TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

CARLOS FERNANDEZ ALGUACIL

NETWORK ON CHIP SIMULATIONS

Master of Science thesis

Examiner: Professor Timo D. Hämäläinen
Examiner and topic approved by the Faculty Council of the Faculty of Pervasive Computing on October 2016

ABSTRACT

CARLOS FERNANDEZ ALGUACIL: Network on Chip simulations
Tampere University of technology
Master of Science Thesis, 44 pages
October 2016
Master's Degree Programme in Information Technology
Major: Software Engineering
Examiner: Professor Timo D. Hämäläinen
Keywords: Network on Chip (NoC), Transaction Generator (TG), traffic models, benchmark.

From the beginning of time, people are concerned to compare different things. On this thesis, it will be analyzed the performance of a NoC architecture. To complete this purpose, it is used a program called Transaction Generator (TG). TG is a program developed in TUT, and it is used to benchmark various parameters of NoC architecture. Different traffic models will be executed using TG, to know how the different characteristics (number of tasks, existence of periodical events or number of PEs) of these models will influence on the performance of the NoC architecture.

To get a better knowledge of the behaviour of the NoC, traffic models are executed several times using TG while some of the attributes of the NoC are modified. With these changes (different traffic models and different attributes), the thesis presents a wider view of the NoC's behaviour.

Results and conclusions are presented on the final chapter of the thesis. The difference on the performance of the NoC is more reduced than expected when different traffic models are executed. This is because the traffic models selected do not present a huge difference between them. Besides, the impact of the attributes modified on the NoC is easily appreciated, and basically do not depend on the traffic model selected. However, this not implies that all the attributes measured have the same impact. For example, the activation of the DMA presents a high difference on the performance, while different NoC frequencies do not have any impact.

TG allows the user to monitor the execution of the traffic model in real time, but that stays out of the scope of this thesis. This tool of TG is called Execution Monitor and it could be a good choice to use it in futures thesis. With Execution Monitor the user could do a deeper research on the performance of a NoC architecture.

PREFACE

I wish to thanks my family for all the support they gave me, specially my parents and my sister.

I also want to thanks Cristina, because only she knows how hard this has been for me.

CONTENTS

TERMS AND DEFINITIONS

CSV	Comma Separated Value
DSE	Design Space Exploration
FIFO	First In, First Out
IP	Intellectual Property
MCSL	Multi Constraint System Level (also Mobile Computing System Lab)
NoC	Network-on-Chip
PE	Processing Element
RTL	Register Transfer Level
TG	Transaction Generator
TLM	Transaction Level Model
TUT	Tampere University of Technology
VLSI	Very Large Scale Integration
VOPD	Video Object Plane Decoder
XML	Extensible Markup Language

1. INTRODUCTION

The purpose of this project is to analyse the performance of a Network on Chip (NoC) system in different situations. There are multiple programs that can be used to achieve this purpose. In this case, Transaction Generator (TG) is the program that has been selected. There have been selected a group of traffic models with different characteristics, such as number of tasks, number of PEs, or existence of periodical events. These traffic models will be simulated with TG. Every traffic model will be simulated several times to compare the performance of the NoC, when the attributes change, such as frequency, packet size or type of modelling. After every simulation, a group of log files are backfilled with the simulations results, and these log files will be used to compare and analyse the different performance of the NoC. Figure 1.1 shows a summary of the process to get these analyses.

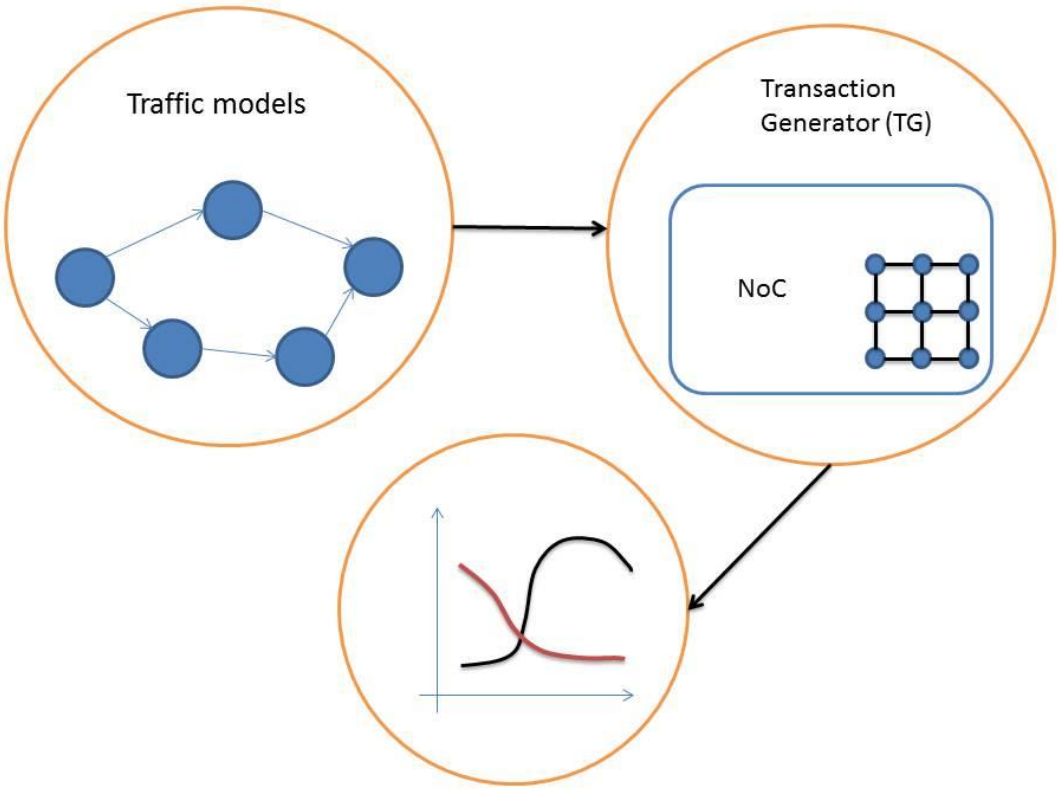


Figure 1.1: Summary of the project. First of all, a group of traffic models with different characteristics are required. Then, these traffic models are simulated using Transaction Generator (TG). Finally, the results obtained are compared to analyze the different behavior of the NoC.

2. RELATED WORK

This chapter presents a briefly introduction about Network-on-Chip (NoC) and Design Space Exploration (DSE). Furthermore, are also presented other simulators and different case studies.

2.1 Network on Chip (NoC)

Network on Chip (NoC) is a communication system built on an integrated circuit. NoC were created to improve the structure, modularity and performance compared to the conventional bus or point-to-point networks. NoC consists on a set of modules where the data is transferred among themselves. NoCs usually utilize a multi-hop topology instead of direct wires [7]. On multi-hop networks the wires can be shared by many modules. The advantages of this paradigm include: local performance is not degraded when scaling; network wires can be pipelined because links are point-to-point. [8]

The modules are the functional units of the NoC architectures, and they are Intellectual Property (IP) blocks. These blocks are connected to the network by a network interface. NoC architectures also have routers to select the routing convention and links to connect these routers in order to define a network topology [9]. Nowadays, there are multiples network topologies defined to implement a NoC. For example, those in Figure 2.1. Some of the design aspects that vary between these topologies are: physical area, power consumption or latency. It is not easy to choose the best topology in each case to get the best trade-off between the design aspects.

NoC architectures can be configurable. The user can change and modify some parameters, such as the scheduling algorithm, the packet size sent or buffer depth. This can be done in order to have the characteristics the user find more suitable for his purposes. Besides, the applications tasks can be mapped and scheduled in many different ways. To find the most efficient combination for each application many tests are required [10].

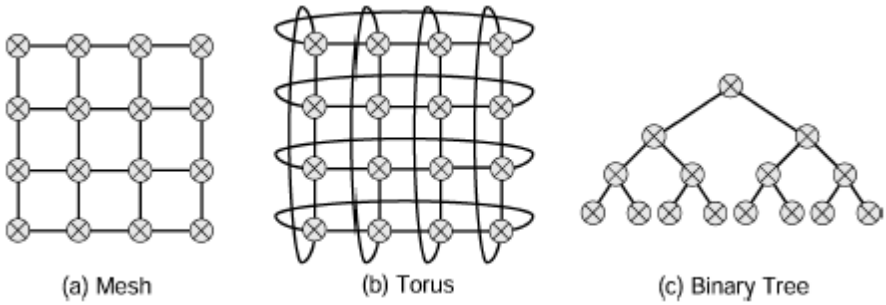


Figure 2.1: Three different NoC topology examples. The image shows the routers and the links that connect them. The functional elements linked to the routers are not shown [9].

2.2 Design Space Exploration (DSE)

It is difficult to find an efficient combination of parameters. In order to find it, many simulations have to be made to find a valid solution. This process is called Design Space Exploration (DSE) [11]. A usual scheme for this process is commonly called "Y-chart", as shown in figure 2.2. The application is mapped to the architecture, and then, this configuration is simulated, gathering all the necessary data to measure if it fits he desired requirements. There is a lot of data that could be measured, such as power consumption, usage of the processors, or time to complete the application.

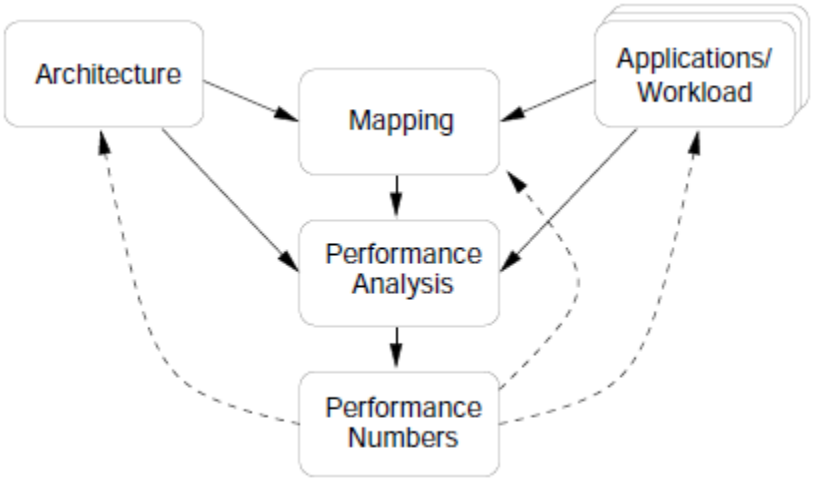


Figure 2**¡Error! No hay texto con el estilo especificado en el documento.**.2: Y-chart Design Space Exploration workflow [10]. Continuous arrows show the steps followed of the main workflow, while the dashed arrows show the feedback paths.

DSE has 3 important aspects.

1) Accuracy. The first simulations do not necessarily need a very precise accuracy, because the point is to discard the configurations that clearly are not suitable and to estimate the variation between solutions. Later, the accuracy can be increased to finally select the configuration to choose.
2) Exploration speed. The exploration speed could be increased by decreasing the accuracy, or moving from Register Transfer Level (RTL) modelling, which is slower but more accurate, to Transaction Level Model (TLM).
3) Amount of work the modelling requires.

2.3 Simulation utilities

Other simulators have been created with a similar purpose than Traffic Generator (TG). The purpose of these simulators is to benchmark different applications mapped in NoC architectures to find the most efficient or suitable configuration in each case.

3. TRANSACTION GENERATOR (TG)

Different programs have been designed to benchmark multiple NoC systems. One of these programs was created in the Technological University of Tampere in 2003. Its name is Transaction Generator (TG) [2]. TG is an open-source program and it allows the user to generate network traffic and benchmarking it to evaluate and explore the architecture of NoCs. 10 years later the program was redeveloped by Lasse Lehtonen for his Master thesis. The new version of the program develops some new features. It is now written in SystemC 2 TLM and it uses OCP-IP TLM sockets. It also implements an Accurate Dynamic Random Access Memory (DRAM) Model (ADM). [3]

To perform the different simulations and evaluate the different NoC architectures and their performances, the models that have been used are written in eXtensible Markup Language (XML). This language eases the understanding of the code, so it is easier for the user to know all the properties of the NoC architecture.

The version of Transaction Generator used during this thesis was implemented as a part of NOCBENCH project [4], funded by the Academy of Finland [5]. TG is now on its second version. C++ is implemented in TG and uses SystemC libraries to allow the notion of time. To enable the use of other different tools like Execution Monitor, it also uses some libraries from Boost [6]. These libraries also let the program to parse the XML input models that are used to evaluate the NoC architecture, which are the files that indicates all the conditions the simulation will have.

A basic structure of how TG works is shown in Figure 3.1. The first stage is Application, which defines the tasks, how they work, and the ports associated to each task. The second stage is Mapping, which defines where the tasks will be executed. The third stage is Platform, which defines PEs and its elements, such as cache or DMA. The fourth and last stage is Interconnection, which defines the characteristics and properties of the NoC architecture.

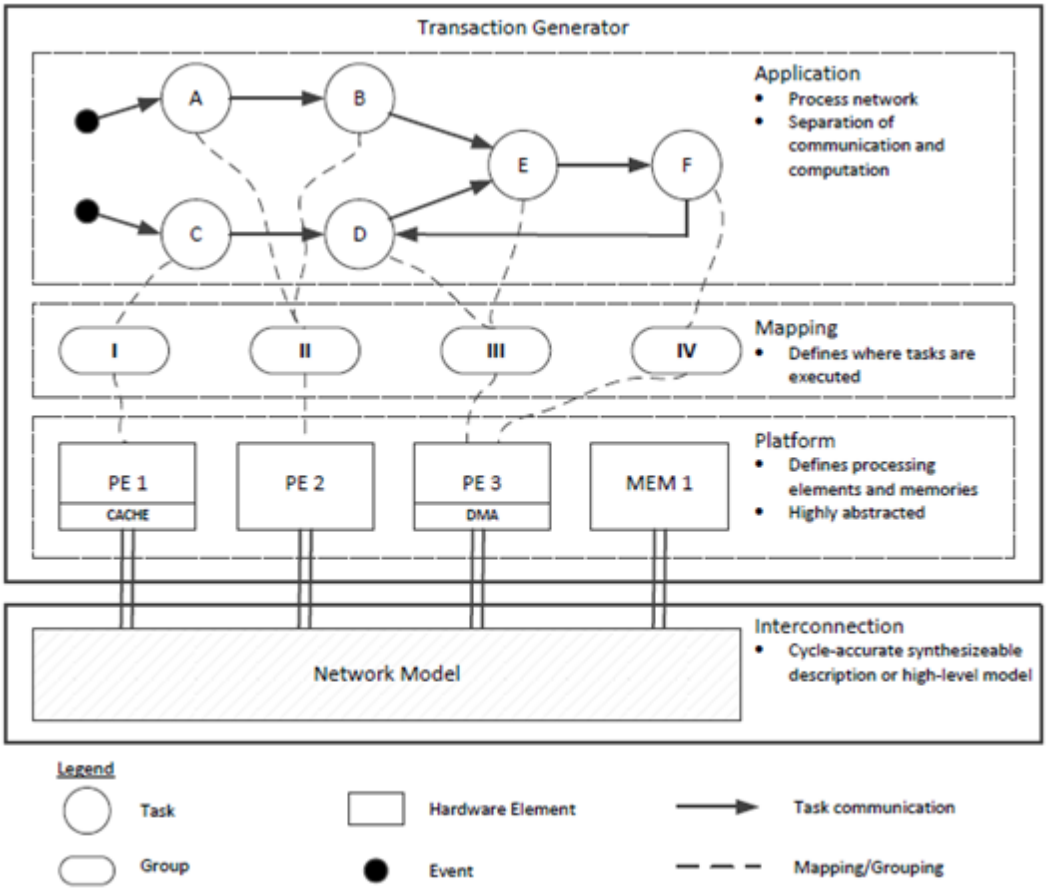


Figure 3.1: Conceptual view of TG. Simulation model is divided into four main parts [3]. This structure is saved into an XML file that is read by TG upon start.

