UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

GEANCARLO ABICH

Extending FreeRTOS to Support Dynamic and Distributed Task Mapping in Multiprocessor Systems

Dissertation presented in partial fulfillment of requirements for degree of Master in Computer Science

Prof. Dr. Ricardo Augusto da Luz Reis
Advisor

Porto Alegre
May 2017

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Abich, Geancarlo
Extending FreeRTOS to Support Dynamic and Distributed Task Mapping in Multiprocessor Systems / Geancarlo Abich - 2017.
85 f.: il.
Advisor: Ricardo Augusto da Luz Reis
Dissertation (master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação. Porto Alegre, BR – RS, 2017.
1. Dynamic Mapping 2. Distributed Mapping 3. Embedded Kernel 4. Multiprocessor Systems 5. Multicore 6. Manycore 7. Modelling 8. Simulation

“*Some will win, some will lose (…)*

Don't stop believin

Hold on to that feelin”

Journey (Jonathan Cain, Steve Perry, Neal Schon)

AGRADECIMENTOS

Gostaria de agradecer a Deus em primeiro lugar por me acompanhar em todos os momentos da vida, principalmente nos difíceis.

A família é a principal estrutura na vida de um grande homem e a minha está sempre ao meu lado. Meus pais que desde o meu primeiro passo neste mundo me incentivam a seguir em frente, apoiam todos os esforços para concretização deste trabalho e deixaram de viver grandes momentos de sua vida para poder me proporcionar o estudo e tudo que tenho hoje. Obrigado pai CARLOS ALBERTO DORNELLES ABICH, vulgo Dornellão, e mãe ELANE ABICH, por me ensinar que com trabalho duro, respeito, dignidade e caráter se atinge qualquer objetivo, seus nomes são a principal citação deste trabalho.

PAULA ANDRESSA FISCHER, tu que permaneceste disposta e atenciosa em todos os momentos, compreendeste todo o tempo que foi dedicado aos estudos e consequentemente o tempo que não estávamos juntos. Às viagens que fiz e não pude te levar junto. Sei que você sempre me acompanha em qualquer caminho que decida traçar. Obrigado por ser tão carinhosa comigo, este é apenas um passo na nossa trajetória de sucesso. Essa conquista também é tua, muito obrigado por tudo.

Caros RICARDO AUGUSTO DA LUZ REIS e LUCIANO COPELLO OST, meus professores e orientadores vocês dignificam ainda mais sua profissão. Obrigado pela compreensão, paciência e pelos puxões de orelha. Talvez uma outra dissertação ou um livro ainda seja pouco para expressar o quanto sou grato. Vocês estão sempre incentivando seus alunos e lutando por eles diante de dificuldades financeiras e pessoais. Não bastasse seu o trabalho árduo, vocês convivem com seus alunos como amigos relatando suas experiências e criando um laço de confiança e amizade. Reis, obrigado pelo exemplo que tu és diante da comunidade brasileira e internacional de pesquisa em computação e microeletrônica. Ost, obrigado por trabalhar duro comigo, se dedicar em me ajudar, por me incentivar todos os dias. É uma honra trabalhar com pessoas como vocês, muito obrigado mesmo.

Agradeço também a todos os demais professores que de alguma forma contribuíram para meu crescimento profissional, em especial ao Leandro Soares Indrusiak, Everton Alceu Carara e ao Altamiro Amadeu Susin que avaliaram e incentivaram a elaboração deste trabalho. À professora Fernanda Gusmão de Lima Kastensmidt e ao professor Sergio Bambi, pelo exemplo que são e pelos conhecimentos passados nas disciplinas. Ao professor Fernando Gehm Moraes pelo apoio e contribuição no desenvolvimento deste trabalho. Ao Marcelo Grandi Mandelli, vai saber, por me ajudar a compreender seus mapeamentos, certo. Ao Rafael Garibotti pelo apoio, incentivo e trabalho em conjunto. Ao Jean Carlo Hamerski e ao professor Alexandre de Morais Amory por acreditar no meu trabalho e pela colaboração na pesquisa. Ao Felipe Rocha da Rosa por sempre me ajudar, tanto nas dúvidas sobre o trabalho e quanto no início do trabalho dentro da UFRGS. Ao Bortolon e ao Vitor pelas diversas caronas e discussões sobre o dia a dia e trabalho. Aos demais membros do laboratório, Alexandra, Calebe, Gracieli, Guilherme, Jucemar, Mateus, Tânia, Tiago, Walter e Ygor, pelo apoio e incentivo em todas as horas.

Por fim, agradeço a todos os demais que de alguma forma contribuíram para este trabalho, tanto os que apoiaram quanto os que dificultaram, pois sempre é um incentivo para seguir em frente. Muito obrigado a todos.

ABSTRACT

Embedded Multiprocessor systems are a reality, in both industry and academia sectors. Such devices offer parallel processing capabilities, aiming at covering the increasing requirements of complex applications. Underlying application workloads are susceptible to variation at runtime, which if not properly handled, may lead to the performance and power efficiency degradation. The continuous increase in the complexity of application workload and the size of emerging multiprocessor systems, calls for dynamic and distributed mapping solutions. The majority of the promoted mapping techniques are bespoke implementations, which consider an in-house operating system developed to a particular processor architecture. This practice restricts its adoption in other platforms, leading to extra design time, re-validation and, consequentially, a hidden cost that may well be quite high. In this scenario, this dissertation proposes a FreeRTOS extension that integrates the support to dynamic and distributed tasks mapping in multiprocessor systems. FreeRTOS is portable to more than 30 embedded processors architectures, increasing software portability and reducing development time. The proposed extension employs mapping techniques allowing FreeRTOS for handle high demands of application mapping in runtime. Another contribution of this work is the development of a framework, which enables the exploration of large systems while providing debugging facilities. The proposed framework provides the automatic generation of multiprocessor platforms, considering parameters of size, processor architecture, and an application set. The resulting platform description is high scalable while allows runtime data extraction and high debugging. These features allowed to validate the proposed FreeRTOS extension in more than one processor architecture from ARM Cortex-M family. Test cases were executed on large-scale platforms and at different levels of abstraction with cases of more than 120 applications incorporating more than 600 tasks processed. The results show that the proposed extension presents better or equal results to the literature.

Index Terms—Dynamic Mapping, Distributed Mapping, Embedded Kernel, Multiprocessor Systems, Multicore, Manycore, Modelling, and Simulation.

