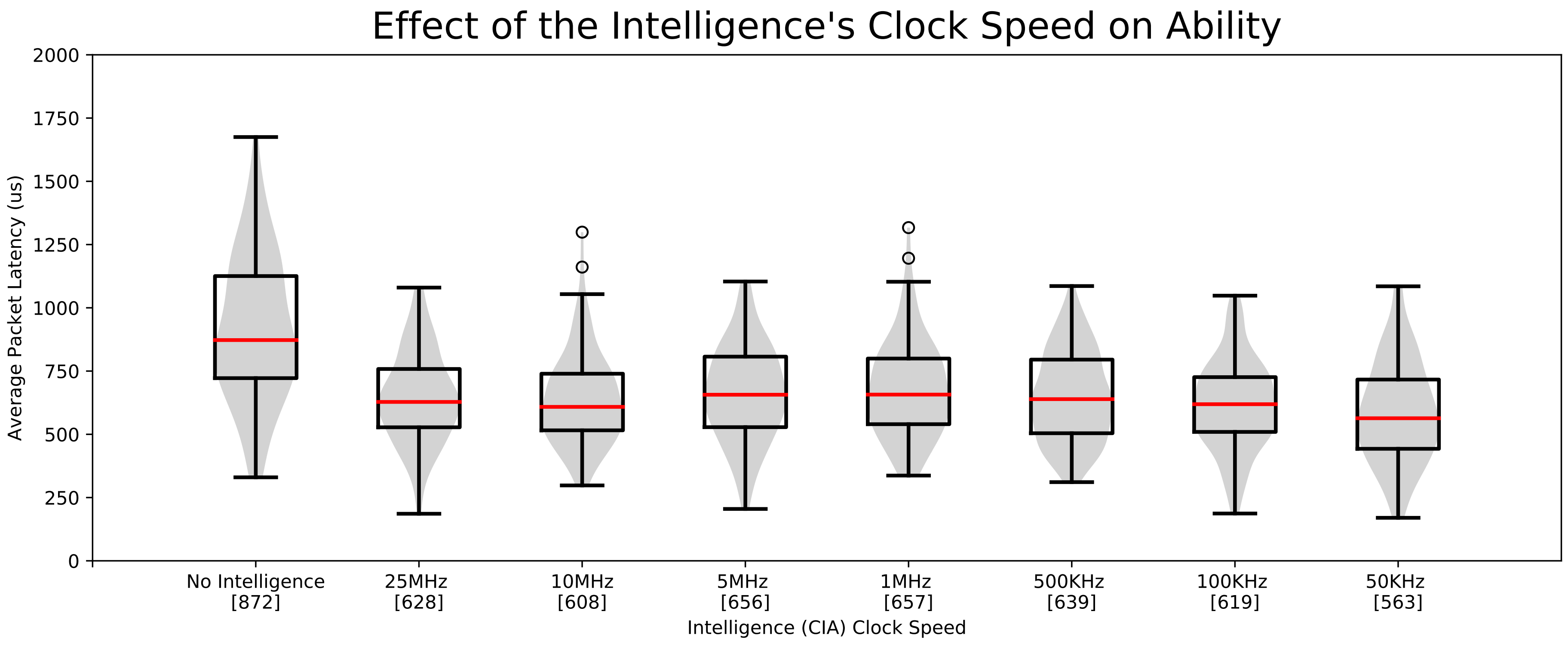


Figure 7.11: CIA Frequency Sweep with Foraging For Work model. This graph shows the distribution of the average packet latency of each of 100 experiment runs with the intelligence clocked at the indicated frequency.

7.5.3. Review

This finding has great implications for using impulse-based embedded intelligence for an on-chip monitoring application. However, further investigation will need to be undertaken to understand the dynamics deeper as these are relatively simple intelligence models requiring latched signal inputs that are not crucial if they are missed. The fact that we can run the intelligence on the CIA at a significant lower frequency than the application node with little impact on its ability is a good argument for placing these embedded pathways in logic that would otherwise be wasted to Dark Silicon.

The number of SOPs that could be missed due to underclocking of the intelligence is highly dependant on the application packet rate and how hotspots of network traffic are distributed. The linear test application in these experiments has a packet rate of 4ms, or 250Hz. In the optimal task allocation case where a single packet is sent and a single packet is received per node this results in an SOP event rate of 500Hz, far beneath the lowest CIA frequency tested in this experiment of 50kHz. In non-optimal topologies or non-linear task graphs this rate will be much higher due to hotspots or nodes sending/receiving multiple packets; however even in this case the rate of packets moving through the router will need around 100 times more than the optimal case before the intelligence starts to lose information. In this application this would be nearly impossible as it would require all packets of all nodes to pass through a single router before information is lost. Therefore, it is likely that the CIA clock frequency can be dropped to 5kHz before hotspots start to lose information and poor decisions may be made.

This frequency will be unique to the application running on the system, although packet generation rates in the millisecond range are a reasonable abstraction for applications where data processing happens on the nodes and so clocking the intelligence in the kHz range will be applicable for many applications with packet rates up to every 10μs. This frequency is also unique to task allocation pathways: events that form the monitors of other decision pathways may occur at a higher rate and so those pathways would need to be clocked at a different rate to the task allocation pathways. Determination of the pathway operating frequencies would require either an application analysis to determine monitoring signal rates (parts of this information would be available at the application design phase) or through use of a design optimisation or self-learning process in the intelligence itself.

7.6. Summary

These experiments have shown that the social insect inspired intelligence is capable of adaptive behaviours at very low hardware overhead: each CIU has only 2 active slices once programmable routing resources are removed. This overhead scales linearly with each extra node and the behaviours it has shown in this chapter have coped with scaling up with over one hundred many-core nodes. The fault tolerance capability does not require any extra hardware resources within the intelligence and is capable of recovering system performance over a wide range of fault injections. Limitations have been found with the social insect model, however it is likely that these can be overcome through some additional intelligence circuitry, albeit the required extensions are possibly not social-insect inspired. This chapter has shown how two bio-inspired models are translated into a form suitable for hardware realisation using the CIA.

As with the previous chapter, the goals of an autonomous system can be used to qualify the behaviours seen in these experiment to gauge the degree to what this system is autonomous:

Self-Configuration: The goals achieved in the previous chapter have also been achieved with the CIA implementation of the intelligence models. Reducing the clock frequency of the CIA does not have a limiting effect on the self-configuration property.

Self-Optimisation: The goals achieved in the previous chapter have also been achieved with the CIA implementation of the intelligence models. Reducing the clock frequency of the CIA does not have a limiting effect on the self-optimisation property.

Self-Healing: The fault tolerance experiment has shown that both of the embedded intelligence models support self-healing capabilities as an intrinsic part of the decentralised task allocation. The ability of the system to recover application performance from even a large number of faulty nodes shows that the model can use spare resources (inactive nodes) and reconfigure active nodes to allow survival of the application.

Self-Protection: The fault model in this chapter considered random high-level faults. Such faults are difficult to predict. A self-protection mechanism would need to monitor wear of the nodes or global resources (such as clock-tree buffers) or other indicators of a fault becoming likely to occur and make informed decisions on how to manage this.

Other fault models could use chip-level sensing or long term monitoring of system characteristics to bring information into the intelligence that can be used to pre-empt a fault happening. This can feed into some of the decision units to stop faults before they occur and so provide a self-protection capability to the system. The experiments in this chapter did not offer such a capability however.

This chapter has demonstrated that self-configuration, self-optimisation and self-healing of the task allocation can be exhibited by both of the emergent intelligence models and that the models will still support self-configuration and self-optimisation even when implemented directly in hardware using the CIA.

Chapter 8

Conclusions

This thesis has covered a wide range of topics from upcoming issues with silicon devices, to biological models of natural complex system to low level FPGA design. This section ties these ends together with the aim of suggesting what the future of such systems could look like and what has been learnt about the applicability of the social insect models to large scale many-core systems.