To make easier the parsing of the input XML files the program is designed to parse their tags in smaller parts. This can be done because the implementation of TG is split in 15 classes. In this way, each class has only its own information and properties, and each tag or sub-tag of the XML files will be analysed on its own class. Figure 3.2 shows a diagram with the 15 classes of the implementation and the main relationships between the different classes.

Amount: During the simulation this class enables the evaluation of the distributions and polynomials that are defined in XML description. This allows knowing the amount of sent or received bytes.

Configuration: This class analyses the constraints of the input XML file and contains the general information of the simulation. It also has other data that is useful like the length of the simulation, the interval between the measurements, the NoC class or the mapping information.

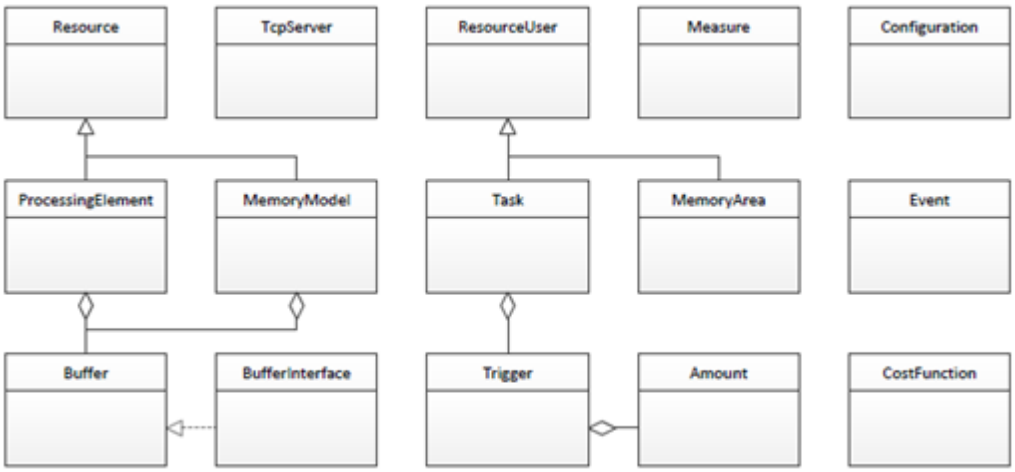


Figure 3.2: Diagram showing the classes of Transaction Generator implementation and the most important relationships between the classes [3].

CostFunction: As its name notices, this class analyses the different cost functions and execute a calculator to know them at the end of the simulation. The cost functions are the most important class to analyse the different performance that a same NoC class has when some parameters have been changed, like the frequencies of the CPUs. Some examples of the cost functions are the amount of times that a task has triggered or the percentage of utilization of a CPU.

Event: This class implements a SystemC thread to start the different events while the simulation is running. It also parses the description of the events.

Buffer: This class sets the model of the PEs internal memory. It also establishes the communication interface between the resources and the network model.

BufferInterface: This class establishes the interface that the network model sees from the Buffer class.

Measure: This class creates the SystemC threads to have all the measurements joined while the simulation is running and communicating with the Execution Monitor.

Resource: This is a base class for the Memory Model and the PEs. It analyses the information they have in common, as the frequency, the packet size or the buffer sizes.

MemArea: This class parses the data relevant to the memory areas.

MemoryModel: This class analyses the information related to memory elements and manages the communication with the ADM DRAM models.

ProcessingElement: This is the most important class and parses the data relevant to PEs and sets some properties, such as the communication and execution of the tasks, the scheduling algorithms (like FIFO), or the cache miss models.

ResourceUser: This is the base class for MemArea and Task classes. It gathers the information they have in common, like input and output ports or the identification numbers.

Task: This class analyses the most general task information from the task tags. It also sets the task model's internal state machine.

Trigger: This class analyses the trigger tags and gathers a set of operations to run after being triggered.

TcpServer: This class builds a TCP server to communicate with the Execution Monitor

3.1 XML Models

To perform the simulations, TG uses models written in eXtensible Markup Language (XML) [1]. This language is made up of storage units that are called entities. Each entity contains data, and this data can be parsed or not. If the data is parsed, it is made up of characters. Some of them are character data, and others are markup. Markup encodes a description of the document's logical structure and storage layout. The most remarkable goals that XML development set are: easily usable over the Internet, support a wide range of applications, easy to write a program to process XML documents, formal and concise design on the documents, which should be human-legible and clear.

The XML documents used to perform the simulations on TG are clearly divided into 4 main parts: Application, Mapping, Platform and Constraints. This division has a main purpose. It helps the user to understand better how the model works. The four sections answer 4 important questions: What, Where, When and How.

3.1.1 Application

The first part of the XML models is the Application part. Here are defined the tasks and the events, how are the tasks connected between them and the triggers for each task. Figure 3.3 shows a task graph with the content of the application part of a XML model. In this case, the example presents: the events that triggers some tasks (i.e. adc); the different tasks, like load or norm; and the channels that communicate the tasks, indicating the amount of bytes sent between each pair of tasks and the input and output ports.

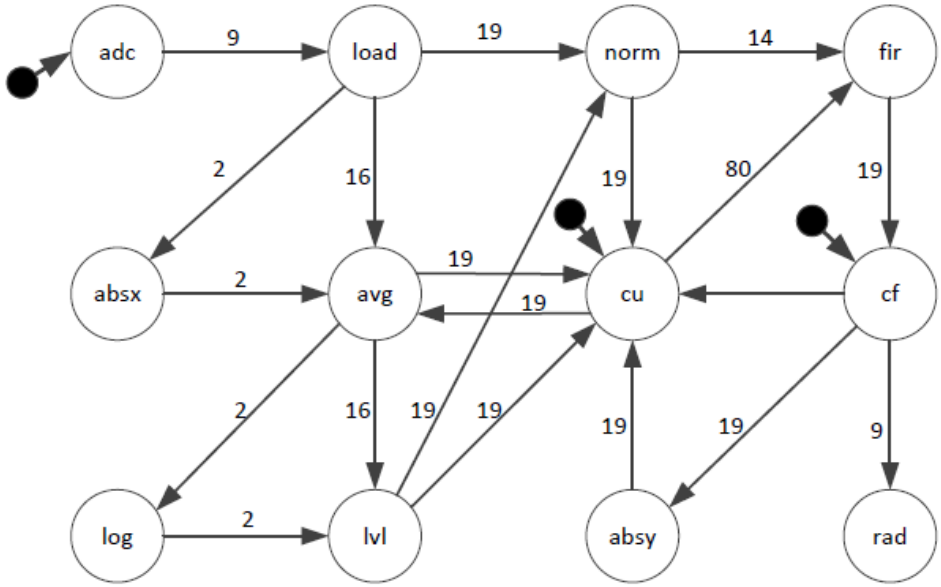


Figure 3.3: Visualized task graph presented in [5]. The events are represented by the small black dots; circles represent the tasks with their names; and the edges represent the communicational channels and the amount of data of the transmissions.

The application part of the models used with TG describes the tasks involved on the execution of the model. This part of the model consists of 3 main groups:

Events: Here are defined the external inputs and the timers. They are used to trigger the execution of some tasks of the model. The difference between them is the timers are executed indefinitely and the external inputs are executed only a certain amount of times. On each of them it can be set the name, the id, the destination (out_port_id), the amount of data to send, the probability to be executed, the offset, the period, and in case of the normal inputs, the amount of times to be executed.

Tasks: After the events, it is the turn for the tasks to be defined. Figure 3.4 shows an example of how are defined the tasks on a XML file. The tasks contain lot of information. First of all, the communication channels are defined, so the input and output ports are set. Then, it has to be defined what is going to be the element that triggers the execution of the task. It could be some of the events previously defined, or some data send by another task. The triggers could be of different types, such as AND or OR. The tasks also carry an internal count (trigger's counter) about the amount of times the trigger has been triggered during the execution of the simulation. This allows the task to perform different operations depending on the amount of the trigger's counter.

```
<task name ="Task1" id="0" class =" general ">
  <in_port id=”10”/>
  <in_port id=”11”/>
  <out_port id=”15”/>
  <trigger dependence_type=”OR”>
    <in_port id=”10”/>
    <in_port id=”11”/>
    <exec_count min="0" max="100" mod_period="150">
       <op_count>
         <int_ops>
          <polynomial>
            <param value="13" exp="0"/>
          </polynomial>
         </int_ops>
       </op_count>
       <send out_id="7" prob="1">
         <byte_amount>
          <polynomial>
            <param value="384" exp="0"/>
          </polynomial>
         </byte_amount>
       </send>
       <next_state value="READY"/>
    </exec_count>
  </trigger>
</task>
```

Figure 3.4: Example of a task on a XML model used with the TG.

After that, it is time to define the execution counts. The execution counts can be executed a certain amount of times, or its execution could be also periodic and executed an unlimited amount of times, stopping only when the simulation finishes. The amount of clock cycles needed to execute a task, or the amount of bytes to send or read for each execution count could be calculated by three different methods. First of them is a polynomial equation (3.1) where we define the a_n terms, and depends on the amount of data received (*x)*.

$$a_n x^n \cdots a_2 x^2 + a_1 x + a_0 \tag{3.1}$$

The second option is a uniform distribution (3.2). In this case, it does not depend on the amount of data received, but it provides a uniform random amount, which is useful when only the range of the amount incoming data is known:

$$U(a, b) \tag{3.2}$$

The last option is a normal distribution (3.3). In this case, the mean of the distribution could be a parameter (μ) or the amount of data received (x), and the standard deviation is a constant (σ^2).

$$N(x;\mu, \sigma^2) \quad (3.3)$$

It exists also the possibility to combine the different options. Finally, each execution count depends also on a probability to be executed. If this probability is less than 1.0 (100%), the execution count will not be executed each time it is triggered. In this case, the probability is checked every time the trigger is fired.

Port connection: Here are defined the channels that connect every task and explains where the tasks send data. It could be possible two tasks have more than one channel between them in common. For example, both tasks could be a sender and a receiver to the other task; or one task could send data through two different channels to the same task because it wants to send different data.

Figure 3.5 shows an example of an application part on a XML model.

```
<application>
      <task_graph>
            <event_list>
                  ...
            </ event_list >
            <task name ="Task1" id="0" class =" general ">
                  ...
            </ task >
            < port_connection src ="1" dst ="10"/>
                  ...
                  ...
            < port_connection src ="95" dst ="100"/>
      </ task_graph >
</ application >
```

Figure 3.5: Application section of a XML model used with the TG.

3.1.2 Mapping

On this section of the XML files is defined define where the tasks are going to be executed. This section is a link between the application section, and the platform section. Tasks will be situated on software platforms, and multiple tasks could share the same platform. If they are grouped together the communication costs could be affected. Tags

could be used to characterize these software platforms. These tags could be used by some external DSE tools to get information about the resources (i.e. PEs and memories), the software platforms and the tasks. There are two kinds of tags: contents and position. The contents tag refers to the possible modification of the mapping by the DSE tool. If a tag is set as mutable, DSE tools are able to modify the mapping. Otherwise, if it is set as immutable, the mapping cannot be modified by DSE tools. Position tag appears on the software platforms, groups (formed by tasks) and tasks, but not on the resources. When it appears on the software platforms or the groups, it defines if the DSE tools are allowed to be remapped them as a whole. In case that is set on the tasks, it defines if the tasks can be remapped or not. If this tag is set as movable, the remapping could be modified by the DSE tools. Otherwise, if it is set as immovable DSE tools cannot modify the remapping. Figure 3.6 shows the mapping section of a XML model.

```
<mapping>
 <resource name="cpu1" id="0" contents="mutable">
  <sw_platform position="movable" id="0" contents="mutable">
   <group position="movable" name="g1" id="0"
          contents="mutable">
    <task position="movable" name="Task1" id="1"/>
    <task position="immovable" name="Task2" id="2"/>
   </group>
  </sw_platform>
 </resource>
 <resource name="cpu2" id="1" contents="immutable">
      ..
       ..
 </resource>
</mapping>
```

Figure 3.6: Extract of an example of the port connections of a XML model used with the TG.

3.1.3 Platform

This section defines the Processing Elements (PEs) where the tasks are going to be executed. These PEs could be different hardware resources, such as hardware accelerators, processors or memories. The descriptions of the different PEs are defined in a separate file, the PE library. So, on the platform section, it has to be described the characteristics of the PEs, but not their definition. The parameters include on the PE library are:

Type: classifies PE's in different groups: general processors, memories and hardware accelerators.

Frequency bounds: defines a range for the operating frequency of the PEs.

Direct Memory Access (DMA) controller: specifies if the PE can carry communication and computation operations at the same time.

Communication overhead: defines the timing modifiers used in the communication transactions.

Computation performance: describes the performance characteristics for the different type of data, such as integer, floating point and memory instructions.

Area: estimates the cost in mm^2 or kilogates of the platform when the DSE optimizations are utilized.

Power consumption: estimates the power consumption in DSE optimizations.

Last two parameters refer to the estimated consumption a PE will cause when it is a part of a NoC chip. However, they are estimations. For example, a PE with a high operating frequency could cost less total power than a PE with a lower operating frequency due to the time spent to execute the tasks. Type parameter defines the kind of applications a PE can execute when the mapping is made automatically. For example, when an application model is used, it will not allow executing some tasks in the PEs that cannot execute them in the real world. Finally, the rest of the parameters refer to the operating speed of the PEs.

It is time to describe the parameters of the PEs that are included in the main XML file. These are the parameters that have to be modified to know the performance of the NoC with PEs with different characteristics.

Frequency: establishes the operating frequency of the PE. The unit is MHz.

