PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

REDUCTION OF ENERGY CONSUMPTION IN MPSOCS THROUGH A DYNAMIC FREQUENCY SCALING TECHNIQUE

THIAGO RAUPP DA ROSA

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science at Pontifícia Universidade Católica do Rio Grande do Sul.

Advisor: Prof. Fernando Gehm Moraes

Porto Alegre, Brazil

January, 2012

R788r Rosa, Thiago Raupp da

Reduction of Energy Consumption in MPSOCS through a Dynamic Frequency Scaling Technique / Thiago Raupp da Rosa. – Porto Alegre, 2012.

85 f.

Diss. (Mestrado) – Fac. de Informática, PUCRS.

Orientador: Prof. Dr. Fernando Gehm Moraes.

1. Informática. 2. Multiprocessadores. 3. Arquitetura de computador. I. Moraes, Fernando Gehm. II. Título.

CDD 004.35

Ficha Catalográfica elaborada pelo
Setor de Tratamento da Informação da BC-PUCRS

Pontifícia Universidade Católica do Rio Grande do Sul
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada *"Reduction of Energy Consumption in MPSOCS Through a Dynamic Frequency Scaling Technique"*, apresentada por Thiago Raupp da Rosa como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, Sistemas Embarcados e Sistemas Digitais, aprovada em 27/03/2012 pela Comissão Examinadora:

[illegible]

Prof. Dr. Fernando Gehm Moraes – Orientador — PPGCC/PUCRS

[illegible]

Prof. Dr. Ney Laert Vilar Calazans – — PGCC/PUCRS

[illegible]

Prof. Dr. Alexandre de Morais Amóry - — PPGCC/PUCRS

[illegible]

Prof. Dr. Marcelo Soares Lubaszewski – — UFRGS

Homologada em 22/06/2012, conforme Ata No. 013 pela Comissão Coordenadora.

[illegible]

Prof. Dr. Paulo Henrique Lemelle Fernandes
Coordenador.

PUCRS
Campus Central
Av. Ipiranga, 6681 – P32– sala 507 – CEP: 90619-900
Fone: (51) 3320-3611 – Fax (51) 3320–3621
E-mail: ppgcc@pucrs.br
www.pucrs.br/facin/pos

AGRADECIMENTOS

O espaço reservado aos agradecimentos acaba tornando-se pequeno quando se relembra tudo que aconteceu ao longo dos 2 anos de mestrado. Muitas pessoas me ajudaram durante esse período que, por alguns meses no início e no fim, não foi de dedicação exclusiva. Além das pessoas, também recebi a ajuda de Deus, que sempre me deu forças e não me deixava abater quando as coisas não iam bem. Tenho certeza que além Dele, meu pai, a quem dedico essa Dissertação, também esteve cuidando de mim lá de cima.

O agradecimento mais especial vai para as pessoas que estiveram mais próximas de mim durante esses 2 anos. À minha mãe, Maria Glória, e à minha irmã Rafaela, obrigado por fazerem tudo por mim, por me apoiarem em todos os momentos e sempre ajudar quando eu mais precisava. À minha namorada, Vivian, obrigado pela paciência, carinho, amor e principalmente compreensão. Saiba que sempre estarei contigo quando precisares. Ao meu tio Nilton e minha tia Marta, que além de tios e vizinhos, são como pai e mãe pra mim, me aconselhando, apoiando em todas as situações, dando carona, emprestando leite quando falta aqui em casa, etc... :-) Muito obrigado também a todos os outro familiares, por estarem sempre unidos e formarem uma família tão bonita quanto a nossa.

Gostaria também de agradecer aos professores do PPGCC e avaliadores que me ajudaram a crescer pessoal, profissional e cientificamente, dentre eles, Ney Calazans, Alexandre Amory, César Marcon e Marcelo Lubaszewski. Além desses, um agradecimento especial ao meu orientador Fernando Moraes, que me atura desde 2006, quando era bolsista de iniciação científica. Muito obrigado por todos esses anos de aprendizado, por todos os conselhos e também por me “descolar” uma bolsa de doutorado na França. Espero que ainda possamos trabalhar juntos por vários anos.

Agradeço também a todos os amigos e colegas de laboratório que de certa forma contribuíram para a realização deste trabalho. Ao longo dos 6 anos trabalhando no GAPH, recebi ajuda de todos os membros, por isso não citarei nomes, pois a lista seria longa. Por isso, todos que trabalharam comigo sintam-se agradecidos, de bolsistas IC a doutorandos. Entretanto, não poderia deixar de citar os dois bolsistas que me ajudaram no início e fim do mestrado, Douglas Cardoso e Vivian Larréa, muito obrigado por toda a ajuda que foi de grande valia. Também não poderia deixar de citar os dois amigos que me ajudaram com a revisão do texto: Leticia Moreira e Luciano Ost, sem vocês essa dissertação com certeza seria em português, muitíssimo obrigado. Agradeço também aos colegas e amigos do CI-Brasil, os quais seguraram a barra com deadlines e trabalhos quando eu não pude estar presente, sem vocês ou eu não teria dissertação ou não teria bolsa, muito obrigado.

Por fim, gostaria de agradecer ao CNPq e a DELL, por financiarem o projeto ao qual esse trabalho está inserido.

A todos, de coração, muitíssimo obrigado.

REDUÇÃO DO CONSUMO DE ENERGIA EM MPSOCS BASEADOS EM NOC ATRAVÉS DA TÉCNICA DE ESCALONAMENTO DINÂMICO DE FREQUÊNCIA

RESUMO

MPSoCs baseados em NoC têm sido empregados em sistemas embarcados devido ao seu alto desempenho, atingido através do uso de múltiplos elementos de processamento (PEs). Entretanto, a especificação da funcionalidade, agregada a especificação de requisitos de consumo de energia em aplicações móveis, pode comprometer o processo de projeto em termos de tempo e/ou custo. Dessa forma, a utilização de técnicas para gerenciamento de consumo de energia é essencial. Além disso, aplicações que possuam carga de trabalho dinâmica podem realizar esse gerenciamento dinamicamente. A utilização de técnicas para escalonamento dinâmico de tensão e frequência (DVFS) mostrou-se adequada para a redução do consumo de energia em sistemas computacionais. No entanto, devido à evolução da tecnologia, a variação mínima de tensão é menor, e o tempo de resposta elevado dos métodos de DVFS pode tornar esta técnica inadequada em tecnologias DSM (*deep sub-micron*). Como alternativa, a utilização de técnicas para escalonamento dinâmico de frequência (DFS) pode prover uma boa relação custo-benefício entre economia e consumo de energia.

O presente trabalho apresenta um esquema de escalonamento dinâmico de frequência distribuído auto-adaptável para MPSoCs baseados em NoC. Ambos os elementos do MPSoC (NoC e PEs) possuem um esquema específico. O esquema para os PEs leva em consideração as cargas de computação e comunicação do mesmo. Na NoC, o esquema é controlado através de informações provenientes do pacote que trafega na rede e da atividade do roteador. Além disso, um módulo para geração local de relógio é apresentado, o qual é responsável por prover o sinal de relógio para PEs e roteadores da NoC. O esquema de geração do sinal de relógio é simples, baseado em roubo de ciclo de um sinal de relógio global. Este ainda fornece uma ampla variedade de frequências, induz baixo custo adicional de área e consumo e responde rapidamente a uma nova configuração.

Para avaliar o esquema proposto, aplicações sintéticas e reais foram simuladas. Os resultados mostram que a redução no número de instruções executadas é de até 65% (28% em média), com um custo adicional de no máximo 14% no tempo de execução (9% em média). Em relação à dissipação de potência, os resultados mostram que a redução é de até 52% nos PEs (23% em média) e de até 76% na NoC (71% em média). O overhead de consumo apresentado pelo esquema dos PEs é de 3% e pelo esquema da NoC é de 10%.

Palavras-Chave: DFS, DVFS, MPSoC, NoC, gerenciamento de consumo, redução do consumo de energia.

REDUCTION OF ENERGY CONSUMPTION IN MPSOCS THROUGH DYNAMIC FREQUENCY SCALING TECHNIQUE

ABSTRACT

NoC-based MPSoCs are employed in several embedded systems due to the high performance, achieved by using multiple processing elements (PEs). However, power and energy restrictions, especially in mobile applications, may render the design of MPSoCs over-constrained. Thus, the use of power management techniques is mandatory. Moreover, due to the high variability present in application workloads executed by these devices, this management must be performed dynamically. The use of traditional dynamic voltage and frequency scaling (DVFS) techniques proved to be useful in several scenarios to save energy. Nonetheless, due to technology scaling that limits the voltage variation and to the slow response of DVFS schemes, the use of such technique may become inadequate in newer DSM technology nodes. As alternative, the use of dynamic frequency scaling (DFS) may provide a good trade-off between power savings and power overhead.

This work proposes a self-adaptable distributed DFS scheme for NoC-Based MPSoCs. Both NoC and PEs have an individual frequency control scheme. The DFS scheme for PEs takes into account the PE computation and communication loads to dynamically change the operating frequency. In the NoC, a DFS controller uses packet information and router activity to decide the router operating frequency. Also, the clock generation module is designed to provide a clock signal to PEs and NoC routers. The clock generation method is simple, based on local selective clock gating of a single global clock, provides a wide range of generated clocks, induces low area and power overheads and presents small response time.

Synthetic and real applications were used to evaluate the proposed scheme. Results show that the number of executed instructions can be reduced by 65% (28% in average), with an execution time overhead up to only 14% (9% in average). The consequent power dissipation reduction in PEs reaches up to 52% (23% in average) and in the NoC up to 76% (71% in average). The power overhead induced by the proposed scheme is around 3% in PEs and around 10% in the NoC.

Keywords: DFS, DVFS, MPSoC, NoC, power management, power reduction.

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

ASIC	Application-Specific Integrated Circuit
CMOS	Complementary Metal-Oxide Semiconductor
CPU	Central Processing Unit
DC-DC	Direct Current – Direct Current
DFS	Dynamic Frequency Scaling
DFT	Design for Testability
DMA	Direct Memory Access
DPM	Dynamic Power Management
DSM	Deep Sub-micron
DVFS	Dynamic Voltage and Frequency Scaling
FIFO	First-In First-Out
FL	Fuzzy Logic
FPGA	Field-Programmable Gate Array
FSM	Finite State Machine
GALS	Globally Asynchronous, Locally Synchronous
HeMPS	Hermes Multiprocessor System
IP	Intellectual Property
JPEG	Joint Photographic Experts Group
LUT	Look-up Table
MPEG	Moving Pictures Expert Group
MPSoC	Multiprocessor System-on-Chip
NI	Network Interface
NoC	Network-on-Chip
OS	Operating System
PE	Processing Element
PID	Proportional–Integral–Derivative
PLL	Phase-Locked Loop
PM	Power Manager
PMU	Power Management Unit
RAM	Random-Access Memory
RFI	Recompute Frequency Interval
RTOS	Real Time Operating System
SoC	System-on-Chip
VFI	Voltage-Frequency Island

VHDL	Very High Speed Integrated Circuits (VHSIC) Hardware Description Language
VOPD	Video Object Plane Decoder

CONTENTS

1 INTRODUCTION

The continuous advance of CMOS technology has enabled the design of high complexity systems. Nowadays, a chip may contain several IPs (Intellectual Property), presenting different functionalities, being called a SoC (System-on-Chip). When a massive computing power is needed, several processors may be employed together, composing an MPSoC (Multiprocessor System-on-Chip). These systems need an interconnection infrastructure to allow modules to communicate. Networks-on-chip (NoCs) are an attractive solution for on-chip communication, providing flexibility and scalability [LEE07] [SHA08]. Thus, NoC-based MPSoCs can provide massive computing power on a single chip, while executing several applications in parallel.

However, high performance systems usually require elevated energy consumption. Indeed, some devices, such as smartphones and tablets, need high performance to deliver multiple services and simultaneously provide several alternative functionalities, while meeting a given power or energy constraint imposed by battery autonomy limitations. Moreover, drawbacks such as increasing leakage current and power density keeps getting worse at each technology node. Furthermore, battery technology does not keep up with the increasing power consumption in complex designs. Hence, designers have to select the best trade-off between performance and energy consumption for the set of applications the system will execute [BEN00].

To achieve a good trade-off between performance and energy consumption, a common way to design systems is to meet minimum performance constraints for a given set of applications, and then, achieve the desired power or energy consumption through some power reduction technique. The energy consumption in systems implemented using CMOS technology is related to the operating frequency, supplied voltage, load capacitance and switching activity. Most techniques rely on controlling the supplied voltage and operation frequency, which can significantly reduce energy consumption.

Still, devices need to deliver the maximum achievable performance when requested. However, this performance is required only in some intervals. Thus, at some point of execution, the system can adjust its operating conditions to consume less energy. The ability of tuning system performance according to the required performance is the key to achieve an energy-efficient system [BEN00]. Controlling the device operating conditions to reduce power consumption is called *power management*. It can be guided by workload characteristics extracted at design time or at runtime. An effective power management using design time data needs an encompassing design space exploration, requiring a significant amount of time in complex designs. Also, manufacturing process variability and circuit aging may lead to malfunction in this kind of methodology. On the other hand, a dynamic power management, which monitors the workload at runtime, may provide better results for systems with high workload variability, which is present in most modern systems.

1.1 Dynamic Power Management

Dynamic power management (DPM) is employed to reconfigure an electronic system to provide required services and performance levels with a minimum number of active components and/or a minimum load on such components [BEN98] [LOR98]. DPM comprises a set of techniques and/or mechanisms to build energy-efficient environments. This is done by turning off or reducing the performance of given system components, which are not fully explored. There are two main premises for the applicability of DPM: (*i*) the system and its components present non-uniform workloads during operation and (*ii*) it is possible to predict, with certain level of accuracy, the oscillation in the workload.

DPM mechanisms differ according to the element aggregation level (e.g., component, system, network) where DPM is applied, as well as on its physical realization style (e.g., timer, hard-wired controller, software routine) [BEN00]. The agent responsible for taking the decisions is called *power manager* (PM). Usually, decisions are based on a set of parameters or rules, often called *policy*. An example of policy is the one used in mobile devices. In this kind of device, some parts can be turned off due to low workload or after a given inactivity time.

Nevertheless, the way the system employs the DPM, the use of dynamic reconfiguration and the chosen policy may present significant impact in the overall power consumption. Thus, the design of an efficient PMs is decisive for maximizing power saving.

1.2 DFS and DVFS

Controlling two main parameters can reduce energy consumption in CMOS circuits: the supplied voltage and the operating frequency. The latter has a linear impact on energy consumption, and a given frequency can only be sustained with some minimum voltage supply. On the other hand, voltage has a quadratic impact on energy consumption. Accordingly, this is the most used factor to reduce energy consumption. Controlling these two variables at runtime is the basis of the Dynamic Voltage and Frequency Scaling (DVFS) techniques. As an alternative, it is possible to control only the operating frequency (Dynamic Frequency Scaling – DFS). DFS is an interesting solution when the area overhead added by the voltage control unit becomes a problem, when the design needs a simple and very fast switch between operation modes, or when there are no hardware mechanisms to support voltage switching, like in most current FPGA devices.

DVFS and DFS techniques can be controlled by hardware components alone, by software algorithms, or by some hybrid hardware-software configuration. When using a pure hardware control, sensors and dedicated monitors can collect information of hardware (e.g temperature, throughput, latency), providing it to the power manager. On the other hand, the operating system (OS) or the *microkernel* is the main engine for pure software control. In this case, the OS has to process all the available data, working as power manager, informing the hardware the desired voltage and/or frequency. Using a mixed approach, the hardware may collect information and provide it to the software, which only becomes responsible for processing control data. This approach presents a small hardware overhead (the hardware does not need to process any data) while providing more accurate information to the software, which processes the data and informs the hardware the actions to be taken. Moreover, the control scheme can be *centralized* or *distributed*. A centralized scheme uses global data to take decisions, while distributed schemes use only local data.

On NoC-based MPSoCs, the DVFS scheme can be introduced into the processing element (PE), in the NoC router or in both. Introducing a DVFS scheme on only one component of the MPSoC may not exploit all potential of power savings. Indeed, most approaches couple the processing element with the router, dealing with it as a single module. Ideally, to explore maximum power savings, the router and processing element should be managed individually, according to their own specific characteristics.

As a drawback of DVFS, the scaling of technology may result in problems like worsening manufacturing process variations. These variations can result in nominally correct DVFS schemes failing to meet frequency or power targets [GAR09] [HEB09]. Therefore, DVFS techniques must cope with design variability in nanoscale technologies to guarantee correct system behavior. In addition, the most recent technologies (45nm and below) are restricting the supply voltage scaling margins, which is the key component behind power savings through DVFS [CHA11]. Thus, designing the system to work at a fixed supply voltage, coupled to a DFS method may be an option to efficiently manage energy consumption in DSM technologies.

1.3 GOALS

The main objective of the present work is to propose and to evaluate a new DFS technique, targeting power dissipation reduction in NoC-Based MPSoCs. The proposed technique is designed to be applied to both processing elements and NoC router in separate, which are the main components of NoC-Based MPSoCs. In processing elements, control is based on tasks workload and individual tasks communication load. In the router, control acts according to packet information and router activity.

The specific goals of the present work include:

- Adjust the communication interface between processor and router to the GALS paradigm;
- Design a clock generation scheme to provide different clock frequencies to the modules (PEs and routers);
- Define a set of rules to guide the DFS controlling scheme behavior in the PE;
- Adapt NoC routers to communicate using different frequencies;
- Design a mechanism for frequency switching in the NoC;
- Implement a set of benchmarks to be executed in the HeMPS Platform;
- Propose a policy to handle DFS with multitask execution;
- Evaluate the proposed techniques considering the power dissipation and reduction in the number of executed instructions, which can be used as energy consumption metric in higher abstraction levels.

1.4 DOCUMENT OUTLINE

The remainder of this document is organized as follows. Chapter 2 reviews the state of the art, classifying it according to the system level where the DFS, or DVFS techniques are applied. This Chapter also compares the reviewed approaches with the proposed technique. Chapter 3 describes the target NoC-MPSoC architecture and the required modifications to enable the proposed DFS technique. Next, in Chapter 4, the processor control scheme, multitask handling policy and the local clock generation module are presented. Chapter 5 describes the NoC router control scheme and the proposed adaptation of the work [GUI08], as well. The experimental setup and evaluation of the proposed technique is depicted in Chapter 6. Finally, conclusions and future works are described in Chapter 7.

2 RELATED WORK

This Chapter surveys the state-of-the-art in Dynamic Voltage and Frequency Scaling techniques according to the system level that it is applied. As mentioned before, DVFS can be applied in any of main elements of NoC-Based MPSoCs (processors and routers). The following Sections discuss DVFS techniques applied to both elements of NoC-Based MPSoCs independently, as well as to the entire system, also covering other architecture types. Lastly, the present work is positioned with regard to the works presented in this Chapter.

2.1 DVFS IN MICROPROCESSORS

DVFS techniques in microprocessors are a well-known method to reduce power consumption. Currently, a significant portion of commercial processors can operate at multiple levels of voltage and frequency. In these processors, a key factor that determines if the voltage and/or frequency have to be scaled up or down is the workload. Basically, when the processor is in idle state, i.e. not executing any job or task, voltage and frequency can be scaled down. At the moment some task is scheduled, the operating voltage and frequency are scaled up.

[SHA09] et al. present a feedback control algorithm that uses DVFS for power savings in processors, depicted in Figure 1. The main idea of the algorithm is to keep the processor operating at its maximum processing capacity (maximum utilization), meeting application constraints while saving as much power as possible. The algorithm is implemented in software and depends on continuous online measurement of the CPU load and a specific hardware to adjust the operation voltage and frequency. The negative feedback loop reduces the processor voltage and frequency when the CPU is not fully utilized. When the application requires higher performance, the controller increases operating voltage and frequency.

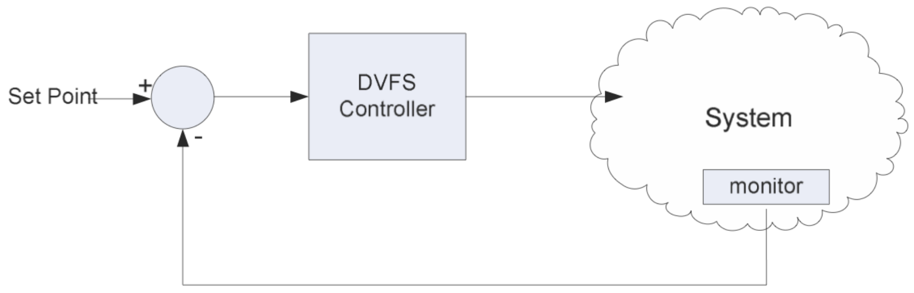


Figure 1 – Negative feedback control DVFS scheme.

The CPU load is measured using a task called IDLE. This task is scheduled when there is no task for the processor to execute. Therefore, it is possible to measure the CPU utilization (load) by calculating how much time the IDLE task executes in a given sampling period. If the amount of time that the IDLE task is executed is greater than zero, it means that the processor load is less than 100%. However, it cannot detect if the processor is over utilized, since the value of 100% may denote either full or overload utilization.

With this information, the algorithm is capable of choosing a suitable voltage-frequency pair. Still, the algorithm follows some rules. If the chosen voltage and/or frequency are not available, e.g. the system only provides three different levels and chosen pair is between two of the available levels, the applied voltage and frequency will be the closest available pair in the system with voltage and frequency higher than the chosen pair. Also, if performance has to be increased, voltage is increased before frequency and vice-versa, in case of decrease in performance. Lastly, when the utilization is at 100%, voltage and frequency are always increased, due to the uncertainty in the measurement, which cannot state if the processor utilization is at 100% or at a higher value. Using this algorithm, the Authors presented power savings between 5% and 24% in an architecture that allows only frequency scaling, for real and synthetic workloads. In addition, the Authors claim that using the proposed scheme in a system that allows voltage scaling can present greater power savings.

Similarly, Pourshaghaghi and de Gyvez [POU09] present an adaptive framework for DVFS in processors that work with different application workloads. Voltage scaling decisions are made by a fuzzy logic (FL) block, based on workload variations. Since there is a strong correlation between supply current and workload [BEN99], the proposed scheme is based on monitoring the supply current of the processor and its variations. The objective of using the variations of the supply current is to predict the variations in the workload, making the controller act beforehand. Figure 2 shows the configuration of the scheme proposed in [POU09]. The current sensors are responsible for measuring the supply current used by the processor and providing the values for the FL controller and the module Derivative, which calculates variations on the supply current. The FL controller is composed by three main blocks: (i) a fuzzification block, responsible for converting a quantified numerical or a control variable into a qualitative value (membership functions); (ii) a fuzzy engine, containing a set of if-then rules that gather the knowledge needed to successfully control the output; and (iii) a deffuzzification block, that converts the resulting fuzzy set into a number to be sent as a control signal. After calculating the voltage and frequency values, the controller acts over the hardware, which commands the circuit operation. In this work there is no mention about the hardware used for voltage and frequency switching. However, the Authors suggest that a DC-DC converter can be used for voltage switching and a look-up-table (LUT) can be used to choose a frequency within a pre-defined set.

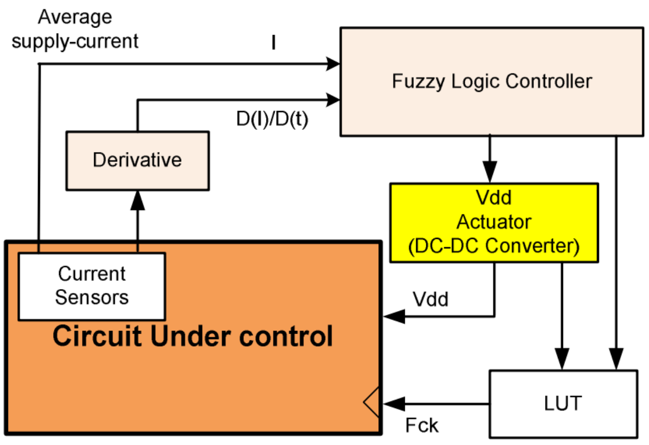


Figure 2 – Pourshaghaghi and de Gyvez [POU09] DVFS configuration.

Results show that the fuzzy controller can track the supply current better than a PID controller, especially when the supply current presents high variability. When the supply current varies, compared to the supply current used to calibrate the PID controller, the PID coefficients are not optimal anymore and new coefficients need to be calculated. Moreover, the fuzzy logic controller can work on-line to track all workload (supply-current) circumstances with higher speed and smaller error.

Shu and Li [SHU10] et al. propose a DVFS controlling scheme based on the current processor temperature, thermal state, and tasks characteristics. The goal is to minimize total energy consumption taking into account the temperature, due to the fact that the leakage power has an exponential dependence on temperature, and the decrease in the size of devices makes the leakage current increases, especially in DSM technologies.

The proposed scheme relies on a platform that provides a Performance Monitor Unit (PMU) and on-chip temperature sensors. The algorithm is implemented in software and the set of tasks is modeled with periodic and aperiodic tasks. Information like number of executed instructions, data and instruction cache misses and number of cycles to execute a given task is used by the algorithm. The used system has two modes: active and sleep. The time to change from one mode to another is non-negligible, although the time to switch from one voltage (frequency) level to another is negligible. The thermal state of the system is calculated through the current temperature, power dissipation in a fixed period, ambient temperature and electrical characteristics of the circuit. Then, based on the collected information, the algorithm finds the optimal operating frequency that fits the parameter values provided (temperature and performance loss).

Results show the amount of energy savings are highly impacted by the actual available voltage-frequency pairs on chip, the performance requirement and the application execution time. In the worst case, where performance loss is 5%, the proposed algorithm saves up to 8% of energy, while in the best case, where performance loss is 20%, the amount of energy savings reaches 37%. On average, the total energy saving achieves 21.6%.

Table 1 summarizes the characteristics of the studied works that present solutions for DVFS in microprocessors. The main difference is between the Shu and Li work and the other two. While in [SHU10] the aim is to reduce power by meeting a given performance loss and temperature, the others aim to reduce power dissipation simply based on the workload. The algorithm proposed in [SHU10] works with a restricted number of tasks and fits better in systems where applications to executed are well defined, while the other two approaches cover systems that cannot predict the applications that will execute and present high variability in the workload. Also, the scheme proposed in [SHA09] and [POU09] are simpler to implement and may use hardware or software structures. A common point between all works is that there is no concern about implementation of the specific hardware support for DVFS. [SHA09] and [SHU10] used a platform which provides frequency scaling support, while [POU09] compared the output of the FL controller to a PID controller.

Table 1 – Microprocessors DVFS schemes comparison.

Author	Monitoring Parameter	Implementation	Hardware Support
Shalan and El-Sissy [SHA09]	CPU Utilization	Software	Native
Pourshaghaghi and de Gyvez [POU09]	CPU Supply Current	Hardware	Proposed, not Implemented
Shu and Li [SHU10]	Tasks characteristics, Temperature, Thermal State	Software	Native

2.2 DVFS in NoCs

The router is the main component of a NoC. In the studied works the routers are treated independently, i.e. each router has an individual DVFS controlling scheme or algorithm. This approach is interesting due to the fact that hardly ever all routers of the NoC will be operating at the same time, making it possible that inactive routers have its voltage and frequency decreased.