Extensão do FreeRTOS para Suporte ao Mapeamento Dinâmico e Distribuído de Tarefas em Sistemas Multiprocessados

RESUMO

Sistemas de Multiprocessados Embarcados são uma realidade, tanto no setor da indústria e quanto no setor acadêmico. Esses dispositivos oferecem capacidades de processamento paralelo objetivando cobrir requisitos cada vez maiores de aplicações complexas. A carga de trabalho subjacente das aplicações é suscetível a variação em tempo de execução o que, se não for tratada adequadamente, pode levar a degradação de eficiência em desempenho e energia. O aumento contínuo da complexidade da carga de trabalho das aplicações, bem como do tamanho dos sistemas multiprocessados emergentes, requer soluções de mapeamento dinâmicas e distribuídas. A maioria das técnicas de mapeamento propostas são implementações personalizadas, considerando um sistema operacional interno desenvolvido para uma arquitetura de processador específica. Essa prática restringe sua aplicação em outras plataformas, levando a um design extra, revalidação e, consequentemente, um custo oculto que pode ser um tanto quanto alto. Neste cenário, esta dissertação propõe a extensão do FreeRTOS para suportar mapeamento dinâmico e distribuído de tarefas em sistemas multiprocessados. O FreeRTOS tem portabilidade para mais de 30 arquiteturas de processadores embarcados, aumentando a portabilidade de software e reduzindo o tempo de desenvolvimento. A extensão proposta utiliza técnicas de mapeamento que permitem ao FreeRTOS atender a altas demandas de mapeamento de aplicações em tempo de execução. Outra contribuição deste trabalho é o desenvolvimento de um framework que permite a exploração de grandes sistemas fornecendo, simultaneamente, resultados para depuração. O framework proposto possibilita a geração automática de plataformas multiprocessadas considerando seu tamanho, a arquitetura do processador e um conjunto de aplicações. A descrição da plataforma resultante é altamente escalável permitindo extração de dados em tempo de execução e alta depuração. Estas características permitiram validar a extensão do FreeRTOS proposta em mais de uma arquitetura de processador da família ARM Cortex-M. Os casos de teste foram executados em plataformas de grande escala e em diferentes níveis de abstração com casos de mais de 120 aplicações incorporando mais de 600 tarefas processadas. Os resultados mostram que a extensão proposta apresenta resultados melhores ou iguais à literatura.

Palavras chave - Mapeamento Dinâmico, Mapeamento Distribuído, Kernel Embarcado, Sistemas Multiprocessados, Multicore, Manycore, Modelagem e Simulação.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

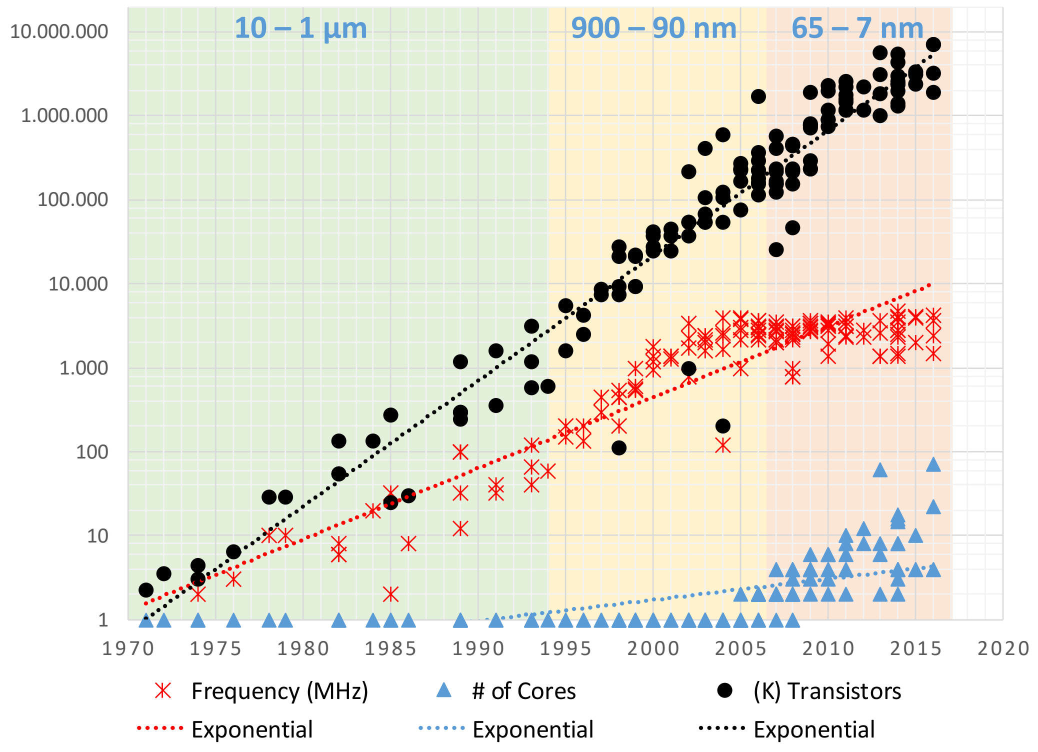
AHB	Advanced High-Performance Bus
AMBA	Advanced Microcontroller Bus Architecture
APB	Advanced Peripheral Bus
API	Application Programing Interface
BSS	Block Started by Symbol
CB	Communication Buffer
CMSIS	Cortex Microcontroller Software Interface Standard
DMA	Direct Memory Access
DPR	Dynamic Partial Reconfiguration
DSP	Digital Signal Processor
DTW	Dynamic Time Wrapping
FIFO	First In First Out
FPGA	Field Programmable Gate Array
GHz	Giga Hertz
GM	Global Manager
GMP	Global Manager Processor
HeMPS	Hermes Multiprocessor System-on-Chip
ID	Identifier
ISA	Instruction Set Architecture
KPN	Kahn Process Network
LEC-DN	Low Energy Communication - Dependencies Neighborhood
LM	Local Manager
LMP	Local Manager Processor
MCSoC	Many-Core System-on-Chip
MHz	Mega Hertz
MIPS	Millions of Instructions Per Second
MPEG	Moving Picture Expert Group
MPI	Message Passing Interface

MPSoC	Multiprocessor System on Chip
MWD	Multi-Window Display
NI	Network Interface
NN	Nearest Neighbor
NoC	Network on Chip
OS	Operating System
OST	OVPsim Simulation Test case
OVP	Open Virtual Platforms
PE	Processor Element
QoS	Quality of Service
RAM	Random Access Memory
ROM	Read Only Memory
RTL	Register Transfer Level
RTOS	Real-Time Operating System
SoC	System-on-Chip
SP	Slave Processor
TLB	Task Location Buffer
TM	Task Manager
TMS	Task Management Structure
UART	Universal Asynchronous Receiver/Transmitter
UML	Unified Modeling Language
VOPD	Video Object Plane Decoder