Type: sets the type of PE is going to be used. It must be defined this type in the PE library with its characteristics.

RX buffer size: defines the maximum amount of bytes of data that can be received and have not been consumed by the task models. A token of data is considered received when the task that has received the token has read it. It could be read it by the task itself or by the DMA unit. When this occurs, the equivalent size of the token is liberated of the RX buffer. If the buffer is full, it will stop reading data, so the model will stall.

TX buffer size: establishes the maximum size of tokens that are going to be sent by the PE. Like in the other buffer, if it ever becomes full of tokens, it will stop the sending of tokens. And this will cause the stall of the tasks and the congestion of the model.

Packet size: defines the maximum amount of bytes of a token, because the interconnection channels cannot send a token with unlimited size. If the token sent is bigger than the size allowed, it will be automatically separated in smaller tokens with the size permitted. When the tokens are received, they will be recombined into the original token.

Scheduler: defines the scheduling algorithm that is used by the PE to execute a task when the previous one has finished its execution. The possible algorithms are: First In First Out (FIFO), fixed priority and sequence scheduling schemes.

Figure 3.7 shows an example of the mapping section in a XML model.

```
<platform>
  <resource id="0" name="c1" frequency="50" type="CPU_TYPE_1"
            rx_buffer_size="262144" tx_buffer_size="1024"
            packet_size="16" scheduler="fifo">
  </resource>

  <resource id="1" name="c2" frequency="500" type="RISC_CPU"
           rx_buffer_size="131072" tx_buffer_size="2048"
           packet_size="16" scheduler="fifo">
     </resource>
</platform>
```

Figure 3.7: Extract of an example of the mapping section of a XML model used with the TG. There are two PEs defined with different characteristics.

The scheduling algorithm selected in each PE could affect the execution time of the tasks, because some tasks could be ready for their execution, while the PE is waiting until the task with preference is ready to its execution.

Figure 3.8 shows the application's state in TG. It is represented as a state machine, and all tasks starts in the WAIT state.

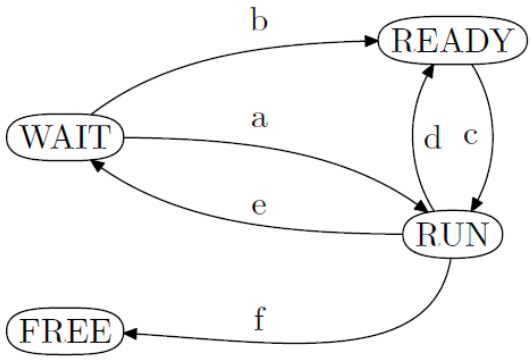


Figure 3.8: Application's state in TG represented as a state machine [3].

Different state transitions that exist in the application's state represented previously are defined in the next list.

a) Task was waiting for a reply from external memory, and before it has executed a READ statement. When the reply arrives, the task will be moved to RUN state.

b) Task's trigger is prepared to be fired because its conditions has been completed, and task moves to READY state waiting to be selected by the scheduling algorithm.

c) Task is selected for its execution by the scheduling algorithm, so it is moved to RUN state.

d) Task has been executed but it has more triggers to execute, but it is been moved to READY state by the scheduling algorithm.

e) Task has been executed, and at the moment it does not have more pending triggers, so it is moved to WAIT state.

f) Task has been executed and its last trigger has been fired also. Besides, task no depends any more on data during the simulation, so it is moved to FREE state.

Finally, another thing it can be described is the time a PE needs to execute the tasks it has assigned. When a task trigger's is fired and the task is moved to RUN state, TG calculates the operations that needs to be executed based on the amount of tokens received and the probabilities of the triggers defined in the application section of the XML model. After that TG calculates the data needed and some random values, and starts to execute the operations in order.

The time needed to execute the operations depends on the type and operating frequency of the PE, and the presence of a DMA controller ready to be use. One of the differences about the various types of PEs is the amount of cycles they need to execute an instruction in an operation. There are also 3 classes of operations: integer, floating-point and memory operations. Integer operations are mainly use in primary instructions that are normally executed in a fixed amount of time and do not depend on the PE they are going to be executed. That is not the case of the floating-point and the memory operations. These operations depend a lot on the PE they are going to be executed. For example, there are defined some types of PEs that have a dedicated floating-point unit. In this case, these types of PEs are going to need less time to execute floating-point operations than the PEs without that unit.

In order to calculate the cycles needed to execute a task, TG follows the next equation [3]

$$N_{cycles,i,pe} = \frac{N_{int,i}}{IPC_{int,pe}} + \frac{N_{float,i}}{IPC_{float,pe}} + \frac{N_{mem,i}}{IPC_{mem,pe}} \quad (3.4)$$

where scripts and subscripts on the equation mean: N_{cycles} is the amount of clock cycles; N_{int}, N_{float} and N_{mem} are the amount of integer, floating-point and memory operations; IPC is the PEs instructions per second; i is the id of the task we want to calculate the amount of cycles needed to its execution; and pe is the PE where the task will be executed. In order to know how much time is needed to execute the task, the amount of cycles needed using the PEs operating frequency f_{pe}, have to be converted into SI units. This is the equation that should be used:

$$t_{i,pe} = \frac{N_{cycles,i,pe}}{f_{pe}} \quad (3.5)$$

3.1.4 Constraints

This is the last of the four sections that a XML model is divided into. In this section it is going to be defined the parameters of the NoC architecture that are going to be used during the simulation. There are also set some parameters of the simulation. Figure 3.9 shows an example of this section in a XML file.

```
<constraints>
  <noc class="mesh_2d" type="sc_tlm_1" subtype="2x2"
       noc_freq_g="500"/>
  <rng_seed     value=”42”/>
  <sim_resolution    time="1.0" unit="ns"/>
  <sim_length  time="180" unit="ms"/>
  <measurements     time="5.0" unit="ms"/>
  <exec_mon    using="true"/>
  <log_exec_mon          file="log_execmon.txt"/>
  <pe_lib       file="examples/pe_lib.xml"/>
  <log_token      file="log_token.csv"/>
  <log_summary    file="log_summary.csv"/>
  <cost_function           func="pu_1"/>
  <cost_function      func="pu_avg"/>
  <cost_function           func="tc_1"/>
</constraints>
```

Figure 3.9: Extract of an example of the constraints section of a XML model used with the TG.

The parameters that are usually set about the NoC architecture in the XML models are class, type, subtype and its operating frequency. However, the definition of a NoC architecture, presents more attributes that can be also changed in the XML models. But, if they are not modified, all of them have a default value defined on the NoC description XML source. Some of these parameters are: depth of the FIFO, operating frequency of the IP or maximum size of the packet.

In this section, there are defined some parameters of the simulation. These parameters are:

rng_seed: sets a seed to calculate the random values. If this parameter is not set, the system will choose a random one.

sim_resolution: establishes the amount of time a clock cycle last on the simulation.

sim_length: defines the length of the simulation with the value and the unit.

measurements: set the amount of time between each measure the system does during the simulation. In other words, it sets the frequency of the measurements.

exec_mon: defines if it is going to be used Execution Monitor or not, and where the log file with the data is going to be saved.

cost_function: defines which cost functions are going to be calculated. It can be added or deleted as many cost functions as users wants.

Finally, it is also set the path where the PEs library is located and the paths where log files are going to be saved.

4. OWN EXPERIMENTS

In this chapter is going to be presented some of the simulations that have been done to acquire more knowledge about how the NoC works. First of all, there is a little description about the installation of TG and how it has been used. Then, the log files that will have been created are described. These files are automatically generated when a simulation is ran using TG, and provide valuable information about the model executed. Finally, some of the models that have been used are presented. In each model, it is included a brief description of it, and detail the analysis that have been made, listing the parameters that have changed in each file.

4.1 Setup

Transaction Generator is the program that has been used to benchmark the NoC architectures. It can be downloaded from the webpage of the NOCBENCH project [12]. Some examples models are included with this version. There are also some traffic models that are included in the zip package [13]. Some of these traffic models are used to perform the simulations described here. Besides, it has been used some Multi-Constraint System Level (MCSL) traffic patterns to have a better knowledge about the performance of the NoC with different kind of models [14]. These MCSL models can be downloaded from here [15].

4.2 Modified parameters

To analyse the different behaviour of the NoC, there has been ran several simulations of each model. Some parameters have been modified to compare how the NoC works on different conditions. There are many parameters that can be modified, so the possible combinations are too huge to approach all of them. Figure 4.1 shows a summary of all the modelling parameters that can be modified to benchmark the different NoC architectures.

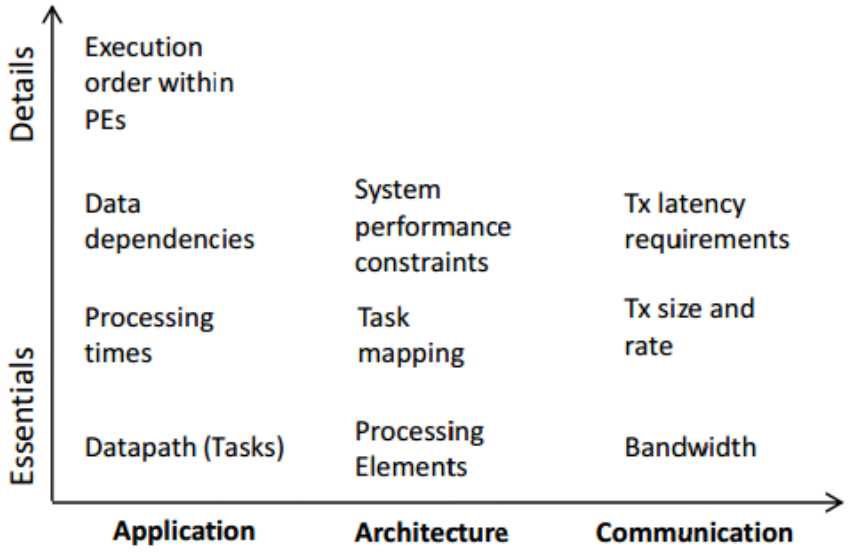


Figure 4.1: Modelling parameters addressed by TG [13].

The parameters that have been modified to run the different simulations are the following:

RTL vs TLM modelling: As it has been explained before, RTL is slower but more accurate. Comparing different simulations it can be estimated how much is the difference between the accuracy and the speed of both of them. If the accuracy of the TLM modelling is at least 98% of th RTL accuracy, and TLM is at least 15% faster, TLM will be chosen to perform our simulations.

Frequency: The frequency of the processors will be modified, and also, the frequency of the NoC. After that, it will be analysed how affect each of them to the performance of the NoC.

DMA: Some simulations will be performed with DMA activated, and some of them with DMA disabled.

PE: The number of PEs will be modified. To make these simulations easily repeatable, the tasks will be mapped by their index. For example, if there are 24 tasks and 2 PEs, task #1 to task #12 will be mapped to PE #1, and the rest of the tasks to the second PE. If there are 8 PEs, tasks will be mapped in groups of 3.

Mapping: It is going to be a remap of some tasks in some of the models to analyse how this affect to the performance of the NoC.

Packet size: Finally, the amount of bytes of the packets will be modified.

4.3 Log files

Five log files are generated automatically when a simulation is executed. They are saved on the path set in the model that has been executed. The format of these files is Comma Separated Value (CSV), which can be read with Excel or OpenOffice Calc. In each log file, user can find information about the simulation, such as processors utilization, tokens sent between tasks or the amount of clock cycles a task need to be completed. Now, each log file is described to know the information each of them have.

4.3.1 App

This log file shows the state and useful information about all the tasks from the beginning of the simulation till the end of it. The interval between the measurements can be changed in the XML model. Table 4.1 shows an example of the information this log file shows.

Time [us]	Id	Name	PE id	Cur. state	Tot #trig	Tot. #cycles	Tot. Tx tokens [B]	Sent local [B]	Sent remote [B]	Cur. RxBufUsage [B]
1000	0	FP1	7	READY	32	32	380288	128	380160	1
1000	1	ME	6	WAIT	0	0	0	0	0	11880
1000	2	DCT	5	WAIT	0	0	0	0	0	0

Table 4.1: Extract of log_app generated with the simulation of av_bench.xml

As it can be seen in the table, the program writes the information of the simulation when it is running and makes the measurements. This log file provides the following information:

Time: It shows the time when the data was measured.

Id: It gives the ID of the task.

Name: It shows the name of the task.

PE id: It provides the ID of the PE where the task is mapped.

Current state: It gives the current state of the task. It could be READY, WAIT or RUN.

Total triggers: It shows the amount of times the task has been triggered.

Total cycles: It provides the amount of cycles the task has been running.

Total Tx Tokens: It gives the amount of tokens the task has sent.

Sent local: It gives the amount of bytes the task has sent to another task mapped in the same PE.

Sent remote: It gives the amount of bytes the task has sent to another task mapped in a different PE.

Current Rx Buffer usage: It shows the amount of bytes there are currently in the Rx buffer.

4.3.3 Packet

This log file shows information about the tokens sent between tasks. It provides information about the amount of time a token of data needs to reach its destination. Besides, it tells the user between which two ports is the token sent. Table 4.2 shows an extract of this log file.

It can be seen on the table every token receive an ID. This allows the user to identify the path follow by a token. The log file also provides the time needed to send the token, and this time converted to amount of clock cycles. If the amount of bytes of the token is bigger than the packet size defined, the log file tells the user when the simulation starts sending that token, when ends, and the cycles needed to send each byte. Besides, it gives the ID of the source and destination ports, and the type of the token.

Rx Time [1 ns]	Token #	Bytes	Token begin	Token end	Time interval	Interv [cyc]	Src port	Dst port	Type	
4	2	4	b	e	2	1	2119	1921	wr	
355	4	8	b		350	175	1	10	wr	21 cyc/Byte
406	0	8	b		404	202	2423	2324	wr	25 cyc/Byte

Table 4.2: Extract of log_packet generated with the simulation of av_bench.xml

4.3.4 PE

This log file shows information about each PE every time the program does the measurements, like in log_app. When the program does the measurements the information about the PEs is filled. It contains very detailed information of every PE. First of all, the log file provides the time when the measurement has been done, the name and the ID of the PE. Then, it provides its current state, that can be send or idle, and its percentage of utilization. It also gives information about the amount of bytes that have been sent and received, and the amount of bytes that are currently in the transmission and receiver buffers. Finally, it provides information about the amount of clock cycles the PE has been doing different tasks, such as being idle, busy, executing, sending or communicating with itself. Besides, it provides the amount of cycles that the buffers have been waiting.

4.3.5 Summary

As it name expresses, this log file provides a summary of the simulation. In this case, the log file is not structure like a big table. First of all, it shows the name of the model used in the simulation, along with some information of it, such as length of the simulation and class, subclass and type of the NoC chosen. Then it provides the values of the cost functions included on the model that has been executed. Finally, it shows information about every PE. Table 4.4 shows an extract of this log file with this information.