Pontes et al. [PON08] use flow priority to control the operating frequency of the routers in the Hermes-GLP NoC. Besides DFS, the Authors also use a clock-gating technique to help in power reduction when there is no activity in the router. A module called clock control is responsible for deciding the operating frequency of the router or applying clock gating. This module receives information from the input ports to take the decisions. Only active ports are taken into account to determine the router frequency. Figure 3 shows an example of execution.

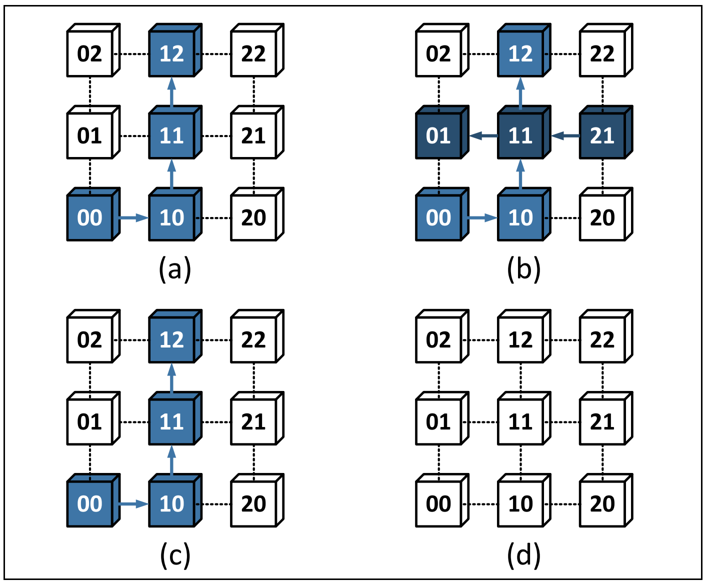


Figure 3 – Multiple priorities scenario in Hermes-GLP.

In Figure 3(a), the NoC routes a flow of low priority. Next (Figure 3(b)), a flow of higher priority is introduced in the network, making the clock control increase the frequency of the router which is routing both flows (router 11), in order to meet the requirements of the higher priority flow. Later, when the flow of higher priority ends (Figure 3(c)), the central router goes back to the lower frequency and the clocks of routers 21 and 01 are gated. Finally, after the first flow is finished, all routers have their clocks gated (Figure 3(d)). Results show the proposed scheme reduces significantly the activation rate of the router when compared to the Hermes-G router

[PON08], with negligible increase in latency. Even without power or energy evaluation, the Authors affirm the scheme allows a significant reduction in power consumption due to the reduction in the activation rate of the routers.

Another way to control the operating frequency is by monitoring the communication load of each router, as done in [MIS09], [YIN09], [OGR08], [OGR09] and [GAR10]. Mishra et al. [MIS09] propose an algorithm for DVFS based in the occupation of queues. The Authors present three techniques to dynamically change the operating frequency of routers. The first technique is called *FreqBoost*, which consists in operating the whole system at a higher frequency and decrease the frequency of routers that are generating congestion in the network, as illustrated in Figure 4.

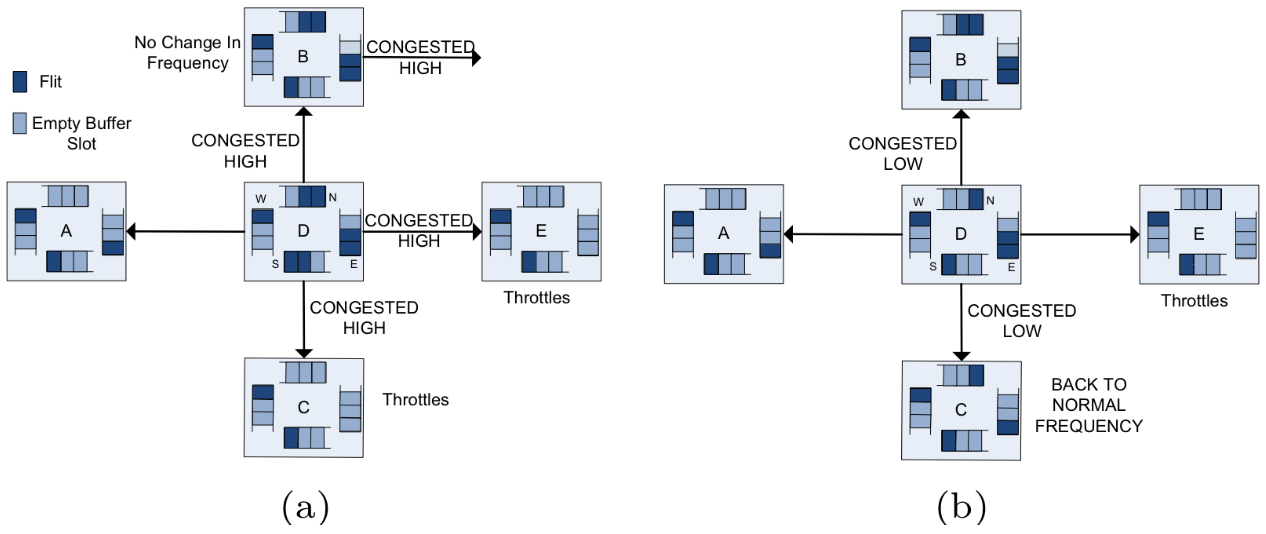


Figure 4 – Scenario with traffic congestion. (a) FreqBoost technique is applied. (b) Neighbors of router D with low congestion in its respective ports back to the original frequency.

In Figure 4(a) the central router (D) is congested. Ports east, south and north are almost full, being fed by routers E, C and B, respectively. First, router D informs its neighbors that it is congested. Then, the routers C and E verify they are not overloaded and can decrease their frequency, reducing the congestion in router D. However, router B verifies that it is congested and cannot decrease its frequency. In Figure 4(b) router D informs routers C and B that their respective ports are not congested anymore, making them return to the original frequency while router E keeps its frequency down.

The second technique is called *FreqThrtl* and consists in operating the system at higher frequencies only when the possibility of congestion is verified. When the congestion is detected, the router has its frequency increased and the routers that are generating the congestion have their frequency reduced. This technique presented reduction in energy consumption, while the technique *FreqBoost* presented reduction in latency. Then, a third technique is proposed by coupling both techniques previously mentioned, called *FreqTune*. This technique exploits the benefits of the other two, presenting reduction in energy consumption and better performance.

Similarly, Yin et al. [YIN09] present an algorithm for DVFS based on the buffers load of each router. In this work, a bisynchronous FIFO [PAN07] is used to connect each neighbor router, which synchronizes the communication between routers of different clock domains. Then, the load of each router is measured by the average occupation of the buffers inside the router and connected FIFOs, being called local load. The algorithm uses the local load value to dynamically adjust the supply voltage by comparing it with threshold values defined by the application. Additionally, a scalable DVFS architecture is proposed (Figure 5).

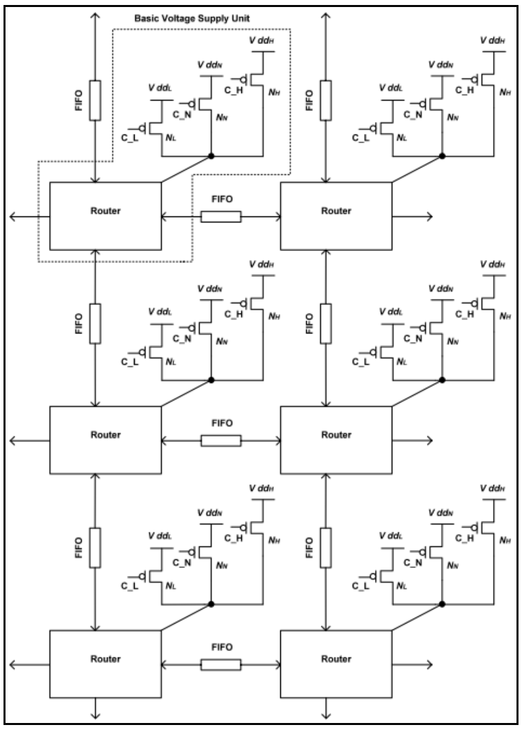


Figure 5 – Architectural view of distributed DVFS on NoCs with multiple voltage supply networks [YIN09].

The proposed architecture works with multiple voltage supply networks and power selection transistors. By using this architecture, the number of voltage levels is restricted, due to the manufacturing feasibility. However, the area and power overhead are negligible when compared to voltage regulators, despite the speed for switching from one level to another. Results show a reduction in energy consumption between 45% and 60%, for synthetic traffic patterns although they do not consider the energy overhead added by FIFOs. The router energy overhead induced by the FIFO ranges from 25% to 31%. Nevertheless, the Authors affirm that using the proposed technique energy benefits are considerable when compared to static voltage setting.

Likewise, in [OGR08], [OGR09] and [GAR10] it is presented a feedback control that acts over queues occupation. The NoC is divided in Voltage-Frequency Islands (VFIs), and the communication between the routers is done via bisynchronous mixed-voltage queues. In [OGR08] and [OGR09] the Authors use a *state-space* matrix, which contains the occupation of each queue in the system. The state-space is modeled into an equation and used by the controller to dynamically change the frequency of the island. However, the limitation of this method relies on the number of queues that can be controlled. Using this controlling scheme, it is possible to control, in the best case, a number of queues equal to the number of VFIs. Experiments showed a reduction of 40% in energy for a hardware MPEG-2 encoder design.

Garg et. al. [GAR10] use the feedback control presented in [OGR08] and [OGR09] to design a new control, called custom feedback, which provides similar performance, while consuming less area. The Authors divide the controllers between fully-centralized and fully-decentralized controls. The fully-centralized control (FC) presents a performance better than the fully-decentralized (FD). However, the fully-decentralized control presents an implementation cost significantly smaller, especially when the system is large, leading to a prohibitive design of global communication.

To efficiently explore the design space between FC and FD control strategies, the Custom Feedback control (CF) is proposed. The CF control is a mix of both FC and FD strategies. In this controlling scheme, the controller uses the information of some queues of the system, instead of using all of them, as it happens for FC. By using the information of some queues of system, the CF control presents a small overhead in implementation cost when compared to the FD, and around the same power saving, when compared to the FC.

Table 2 summarizes the characteristics of the studied works that present solutions for DVFS in NoCs. The main difference is related to the monitoring parameter used by Pontes et. al. [PON08]. Instead of designing a specific controlling scheme, the packet defines the frequency in which the routers will operate. On the other hand, Mishra, Yin, Ogras and Garg provide a generic solution, at cost of area and power overheads. Ogras and Garg et. al. presented a software implementation of the algorithm, though it is said it can be easily implemented in hardware. Only Mishra et. al. presented a hardware implementation for voltage and frequency scaling, while Ogras and Garg et. al. used existing FPGA components to enable frequency scaling, not using voltage scaling. Generally the approaches can be divided into guided by application requirements ([PON08]) and guided by current workload scenario ([MIS09][YIN09][OGR08][OGR09][GAR10]).

Table 2 – Comparison of DVFS techniques for NoCs.

Author	Monitoring Parameter	Implementation	Hardware Support
Pontes [PON08]	Flow Priority	Hardware	Not Implemented
Mishra [MIS09]	Queues Occupation	Hardware	Implemented in ASIC
Yin [YIN09]	Queues Occupation	Hardware	Proposed, not Implemented
Ogras and Garg [OGR08][OGR09][GAR10]	Queues Occupation	Software	Partially Implemented

2.3 DVFS IN NOC-BASED MPSOCS

A NoC-Based MPSoC is generally composed by a set of processing element, which each of usually contains a processor and some peripherals, and a network-on-chip (NoC). Therefore, to apply DVFS in MPSoCs the designer has to take into account some characteristics that do not appear in mono-processor architectures such as task communication by message exchanging, network congestion and several services provided by the architecture.

Gligor et. al. [GLI09] propose a task migration aware DVFS algorithm. The algorithm identifies processor workloads and calculates the frequency needed by the tasks to finish execution before the next deadline. The system workload is estimated by dividing the number of clock cycles the system was active during a given time interval, by the total number of cycles available at the maximum frequency in this time interval. However, processors without useful work only at part of the interval may lead to erroneous analysis when a task is migrated within this interval. To overcome this problem, time intervals are divided in subintervals. At each subinterval the frequency is recomputed using the information of non-idle processors. Thus, if a task is migrated within this time interval to a previously idle processor, the computation in subintervals can handle the new configuration of the system. Figure 6 illustrates a system workload in two time intervals without frequency scaling.

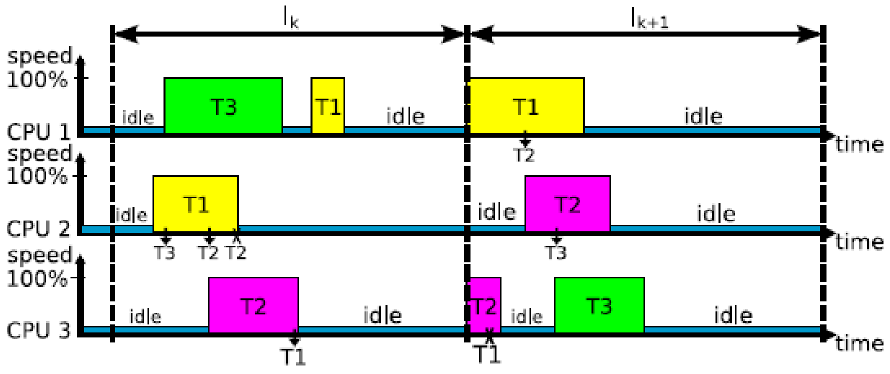


Figure 6 – Intervals I_k and I_{k+1} at maximum frequency with task migration within each interval.

In this example, the calculated system workload is around 33% in the interval I_k. However, setting the frequency at 33% for the entire next interval (I_{k+1}) is not a good solution, because not all processors have useful work to execute during the next interval. For example, idle periods caused by a blockage and task migration costs may interfere in the total system workload. In Figure 7 it is shown how the proposed algorithm works, acting in the subintervals, represented by the dotted lines and called Recompute Frequency Interval (RFI). The operating frequency is defined by the remaining time the task has to finish and how much work needs to be processed. The Authors used a cycle accurate simulation platform to present the results and claim that the energy savings can achieve 55%, but there are no details on how this value is obtained.

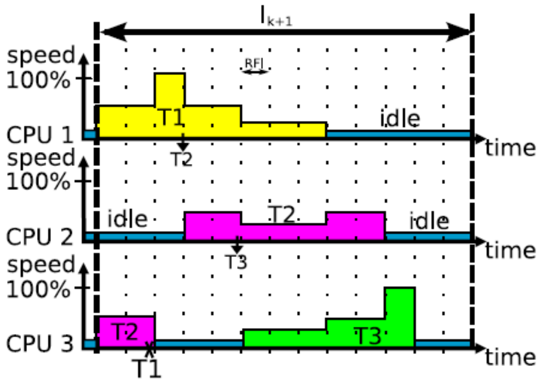


Figure 7 – [GLI09] algorithm evaluates the system workload in each RFI to dynamically change the operating frequency.

Another characteristic present in MPSoCs is the concept of composability. Composability is a term used to refer to a property that the behavior of an application is not affected by the absence or presence of other application(s) executing in the same system. Goossens et. al. [GOO10] present a composable and predictable power management through a composable DVFS hardware. The Authors propose an architecture that uses an existing DVFS hardware to build composable and predictable SoCs running multiple applications. The DVFS scheme is controlled by an RTOS. The execution is divided in *system time slices*, composed by the task time slice and the RTOS time slice. Composability is achieved by making the *system time slice* constant, giving to the tasks of different applications the same amount of execution time, and by programming DVFS operations at precise points in the future.

As done by the Authors of [GLI09], the difference between the remaining budget and remaining work defines the operating frequency for a given task. This means that without slack a task would run at maximum frequency. Otherwise, a lower frequency can be used. Yet, a slack left by a given task can only be used by another task of the same application, to guarantee composability. Results show a reduction of 42% in energy consumption when the technique is compared to only clock-gating idle slices, and a reduction of 68% when compared to no power management in processing elements. No mention about NoC power management is made in this work.

Puschini et al. [PUS08] [PUS09] propose an approach for frequency tuning based on game theory, which considers the state of other processing elements in the MPSoC to compute the decisions. Game theory is a branch of applied mathematics that studies interactions among rational individuals or decision makers. A game is a scenario with several players interacting by actions and consequences. Each action taken by a given player results in consequences or outcomes. Still, each action has as objective to maximize the outcome of a player. The outcome of a player is calculated by a function, in this work called *objective function*, and represented by a score: the higher the score is, the closer it is to the optimal point. In addition, the function that computes the score of a player must consider not only the player's choices, but the choices of other players as well. Accordingly, some system variables, e.g. latency, energy, temperature, were mathematically modeled to be used in an objective function. In the context of these works, a player is represented by a processing element, which targets to achieve the optimum utility (score) value. In [PUS08] the objective is to reach a state where the whole system is in a restricted range of temperature, by changing the frequency of each processing element. In this case, the variables modeled to be used in the objective function were task synchronization and temperature. Results show that the temperature profile was optimized without loss of task synchronization, and the performance of the algorithm is equivalent when compared to an off-line method. In [PUS09] the goal is to optimize energy consumption, scaling the frequency with no latency penalties. Latency and energy were modeled and used in the objective function. Results show reductions between 10% and 25% in energy, compared to nominal frequency assignment, with no latency penalty.

Differently from the previously revised works, Beigné et. al [BEI08] propose a DVFS architecture that can be controlled at system level. There are six modes that can be used by each IP unit, composed by a processor and peripherals, in the system, detailed in Table 3. According to application requirements, the software developer can either use an explicit way to program each IP unit using the HIGH, LOW, IDLE, OFF modes; or use an implicit way to program each IP for a pre-determined average "V_{CORE}" value using the HOPPING mode. In the explicit way, the software must configure at real time the appropriate mode, while in the implicit way, the pre-programmed V_{CORE} value is fully managed by hardware.

Table 3 – System operation modes in [BEI08].

Mode	Description
INIT	At reset, the unit is at VHIGH with no clock
HIGH	The unit is supplied by VHIGH voltage
LOW	The unit is supplied by VLOW voltage
HOPPING	The unit is automatically switched between VHIGH and VLOW voltages, for DVFS
IDLE	The unit is idle, keeping the current state at VLOW voltage, for reduced leakage power
OFF	The unit is switched OFF, without saving the current state, for minimal leakage power

Table 4 summarizes the characteristics of the studied works that present solutions for DVFS in NoC-Based MPSoCs. All works present solutions for DVFS only in the processing element. However, Puschini and Beigné et al. use an asynchronous NoC in the MPSoC. Apart from Beigné et al., the controlling algorithm is related to the task characteristics, similarly to the techniques presented in Section 2.1.

Also, most of the works do not implement the hardware support needed for DVFS. Puschini et al. use a platform that already supports DVFS. Goossens et al. implement a frequency scaling mechanism in FPGA, but do not use voltage scaling, while Gligor et al use a simulation platform to generate the results.

Table 4 – DVFS techniques comparison for NoC-Based MPSoCs.

Author	Monitoring Parameter	Implementation	Hardware Support	Memory Architecture
Gligor et al. [GLI09]	Task Slack	Software	Not Implemented	Shared
Goossens et al. [GOO10]	Task Slack	Hardware	Partially Implemented	Shared
Puschini et al. [PUS08] [PUS09]	Task Synchronization, Latency, Temperature	Software	Native	Distributed
Beigné et al. [BEI08]	-	Hardware	Implemented	Distributed

2.4 DVFS IN BUS-BASED MPSOCS

A bus-based MPSoC is composed by several processing elements, interconnected by a bus. The main difference to NoC-Based MPSoCs is the impossibility of parallel communications, being a non-scalable solution as processor counting increases. However, this kind of architecture is a good solution for MPSoCs with few processing elements. Works that address this kind of architecture for DVFS aims only the processing element as the component to control.

In the architecture used by Alimonda et. al. [ALI06] [ALI09] each processor has an output queue, responsible for storing messages for target processor(s). The Authors propose a non-linear feedback control approach for DVFS in data-flow applications. A data-flow application is composed by a set of tasks, organized as a pipeline. A process is called producer when it provides data to another process. A worker is a process that receives data from a given process and delivers it, with modification or not, to another process. Finally, a process is called consumer when it only receives data from another process. A significant portion of multimedia applications can be modeled as a pipeline.

In the same way it is done in most works presented in Section 2.2, the idea is to keep a constant queue occupation between a consumer and a producer process. The controller acts upon the queue occupation and the error between the current and desired occupation. Also, the interval between each evaluation and the desired queue occupation are configurable. Moreover, besides controlling the queue occupation, the goal is to avoid frequent frequency changes, due to the substantial energy consumed in switching frequencies and the computation delay introduced by this action.

In [ALI06] the controller is presented and a test case with one producer, three workers and one consumer is performed. Results show that the developed strategy consumes 50% less energy compared to the ON-OFF controller, which simply shuts down the processors that have empty or full conditions in queues. In [ALI09] two real applications were evaluated. The results were compared against the Vertigo policy [FLA02], a frequency-setting algorithm, and some energy saving is obtained, without a specified quantitative value. The main result is that using a variable to trigger the controller evaluation, e.g. number of messages stored in the buffer, provides better performance in energy saving and ease of tuning.

Similarly to [GLI09] and [GOO10], Liu et. al. [LIU09] propose an algorithm based in tasks slacks. However, differently from the other approaches that stretch each task execution time in order to occupy the entire deadline period of the application, the latter presents a two-phase DVFS algorithm. The first phase consists in applying the DVFS algorithm to compute a new frequency/voltage pair, based on the critical path of the task graph. After the first DVFS algorithm runs, the task graph is unrolled, exploiting the remaining task's slack to stretch the whole execution time. Then, with the unrolled task graph, the DVFS algorithm is applied again, resulting in a new voltage/frequency level for each processor. Using this algorithm, the Authors show energy savings of 25% in average, reaching approximately 41% for the best case, compared with previous similar approaches.

Basically, Alimonda et. al. presents a generic solution for a wide set of applications, while Liu et. al. propose an algorithm that is based on the characteristics of the tasks that will be executed. However, the first approach does not work in architectures which use blocking communication primitives instead of queues, while the second approach cannot be implemented in systems that execute a large number of applications or generic environments, where there is no knowledge of what application will be executed first.

2.5 Conclusion

Several works addressing DVFS for different architectures were studied. Generally, the adopted DVFS scheme is driven mostly by the architecture where it will be applied. In mono-processor architectures most works use the processor workload to define the operating frequency ([SHA09][POU09]). However, this is not a good solution for multi-processor architectures, because it expresses only the local state, not considering the effects of communication with other processors.

Another monitoring parameter used in several architectures to control the DVFS scheme is the task slack ([SHU10][GLI09][GOO10][LIU09]). Even being similar to a workload computation, some characteristics need to be known prior to execution. These approaches become prohibitive since all the applications should be completely characterized at design time. Nevertheless, the work proposed by Goossens et. al. [GOO10] may present constant frequency switching if two tasks with different characteristics (e.g. CPU intensive and communication intensive) are executing in the same processor, leading to undesirable behavior.

On the other hand, using the queue loads in the architecture to control the DVFS ([MIS09][YIN09][OGR08][OGR09][GAR10][ALI06][ALI09]) makes the applications transparent to the controlling scheme. This solution is more generic and may fit better with several different architectures. By monitoring the queues occupancy of a distributed system, it is possible to observe not only the local state, but also the relation of the PE with its communicating modules. Another way is to provide power management mechanism to a higher abstraction level ([PON08][BEI08]), leaving the power management to the programmer.

As it is not the focus of the studied works, only few mentioned or showed an implementation of the hardware support for DVFS. Also, most of them did not take into account the overhead added by the voltage-frequency switching.

Since the focus of this work is to propose a DFS technique for NoC-Based MPSoCs, the design of the hardware support for DFS is done as a proof of concept. Nonetheless, it is synthesized using a standard cell library and considered in power evaluation. This hardware targets fast frequency switching and low area overhead. A solution that could be adopted is presented in [TSC07], which also provides fast frequency switching. However, the Authors use two extra PLLs in the proposed scheme, inducing large area and power overheads.

Regarding to the proposed technique, the monitoring parameters were chosen according to the target architecture, based in the works presented in this Chapter. Differently from most of the presented works, the target platform employs distributed memory and message exchanging mechanism for task communication. Thus, the set of monitoring parameters differs from those in revised works. As presented, CPU utilization and communication load are used in most works, but always alone. In the target architecture, using only one of them turns the controlling scheme sub-optimal, as presented in next Chapters. Therefore, both monitoring parameters must be considered to design an efficient scheme. Nevertheless, schemes are proposed for controlling the PE and NoC, covering the entire MPSoC structure, which do not appear in any studied work.

3 SYSTEM ARCHITECTURE

This Chapter presents the target architecture of this work and the modifications performed on it to enable the proposed DFS scheme. The target architecture is the Hermes Multiprocessor System-on-Chip (HeMPS) MPSoC [WOS07] [CAR09], which will be described in Section 3.1. Section 3.2 details the adaptations and modifications performed in the modules of HeMPS to enable the DFS.

3.1 ORIGINAL HEMPS ARCHITECTURE

HeMPS is a homogenous NoC-Based MPSoC, which is composed by two main modules: (*i*) Plasma-IPs (the PEs) and (*ii*) the Hermes NoC [MOR04]. Figure 8 shows an instance of size 2 by 2 of HeMPS. Each PE includes the following modules: (*i*) a 32-bit Plasma processor [RHO09]; (*ii*) a local memory (RAM); (*iii*) a DMA module; (*iv*) a network interface (NI). Two types of PEs are used: slave and master. Slave-PEs (Plasma-IP SL) are responsible for executing application tasks, while the single Master-PE (Plasma-IP MP) is responsible for managing task mapping and system debug. The task repository is an external memory, responsible for storing all the object codes of applications that will eventually be executed. The following subsections detail each of the HeMPS main modules.

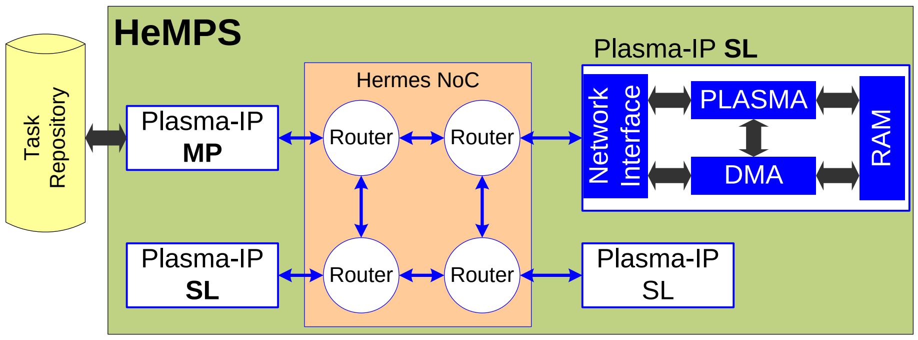


Figure 8 – HeMPS Block diagram.

3.1.1 HERMES NOC

Hermes is a 2D mesh NoC used to interconnect PEs of the HeMPS MPSoC. A NoC was chosen as interconnection architecture due to the several advantages that it provides to MPSoC designs, e.g. scalability and parallel communication [WOS07]. Hermes uses packet switching and wormhole routing mechanisms, where each router is responsible for calculating the next destination of each packet (packet switching), sending it flit by flit (wormhole routing). Moreover, Hermes uses credit-based flow control, where data is transmitted only when the router signs availability to receive it. The employed routing algorithm is the XY, which routes the packet first in horizontal direction, and then in the vertical direction.

