Response Time Analysis for Real-Time Communications on Networks-On-Chips

Undergraduate Thesis

Submitted in partial fulfillment of the requirements of
BITS F421T Thesis

By

Parv Dani
ID No. 2016A7TS0722G

Under the supervision of:

Dr. Alan Burns
&
Dr. Shubhangi Gawali

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI, GOA CAMPUS
December 2019

Declaration of Authorship

I, Parv Dani, declare that this Undergraduate Thesis titled, 'Response Time Analysis for Real-Time Communications on Networks-On-Chips' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: Parv D. .

Date: 07TH DECEMBER, 2019

Certificate

This is to certify that the thesis entitled, "*Response Time Analysis for Real-Time Communications on Networks-On-Chips*" and submitted by Parv DANI ID No. 2016A7TS0722G in partial fulfillment of the requirements of BITS F421T Thesis embodies the work done by him under my supervision.

[illegible]

Supervisor	*Co-Supervisor*
Dr. Alan BURNS	Dr. Shubhangi GAWALI
Professor,	Professor,
University of York, UK	BITS-Pilani Goa Campus
Date: 9 Dec 2019.	Date:

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI, GOA CAMPUS

Abstract

Bachelor of Engineering (Hons.)

Response Time Analysis for Real-Time Communications on Networks-On-Chips

by Parv Dani

We are quickly moving towards Systems-On-Chips (SoCs) because of its speed and energy efficiency. Using Network-On-Chips for these systems provides efficient use of multi-cores. The response time analysis for mixed-criticality systems on a Network-On-Chip (NoC) has been continually talked about for tasks at the cores and for messages at the links. This report aims to integrate the analysis of both these sections and develop response time analysis for a complete flow through the network on a system.

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Dr. Alan Burns, who had the faith to provide me with this opportunity. He was a constant guide through each and every step and without his supervision, this project would not have been possible.

A huge thanks to my co-supervisor, Dr. Shubhangi Gawali, who laid the technical foundation required for this project. She is responsible for planting the interest in me towards the field of real-time systems. I used the theoretical knowledge she developed in every aspect of this research.

I sincerely appreciate Dr. Leandro S. Indrusiak for introducing me to embedded systems and genetic algorithms, and more importantly, giving direction to my research. I also thank James Harbin for helping me with the minute technical details which helped the research become a success. A huge thanks to Dr. Biju K. Raveendran who took my advanced course on real-time systems and helped me with the required knowledge.

Finally, I would like to acknowledge my parents, Dr. Pravin Dani and Dr. Ruchi Dani, for motivating me through this journey and supporting me through life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Real-time systems involve a broad spectrum of increasingly complex applications. Ranging from autonomous vehicles to advanced communication technologies, each of these applications require tasks to be carried out in limited time. The quest to find the perfect balance between time and energy efficiency continues to intrigue researchers. The embedded systems domain is now shifting towards the System-on-Chips (SoCs) to find this perfect balance. The use of SoCs have led to improvements in cost, power generation, heat dissipation, space and weight. According to [5], the system configurations that affect real-time behaviour are:

- Task allocation to each core.
- Communication routing between interconnected cores.
- Memory allocation for a task to store its buffers.
- Scaling voltage and frequencies assigned to each processor.

To find optimal solutions to all these configurations is unlikely. Unfortunately, for a moderately complex system, it is already infeasible to optimize even one of these configurations. In this report, we focus on the first two configurations, i.e. allocation of tasks to a core and the path traveled by messages on links that connect these cores. As stated by [7], rearranging tasks in a multi-processor system is a NP-complete problem. Thus, we need heuristic improvements to generate close to optimal solutions for the same.
The improvements for these configurations begin with a proper time analysis for processes occurring in the system. Burns et al, [4] and Baruah et al, [1] have presented an analysis of response times at links and cores respectively for tasks. We will call a process that travels its defined links and cores as a **transaction**. This project aims to generate response time equations for each transaction, find the worst case response time and finally compare different scenarios of varying parameters.

1.1 Theory

This section provides basic theoretical concepts involved in this report. Real-time systems, in general, are systems that contain tasks with bounded deadlines. These systems can either be soft, firm or hard. Soft systems allow a few deadline misses, which results in reduced efficiency but no severe harm to the system. Firm systems allow a particular number of deadline misses, and act as hard after this number is achieved. Hard real-time systems have strong deadlines - missing these result in catastrophic failures. While analysing hard real-time systems, it is made sure that all the tasks are schedulable, and not even one task misses its deadline.

Real-time tasks can essentially be of three types: periodic, aperiodic and sporadic. **Periodic** tasks are those bound by a certain period. They are repeatedly released after every period. They can have deadlines less than, equal to or greater than their periods. For the sake of simplicity, this report deals only with the cases where periods are equal to deadlines. **Aperiodic** tasks are those who have irregular arrival times. Aperiodic tasks with hard deadlines are **sporadic** tasks.

A typical real-time system has certain characteristics. These characteristics include the following control facilities for real-time programmers:

- To specify when actions are performed
- To specify by when the actions are to be completed
- To program periodic, aperiodic and sporadic tasks
- To control jitter times on input and output operations
- To respond to situations when some timing requirements can not be met
- To respond to dynamically changing conditions within a system. For example, a mode change in a mixed-criticality system.

The tasks in real-time systems can be scheduled using several algorithms. The one necessary to understand for this report is the fixed priority scheduling with deadline-monotonic priority assignment. This algorithm is based on a simple concept of giving higher priorities to tasks with earlier deadlines. Thus, the tasks with tighter deadlines are run before others.