1 INTRODUCTION

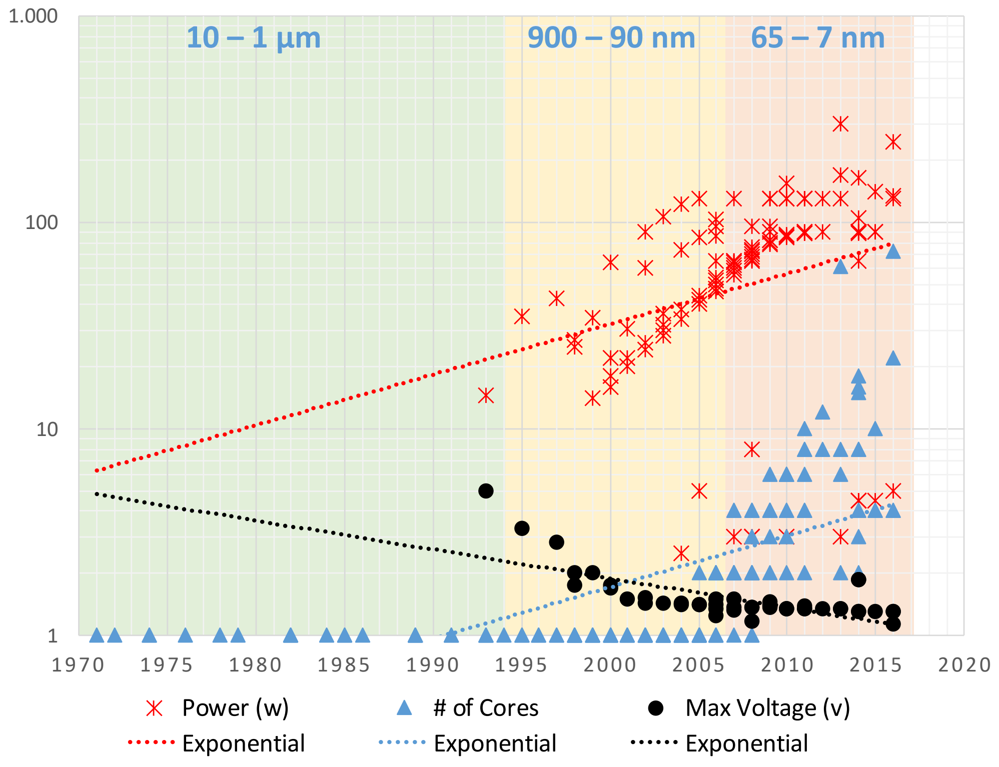
The evolution of the processors is directly related to the performance limitations of the semiconductor manufacturing technology (ITRS 2015). As shown in Figure 1.1, with the reduction of the size of the transistors, the frequency variation of the processors reached an average bound of 4GHz (Figure 1.1). The voltage has reduced significantly, from 5V to 1.25V (Figure 1.2), and the dissipated power has stabilized in 90W (Figure 1.2). The tendency curves in Figures 1.1 and 1.2 show that the number of transistors and the number of cores will continue increasing, although frequency, voltage and dissipated power bounds tend to remain stable. In embedded systems, the power wall is the main challenge, and the average values are even lower for frequency (2GHz) and power (5W).

Figure 1.1. Evolution of processors during the last decades comparing the number of cores with frequency and number of transistors.



Source: Adapted from ITRS 2015.

Figure 1.2. Evolution of processors during the last decades comparing the number of cores with power and voltage.



Source: Adapted from ITRS 2015.

During the last decades, it was evident a change in the embedded processor architectures, from a single core to multicore and manycore processors (BURGIO et al., 2014). The major trend in embedded SoC (System-on-Chip) is the design of MPSoCs (Multiprocessor Systems-on-Chip) to satisfy the ever-increasing computing demands while drawing less battery power (BAKLOUTI et al., 2017). Such systems, increase performance by using multiple, homogeneous or heterogeneous processing elements (PEs, i.e. single or multicore processors).

MPSoCs have been explored to fill the ever-increasing demands of applications and performance while maintaining energy efficiency during the execution of multiple applications (CASTILHOS et al., 2016). While MPSoCs with up to 1000 processors already exists in the industry (BOHNENSTIEHL et al., 2017; MELLANOX 2016), MPSoCs with up to 100 processors have been employed in academia to explore different challenges and novel techniques that may be used to improve the efficiency of underlying systems (BUSSEUIL et al. 2013; MANDELLI et al., 2013). For example, programmability is explored in (GARIBOTTI et al., 2013), task mapping approaches (MANDELLI et al., 2015a; DAS et al., 2016; CASTILHOS et al., 2016), among others (CASTILHOS et al., 2013; MARTINS et al., 2014; MANDELLI et al., 2015b). Challenges, in MPSoCs, are linked to the diversity of application workloads, which demand energy efficiency, performance scalability, and reliability (MANDELLI et al., 2013). Given the vast variety of applications of multicore and multiprocessor systems (e.g., automobiles, smartphones, wearable devices and many other smart gadgets), both hardware and software architecture must provide some degree of flexibility and adaptability. Aiming to scale up the system performance, the workload of an application is divided among multiple threads or tasks (HOLT et al., 2009). The way such tasks are mapped onto the PEs has a significant impact on system performance, energy-efficiency, and reliability (HAGHBAYAN et al., 2016). With large scale platforms, already available in the embedded community, grows the demand for dynamic and distributed mapping techniques, capable of allocating multiple application tasks efficiently.

Multiprocessor system management may include different functions such as monitoring, task mapping, and task migration. The system management may be either centralized or distributed. In centralized management, there is a single processor responsible for monitoring the system, which may be overloaded very quickly. However, distributed management shares the management functions, improving reliability and avoiding hot-spots around a central manager (CASTILHOS et al., 2013). Mapping techniques decisions may drastically influence the system performance, which has been investigated considering different optimization goals such as power consumption, workload distribution, communication, and latency (SINGH et al., 2013). The classification criteria of task-mapping approaches consider the moment at which a task is defined to be executed in a PE, whatever can be either static or dynamic (CARVALHO et al., 2010). While static mappings determine task allocation at design time, dynamic mappings handle task allocation requests at runtime, enabling the system to deal with high demands of applications. Most of such mapping techniques are customized implementations, which are developed based on an in-house OS. Even providing optimized and efficient mapping methods, in-house implementations are usually dependent on a particular processor. With the increasing complexity of the embedded applications, at some point, it will become necessary to port such in-house systems to a better performance processor architecture. The underlying porting process is likely to lead to extra design, re-validation and, consequentially a hidden cost that may be quite high.

Considering the constant shifts in both software and processors architecture, the operating system must address the management complexity of those platforms regularly. The architecture variations of an MPSoC concern the number of processors, the instruction set architecture (ISA) of the processors, the memory organization, and the communication between the resources. Most MPSoCs possess homogeneous architecture, where all cores have the same ISA, access to a shared memory and a bus (HOLMBACKA et al., 2014). Some processors integrate different ISAs aiming to achieve higher performance or DSPs to perform specific tasks such as image and video processing (e.g. big.LITTLE and MALI architectures, ARMTECH 2017). In both cases, shared memory complexity is related to the number of processors competing for access data to and the lack of data coherency (MADALOZZO et al., 2016). Regarding shared-busses, issues such as wire delay, cross-talk noise, and power dissipation, exposes the interconnect technology will be the limiting factor for achieving MPSoCs operational goals (BENINI and DE MICHELI 2002). Creating complex MPSoCs requires a modular, component-based approach to both hardware and software design. Based on these premises, the use of intra-chip networks (NoCs) performs scalability of multiprocessor systems while improving energy efficiency and reliability (MORAES et al., 2004). In NoC-based MPSoCs, each processor is connected to a NoC router, becoming independent, and having distributed memory. The communication through the NoC is abstracted to the user while the operating system manages the communication primitives.