Each router is composed by up to 5 input buffers, an arbiter and a crossbar. Input buffers are responsible for storing incoming flits from other routers (East, West, North and South ports) and the IP module connected to the router (Local port). After the first flit of the packet is received (header flit), the input buffer request access to the arbiter and waits the acknowledgement. The arbiter is responsible for managing requests from input ports using a Round-Robin policy, which serves requests in a cyclic order with no priority, and routes the packet. When a request is served, the crossbar connects the output of the input buffer to the chosen output port.

3.1.2 PLASMA-IP

The Plasma-IP is the PE of HeMPS, and its block diagram is presented in Figure 9. The Plasma CPU [RHO09] is the processor of the PE. The local RAM is a dual-port memory, responsible for storing the object code of the microkernel and the applications tasks. It is divided into pages of 16KB or 32KB, being the first page allocated for the microkernel and the remaining for the applications tasks. The size of the memory is configurable in the HeMPS Platform [CAR09], although, in this work the memory size is fixed in 64KB. The DMA module is responsible for transferring the task object code received from the NoC to the memory of a Slave-PE and messages to/from the NoC from/to the local memory. The Network Interface (NI) module is responsible for interfacing PE and NoC, sending and receiving packets. Also, the NI informs the PE its address in the system and interrupts the processor when a new message is available. Memory-Mapped Registers are used to make hardware information available to the microkernel. Additionally, the Master-PE contains the Repository Access block, which is used to read data from the task repository.

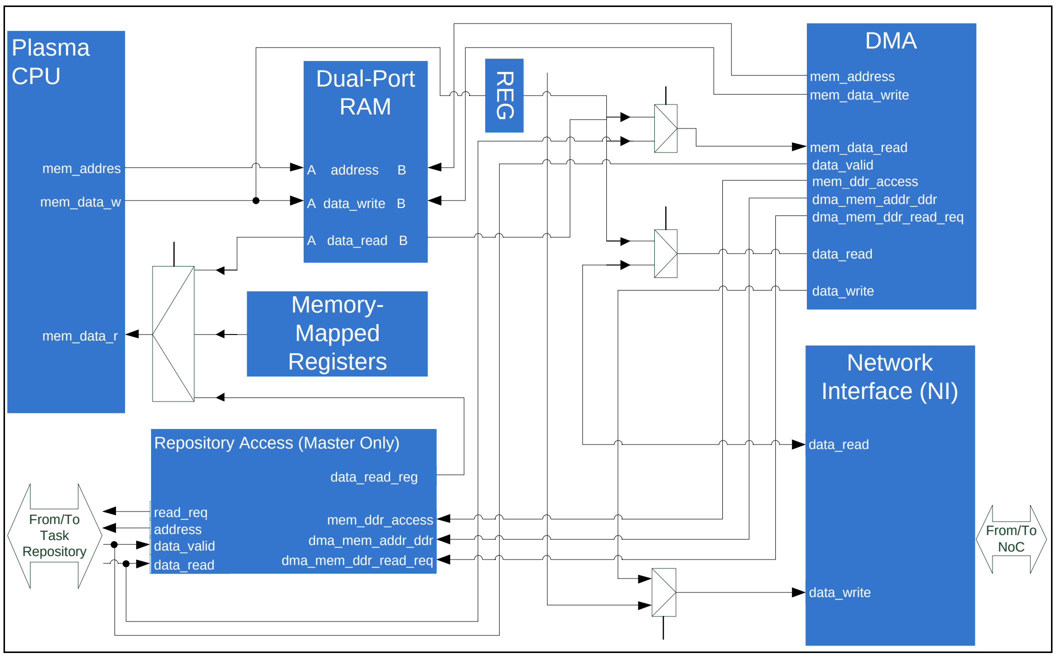


Figure 9 – Plasma-IP Processing Element.

HeMPS uses a distributed memory architecture. The communication between the tasks is made through message exchanging. Two primitives to send and receive data are available, respectively called *Send()* and *Receive()*. The *Send()* primitive is non-blocking, i.e. the processor continues the task execution after a *Send()*. On the other hand, the *Receive()* primitive is blocking, i.e. the task which executed the primitive is preempted if the message is not available, and scheduled again only after the message is available in the PE. To implement this mechanism, each microkernel contains a vector, named *pipe*, which contains the messages to exchange between the tasks. Additionally, services were created to favor task communication and control application execution. Table 5 describes the available services. The service code is the first field of the NoC packet payload delivered to the PE, notifying the microkernel about the action to be taken.

Table 5 – Available services in HeMPS MPSoC.

Service	Hexadecimal Code	Description
REQUEST_MESSAGE	0x00000010	Request of a message.
DELIVER_MESSAGE	0x00000020	Delivery of a message previously requested.
NO_MESSAGE	0x00000030	Information that requested message does not exist.
TASK_ALLOCATION	0x00000040	A task to be allocated in the PE.
ALLOCATED_TASK	0x00000050	Information that a given task was allocated in a remote processor.
REQUEST_TASK	0x00000060	Request of allocation of a given task.
TERMINATED_TASK	0x00000070	Information that a given task finished its execution.
DEALLOCATED_TASK	0x00000080	Information that a given task finished its execution and can be deallocated.
FINISHED_ALLOCATION	0x00000090	Information that master PE finished the initial task allocation.
DEBUG_MESSAGE	0x00000100	Message containing debug information.

3.2 HEMPS ARCHITECTURE WITH SUPPORT FOR DFS

Figure 10 presents the proposed architecture to enable DFS in HeMPS. The first difference between the proposed architecture and the reviewed works (Chapter 2) is that the DFS is applied to both PE and routers. Also, as it is shown, the controlling scheme is individualized, as is the local clock generation, inserted into each PE and each router. The second difference is the set of monitoring parameters used to control the proposed DFS scheme, where, as explained in Chapters 4 and 5, is better suited to distributed system.

The performed adaptations are highlighted in Figure 10. The first modification was the replacement of existing router input FIFOs by bisynchronous FIFOs (GALS FIFO), depicted in Figure 10(a) as red rectangles. A synchronization scheme is needed to avoid metastability [RAB03], since modules working at different frequencies and in different clock domains will communicate with each other. A FIFO was chosen as synchronization scheme due to the high throughput and low latency provided by it [CHE00] [PAN07].

Figure 10(b) and Figure 10(c) depict the different DFS controller types. Figure 10(b) represents the DFS controller of the PE and the clock generation module, which is described in Chapter 4. Figure 10(c) represents the router controller, described in Chapter 5. Figure 10(d) displays the replacement of the NI input FIFO by a bisynchronous FIFO. Also, the NI was modified to make available the operating frequency of a remote processor to the PE controller using information extracted from the NoC packet. Moreover, several memory-mapped registers were included in the PE, although this is not shown in Figure 10. Besides hardware modifications, the microkernel was modified to monitor CPU utilization and *pipe* occupation, storing these values into new memory-mapped registers. These modifications will be described in Section 4.2.

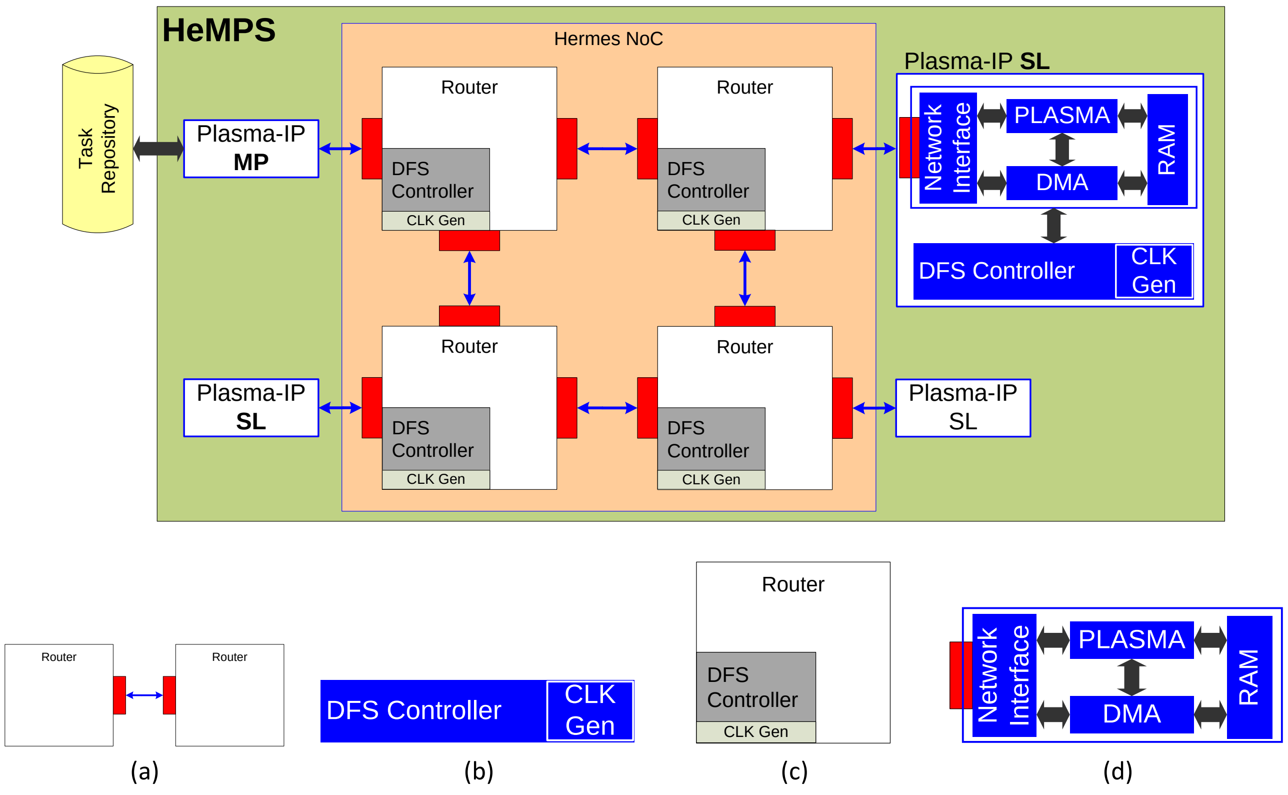


Figure 10 – Proposed HeMPS Architecture for DFS.

It is important to highlight that each individual clock generation module receives the system clock as reference, generating a new clock from it. The benefit of such approach is that the global clock has to feed only the clock generation modules, significantly reducing the global clock load, and hence simplifying the clock tree generation and its power consumption, which is responsible for at least 40% of the power consumption in present MPSoCs. The global system clock is called in the following Chapters *Reference Frequency*.

4 DFS AT THE PROCESSOR LEVEL

This Chapter presents the proposed DFS scheme for the HeMPS PE, which is the *first contribution* of this work. The DFS controller computes the communication load and CPU utilization level according to values provided by the microkernel. The controller uses such values to define the frequency of each PE. The controller always operates at the reference frequency, which is the highest frequency in the system, and provides the computed frequency to the PE. The provided frequency is generated by a clock generation module, detailed in Section 4.1. Section 4.2 presents the controlling scheme, responsible for computing the PE frequency. Finally, Section 4.3 explains the proposed policy to adapt the controlling scheme for multitask execution.

4.1 CLOCK GENERATION

The clock generation module is the same for both PE and router DFS controllers. This module is responsible for generating the local clock, using the reference clock as input. The objective of designing this module is not to propose a new clock generation technique, but to provide a mechanism in hardware to enable the proposed DFS scheme and evaluate its impact on the results. The principle of the clock generation process is to achieve clock division by simply omitting selected cycles of the reference clock, as Figure 11 illustrates: for example, inputs *num_i* and *den_i* are natural numbers 2 and 5, respectively. This corresponds to setting the frequency of the clock generator to two-fifths (40%) of the reference clock. In other words, for each *den_i* reference clock cycles, *num_i* cycles are propagated to the output clock.

Frequencies are obtained by changing *num_i* and *den_i* values, with some defined exceptions (*den_i*=0 is not an acceptable value, *num_i*=0 corresponds to a clock gating action and the constraint *num_i* ≤ *den_i* must be respected). Before changing the *num_i* and *den_i* values, the *restart_i* signal must be asserted to momentarily stop the output clock and reinitialize internal registers. After releasing *restart_i*, the new frequency, defined by the modified values of *num_i* and *den_i* appears at the output.

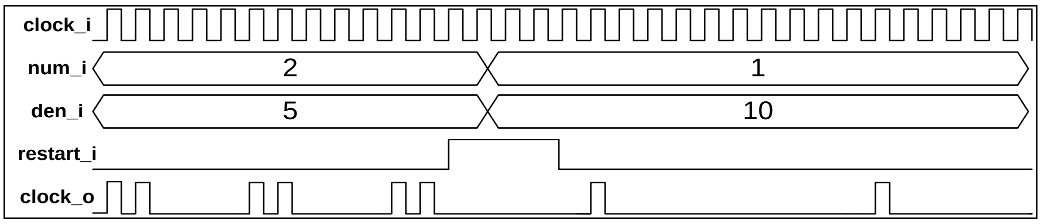


Figure 11 – Example of the proposed clock generation process. Signal clock_i is the reference clock and clock_o is the output of the clock generator.

This clock generation scheme translates into a simple logic that can multiply the reference clock frequency by selected values distributed inside the closed interval [0,1]. The amount of distinct values is dictated by the range of values assignable to *num_i* and *den_i*, which defines the size of internal registers used to store them. This clock generation scheme can be classified as ratiochronous [CHA09], since any relation between two frequencies used in the system is a rational number. In the scheme, if all clock edges of all clocks can always be kept in phase with the corresponding edge of the reference clock, the system can simply dispense with the use of synchronizers. Although this would clearly improve performance of the system, since synchronization overhead would be eliminated, this may have a strong impact on clock distribution control, and is ignored here to keep clock generators very simple.

Figure 12 presents the block diagram of the clock generation module. The module has only one output (*clock_o*), which is the clock used by PEs or routers. The inputs *num_i* and *den_i* are used to configure the clock generator, and the input *restart_i* is used to reset the counters *count_num* and *count_den*. As in the discussed example, *num_i* represents the numerator of the fraction that multiplies the reference clock, while *den_i* is the denominator of the fraction. A signal called *enable* is generated to stop or release the output clock. For the sake of simplicity, a simple *AND* cell is used to gate the clock in this work. Still, a clock gating cell (GCLK in Figure 12) can be used to avoid the need to design a special cell, since several standard cell libraries include clock gating cells.

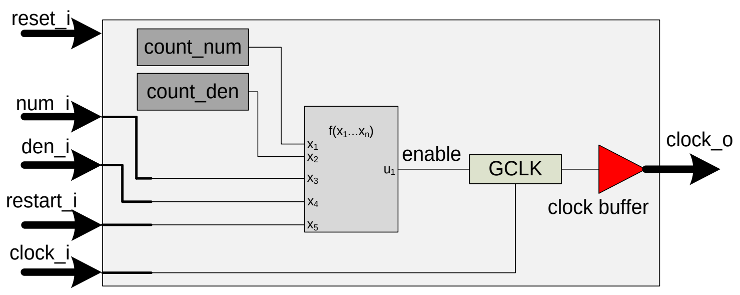


Figure 12 – Clock generation module.

The main advantages of this clock generation module are the low area overhead and a large set of generated frequencies. For example, for *num_i* and *den_i* being 4-bit values the module takes 107 cells for a 65nm standard cell library from ST Microelectronics. In this same example, 120 different fractions can be obtained. Although several of these correspond to a same frequency (e.g. 1/1, 2/2, etc) still a large number of distinct frequencies can be produced with only 4 bits for *num_i* and *den_i* (71 different frequencies in this example). In addition, the clock output is always stable, contrary to what happens in standard DFS methods, where the time required to stabilize a new frequency can be relevant. The proposed module is also *glitch free* by construction.

The main drawbacks of the approach are how to couple it with voltage scaling and the need to design the critical paths of all modules in the system to support the reference clock frequency. As Figure 11 exemplifies, for any generated frequency, the duration of one clock period will be the same as that of the reference frequency clock period, making voltage scaling prohibitive. Nevertheless, the clock generation module can be modified to generate frequencies with 50% of duty cycle, and extended to cope with voltage scaling by using a power supply control, as presented e.g. in [MEI05]. Also, for DFT purposes, it is necessary to make the output clock observable and controllable. This can be achieved by including signals such as scan input and scan enable in the module. However, as mentioned, the clock generation module was designed only to enable DFS in HeMPS, and was kept simple for the lowest possible area overhead.

4.2 PROCESSING ELEMENT DFS CONTROLLING SCHEME

Figure 13 presents the architecture proposed for a HeMPS PE with DFS. Initially, the NoC-processor interface needs to be modified to work according to the GALS paradigm. This is achieved by adapting the existing local buffer in the NoC and network interface to work as a bisynchronous FIFO, and introducing two-flop synchronizers into control signals. The NoC-processor interface contains two buffers (GALS FIFOs), one in the router, to receive data from the NI, and another in the NI to receive data from the NoC. In this work, it is proposed a DFS scheme only for slave PEs. The Master-PE always works at the reference frequency to avoid creating a bottleneck in the system. Still, a scheme using a specific controlling metric for the Master-PE can be implemented as future work.

The metrics chosen for the DFS scheme are the PE communication load and the CPU utilization. Contrary to most of the works presented in Chapter 2, monitoring only the communication load or only the CPU utilization does not work for the targeted MPSoC. Monitoring only the communication load might present problems when the *pipe* has few or no messages stored. According to the works that used this metric, the frequency should be increased in this case. However, if the task is blocked, e.g. waiting for a message from other task, and the *pipe* has low occupation the frequency can be kept or even decreased (scenario 4 in Table 6). Similarly, monitoring only the CPU utilization might cause problems when it is near 100%. According to the works that used this metric, the frequency should be increased in this case. However, with high CPU utilization and high *pipe* occupation (scenario 1 in Table 6) the frequency does not need to be increased, since the messages are being produced at a higher rate than they are being consumed. The same reasoning can be used for the other 2 scenarios in Table 6. In scenario 2, high pipe utilization and low CPU utilization means that, even at low processing rate, the message producing rate is higher than the message consuming rate, leading to decrease the operating frequency. In scenario 3, two actions can be taken: (*i*) increase frequency if the processor is near of 100% utilization to avoid *pipe* reaches empty state and (*ii*) keep the current frequency if the processor is not near of 100% utilization, i.e. the processor still not fully utilized at the current frequency. Therefore, monitoring these two parameters is mandatory for efficient control.

Table 6 – Possible monitored parameter scenarios and actions.

		CPU Utilization	
		High	**Low**
Communication Load	**High**	Decrease Frequency **(1)**	Decrease Frequency **(2)**
	Low	Increase/Keep Frequency **(3)**	Decrease/Keep Frequency **(4)**

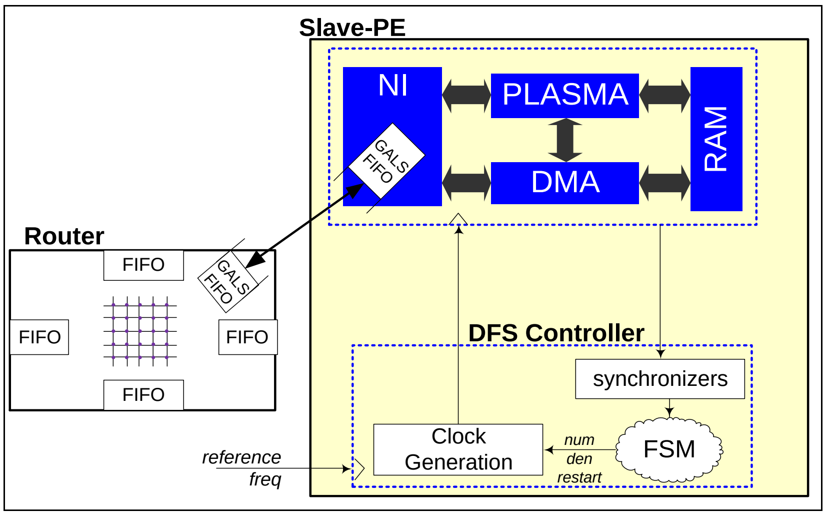


Figure 13 – Router-PE GALS interface and the PE DFS controller.

Figure 14 shows the DFS controller interface. It uses the following input signals:

- *pipe_ocup* and *req_msg* related to the communication load, represent the number of messages stored in the *pipe* (an integer value), and a notification of a request for a message not yet produced by the processor (a Boolean value).
- *not_scheduled* (Boolean value): when true, only the microkernel is running, meaning that no task is being executed; when false, at least one task is being executed. The DFS controller may define CPU utilization by counting how many clock cycles this signal is asserted in a sampling period.
- *msg_transfer* (Boolean value), *dma_active* (Boolean value), *rem_freq* (two integer values): set of signals used to increase the processor frequency, if necessary, when transmitting a data packet.

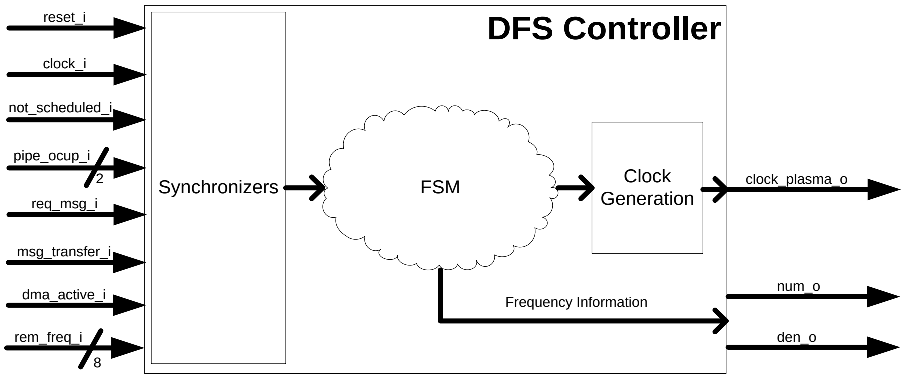


Figure 14 – DFS Controller Interface.

As the DFS controller works at the reference frequency and the processor at a different frequency, a synchronization scheme between them is necessary. This synchronization is carried out in the block *Synchronizers* (Figure 14). The controller uses the Clock Generation module, detailed in Section 4.1, to provide the processor frequency. The FSM represented in Figure 14 corresponds to the behavior detailed in Table 7. The role of the FSM is to choose the PE frequency based on the communication load and CPU utilization. The frequency is chosen by evaluating the following information:

- **Pending message requests from other tasks**. This situation takes place when the processor is not producing data to the consumer task (*req_msg* = 1). This information is related to the communication load.
- ***Pipe* occupation.** If the *pipe* has a high occupation, the processor is certainly producing messages at a higher rate than the consumer tasks can consume, while the inverse scenario means a lack of produced messages. Upper and lower parameterizable thresholds, determined by the designer, define the high and low occupation states, respectively. Occupation between these values defines an operational state. This information is related to the communication load.
- **CPU utilization.** When the utilization is low, the CPU is not executing any task (*not_scheduled*= 1) or all tasks are blocked, e.g., waiting message(s) from other tasks. When the utilization is high, tasks are using the processor at the maximum rate. Two parameterizable thresholds, determined by the designer, define high, low and operational CPU utilization states.

The first monitoring parameter evaluated is the presence of pending message requests. Next, the controller evaluates the *pipe* occupation and CPU utilization, respectively. As Table 7 denotes, the controller is guided mainly by the communication load, leading the tasks to produce and consume data at a uniform rate, with no penalties for the overall application performance. The CPU utilization parameter prevents the controller from increasing the frequency in situations where the communication load could normally denote an increase in the frequency, e.g. actions 3 and 6 of Table 7. Also, the parameterizable thresholds of CPU utilization define the reactivity of the controller, i.e. depending on the values, the controller reacts faster or slower.

Table 7 – DFS Controller behavior (↓/↑ mean decrease/increase one frequency step, ↑↑ means increase two frequency steps, = means keep frequency unchanged and - denotes don't care conditions).

Action in frequency	*Pending Message*	*Current Pipe Occupation*	*Previous Pipe Occupation*	*CPU Utilization*
1 - ↓	0	high	-	-
2 - ↓	0	operational	low	-
3 - ↓	0	low	-	low
4 - =	0	operational	operational	-
5 - =	0	low	-	operational
6 - =	1	-	-	low
7 - ↑↑	1	-	-	operat./high
8 - ↑	0	low	-	high
9 - ↑	0	operational	high	-

Frequency decreases in three situations: (i) the *pipe* is almost full (action 1 of Table 7); (ii) the *pipe* occupation is increasing, i.e. in the previous evaluation its state was low and the present state is operational (action 2). In situations (i) and (ii) the processor is producing messages at higher rate than they are being consumed; (iii) the *pipe* occupation is almost empty and the CPU usage is low, meaning that even at a lower frequency the data in the *pipe* is being consumed (action 3).

Frequency increases in three situations: (i) existence of pending messages with operational or high CPU utilization (action 7) – the clock generator increases the frequency in two steps; (ii) the *pipe* is almost empty and the CPU has high utilization (action 8); (iii) the *pipe* occupation is dropping, e.g. in the previous evaluation its state was high and the present state it is operational (action 9). In these three situations, the frequency is increased to avoid the remote task wait for long periods to receive the requested message, penalizing the application performance. Still, in situation (i), when the message was already requested and not produced yet, the increasing is even higher to produce the requested message as soon as possible.

The period between consecutive evaluations is also parameterizable by the designer. In this work, the evaluation period corresponds to four time slices. A time slice is a fixed time interval, defined in the microkernel, in which the system executes a given task. Other values for evaluation period were evaluated, but higher values (e.g. 8 and 16) lead to slow responses by the controller since less evaluations will occur in the same period of time, while smaller values (e.g. 1 and 2) may turn the controller unstable, i.e. the frequency changes too much, even when is not needed.

When an evaluation is triggered, the controller stores the values generated by the microkernel (current *pipe* occupation and CPU utilization) and evaluates the monitoring parameters. Similarly to [SHA09], the values of *pipe* occupation and CPU utilization used by the controller are an average of the current and previous values. By using an average value, the controller changes the frequency smoothly in the presence of high variations in the *pipe* occupation and CPU utilization.

Also, the DFS controller implements a communication mechanism to balance power consumption and performance. It takes into account that a consumer processor must receive the data in a frequency that is not inferior to its operating frequency. Thus, if the producer processor is operating at higher frequency than the consumer processor, the message can be sent at a lower frequency to save power. On the other hand, if the producer is operating at a lower frequency than the consumer, its frequency should be temporally increased to avoid penalizing the consumer performance. The set of signals {*msg_transfer*, *dma_active*, *rem_freq*} are responsible for accomplishing this action.

The reception of a packet with REQUEST_MESSAGE service asserts the signal *msg_transfer*. This packet contains, among other fields, the frequency of the PE requesting data. This frequency is coded in the signal *rem_freq* (remote frequency). According to this, the following situations may occur:

- If *rem_freq* is lower than the PE frequency, the PE frequency does not need to be changed. The packet is sent through the NoC with the *rem_freq* coded in it.
- If *rem_freq* is higher than the PE frequency and there is data in the *pipe*, the PE frequency is set to the *rem_freq* during the transmission of the DELIVER_MESSAGE packet. The PE frequency returns to the original frequency monitoring the *dma_active* signal. When this signal returns to zero, it means the DELIVER_MESSAGE transmission has finished.
- If *rem_freq* is higher than the PE frequency and there is no data in the *pipe*, the rules of the FSM are used. In this case, it is highly probable that the FSM will apply rule 7, increasing by

two steps the processor frequency, to produce data faster to the consumer.