Criticality, as defined by [2], is a designation of the level of assurance against failure needed for a system component. A mixed criticality system (MCS) is one that supports more than one criticality levels. MCS have an additional characteristic of supporting mode change during task execution, which helps prioritize higher critical tasks to meet their requirements. According to

ISO 26262 standards, there are five different levels of criticailty defined. This report deals with two levels of criticality, high and low. High critical (HI-crit) tasks are defined as those that need to be executed with the highest preference once the system goes into the HI-crit mode. Low critical (LO-crit) tasks are those that execute normally when the system is in a LO-crit mode, but are no longer executed once the system moves into a HI-crit mode. It is generally stated that for a given task, its worst case execution time (WCET) in a HI-crit mode is higher than its WCET in a LO-crit mode, on the same core.

1.2 Outline

Section 1 of the report starts with a basic introductory theory about real-time systems and the concepts used for the proposed response-time analysis.

Section 2 defines the multi-core model considered to write the equations and to run the simulations. Section 2.2 presents the nomenclature used and Section 2.3 describes how the model reacts in different situations.

Section 3 presents the equations at the cores and the links. Section 4 contains a simulation run to compare results obtained by changing parameters.

Finally, section 5 presents the conclusion and the future work to be done on this report.

Chapter 2

The Model

2.1 Defining the Multi-core Model

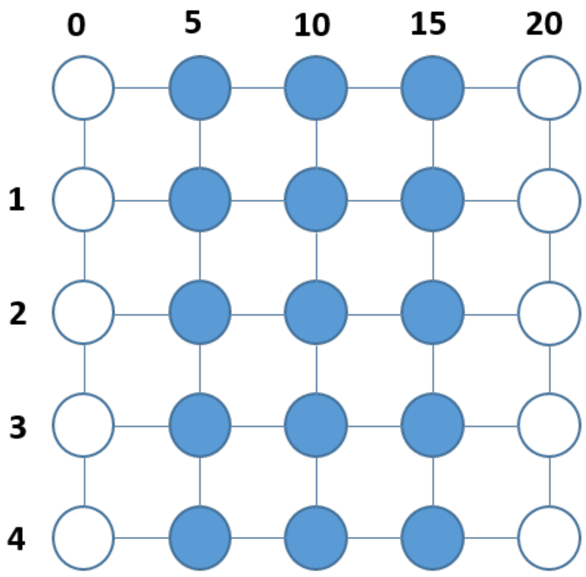


Figure 2.1: System Model

The model consists of a 5×5 matrix of cores (Figure 2.1). The cores are connected by links. For each transaction, we have one input core, one main computation core, and one output core (Figure 2.2). Corresponding to figure 1, the input core can be one of cores 0-4. The computation core can be one of cores 5-19. The output core can be one of cores 20-24. Each core and each link can be either in low critical (LO-crit) mode or in high critical (HI-crit) mode. The basic core-router architecture can be found in [3].

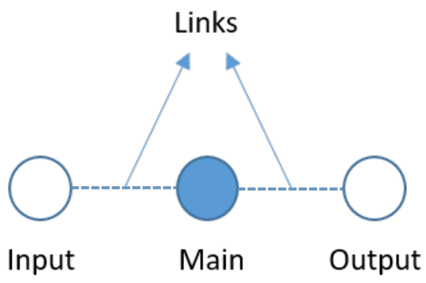
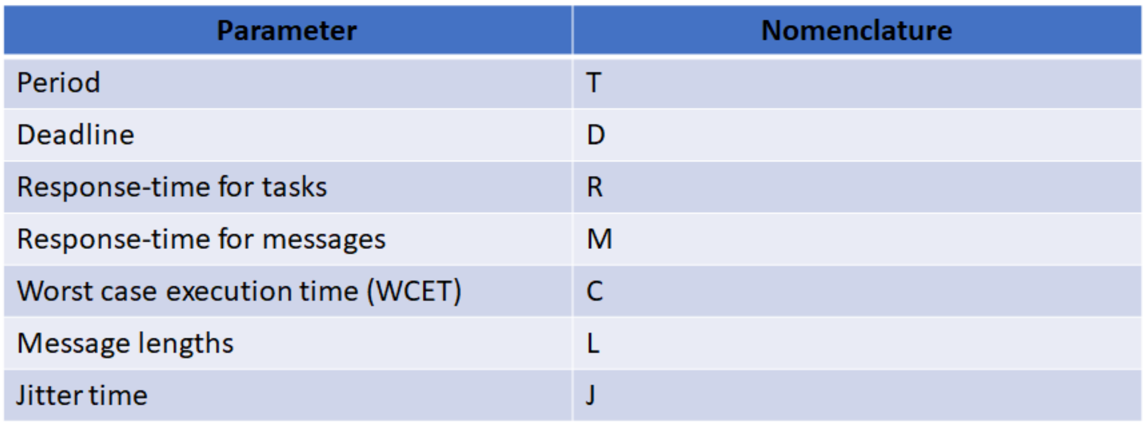


FIGURE 2.2: Simple Transaction

2.2 Nomenclature

For a transaction i, the parameters defined are:



Parameter	Nomenclature
Period	T
Deadline	D
Response-time for tasks	R
Response-time for messages	M
Worst case execution time (WCET)	C
Message lengths	L
Jitter time	J

TABLE 2.1: Nomenclature

2.3 Describing the Multi-core Model

Before stating the response time equations, we briefly describe how our model reacts to different scenarios. A mode change at a core occurs when either it receives a larger message, or as a response to the environment. Note that at any point of time, the mode of each core and each link is known. It has also been proved that $C(HI) \geq C(LO)$ and $J(HI) \geq J(LO)$. In our analysis, it is assumed that jitter times are equal for both modes.
A LO-crit transaction, in a LO-crit mode continues as a normal flow, and is stopped in case of a mode change. At the input core a HI-crit transaction generates four cases:

1. The execution time corresponds to HI-crit and it generates a larger message (which corresponds to HI-crit).

2. The execution time corresponds to HI-crit and it generates a smaller message (which corresponds to LO-crit).
3. The execution time corresponds to LO-crit and it generates a larger message (which corresponds to HI-crit).
4. The execution time corresponds to LO-crit and it generates a smaller message (which corresponds to LO-crit).