8.1. Summary

8.1.1. Social-Insect Inspired Adaptive Task Allocation

The experiments in Chapter 6 translate and implement two of the social-insect intelligence models from Chapter 3 to run emergent task allocation on Centurion, using a Picoblaze microcontroller to implement the intelligence models. The two intelligence models chosen, Network Interaction and Foraging For Work, use simple response-threshold decision models to interpret environmental stimuli and are considered to be hardware efficient to implement. Foraging For Work has a temporal aspect to the model whilst Network Interaction only relies on dynamics set up by neighbouring nodes.

In the baseline experiments with no intelligence, there was a clear correlation between using random routing settings as a starting configuration and poor application throughput and high number of deadlocked packets. This provides a challenging problem space for the intelligence models to operate with, but gives confidence that the intelligence will work with even larger-scale systems as it does not require any analysis of the routing or task allocation before deployment.

In the experiments, the Network Interaction model settles down to topologies that are stable with respect to packet latency and the number of deadlocks, however the settledness seems to come from reducing the number of Task 1 nodes (the primary producer tasks) to a level that means the task graph is very poorly supported. In some experiments this had the effect that the ratio of Task 1 nodes to 2 and 3 was still not satisfactory to reach the performance levels of the Manhattan routed case. Foraging For Work performs similarly to the interaction model and also suffers from the same pitfalls regarding premature settledness coming from reducing the number of Task 1 nodes to a level where the task graph cannot be sustained.

To counter this undesirable effect, an extension from the Network-Interaction biological model is added to both models in the form of self-regulation. This forced nodes to switch to Task 1 if they are idle for a period of time. This adds extra dynamics that improve the task throughput of the system for all task graphs, but the performance improvement and performance distribution is markedly better for Foraging For Work. The fact that both the self-regulation and the Foraging For Work are both temporally driven could also be a factor in their good compatibility.

Chapter 7 tested the fault tolerance capabilities of the bio-inspired models by injecting a number of node-level faults and measuring the system response. No extra intelligence capability (and therefore no extra hardware) was added to the bio-inspired models, demonstrating that fault tolerance is an inherent property of the emergent models. Both models could cope with up to 8 failed nodes before performance was diminished compared to a no-fault mapping, indeed Foraging For Work did not start to diminish performance until 32 faults were injected. This confirms the presence of inherent fault tolerance in the mechanisms of the bio-inspired models.

Finally, the hardware overhead of using a microcontroller for the intelligence was reduced in Experiments 6, 7 and 8 by implementing the models using the CIA. Both the Network Interaction Model and Foraging For Work were implemented using parallel decision pathways located in the threshold units of the CIA. The emergence of task allocation in both models' behaviour was verified experimentally: both models exhibit equivalent emergent behaviour as seen in their microcontoller implementations. This allows a hardware efficient implementation by removing the configurable aspects of the CIA to leave only the threshold decision units (a number of 6-bit counters).

The potential for a low power intelligence implementation was demonstrated by slowing the intelligence and observing that the emergent task allocation still occurs. It was found that even at the CIA's slowest operating speed (50kHz) very little effect on the quality of the emergent task allocation is seen. It is suggested that this is likely due to the low sampling frequency required for the signals that the CIA monitors for the task allocation intelligence (packet SOP); this signal triggers at around 1kHz for a good task topology.

8.1.2. Effectiveness of the Centurion Platform

The final iteration of the Centurion platform resulted in a 128 node many-core arranged in a 8 by 16 grid NoC. This core count fulfils Centurion's goal of supporting large-scale many-core research and allowed experimentation in hardware that would not have been possible with any existing off-the-shelf single chip many-cores. The low level control signals (knobs and monitors) allow the intelligence to interact with the router control logic to successfully manage aspects of router behaviour. The experiment monitoring capabilities (event logging and hi-speed retrieval) supported online data collection whilst the experiment is running, allowing a rich set of events to be stored for later analysis.

Some of the advanced features of Centurion such as thermal monitoring and dynamic clock frequency scaling for each node were implemented and tested but ultimately not used for any experiments. Accurate representations of these advanced knobs and monitors are difficult to simulate and so with the foresight that these would not be used could make the justification of building Centurion weaker; node-level simulations could have achieved the same experimental results by marrying a NoC simulator, microcontroller simulator and a microprocessor simulator together to represent the router, intelligence and PE respectively. However, the compute power required to support such a simulation for at least 128 nodes with the same number of initial conditions explored on the platform (i.e. 1000 runs for the baseline experiments, 100 runs for each intelligence configuration) would have resulted in a huge amount of time required to generate the same result set. For this reason alone the development of the hardware platform for this work is justified.

The analysis of the experiments would have benefited from more event data, specifically from the routers as packets are routed or by adding information to the packet as it traverses the network (e.g. path taken, degree of deadlock resolution at each router, time spent waiting at each router). This would allow better understanding of the task distribution, as traffic and deadlock hotspots could be identified and their impact on the effectiveness of the task allocation understood. This would allow metrics that are more specific to many-core systems and NoCs to be used for determining the effectiveness of the experiments.

8.1.3. Implementation of Decision Pathways using the CIA

The response-threshold nature of the decision making elements of the implementation of both Network Interaction and Foraging For Work lends itself well to a digital hardware implementation. The *Configurable Intelligence Array (CIA)* allows these decision making pathways to be prototyped in hardware by combining low hardware cost decision threshold units (2 slices for decision making, 2 slices for programmable settings) with programmable routing structures. This allows impulse-based inputs from Centurion to be routing into decision making units and the output of these decisions to be passed to outputs into the routing control or into other decision making units. Both bio-inspired models were successfully implemented in the form required by the CIA and equivalent emergent behaviour observed in the experiments. This is despite the manual translation of the model directly into CIA decision pathways, verified with hardware implementation without use of optimisation tools or a simulation of the pathways with a representative simulation of the hardware platform. Further tuning of CIA parameters may have given better performance results through more efficient settings for threshold units for the applications they were applied to.

8.1.4. Autonomous System Evaluation

A goal of the embedded intelligence is to self-manage the many-core without inputs from a user or even from the application designer, the intelligence should strive to manage the system such that it adapts to the needs of the functions running on it. This process is on-going through the system's deployment lifetime and aligns with self-* aims aligned with the field of *Autonomic Computing*. The four main *self-** properties of autonomous system management were introduced in Section 2.4 as goals for the intelligence to be able to support. In nature social-insect colonies exhibit behaviours that can fulfil aspects of all of the following goals. In this work the intelligence displayed self-* capabilities against the following criteria:

Self-Configuration: The ability of the intelligence to adapt to the random routing such that the task graphs can complete a full Task 1 to Task 3 flow of data is considered self-configuration, as without this configuration no processing would happen. This is captured in the experiments through the measurement of Task 3 throughput and the number of deadlocks in the system (a deadlock ultimately means a node of an incorrect task has sunk the packet). Both of the emergent task allocation models exhibit self-configuration and this is also achieved with the CIA implementation of the intelligence models.

Self-Optimisation: Reducing the number of routing steps a packet needs to do between nodes provides an optimisation path on the self-configured task topology that the intelligence can exploit. For the Network Interaction model, the self-optimisation is inherent in the way the intelligence will reduce the number of routing hops. A large task packet response is taken up by a node immediately if the stimulus is greater than its current task. Foraging for Work only has motivation to optimise once a node becomes inactive enough to trigger the Foraging For Work window where it will accept the next task that arrives, the self-regulation extension is a source of disruption to the topology such to provide this motivation to Foraging For Work. Evidence of self-optimisation is seen in the experiments with decreasing packet latency seen at the start of the time domain plots of most experiments.

Self-Healing: The capability of the emergent task allocation to cope with failed nodes was demonstrated in the fault tolerance experiment. The ability of the system to self-configure and self-optimise after the injection of faults allows application performance to be recovered to pre-fault levels for a low number of faults and with a small loss of performance for cases up to 32 faults.

Self-Protection: Supporting self-protection requires allocation of spare resources and anticipation of faults occurring and mitigating before they occur. The intelligence models do not currently exhibit these properties. Allocation of spare resources (nodes) does help with the self-healing case but these are an accidental by-product of the task allocation not mapping perfectly to resources available, as opposed to a feature of the intelligence determining that it should allocate spare resources for self-protection.