All other control packets (as REQUEST_MESSAGE, REQUEST_TASK, TASK_ALLOCATION, etc) are injected into the NoC at the maximum frequency, making such packets available to the target processor as soon as possible.

4.3 MULTITASK SUPPORT

One of the main characteristics of the target MPSoC is the support to multitask execution. The number of tasks that a PE can execute is defined by the available amount of RAM memory and by the page size. Although each task has an individual memory address space, the communication *pipe* is located in the *microkernel* page, being shared by all tasks under execution in the PE. Also, the *microkernel* scheduler uses the *Round Robin* algorithm, evenly sharing CPU time among all tasks. However, sharing the resources (CPU and *pipe*) used to generate the DFS monitoring parameters may lead to incorrect analysis of the information required to adjust the frequency. Therefore, it is necessary a mechanism to ensure the correct management of resources and at the same time leading the DFS controller to act properly.

In [GOO10], the proposed algorithm saves the task context when the task is preempted. Later, when the task is scheduled again, the frequency in which it was operating before being preempted is retrieved and used. Thus, for each task executing in the processor, the controller may change the operating frequency. However, this approach may present problems when several tasks with different characteristics are being executed in the same processor. In this case, at each time slice, the processor may have its frequency drastically changed, e.g. from the highest to the lowest available frequency. Moreover, if voltage scaling is used, the time to perform such switching requires high stabilization periods, which becomes prohibitive in some systems. Thus, the controlling scheme must adjust the processor to operate in a frequency which satisfies the requirements of all tasks being executed, switching the frequency the less often the possible.

Using the proposed DFS approach in a multitask environment with no adaptations may present undesirable results, depending in the characteristics of the tasks being executed. Figure 15 illustrates a scenario that leads to undesirable behavior of the original approach. The horizontal bar represents the *pipe*, the vertical bar inside the CPU block represents the CPU utilization and the circles represent tasks A and B. Initially, the *pipe* is empty and the CPU is idle (Figure 15(a)). Next, Task A is scheduled. It is assumed this task presents low computation workload and high communication workload. Thus, the CPU utilization barely increases, while the *pipe* becomes almost full (Figure 15(b)). Later, Task A is preempted and Task B is scheduled. It is assumed Task B presents high computation workload and low communication workload. Accordingly, the CPU utilization increases to the maximum, and the *pipe* remains with the same occupation (Figure 15(c)). Then, Task A is rescheduled, producing new messages and filling the *pipe* (Figure 15(d)). After, Task B is rescheduled, producing its first message. However, the *pipe* is full, and Task B writing is blocked (Figure 15(e)).

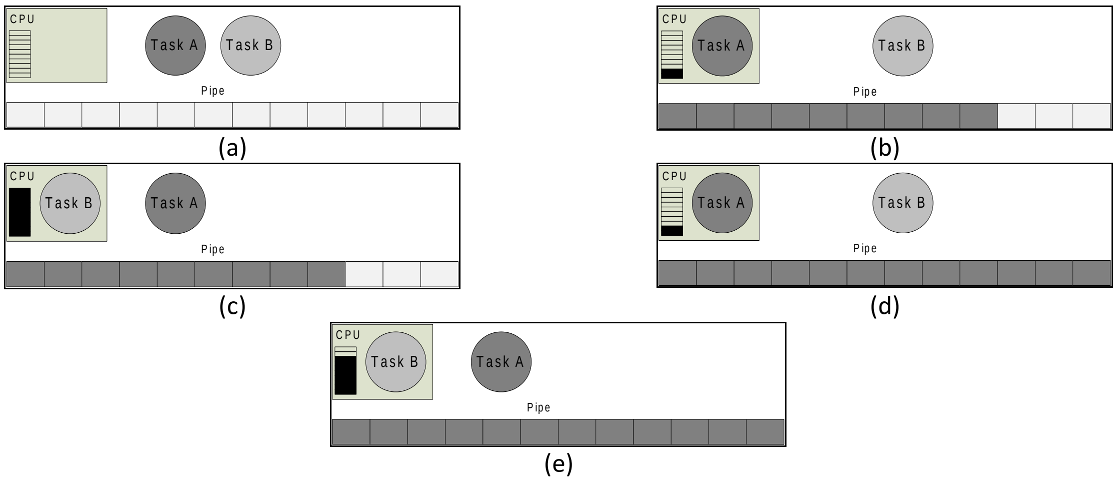


Figure 15 – Resource sharing problem in multitask execution.

Considering this scenario, the proposed scheme should decrease the operating frequency when the *pipe* becomes almost full, as in Figure 15(b), and decrease one more step when it becomes full, as in Figure 15(e). However, the frequency should be kept or increased, due to the high computation workload of Task B. Thus, if the frequency is decreased, the application that contains Task B will present a severe performance penalty. Therefore, besides monitoring *pipe* occupation, it is necessary to control the amount of resources used by each task.

Initially, a monitoring scheme was implemented in hardware to inform the controller the *pipe* occupation of the running task. However, the set of rules needed to implement the multitask policy in hardware could increase the controller complexity significantly. Thus, a simple solution in software was adopted to overcome this difficulty. Since the cause of the problem is the sharing in resources used by the DFS controller, it was decided to limit the amount of resources that a given task can use. Regarding CPU utilization, no modifications are needed, since the *Round Robin* provides the same amount of time for each task to execute, and the tasks are preempted in case of blocking. On the other hand, the *pipe* utilization needs to be controlled. To avoid the use of the entire *pipe* by a single task, the *microkernel* scheduler was modified to also take into account the *pipe* occupation of the task to be scheduled. Thus, the task is scheduled only if it is using a given number or fewer positions in the pipe. Figure 16 shows the scheduler pseudocode, which handles the task scheduling.

```
if (current->status == READY && ( (allocated_tasks > 1 && current->pipe_util < 8) || allocated_tasks <= 1 ) ) {
    //ready to execute
    current->status = RUNNING;
    scheduled = TRUE;

    //enable timeslice counter
    OS_InterruptMaskSet(IRQ_COUNTER18);

    //restart timeslice for the task
    MemoryWrite(COUNTER_REG, 0);

    MemoryWrite(NOT_SCHEDULED, 0);

    break;
}
```

Figure 16 – *Microkernel* scheduler for multitask execution with DFS.

Nevertheless, the new conditions are used only when there is more than one task running in the processor, i.e. in mono task execution the task can use the entire *pipe* if needed. With this modification, the DFS controller is able to adjust the PE frequency also in multitasking scenarios.

5 DFS AT THE NOC LEVEL

This Chapter presents the proposed architecture and controlling scheme for DFS at the NoC level, representing the *second* contribution of this work. First, a brief discussion about the router architecture is carried out in Section 5.1. Section 5.2 describes the modifications performed in the Hermes router to enable DFS and its controlling mechanism. Finally, Section 5.3 presents the calibration phase of the method proposed by Guindani et. al. [GUI08] in the DFS-enabled router.

5.1 ARCHITECTURAL EXPLORATION

Designing a router that may work at multiple frequencies and communicate with neighbor routers working at different frequencies requires adaptations on all input/output buffers. The same way it was done in Section 4.2, all input buffers (*east, west, north* and *south*) of the router have to be replaced by bisynchronous FIFOs to synchronize router communication. In addition, it is necessary to modify the router arbiter, which is responsible for requesting frequency changes.

Two strategies were evaluated for the router architecture with DFS: distributed and centralized. Figure 17 illustrates each strategy, where continuous lines represent the originally existing connections, and dashed lines represent connections created to support DFS.

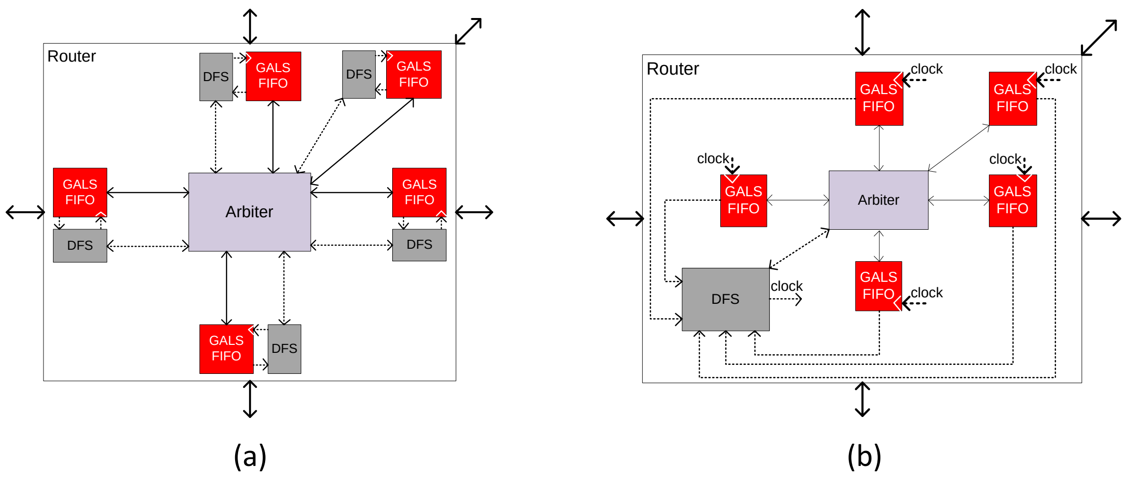


Figure 17 – Distributed and centralized strategies for Hermes router architecture with DFS.

In the distributed strategy (Figure 17(a)) it is necessary to control each input buffer independently. This strategy is more flexible, enabling different packets to be sent at different frequencies. However, it requires one frequency controller and one clock generation module per buffer, increasing area and power. Thus, the decision to use this strategy needs to be based in the tradeoff between power overhead and power savings. On the other hand, the centralized strategy (Figure 17(b)) induces lower power and area overhead, while decreasing flexibility. In this strategy, the optimal energy saving cannot be achieved, due to the need of using the highest requested frequency when two or more packets are being switched simultaneously. Nevertheless, if only one packet is being switched, the router will operate at some frequency that is not the minimum, leading the buffers that are not being used to operate at this frequency as well. In both strategies it is assumed that when there is no activity in the router, the controller sets the frequency to the minimum possible value.

To evaluate both strategies, an early analysis was performed with two traffic scenarios using the data collected in Section 5.3 (Figure 18). The NoC configuration is the same used in HeMPS, with 16-bit *flit* size and 16-position buffers. Router utilization is calculated by dividing the time that the router is active, by the total time needed to switch all flows. In the first scenario (Figure 18(a)) four traffics were injected through a router with 5 input ports in parallel. Each traffic was injected at a different frequency (400MHz, 240MHz, 160MHz and 100MHz). In this scenario, during the time that the router is active, it operates at 60% of the time at 400MHz, 25% at 240MHz, 8% at 160MHz and 7% at 100MHz. In the second scenario (Figure 18(b)) two traffics were injected in parallel, being one traffic injected at 240MHz and the other at 100MHz. In this scenario, during the activity time, the router operates 35% at 240MHz and 65% at 100MHz.

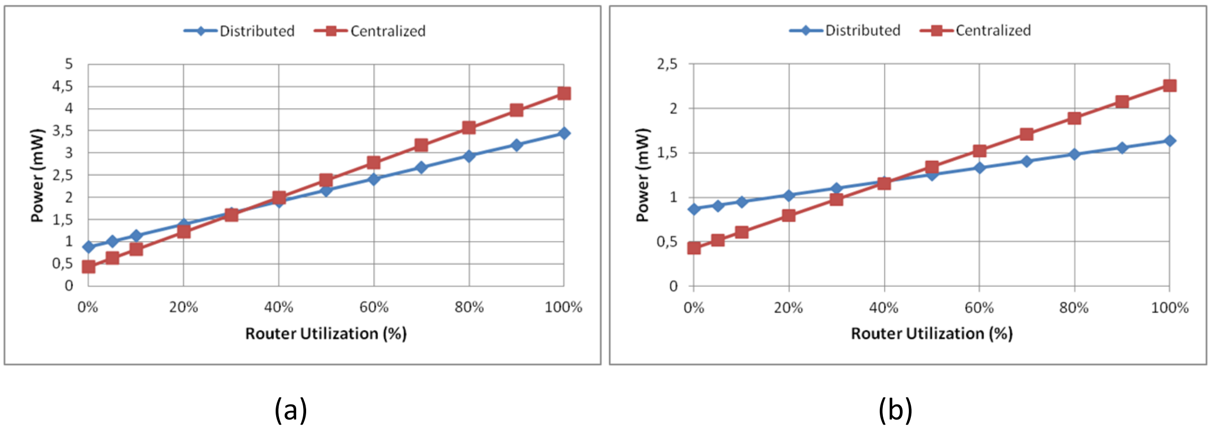


Figure 18 – Comparisons between centralized and distributed DFS strategies. (a) Four traffics at high frequencies injected in the router; (b) Two traffics at low frequencies injected in the router.

According to Figure 18, the distributed strategy may present better energy savings in scenarios that utilize large NoC bandwidth. Considering a scenario that presents several traffics passing through a router, with different frequencies and at a high injection rate (Figure 18(a)), e.g. hotspot scenario, the DFS controller sets the frequency of its respective buffer to the frequency required by the packets. Nevertheless, due to the high injection rate, each buffer will operate at the configured frequency most of the time. Thus, the power overhead induced by the controllers in this strategy may become lower than the power overhead in a centralized strategy, which would set the frequency of all buffers to the maximum required frequency.

However, scenarios presenting low bandwidth utilization and few packets being routed (Figure 18(b)) make the distributed strategy less attractive than the centralized. As Figure 18(b) shows, router utilization in the second scenario must be greater than in the first scenario (Figure 18(a)) to justify the use of a distributed strategy. In these scenarios, the mechanism that verifies te inactivity in the router would make it operate at the minimum possible frequency most of the time. As the traffic presents low bandwidth utilization, the router will operate at higher frequencies only for short periods, quickly returning to the lowest frequency. Therefore, in these scenarios, it is more attractive to use a centralized strategy, which induces lower area and power overheads than a decentralized one.

Notably, most distributed applications running at MPSoCs are not able to generate traffics that may exceed 20% of the maximum bandwidth [KAO11]. Thus, it is not realistic to say that a scenario encouraging the use of the distributed strategy may occur in a substantial number of applications. Moreover, a large number of applications can be expressed as a data pipeline, e.g. video and audio applications [ALI06]. Tasks of this kind are mostly CPU-intensive, characterizing a scenario with low bandwidth utilization, since the sending of two consecutive packets is separated typically by a large period of time. Thus, this work chooses to use the centralized strategy in the router architecture with DFS support.

5.2 ROUTER DFS CONTROLLING SCHEME

The router DFS controller (Figure 19) is responsible for defining the router frequency and putting it in a low power mode in case of inactivity. It receives the frequency and activity information from each input buffer and the control signals from the arbiter (*clk_change, activity, header, selection, ack_header*), to set the required operating frequency of the router (*clk_r*).

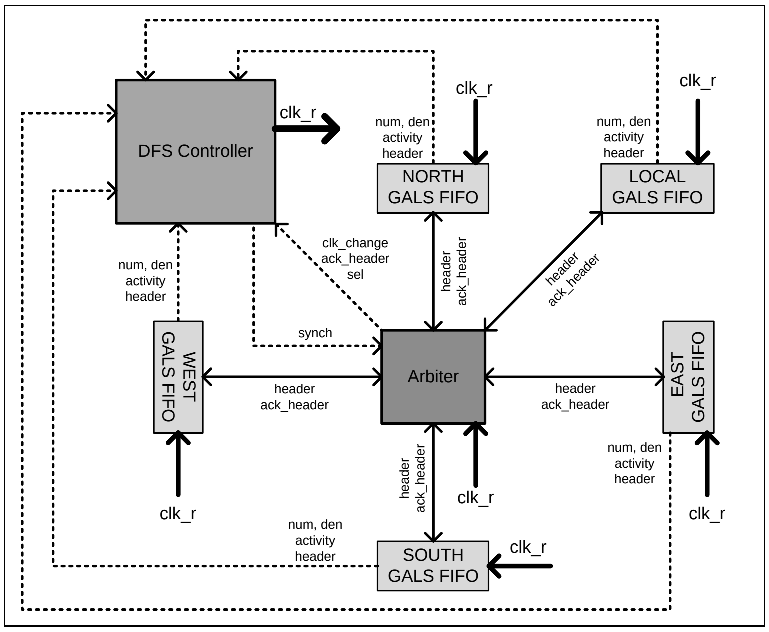


Figure 19 – Router DFS controller structure.

Frequency values are obtained from the packets sent through the NoC, which carry in the header field the frequency value that it needs to be transmitted (Figure 20). This value is an 8-bit array, 4-bits for numerator and 4-bits for denominator, of the required frequency level. This field was designed to cope with the clock generation module presented in Section 4.1.

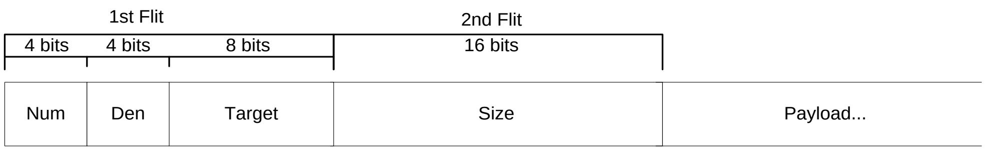


Figure 20 – Hermes packet with frequency information.

When the packet is received, the FSM in the input buffer extracts the frequency information and provides it to the DFS controller (Figure 21(a)). Next, the buffer sends a routing request to the arbiter and starts waiting for the arbiter acknowledgement (Figure 21(b)). When the arbiter receives the routing request, it sends a frequency switching request to the DFS Controller (*clk_change_i*), and starts waiting the clock synchronization signal from the DFS Controller (*synch_o*) (Figure 21(c)). Then, using the information provided by all buffers and the frequency switching request, the DFS Controller checks from which port the routing request was made, and stops the provided clock (Figure 21(d)).

The decision of changing the frequency is made by comparing the frequency required by the requesting input buffer with the current router frequency, i.e. when two or more flows are passing through the router, the controller will always select the highest frequency, avoiding loss of performance in the network. If the required frequency is lower than the current frequency, the controller only informs the arbiter that the clock is synchronized, without changing the frequency. If the required frequency is higher than the current frequency, the controller replaces the values of numerator and denominator of the clock generation module and, after releasing the *restart* signal of clock generation module, informs the arbiter that the clock is synchronized (Figure 21(e)). Lastly, the arbiter sends the acknowledgment to the input buffer and routes the packet (Figure 21(f)).

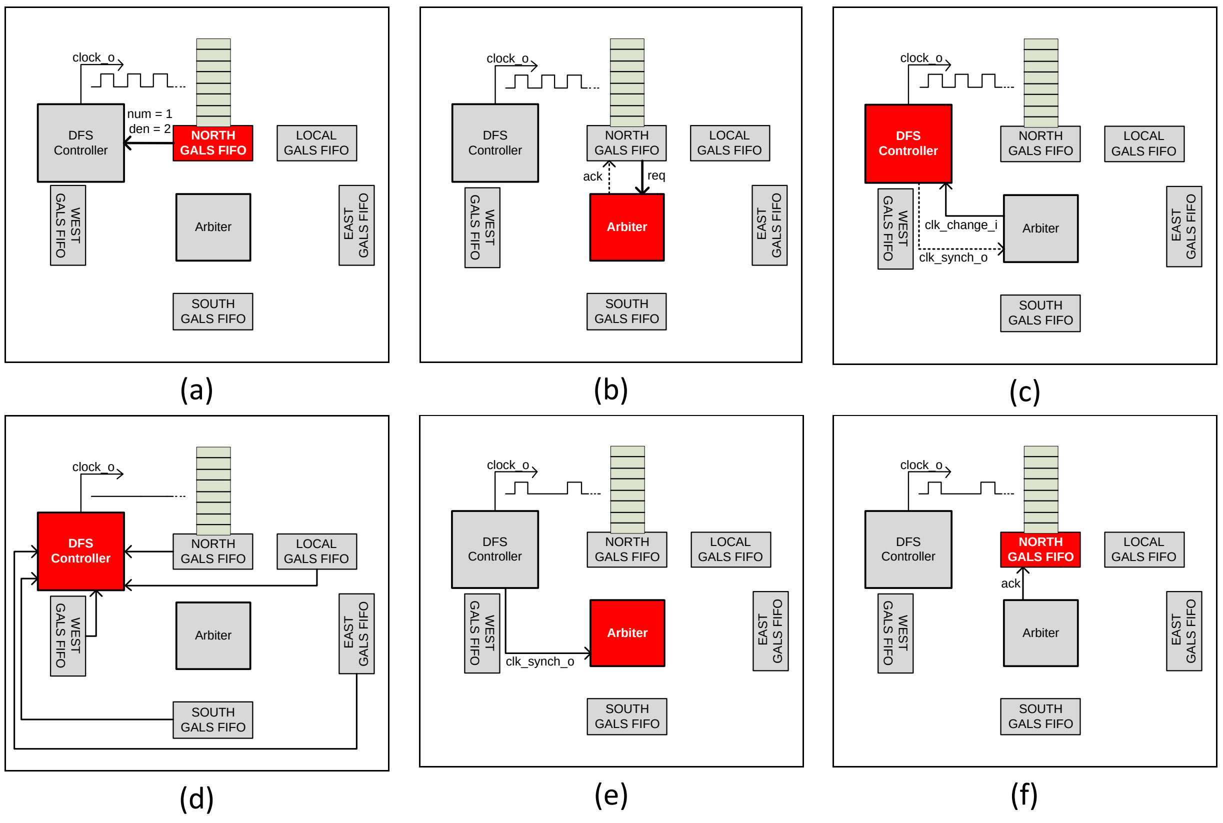


Figure 21 – Proposed frequency switching mechanism for the Hermes router.

Additionally, when no traffic is being switched through the router, it must decrease its frequency to the minimum available value. To implement this mechanism, it is necessary to monitor router activity. This is achieved by monitoring the buffer occupation and packet receptions. The inactivity comes from empty buffers and no packet reception in all router ports. Also, after finishing a packet transmission, the buffer resets the values of numerator and denominator to the minimum available frequency. This must be done to avoid the router from stalling its operating frequency after finishing the transmission of high frequency packets.

5.3 Calibration

Since commercial tools may take several hours to estimate the power dissipation of complex systems, Guindani et. al. [GUI08] propose a higher abstraction level method to estimate power dissipation which presents small errors compared to power estimation commercial tools while providing results fast. This method is composed by two main phases: calibration and evaluation. In the calibration phase, a flow with controlled rate is injected in the router, and the power evaluation is performed by a commercial tool. This process is repeated for several injection rates, and an equation relating power and injection rate is obtained. Each module of the router (buffer, arbiter and crossbar) has an equation. Later, in the evaluation phase, the NoC is monitored and the injection rate of each router is extracted. Finally, these injection rates are used in the equations obtained in the calibration phase to calculate the total power dissipation of the simulated scenario.

Using the method proposed in [GUI08], the router with the proposed DFS controlling scheme was calibrated to provide information for fast power evaluation. However, the power dissipation is directly related to the operating frequency. Thus, only a single equation for each module of the router is not adequate to characterize the proposed architecture in terms of power. Since the router handles two different clocks (transmission clock and reception clock), there are n^2 different operating configurations, where *n* represents the number of available frequencies. Figure 22, Figure 23 and Figure 24 show the three possible cases that may occur during writing and reading actions. All three cases adopt as reference the writing and reading frequencies in the West (W) buffer.

Figure 22 shows a scenario where the writing clock of the buffer will be the same as the reading clock, since the flow being injected in the West buffer has higher frequency than the flow injected in East (E) buffer.

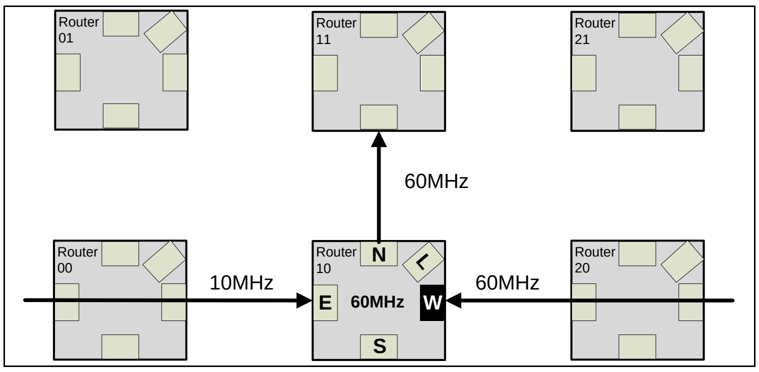


Figure 22 – Calibration scenario with same reading and writing clocks.

Figure 23 presents a scenario where the reading clock has a higher frequency than the writing clock. This situation arises when there is a traffic with higher injection rate being switched in the same router. In this case, the traffic being injected at the West buffer has lower frequency than the traffic being injected at the East buffer. In this scenario, the power dissipation of East and West buffers is different, since the East buffer is being written and read at 100MHz and the West buffer is being written at 60MHz and read at 100MHz.

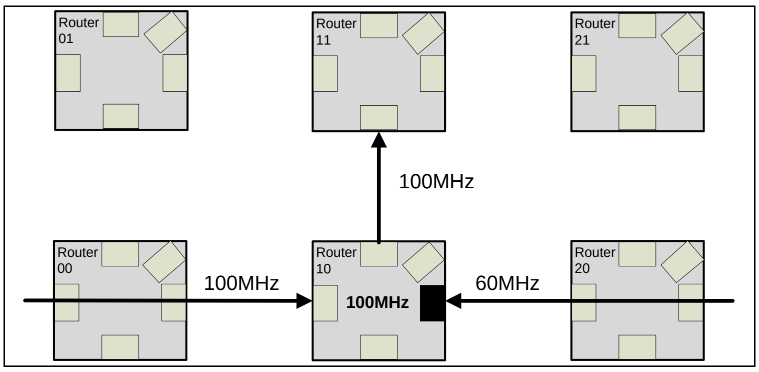


Figure 23 – Calibration scenario with reading clock higher than writing clock.

Figure 24 presents the last scenario. This is a special case, where the reading clock has lower frequency than the writing clock. This may occur due to neighbor routers working at higher frequency, sending their transmission clocks even when no traffic is being sent. However, this situation only occurs when the buffer is not receiving any data, i.e. injection rate equal to 0. Therefore, in cases where the writing clock has higher frequency than the reading clock the power dissipation is fixed.

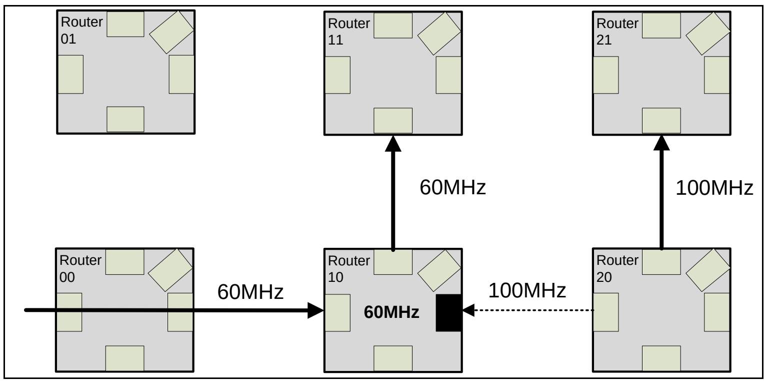


Figure 24 – Calibration scenario with writing clock higher than reading clock.