At the software level, the kernel and its integrated Application Programming Interfaces (APIs) are responsible for managing each resource of the platform. The software variations are related to the kernel type, memory management, task scheduling, and task mapping. In general, in-house kernels have APIs for specific applications which support a particular processor architecture, while commercial kernels have several integrated APIs that support multiple processor architectures. Memory management performs memory allocation for system data storage, which can be either static or dynamic. Static memory allocation, the system compilation defines both memory amount and addressing, which is simple but causes high fragmentation and unused memory as it is not deallocated. Dynamic memory allocation improves the scalability of the system according to demand, but the memory must be freed after being used to reduce memory fragmentation and memory leakage. Task scheduling defines the execution time of each task on a PE, which may match real-time priorities (RUARO et al., 2016; BARRY 2011) and allow multitasking execution on a single PE. Further, task mapping defines the distribution of workload between PEs driving the system performance and energy-efficiency.

The foregoing context provides the motivation for this Dissertation, which aims at providing a version of FreeRTOS that supports a set of distributed and dynamic task mapping heuristics. These extensions can be easily employed by almost thirty different processor architectures as well as integrating a set of tools into a framework, allowing early performance analysis of different design alternatives of NoC-based MPSoCs.

This dissertation was possible due to the collaboration between Academia and Industry, where the concepts high validated in academia were applied in a commercial attractive platform. The Universities involved in this project are the Pontifícia Universidade Catolica do Rio Grande do Sul (PUCRS), Universidade Federal do Rio Grande do Sul (UFRGS), Universidade de Brasília (UnB) and University of Leicester, which have been researching in MPSoC related areas over the last years. The work was validated under a commercial tool provided by a partnership with Imperas (IMPERAS 2017) which have interest on the results generated in this work.

1.1 Goals, Objectives, and Contributions

The goal of this work is to provide a version of FreeRTOS that supports a set of distributed and dynamic task mapping heuristics, which can be easily employed by almost thirty different processor architectures. Due to the non-intrusive and flexible implementation, promoted extensions provide an efficient means not only to use and extend available heuristics but also to integrate new ones. The objective of this dissertation includes different targets regarding the acquired knowledge and the resultant work. The objectives are described as follows:

- Research about issues in embedded MPSoCs area, what includes platform and PE architecture, Multiprocessor OS, resources management, task mapping techniques, workload distribution, and communication protocols;
- Provide a robust and reliable commercial attractive MPSoC platform to explore different issues in future research.

The main contributions of this work are the following:

- Extend FreeRTOS to support large scale homogeneous NoC-based MPSoCs;

- Extend FreeRTOS to support NoC communication;
- Extend FreeRTOS to support runtime dynamic and distributed task mapping;
- Development of a framework based on OVPsim unifying software development, debugging capabilities as well as platform configuration and evaluation;
- Extensive FreeRTOS extension evaluation by using several scenarios, including multiple real benchmarks and platform models;
- RTL level validation under different PE architecture.

1.2 Work Structure

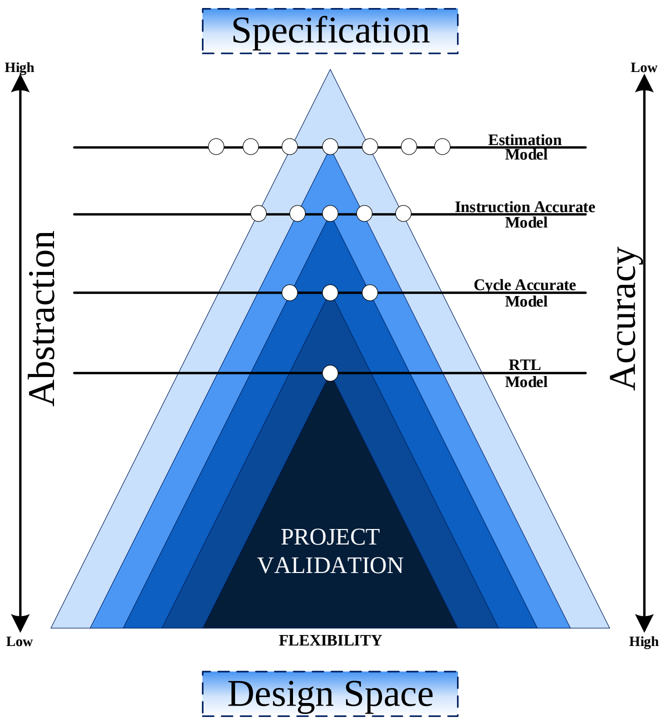
This dissertation exploits different technical aspects, contributions, and faced challenges during the development of the proposed FreeRTOS extension. The dissertation is organized as follows. Chapter 2 presents the state-of-the-art in OS-based task mapping techniques. Chapter 3 explain the developed framework with the platform, system, and application set generation, test case execution, and the data debugging extraction. Chapter 4 introduces the FreeRTOS features, and task mapping algorithms. Then, it presents the proposed extension. Chapter 5 describes the extensive system validation through the different test cases and the results are evaluated. Chapter 5.5 draw this work conclusion and lists the future research to be exploited.

2 STATE-OF-ART

This Chapter discusses the state-of-art related to the two main contributions of this dissertation. First, Section 2.1 presents some works underlying the modeling and validation of mapping techniques, including different modeling approaches and specific design goals. Section 2.2 first introduces about multiprocessor kernels, then, focuses on the state-of-art of kernels which employs task mapping techniques.

2.1 Framework Infrastructures Used to the Exploration of Mapping Techniques for Multiprocessor Systems

Figure 2.1. Project abstraction levels accuracy, possibilities, and validation time.



Source: Adapted from BUTKO et al., 2012.

Researchers have been proposing and validating their mapping techniques considering different approaches and platform architectures. Following this direction, there are different models validated in different abstraction levels which variation is about flexibility, accuracy, developing and execution time. Figure 2.1 shows this variation, where the lowest is the abstraction level, the more accurate is the system validation. However, the developing and execution time are hard constraints to be considered. In addition, high level platforms are flexible (white balls in Figure 2.1) for architectural modeling and parameter calibration based on low level models, improving design and development times.

The literature presents several frameworks developed to validate MPSoC platforms at different abstraction levels, differing regarding accuracy, simulation cost and design flexibility (MANDELLI thesis 2015). Due to the time constraints and time to market, some approaches prefer to use the high abstraction models first, to evaluate and validate the mapping models. Some works rely on simplified high-level abstract models, which are effective means to propose and compare mapping techniques (KANGAS et al., 2006; VIDAL et al., 2010; INDRUSIAK et al., 2010; OST et al., 2011). Even, there are high-level executable models that simulates the system behavioral. These models are instruction accurate (MANDELLI et al., 2015b; WEHNER et al., 2016; MADALOZZO et al., 2015), SystemC PE model with clock accuracy (OST et al., 2009; MANDELLI et al., 2015a; CASTILHOS et al., 2016; WEHNER et al., 2016). However, the real system behavioral validation with high accuracy is made by using RTL level models (CARVALHO and MORAES 2008; SINGH et al., 2010; BUSSEUIL et al., 2011).