From this evidence it is suggested that autonomous task allocation can be considered a property of both Network Interaction and Foraging For Work intelligence models for the cases used in these experiments, however it is not claimed that they can be applied to all types of task graphs and system applications and expect the system to autonomously manage the task allocation. Further experimentation is required with these models to show that they can work in the general case. This work has not found any immediate reasons as to why a general application task graph, even larger scale systems or different sets of application execution profiles could not be supported.

Self-protection does not have a test case as of yet. It can be seen how it could be supported by the models by using thermal and application-based monitors to inform decision pathways that can be designed or trained to detect likely conditions of out of envelope use, before the system reaches the point of failure.

8.1.5. Overall Conclusions

The introduction proposed that *Embedded social insect intelligence models derived from studies of the social insects can exhibit highly-scalable adaptive behaviours suitable for managing complex digital electronic systems*. The social insect intelligence models presented and explored in Chapter 3 suggested a range of fundamentally high-scalability models of intelligence at both the individual and at the colony level. This included discussion of tradeoffs such as the cost of learning versus the cost of memory, the size and complexity of an individual versus the size and social complexity of an interacting colony of organisms. Such tradeoffs are extremely relevant to modern embedded artificial intelligence as single chip systems have reached a size where such tradeoffs can start to be considered.

The following sub hypotheses were also proposed:

Models of task allocation in social insect colonies provide appropriate inspiration

for enabling self-organising task mappings for many-core systems.

The Centurion platform has been a key resource for assessing the suitability of these models, not only for their self-organising abilities but also for their suitability for implementation. If this work was done in either a NoC simulator or more abstractly in an multi-agent based simulation, it would be much harder to gauge the effectiveness of these models at an embedded circuitry level. The implementation of these models, both in the Picoblaze and the CIA has shown that they can be implemented using a reduction to a digital form (8-bit processor ALU or 6-bit thresholder) which has extremely low hardware overhead. The experiments in Chapter 6 and 7 showed that emergence could be achieved and so this translation can be considered successful: the implementation was inherently highly scalable and no pre-analysis or global knowledge was required when using the intelligence models. On reflection however there are several issues with the translation these models:

- Ensuring that a full set of stimuli is covered and translated from the application domain into the social-insect model. With more complex applications or a more comprehensive social insect model it will be difficult to ensure that all stimuli are covered and in a form that captures both the application need and the input form required by the model. For example the translation of individual movement in the ant colony and information transfer was successful in this model but there will be other physical-world versus digital-world translations that will not translatable.

- Several necessary simplifications of the many-core applications. Ants have task repertories but they are still very small when compared to a general purpose computing platform. Whilst the representative task graphs helped prove the concept, it is likely that an application translation methodology would be needed to take a user application and apply it to a system managed by these intelligence models (however this is an open problem for many-core system design, even for non-bio inspired management designs).

- Response-thresholds still have a design element as their threshold values need relating to the application environment. In the experiments of Chapter 6 this was mainly in the temporal domain as the FFW threshold was related to the task period. For general purpose applications such values will be critical and difficult to extract, and it is likely that self-learning or other optimisation techniques will be needed to capture the correct parameters. Although this will be a markedly smaller problem space than the NoC design space for large NoCs.

The results from the dynamic task allocation experiments have shown that both bio-inspired runtime management models exhibit emergent adaptive properties that are useful in large fault cases, and hence indicate a degree of inherent scalability. Whilst such global failure cases may be mitigated through circuit hardening or additional redundancy, this fault case is also relevant for high-processing power devices that require the parallel throughput of a many-core system but are deployed in remote application scenarios with requirements of autonomous operation and long lifetime. As faults develop in the field over the lifetime of a device the emergent task allocation can adapt the task topology to achieve a managed degradation of system performance that may allow a device to operate for longer in its deployed environment.

The digital translation of these decision pathways has been a key contribution. The response-threshold units appear in many biological models and represent a very simplified form of neural pathway that maps well to digital hardware fabrics, such as FPGA, and can be used to implement the bio-inspired social insect intelligence models presented in this paper in low-level hardware. This will provide a pathway for creating a design methodology for a generic response-threshold based intelligence systems. It could be a worthwhile activity to revisit the neural aspects of the hardware platforms detailed at the end of Chapter 2 with such an FPGA efficient impulse network and see if some older models should be resurrected on modern hardware platforms.

8.2. Further Work

8.2.1. Potential Centurion Developments

Further improvements to Centurion would provide new capabilities that the intelligence models can exploit. The most pressing addition is the support of adaptive routing and multi-cast functionality for better support of task graphs with a parallel element to them. An adaptive routing function is currently present in the Picoblaze version of the intelligence: a flag can be set that allows the router FSMs to source their routing directions from the Picoblaze instead of the RCAP tables. This knob does not exist in the CIA version however and implementing it for the CIA may prove more challenging due to the need to deal with the impulse outputs from the CIA. Therefore an adaptive routing capability or multi-cast option built into the control FSMs of the router which the CIA can manipulate the use of (i.e. likelihood of taking the parallel options for each task) will be the most effective method of implementing this functionality. A further step may be to replace the router control logic with a microcontroller. This could allow more routing capabilities to be supported such as packet priority, the ability to add to or edit packet headers as it passes through the router and the ability for the intelligence to source/emit packets. The Picoblaze intelligence has demonstrated how a microcontroller can be used with a only a small hardware overhead.

Logging and debug capabilities for the routing functions would also allow a much richer dataset to be collected for the experiments. This would allow a deeper insight into how and why decisions are made by the intelligence and will make routing hotspots much easier to locate. Centurion would also have benefited from a FPGA board that had power monitoring capabilities of the FPGA's power rails, as this would allow the link between power efficiency and efficient task allocations to be explored. This could be provided as a global monitor signal to all nodes to allow decisions based on power capabilities to be supported.

Finally, a series of ultra-high density FPGA devices has been released since Centurion has been developed. This would allow massive scaling-up of the platform, in the region of 700 - 1000 nodes for some of the largest FPGA devices available now. The smaller process technology should also allow a higher clock frequency for both the nodes and the NoC, bringing the total compute throughput of Centurion into a region more competitive with high-speed multi-core processors. The high-density FPGAs would present new challenges in the form of inter-die boundaries (the FPGA is made up of several dies and there are different performance constraints for signals crossing die boundaries) and implementing such a large design.

8.2.2. Further Experiments with the Social Insect Models

Several further improvements to the embedded intelligence are conceivable that would go beyond the models presented here. The addition of adaptive and multi-cast routing to Centurion would allow greater throughput as the intelligence can exploit the inherent parallelism of a task graph. Whilst there is not a direct mapping of a social insect model for this kind of adaptive routing, the stimulus-response threshold model could be used to allow the embedded intelligence to make decisions on the destination output port of incoming packets. Many of the models shown in Figure 3.1 feature mechanisms for adaptive thresholds, which are not yet considered in this work and could be used to allow the intelligence to learn which ports are better to use in an adaptive routing application.

Further experiments that tie the operating conditions of a node to application demand will be vital for managing the variability and thermal management of the many core to reduce the effects of Dark Silicon. The per-node thermal monitors presented in Chapter 4 are intended to monitor the local temperature of a node and be coupled to the node clock frequency knobs to provide run-time thermal management. These can also be combined with the task allocation decision pathways to make the decision space much more rich, a node deciding to move to an idle state or run much slower will have an impact on the workload of its peers. Social insect colonies exhibit idle colonies members and colony members that spend their whole lifetime in a low-productivity state and so it is likely that an suitable emergent decision model exists that can be translated for this purpose.