The calibration process was performed for nine frequency levels, which will be detailed in Section 6.1. For each router sub-block (buffer, switch control, crossbar and DFS controller), a matrix of size 9x9 was generated to store the equation coefficients according to the possible frequency levels. The matrices are presented in Appendix A.

6 EXPERIMENTAL RESULTS

This Chapter describes the experimental setup (Section 6.1) and the results obtained for monotask (Section 6.2) and multitask execution (Section 6.3) in the reference MPSoC, with the proposed DFS scheme in both PE and NoC. Sections 6.2 and 6.3 are divided into other sections containing the performed evaluations for synthetic and real applications. Each application used a different platform configuration, which will be detailed within its section. The gains obtained in PEs are evaluated in terms of the reduction in the number of executed instructions and power consumption. The gains in the NoC are obtained by measuring its power consumption. Also, it is compared, for each application, the power consumption relation between the PE and the NoC router.

6.1 EXPERIMENTAL SETUP

A set of six applications was evaluated in the proposed architecture: (*i*) Communication; (*ii*) Pipeline with 6 tasks – pipeline_6; (*iii*) MPEG partial filter; (*iv*) JPEG decoding; (*v*) matrix multiplication; (*vi*) Video object plane decoder (VOPD). Applications (*i*) and (*ii*) are synthetic, while the applications (*iii*), (*iv*), (*v*) and (*vi*) are real.

Figure 25 shows the task graph of application (*i*). The application has two tasks which only produce data, being called *producers* (tasks A and B), one task which consumes the data produced by the *producers* and provides it to another task, being called *worker* (task C) and a task which only consumes data, being called *consumer* (task D). Each task executes a configurable loop to emulate a given processing load. Three scenarios were evaluated with different emulated processing loads for the tasks in each one of them.

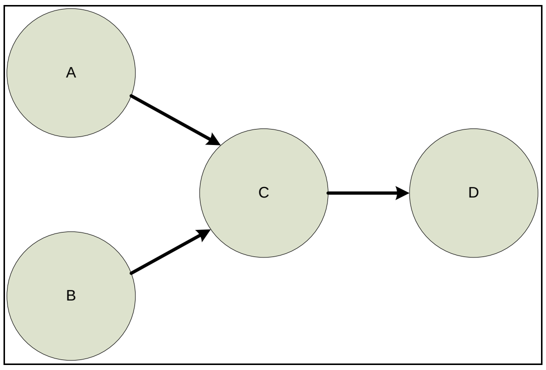


Figure 25 – Communication task graph.

Figure 26 depicts the application (*ii*) – pipeline_6. Task *A* is the only *producer*, task *F* is the only *consumer*, and the other tasks (*B, C, D* and *E*) are *workers*. Similarly to application (*i*), each task executes a configurable loop to emulate a processing workload. Two scenarios were evaluated for mono-task execution: scenario 1 configures all tasks with the same workload; scenario 2 configures the tasks according to three different workloads. For multi-task execution each scenario was evaluated using two different task mappings. The configuration details and results are shown in Section 6.2.2 for mono-task execution and in Section 6.3.1 for multi-task execution.

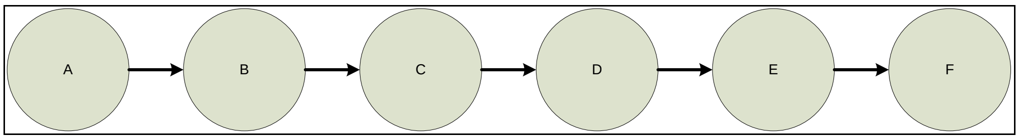


Figure 26 – Pipeline_6 application task graph.

Figure 27 presents the task graph of application (*iii*). In this application, *iVLC* is a CPU-intensive task. Tasks *iQuant* and *iDCT* are CPU-intensive, but present less processing workload than *iVLC*. Tasks *Start* and *Print* are used to initialize the system and to print the results, respectively. In this test case, 200 frames were transmitted. For multi-task execution, this application was simulated along with application (*i*) to simulate a disturbance. Section 6.2.3 presents the results for mono-task execution, while Section 0 presents the results for multi-task execution.

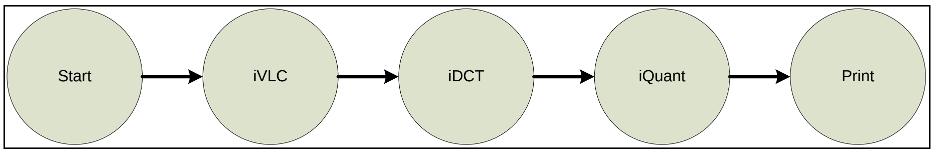


Figure 27 – MPEG partial filter application task graph.

Application (*iv*) task graph is depicted in Figure 28. *Task1* is responsible for reading the image attributes, sending it to *task3*, and processing an image block, sending it to *task2*. T*ask2* receives the processed block from *task1* and applies an inverse discrete cosine transform (iDCT) before sending it to *task3*. Finally, *task3* performs the color conversion of the blocks received from *task2* based in the information received from *task1*, generating the final image. In this test case, the image is decoded three times. The results are presented in Section 6.2.4.

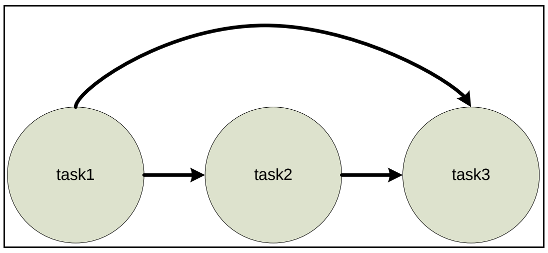


Figure 28 – JPEG decoding task graph.

Application (*v*) task graph is presented in Figure 29. It is composed by 10 tasks, being one caller *Master,* which divides the matrix in submatrices, sending it to 9 *slaves*. The algorithm starts with the *Master* Task sending the sub matrices to all of the *slaves*. The *slave* tasks apply the Fox algorithm [FOX11], exchanging messages among them. The result is sent back to the *Master* task after computation finishes. To increase the computation time of each task, each *slave* repeats the multiplication process 25 times. This is performed to allow the DFS controller to evaluate and act over the system, since in the original algorithm each *slave* task quickly finishes its computation. In this test case a matrix of 11 rows and 11 columns (11x11) was used to carry out the evaluation. The results are shown in Section 6.2.5.

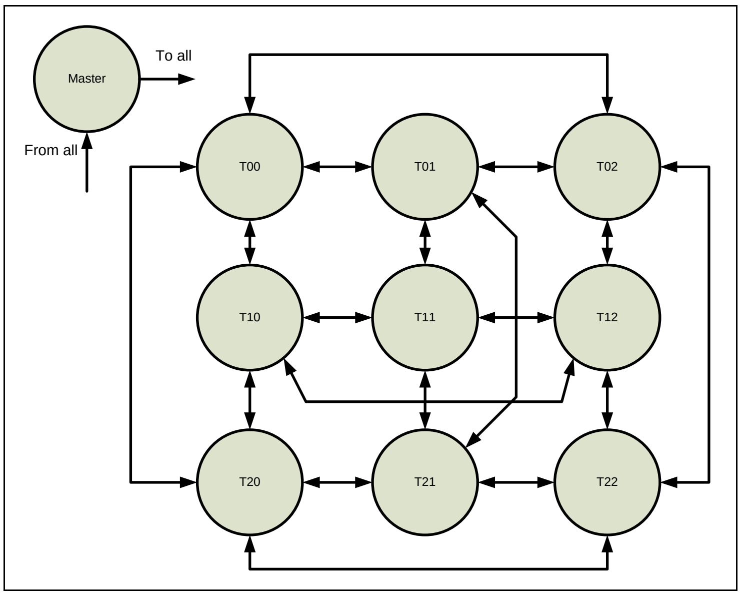


Figure 29 – Fox algorithm for matrix multiplication task graph.

Application (*vi*) is a benchmark in the multimedia field and its task graph is showed in Figure 30. The application is composed by 12 tasks. Two tasks represent memory elements (*Stripe-Mem* and *VOP-Mem*), i.e. no computation is performed by these tasks. A loop was included for each task to perform a computation, since the available application for use models only the communication workload. The duration of the loop is configured for each task according to its respective complexity [VAN02]. The results are presented in Section 6.2.6 for mono-task execution and in Section 6.3.3 for multi-task execution.

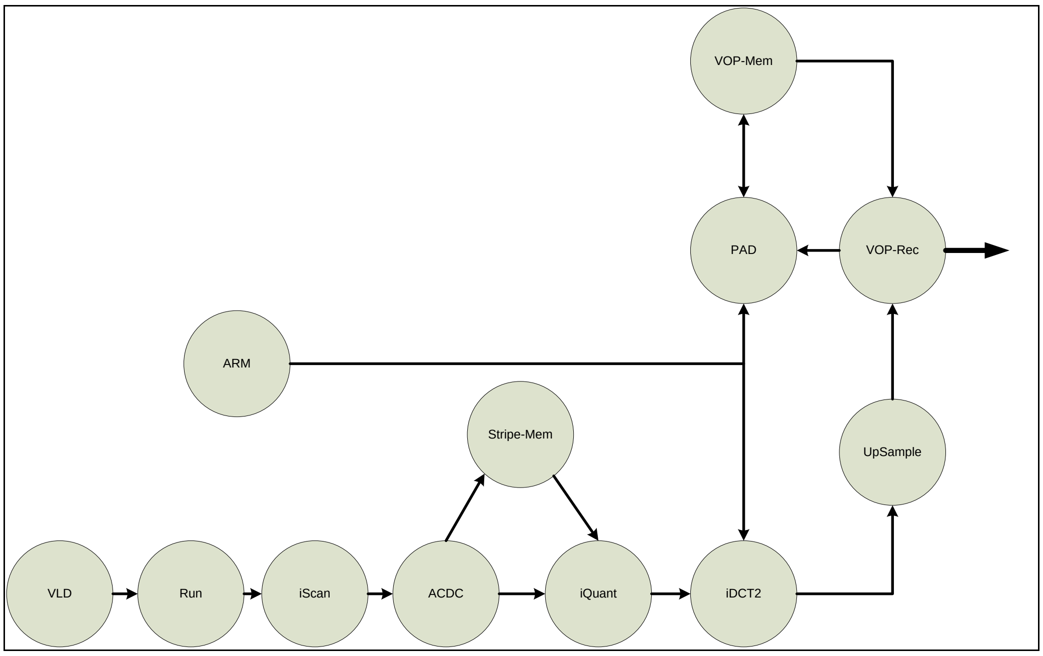


Figure 30 – VOPD task graph.

Nine levels compose the set of available frequencies for all experiments. Increasing or decreasing the frequency in one or two steps means increasing or decreasing the frequency in one/two levels according to the available values. In this work, the maximum available frequency was set at 100MHz, while the minimum value is 6,67MHz. The other available values are 90%, 75%, 60%, 50%, 40%, 25% and 10% of the maximum frequency.

6.1.1 PROCESSING ELEMENT EVALUATION

Three parameters were used to carry out the evaluations for the PEs: (*i*) number of executed instructions, (*ii*) total power dissipation and (*iii*) total execution time. Since a task uses a fixed number of instructions to finish its execution, the reduction or increase in the number of executed instructions comes from the difference that the CPU schedules the idle task, i.e. the lesser the processor executes the idle task, the less instructions will be executed and vice-versa. The evaluation in the number of executed instructions is performed by simulating the entire system using and not using the proposed DFS scheme. In these simulations, the NoC, NI, DMA and DFS controller are described in VHDL, while the Plasma CPU and the RAM memory are modeled in SystemC.

Likewise, the evaluation of power dissipation was performed by simulating the system using and not using the proposed DFS scheme. However, to estimate the power dissipation it was necessary to synthesize the PE with and without the DFS scheme, using a standard cell library. Thus, the generated *netlist* is used in the simulation to annotate the circuit switching activity. Then, both files (*netlist* and switching activity) are used as input for the power estimation tool. A 65nm standard cell library from ST Microelectronics was used to synthesize the PE in the Encounter RTL Compiler, a logic synthesis tool from Cadence. The generated Verilog *netlist* was simulated using the Incisive simulator, also from Cadence, and a *tcf* (toggle common format) file, containing all the switching activity was created. Then, the Verilog *netlist* and *tcf* file are used in the Encounter RTL Compiler for power evaluation Also, a wire load model is used to estimate wires power dissipation.

Finally, for each scenario, two reactivity levels of DFS controller were evaluated. The reactivity of the controller means how fast it acts over a variation in the monitoring parameters. Lower reactivity represents slower responses in the presence of variations, while higher reactivity represents faster responses. In the first reactivity level, lower, the controller takes into account only the computed average CPU utilization, i.e. large variations in the last evaluated period do not have great impact in the controller decision. In the second reactivity level, higher, the controller takes into account the computed average and the last measured CPU utilization, i.e. variations in the last evaluation may modify the controller decision. The total execution time, which is one of the evaluated parameters, is directly affected by the reactivity level as shown later.

6.1.2 NOC EVALUATION

The NoC evaluation is performed only in terms of power dissipation. Similarly to the PE power evaluation, both original and proposed architecture are synthesized, simulated with annotated switching activity and evaluated by a commercial tool. The tools used for logic synthesis, simulation and power evaluation are the same used for the PE. Besides the NoC size, the following parameters are fixed for all applications: *flit* size equal to 16 bits and buffer size equal to 16 positions. The NoC size varies according to the evaluated benchmark.

6.2 MONO-TASK EXECUTION

The six applications were evaluated in a monotask execution, i.e. only one task is executed in each PE. The applications are divided into synthetic and real. In synthetic experiments, application (*i*) was evaluated only in terms of number of executed instructions, due to its similarity with application (*ii*). In real experiments, application (*iv*) was evaluated only in terms of executed instructions due to its different memory configuration, which will be detailed in Section 6.2.4. Yet, as presented by [FIL09], the energy consumption of a processor is directly related to the number of executed instructions. Thus, the evaluation in the number of executed instructions also reflects in dissipated power. Finally, Section 6.2.7 draws conclusions for monotask executions.

6.2.1 COMMUNICATION

Each task in this application is composed by a loop that executes a sum operation over the transmitted/received vector values and a send and/or receive primitive. The number of loop iterations defines the task data processing rate and is configured according to each scenario, i.e. the greater the loop the processor executes, the higher the data processing rate will be. Three scenarios were evaluated using an MPSoC of size 2x3. Each scenario used a different loop configuration for the tasks. Table 8 details the relation between the data processing rates for each scenario. Figure 31 depicts the tasks frequency behavior using the second level of controller reactivity for each scenario, since it presented better results when compared to the first reactivity level.

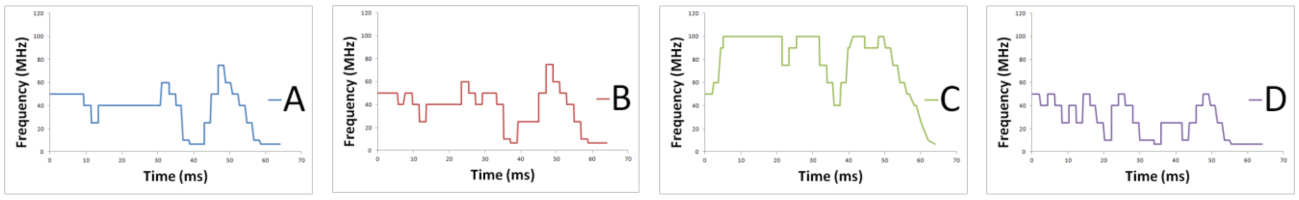
Table 8 – Evaluated scenarios for Communication application. '+' represents the data processing rate.

Scenario	*Data processing rate*		
	Producers	*Worker*	*Consumer*
1	++	++++	+
2	++	+	++++
3	++++	+	++

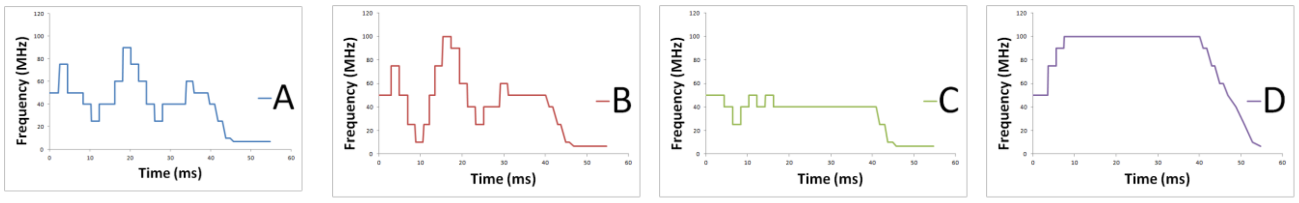
In scenario 1, the *worker* task (task C), which receives data from two producer tasks, reaches the reference frequency, since it is the slowest task. Note that the relationship between the *worker* and *consumer* frequency is around two (*worker* frequency equal to 100MHz and *consumer* frequency equal to 60MHz), even if the generation rate between them is four (Table 8 - Scenario 1). The reason is that the *worker* receives data from 2 producers, transmitting them to the consumer.

In scenario 2, the *consumer* is the slowest task, fastly reaching and staying at the reference frequency. Also, both *producers* had their frequencies increased, due to pending messages requested by the *worker* (the fastest task in this case). However, as the *consumer* consumes data too slowly, the other three tasks had their frequencies reduced due to the high pipe occupancy (20 – 30 ms). The system achieves a steady state between 30 to 35 ms.

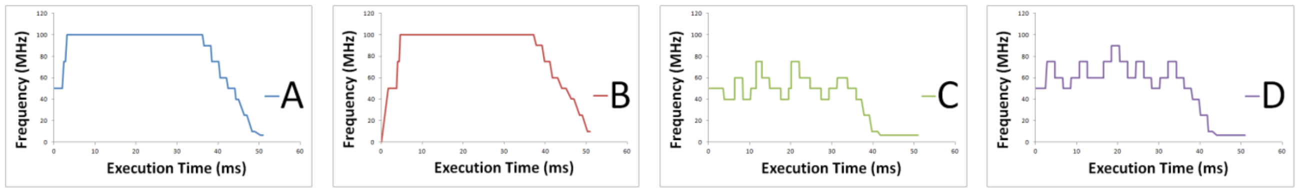
The third scenario stabilizes with the *producers* working at the reference frequency, the *consumer* working around 80% of the reference frequency and the *worker* operating at half of the reference frequency. The data processing rate relation between the three tasks explains this behavior.



Scenario 1



Scenario 2



Scenario 3

Figure 31 – Communication application tasks frequency behavior. (1) Worker as the slowest task and the consumer as fastest task. (2) Worker as the fastest task and consumer as the slowest task. (3) Producers as the slowest tasks and worker as the fastest task.

Table 9 presents the number of executed instructions for each scenario, considering both controller reactivity levels, as well as the total execution time. It is shown that increasing the controller reactivity, the number of executed instructions is penalized, but the total execution time is decreased. The first reactivity level reduced the average number of executed instructions by 29.2% and penalized the average execution time in 38.37%. On the other hand, the second reactivity level reduced the average number of executed instructions by 28.7%, but only penalized the average execution time in 13.4%.

Table 9 – Communication application results.

Scenario	*Number of executed machine instructions (in millions)*			*Execution Time (ms)*		
	***Without** DFS*	***With** DFS – React. 1*	***With** DFS – React. 2*	***Without** DFS*	***With** DFS – React. 1*	***With** DFS – React. 2*
1	14.65	9.09	9.37	39.50	52.54	49.20
2	13.30	9.26	9.12	36.85	48.43	40.15
3	13.61	10.96	11.06	34.77	52.40	37.11

6.2.2 PIPELINE WITH 6 TASKS

Two scenarios were evaluated for this application using an MPSoC of size 3x3. In the first scenario, all tasks had the same computation workload, modeled by a configurable loop as in application (*i*). In the second scenario, three computation workloads were used: low (tasks *B* and *E*), medium (tasks *A* and *D*) and high (tasks *C* and *F*). The data processing rate for both scenarios is presented in Table 10. Figure 32 shows the frequency behavior of each task for scenario 1, while Figure 33 shows the frequency behavior of each task for scenario 2.

Table 10 – Evaluated scenarios for 6 tasks Pipeline application. '+' represents the data processing rate.

Scenario	*Data processing rata*					
	A	*B*	*C*	*D*	*E*	*F*
1	++	++	++	++	++	++
2	++	+	++++	++	+	++++

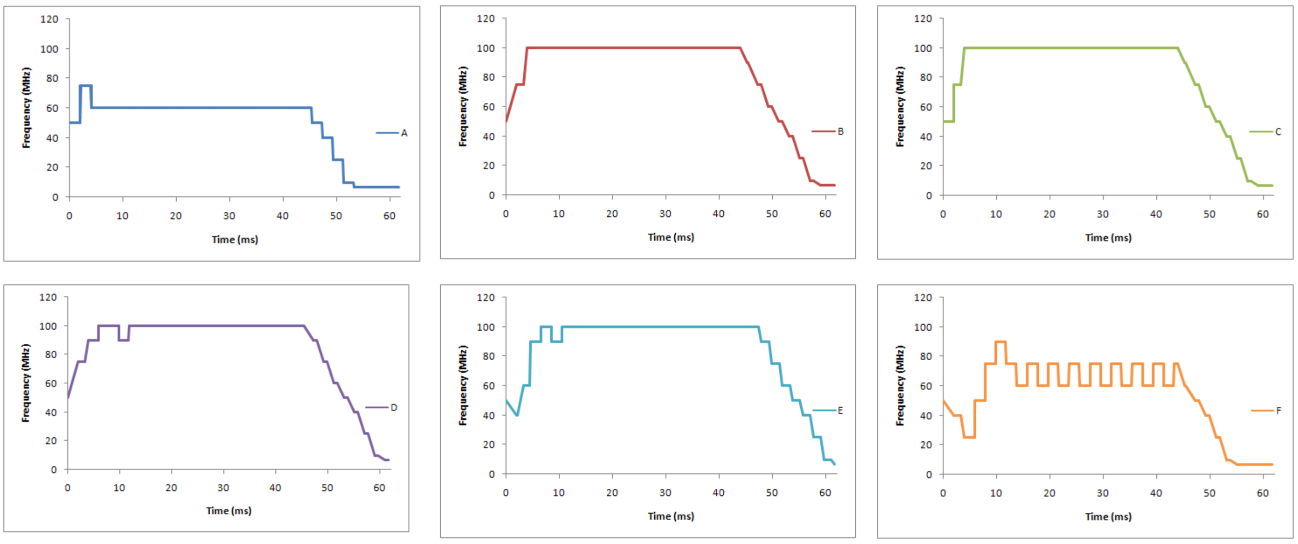


Figure 32 – Pipeline with 6 tasks application frequency behavior for scenario 1. All Tasks have the same workload.

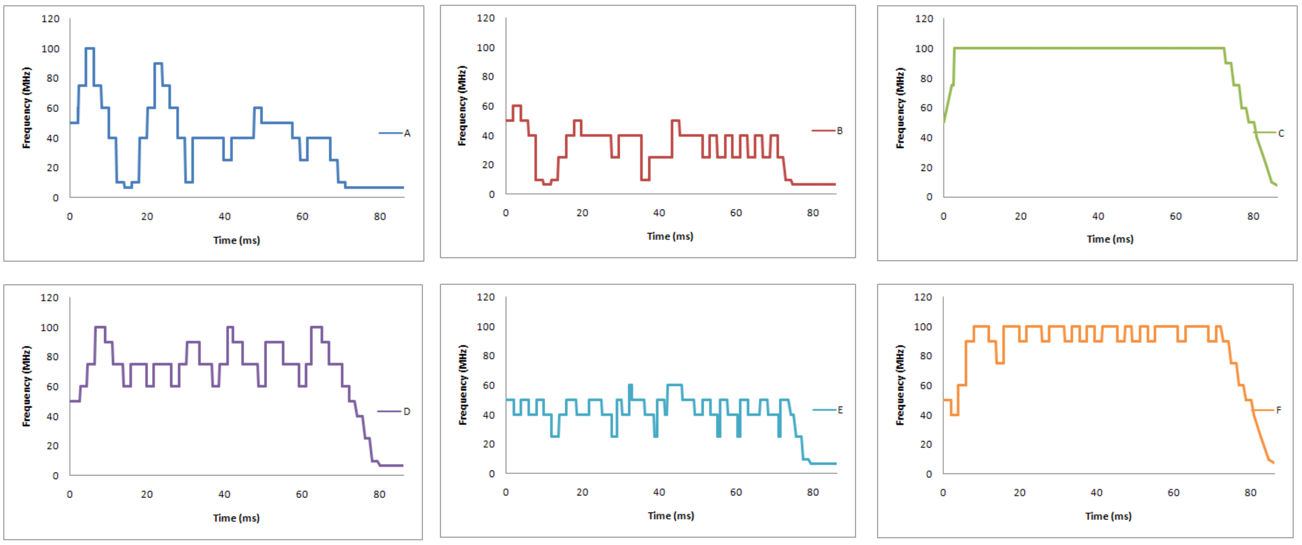


Figure 33 – Pipeline with 6 tasks application frequency behavior for scenario 2. Tasks have different workload.

In scenario 1, all tasks except *A* and *F* had their frequencies increased to the reference frequency. Since *A* and *F* have lower communication load compared to the other tasks (one send for task *A* and one receive for task *F*, against one send and one receive for other tasks), their frequency stabilizes at around 60% of the reference frequency. Since the data produced by task *A* is always available, task *B* increases its frequency to the reference frequency. Similarly, tasks *C*, *D* and *E* follow the same behavior. All tasks with same communication load and data processing rate (CPU utilization) presented same frequency behavior. Thus, the DFS scheme could only change the frequency of tasks A and F, since their communication load is smaller.

In scenario 2, the frequency is defined by the computation workload. Tasks with higher computation workload (*C* and *F*) had their frequencies increased to the reference frequency. Task *F* presented a small variation due to its lower communication load. Tasks with medium computation workload (*A* and *D*) had their frequencies increased in the beginning of the simulation. While task *D* kept the frequency around 75% of the reference frequency, task *A* presents an oscillation in the beginning of the simulation. This is explained by the time that task *C* needs to increase its frequency and start consuming messages from task *B*, which also reflects in task *A*. Later, task *A* had its frequency stabilized around 40% of the reference frequency due to the lower communication load, compared to task *D*. Tasks with low computation workload (*B* and *E*) had their frequencies decreased to around 40% of the reference frequency, an expected behavior since these tasks need less time to produce data.