Case 1 is the case of a HI-crit flow through the core. Assuming that more work at the core results in a larger message, case 2 is rejected. Case 3 is a mode change from LO-crit to HI-crit occurring at the core. Case 4 is a normal LO-crit flow.
The main cores receive the message generated by the input cores, process them, and send a message to the output core. As per the length generated at the input cores, each of the above 4 cases can either receive a small or a large message. As nomenclature, we name cases 1-4 as corresponding cases that receive a smaller message, and cases 5-8 as those cases that receive a larger message.
Case 1 and 3 are those where a mode change occurs at the core. Case 2 is appropriately rejected. Case 4 is a normal LO-crit flow. Among the next 4 cases, assuming that the mode change occurs once a larger message is received, cases 6,7 and 8 are rejected. Case 5 is that of a HI-crit flow.

Chapter 3

Response Time Analysis

The equation without considering mixed-criticality as stated by [2] for task processing at each core is given by:

$$R_i \ = \ C_i \ + \sum_{\tau_j \in \mathbf{Shp}(i)} \left\lceil \frac{R_i + J_j^D}{T_j} \right\rceil C_j$$

Where $\mathbf{Shp}(i)$ is the set of all tasks at that core with a priority higher than that of i. The equation as stated by [4] for messages at each link is given by:

$$M_i \ = \ L_i \ + \sum_{\tau_j \in \mathbf{Shp}(i)} \left\lceil \frac{M_i + J_j^D + J_j^I}{T_j} \right\rceil L_j$$

An extra indirect jitter term is added due to the interference that the higher priority messages face due to messages on other links.
These equations are solved in a recursive manner. First, we start with the response time ($R(0)$) equal to 0. Then, we put this value on the right hand side of the equation and calculate $R(1)$. We continue for further iterations until $R(n) = R(n-1)$ is obtained. This is the solution of the equation.
Moving to the mixed-criticality system, several cases have to be analysed.

3.1 Analysis for Cores

Following are the cases at each core:

1. For a normal LO-crit flow, the above response time equations are used, but with parameters corresponding to the LO-crit mode. The equation at each core becomes:

$$R_i^1(LO) \;=\; C_i(LO) \;+ \sum_{\tau_j \in \mathbf{ShpL}(i)} \left\lceil \frac{R_i^1(LO) + J_j^D}{T_j} \right\rceil C_j(LO)$$

2. For a task which supports HI-crit mode, there is an interference from the higher priority HI-crit tasks and an interference from the higher priority LO-crit tasks which were running before the mode change. For the worst case scenario, it can be assumed that the mode change occurs after all the higher priority tasks run. Baruah et al, [1] have proved that using a rather optimistic approach is not very effective. The response-time equation for mode change at a core is given by:

$$R_i^2(HI) \;=\; C_i(HI) \;+ \sum_{\tau_j \in \mathbf{ShpH}(i)} \left\lceil \frac{R_i^2(HI) + J_j^D}{T_j} \right\rceil C_j(HI) + \sum_{\tau_j \in \mathbf{ShpL}(i)} \left\lceil \frac{R_i^1(LO) + J_j^D}{T_j} \right\rceil C_j(LO)$$

3. For a core which receives a large sized message (the task already corresponding to a HI-crit mode), the LO-crit tasks will not run at all. The response-time equation at the core is given by:

$$R_i^3(HI) \;=\; C_i(HI) \;+ \sum_{\tau_j \in \mathbf{ShpH}(i)} \left\lceil \frac{R_i^3(HI) + J_j^D}{T_j} \right\rceil C_j(HI)$$

Baruah et al [1] prove that R_i^2 has the largest value among the three cases. Hence, to verify whether the system meets its deadlines, we only need to check if the sum of R_i^2 on all the three cores (adding with the response time at the links) meets its deadline.

3.2 Analysis for Links

The indirect jitters (J_j^I) are considered while writing response time equations for the links. It is given by:

$$J_j^I = R_j - C_j$$

Following are the cases at each link:

1. For a HI-crit flow already in a HI-crit mode, the equation is given by:

$$M_i^1(HI) \;=\; L_i(HI) \;+ \sum_{\tau_j \in \mathbf{ShpH}(i)} \left\lceil \frac{M_i^1(HI) + J_j^D(HI) + J_j^I(HI)}{T_j} \right\rceil L_j(HI)$$

2. For a flow running in LO-crit mode which faces interference from LO-crit higher priority messages and direct interference from other HI-crit flows, the equation is given by:

$$
\begin{aligned}
M_i^2(LO) \;=\; L_i(LO) \;&+ \sum_{\tau_j \in \mathbf{ShpH}(i)} \left\lceil \frac{M_i^3(LO) + J_j^D(HI) + J_j^I(HI)}{T_j} \right\rceil L_j(HI) \\
&+ \sum_{\tau_j \in \mathbf{ShpUL}(i)} \left\lceil \frac{M_i^3(LO) + J_j^D(HI) + J_j^I(HI)}{T_j} \right\rceil L_j(LO) \\
&+ \sum_{\tau_j \in \mathbf{ShpDL}(i)} \left\lceil \frac{M_i^2(HI) + J_j^D(HI) + J_j^I(LO)}{T_j} \right\rceil L_j(LO)
\end{aligned}
$$

3. For a flow running in LO-crit mode which faces interference from LO-crit higher priority messages can face increased indirect interference from other HI-crit flows in the system. The equation for response time according to [4] is given by:

$$
M_i^3(LO) \;=\; L_i(LO) \;+ \sum_{\tau_j \in \mathbf{ShpL}(i)} \left\lceil \frac{M_i^3(LO) + J_j^D(LO) + J_j^I(HI)}{T_j} \right\rceil L_j(LO)
$$