8.2.3. Future Development of Decision Pathways based on the CIA

The digital counter based decision-threshold units of the CIA have shown how these decisions can be made in a form that is hardware efficient, highly parallel and so likely to be power efficient over a comparable processor implementation. The ability to clock the CIA pathways much slower than the application they monitor will help reduce their dynamic power consumption to a point that is negligible when compared to the application they are applied to. If a design process is developed that can remove the programmable aspects of the CIA pathways once an intelligence circuit is prototyped, then this power consumption can be reduced to the point where only the counters and the propagation of the impulses is required. These pathways could then be made even more power efficient in an ASIC implementation by translating the pathways into asynchronous decision pathways or analogue implementations based on capturing charge with each excitatory impulse, with a controllable leakage representing the inhibitory factors.

The design of these pathways and the choice to use digital thresholders came from the biological social insect models. The biological models of Nervous Systems may be a better-suited source of inspiration for system runtime management. Neural pathways present in Nervous Systems combine a large number of monitoring elements into closed loop, threshold-based decision units to muscular (or otherwise) effector cells. It is seen that many sensory cells in insects and other simple creatures create impulses as their output and so the model should also translate well to monitoring digital hardware. Given the large number of sensory cells in a typical creature, development and growth of Nervous Systems must also exhibit power and area efficiency to suit the environment the creature has evolved to live in. Changing the biological metaphor may also offer a clearer route to how the pathways are designed. Exploring the effect of genetic factors, online learning and Nervous System growth as a creature develops may give insights that can be applied to creating a threshold-based Nervous System for protection of complex, large scale hardware systems.

Appendices

Appendix A

Device Clock Domain Mapping

Table A.1: Clock resources used per clock region. Each clock region supports a total of 12 global (BUFG) or local (BUFH) clocks.

Clock Region (L)	**Type**	**Clock name**		**Clock Region (R)**	**Type**	**Clock name**
X0Y8	BUFG	NoC_clk		X1Y8	BUFG	NoC_clk
	BUFG	DDR_200MHz			BUFG	LVDS_25MHz
	BUFG	DDRVP_200MHz			BUFG	DDR_200MHz
	BUFG	MDM_clk			BUFG	DDRVP_200MHz
X0Y7	BUFG	NoC_clk		X1Y7	BUFG	NoC_clk
	BUFG	DDR_200MHz			BUFG	DDR_200MHz
	BUFG	DDRVP_200MHz			BUFG	DDRVP_200MHz
	BUFG	clk_600MHz			BUFG	clk_600MHz
	BUFH	div_clk[0]			BUFH	div_clk[4]
	BUFH	div_clk[1]			BUFH	div_clk[5]
	BUFH	div_clk[2]			BUFH	div_clk[6]
	BUFH	div_clk[3]			BUFH	div_clk[7]
	BUFH	div_clk[8]			BUFH	div_clk[12]
	BUFH	div_clk[9]			BUFH	div_clk[13]
	BUFH	div_clk[10]			BUFH	div_clk[14]

	BUFH	div_clk[11]		BUFH	div_clk[15]
X0Y6	BUFG	NoC_clk	X1Y6	BUFG	NoC_clk
	BUFG	MDM_clk		BUFG	LVDS_25MHz
				BUFG	LVDS_100MHz
	BUFG	clk_600MHz		BUFG	clk_600MHz
	BUFH	div_clk[16]		BUFH	div_clk[20]
	BUFH	div_clk[17]		BUFH	div_clk[21]
	BUFH	div_clk[18]		BUFH	div_clk[22]
	BUFH	div_clk[19]		BUFH	div_clk[23]
	BUFH	div_clk[24]		BUFH	div_clk[28]
	BUFH	div_clk[25]		BUFH	div_clk[29]
	BUFH	div_clk[26]		BUFH	div_clk[30]
	BUFH	div_clk[27]		BUFH	div_clk[31]
X0Y5	BUFG	NoC_clk	X1Y5	BUFG	NoC_clk
	BUFG	clk_600MHz		BUFG	clk_600MHz
	BUFH	div_clk[32]		BUFH	div_clk[36]
	BUFH	div_clk[33]		BUFH	div_clk[37]
	BUFH	div_clk[34]		BUFH	div_clk[38]
	BUFH	div_clk[35]		BUFH	div_clk[39]
	BUFH	div_clk[40]		BUFH	div_clk[44]
	BUFH	div_clk[41]		BUFH	div_clk[45]
	BUFH	div_clk[42]		BUFH	div_clk[46]
	BUFH	div_clk[43]		BUFH	div_clk[47]
X0Y4	BUFG	NoC_clk	X1Y4	BUFG	NoC_clk
	BUFG	clk_600MHz		BUFG	clk_600MHz
	BUFH	div_clk[48]		BUFH	div_clk[52]
	BUFH	div_clk[49]		BUFH	div_clk[53]
	BUFH	div_clk[50]		BUFH	div_clk[54]
	BUFH	div_clk[51]		BUFH	div_clk[55]
	BUFH	div_clk[56]		BUFH	div_clk[60]
	BUFH	div_clk[57]		BUFH	div_clk[61]
	BUFH	div_clk[58]		BUFH	div_clk[62]

	BUFH	div_clk[59]			BUFH	div_clk[63]
X0Y3	BUFG	NoC_clk		X1Y3	BUFG	NoC_clk
	BUFG	clk_600MHz			BUFG	clk_600MHz
	BUFH	div_clk[64]			BUFH	div_clk[68]
	BUFH	div_clk[65]			BUFH	div_clk[69]
	BUFH	div_clk[66]			BUFH	div_clk[70]
	BUFH	div_clk[67]			BUFH	div_clk[71]
	BUFH	div_clk[72]			BUFH	div_clk[76]
	BUFH	div_clk[73]			BUFH	div_clk[77]
	BUFH	div_clk[74]			BUFH	div_clk[78]
	BUFH	div_clk[75]			BUFH	div_clk[79]
X0Y2	BUFG	NoC_clk		X1Y2	BUFG	NoC_clk
	BUFG	clk_600MHz			BUFG	clk_600MHz
	BUFH	div_clk[80]			BUFH	div_clk[84]
	BUFH	div_clk[81]			BUFH	div_clk[85]
	BUFH	div_clk[82]			BUFH	div_clk[86]
	BUFH	div_clk[83]			BUFH	div_clk[87]
	BUFH	div_clk[88]			BUFH	div_clk[92]
	BUFH	div_clk[89]			BUFH	div_clk[93]
	BUFH	div_clk[90]			BUFH	div_clk[94]
	BUFH	div_clk[91]			BUFH	div_clk[95]
X0Y1	BUFG	NoC_clk		X1Y1	BUFG	NoC_clk
	BUFG	clk_600MHz			BUFG	clk_600MHz
	BUFH	div_clk[96]			BUFH	div_clk[100]
	BUFH	div_clk[97]			BUFH	div_clk[101]
	BUFH	div_clk[98]			BUFH	div_clk[102]
	BUFH	div_clk[99]			BUFH	div_clk[103]
	BUFH	div_clk[104]			BUFH	div_clk[108]
	BUFH	div_clk[105]			BUFH	div_clk[109]
	BUFH	div_clk[106]			BUFH	div_clk[110]
	BUFH	div_clk[107]			BUFH	div_clk[111]
X0Y0	BUFG	NoC_clk		X1Y0	BUFG	NoC_clk

	BUFG	clk_600MHz			BUFG	clk_600MHz
	BUFH	div_clk[112]			BUFH	div_clk[116]
	BUFH	div_clk[113]			BUFH	div_clk[117]
	BUFH	div_clk[114]			BUFH	div_clk[118]
	BUFH	div_clk[115]			BUFH	div_clk[119]
	BUFH	div_clk[120]			BUFH	div_clk[124]
	BUFH	div_clk[121]			BUFH	div_clk[125]
	BUFH	div_clk[122]			BUFH	div_clk[126]
	BUFH	div_clk[123]			BUFH	div_clk[127]

Appendix B

Floorplanning Figures

Figure B.1 is a close up of the first three nodes in the many-core and shows how densely packed the design is. It also shows how the tools have to fit the nodes close to the embedded BRAM (at this location in the device, the BRAM columns for the second and third nodes are located very close together), which is an interesting observation when Dark Silicon based optimisations are considered. Figure B.2 shows all of the divided clock nets for each node. It is clearly seen that the horizontal clock distribution nets are used and that most of the nodes are mapped in a similar fashion within their partitions. The effect of clustering of nodes around BRAM resources is also seen stretching down the entire device.