Table 11 presents the results for both scenarios in terms of number of executed instructions and total execution time. In scenario 1, with the same computing workload for all applications, the savings in the number of executed instructions are small (7.6% and 4.4% for reactivity levels 1 and 2 respectively). This can be explained due to the nature of the application, which makes almost all tasks operate at the reference frequency. The decrease in the number of executed instructions comes from the tasks which had their frequencies reduced. The execution time increased by 13.64% for reactivity level 1 and 11.93% for reactivity level 2. On the other hand, scenario 2 presented significant savings in the number of executed instructions and a smaller execution time overhead. The reactivity level 1 presented a reduction of 37% in the number of executed instructions and a 4.88% increase in the execution time, while reactivity level 2 presented 34.56% and 3.23% respectively. Again, with higher reactivity level the execution time overhead is decreased and the number of executed instructions is increased, compared to a lower reactivity level.

Table 11 – Pipeline with 6 tasks results.

Scenario	*Number of executed machine instructions (in millions)*			*Execution Time (ms)*		
	***Without** DFS*	***With** DFS – React. 1*	***With** DFS – React. 2*	***Without** DFS*	***With** DFS – React. 1*	***With** DFS – React. 2*
1	20.00	18.48	19.12	41.06	46.66	45.96
2	36.00	22.69	23.56	70.03	73.45	72.29

Due to the simulation time required to evaluate the system, only the scenario with the second controller reactivity level was simulated to evaluate the power consumption. Table 12 shows the obtained results. The power reduction was 20.85% for PEs and 72.58% in the NoC. It is important to note that the power reduction in the CPUs, 32.53%, is close to the reduction in the number of executed instructions, 34.56%.

Table 12 – Pipeline with 6 tasks power consumption for scenario 2.

	Total Power (mW)				
	RAM	***CPU***	***DFS Controller***	***PEs***	***NoC***
With DFS	83.04	17.97	2.46	105.73	3.15
Without DFS	99.08	26.64	-	133.58	11.48
Reduction	**16.19%**	**32.53%**	**-**	**20.85%**	**72.58%**

The total power reduction for NoC and PEs is around 25% for this scenario (5.7% by NoC and 19.3% by PEs). Also, the significant reduction in the NoC power consumption does not represent the same percentage in the total power reduction, as it is show in Figure 34.

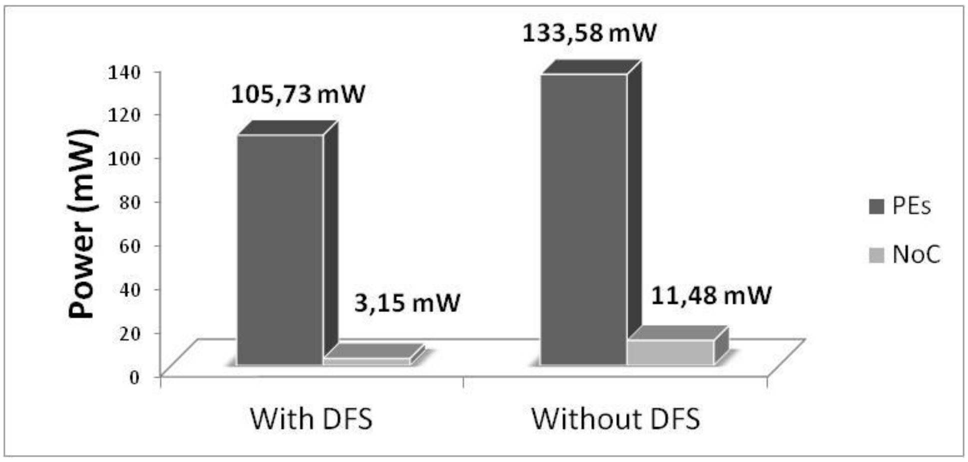


Figure 34 – PEs and NoC power dissipation comparison for 6 tasks pipeline application.

6.2.3 MPEG

This application was evaluated using an MPSoC of size 2x3. Using the proposed DFS scheme, only the task with high computation workload (*iVLC*) had its frequency increased to the reference frequency. The tasks *Print* and *Start* had their frequency decreased to the lowest frequency, while tasks *iDCT* and *iQuant* oscillated between 25MHz and 40MHz, due to the lower computation workload when compared to *iVLC* task. Figure 35 shows each task frequency behavior along the time.

Table 13 shows the obtained results in terms of number of executed instructions and execution time. The first reactivity level presented a reduction of 65.2% in the number of executed instructions and 10.4% of execution time overhead. Yet, the second reactivity level presented a reduction of 65.1% in the number of executed instructions and 9.4% of execution time overhead. Thus, for this application, using the second reactivity level presented better results, since the difference in the number of executed instructions is negligible and the execution time overhead is decreased by 1%, compared to the first reactivity level.

Table 13 – Number of executed instructions and execution time results for MPEG application.

Scenario	*Executed Instructions (in millions)*	*Reduction (Instructions)*	*Execution Time (ms)*	*Overhead (Exec. Time)*
Without DFS	37.93	-	86.24	-
With DFS – React. 1	13.20	**65.2%**	95.23	**10.4%**
With DFS – React. 2	13.23	**65.1%**	94.40	**9.4%**

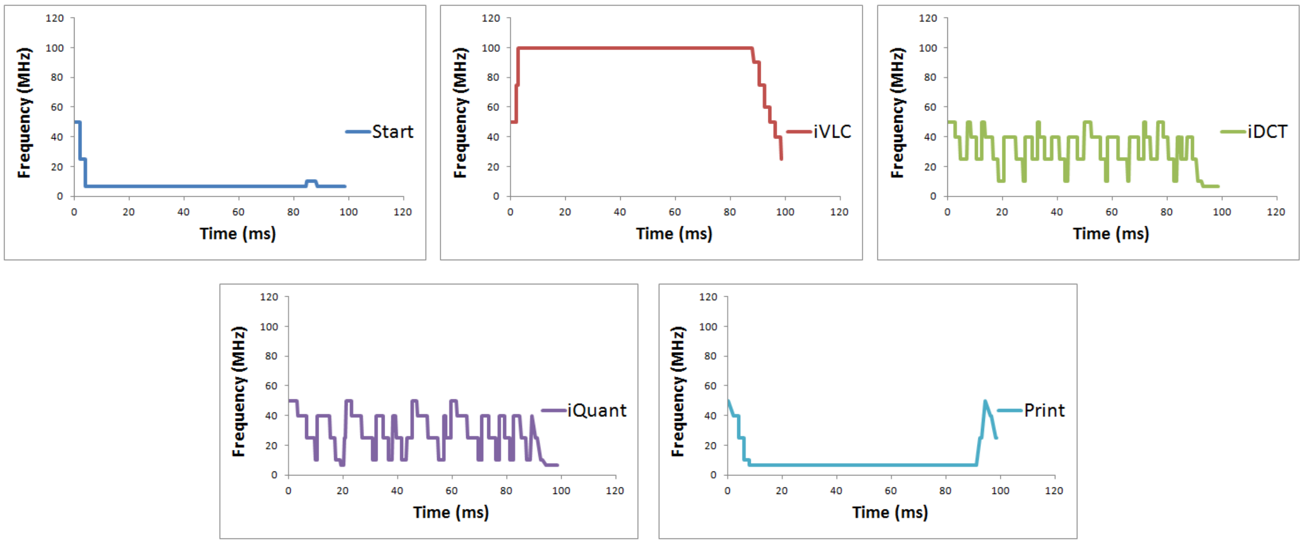


Figure 35 – Frequency behavior of MPEG application tasks.

Table 14 shows the obtained results in terms of power dissipation. Since the second reactivity level presented better results, only this scenario is evaluated in this analysis. The total power reduction in the system was 52.28% (4.85% by NoC and 47.43% by PEs). The power reduction in CPUs is similar to the reduction in number of executed instructions (~57% and ~65%), however, as the RAM memory had lower power reduction (~49%) and is the module that has the highest power dissipation, the PE presented a power reduction of 50.62%.

Table 14 – Power dissipation results for MPEG application with and without the proposed DFS scheme.

	Total Power (mW)				
	RAM	*CPU*	*DFS Controller*	*PEs*	*NoC*
With DFS	39.04	9.05	1.77	51.17	1.61
Without DFS	76.34	21.09	-	103.63	6.98
Reduction	**48.86%**	**57.08%**	**-**	**50.62%**	**76.9%**

Although the achieved NoC power reduction is around 77%, its contribution in the total power reduction is only around 5%. Figure 36 shows the relation for the power consumption with and without the proposed DFS scheme. It can be seen that the contribution of the PEs power dissipation is much larger than the NoC contribution. Considering only the obtained power reduction, the contribution of PEs power reduction is around 90% and the contribution of the NoC is around 10%.

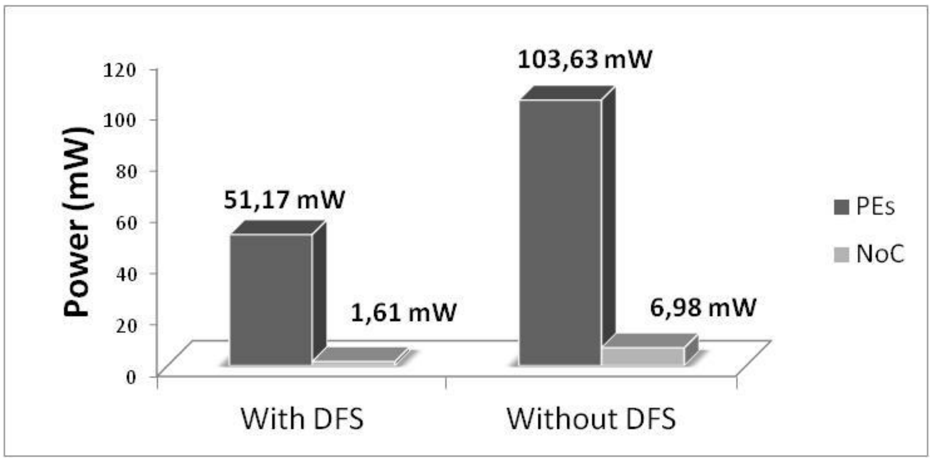


Figure 36 – PEs and NoC power dissipation comparison for MPEG application.

6.2.4 JPEG

This application was evaluated using an MPSoC of size 2x2. Also, the PE RAM memory has to be modified to divide the pages into 32KB pieces each, instead of 16KB, to accommodate the object code of task1. This modification implies changing the internal structure of the Plasma CPU, and consequently, in synthesizing the PEs for this configuration. Thus, for the sake of simplicity, the evaluation is performed only in terms of number of executed instructions, since the latter can be directly related to power dissipation as shown in previous analysis.

Tasks 2 and 3 have higher computation workload when compared to task 1. Thus, tasks 2 and 3 have their frequencies increased to the reference frequency, while task 1 has its frequency reduced to around 60% of the reference frequency. This is explained by the higher data rate generation of task 1, which sends the processed data faster than task 2 can consume (task 1 *pipe's* becomes full). Task 2 adjusts its frequency according to the pending message requests from task 3. In other words, the CPU utilization becomes high, since task 1 data is always available, and *pipe* occupation stays between operational and low states, since task 3 is requesting data. Consequently, after task2 reaches the reference frequency and makes the data available when it is requested, task3 reaches the reference frequency.

Table 15 presents the results for this application. The execution time overhead, compared to the execution with the whole system at the reference frequency is around 19% for the first level of reactivity and around 14% for the second level. The number of executed instructions is reduced by 4% using the first level of reactivity and by 5.5% for the second level. The low reduction in the number of executed instructions is explained by the decrease in the frequency of only one task, i.e. only one PE can have the number of executed instructions reduced. Also, the second level of reactivity benefits both executed instructions and execution time overhead. It is explained by the increase in the frequencies of tasks 2 and 3 to the reference frequency faster than the first reactivity level, making the processors that execute these tasks schedule the idle task for less time.

Table 15 – Number of executed instructions and execution time results for JPEG application.

Scenario	*Executed Instructions (in millions)*	*Reduction (Instructions)*	*Execution Time (ms)*	*Overhead (Exec. Time)*
***Without** DFS*	14.60	-	61.24	-
***With** DFS – React. 1*	14.02	**4.0%**	73.14	**19.4%**
***With** DFS – React. 2*	13.79	**5.5%**	69.90	**14.1%**

6.2.5 FOX 3X3

This application was evaluated using an MPSoC of size 4x3, with 9 processors executing the slave tasks, one processor executing the *Master* task and one unused processor. The application starts with the *Master* task sending submatrices to *slave* tasks. After receiving the submatrices, *slaves* start processing and exchanging messages with their neighbors. Since all tasks process the same size of submatrices, they send and receive the same amount of messages, and the CPU utilization and communication load are the same for all processors. Thus, all processors executing a *slave* task had their frequencies increased to the reference frequency. The only exception is the processor which executes the *Master* task, which had its frequency decreased to the lowest level. Table 16 presents the results in terms of number of executed instructions and execution time.

Table 16 – Number of executed instructions and execution time results for FOX 3x3 application.

Scenario	*Executed Instructions (in millions)*	*Reduction (Instructions)*	*Execution Time (ms)*	*Overhead (Exec. Time)*
***Without** DFS*	22.60	-	55.23	-
***With** DFS – React. 1*	20.87	**7.7%**	67.53	**22.3%**
***With** DFS – React. 2*	20.79	**8.0%**	61.05	**10.5%**

As the controller acts only over the *Master* task, the savings in the number of executed instructions are only limited to the latter. Also, since the best case is achieved with all *slave* tasks working at the reference frequency, it is impossible to obtain the same execution time and number of executed instructions of the system without the proposed scheme, due to the delay induced by each evaluation and the time that the controller needs to stabilize. Thus, the *slave* tasks presented an overhead in the number of executed instructions around 10%, while the *Master* task presented a reduction around 90%, explaining the average reductions around 8% for both reactivity levels. The execution time overhead presented by the first reactivity level is around 22%, against 10.5% obtained when using the second reactivity level. It shows that for applications with tasks containing the same amount of processing workload, faster responses to communication workload variability present better results. Consequently, the power evaluation is performed only for the second reactivity level, which is presented in Table 17.

Table 17 – Power dissipation results for FOX 3x3 application with and without the proposed DFS scheme.

	Total Power (mW)				
	RAM	***CPU***	***DFS Controller***	***PEs***	***NoC***
***With** DFS*	164.11	32.16	4.37	205.23	4.75
***Without** DFS*	177.19	40.16	-	231.14	15.51
Reduction	**7.8%**	**19.92%**	**-**	**11.21%**	**69.34%**

Only PEs executing some task are considered in the evaluation. The results show a reduction of 11.2% for PEs (including the DFS controller) and 69.3% for the NoC. In this application the power reduction is greater than the reduction in the number of executed instructions (11.21% and 8.01%). A detailed study is needed to explain this behavior, and it is left as a future work. Figure 37 presents the relation between the PEs and NoC consumption using and not using the proposed DFS scheme. The reduction of dissipated power in the whole system was of 14.86%, being 10.50% by PEs and 4.36% by the NoC.

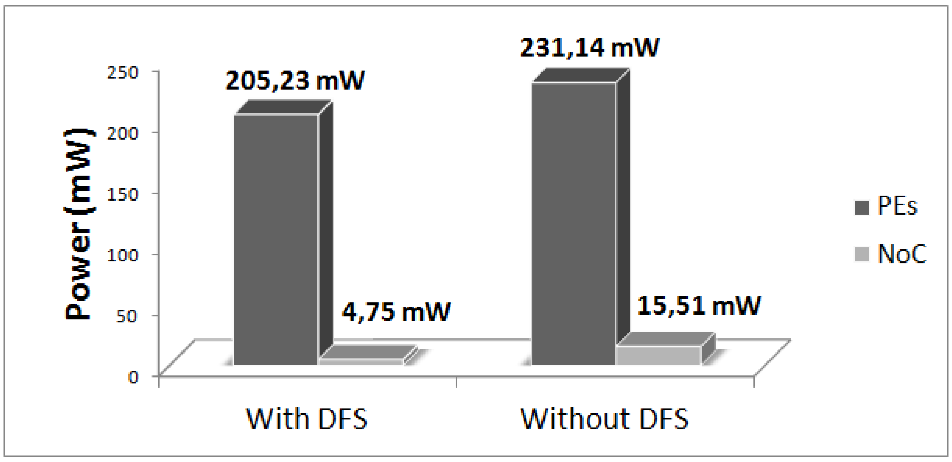


Figure 37 - PEs and NoC power dissipation comparison for FOX 3x3 application.

6.2.6 VOPD

This application was evaluated in an MPSoC of size 4x4, using 12 Slave-PEs and 1 Master-PE. The 3 remaining processors are not taken into account in the evaluations. The task mapping was performed so that the initial tasks are the first to be allocated (*VLD*, *ARM*, and so forth), and the final tasks (*PAD*, *VOP-Rec* and *VOP-Mem*) are the last allocated. Figure 38 shows the frequency along the time for each task using the second reactivity level. The two initial tasks (*VLD* and *ARM*) start producing messages and quickly reach the reference frequency. Since the task *VLD* produces messages for a task that is subsequently allocated, it keeps its frequency around the maximum level until the end of the application. On the other hand, the task *ARM*, which produces messages for *IDCT2* and *PAD*, fills its *pipe* and has its frequency decreased. Later, with all tasks allocated (around 50 ms), the *ARM* task has its frequency increased, and all the other tasks stabilize their frequencies according to their respective computation workload. Still, some variation can be noted due to the fact that frequency levels are not continuous, i.e. not all frequencies are available to be used. Thus, if a processor oscillates between 90MHz and 100MHz, for the same amount of time, it is possible that the ideal frequency to operate would be around 95MHz.

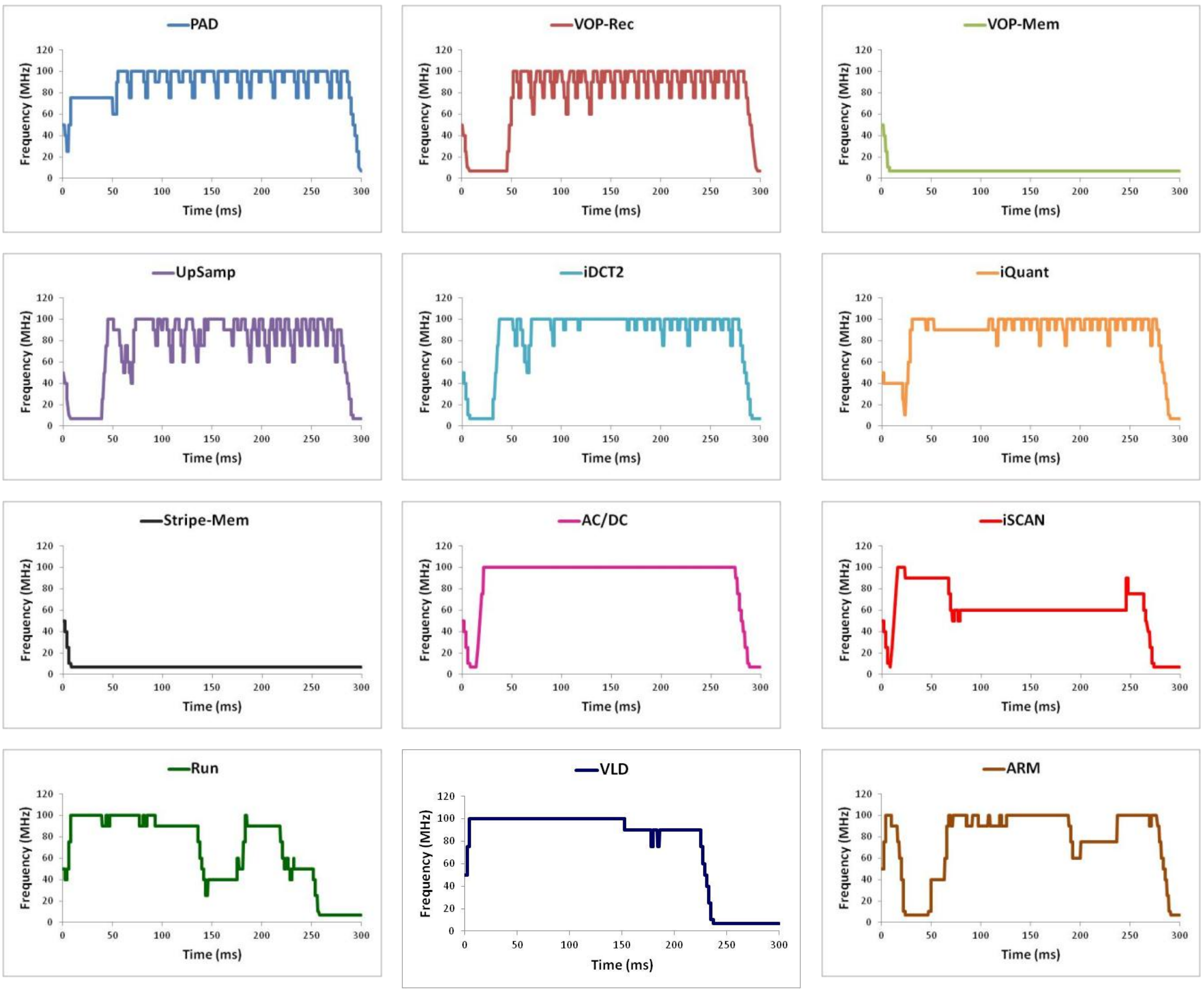


Figure 38 – Frequency behavior of VOPD application tasks.

Table 18 presents the results in terms of number of executed instructions and execution time. The first reactivity level reduces the number of executed instructions by 27.70% and induces 18.33% overhead in execution time, while the second level reduces by 27.30% the number of executed instructions and induces 7.25% overhead in execution time. The significant difference in the execution time overhead between the two reactivity levels occurs due to the quick increase in the frequencies of tasks allocated later (*IDCT2*, *UpSamp*, *VOP-Rec*).

Only the second reactivity level was evaluated in terms of power dissipation. Results are presented in Table 19. Similarly to other applications (besides application (*v*)), the reduction in power dissipation is nearly the same as the reduction in number of executed instructions, especially when only the CPUs are taken into account (27.3% and 26.6% respectively). Power reduction in the NoC achieved around 75%. The relation between NoC dissipation and PEs dissipation is depicted in Figure 39.

Table 18 – Number of executed instructions and execution time results for VOPD application.

Scenario	*Executed Instructions (in millions)*	*Reduction (Instructions)*	*Execution Time (ms)*	*Overhead (Exec. Time)*
Without DFS	313.90	-	266.30	-
With DFS – React. 1	226.94	**27.7%**	315.10	**18.3%**
With DFS – React. 2	228.19	**27.3%**	285.6	**7.3%**

Table 19 – Power dissipation results for the VOPD application with and without the proposed DFS scheme.

	Total Power (mW)				
	RAM	*CPU*	*DFS Controller*	*PEs*	*NoC*
With DFS	163.52	34.55	4.92	206.98	5.48
Without DFS	219.93	45.82	-	279.31	22.18
Reduction	**25.31%**	**26.60%**	**-**	**25.89%**	**75.29%**

Similarly to other applications, the reduction of NoC power dissipation is a small fraction of the total power reduction. In this application, the total power reduction for the whole system is 29.53%, being 5.54% the NoC contribution and 23.99% the PEs contribution.

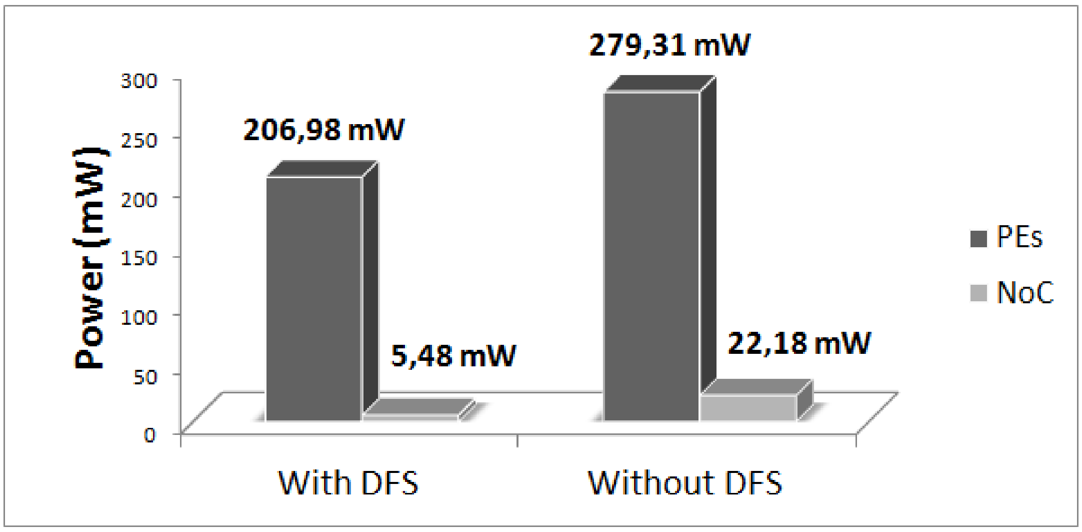


Figure 39 – PEs and NoC power dissipation comparison for VOPD application.

6.2.7 MONO TASK FINAL REMARKS

The results for mono task execution show that the proposed DFS scheme is able to tune the frequency of each processor in the MPSoC according to the application characteristics. Decreasing the operating frequency of a given processor leads to decrease in power dissipation. However, with the induced execution time overhead, application performance is penalized. Thus, a DFS controller scheme must induce as the least possible execution time overhead, to maximize both power and energy savings.

All the evaluated applications presented reduction in number of executed instructions and in power dissipation. However, in some scenarios, the induced execution time overhead could increase the final energy consumption, since energy and time are directly correlated. Still, for each evaluated scenario, two reactivity DFS controller levels were evaluated. The first level is less reactive, which means the responses are slower when a monitoring parameter changes. This reactivity level presented high execution time overhead in some scenarios, e.g. 50% in application (*i*), and 22% in application (*v*). Contrarily to this, the second reactivity level showed better performance than the first level for all scenarios. This level is more reactive, which means the responses are faster when a monitoring parameter changes. In other words, the second reactivity level increases, or decreases operating frequency faster than the first one. As a drawback, this reactivity level presents lower reduction in the number of executed instructions compared to the first one. The second reactivity level only increased the number of executed instructions between 3.2% and 2% when compared to the first one. For these reasons, the second reactivity level was used for power evaluations and discussing the frequency behavior in each scenario.

Considering all experiments, the reduction in the number of executed instructions ranges from 4.4% to 65.1%, while the execution time overhead ranges from 3.2% to 14.1%.

Applications composed by tasks with similar computation and communication workload ((*iv*), (*v*) and scenario 1 of application (*ii*)) present lower reduction in the number of executed instructions and lower power savings, with higher execution time overhead, when compared to applications with varied computation and communication workload. This is due to the fact the controller sets the tasks frequencies to the highest level, finishing their execution as soon as possible. Therefore, the reduction in the number of executed instructions comes from the task with the lower computation and/or communication workload, while the initial time needed to stabilize the system penalizes execution time.

On the other hand, applications composed by tasks with varied computation and communication workload provide higher flexibility for the controller. In such applications, the reduction in the number of executed instructions and power savings ranges from 18.7% to 65.1% and from 25% to 52%, respectively. The execution time overhead ranges from 3.2% to 9.5%. For comparison purposes, the reduction in the number of executed instructions for applications with similar computation and communication workload ranges only from 4.4% to 8%. Nevertheless, the application behavior does not interfere in the NoC power savings, which was around 70% for all applications.