Resource DSP4, id=1,freq=200

	cycles	%	Mcycles	
idle	29918400	99.7%	29.9	
busy	81600	0.272%	0.0816	
total	30000000	100.0%		30

Operation	Cycles	Bytes	Comment
Task exec	4800	-	
Intra PE TX	0	0	0 MB/s
Inter PE TX	76800	307200	2.05 MB/s
Inter PE RX	0	230400	2 MB/s
Instr miss	0	0	Tx + Rx = 0 * (8 + 4) bytes
Data miss	0	0	Tx + Rx = 0 * (8 + 4) bytes
Sum	81600 81 k	537600 537 kB	Intra + Tx + Rx = 0 + 307200 + 230400 bytes

Buffer usages	Max [Bytes]	Max [%]	Size [Bytes]
Tx	56	1.3e-06	4294967295
Rx:	48	1.12e-06	4294967295

Table 4.4: Extract of log_packet generated with the simulation of av_bench.xml. It shows the information related to the PEs.

First of all, it shows the name of the PE with its ID and its frequency expressed in MHz. Then, it provides information about the amount of cycles, and percentage of time, the PE had been busy or idle during the simulation. Furthermore, it presents the different operations done by the PE, such as execution of tasks, communication with itself, sending data to other PEs, or receiving data from other PEs. It provides the amount of cycles and bytes used to perform the different operations, along with the speed of transmission of the data. Besides, it appears if there has been any data or instruction miss, and in that case, the amount of bytes and cycles of them. Finally, it presents the size of the buffers, and the amount of bytes in the buffers on the moment when they were being more used.

4.3.6 Token

This log file provides the ID of the token, the time when the token was sent and when the complete token arrived to its destination. With this data, it shows the latency of this transmission, which is the difference between when the time was sent, and when it ar-

rived. This latency is calculated in nanoseconds and also in clock cycles. Then it provides the amount of bytes of the token and the number of packets that the token was divided to be sent. Furthermore, it gives the information about the task, PE and port that were source and destination of the token. Finally, it shows the type of the token, and if the token has more than one packet, it shows the speed of transmission.

5. RESULTS AND ANALYSIS

This is the main chapter of the thesis. Here are showed the results of the simulations that have been done with the models selected. Besides, it compares results of the simulations and analyse how the behaviour of the NoC change when some parameters of the models are modified. First of all, a brief summary of every model is presented, showing in some cases a figure that can help to understand how the model works. It is also explained the parameters we are going to analyse in each case. Then, some graphs are showed with the results obtained during the simulations. Finally, these results are explained to ease the understanding of the graphs.

5.1 Selected traffic profiles

There have been selected 6 models to perform the simulations. 4 of them are some traffic models gathered from different publications and included with the current version of TG. These models are: av_bench [16], VOPD [17], mpeg4_decoder [17] and radio_sys [18]. The other two models are some MCSL NoC traffic patterns. These two models are: H264-1080p_dec [14] and Robot [14].

5.1.1 Av_bench

This model is an audio-visual benchmark. It has 40 tasks, 56 edges and 16 PEs to execute them. Figure 5.1 shows the tasks that are mapped on the PEs and how they communicate between them.

On the figure, it can be seen the tasks included in this traffic model, and the PEs that execute them. It can also be observed which tasks send data to others and the amount of bytes sent. The tasks and PEs come with their names instead of their ID, so it is easier to understand the graph.

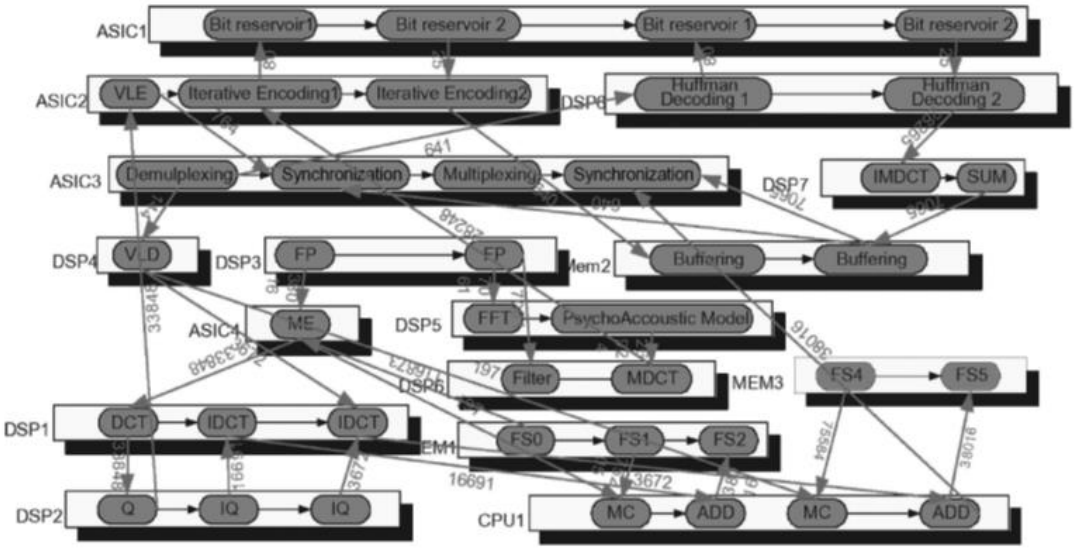
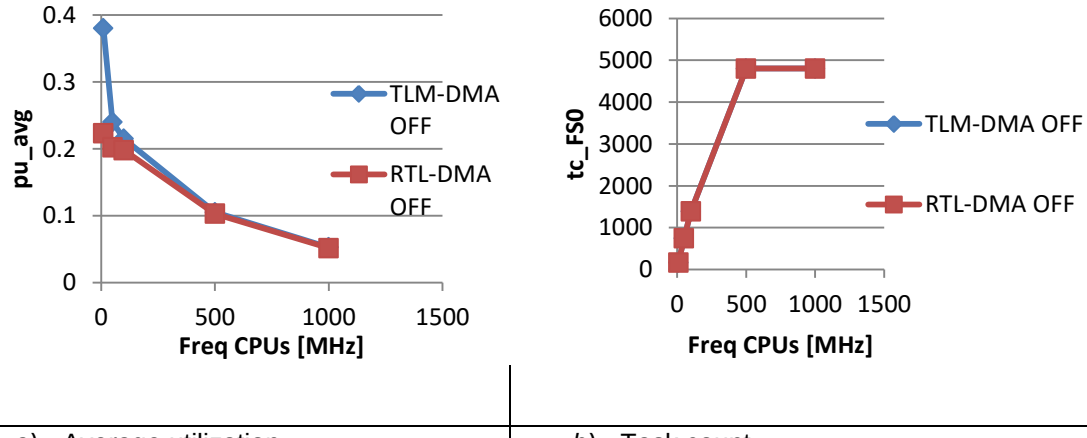


Figure 5.1: Dot graph of the av_bench traffic model. White rectangles represent the Pes, grey rounded rectangles inside the PEs represent the tasks and the arrows represent the communication channels between tasks and the amount of bytes transmitted by them.

In this case, it is analysed the effects of modifying the following parameters: RTL vs TLM, PEs frequencies, DMA and packet sizes.



a) Average utilization

b) Task count

Figure 5.2: The difference between RTL or TLM modelling measured with AV bench is small (<2%). The freq is scaled 1-1000 MHz.

Figure 5.2 a) shows the difference about RTL and TLM modelling in the PEs average utilization. At low frequencies, the PEs utilization is 41% less in RTL case. This difference is reduced when the frequencies of the PEs increases, but is always lower than

TLM modelling. Figure 5.2 b) presents the amount of times task FS0 is executed. In this case, there is no difference in this parameter when RTL or TLM are selected.

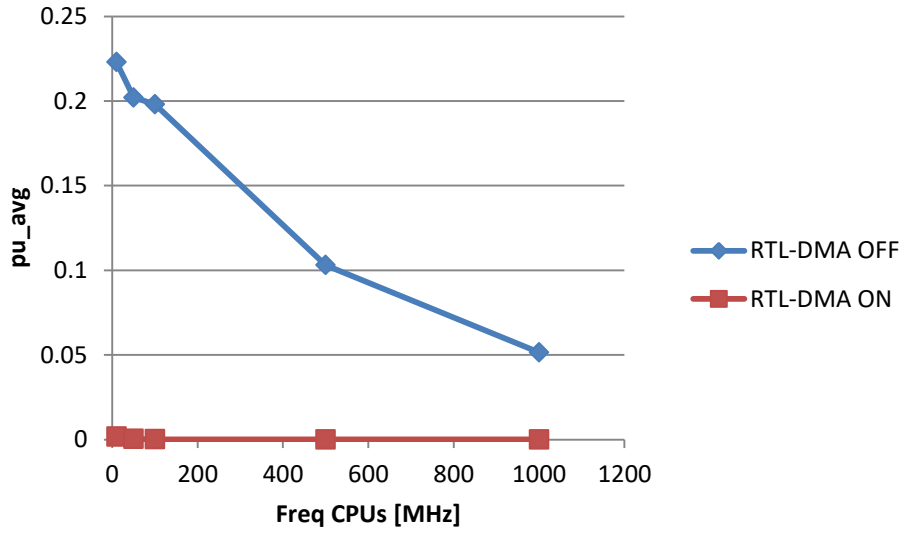


Figure 5.3: This graph shows the PEs average utilization when the DMA is OFF or ON along different CPUs frequencies.

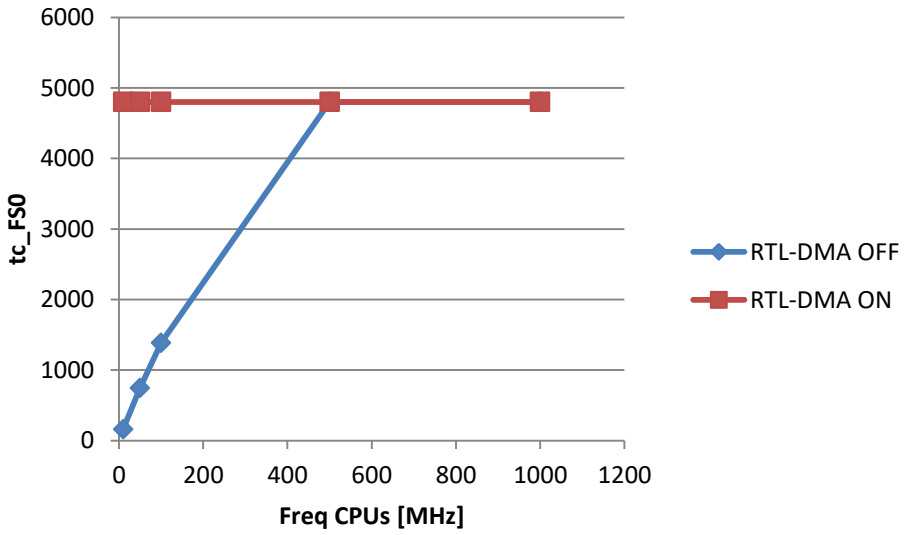


Figure 5.4: This graph shows the amount of times task FS0 is executed when DMA is ON or OFF along different CPUs frequencies.

Figures 5.3 and 5.4 present the different behaviour of the NoC if DMA is selected or not. Figure 5.3 shows the average PE utilization. In this case, when the DMA is OFF, the average PEs utilization is 13500% higher than the utilization when the DMA is on

and the CPUs frequency is 10 MHz. This difference is higher when the CPUs frequency is increased. At 1000 MHz, the average PE utilization without DMA is 313924% higher than the PE utilization with the DMA. Figure 5.4 shows the amount of times task FS0 is executed with and without the DMA. While without the DMA this task is only executed 161 times at 10 MHz, with the DMA it is executed 4800 times, which is the maximum number of times that task is executed. Without the DMA, the task is executed that amount of times only when the CPUs frequency is 500 MHz or higher.

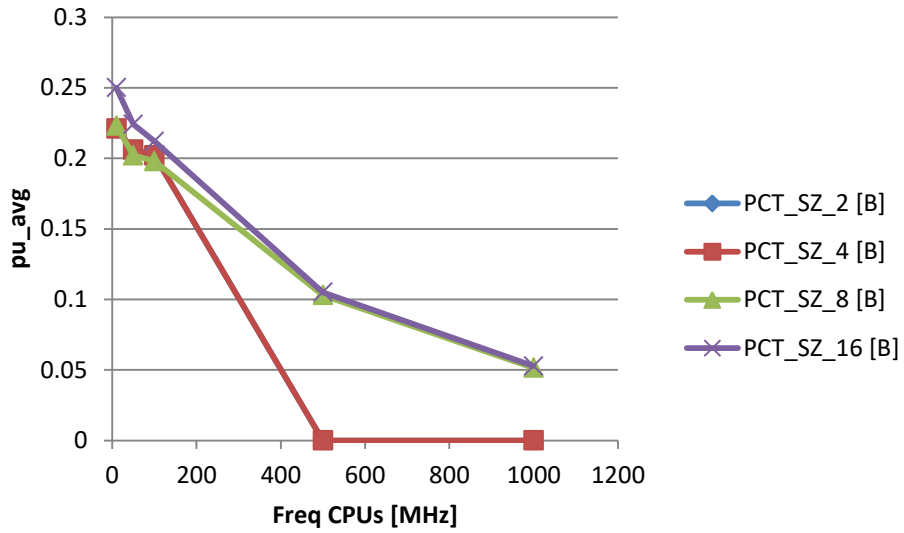


Figure 5.5: This graph shows the PEs average utilization along different CPUs frequency with different packet sizes.

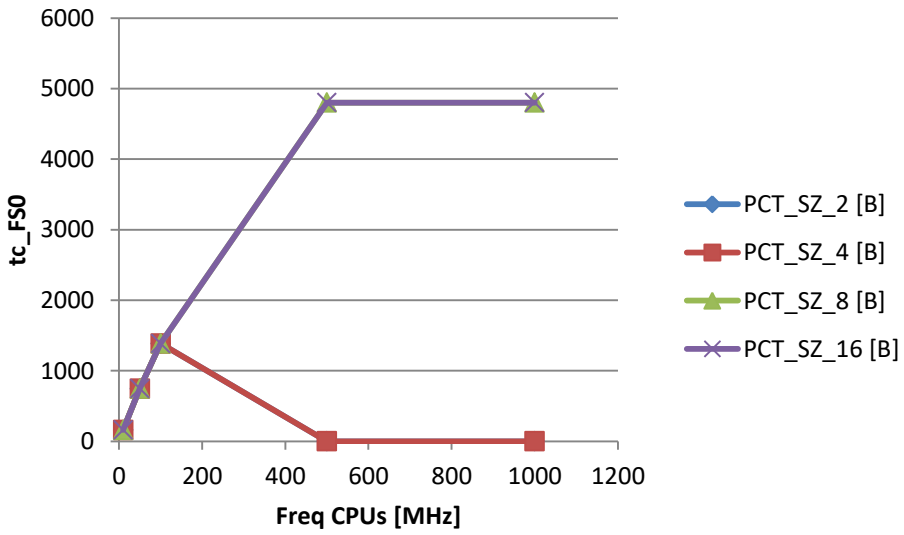


Figure 5.6: This graph shows the number of times task FS0 is executed along different CPUs frequency with different packet sizes.