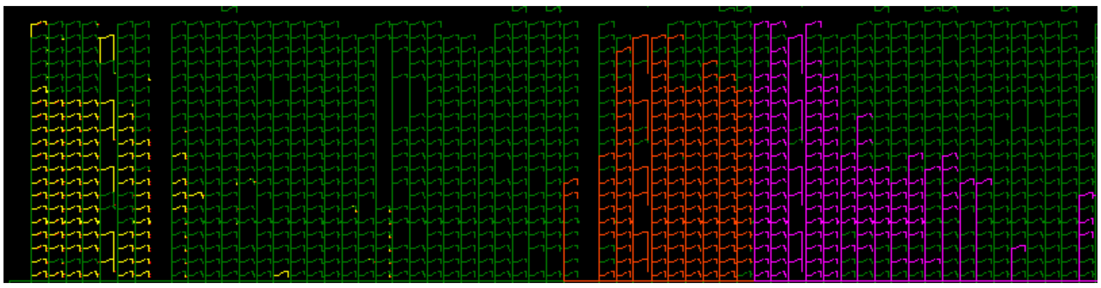


Figure B.1: A close up of three Centurion nodes and the NoC logic. The processing cores of each node is coloured as follows: node 0 - Yellow, node 1 - Orange and node 2 - Purple. NoC routers and logic is given in green. It shows the density of the design and how the nodes are centred around BRAM resources, potentially increasing the effect of processing core hot-spots.

Finally, Figure B.3a and B.3b show the node reset scheme with Figure B.3a showing the difference between the processing node resources reset from the RCAP register

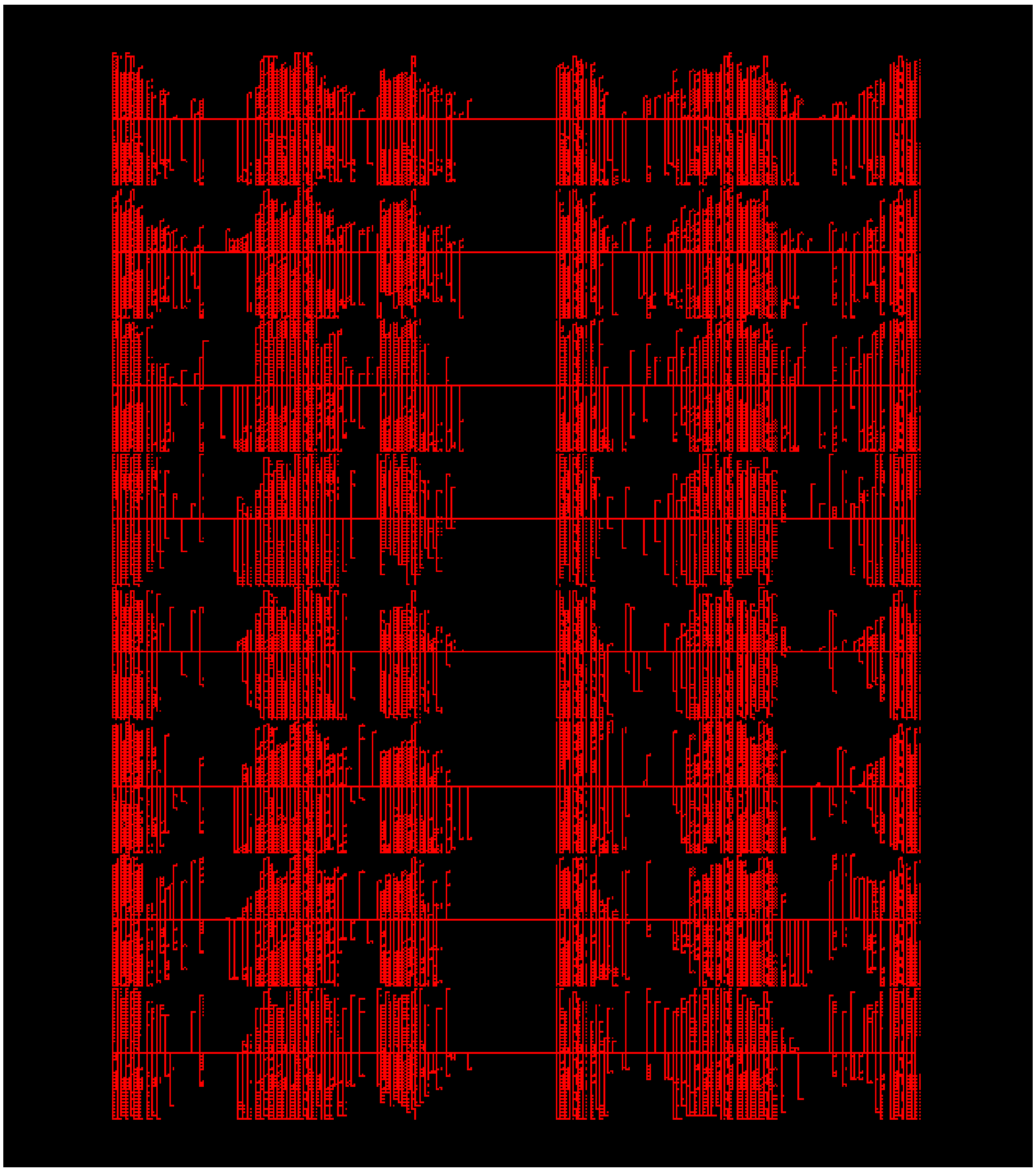
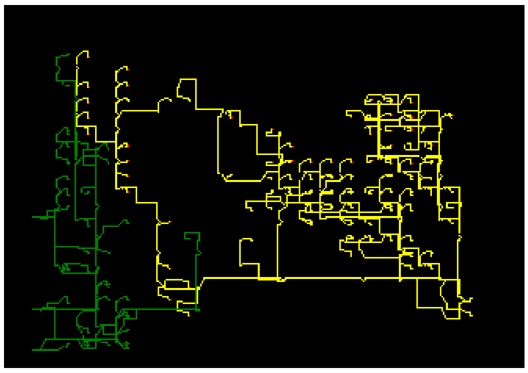
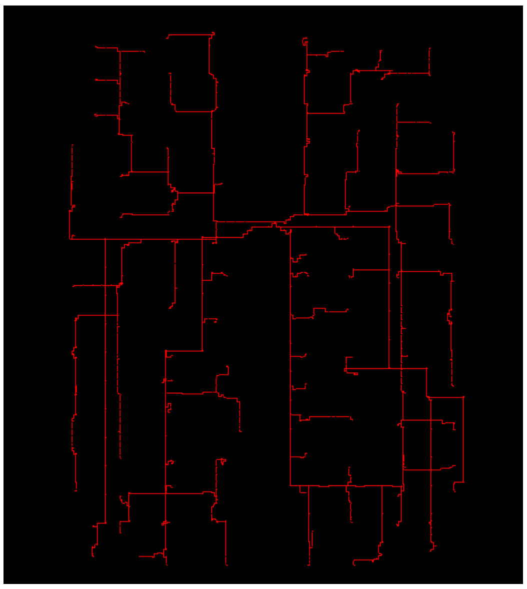


Figure B.2: Another post-implementation view showing the BUFH sourced clock nets that clock the processing cores of each node. The horizontal clock buffers within each clock region can clearly be seen. All red logic seen in this diagram is clocked by dynamic clock dividers.

within the router and the NoC router registers reset from the global reset. Figure B.3b shows that the fan-out of the global NoC reset has been managed through buffering of this signal; it also shows the location of each node at the endpoint of this signal.



(a) Resets within a node. This shows the processing node reset (green, driven from the RCAP) and the local NoC reset buffer. Use of a local registered version of the global reset help reduces the fan-out of the global signal and makes it far easier to place and route the NoC to meet the 100MHz timing constraint.