6.3 MULTI-TASK EXECUTION

Four applications were used to evaluate multi-task execution: Communication (*i*) (only used in one scenario of Section 0), Pipeline with 6 tasks (*ii*), MPEG (*iii*) and VOPD (*vi*). The first evaluation is performed by four configurations of the application (*ii*) and presented in Section 6.3.1. Section 0 presents the results for the second evaluation, where the application (*iii*) is evaluated in three configurations. The last evaluation comprises the application (*vi*) and is presented in Section 6.3.3. Finally, in Section 6.3.4, conclusions are drawn about the multi-task execution results.

6.3.1 PIPELINE WITH 6 TASKS

This application was evaluated in four different configurations. First, the tasks are mapped forming *macro tasks*, as it is shown in Figure 40. The term *macro task* is used to define a set of tasks mapped in a processor that is viewed as only one task by the system. For example, when task B needs to send a message to task C, the system can write the data in the memory space reserved for task C, avoiding the use of the NoC. Next, the computation workload of tasks are varied, composing two configurations for the *macro task* mapping: (a) *average*, where all tasks present the same computation workload; (b) *mix*, where the tasks present different computation workloads. Also, each configuration was evaluated with both DFS controller reactivity levels. The frequency behavior of each task for the second reactivity level is presented in Figure 41 and Figure 42 for *average* and *mix* configurations, respectively.

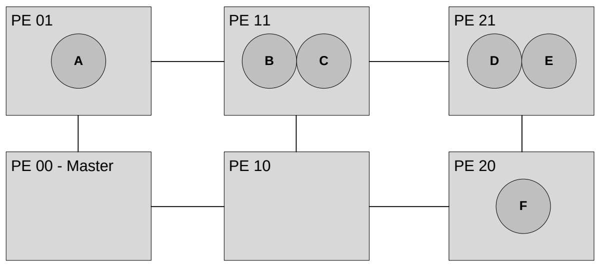


Figure 40 – Macro task mapping.

In the *average* configuration (Figure 41), processors executing two tasks had their frequency increased to the reference frequency, while processors executing one task had their frequency oscillating around 50% of the reference frequency, following the data generation rate imposed by the computation workload. In the *mix* configuration (Figure 42) three levels of computation workload were used: low (tasks C and E); medium (tasks A and D); high (tasks B and F). Accordingly, the processor executing task A had its frequency oscillating between 25MHz and 40MHz, since task B consumes data more slowly compared to the *average* configuration. The processor executing tasks B and C had its frequency increased to the reference frequency, satisfying the workload of both tasks. On the other hand, the processor executing tasks D and E oscillates between 75MHz and 100MHz, since tasks D and E present lower computation workload.

Finally, the processor executing task F had its frequency increased to supply the high computation workload demanded by its task.

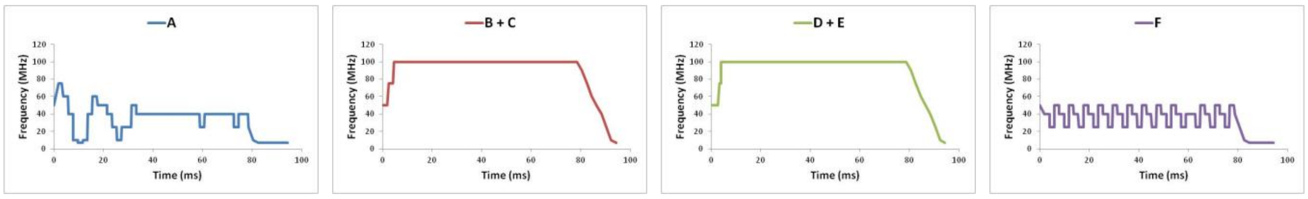


Figure 41 – 6 tasks pipeline application frequency behavior for multitask execution. Macro task mapping with the same computation workload for all tasks.

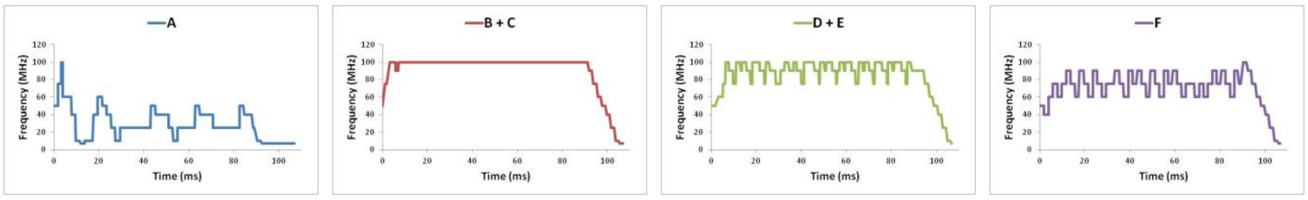


Figure 42 – 6 tasks pipeline application frequency behavior for multitask execution. Macro task mapping with varied computation workload for each task.

The other two evaluated scenarios use a task mapping that isolates the communicating tasks in different processors, as it is shown in Figure 43. As done for the *macro task* mapping, the computation workload of the tasks is varied to generate two scenario configurations, which received the same names (*average* and *mix*). Also, both DFS controller reactivity levels were evaluated. Figure 44 and Figure 45 present the frequency behavior for the second reactivity level in *average* and *mix* configurations respectively.

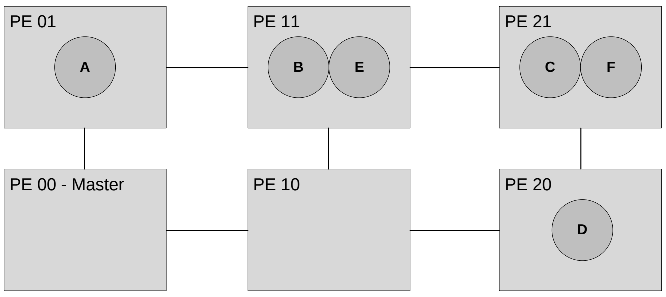


Figure 43 – Isolated task mapping.

Figure 44 presents results similar to Figure 41. The difference is in the processor that executes task D. Since task D presents a higher communication load compared to task F, its processor had to increase frequency to around 60MHz to supply the demand, contrary to what happened for task F in Figure 41. In the *mix* configuration, the task mapping was done in a way to place together tasks with contrasting characteristics. Thus, tasks with high and low computation workload were mapped in the same processor. The results are presented in Figure 45. While the processors with two tasks had to increase their frequency to the reference frequency to supply the demand of both tasks, the processor executing task A stabilized its frequency around 25MHz and the processor executing task D oscillated mostly between 40MHz and 50MHz. Again, the difference between the processors executing tasks A and D is due to the difference in communication load between the tasks.

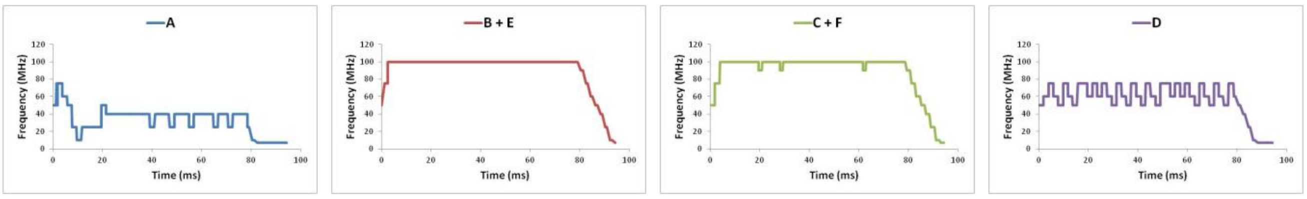


Figure 44 – 6 tasks pipeline application frequency behavior for multitask execution. Isolated task mapping with the same computation workload for all tasks.

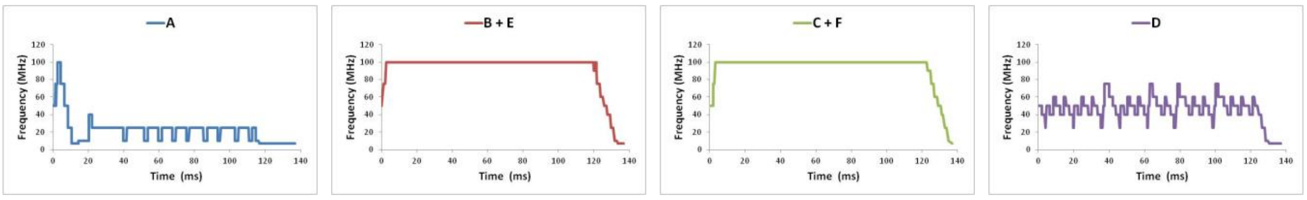


Figure 45 – 6 tasks pipeline application frequency behavior for multitask execution. Isolated task mapping with varied computation workload for each task.

Table 20 presents the results in terms of executed instructions and execution time for all the simulated scenarios.

Table 20 – Number of executed instructions and execution time results for the evaluated scenarios of 6 tasks pipeline application in multitask execution.

Config.	*Scenario*	*Executed Instructions (in millions)*	*Reduction (Instructions)*	*Execution Time (ms)*	*Overhead (Exec. Time)*
Macro Average	*Without DFS*	25.75	**-**	77.57	**-**
	With DFS – React. 1	17.41	**32.39%**	82.10	**5.84%**
	With DFS – React. 2	17.42	**32.35%**	80.16	**3.34%**
Macro Mix	*Without DFS*	30.73	**-**	90.60	**-**
	With DFS – React. 1	21.84	**28.93%**	92.98	**2.63%**
	With DFS – React. 2	22.87	**25.58%**	91.94	**1.48%**
Isolated Average	*Without DFS*	25.73	**-**	77.81	**-**
	With DFS – React. 1	18.84	**26.78%**	79.23	**1.82%**
	With DFS – React. 2	19.30	**24.99%**	79.05	**1.59%**
Isolated Mix	*Without DFS*	41.56	**-**	119.24	**-**
	With DFS – React. 1	26.14	**37.10%**	122.36	**2.62%**
	With DFS – React. 2	28.77	**30.77%**	121.86	**2.20%**

Due to the simulation time required to evaluate the system, the power dissipation analysis was not performed for this application. However, based on data collected in Section 6.2, the reduction in the number of executed instructions should lead to proportional power savings. Generally, the second reactivity level induces smaller execution time overhead at the cost of a lower reduction in the number of executed instructions. Yet, the difference between the two reactivity levels is smaller than presented in Section 6.2. This is explained by the fact that the tasks in this application do not drastically change their behavior during execution as it happens, for example, in task *VOP-Rec*, described in Section 6.2.6. The reduction in the number of executed instructions ranges from 25% to 37%, while the execution time overhead ranges from 1.5% to 5.8%.

6.3.2 MPEG

This application was evaluated in three different scenarios. The first scenario evaluates tasks with similar characteristics mapped to a same processor, while the second scenario evaluates tasks with different characteristics mapped to a same processor. The third scenario evaluates the MPEG execution in the presence of another application in the system. The task mappings for the first, second and third scenarios are showed in Figure 46(a), Figure 46(b) and Figure 46(c) respectively.

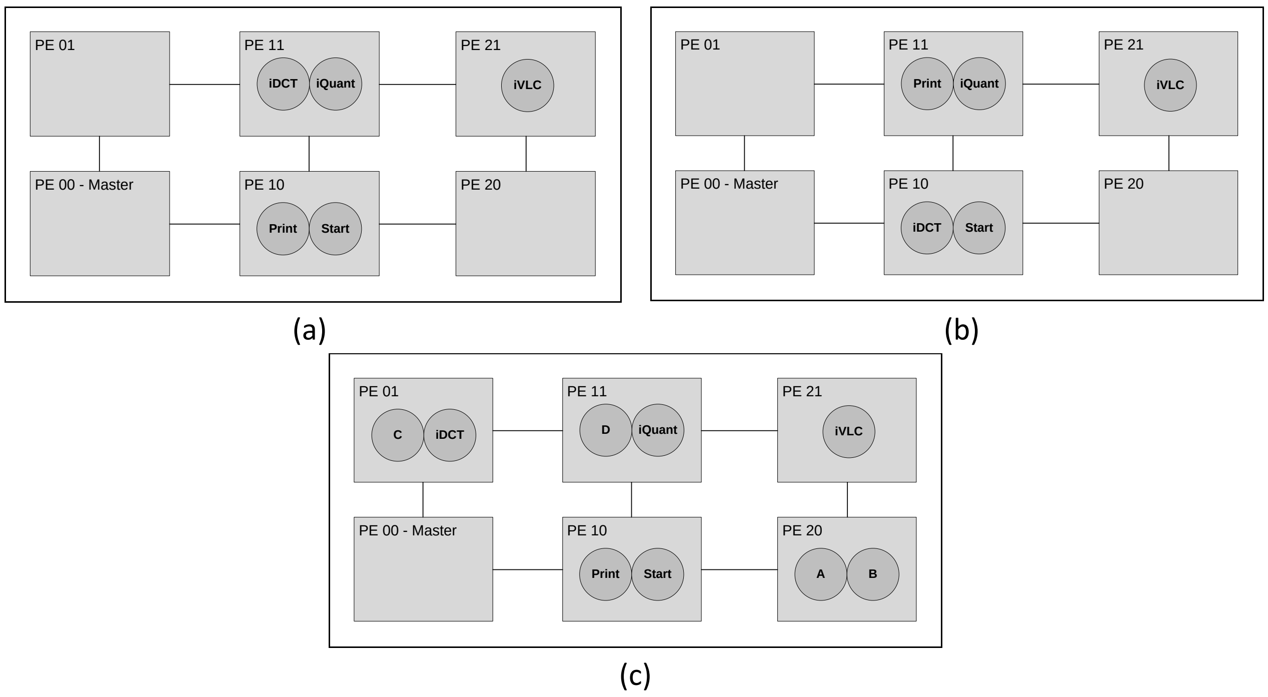


Figure 46 – Task mapping for the evaluated scenarios of MPEG application in multitask execution.

Figure 47 illustrates the processors frequency behavior for the first scenario with the second DFS controller reactivity level. In this scenario, the processor executing tasks *Print* and *Start*, which present low computation workload, barely increased its frequency. The processor executing tasks *iDCT* and *iQuant* oscillated mostly between 40MHz and 60MHz, while the processor executing task *iVLC* kept its frequency at the reference frequency. It shows that, even with both tasks *iDCT* and *iQuant* being allocated in the same processor, there is no need to increase the frequency due to the high computation workload demanded by *iVLC*.

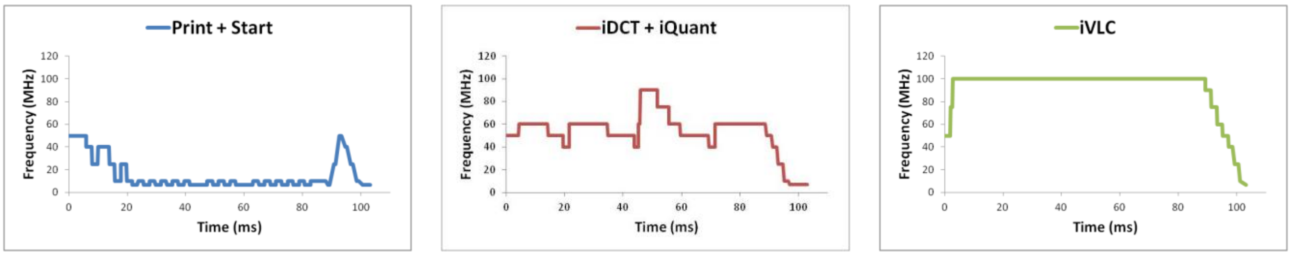


Figure 47 – Frequency behavior of MPEG application tasks in the first scenario of multitask execution.

Figure 48 illustrates the processor frequency behavior for the second scenario. Again, the processor executing task *iVLC* kept its frequency at the reference frequency. On the other hand, the other two processors had their frequencies oscillating mostly between 25MHz and 50MHz. In the processor executing tasks *iDCT* and *Start*, the policy proposed in Section 4.3 does not let task *Start* occupy the entire *pipe*, leading task *iDCT* to be scheduled frequently, and making its characteristics guide the DFS controller. Similarly, the processor executing tasks *iQuant* and *Print* is guided by the characteristics of task *iQuant*, since task *Print* presents low computation workload and is scheduled only when a new message is available.

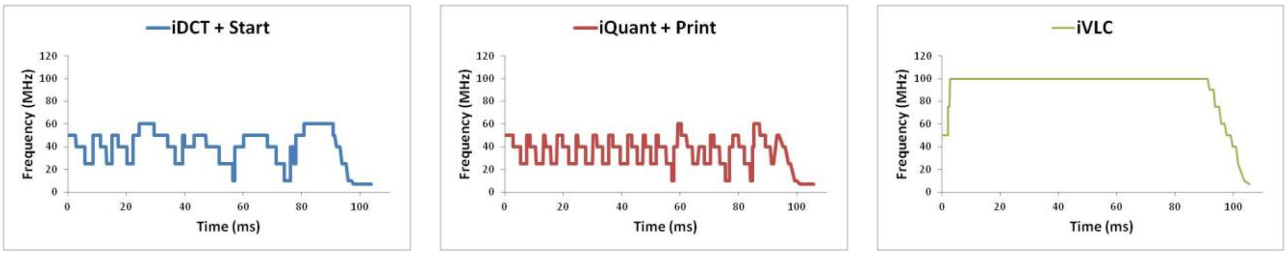


Figure 48 – Frequency behavior of MPEG application tasks in the second scenario of multitask execution.

The frequency behavior of the processors for the third scenario is showed in Figure 49. The configuration used for application (*i*) is the same as the one in line 3 of Table 8. The processor executing task *iVLC* and the processor executing tasks *Print* and *Start* presented the same behavior of the first scenario. The processor executing tasks *iQuant* and D had its frequency increased to the reference frequency, since both tasks present a significant computation workload. The processor executing tasks *iDCT* and C kept its frequency around 50MHz. As it happened in the previous scenario, the multi task policy prevents task C, which has a high data generation rate, from filling the *pipe*, scheduling task *iDCT* more frequently. Thus, this processor does not need to increase the frequency to supply the demand of both tasks. Consequently, the processor executing tasks A and B oscillates between 75MHz and 100MHz, since their consumer (task C) shares the processor with another task.

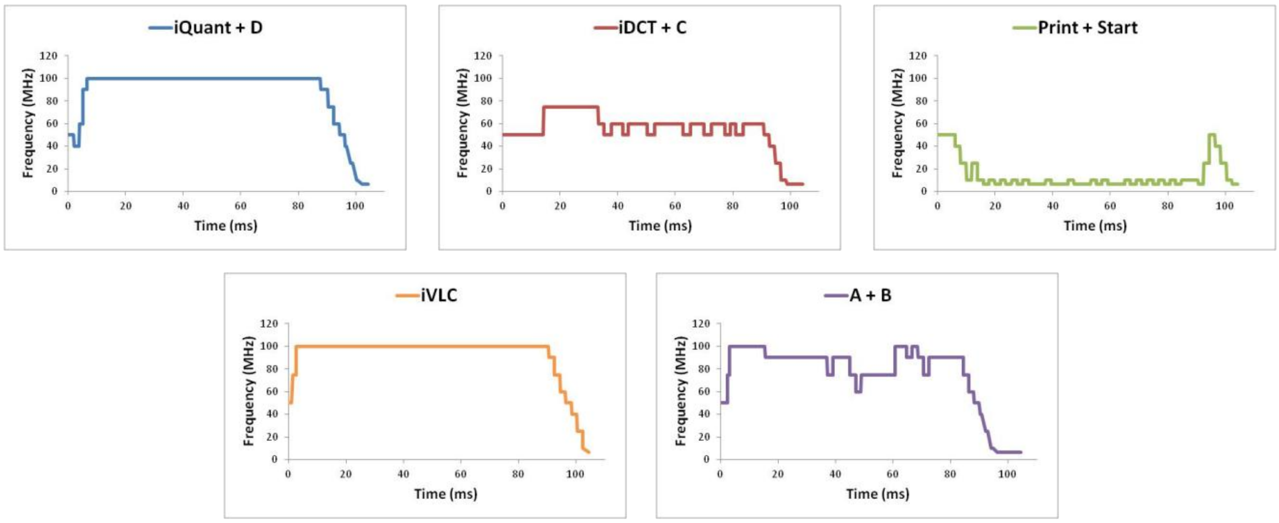


Figure 49 – Frequency behavior of MPEG and Communication applications tasks in the third scenario of multitask execution.

The reductions in the number of executed instructions and execution time overhead are presented in Table 21. In general, the first reactivity level presented better results for this application, compared to the second level. The second reactivity level outperforms the first level only in the first scenario, where the execution time overhead was reduced by 2% more than the first level and the reduction in executed instructions is very close. The other two scenarios present very similar reduction in the number of executed instructions and execution time overhead for both reactivity levels, with a small advantage for the first one. The reduction in the number of executed instructions ranges from 25% to 41%, while the execution time overhead ranges from 8.6% to 12%.

Table 21 – Number of executed instructions and execution time results for the evaluated scenarios of MPEG application in multitask execution.

Scenario	*Scenario*	*Executed Instructions (in millions)*	*Reduction (Instructions)*	*Execution Time (ms)*	*Overhead (Exec. Time)*
1	***Without DFS***	23.03	**-**	85.45	**-**
	With DFS – React. 1	13.56	**41.12%**	95.63	**11.91%**
	With DFS – React. 2	13.58	**41.03%**	91.81	**8.61%**
2	***Without DFS***	22.97	**-**	85.40	**-**
	With DFS – React. 1	13.55	**40.93%**	93.01	**8.91%**
	With DFS – React. 2	13.97	**39.10%**	93.72	**9.74%**
3	***Without DFS***	37.99	**-**	85.54	**-**
	With DFS – React. 1	28.14	**25.93%**	95.80	**11.99%**
	With DFS – React. 2	28.44	**25.14%**	94.75	**10.77%**

Table 22 present the results in terms of power dissipation for the third scenario using the second reactivity level. Although the first level presented better results for this scenario, it was chosen to evaluate the second level due to the small difference of performance in this scenario and the overall performance presented in previous applications. The power reduction in PEs was around 10%, while in the NoC it was around 65%. Similarly to Section 6.2.2, considering only the savings in the CPUs, the reduction in the number of executed instructions is similar to the power savings, however, the small savings in the RAM reduce the average reduction in PEs.

Table 22 – Power dissipation results for the third scenario of MPEG application in multitask execution, with and without the proposed DFS scheme.

	Total Power (mW)				
	RAM	***CPU***	***DFS Controller***	***PEs***	***NoC***
With DFS	75.64	16.90	2.10	96.77	2.42
Without DFS	78.50	22.87	-	107.62	7.02
Reduction	**3.64%**	**26.12%**	**-**	**10.08%**	**65.56%**

The relation between NoC dissipation and PEs dissipation is depicted in Figure 50. Similar to what was presented along Section 6.2, the reduction in NoC power dissipation contributes only with a small percentage to the total power savings. The total power savings for the whole system was 13.48%, being 9.47% from PEs and 4.01% from the NoC.

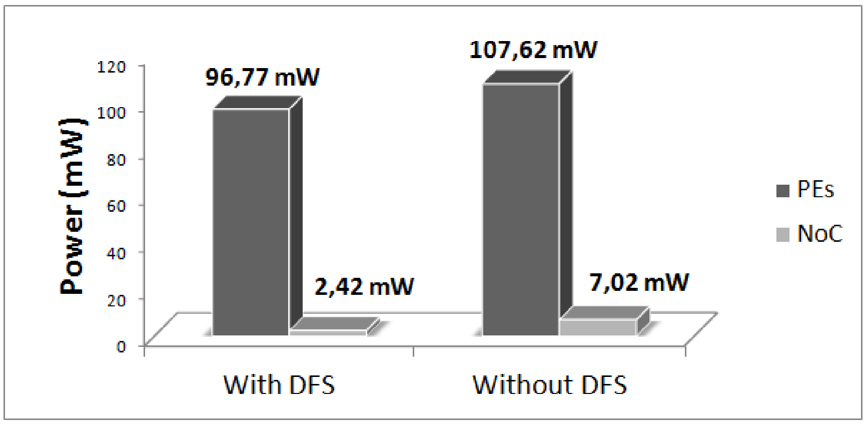


Figure 50 – PEs and NoC power dissipation comparison for MPEG and Communication applications in multitask execution.

6.3.3 VOPD

The multitask execution of the VOPD application is performed using 7 PEs. Tasks with similar computation workload characteristics, e.g. *VOP-Mem* and *StripeM*, are mapped to the same processor. Figure 51 shows the frequency behavior of each processor for the second reactivity level of the controller. Besides the processors executing tasks *VOP-Mem* plus *StripeM* and *IQUANT*, all processors oscillated mostly between 90MHz and 100MHz. This is explained by the computation workload of tasks that are placed in the same processor, which demands full utilization most of the time. The processor executing tasks *VOP-Mem* and *StripeM* had its frequency decreased to the lowest level, since these tasks do not perform any computation. The oscillation in the frequency of the processor executing task *IQUANT* is explained by the fact that it consumes data from task *VOP-Mem*, which shares the respective *pipe* with another task. Thus, in some moments, the *IQUANT* task has to wait until some data is consumed by another task to have the requested message in the *pipe*. Therefore, improvements in the multitask policy can still be performed, being the target of future work.

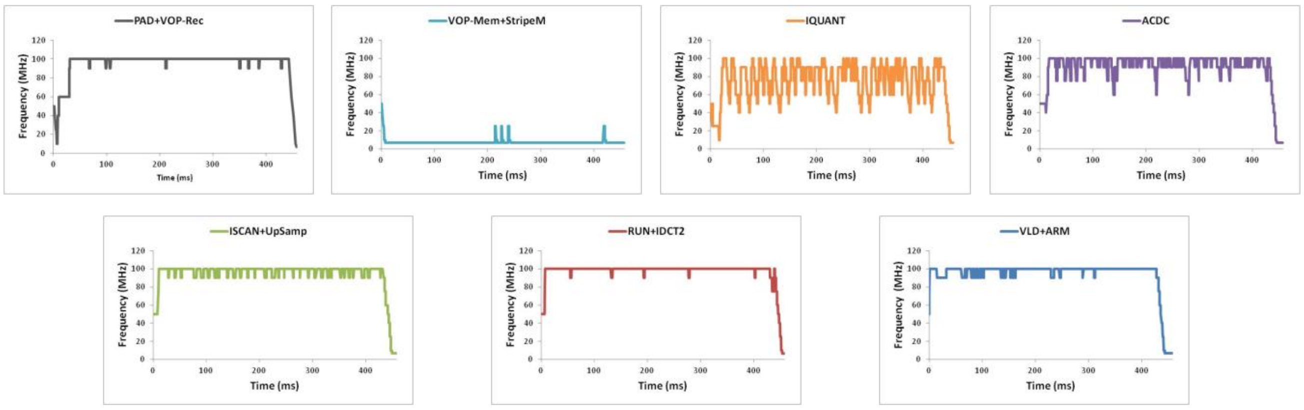


Figure 51 – Frequency behavior of VOPD application tasks in multitask execution.

Table 23 presents the results in terms of reduction in the number of executed instructions and execution time overhead. Since the required time to evaluate the system in terms of power dissipation could reach an entire week, due to the application execution time, this analysis is not done for this application. Also, the first reactivity level outperforms the second one in this application. The reduction in the number of executed instruction is around 18% and the execution time overhead is around 2% for the second level, while it is around 22% and less than 1%, respectively, for the first level. This can be explained by the higher sensibility of the second level with the monitoring parameters variations. As it can be seen in Figure 51, even the processors that operate at the reference frequency oscillate to a lower frequency at some moments. Using the first reactivity level, these oscillations do not appear, making processors operate at the reference frequency during the whole simulation, consequently reducing the execution time overhead.