High abstraction level modeling simplifies the development and validation of complex MPSoCs to speed up the simulation and to have a high debugging capability (OST et al., 2009). Literature presents different high-level modeling approaches using Unified Modeling Language (UML) to validate their mapping techniques based on model assumptions. The application modeling as Kahn Process Network (KPN) is used to generate application profiles with UML to randomly or manually mapping those applications in a UML-based platform model (KANGAS et al., 2006). Also, the actor-oriented modeling for application design allows to get preliminary performance estimations and to validate the mapping into different abstraction levels (INDRUSIAK et al., 2010). The unified modeling considers distinct steps to improve the design flow, such as application, mapping, and platform. These steps enables fast design space exploration while integrating dynamic mapping heuristics into a unified model of a NoC-based MPSoC (OST et al. 2011). There are co-design methodologies for reconfigurable platforms considering applications, platform, and task mapping to be executed on an

FPGA (VIDAL et al., 2010). Even with the modeling flexibility, most UML-based approaches are limited to static mapping. These models estimate the system behavioral before applying applications to platforms in different abstraction levels, such as FPGA and simulating models. However, they have no accuracy metrics when comparing with both executable virtual and real platforms.

Virtual platforms emulate hardware behavior (e.g. CPU microarchitecture) as full system execution making software validation like it is running on a real physical hardware (MANDELLI et al., 2013). The literature presents some approaches using different accuracy models to improve high performance, high debuggability, and low-cost time in the simulation. Examples of such simulators are Simics (SIMICS 2017), MPTLsim (ZENG et al., 2009), gem5 (BINKERT et al., 2011), and OVPsim (IMPERAS 2017). Those simulators offer a set processor and memory models, which differ regarding simulation time, functionality, and can be instruction (MPTLsim and OVPsim) or quasi-cycle (Simics and gem5) accurate. The OVPsim is an instruction-accurate virtual platform simulator which supports single, multi, and many core bus-based platforms with support to different processor architectures such as ARM, Imagination, Synopsys, Renesas, OpenCores, PowerPC, Altera, and Xilinx (OVP 2017). More recently some works (MANDELLI et al., 2013 and WEHNER et al., 2016) integrated NoC models to OVPsim, allowing to validate their mapping and design space exploration by large scale MPSoCs. The dynamic and distributed task mapping heuristics validated in this simulator platform are based on processors load and NoC communication volume (MANDELLI et al., 2015). There are static mappings to evaluate the performance of real-time scheduling models in different memory organizations (MADALOZZO et al., 2015). Also, integrating a dynamic partial reconfiguration (DPR) interfaces to OVPsim enabling the exchange of simulated IP cores and processor models by connecting the NoC with the reconfigurable regions (WEHNER et al., 2016). There are not many works using OVPsim to task mapping validation and design space exploration of NoC-based multiprocessors. But, all reviewed works are executing their proposed contribution in large scale scalable multiprocessors with low development and simulation time. Further, those proposed models are extensively validated even with metrics based on high-level modeling.

Framework platforms with cycle-accuracy target microarchitecture exploration since specific modeling details as the pipeline implementation of processors, memory models, among others. However, these platforms are not scalable to a large number of processors, specifically when it comes to simulation speed and debugging usability. Even with the increasing number of PEs, the platform distribution into cluster areas improves the workload distribution. The integration of dynamic task mapping to this distributed system management exploring the reclustering and task migration techniques reduces the communication hopes and improves system reliability (CASTILHOS et al., 2013). Furthermore, energy-aware runtime task mapping (MANDELLI et al., 2015) balances the temperature under MPSoC platforms based on the total energy spent (CASTILHOS et al., 2015) monitoring the executed instructions and the NoC communication. Differing to the previous dynamic and distributed mapping approaches, static task mapping may be used to evaluate performance constraints such as real-time oriented task execution due to time response constraints (MADALOZZO et al., 2016). These works use the same SystemC-based platform, which allowed the developers to employ different models such as energy, power, and instructions. However, the processor architecture and simulation are limited to a single architecture.

Frameworks to design space exploration of synthesizable RTL platforms have the lowest abstraction level simulation before the final foundry project. These models simulate the full system behavioral in transactional level with high accuracy results as cycle-accuracy, area, and power. However, the main cost of this kind of platform is simulation time, scalability, and design time. Regarding frameworks to evaluate task mapping heuristics in RTL multiprocessor platforms, in (CARVALHO and MORAES 2008) four congestion-aware dynamic task mapping heuristics based on the Nearest Neighbor (NN) PE were employed to evaluate the NoC congestion, load, and latency. Based on this modeling, (SINGH et al., 2010) executed communication aware dynamic task mappings with clustering on the same platform to evaluate the workload distribution. While these platforms consider dynamic mapping, the Open-Scale framework proposes to explore MPSoC scalability employing static mapping to evaluate the task migration fulfilling its RTOS constraints (BUSSEUIL et al., 2011). In this abstraction level, most frameworks exploit the platform characteristics like power and area. The mapping heuristics applies to little platforms aiming to performance optimizations and communication control. Therefore, these works do not explore platform size, high application workloads, and resources allocation.

The reviewed works were presented according to the abstraction levels illustrated in Figure 2.1. Although, the timeline evidence the use of high-level execution models such as OVPsim are the solution to evaluate large-scale multiprocessor systems. The high abstraction level was UML which perform a high application mapping modeling and validation. The lowest abstraction level performs a high accuracy system validation under an RTL hardware platforms. However, the UML has no accuracy metrics and the RTL demand high simulation time and low scalability. The cycle-accurate platforms validate various concepts around task mapping and monitoring models but are not too scalable regarding processor architectures and large-scale multiprocessors. The OVPsim modeling is an easily scalable and adaptable choice to perform the design space exploration of multiprocessor systems. Further, it can integrate some metrics evaluation as timing and energy models by high-level behavioral modeling based on data extraction from lower abstraction levels.

2.2 State-of-art in OS-based mapping techniques

Given the increasing complexity of embedded applications, mapping techniques are likely to be implemented based on a kernel/OS to deal with time-varying workloads. Multiprocessor kernel approaches or processor architectures are abstracted, which may lead to a gap between what is proposed and its adoption in a real platform. Table I presents the state-of-art in OS based mapping techniques, targeting multiprocessor platforms. These works are ordered by publication year and classified according to different criteria.

(BUSSEUIL et al., 2011) proposed the Open-Scale, a distributed memory NoC-based multiprocessor with SecretBlaze, a MicroBlaze-based processor, as a local processor. Each processor runs an in-house RTOS kernel which supports multi-tasking preemptive scheduler and MPI-like communication primitives. Tasks are statically mapped in design time but can be remapped (i.e. task migration) in runtime depending on the user requirements (e.g. computation time, energy consumption). Finally, Open Scale evaluation considers 6x6 scenarios with three different applications, one for each scenario, running in RTL level.

(MANDELLI et al., 2013) proposed the HeMPS (*Hermes Multiprocessor System on Chip*) multilevel approach, where the simulation can be executed as a virtual platform (OVPsim), on the SystemC-based platform or in RTL level. HeMPS is a distributed memory NoC-based multiprocessor with an MIPS-like (Plasma) as a local processor. It uses a hierarchical/distributed management based on clustered approach which defines system features on design time. Each cluster has a Local Master Processor (LMP) responsible for dynamic mapping and resources management, and the remaining are the Slave PEs responsible for tasks execution. One of the LMPs has an external memory interface called Application Repository which defines the LMP as Global Master Processor (GMP). Each processor executes a different kernel depending on its defined function (i.e. kernel master, kernel slave). Finally, HeMPS evaluation considers 16x16 scenarios with at least ten applications running at each abstraction level.