(b) The global NoC reset issued from the NoC interface on the experiment controller. This signal could have a fanout of hundreds of thousands of endpoints if not managed, making timing constraints very difficult to meet. The use of local buffering allows this signal to only have a direct fan-out of 128 endpoints.

Figure B.3: The reset architectures for Centurion. The reset is an important signal as with such a huge design it will have a huge fan-out and could result in nodes not coming out of reset correctly if the reset timing is ignored.

References

[1] International Roadmap for Devices, "Executive Summary 2018," International Roadmap for Devices (IRDS), Tech. Rep., 2019.

[2] ——, "More Moore 2018," International Roadmap for Devices (IRDS), Tech. Rep., 2019.

[3] J. Sartori, A. Pant, R. Kumar, and P. Gupta, "Variation-aware speed binning of multi-core processors," in *Proceedings of the 11th International Symposium on Quality Electronic Design, ISQED 2010*, 2010, pp. 307–314.

[4] J. H. Patel, "Cmos Process Variations: A Critical Operation Point Hypothesis," Stanford University, Tech. Rep., 2008.

[5] P. Bose, "Designing reliable systems with unreliable components," *IEEE Micro*, vol. 26, no. 5, pp. 5–6, 2006.

[6] D. M. Gordon, *Ant encounters: Interaction networks and colony behavior*. Princeton University Press, 2010, pp. 1–167.

[7] E. H. Davidson, "Emerging properties of animal gene regulatory networks," *Nature*, vol. 468, no. 7326, pp. 911–920, 2010.

[8] M. K. Gould and H. P. De Koning, "Cyclic-nucleotide signalling in protozoa," *FEMS Microbiology Reviews*, vol. 35, no. 3, pp. 515–541, 2011.

[9] J. B. Jackson, "A functional biology of clonal animals," *Trends in Ecology & Evolution*, vol. 5, no. 12, pp. 425–426, 1990.

[10] F. R. Prete, *Complex worlds from simpler nervous systems*. MIT Press, 2004.

[11] N. Goulding-Hotta, J. Sampson, G. Venkatesh, S. Garcia, J. Auricchio, P. C. Huang, M. Arora, S. Nath, V. Bhatt, J. Babb, S. Swanson, and M. B. Taylor, "The greendroid mobile application processor: An architecture for silicon's dark future," *IEEE Micro*, vol. 31, no. 2, pp. 86–95, 2011.

[12] T. Mattson, "The Future of Many Core Computing : A tale of two processors," Intel Labs, Tech. Rep. January, 2010, p. 79.

[13] H. Esmaeilzadeh and E. Blem, "Dark silicon and the end of multicore scaling," in *Proceedings of the 38th Annual International Symposium on Computer Architecture*, 2011, pp. 365–376.

[14] M. B. Taylor, "A landscape of the new dark silicon design regime," *IEEE Micro*, vol. 33, no. 5, pp. 8–19, 2013.

[15] ARM Limited, "big.LITTLE Technology: The Future of Mobile," ARM Ltd, Tech. Rep., 2013.

[16] R. Marculescu, U. Y. Ogras, L. S. Peh, N. E. Jerger, and Y. Hoskote, "Outstanding research problems in NoC design: System, microarchitecture, and circuit perspectives," English, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 1, pp. 3–21, 2009.

[17] P. Bremner, Y. Liu, M. Samie, G. Dragffy, A. G. Pipe, G. Tempesti, J. Timmis, and A. M. Tyrrell, "SABRE: A bio-inspired fault-tolerant electronic architecture," *Bioinspiration and Biomimetics*, vol. 8, no. 1, p. 016 003, 2013.

[18] A. J. Greensted, "A Reliability Engineered Multicellular Architecture Inspired by Endocrinology : The BioNode System," Ph.D. dissertation, University of York, 2004.

[19] A. Von Renteln, U. Brinkschulte, D. Kramer, W. Karl, C. Schuck, and J. Becker, "Digital on-demand computing organism - Interaction between monitoring and

middleware," *Proceedings - 2011 14th IEEE International Symposium on Object Component Service-Oriented Real-Time Distributed Computing, ISORC 2011*, pp. 189–196, 2011.

[20] J. Rossier, Y. Thoma, P. A. Mudry, and G. Tempesti, "MOVE processors that self-replicate and differentiate," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 3853 LNCS, pp. 160–175, 2006.

[21] C. Rouff and A. Vanderbilt, "Verification of NASA emergent systems," in *Engineering Complex Computer Systems, 2004. Proceedings. Ninth IEEE International Conference on*, 2004, pp. 231–238.

[22] J. Cross. (2019). "Inside Apple's A13 Bionic system-on-chip," [Online]. Available: `https://www.macworld.com/article/3442716/inside-apples-a13-bionic-system-on-chip.html` (visited on 2020-05-19).

[23] S. N. Beshers and J. H. Fewell, "Models of Division of Labor in Social Insects," *Annual Review of Entomology*, vol. 46, pp. 413–40, 2001.

[24] D. Gordon, B. Goodwin, and L. Trainor, "A Parallel Distributed Model of the Behaviour of Ant Colonies," *Journal of theoretical Biology*, 1992.

[25] R. Pagliara, D. M. Gordon, and N. E. Leonard, "Regulation of harvester ant foraging as a closed-loop excitable system," *PLoS Computational Biology*, vol. 14, no. 12, 2018.

[26] C. Tofts, "Algorithms for task allocation in ants. (A study of temporal polyethism: Theory)," *Bulletin of Mathematical Biology*, vol. 55, no. 5, pp. 891–918, 1993.

[27] M. Monchiero, J. H. Ahn, A. Falcón, D. Ortega, and P. Faraboschi, "How to simulate 1000 cores," in *2008 Workshop on Design, Analysis and Simulation of Chip Multiprocessors*, HP Laboratories, 2008.

[28] E. Salminen, A. Kulmala, and T. D. Hämäläinen, "Survey of Network-on-chip Proposals," Tampere University of Technology, Tech. Rep., 2008, 13 p.

[29] G. Moore, "Cramming more components onto integrated circuits," *Proceedings of the IEEE*, 1998.

[30] K. Roy, S. Mukhopadhyay, and H. Mahmoodi-Meimand, "Leakage current mechanisms and leakage reduction techniques in deep-submicrometer CMOS circuits," *Proceedings of the IEEE*, vol. 91, no. 2, pp. 305–327, 2003.

[31] R. Dennard and V. Rideout, "Design of ion-implanted MOSFET's with very small physical dimensions," *Solid-State Circuits, IEEE Journal of*, vol. 9, no. 5, pp. 256–268, 1974.

[32] International Technology Roadmap for Semiconductors. (2015). "International Technology Roadmap for Semiconductors - ITRS 2.0 Home Page," [Online]. Available: `http://www.itrs2.net/` (visited on 2020-04-13).

[33] International Roadmap for Devices. (2019). "International Roadmap for Devices - IRDS 2018 Edition Homepage," [Online]. Available: `https://irds.ieee.org/editions/2018` (visited on 2020-04-13).

[34] International Technology Roadmap for Semiconductors, "Executive Summary 2015," International Technology Roadmap for Semiconductors (ITRS), Tech. Rep., 2015.

[35] C. C. Lee, A. L. Palisoc, and Y. J. Min, "Thermal Analysis of Integrated Circuit Devices and Packages," *IEEE Transactions on Components, Hybrids, and Manufacturing Technology*, vol. 12, no. 4, pp. 701–709, 1989.

[36] J. Henkel, H. Khdr, S. Pagani, and M. Shafique, “New trends in dark silicon,” in *Proceedings - Design Automation Conference*, vol. 2015-July, New York, New York, USA: Institute of Electrical and Electronics Engineers Inc., 2015, pp. 1–6.

[37] A. Shafaei, Y. Xue, Y. Wang, P. Bogdan, S. Ramadurgam, and M. Pedram, “Analyzing the dark silicon phenomenon in a many-core chip multi-processor under deeply-scaled process technologies,” in *Proceedings of the ACM Great Lakes Symposium on VLSI, GLSVLSI*, vol. 20-22-May-, New York, New York, USA: Association for Computing Machinery, 2015, pp. 127–132.