Figures 5.5 and 5.6 show the behaviour of the NoC when the packet sizes are modified. Figure 5.5 presents the average PEs utilization. There is no difference when the packet size is 2 or 4 bytes, but when the packet size is 8 bytes the utilization is 1% higher at 10 MHz, and when the amount of bytes is 16, the utilization is 13% higher. When the packet size is 2 or 4 there is no information about the average PE utilization if the CPUs frequency is 500 MHz or higher, because the simulation is killed before the simulation is completed. However, figure 5.6 shows no difference about the amount of times task FS0 when the packet size is modified.

5.1.2 Test_mesh

This file is an example included with TG. It has 11 tasks and 4 PEs. It also has 2 periodical events, and it includes operation counts. In this case, it is going to be analysed the effects of modifying the following parameters: PEs frequencies, DMA and NoC frequencies.

Figures 5.7 and 5.8 show the different performance of the NoC when the DMA is activated or not using test_mesh file. Figure 5.7 presents the average PEs utilization. As it can be seen on the graph there is no difference when the frequency is low (10 and 50 MHz). Nevertheless, when the frequency is increased it can be observed a slightly difference. For example, when the frequency is 100 MHz, the PEs utilization is 0.1% lower when the DMA is activated, and when the frequency is 500 MHz this difference increases to 0.8%. Figure 5.8 shows the amount of tasks that have been executed when the frequency changes. In this case, as it can be seen on the graph, the influence of the DMA is observed at low frequencies (10, 50 and 100 MHz). When the frequency is higher the NoC complete all the tasks included on the file, and the presence of the DMA is only noticed on the PE utilization, as it can be seen in figure 5.7. When the frequency is 10 MHz, the amount of tasks executed with the DMA activated is 0.3% higher than the amount of tasks executed without the DMA. When the frequency is increased to 50 MHz, this difference increases to 0.4%. Finally, if the frequency is 100 MHz, the difference between the amount of tasks executed when the DMA is activated or not, is over 1%.

The impact of DMA in this file is really low. Nevertheless, in the previous file (Av_bench), DMA has a great influence in the performance of the NoC. This could be caused because test_mesh file is less complex than Av_bench.

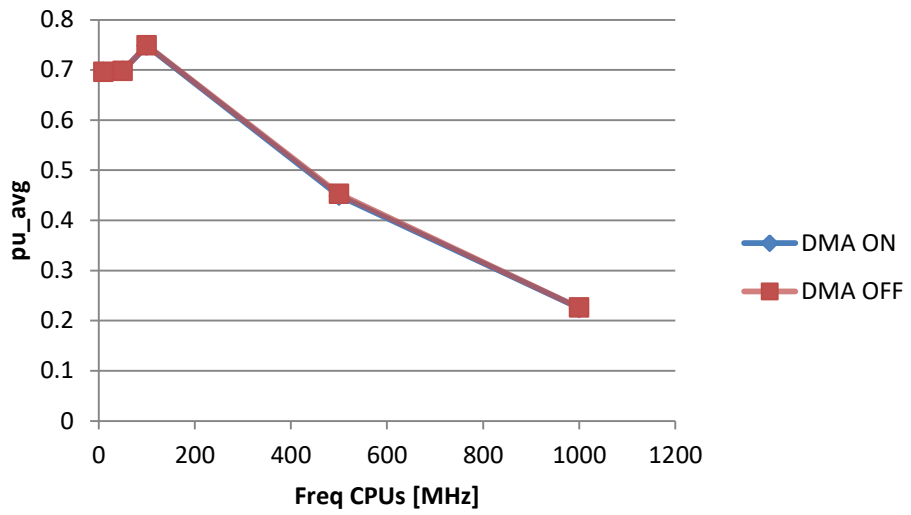


Figure 5.7: This graph shows the PEs average utilization when the DMA is OFF or ON along different CPUs frequencies.

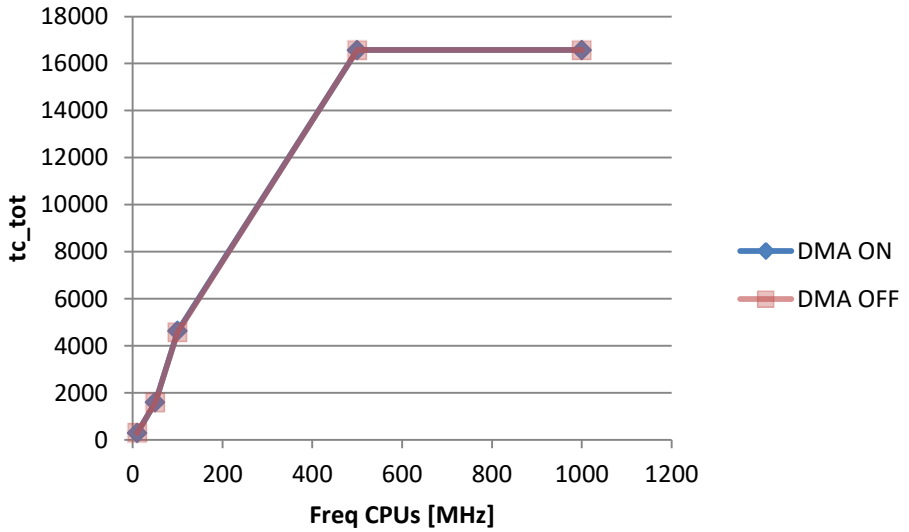


Figure 5.8: This graph shows the amount of tasks executed when the DMA is OFF or ON along different CPUs frequencies.

Figures 5.9 and 5.10 present the different behavior of the NoC when test_mesh is executed and the frequency of the NoC is changed with different frequencies of the PEs. Figure 5.9 shows the average PE utilization. As it can be seen on the graph there is no difference in this output except when the CPU frequency is 100 MHz. With other values of CPU frequency the average PE utilization is the same even though the NoC frequencies are different. When the CPU frequency is 100 MHz, the NoC frequency that gives a higher value is 10 MHz (0.759), and the lower value is given when the NoC frequency is 1000 MHz (0.749). Therefore, when the CPU frequency is 100 MHz, the average PE utilization is 1.34% higher when the NoC frequency is 10 MHz than when it is 1000 MHz. Figure 5.10 presents the amount of tasks executed along different CPU frequen-

cies when the NoC frequencies are changed. As it happens in figure 5.9, the amount of tasks executed only is different when the CPU frequency is 100 MHz. With other values of CPU frequency, the amount of tasks executed with different NoC frequencies is the same. When the CPU frequency is 100 MHz, the higher amount of tasks executed occurs when the NoC frequency is 10 MHz (4681 tasks), and the lower amount when the NoC frequency is 1000 MHz (4576 tasks). Therefore when the NoC frequency is 10 MHz, the amount of tasks executed is 2.29% higher than when the frequency is 1000 MHz.

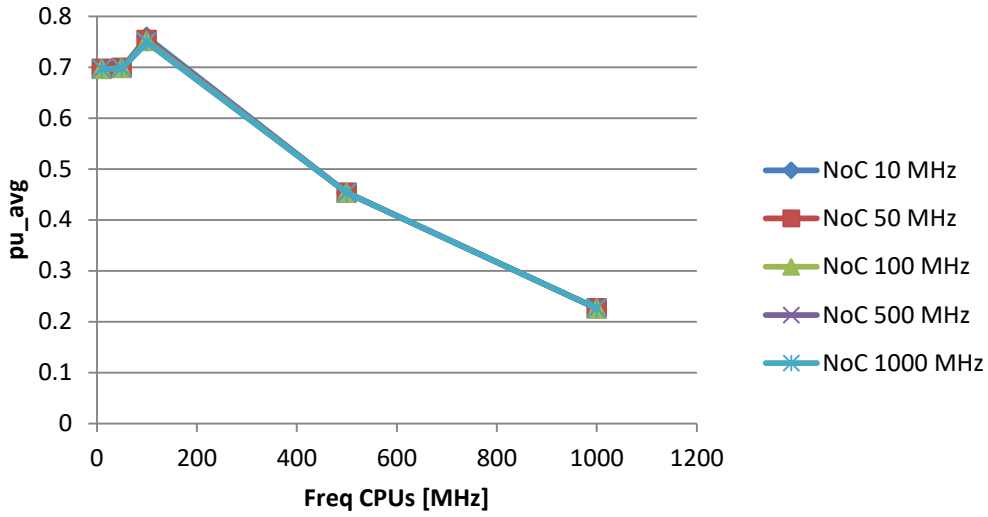


Figure 5.9: This graph shows the PEs average utilization when the NoC frequency has different values along different CPUs frequencies.

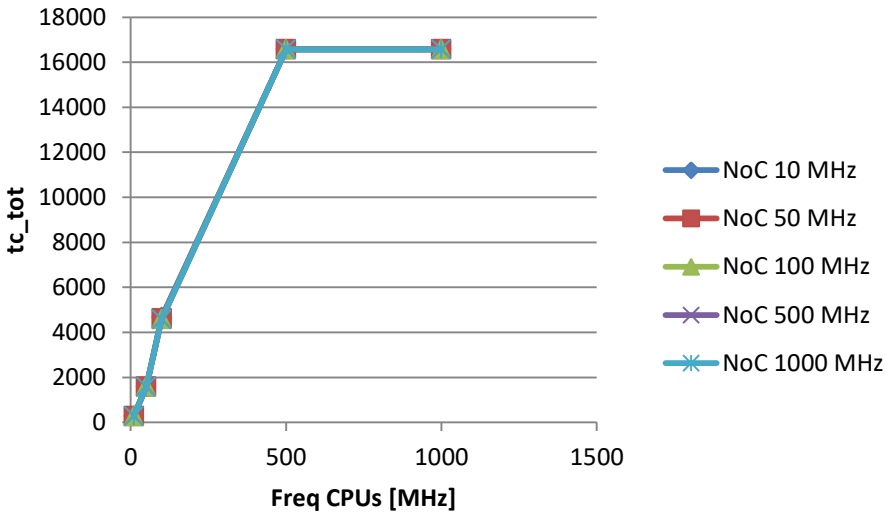


Figure 5.10: This graph shows the amount of tasks executed when the NoC frequency has different values along different CPUs frequencies.

5.1.3 Mpeg4_decoder

This traffic model is a video decoder for the MPEG-4 format. The model has 12 tasks, 16 PEs, 26 edges and 3 periodical events. Simulating this model it will be compared the effects of modifying the following parameters: frequencies, RTL-TLM and packet sizes. Figure 5.11 shows a pattern of the traffic model. White circles are the different tasks are going to be executed in this traffic model and their respective names. Arrows connecting the tasks represent the communication channels between the tasks and the required bandwidth in MB/s.

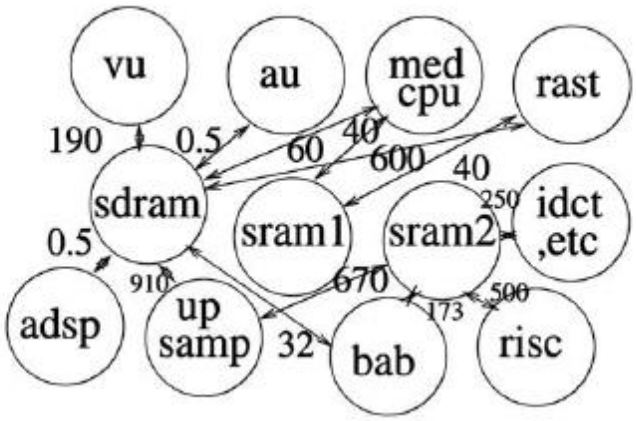


Figure 5.11: Dot graph of the Mpeg4_decoder traffic model. White circles represent the tasks to be executed and the arrows represent the communication channels between tasks and the required bandwidth in MB/s.

Figures 5.12 and 5.13 shows the different performance of the NoC when the traffic model is executed with RTL and TLM modelling. Figure 5.12 presents the average PE utilization with these types of modelling. As it can be seen on the graph, RTL modelling involves less PE utilization than TLM modelling. The difference between both modellings decreases when the frequency is increased. When the frequency is 10 MHz, TLM modelling requires 166.8% more PE utilization than RTL modelling. Nonetheless, when the frequency is 1000 MHz, TLM only requires 0.9% more PE utilization than RTL. Figure 5.13 shows the amount of tasks executed when “Mpeg4_decoder” is executed with RTL and TLM modelling. In this case, there is no difference in this output when RTL or TLM are used. Therefore, using RTL or TLM only involves a difference in terms of average PE utilization. As it has been described before, RTL modelling needs less time to execute the traffic models.

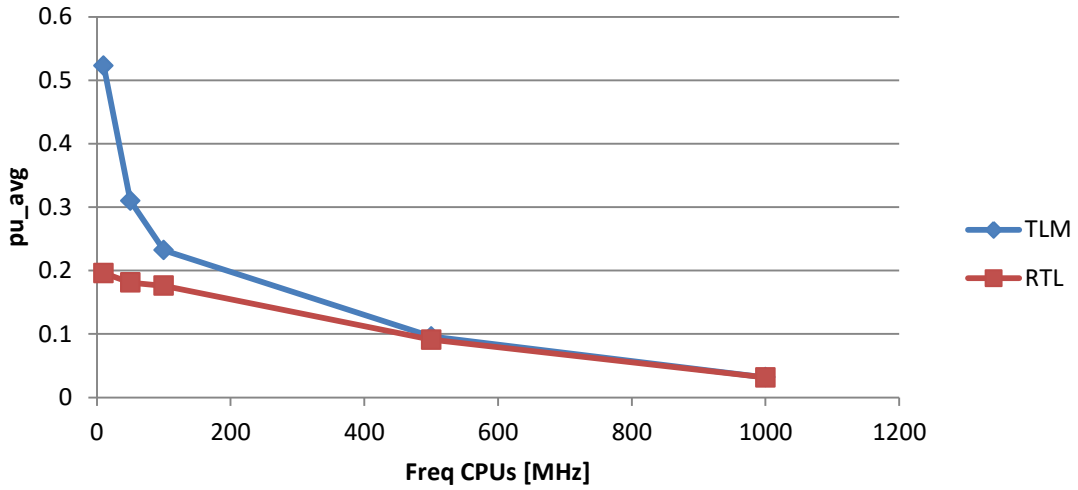


Figure 5.12: This graph shows the PEs average utilization when the "Mpeg4_decoder" is measured with TLM and RTL modelling.