Burns et al [4] conclude that the maximum of these 3 response times can be used to calculate the final response time of the transaction in the system.
Note that this is the analysis at each core and at each link. So to get the final worst case response time for the whole transaction, we add the maximum individual response times at each node and each link. The approach is quite pessimistic but extremely safe.
Thus, the final response time is given by:

$$
R_i^{total} = max(R_i^{input}) + max(R_i^{main}) + max(R_i^{output}) + \sum_{i \in links} max(M_i)
$$

Chapter 4

Simulation

To verify the credibility of the model, several simulations were carried out with varying parameter ranges for the model. Following is a list of assumptions:

1. The cores are exactly similar in characteristics.
2. All the tasks are periodic and the deadlines of tasks are equal to periods. $(D_i = T_i)$
3. The priorities at each core and each link are assigned based on the fixed priority scheduling with deadline monotonic priority assignment, i.e., the task with an earlier deadline will have a higher priority.
4. The system consists of hard deadlines only, i.e., no deadline misses are allowed.
5. The routing algorithm used is that of the shortest path, wherein the transaction first moves horizontally, then moves vertically.
6. The direct jitter values are considered to hold the value 1. This value is taken for ease of computation and can be changed if necessary.

The tables show the parameter ranges and the figures show a graphical comparison of the results that follow.

4.1 Increasing Periods

Each of these graphs have the Y-axis as **schedulability**. For this report, each set of parameter values were run a hundred times and the number of successful (all transactions meeting their deadlines) runs were recorded as the schedulability percentage.

The randomized parameterization means any of the values from the given ranges can be randomly selected. This also includes selecting a random input, main and output core.

Parameters	Values/Ranges of Values
Probability of HI-crit	0.5
Message length (LO)	2
Message Length (HI)	4
I/O computation time (LO)	3-5
I/O computation time (HI)	7-12
Main computation time (LO)	5-7
Main computation time (HI)	12-16
WCET Ratios (HI:LO)	0.25-0.7

TABLE 4.1: Increasing periods

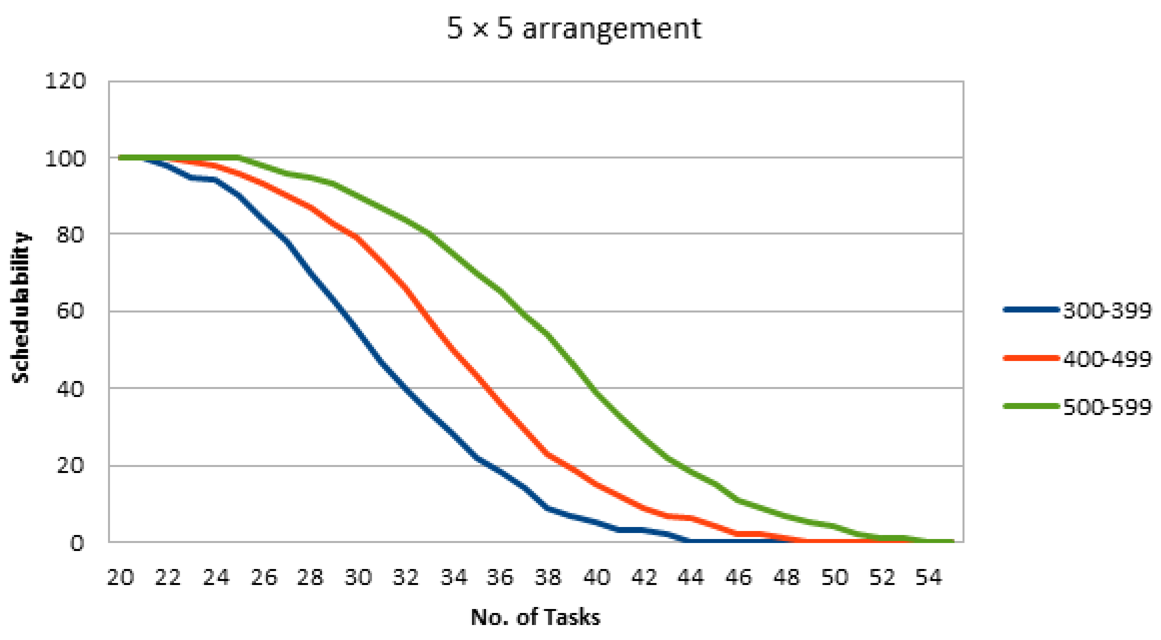


FIGURE 4.1: Increasing periods

The graph shows an approximate 27 percent increase in the schedulability when we increase the period range from 300-399 to 400-499 and an approximate 34 percent increase in the schedulability when we increase the period range from 400-499 to 500-599.

4.2 Increasing Cores

Parameters	Values/Ranges of Values
Probability of HI-crit	0.5
Message length (LO)	2
Message Length (HI)	4
I/O computation time (LO)	3-5
I/O computation time (HI)	7-12
Main computation time (LO)	5-7
Main computation time (HI)	12-16
WCET Ratios (HI:LO)	0.25-0.7