[38] International Technology Roadmap for Semiconductors (ITRS), “Emerging Research Materials (ERM),” International Technology Roadmap for Semiconductors (ITRS), Tech. Rep., 2013, pp. 1–98.

[39] International Technology Roadmap for Semiconductors, “Emerging Research Devices Summary,” International Technology Roadmap for Semiconductors (ITRS), Tech. Rep., 2013.

[40] ——, “Process Integration, Devices and Structures Summary,” International Technology Roadmap for Semiconductors (ITRS), Tech. Rep., 2013.

[41] International Roadmap for Devices, “Metrology 2018,” International Roadmap for Devices (IRDS), Tech. Rep., 2018.

[42] F. a. Gerges, “The Rise and Fall of Dark Silicon,” *Usenix12*, vol. 37, no. 2, pp. 7–17, 2012.

[43] D. Belete, A. Razdan, W. Schwarz, R. Raina, C. Hawkins, and J. Morehead, “Use of DFT techniques in speed grading a 1GHz+ microprocessor,” in *IEEE International Test Conference (TC)*, 2002, pp. 1111–1119.

[44] J. Keane and C. H. Kim, “Transistor Aging,” *IEEE Spectrum*, pp. 1–5, 2011.

[45] H. T. Kung, “Why Systolic Architectures?” *Computer*, vol. 15, no. 1, pp. 37–46, 1982.

[46] A. L. DeCegama, *The Technology of Parallel Processing - Parallel Processing Architectures and VLSI Hardware*. Prentice Hall, 1989, vol. 1.

[47] Z. J. Czech, *Introduction to Parallel Computing*. Oxford University Press, 2016, p. 259.

[48] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design: The Hardware/Software Interface*. Elsevier, 2011, p. 703.

[49] A. Hemani, A. Jantsch, S. Kumar, A. Postula, and J. Öberg, “Network on a Chip : An architecture for billion transistor era,” in *IEEE NorChip Conference*, 2000.

[50] L. Benini and G. D. Micheli, “Networks on Chips : A New SoC Paradigm,” *Computer*, pp. 70–78, 2002.

[51] H. G. Lee, N. Chang, U. Y. Ogras, and R. Marculescu, “On-chip communication architecture exploration: A quantitative evaluation of point-to-point, bus, and network-on-chip approaches,” *ACM Transactions on Design Automation of Electronic Systems*, vol. 12, no. 3, 23–es, 2007.

[52] K. Srinivasan, K. S. Chatha, and G. Konjevod, “Linear programming based techniques for synthesis of Network-on-Chip architectures,” in *Proceedings - IEEE International Conference on Computer Design: VLSI in Computers and Processors*, IEEE, 2004, pp. 422–429.

[53] U. Y. Ogras and R. Marculescu, “Energy- and performance-driven NoC communication architecture synthesis using a decomposition approach,” in *Pro-*

ceedings -Design, Automation and Test in Europe, DATE '05, vol. I, IEEE, 2005, pp. 352–357.

[54] T. Bjerregaard and S. Mahadevan, "A survey of research and practices of network-on-chip," *ACM Computing Surveys*, vol. 38, no. 1, pp. 71–121, 2006.

[55] A. Agarwal, C. Iskander, and R. Shankar, "Survey of network on chip (NoC) architectures & contributions," Florida Atlantic University, Tech. Rep. 1, 2009.

[56] P. Mohapatra, "Wormhole Routing Techniques for Directly Connected Multicomputer Systems," *ACM Computing Surveys*, vol. 30, no. 3, pp. 374–410, 1998.

[57] S. Liu, A. Jantsch, and Z. Lu, "Analysis and evaluation of circuit switched NoC and packet switched NoC," in *Proceedings - 16th Euromicro Conference on Digital System Design, DSD 2013*, IEEE, 2013, pp. 21–28.

[58] E. Rijpkema, K. Goossens, A. Rădulescu, J. Dielissen, J. Van Meerbergen, P. Wielage, and E. Waterlander, "Trade-offs in the design of a router with both guaranteed and best-effort services for networks on chip," *IEE Proceedings: Computers and Digital Techniques*, vol. 150, no. 5 SPEC. ISS. Pp. 294–302, 2003.

[59] K. Goossens, J. Dielissen, and A. Rădulescu, "Æthereal network on chip: Concepts, architectures, and implementations," *IEEE Design and Test of Computers*, vol. 22, no. 5, pp. 414–421, 2005.

[60] A. Morgenshtein, I. Cidon, A. Kolodny, and R. Ginosar, "Comparative analysis of serial vs parallel links in NOC," in *2004 International Symposium on System-on-Chip Proceedings*, 2004, pp. 185–188.

[61] U. Y. Ogras, J. Hu, and R. Marculescu, "Key research problems in NoC design," *2005 Third IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS'05)*, p. 69, 2005.

[62] C. Ptolemaeus, *System Design, Modeling, and Simulation. Using Ptolemy II*, C. Ptolemaeus, Ed. Ptolemy.org, 2014, p. 690.

[63] A. K. Singh, P. Dziurzanski, H. R. Mendis, and L. S. Indrusiak, "A survey and comparative study of hard and soft real-time dynamic resource allocation strategies for multi-/many-core systems," *ACM Computing Surveys*, vol. 50, no. 2, pp. 1–40, 2017.

[64] C. Bolchini, M. Carminati, and A. Miele, "Self-adaptive fault tolerance in multi-/many-core systems," *Journal of Electronic Testing: Theory and Applications (JETTA)*, vol. 29, no. 2, pp. 159–175, 2013.

[65] H. Paul, "'Autonomic computing: IBM's perspective on the state of information technology', also known as IBM's Autonomic Computing Manifesto," IBM, Tech. Rep., 2001.

[66] C. V. Toller and S. V. Toller, *The nervous body: an introduction to the autonomic nervous system and behaviour*. John Wiley & Sons Ltd, 1979.

[67] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, 2003.

[68] J. Lala, "DARPA's path to self-regenerative systems," *42nd IFIP WG: Workshop on Dependability and Survivability*, 2002.

[69] S. Dobson, R. Sterritt, P. Nixon, and M. Hinchey, "Fulfilling the vision of autonomic computing," *Computer*, vol. 43, no. 1, pp. 35–41, 2010.

[70] W. F. Truszkowski, M. G. Hinchey, J. L. Rash, and C. A. Rouff, "Autonomous and autonomic systems: A paradigm for future space exploration missions,"

IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews, vol. 36, no. 3, pp. 279–291, 2006.

[71] J. R. Heath, P. J. Kuekes, G. S. Snider, and R. S. Williams, "A defect-tolerant computer architecture: Opportunities for nanotechnology," *Science*, vol. 280, no. 5370, pp. 1716–1721, 1998.

[72] A. M. Tyrrell, P. C. Haddow, J. Torresen, A. M. Tyrrell, E. Sanchez, D. Floreano, G. Tempesti, D. Mange, J.-M. Moreno, J. Rosenberg, and A. E. P. Villa, "POEtic Tissue: An Integrated Architecture for Bio-inspired Hardware," in *Evolvable Systems: From Biology to Hardware*, A. M. Tyrrell, P. C. Haddow, and J. Torresen, Eds., Springer Berlin Heidelberg, 2003, pp. 129–140.

[73] A. Upegui, Y. Thoma, E. Sanchez, A. Perez-Uribe, J. M. Moreno, J. Madrenas, and G. Sassatelli, "The PERPLEXUS bio-inspired hardware platform: A flexible and modular approach," *International Journal of Knowledge-Based and Intelligent Engineering Systems*, vol. 12, no. 3, pp. 201–212, 2008.

[74] A. F. Bourke, "Colony size, social complexity and reproductive conflict in social insects," *Journal of Evolutionary Biology*, vol. 12, no. 2, pp. 245–257, 1999.