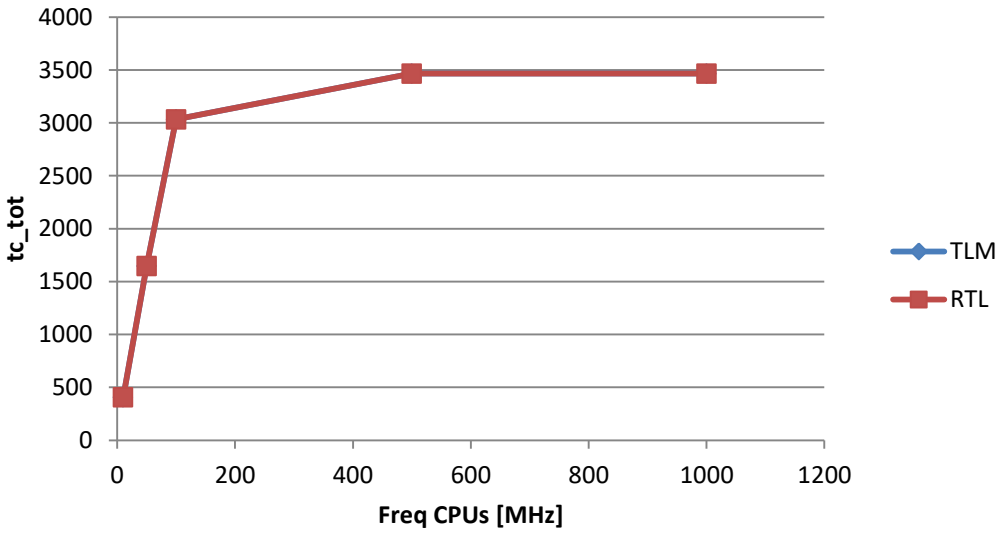


Figure 5.13: This graph shows the amount of tasks executed when the "Mpeg4_decoder" is measured with TLM and RTL modelling.

Figures 5.14 and 5.15 presents the different behavior when "Mpeg4_decoder" is executed with different packet sizes. It has been chosen packet sizes of 2, 4, 8 and 16 bytes. When the frequency is 500 or 1000 MHz, the execution of the traffic model stops and do not give data when the packet sizes are 2 or 4 bytes. Figure 5.14 shows the average PEs utilization. A packet size of 8 bytes involves less PE utilization than other packet sizes, and a packet of 16 bytes involves higher PE utilization. Higher differences are obtained at low frequencies (10 MHz), when the PE utilization with a packet size of 16 bytes is 10.7% higher. When the frequency is 1000 MHz, this difference reduces to 0.95%. In figure 5.15 can be observed the amount of tasks executed. In this case, the situation is pretty similar to when RTL and TLM modelling are compared. Changing

the packet size does not involve any difference in the amount of tasks executed, no matter the frequency chosen to execute the traffic model.

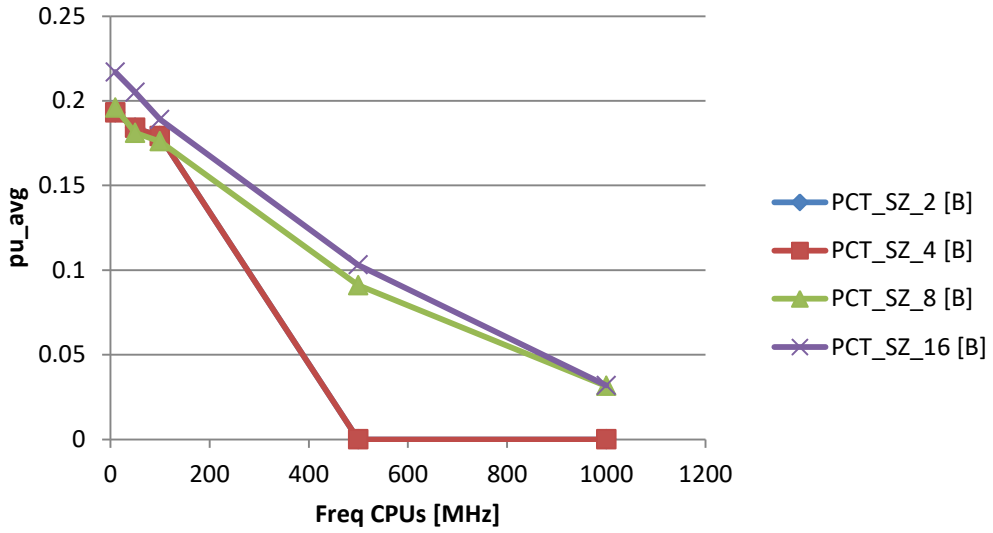


Figure 5.14: This graph shows the PEs average utilization when the "Mpeg4_decoder" is executed using different packet sizes.

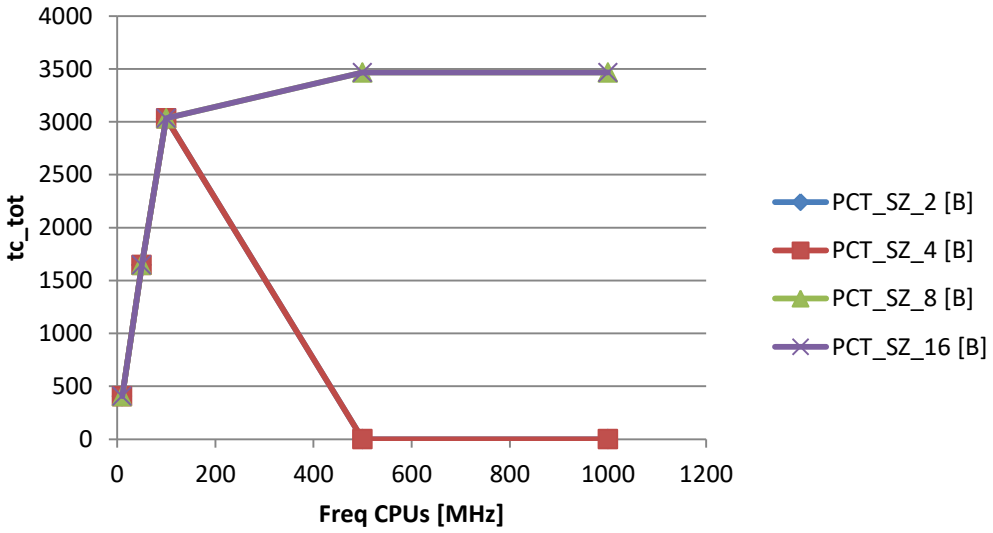


Figure 5.15: This graph shows the amount of tasks executed when the "Mpeg4_decoder" is executed using different packet sizes.

5.1.4 Radio_sys

This traffic model is an Ericson radio system. This model has 15 tasks, 16 PEs, 26 edges and 8 periodical events with different frequencies. In this case, it is going to be analysed the effects of modifying the following parameters: frequencies, DMA and packet sizes. Figure 5.16 shows a pattern of a radio system representing its node to node traffic-flow. Each square represents a PE, and the arrows the communication channels and the amount of data transmitted in each case.

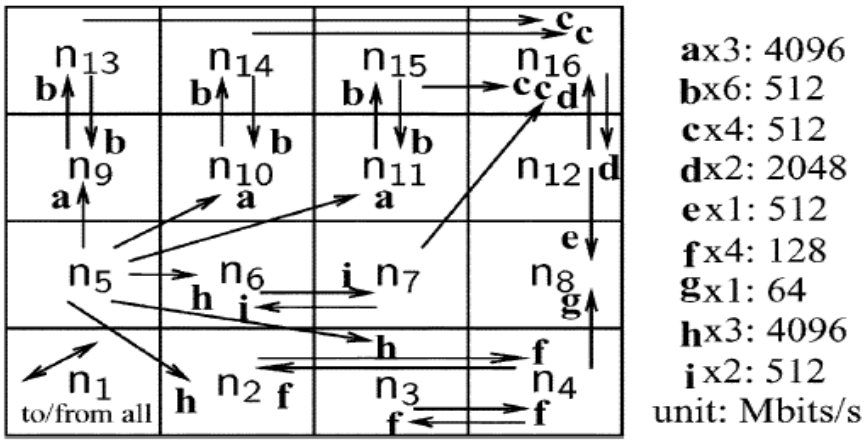


Figure 5.16: The graph represents a node-to-node traffic flow of the "Radio_sys" traffic model

Figures 5.17 and 5.18 show the different performance of the NoC when "Radio_sys" is executed with the DMA activated and not. On figure 5.17, it can be seen the average PE utilization. This output is very different when the DMA is active or not. When the frequency is 10 MHz, the average PE utilization without the DMA is 8286% higher than the case with the DMA. This difference increases when the frequency is higher. When it is 1000 MHz, the difference boosts up to 223000%. Therefore, the presence of the DMA has a really influence in this output. Figure 5.18 presents the influence of the DMA on the total amount of tasks executed. As it can be observed on the graph, when the DMA is activated, the total amount of tasks executed is the same for all the frequencies tested. Nevertheless, without the DMA, the total amount of tasks executed increases when the frequency increases too. With frequencies of 500 and 1000 MHz, this output has the same value than when the DMA is activated. Nonetheless, at 10 MHz, the amount of tasks executed without DMA is only 5% of the amount executed with DMA active.

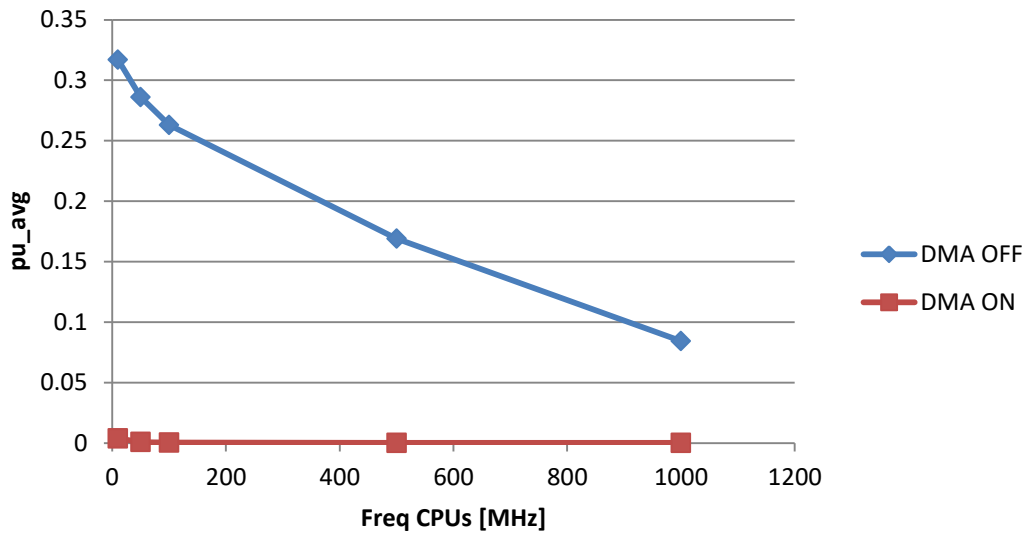


Figure 5.17: This graph shows the PEs average utilization when the DMA is OFF or ON along different CPUs frequencies.

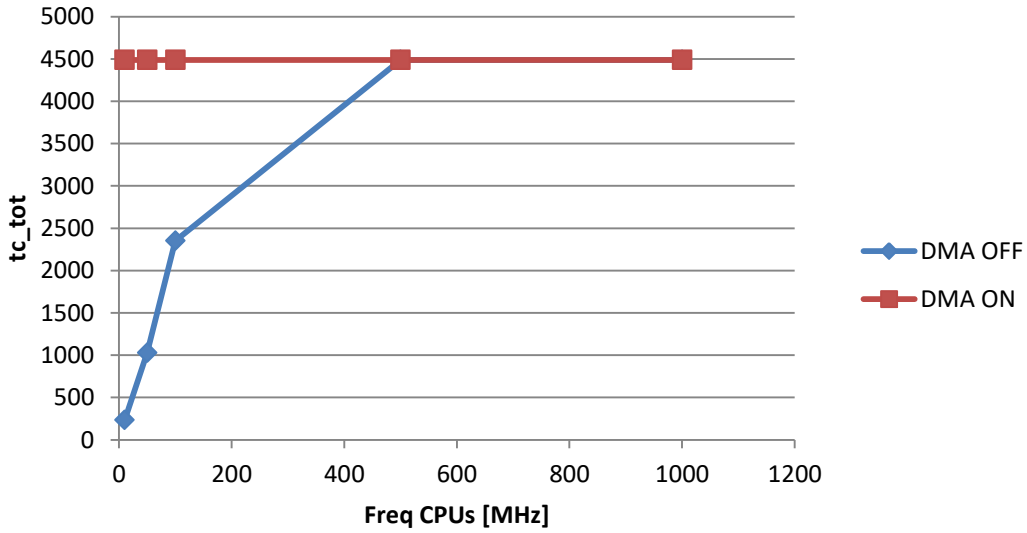


Figure 5.18: This graph shows the total amount of tasks executed in "Radio_sys" when the DMA is OFF or ON along different CPUs frequencies.

Figures 5.19 and 5.20 present the performance of the execution of "Radio_sys" using different packet sizes. In this case, TG does not give data when the packet size is 2 or 4 bytes and the frequency is 100 MHz or lower. On figure 5.19, it can be seen the average PE utilization with the different values of packet size. A packet size of 8 bytes involves less PE utilization than other packet sizes at every frequency tested. At 10 MHz, PE utilization when a packet size of 16 bytes is 5.68% higher than with an 8 bytes packet size. This difference decreases to 0.7% when the frequency chosen is 1000 MHz. On figure 5.20, the output analyzed is the amount of tasks executed. In this case, there is no difference in the output no matter the packet size is chosen. Every packet size chosen

execute the same amount of tasks. Therefore, the only difference related to the packet sizes is on the average PE utilization.