TABLE 4.2: Increasing cores

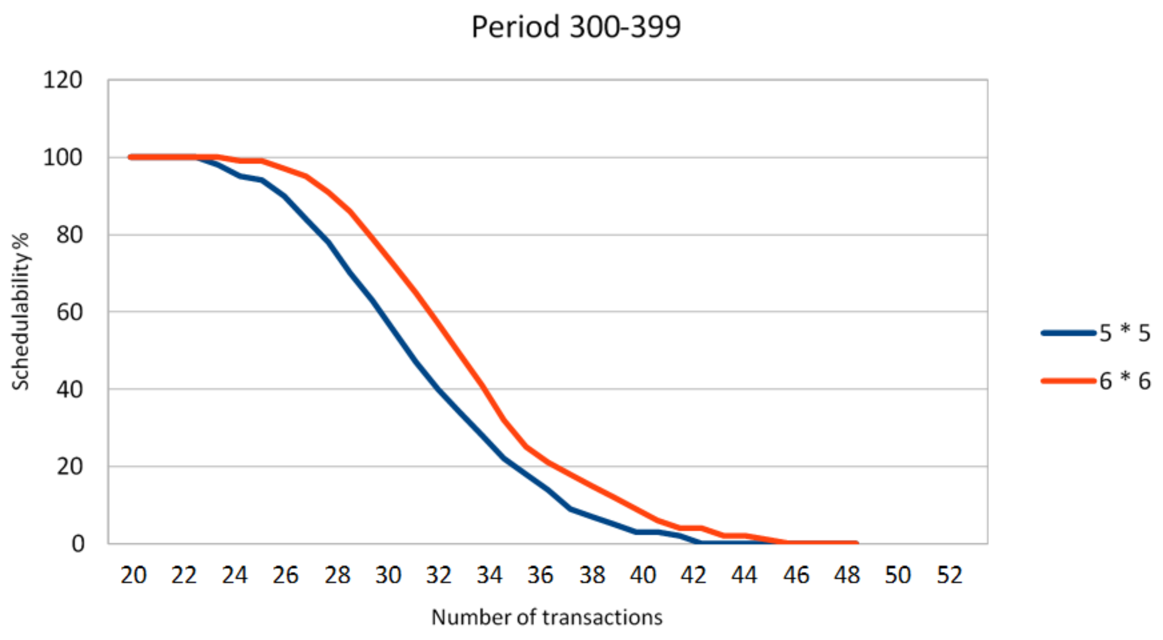


FIGURE 4.2: Increasing cores

Figure 4.2 shows approximately a 16 percent increase in schedulability as we increase the number of cores from 5×5 matrix to a 6×6 matrix.

Figure 4.3 shows approximately a 22 percent increase in schedulability as we increase the number of cores from 5×5 matrix to a 6×6 matrix.