(MA et al., 2013) proposed the use of a highly portable commercial kernel (µC-OS II) considering each PE as a local cluster with four cores, which communicate through a native message passing interface (similar to MPI). The work is mainly devoted to researching the local task allocation, which is defined based on system workload and resource availability. The drawbacks of this work are: first, the mapping control is centralized and adopted heuristic is not efficient for large-scale systems; second, the evaluation scenario considers a 3x3 NoC-based platform that executes a single matrix multiplication on each PE.

(AGUIAR et al., 2014) proposed a framework for design space exploration of Multiprocessor architectures. The platform can be bus or NoC based with shared memory and supports MIPS and RiscV as local processors. Each processor executes a HellfireOS (AGUIAR et al., 2010) which supports multi-tasking and task migration to remap tasks. The tasks are statically mapped at one PE. A task is migrated in runtime by three steps: initial mapping, characterization and optimized mapping. The evaluation scenarios consider a bus-based platform with 30 nodes of processors and 6x5 NoC-based platform, both with three applications.

The works presented in (AGUIAR et al., 2014) and (BUSSEUIL et al., 2011) are the only ones to support static mapping which is defined at design time. In both works, tasks may be remapped based on a task migration technique, which employs different activities (e.g. context saving and restoring) that are not handled by the task mapping process. Excluding the works proposed by (AGUIAR et al., 2014) and (MA et al., 2013), the mapping control management is distributed, which is scalable since more than one processor is responsible for mapping the tasks.

Beyond the proposed approach, only the work described in (MA et al., 2013) uses a highly portable commercial kernel. The majority of the reviewed works employ RTL platforms described either in VHDL (BUSSEUIL et al., 2011; MA et al., 2013; AGUIAR et al., 2014; MANDELLI et al., 2013) or SystemC (MANDELLI et al., 2013) to promote their mapping techniques. While VHDL-based platforms were validated through small scenarios, for example, 6x6 platform executing three applications (BUSSEUIL et al., 2011), (MANDELLI et al., 2013) employs a 16x16 NoC-based multiprocessor platform with up to 10 applications. The SystemC-based work validates distributed and dynamic mapping heuristics using two kernels: one for Mapper PEs and another to slave PEs. This approach leads to extra design efforts once different kernels need to be developed and maintained.

Table I: State-of-art in multiprocessor kernels with task mapping heuristics.

Author	Kernel	Footprint	Mapping/ Management	Validation Test cases	Processor Architecture	Abstraction Level
Busseuil et al., 2011	In-house	~60KB	Static/ Distributed	6x6 3 applications	SecretBlaze	RTL
Ma et al., 2013	µC-OS II	~24KB	Dynamic/ Centralized	3x3 1 application	>30 architectures	RTL
Aguiar et al., 2014	In-house	~24KB	Static/ Centralized	6x5 3 applications	MIPS & RiscV	RTL
Mandelli et al., 2015	In-house	~25KB	Dynamic/ Distributed	16x16 10 applications	Plasma	Multilevel
This Work	**FreeRTOS**	**~16KB**	**Dynamic/ Distributed**	**20x20 120 applications**	**>30 architectures**	**OVPSim and RTL**

Source: Author.

Multiprocessor embedded kernel and processor architecture are highly abstracted in task mapping literature, requiring a classification considering different systems criteria. In this context, this Dissertation classifies the kernels according to seven criteria: (i) the kernel origin; (ii) the kernel ROM footprint; (iii) the mapping approach; (iv) system management approach; (v) the validation scenario size; (vi) the amount of executed applications; (vii) the supported processor architectures; (viii) the abstraction level. The Table I presents the state-of-art in OS-based mapping techniques targeting multiprocessor platforms. These works are ordered by publication year and classified according to the previous criteria.

The original contribution of this work is the inclusion of dynamic and distributed task mapping techniques in a market leading RTOS kernel, which eliminates system extra design and verification time. Different from the reviewed work, the proposed approach has been validated over different processor architectures (e.g. ARMv6-M and ARMv7-

M) considering several and large scenarios. Further, each PE has the same kernel for Master and Slave.

Therefore, this Dissertation proposes a FreeRTOS extension that differs from literature, and it includes all the following characteristics:

- The development of a Framework to design space exploration and a real operating system validation under different OVPsim NoC-based platform models and sizes.
- Integration of literature known runtime dynamic and distributed task mapping under a commercially attractive kernel (FreeRTOS) and processor models (ARM).
- Multi-architecture system validation (ARMv6-M and ARMv7-M).
- Validated in large scale Multiprocessors (10x10 size) with high workload execution (120 applications and more than 600 tasks).

3 PROPOSED DESIGN FRAMEWORK

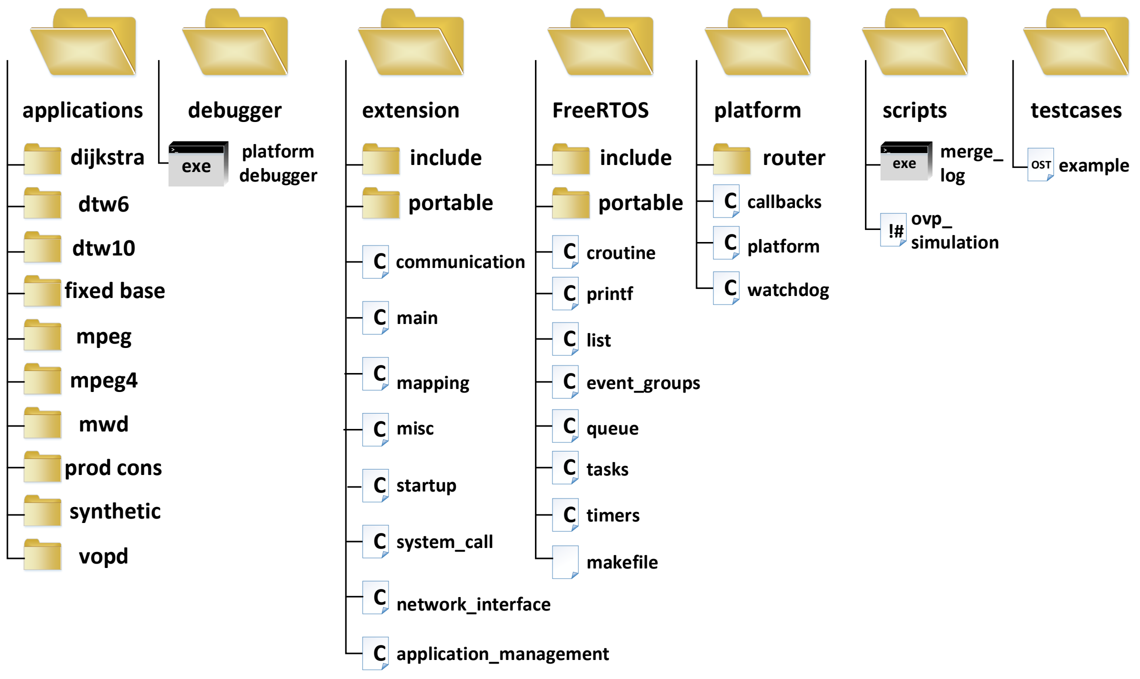
This chapter describes the proposed framework which enables the design space exploration of NoC-based MPSoCs. The framework has three main features: generation of the application set; platform generation; runtime data extraction and debugging. These features are described as follows. Section 3.1 describes the framework infrastructure and the parameters to the configuration of the platform characteristics in the system design and the study cases. Section 3.2 describes the platform architecture and generation. The following subsections present: the OVP PE architecture and memory organization (3.2.1), and the OVP NoC model (3.2.2). Section 3.3 describes the application repository and the application set used to validate the platform and the proposed extension (Chapter 4). Finally, Section 3.4 describes the resulted information metrics generated by the simulation and describes the system evaluation and analysis capabilities considering communication and workload distribution metrics.