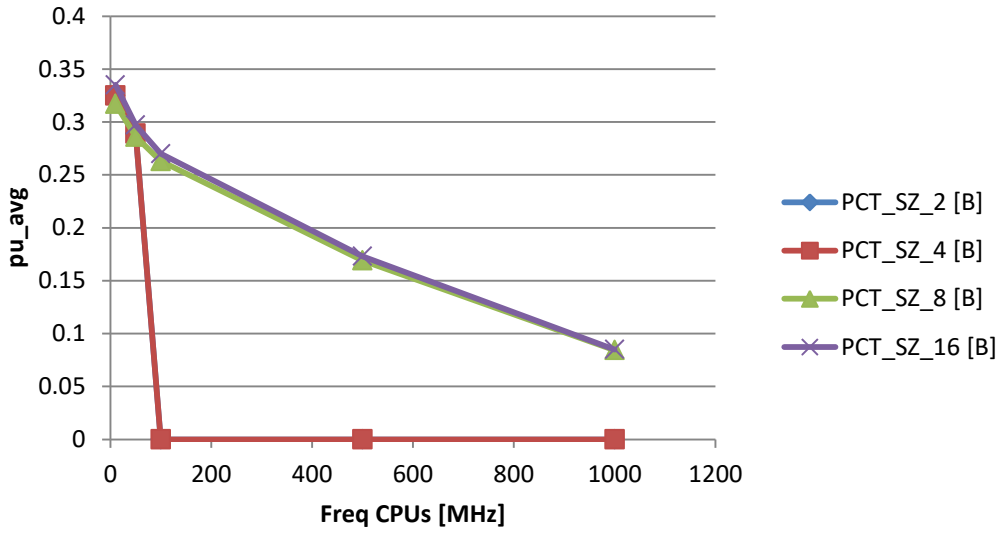


Figure 5.19: This graph shows the PEs average utilization when the "Radio_sys" is executed using different packet sizes.

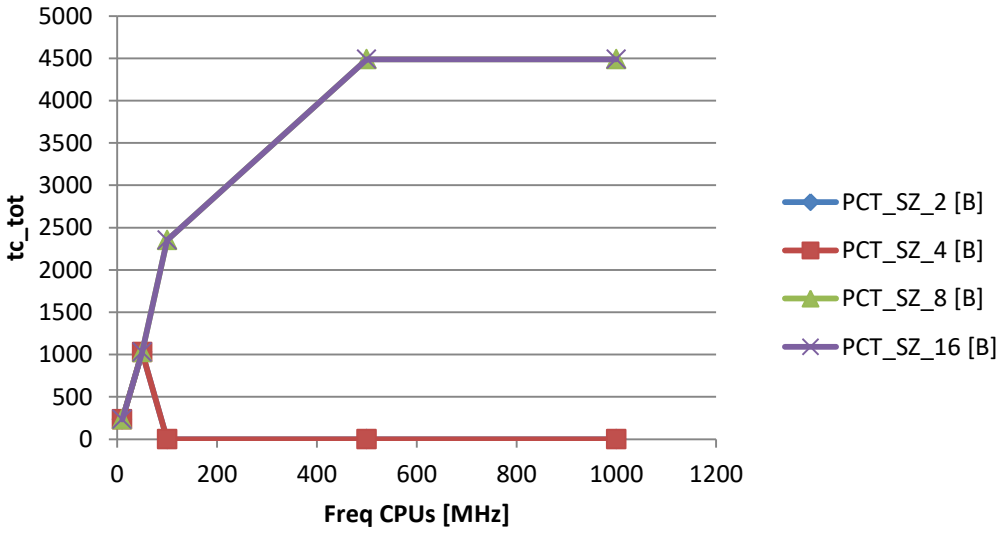


Figure 5.20: This graph shows the amount of tasks executed when the "Radio_sys" is executed using different packet sizes.

5.1.5 H264-1080p_dec

This model is a video decoder with a resolution of 1080p with the H.264 format, and it is one of the MCSL NoC traffic patterns it is going to be simulated. The MCSL files have much more tasks than the traffic models. In this case, this model has 5191 tasks

and 7781 edges. It will be analysed the effects of modifying the following parameters: RTL vs TLM, frequencies and DMA. Figure 5.21 shows a graph showing an extract of the task communication of this model. In this graph, the tasks are represented by white circles, and the arrows connecting the circles, represent the communication channels between the tasks.

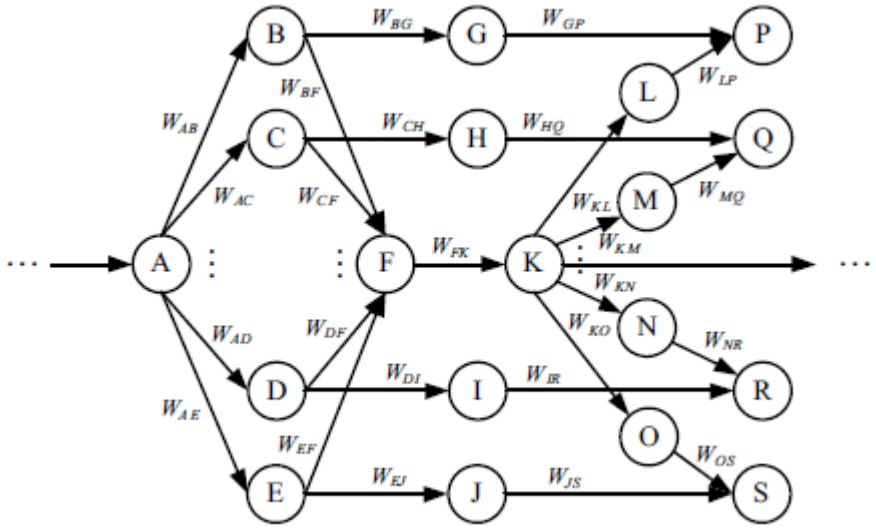


Figure 5.21: Extract of the H.264 decoder's task graph. Circles represent tasks, while arrows represent communication channels between tasks [14].

Figure 5.22 and 5.23 show the performance of the NoC while "H.264 decoder" is executed using two types of modelling: RTL and TLM. Figure 5.22 shows the influence of these types of modelling on the average PE utilization. At every frequency tested, the value of the output is lower when "H.264 decoder" was executed using RTL model. The highest difference between both modelling is produced at 1000 MHz. At this frequency, average PE utilization using TLM modelling is 76.4% higher than when RTL modelling is used. At 10 MHz, the difference between the modelling is 29.2%. Therefore, the relative difference involving this output increases with the frequency. Figure 5.23 presents the amount of tasks executed using both modelling. In this case, as it has been seen in previous traffic models, using different modelling does not change the amount of tasks executed at every frequency. In both modelling, the execution of "H.264 decoder" only gets the total amount of tasks (5191), at the higher frequency, 1000 MHz.

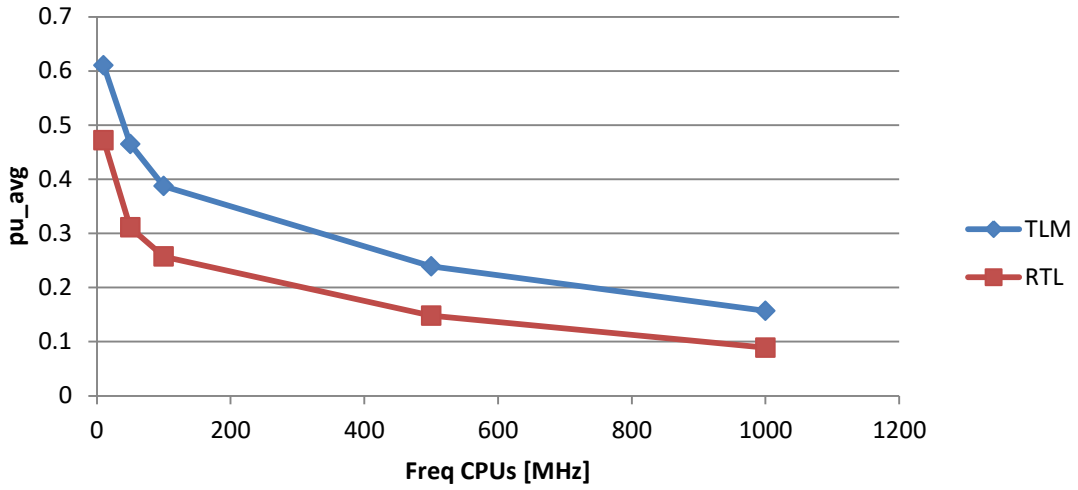


Figure 5.22: This graph shows the PEs average utilization when the "H.264 decoder" is measured with TLM and RTL modelling.

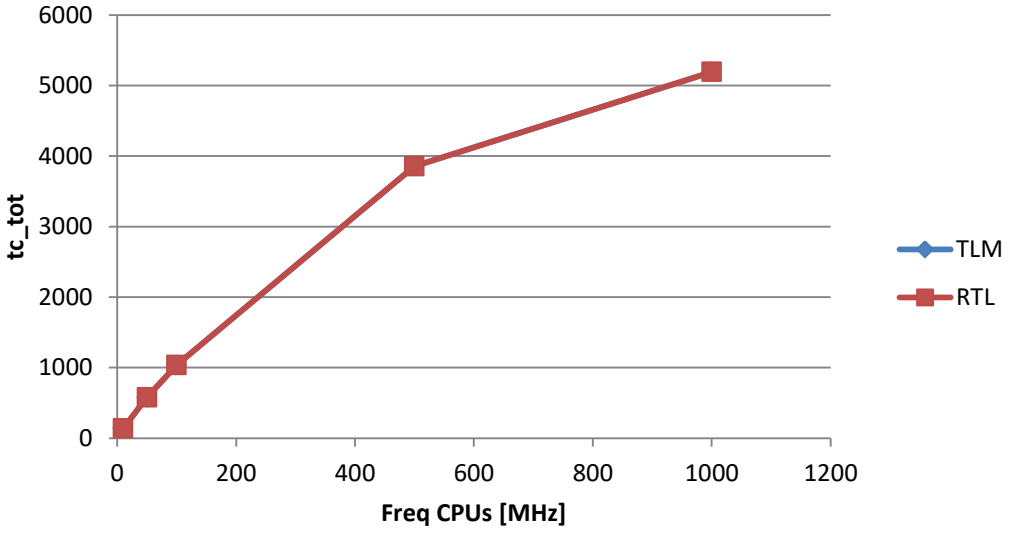


Figure 5.23: This graph shows the amount of tasks executed when the "H.264 decoder" is measured with TLM and RTL modelling.

Figures 5.24 and 5.25 show the influence of the DMA in the execution of "H.264 decoder". Figure 5.24 presents the results obtained with and without DMA when the average PE utilization was being evaluated. As it can be seen on the graph, the presence of the DMA has a great influence on this output. The relative difference increases when the frequency increases, but not as fast as it occurs in other traffic models. At 10 MHz, the average PE utilization without the DMA is 1264% higher than the PE utilization with the DMA. At 1000 MHz, the difference is 1999%. Figure 5.25 shows the impact of the DMA on the amount of tasks executed. When the DMA is active, the execution of the model completes all the tasks (5191), except when the frequency is 10 MHz, when

they are executed only 89.6% of the tasks (4653). Without the DMA, it is the opposite case. All the tasks are only completed at the highest frequency, 1000 MHz. At 10 MHz, they are only executed 137, and that means only 2.94% of the tasks executed at that frequency with the DMA.

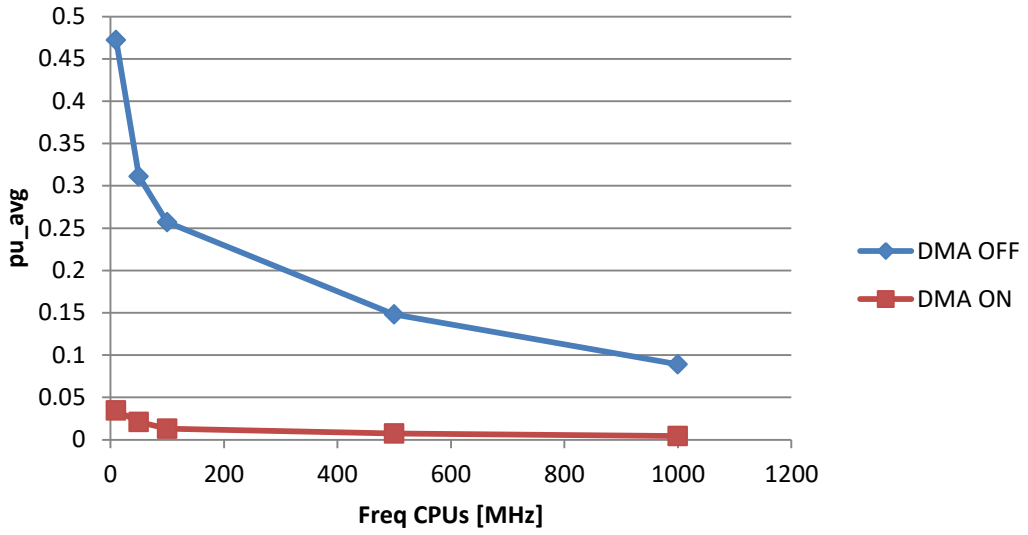


Figure 5.24: This graph shows the PEs average utilization when the DMA is OFF or ON along different CPUs frequencies.

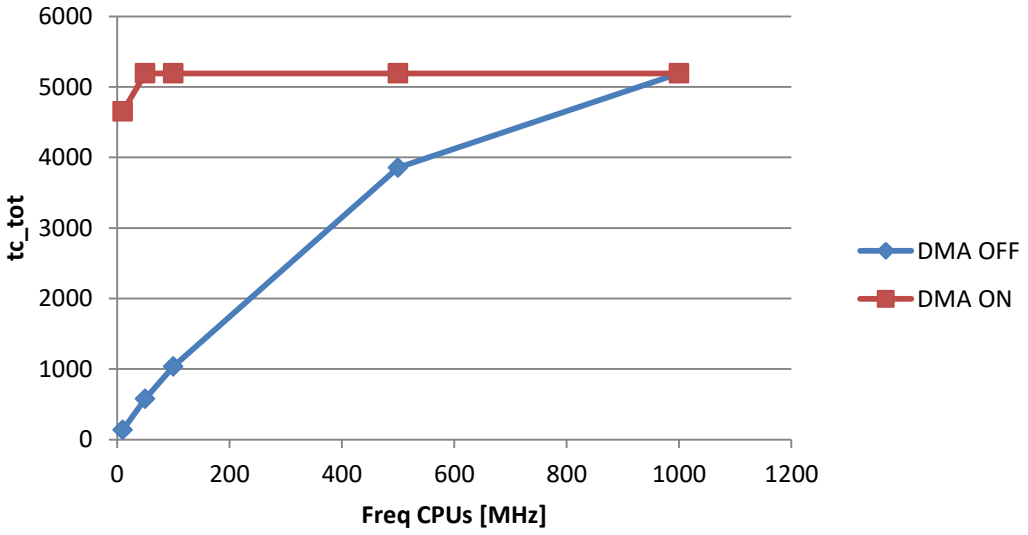


Figure 5.25: This graph shows the total amount of tasks executed in "H.264 decoder" when the DMA is OFF or ON along different CPUs frequencies.

5.1.6 Robot

This is the last file that will be analysed. This model is a MCSL NoC traffic pattern, which represents Newton-Euler dynamic control calculation for the 6-degrees-of-freedom Stanford manipulator. The model has 88 tasks and 131 edges. With this last model, it is going to be analysed the effects of modifying the following parameters: frequencies, DMA and NoC frequency.