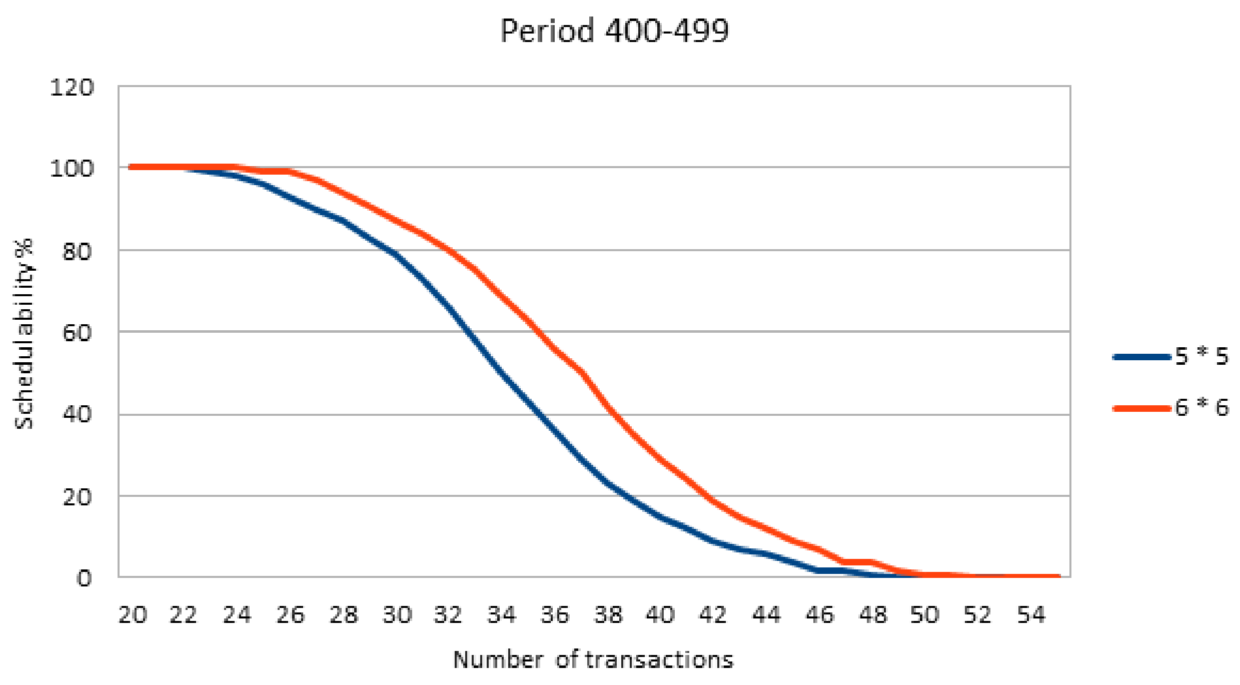


FIGURE 4.3: Increasing cores

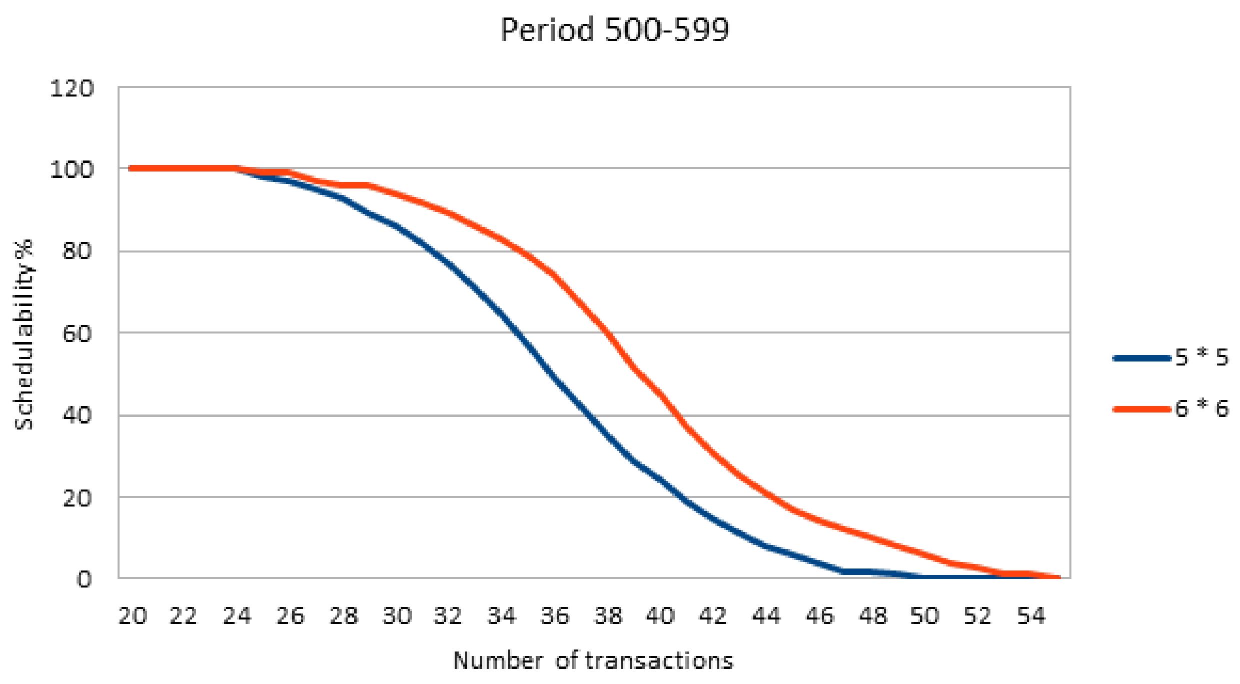


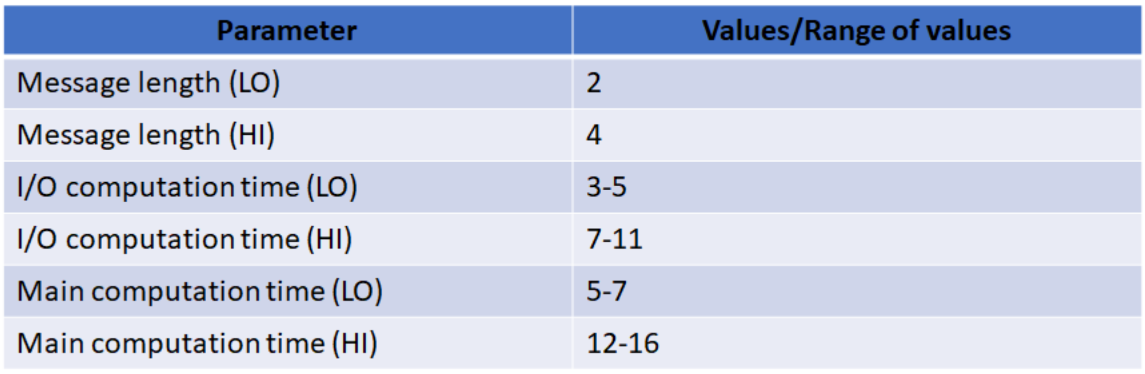
FIGURE 4.4: Increasing cores

Figure 4.4 shows approximately a 29 percent increase in schedulability as we increase the number of cores from 5×5 matrix to a 6×6 matrix.

Increasing the number of cores increases the energy consumption drastically. However, our results show the manner in which the schedulability rises. The effect of increasing the number of cores is much higher when we move to higher period ranges. The future work includes finding the most effective balance between the period ranges and the number of cores to get highest schedulability and energy efficiency.

4.3 Increasing HI-crit Probability

In this section, the probability of a flow to be in a HI-crit mode was made to vary keeping other parameters constant.



Parameter	Values/Range of values
Message length (LO)	2
Message length (HI)	4
I/O computation time (LO)	3-5
I/O computation time (HI)	7-11
Main computation time (LO)	5-7
Main computation time (HI)	12-16

TABLE 4.3: Increasing HI-crit probability

The results show a decreasing trend in schedulability as the probability of a transaction to be in HI-crit mode is increased. This should be because of the reduced interference caused by the transactions flowing in a HI-crit mode.