Table 23 – Number of executed instructions and execution time results for VOPD application in multitask execution.

Scenario	*Executed Instructions (in millions)*	*Reduction (Instructions)*	*Execution Time (ms)*	*Overhead (Exec. Time)*
Without DFS	299.61	-	434.14	-
With DFS – React. 1	233.11	**22.2%**	436.86	**0.6%**
With DFS – React. 2	245.12	**18.2%**	442.70	**2.0%**

6.3.4 MULTI TASK FINAL REMARKS

In most applications, the second DFS controller reactivity level presented lower execution time overhead and reduction in the number of executed instructions than the first one. However, for multi task execution, the difference between the results of both levels is small. The reduction in the number of executed instructions ranges from 25% to 41%, while for the second level it ranges from 18% to 41%. The execution time overhead ranges from 0.63% to 11.99% for the first reactivity level and from 1.48% to 10.77% for the second level. This shows that in multi task execution, changes in computation and communication workload are smooth compared to mono task execution, since two or more tasks share the computation and communication resources. For example, in Section 6.2.6, tasks *PAD* and *VOP-Rec* are allocated in the same PE. In Figure 38 the task *VOP-Rec* drastically changes its computation workload due to the messages becoming available at 50ms, making the controller quickly increase its frequency. However, in multitask execution (Figure 51), task *PAD* produces a large variation, since it is working while task *VOP-Rec* is blocked. Thus, both reactivity levels can be used to control a multi task scenario.

Finally, the NoC power dissipation was reduced by 65.5% in the evaluated scenario, against an average of 70% in mono task execution. This shows that with more flows being sent through the NoC, the reduction becomes lower. Still, the proposed DFS scheme for the NoC produces significant power savings.

7 CONCLUSION AND FUTURE WORKS

The workload variations present in many applications may use a dynamic power management technique to design an energy-efficient device. This Dissertation addresses the reduction of power and energy consumption in NoC-Based MPSoCs through the DFS technique. The contributions comprise three main topics:

- A DFS scheme for PEs of a homogeneous NoC-Based MPSoC;
- A Multitask policy to support multitask execution;
- A DFS scheme for the Hermes NoC.

The proposed DFS scheme for PEs is based on its computation and communication loads. The DFS controller only takes into account local information to choose the suitable frequency, characterizing a totally distributed scheme. The novelty of the solution relies on the parameters used to control the proposed scheme, which are only used individually in most of the reviewed works. Moreover, a clock generation module was designed to provide the clock signal for the PE. This module uses an input clock as reference to generate an output clock with lower or equal frequency, by omitting a programmable number of cycles of the input clock. Also, the controller was designed with two reactivity levels, which directly affect its behavior. The results presented in Section 6.2 shows that the DFS scheme is able to adjust the PE frequency according to the relation between the computation/communication load of its executing task and the remaining tasks of the application. Also, it is shown that reduction in the number of executed instructions is directly related to power dissipation.

Next, a multitask policy is presented to enable the DFS controller to work with a PE executing multiple tasks. Since the target MPSoC supports multitask execution, it is mandatory that the proposed scheme handle this scenario. By executing multiple tasks in the same processor, the concurrency for resources may lead to erroneous interpretation of the monitoring parameters. For example, if a task presents high communication load, it can quickly fill the *pipe*, blocking the writings of a second task, consequently decreasing operating frequency. The proposed policy avoids this situation by controlling the *microkernel* scheduler according to the resources being used by each task. Results in Section 6.3 show that the multitask policy enables the DFS controller to adjust the PE frequency without any hardware modification.

The proposed DFS scheme for the routers of the Hermes NoC is guided by packet information. In the target MPSoC, PEs are responsible for defining the frequency that the packet must be sent through the NoC and code this information in the packet. Then, the router frequency is adjusted according to the packets that are currently being switched. Additionally, the router DFS controller may set the operating frequency to the lowest frequency level in case of inactivity.

Results show a reduction in the number of executed instructions from 3.2% (pipeline with 6 tasks – monotask) to 65.1% (MPEG - monotask execution) and in power dissipation from 13.5% (MPEG-Communication – multitask execution) to 52% (MPEG – monotask execution) considering all the evaluated applications in PEs. The difference in the number of executed instruction is explained by the tasks behavior. When all tasks present similar behavior savings are smaller, since the DFS controller tends to increase the operating frequency of all tasks to the reference frequency, e.g. pipeline with 6 tasks in scenario 1, FOX and JPEG applications. On the other hand, applications composed by tasks with imbalanced data processing rates present higher savings, since the DFS controller can decrease the frequency of tasks with low data processing rate. In average, the reduction in the number of executed instructions was 28% and in power dissipation 23%. In the NoC, power saving was 71% in average, ranging from 65.5% (MPEG-Communication – multitask execution) to 76.9% (MPEG – monotask execution). However, this reduction only represents around 5% of the whole system power dissipation. The power overhead induced by the DFS controller in the PE is around 3%, while in the router it is around 10%. Moreover, the proposed DFS schemes for PEs and NoC have a small impact in the total execution time, leading also to significant energy savings.

7.1 FUTURE WORKS

This work is an initial effort to implement a dynamic power management technique for the HeMPS platform. Improvements of the proposed technique can be addressed as future works. Salehi et. al. [SAL10] present a scheme with variable evaluation period, which present better results than a fixed evaluation period. This is showed also in [ALI09], where the buffer occupation controls the evaluation period, leading to less frequency changes. Thus, with a variable evaluation period, the system is evaluated only when the frequency should be changed, consequently reducing overall switching activity and increasing power savings. Also, the proposed scheme may be modified to work with task's deadlines, throughput, latency and so forth. Currently, if the task is not blocked and the produced data is consumed at equal or higher rate, the proposed scheme tends to increase the operation frequency to the reference frequency. However, tasks can execute at lower frequencies and still meet the required performance, e.g. if a given task deliveries the double of the required throughput working at the reference frequency, the operation frequency can be decreased by half of the reference frequency and the required throughput will be met. Still, deeper study for multitask execution is needed to improve the proposed multitask policy. In this work, the number of *pipe* slots that a task can use in multitask execution is fixed. However, when working with multiple applications, this number may be controlled dynamically, according to each application characteristic.

In addition to improvements in the proposed techniques, deeper evaluations can be addressed as future works. In this work, the Hermes NoC was calibrated for several operation frequencies, using the technique proposed in [GUI08]. Thus, the collected data (Appendix A) can be used by a higher level power estimation tool to provide fast power evaluation. Also, as it was showed, the number of executed instructions is directly related to the dissipated power, and can be also used as a metric for power estimation at higher abstraction levels, e.g. RTL simulation. Still, the proposed scheme may be used in more complex processors (e.g. Leon, ARM) to evaluate power reduction in the CPU, since power dissipation in the present PE (Plasma based) is dominated by the memory. Nevertheless, with more complex processors the NoC may be more explored, and its contribution in total power dissipation and/or energy consumption may change. Finally, a comparison between the proposed technique and a simple clock-gating approach and comparisons with other DFS/DVFS techniques also comprise future works.

REFERENCES

[ALI06] Alimonda, A.; Carta, S.; Acquaviva, A.; Pisano, A. "Non-Linear Feedback Control for Energy Efficient On-Chip Streaming Computation". In: International Symposium on Industrial Embedded Systems (IES), 2006, pp. 1-8.

[ALI09] Alimonda, A.; Carta, S.; Acquaviva, A.; Pisano, A.; Benini, L. "A Feedback-Based Approach to DVFS in Data-Flow Applcations". IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol.28, no.11, pp. 1691-1704, Nov. 2009.

[BEI08] Beigné, E.; Clermidy, S.; Miermont, P. "Dynamic Voltage and Frequency Scaling Architecture for Units Integration within a GALS NoC". In: Second ACM/IEEE International Symposium on Networks-on-Chip (NoCs), 2008.

[BEN00] Benini, L.; Bogliolo, A.; De Micheli, G. "A survey of design techniques for system-level dynamic power management". IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol.8, no.3, pp. 299-316, Jun. 2000.

[BEN98] Benini, L.; De Micheli, G. "Dynamic Power Management: Design Techniques and CAD Tools". Kluwer Academic Publishers, Boston, MA, 1998.

[BEN99] Benini, L.; Bogliolo, A.; Paleologo, G. A.; De Micheli, G. "Policy optimization for dynamic power management" IEEE Transactions On Computer-Aided Design of Integrated Circuits and Systems, vol.18, no.6, pp. 813-833, 1999.

[BUR02] Burd, T.; Brodersen, R. "Energy Efficient Microprocessor Design". Kluwer Academic Publishers, Boston, MA, 2002, 357p.

[CAR09] Carara, E.A.; de Oliveira, R.P.; Calazans, N.L.V.; Moraes, F.G. "HeMPS - a framework for NoC-based MPSoC generation". In: International Symposium on Circuits and Systems (ISCAS), 2009, pp. 1345-1348.

[CHA09] Chabloz, J.-M.; Hemani, A. "A flexible communication scheme for rationally-related clock frequencies". In: International Conference on Computer Design (ICCD), 2009, pp. 109-116.

[CHA11] Chakraborty, K.; Roy, S. "Topologically Homogeneous Power-Performance Heterogeneous Multicore Systems". In: Design, Automation & Test in Europe (DATE), 2011, pp. 125-130.

[CHE00] Chelcea, T.; Nowick, S. "A low latency FIFO for mixed-clock systems". In: Proceedings of IEEE Computer Society Workshop on VLSI, 2000, pp. 119-126.

[FIL09] Filho, S. J. "Estimativa de Desempenho de Software e Consumo de Energia em MPSoCs", MSc Dissertation, FACIN, PUCRS, Brasil, March 2009, 81p.

[FLA02] Flautner, K.; Mudge, T. N. "Vertigo: Automatic performance-setting for Linux". In: USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2002, pp. 105-116.

[FOX11] http://www.caam.rice.edu/~caam520/Topics/ParallelAlgorithms/LinearAlgebra/fox.html. Captured in: August, 2011.

[GAR09] Garg, S.; Marculescu, D.; Marculescu, R.; Ogras, U. "Technology-driven limits on DVFS controllability of multiple voltage-frequency island designs: A system-level perspective". In: 46th ACM/IEEE Design Automation Conference (DAC), 2009, pp. 818-821.

[GAR10] Garg, S.; Marculescu, D.; Marculescu, R. "Custom Feedback control: Enabling truly scalable on-chip power management for MPSoCs." In: ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED), 2010, pp. 425-430.

[GLI09] Gligor, M.; Fournel, N.; Petrot, F. "Adaptive Dynamic Voltage and Frequency Scaling Algorithm for Symmetric Multiprocessor Architecture". In: 12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools (DSD), 2009, pp. 613-616.

[GOO10] Goossens, K.; She, D.; Milutinovic, A.; Molnos, A. "Composable Dynamic Voltage and Frequency Scaling and Power Management for Dataflow Applications". In: 13th Euromicro Conference on Digital System Design, Architectures, Methods and Tools (DSD), 2010, pp. 107-114.

[GUI08] Guindani, G.; Reinbrecht, C.; da Rosa, T.; Calazans, N.; Moraes, F. "NoC Power Estimation at the RTL Abstraction Level." In: IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 2008, pp. 475-478.

[HEB09] Herbert, S.; Marculescu, D. "Variation-aware dynamic voltage/frequency scaling". In: IEEE International Symposium on High Performance Computer Architecture (HPCA), 2009, pp. 301-312.

[KAO11] Kao, Y.; Yang, M.; Artan, N. S.; Chao, H. J. "CNoC: High-Radix Clos Network-on-Chip". IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 30, no.12, pp. 1897-1910, Dec. 2011.

[LEE07] Lee, H. G.;Chang, N.;Ogras, U. Y.;Marculescu, R. "On-chip communication architecture exploration: A quantitative evaluation of point-to-point, bus, and network-on-chip approaches". ACM Transactions on Design Automation of Electronic Systems, vol.12, no.3, pp. 1–20, Aug. 2007.

[LIU09] Liu, S.; Qiu, M. "A Discrete Dynamic Voltage and Frequency Scaling Algorithm Based on Task Graph Unrolling for Multiprocessor System". In: International Conference on Scalable Computing and Communications - International Conference on Embedded Computing (SCALCOM-EMBEDDEDCOM), 2009, pp. 3-8.

[LOR98] Lorch, J.; Smith, A. "Software strategies for portable computer energy management". IEEE Personal Communications, vol.5, no.3, pp. 60–73, Jun. 1998.

[MEI05] Meijer, M.; de Gyvez, J.P.; Otten, R."On-chip digital power supply control for system-on-chip applications," In: Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED), 2005. pp. 311- 314.

[MIS09] Mishra, A.K.; Das, R.; Eachempati, S.; Iyer, R.; Vijaykrishnan, N.; Das, C.R. "A Case for Dynamic Frequency Tuning in on-chip Networks". In: 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-42), 2009, pp. 292-303.

[MOR04] Moraes, F.; Calazans, N.; Mello, A.; Mello, A.; Moller, L.; Ost, L. "HERMES: an Infrastructure for Low Area Overhead Packet-switching Networks on Chip". Integration the VLSI Journal, vol.38, no.1, pp. 69-93, March 2004.

[OGR08] Ogras, U.Y.; Marculescu, R.; Marculescu, D. "Variation-adaptive feedback control for networks-on-chip with multiple clock domains". In: 45th ACM/IEEE Design Automation Conference (DAC), 2008, pp. 614-619.

[OGR09] Ogras, U.Y.; Marculescu, R.; Marculescu, D.; Jung, E. G. "Design and Management of Voltage-Frequency Island Partitioned Networks-on-Chip". IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol.17, no.3, pp. 330-341, March 2009.

[PAN07] Panades, I.; Greiner, A. "Bi-Synchronous FIFO for Synchronous Circuit Communication Well Suited for Network-on-Chip in GALS Architectures". In: International Symposium on Networks-on-Chip (NoCs), 2007, pp. 83-94.

[PON08] Pontes, J.; Moreira, M.; Soares, R.; Calazans, N. “Hermes-GLP: A GALS Network on Chip Router with Power Control Techniques”. In: IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 2008, pp. 347-352.

[POU09] Pourshaghaghi, H.R.; de Gyvez, J.P. "Dynamic voltage scaling based on supply current tracking using fuzzy Logic controller". In: International Conference on Electronics, Circuits, and Systems (ICECS), 2009, pp. 779-782.

[PUS08] Puschini, D.; Clermidy, F.; Benoit, P.; Sassatelli, G.; Torres, L. "Temperature-Aware Distributed Run-Time Optimization on MP-SoC Using Game Theory". In: IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 2008, pp. 375-380.

[PUS09] Puschini, D.; Clermidy, F.; Benoit, P.; Sassatelli, G.; Torres, L. "Adaptive energy-aware latency-constrained DVFS policy for MPSoC". In: IEEE SOC Conference (SOCC), 2009, pp. 89-92.

[RAB03] Rabaey, J. M.; Chandrakasan A.; Nikolic, B. “Digital Integrated Circuits a Design Perspective”. Upper Saddle River: Pearson Education, 2003, 761p.

[RHO09] Rhoads, S. “PLASMA Processor”. Downloaded from:

http://www.opencores.org/?do=project&who=mips, 2009.

[SAL10] Salehi, M. E.; Samadi, M.; Najibi, M.; Afzali-Kusha, A.; Pedram, M.; Fakhraie, S. M. "Dynamic Voltage and Frequency Scheduling for Embedded Processors Considering Power/Performance Tradeoffs". IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol.19, no.10, pp. 1931-1935, Oct. 2011.

[SHA08] Shafik, R. A.; Rosinger, P.; Al-Hashimi, B. M. “MPEG-based Performance Comparison between Network-on-Chip and AMBA MPSoC”. In: Design and Diagnostics of Electronic Circuits and Systems (DDECS), 2008, pp. 1-6.

[SHA09] Shalan, M.; El-Sissy, D. "Online power management using DVFS for RTOS". In: 4th International Design and Test Workshop (IDT), 2009, pp. 1-6.

[SHU10] Shu, L.; Li, X. "Temperature-aware energy minimization technique through dynamic voltage frequency scaling for embedded systems". In: International Conference on Education Technology and Computer (ICETC), 2010, pp. V2-515-V2-519.

[SPA01] Sparsø, J.; Furber, S. “Principles of Asynchronous Circuit Design – A Systems Perspective”. Kluwer Academic Publishers, 2001, 337p.

[TSC07] Tschanz, J. et. al. "Adaptative Frequency and Biasing Techniques for Tolerance to Dynamic Temperature-Voltage Variations and Aging". In: International Solid-State Circuits Conference (ISSCC), 2007, pp. 292-293.

[WOS07] Woszezenki, R. C. “Alocação De Tarefas E Comunicação Entre Tarefas Em MPSoCS”. MSc Dissertation, FACIN, PUCRS, Brasil, March 2007, 121p.

[VAN02] van der Tol, E. B.; T. Jaspers, E. G. "Mapping of MPEG-4 Decoding on a Flexible Architecture Platform". Proceedings of SPIE, vol.4674, pp. 1-13, Jan. 2002.

[YIN09] Yin, A. W.; Guang, L.; Nigussie, E.; Liljeberg, P.; Isoaho, J.; Tenhunen, H. "Architectural Exploration of Per-Core DVFS for Energy-Constrained On-Chip Networks". In: Euromicro Conference on Digital System Design: Architectures, Methods & Tools (DSD), 2009, pp. 141-146.

APPENDIX A – CALIBRATION DATA FOR ROUTER MODULES

The values were obtained using a 65nm standard cell library from ST Microelectronics. The Synopsys Design Compiler tool was used for logic synthesis, Modelsim, from Mentor, was used for functional simulation, and Synopsys PrimeTime, used for power evaluation. The values represent angular and linear coefficients of the equation obtained by the linear regression technique [GUI08].

A.1 Buffer

Table 24 – Angular Coefficients for buffer power equation.

Angular Coefficient (a)		Read Clock (MHz)								
		6.67	10	25	40	50	60	75	90	100
Write Clock (MHz)	6.67	**6.92E-05**	6.66E-05	6.33E-05	5.19E-05	4.92E-05	4.62E-05	4.57E-05	4.44E-05	4.42E-05
	10	0	**6.92E-05**	2.50E-04	2.58E-04	2.60E-04	2.55E-04	2.59E-04	2.62E-04	2.49E-04
	25	0	0	**6.97E-05**	4.85E-04	5.36E-04	5.50E-04	5.61E-04	5.73E-04	5.83E-04
	40	0	0	0	**6.90E-05**	6.70E-04	6.57E-04	6.27E-04	6.22E-04	6.33E-04
	50	0	0	0	0	**6.94E-05**	8.01E-04	8.36E-04	8.02E-04	7.37E-04
	60	0	0	0	0	0	**6.93E-05**	8.73E-04	7.90E-04	7.66E-04
	75	0	0	0	0	0	0	**6.93E-05**	8.29E-04	8.30E-04
	90	0	0	0	0	0	0	0	**6.98E-05**	8.12E-04
	100	0	0	0	0	0	0	0	0	**6.90E-05**

Table 25 – Linear Coefficients for buffer power equation.

Linear Coefficient (b)		Read Clock (MHz)								
		6.67	10	25	40	50	60	75	90	100
Write Clock (MHz)	6.67	**6.32E-02**	6.99E-02	8.65E-02	9.11E-02	9.16E-02	9.37E-02	9.27E-02	9.31E-02	9.41E-02
	10	1.08E-04	**1.18E-01**	8.00E-02	8.62E-02	8.92E-02	9.33E-02	9.32E-02	9.62E-02	9.57E-02
	25	1.10E-04	1.82E-04	**2.35E-01**	7.51E-02	7.15E-02	7.53E-02	7.47E-02	7.60E-02	7.87E-02
	40	1.08E-04	1.81E-04	2.94E-04	**3.53E-01**	7.35E-02	8.06E-02	8.50E-02	8.59E-02	8.68E-02
	50	1.09E-04	1.82E-04	2.94E-04	3.05E-04	**4.71E-01**	8.76E-02	8.27E-02	8.78E-02	9.53E-02
	60	1.17E-04	1.95E-04	3.05E-04	3.18E-04	3.30E-04	**5.89E-01**	1.06E-01	1.16E-01	1.21E-01
	75	1.13E-04	1.84E-04	2.97E-04	3.09E-04	3.16E-04	3.19E-04	**7.07E-01**	1.44E-01	1.46E-01
	90	1.13E-04	1.85E-04	2.97E-04	3.08E-04	3.17E-04	3.29E-04	3.14E-04	**8.24E-01**	1.59E-01
	100	1.08E-04	1.80E-04	2.93E-04	3.04E-04	3.14E-04	3.29E-04	3.14E-04	3.15E-04	**9.43E-01**

A.2 Switch Control

Table 26 – Angular Coefficients for switch control power equation.

Angular Coefficient (a)		Read Clock (MHz)								
		6.67	**10**	**25**	**40**	**50**	**60**	**75**	**90**	**100**
Write Clock (MHz)	**6.67**	**9.7E-06**	2.8E-06	6.4E-06	5.4E-06	7.8E-06	8.4E-06	8.2E-06	7.9E-06	7.8E-06
	10	0	**9.8E-06**	9.9E-06	2.5E-05	3.2E-05	3.3E-05	3.7E-05	4.5E-05	3.7E-05
	25	0	0	**9.9E-06**	2.9E-05	4.5E-05	5.9E-05	6.5E-05	7.3E-05	7.7E-05
	40	0	0	0	**9.9E-06**	5.0E-05	5.4E-05	6.1E-05	6.0E-05	6.7E-05
	50	0	0	0	0	**1.0E-05**	6.6E-05	6.6E-05	6.7E-05	7.1E-05
	60	0	0	0	0	0	**9.8E-06**	7.2E-05	7.1E-05	7.0E-05
	75	0	0	0	0	0	0	**1.0E-05**	7.8E-05	7.5E-05
	90	0	0	0	0	0	0	0	**1.0E-05**	7.2E-05
	100	0	0	0	0	0	0	0	0	**1.0E-05**

Table 27 – Linear Coefficients for switch control power equation.

Linear Coefficient (b)		Read Clock (MHz)								
		6.67	**10**	**25**	**40**	**50**	**60**	**75**	**90**	**100**
Write Clock (MHz)	**6.67**	**9.35E-03**	2.51E-02	4.32E-02	4.71E-02	4.76E-02	4.99E-02	4.89E-02	4.96E-02	5.08E-02
	10	5.32E-05	**1.74E-02**	4.28E-02	4.51E-02	4.63E-02	4.84E-02	4.82E-02	4.93E-02	4.93E-02
	25	5.55E-05	6.45E-05	**3.47E-02**	4.52E-02	4.44E-02	4.53E-02	4.47E-02	4.49E-02	4.63E-02
	40	5.37E-05	6.25E-05	7.74E-05	**5.20E-02**	4.48E-02	4.66E-02	4.63E-02	4.71E-02	4.72E-02
	50	5.54E-05	6.43E-05	7.75E-05	7.36E-05	**6.94E-02**	4.69E-02	4.72E-02	4.75E-02	4.78E-02
	60	6.41E-05	8.04E-05	8.84E-05	8.80E-05	8.76E-05	**8.68E-02**	4.86E-02	4.94E-02	5.09E-02
	75	5.98E-05	6.67E-05	7.98E-05	7.72E-05	7.32E-05	6.94E-05	**1.04E-01**	5.13E-02	5.33E-02
	90	5.97E-05	6.79E-05	8.01E-05	7.72E-05	7.37E-05	8.45E-05	6.38E-05	**1.21E-01**	5.37E-02
	100	5.32E-05	6.20E-05	7.63E-05	7.24E-05	7.10E-05	8.42E-05	6.37E-05	6.36E-05	**1.39E-01**

A.3 CROSSBAR

Table 28 – Angular Coefficients for crossbar power equation.

Angular Coefficient (a)		Read Clock (MHz)								
		6.67	10	25	40	50	60	75	90	100
Write Clock (MHz)	6.67	**2.85E-07**	1.30E-06	1.96E-06	1.32E-06	1.04E-06	8.27E-07	7.32E-07	6.91E-07	6.17E-07
	10	0	**2.69E-07**	2.25E-06	1.98E-06	1.97E-06	1.72E-06	1.58E-06	1.56E-06	1.36E-06
	25	0	0	**3.05E-07**	1.86E-06	2.38E-06	2.34E-06	2.11E-06	2.05E-06	2.01E-06
	40	0	0	0	**3.22E-07**	1.72E-06	1.82E-06	1.97E-06	1.63E-06	1.60E-06
	50	0	0	0	0	**3.37E-07**	1.71E-06	1.72E-06	1.89E-06	2.32E-06
	60	0	0	0	0	0	**3.01E-07**	1.51E-06	1.63E-06	1.64E-06
	75	0	0	0	0	0	0	**2.92E-07**	1.49E-06	1.53E-06
	90	0	0	0	0	0	0	0	**3.12E-07**	1.28E-06
	100	0	0	0	0	0	0	0	0	**3.11E-07**

Table 29 – Linear Coefficients for crossbar power equation.

Linear Coefficient (b)		Read Clock (MHz)								
		6.67	10	25	40	50	60	75	90	100
Write Clock (MHz)	6.67	**1.70E-05**	5.20E-04	9.06E-04	8.35E-04	7.24E-04	6.99E-04	6.14E-04	5.92E-04	5.71E-04
	10	7.60E-07	**1.26E-05**	7.09E-04	6.59E-04	5.66E-04	5.57E-04	5.31E-04	5.11E-04	4.92E-04
	25	7.85E-07	9.92E-07	**2.33E-05**	5.27E-04	4.38E-04	4.11E-04	4.32E-04	4.22E-04	4.19E-04
	40	7.64E-07	9.54E-07	1.19E-06	**2.36E-05**	3.97E-04	3.74E-04	3.35E-04	3.45E-04	3.57E-04
	50	7.49E-07	8.28E-06	1.19E-06	9.24E-06	**2.33E-05**	3.77E-04	3.62E-04	3.73E-04	3.58E-04
	60	8.45E-07	1.21E-06	1.40E-06	1.22E-06	1.73E-06	**2.36E-05**	3.63E-04	3.22E-04	3.32E-04
	75	7.79E-07	9.19E-07	1.02E-05	1.01E-06	1.32E-06	9.47E-07	**2.36E-05**	3.51E-04	3.55E-04
	90	7.64E-07	1.04E-06	1.24E-06	1.01E-06	1.33E-06	1.08E-06	8.85E-07	**2.39E-05**	3.56E-04
	100	7.60E-07	9.47E-07	1.18E-06	9.39E-07	1.18E-06	1.08E-06	8.85E-07	7.98E-07	**2.42E-05**

A.4 DFS CONTROLLER

Since the DFS controller has only one variable parameter. which is the output clock. a simpler table was generated to relate output frequency and power dissipation.

Table 30 – Router DFS Controller power dissipation for different output clocks.

	Output Clock (MHz)								
	6.67	10	25	40	50	60	75	90	100
Power Dissipation (mW)	0.111	0.125	0.160	0.185	0.230	0.245	0.282	0.304	0.331