Figures 5.26 and 5.27 present the influence of the DMA on the execution of the "Robot" model. Figure 5.26 shows this influence in the average PE utilization. The PE utilization is lower with the DMA activated at every frequency. The difference is higher when the frequency is also higher. At 10 MHz, the PE utilization without the DMA is 53% higher than the utilization with the DMA. Besides, at 1000 MHz, this difference increases up to 147%. On figure 5.27, it can be observed the influence of the DMA on the amount of tasks executed. When the DMA is not active, all the tasks (88) of the model are executed at every frequency tested, except at 10 MHz. At this frequency, only 76.1% of the tasks (67) are executed. DMA allow the program to execute the tasks faster. Therefore, it is normal that in this case, all the tasks are executed at every frequency without exceptions.

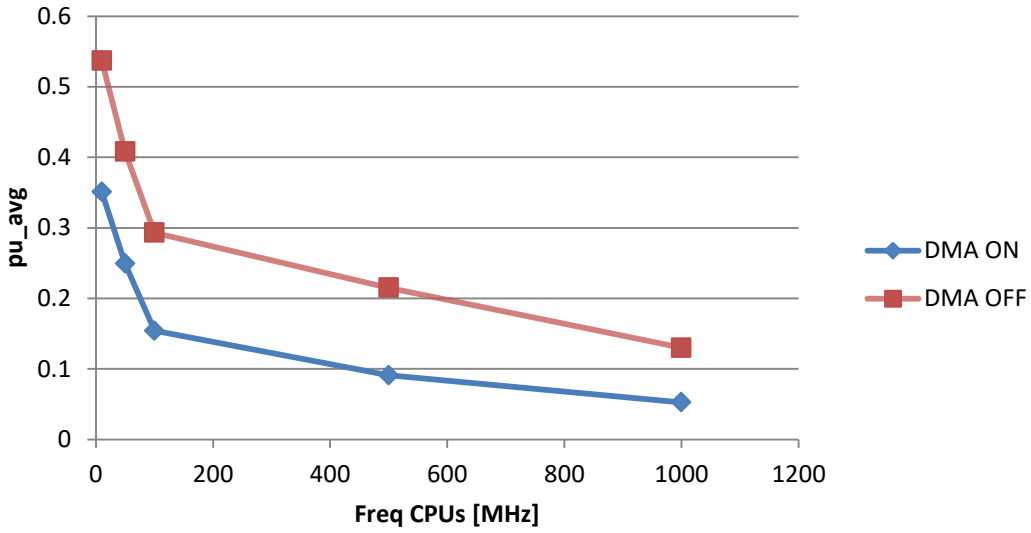


Figure 5.26: This graph shows the PEs average utilization when the DMA is OFF or ON along different CPUs frequencies.

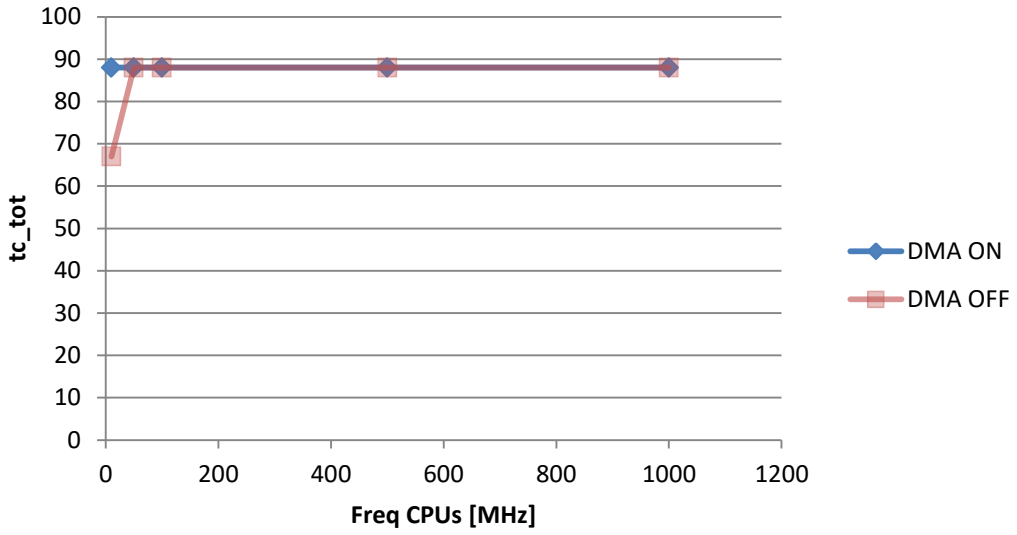


Figure 5.27: This graph shows the total amount of tasks executed in "Robot" when the DMA is OFF or ON along different CPUs frequencies.

Figure 5.28 and 5.29 present the different performance when the frequencies of the NoC are changed. On figure 5.28 it is analyzed the effect of the variation of this parameter on the average PE utilization. The PE utilization is the same for every NoC frequency at every CPU frequency, except when the CPU frequency is 50 MHz. In this case, the highest PE utilization is produced when the NoC frequency is 10 or 50 MHz, and the lowest when the NoC frequency is 1000 MHz. It is 97% of the PE utilization when NoC frequency is 10 or 50 MHz. Figure 5.29 shows the influence on the amount of tasks executed. In this case, there is no difference at all when different NoC frequencies are used. When CPU frequency is 10 MHz, 67 tasks are completed at every NoC frequency. At the rest of CPU frequencies, all the tasks (88) are completed at every NoC frequency.

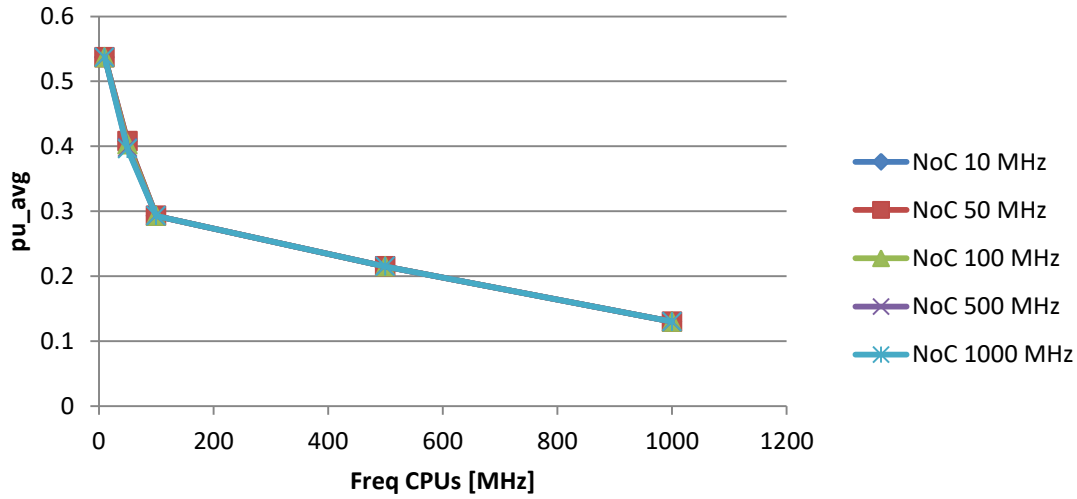


Figure 5.28: This graph shows the PEs average utilization when the NoC frequency has different values along different CPUs frequencies.

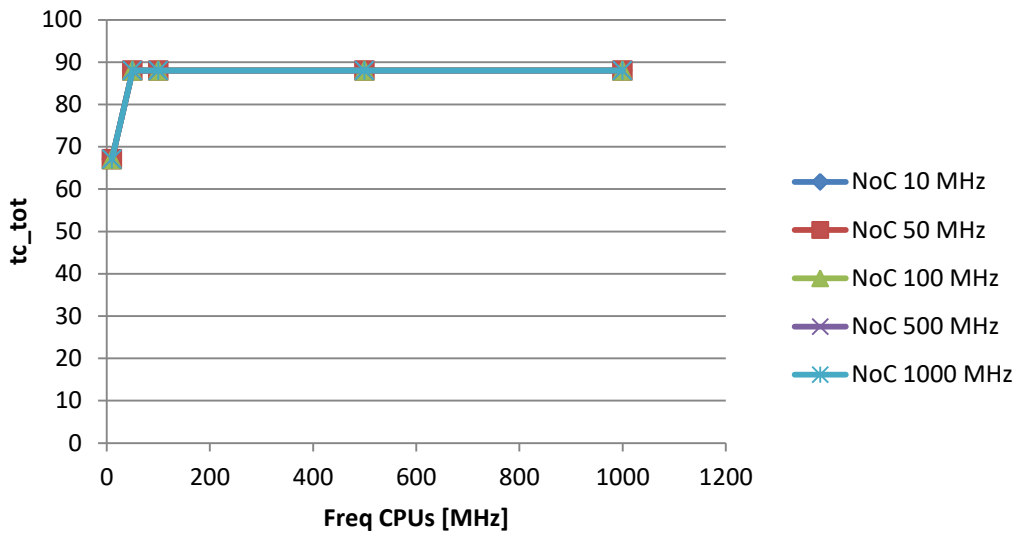


Figure 5.29: This graph shows the amount of tasks executed when the NoC frequency has different values along different CPUs frequencies.

6. CONCLUSIONS

In this chapter it is presented a summary of the results obtained on the parameters analyzed. Table 6.1 shows a table comparing the parameters used with the outputs that have been studied.

	PE utilization	**Tasks executed**
RTL- TLM modelling	RTL involves less PE utilization	No influence
DMA	DMA involves less PE utilization	DMA involves faster completion of all tasks
Packet sizes	Slight difference	No influence
NoC frequencies	No influence	No influence

Table 6.1: Summary of the influence of the parameters on the outputs observed.

RTL-TLM modelling: RTL and TLM modelling have been analyzed with the next traffic models: Av_bench, Mpeg4_decoder and H.264_decoder. RTL modelling involves less PE utilization than TLM and the execution is faster. Nevertheless, the amount of tasks executed is the same with both modellings.

DMA: The influence of the DMA has been analyzed with the next traffic models: Av_bench, Test_mesh, Radio_sys, H.264_decoder and Robot. The presence of the DMA involves less PE utilization and the amount of tasks executed is higher than when the DMA is not active.

Packet sizes: Packet sizes is a parameter analyzed in the next traffic models: Av_bench, Mpeg4_decoder and Radio_sys. Different packet sizes involve a little difference on the PE utilization. Usually one of the packet sizes implies less PE utilization and the other packet sizes have the same PE utilization. Nonetheless, packet sizes have no influence on the amount of tasks executed. Besides, packet sizes of 2 and 4 bytes have problems at high frequencies and do not give data.

NoC frequencies: This attribute has been tested on the next traffic models: Test_mesh and Robot. The impact of the NoC frequency on the PE utilization is null. This applies also on the amount of tasks executed.

REFERENCES

[1] T. Bray, J. Paoli, C.M Sperberg-McQueen, E.Maler, F. Yergeau (eds.) *Extensible Markup Language (XML) 1.0 (Fifth Edition)* http://www.w3.org/TR/REC-xml/#sec-origin-goals Referenced 26.11.2014

[2] T. Kangas, J. Riihimaki, E. Salminen, K. Kuusilinna, T. D. Hämäläinen, *Using a communication generator in SoC architecture exploration*, International Symposium on System-on-Chip, Pages 105- 108, November 2003.

[3] L.Lehtonen. *Transaction Generator – Tool for Network-on-Chip benchmarking*. Master's Thesis, Tampere University of Technology, 55 pages, 2014

[4] NOCBENCH Project, *Standardization of Benchmarking Methodology for Network-on-Chip*, http://www.tkt.cs.tut.fi/research/nocbench Referenced 26.11.2014

[5] A. Moonen, M. Bekooij, R. van den Berg and J. van Meerbergen, *Evaluation of the throughput computed with a dataow model - A case study*, Eindhoven University of Technology, Department of Electrical Engineering, Electronic Systems, ISSN 1574-9517, 2007.

[6] Boost, Boost C++ Libraries, http://www.boost.org/ Referenced 25.11.2014

[7] W. J. Dally, B. Towles, *Route packets, not wires: on-chip interconnection networks*, Design Automation Conference (DAC), Pages 684- 689, 2001.

[8] P. Guerrier, A. Greiner, *A generic architecture for on-chip packet-switched interconnections*, Design, Automation and Test in Europe Conference and Exhibition 2000, Pages 250-256

[9] T. Bjerregaard, A. Mahadevan, *A survey of research and practices of Network-on-chip*, Journal ACM Comput. Surv, Volume 38, Issue 1, ISSN 0360-0300, ACM, New York, NY, USA, June 2006.

[10] M. Gries, *Methods for Evaluating and Covering the Design Space during Early Design Development*, Integration, the VLSI Journal, Volume 38, Pages 131- 183, 2003

[11] A. Sangiovanni-Vincentelli, *Quo Vadis, SLD? Reasoning About the Trends and Challenges of System Level Design*, Proceedings of the IEEE, Volume 95, Number 3, Pages 467- 506, ISSN 0018-9219, March 2007.

[12] NOCBENCH Project. *Transaction Generator 2.* Referenced 08.12.2014 http://www.tkt.cs.tut.fi/research/nocbench/download.html

[13] E. Pekkarinen, L. Lehtonen, E. Salminen, T. Hämäläinen *A Set of Traffic Models for Network-on-Chip Benchmarking on Network-on-Chip Simulation*, Norchip Conference, Tampere, Finland, Nov. 2011, pp 78-81

[14] W. Liu, J. Xu, X. Wu, Y. Ye, X. Wang, W. Zhang, M. Nikdast, Z. Wang, *A NoC Traffic Suite Based on Real Applications,* IEEE Computer Society Annual Symposium on VLSI (ISVLSI), July 2011

[15] Z. Wang, *Realistic Network-on-Chip Traffic patterns.* Mobile Computing System Lab. http://www.ece.ust.hk/~eexu/index_files/traffic.htm Referenced 08.12.2014

[16] J. Hu, U. Ogras, and R. Marculescu, *System-level buffer allocation for application-specific networks-on-chip router design*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 25, no. 12, Dec. 2006. pp. 2919–2933

[17] D. Bertozzi, A. Jalabert, S. Murali, R. Tamhankar, L. Benini and G. de Micheli. *Noc synthesis flow for customized domain specific multiprocessor systems-on-chip*. IEEE Trans. Parallel Distrib. Syst. vol. 16. no 2. pp. 113-129. Feb. 2005

[18] Z. Lu, A. Jantsch, *TDM virtual-circuit configuration for NoC,* IEEE Transactions on Very Large Scale Integration (VLSI) systems, VOL. 16, NO. 8, Aug. 2008, pp. 1021-1034.