3.1 Framework Infrastructure

Multiprocessor platforms have a set of requirements that must be considered to define both the hardware and the software architectures. The resulting range of options calls for frameworks that enable fast and efficient software validation while being easily portable to different processor architectures. Regarding OS validation in multiprocessor systems, most reviewed works propose frameworks to design space exploration of multiprocessor systems (BUSSEUIL et al., 2011; AGUIAR et al., 2014; MANDELLI et al., 2015). Although the OVPsim tool improves fast simulation, just one of these works improves system validation through OVPsim platform models (MANDELLI et al., 2015).

The proposed OVPsim-based framework allows fast simulation and evaluation of MPSoC platforms. Figure 3.1 shows the developed framework source project, where the folder organization tree has directory depth of 4 folders, which means a simple project organization. The *platform* folder contains the OVPsim PE and NoC router description files. *FreeRTOS* and *extension* folders contain the OS source code and the proposed extension respectively. Thus, the FreeRTOS source updates and the developed extensions are easily updatable in the project. *Applications* folder have the ported application set with their task codes, applications profiles, and make files. The *Scripts* folder has the framework bash script, which is used to generate the test case resources based on the *OVPsim Simulation Test case* (OST) description file. The OST description file is located inside the *test cases* folder, and it contains the following information: test case name, platform size, cluster size, task mapping algorithm, NoC buffer size, PE with repository interface, and application set. Finally, the *debugger* folder has the platform debugger tool, which allows evaluating the test case execution results.

Figure 3.1. Project organization with project files and folders overview.

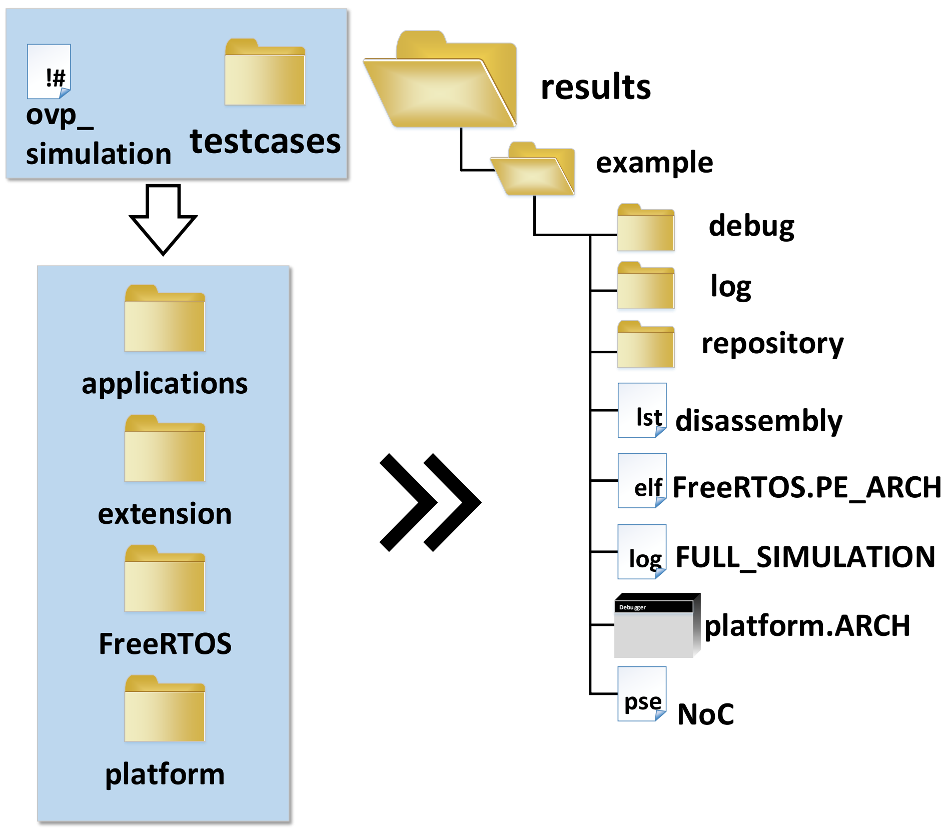


Source: Author.

Once the first test case is generated and executed, the framework creates the *results* folder where the test cases files are stored. For each test case, the framework generates a folder to store all the generated files including OVPsim platform, system binaries, application repository, and execution log.

Figure 3.2 shows an example test case resultant folder. The *debug* folder has the information to feed the platform debugger detailed below. The *log* folder has the system output information for each PE. The *repository* folder stores the repository file with the application set which will be executed in the test case. The remaining files are the platform simulation log, the system disassembly, the platform executable file, the OVPsim NoC model, and the FreeRTOS binary.