[75] C. Anderson and D. W. McShea, "Individual versus social complexity, with particular reference to ant colonies," *Biological Reviews of the Cambridge Philosophical Society*, vol. 76, no. 2, pp. 211–237, 2001.

[76] L. Chittka and J. Niven, "Are Bigger Brains Better?" *Current Biology*, vol. 19, no. 21, R995–R1008, 2009.

[77] J. G. Burns, J. Foucaud, and F. Mery, "Costs of memory: Lessons from 'mini' brains," *Proceedings of the Royal Society B: Biological Sciences*, vol. 278, no. 1707, pp. 923–929, 2011.

[78] M. A. Changizi, *The Brain from 25000 Feet High; Level Explorations of Brain Complexity, Perception, Induction and Vagueness*. Springer, 2003, vol. Synthese L.

[79] N. R. Franks, S. C. Pratt, E. B. Mallon, N. F. Britton, and D. J. Sumpter, "Information flow, opinion polling and collective intelligence in house-hunting social insects," *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 357, no. 1427, pp. 1567–1583, 2002.

[80] G. Theraulaz, E. Bonabeau, and J. L. Deneubourg, "Response threshold reinforcement and division of labour in insect societies," in *Proceedings of the Royal Society B: Biological Sciences*, vol. 265, 1998, pp. 327–332.

[81] A. B. Sendova-Franks and N. R. Franks, "Spatial relationships within nests of the ant Leptothorax unifasciatus (Latr.) and their implications for the division of labour," *Animal Behaviour*, vol. 50, no. 1, pp. 121–136, 1995.

[82] A. Dornhaus, J. A. Holley, V. G. Pook, G. Worswick, and N. R. Franks, "Why do not all workers work? Colony size and workload during emigrations in the ant Temnothorax albipennis," *Behavioral Ecology and Sociobiology*, vol. 63, no. 1, pp. 43–51, 2008.

[83] D. Charbonneau, N. Hillis, and A. Dornhaus, "Lazy in nature: ant colony time budgets show high inactivity in the field as well as in the lab," *Insectes Sociaux*, vol. 62, no. 1, pp. 31–35, 2014.

[84] D. Charbonneau and A. Dornhaus, "Workers 'specialized' on inactivity: Behavioral consistency of inactive workers and their role in task allocation," *Behavioral Ecology and Sociobiology*, vol. 69, no. 9, pp. 1459–1472, 2015.

[85] ——, “When doing nothing is something. How task allocation strategies compromise between flexibility, efficiency, and inactive agents,” *Journal of Bioeconomics*, vol. 17, no. 3, pp. 217–242, 2015.

[86] D. Charbonneau, C. Poff, H. Nguyen, M. C. Shin, K. Kierstead, and A. Dornhaus, “Who Are the ”Lazy” Ants? The Function of Inactivity in Social Insects and a Possible Role of Constraint: Inactive Ants Are Corpulent and May Be Young and/or Selfish,” *Integrative and comparative biology*, vol. 57, no. 3, pp. 649–667, 2017.

[87] D. Charbonneau, T. Sasaki, and A. Dornhaus, “Who needs ‘lazy’ workers? Inactive workers act as a ‘reserve’ labor force replacing active workers, but inactive workers are not replaced when they are removed,” *PLoS ONE*, vol. 12, no. 9, 2017.

[88] C. Detrain and J. L. Deneubourg, “Scavenging by Pheidole pallidula: A key for understanding decision-making systems in ants,” *Animal Behaviour*, vol. 53, no. 3, pp. 537–547, 1997.

[89] Intel Corp. (2018). “Intel Core i9-7980XE Extreme Edition Processor,” [Online]. Available: `https://www.intel.co.uk/content/www/uk/en/products/processors/core/x-series/i9-7980xe.html` (visited on 2018-09-30).

[90] Mellanox Technologies, “TILE-Gx72 Processor,” Mellanox Technologies, Tech. Rep., 2015, p. 2.

[91] J. Gray, “GRVI Phalanx: A Massively Parallel RISC-V FPGA Accelerator Accelerator,” in *Proceedings - 24th IEEE International Symposium on Field-Programmable Custom Computing Machines, FCCM 2016*, IEEE, 2016, pp. 17–20. arXiv: `1606.01037`.

[92] X. Inc, “MicroBlaze Processor Reference Guide,” Tech. Rep., 2019.

[93] Intel Corp., “Nios II Processor Reference Guide; Nios II Processor Reference Guide,” Tech. Rep., 2019.

[94] A. Gaisler, “SPARC V8 32-bit Processor LEON3/LEON3-FT Companion-Core Data Sheet,” 2011.

[95] E. Matthews, L. Shannon, and A. Fedorova, “Shared memory multicore microblaze system with SMP Linux support,” *ACM Transactions on Reconfigurable Technology and Systems*, vol. 9, no. 4, pp. 1–22, 2016.

[96] J.-T. Kao, “Performance and Area Evaluations of Processor-based Benchmarks on FPGA Devices,” Tech. Rep., 2014.

[97] Altera Corp., “Creating Multiprocessor Nios II Systems Tutorial,” Tech. Rep. June, 2010.

[98] Xilinx Inc, “MicroBlaze Micro Controller System v1.4 (PG048),” Tech. Rep., 2013.

[99] ——, “PicoBlaze 8-bit Embedded Microcontroller User Guide for Extended Spartan ®-3 and Virtex ®-5 FPGAs Introducing PicoBlaze for Spartan-6, Virtex-6, and 7 Series FPGAs,” Tech. Rep., 2011.

[100] ECSS, “SpaceWire Standard - ECSS-E-ST-50-12C,” ECSS, Tech. Rep., 2012.

[101] Xilinx Inc. (2011). “UG628: Command Line Tools User Guide,” [Online]. Available: `http://www.xilinx.com/support/documentation/sw%7B%5C_%7Dmanuals/xilinx13%7B%5C_%7D2/devref.pdf` (visited on 2013-05-06).

[102] M. Rowlings, A. M. Tyrrell, and M. A. Trefzer, "Hardware implementation of Social-Insect-Inspired Adaptive many-core task allocation," in *2016 IEEE Symposium Series on Computational Intelligence, SSCI 2016*, Institute of Electrical and Electronics Engineers Inc., 2017.

[103] Xilinx Inc, "Virtex-6 FPGA Configurable Logic Block User Guide," Xilinx Inc, Tech. Rep., 2012.

[104] ——, "Xilinx UG362 Virtex-6 FPGA Clocking Resources User Guide," Xilinx Inc, Tech. Rep., 2014.

[105] K. (L. Chapman, "Frequency Counter for Spartan-3E Starter Kit (with test oscillators)," Xilinx Inc, Tech. Rep. March, 2006, pp. 1–11.

[106] Xilinx Inc, "Virtex-6 FPGA Electrical Characteristics Virtex-6 FPGA DC Characteristics Virtex-6 FPGA Data Sheet: DC and Switching Characteristics," Xilinx Inc, Tech. Rep., 2009.

[107] ——, "Virtex-6 Family Overview," Xilinx Inc, Tech. Rep., 2009.

[108] P. J. Simmons and D. Young, *Nerve cells and animal behaviour*, 3rd. Cambridge University Press, 2010, pp. 1–284.

[109] L. Chin Wei and A. Long. (2015). "Synchronous Counters - Final Report," [Online]. Available: `http://www.doc.ic.ac.uk/%7B~%7Dnd/surprise%7B%5C_%7D96/journal/vol4/cwl3/report.html%7B%5C#%7Dcompare` (visited on 2018-09-30).

[110] Xilinx Inc, "Xilinx Constraints Guide (UG625)," Xilinx Inc, Tech. Rep., 2013.

[111] Y. K. Kwok and I. Ahmad, "Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 5, pp. 506–521, 1996.

[112] P. E. Black. (2019). "Manhattan Distance," [Online]. Available: `https://xlinux.nist.gov/dads/HTML/manhattanDistance.html` (visited on 2020-04-21).

[113] D. M. Gordon, "The organization of work in social insect colonies," *Complexity*, vol. 8, no. 1, pp. 43–46, 2002.