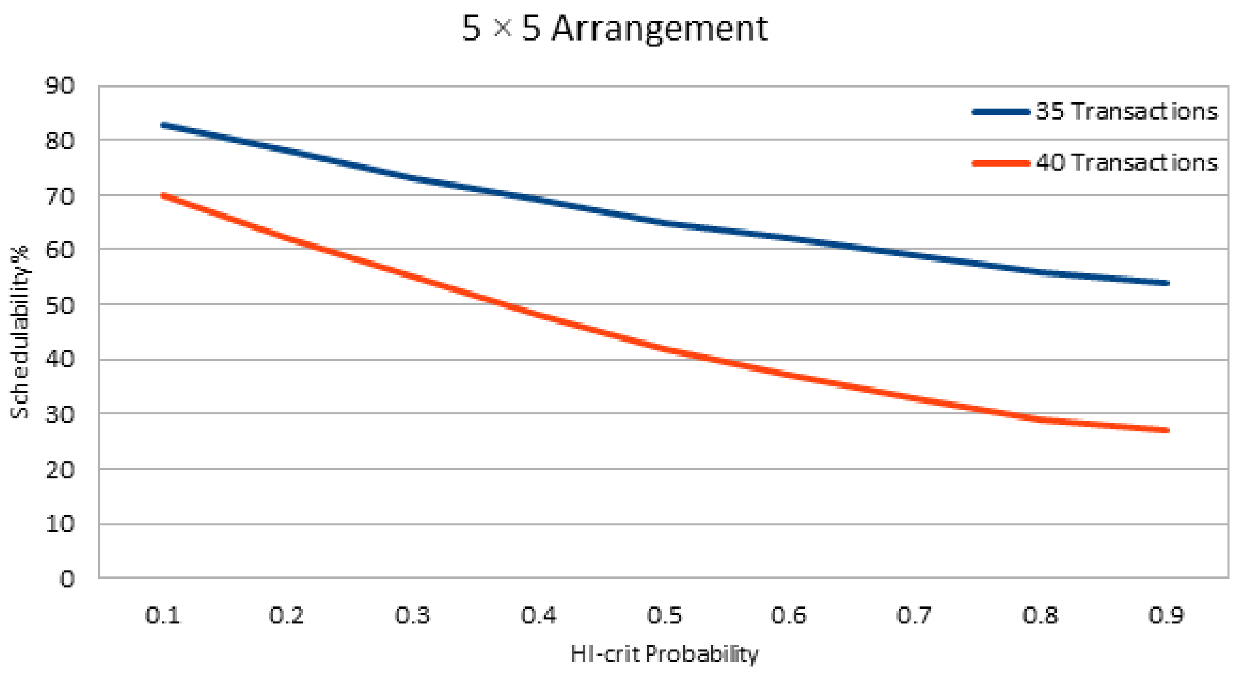


FIGURE 4.5: Increasing HI-crit probability

4.4 Increasing LO:HI WCET Ratios

In this section, we tried to increase the LO:HI computation time ratios at the main cores. To maintain uniformity, the sum of low and high times have been taken constant, i.e., equal to 20.

Parameter	Values/Range of values
Message length (LO)	2
Message length (HI)	4
I/O computation time (LO)	3-5
I/O computation time (HI)	7-11
HI-crit Probability	0.5

TABLE 4.4: Increasing LO:HI WCET ratios

The results show a slight increase in the schedulability when the values of low and high computation times are closer to each other. This trend is expected as the sum of execution times are kept constant which lowers the value of HI-crit computation times, thus reducing the interference.

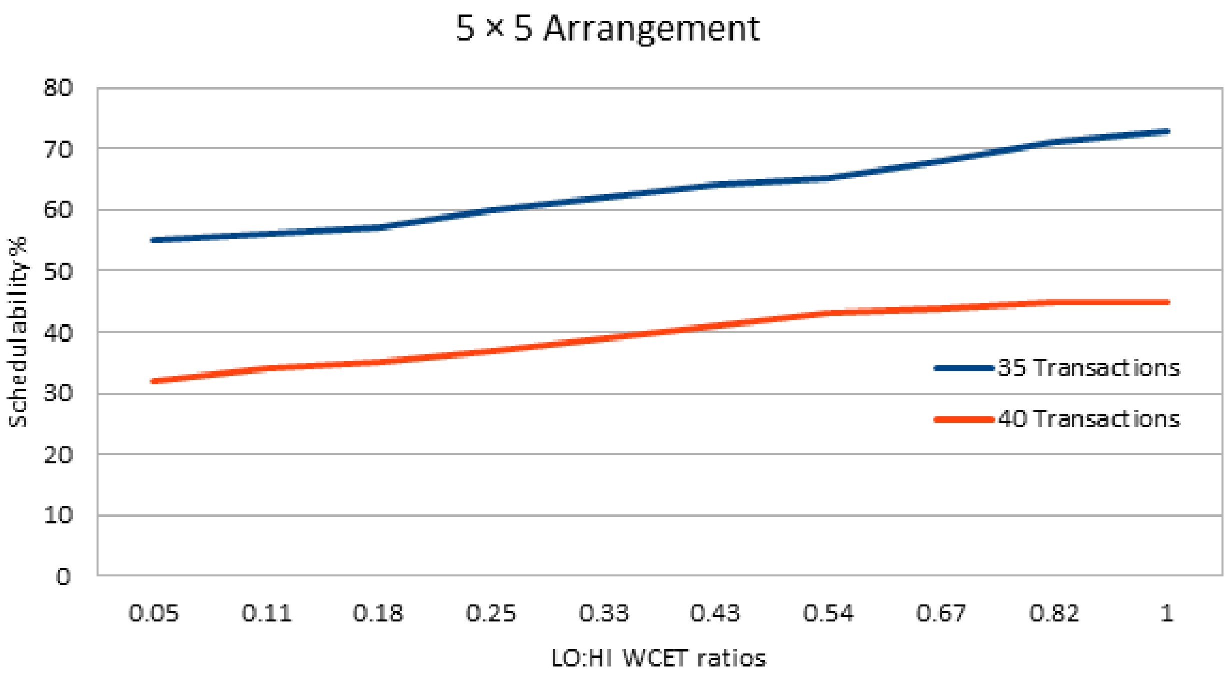


FIGURE 4.6: Increasing LO:HI WCET ratios

4.5 Heuristic Development

We also developed a heuristic path allocation algorithm to check how the system behaves in a more predictable manner. The tests run on a 5×5 matrix and are only to check whether the algorithm is an improvement, and so are carried out without considering mixed-criticality of tasks. However, the results can be extended to mixed-critical systems.
In the heuristic algorithm, the parameters corresponding to periods, execution times and message lengths are still picked randomly to maintain uniformity and comparable results. However, the path allocation is selective. The following is the algorithm for a n×n (n=5 in this model) to choose the path:

- Tasks are arranged in an increasing order of priority, i.e, the highest priority task is kept first.
- The first task is allocated to core 0, second to core 1 and so on till task n. Task n+1 is allocated to core n-1, task n+2 to core n-2 and so on till task 2n. Further tasks are allocated in the same vertical pattern to provide homogeneity among the priorities.
- Now, for the first horizontal row of the network, which contains tasks 0, 2n-1, 2n, 4n-1 and so on, the cores are distributed in the same way, but horizontally. Core n gets task 0, core 2n gets task 2n-1 and so on for each main computation cores.
- Finally all the tasks of the row are allocated to the corresponding output processor.

This algorithm tries to reduce the interference from different messages. And because of a homogeneous distribution, the interference from the main cores is evened out. This does increase the interference at some potentially less crowded cores, but reduces it at the heavily crowded cores. Thus, considering cases where all the tasks need to meet their deadline, improved results should be obtained. The results obtained are:

Parameters	Values/Ranges of Values
Period	90-140
Message length	1-2
I/O computation time	1-2
Main computation time	4-6
Cores	25

TABLE 4.5: Parameters for Randomized vs Heuristic

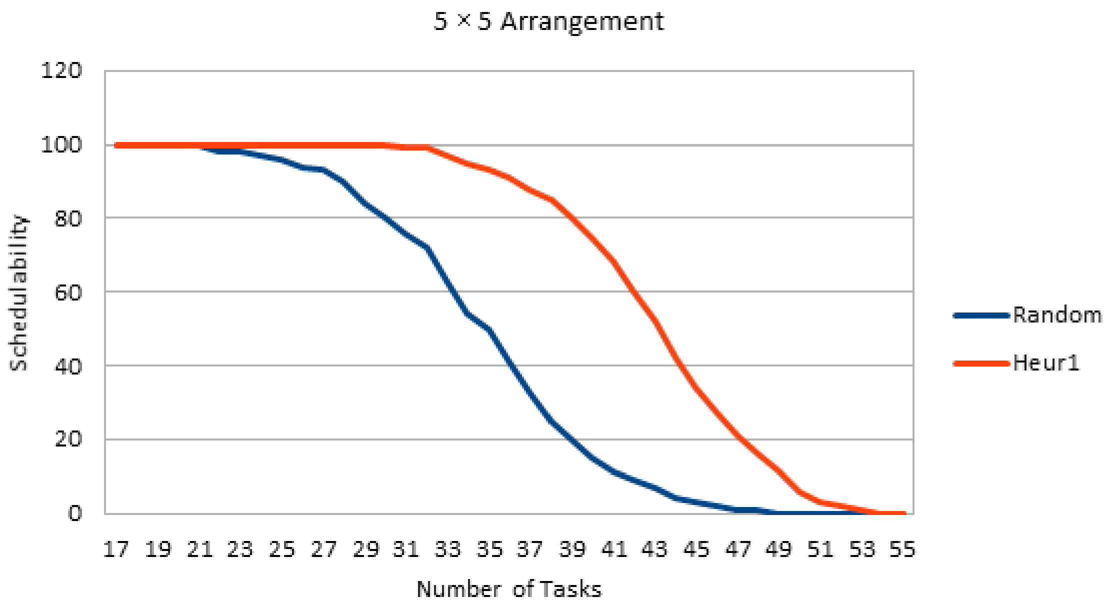


FIGURE 4.7: Randomized vs Heuristic

The resulting graph shows an approximate 40 percent improved schedulability for the heuristically developed model. The drawbacks for this model might be seen in extreme cases when a few tasks have much tighter deadlines than other tasks. Though, the algorithm is expected to give similar results when applied to the mixed-criticality systems.

Chapter 5

Conclusions and Future Work

The research has tried to develop a more thorough and realistic analysis of the response-times in a Network-on-Chip system. The simulation carried out gave results which showed how changing parameters affect the schedulability of the system in our analysis. It is an extension of [4] and [1].

Towards the end, a heuristic development to the basic task allocation is proposed and proved to produce significantly better results compared to the randomized algorithm. This algorithm can further be compared to several other results produced by using improved algorithms, such as genetic algorithms.

The model described and used for simulations in the report have a few assumptions which need to be catered. Future work will address these assumptions and extend the model in the following way:

1. Multiple criticality levels will be included. All the tasks can not be divided into either high or low criticality levels.
2. Genetic algorithms will be applied to obtain efficient task schedules over the network. The scheduling problem being NP-hard needs heuristic algorithms for its development making genetic algorithms a useful tool. The work done by [5] will be extended.
3. A proper mode change protocol will be developed to bring the state back to the low criticality state.
4. Some case studies, possibly one automotive and one avionics, will be undertaken.

Bibliography

[1] Sanjoy K Baruah, Alan Burns, and Robert I Davis. "Response-time analysis for mixed criticality systems". In: *2011 IEEE 32nd Real-Time Systems Symposium*. IEEE. 2011, pp. 34–43.

[2] A. Burns and A. Wellings. *Analysable Real-time Systems: Programmed in Ada*. Createspace Independent Publishing Platform, 2016. ISBN: 9781530265503. URL: `https://books.google.co.uk/books?id=ieZ9vgAACAAJ`.

[3] Alan Burns and Robert Ian Davis. "Response Time Analysis for Mixed Criticality Systems with Arbitrary Deadlines". In: *5th International Workshop on Mixed Criticality Systems (WMC 2017)*. York. 2017.

[4] Alan Burns, James Harbin, and Leandro Soares Indrusiak. "A wormhole noc protocol for mixed criticality systems". In: *2014 IEEE Real-Time Systems Symposium*. IEEE. 2014, pp. 184–195.

[5] Leandro Soares Indrusiak and Piotr Dziurzanski. "Evolutionary Optimisation of Real-Time Systems and Networks". In: *arXiv preprint arXiv:1905.01888* (2019).

[6] Fan Liu, Ajit Narayanan, and Quan Bai. *Real-Time Systems*. 2000.

[7] U ManChon et al. "GART: A genetic algorithm based real-time system scheduler". In: *2011 IEEE Congress of Evolutionary Computation (CEC)*. IEEE. 2011, pp. 886–893.

[1] [4] [5] [7] [3] [6] [2]