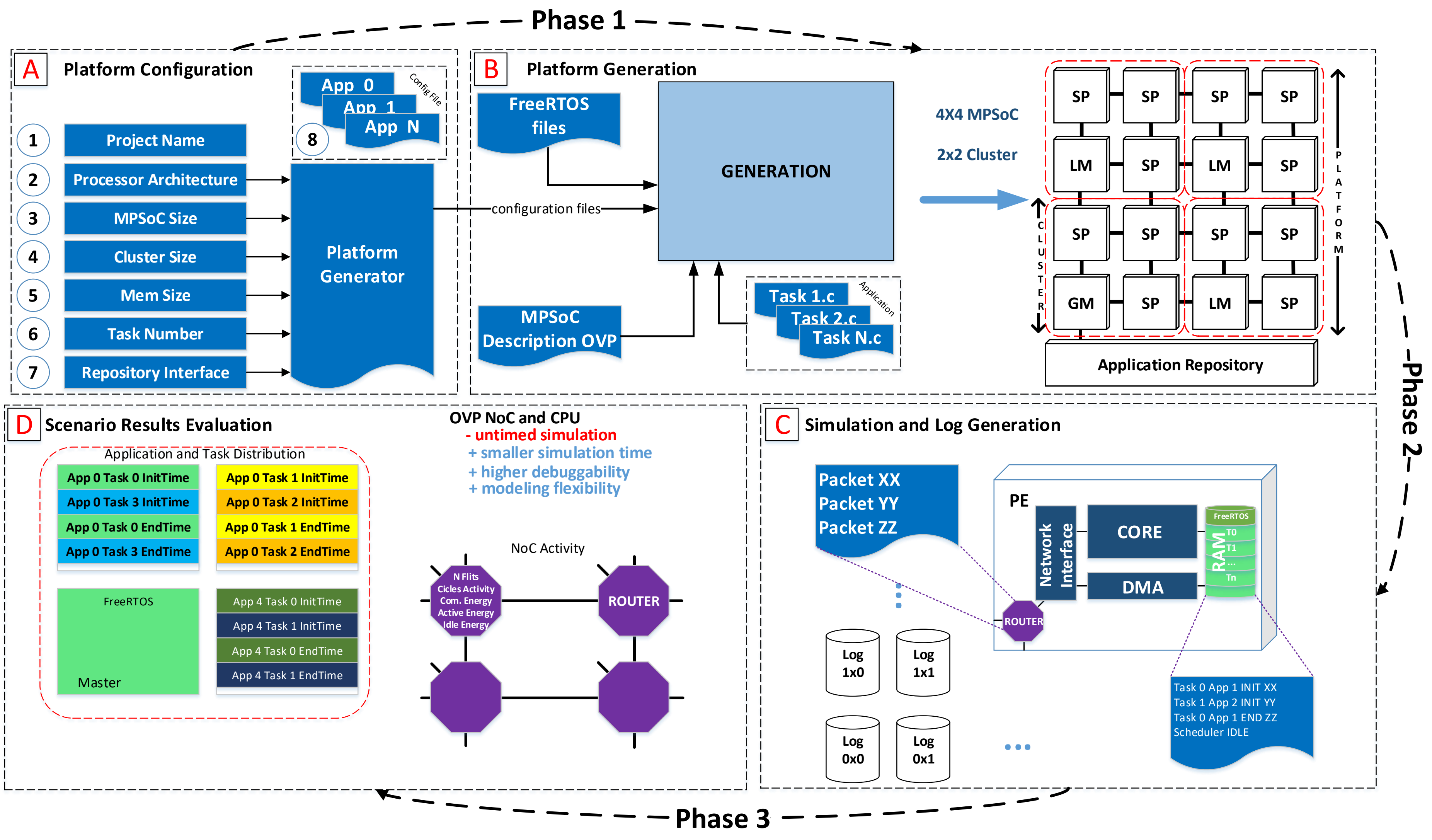
Figure 3.2. Generation and organization of *results* folder.



Source: Author.

The features from OVPsim platform are based on the OST configuration file which defines hardware features such as processor architecture, multiprocessor size, and external repository connection. Also, the FreeRTOS extension features are the number of executing tasks, communication buffer size, task mapping heuristic, and clusters sizes. Finally, the set of applications which the system will map and execute. Consequently, these system features will define the amount of memory needed for each PE.

Figure 3.3. Framework Infrastructure.



Source: Author.

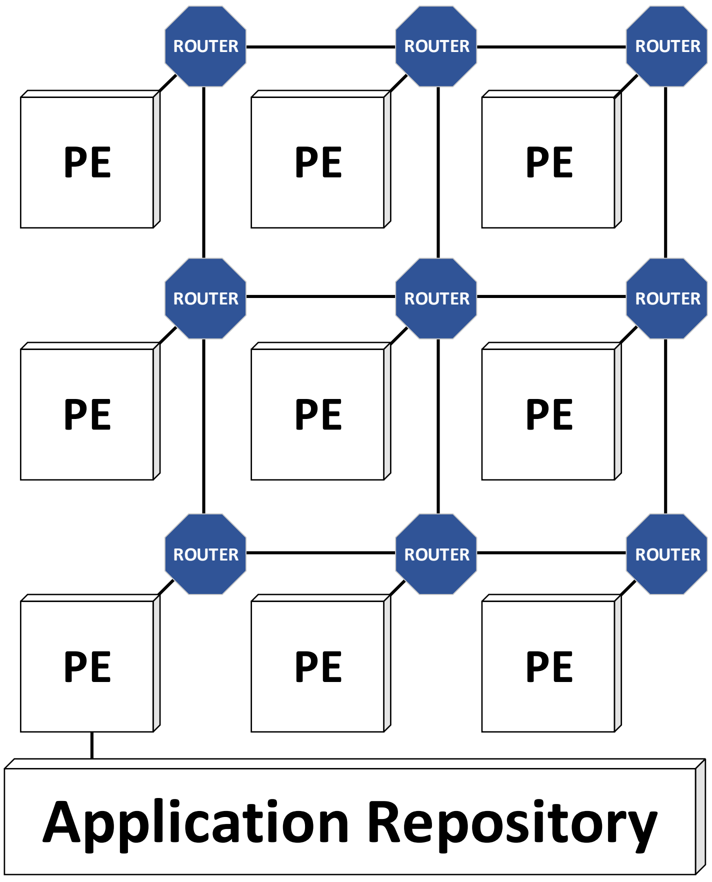
Figure 3.3 shows the framework phases for platform generation, execution and debugging. Once the configuration file has the test case information (Step A), the first phase of the framework generates platform, FreeRTOS and application set. With the OVPsim platform, system binaries, and the repository binaries, the test case environment is done to be executed (Step B). In the next phase the system execution under the platform and the log generation by the PE and NoC router behavioral through the OVPsim models. Each PE loads the system on its memory; the NoC interconnections are made based on platform size and the execution starts (Step C). Finally, the debugging phase analyzing the log files, over the NoC interactions with communication volume and energy, and over the PEs with the workload distribution (Step D).

3.2 Platform Overview

As previously explained, the OVPsim have various processor models available. Most of the reviewed frameworks use up to two processor architectures which have very limited and simple instruction set (e.g. MicroBlaze and MIPS). This work proposes to use a NoC-based homogeneous multiprocessor based on different commercial processor architectures. The framework allows to select the processor architecture and generate large scale NoC-based multiprocessor systems automatically. Then, the resultant OVPsim platforms are high scalable, easily defined and improve a fast system validation by using hundreds of PEs.

Figure 3.4 shows a 3x3 NoC-based multiprocessor platform, including routers, PEs and an external memory connection (application repository). The bottom left PE is the only processor that has access to the application repository. The following subsections detail the PE and NoC architectures.

Figure 3.4. 3x3 NoC-based multiprocessor platform example.

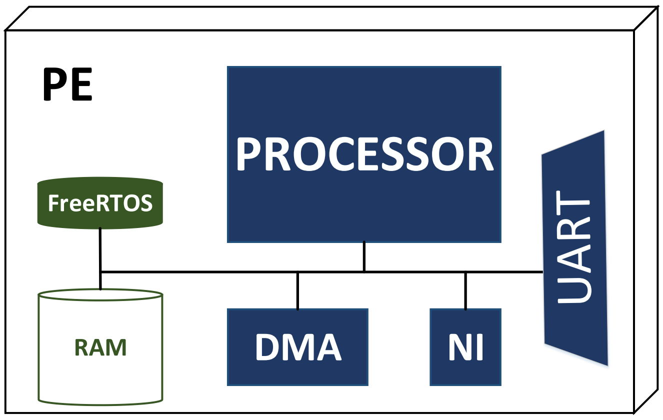


Source: Author.

3.2.1 Processor Elements

The adopted PE model has three main components: the processor model, the memory, and the network interface (NI). Eventually, one can use a direct memory access (DMA) module by implementing OVPsim register bank callbacks to reduce software development and to optimize the sending of data from memory to NI. Also, it is possible to use a universal asynchronous receiver/transmitter (UART). These components are connected by a local bus, and Figure 3.5 shows the PE organization.

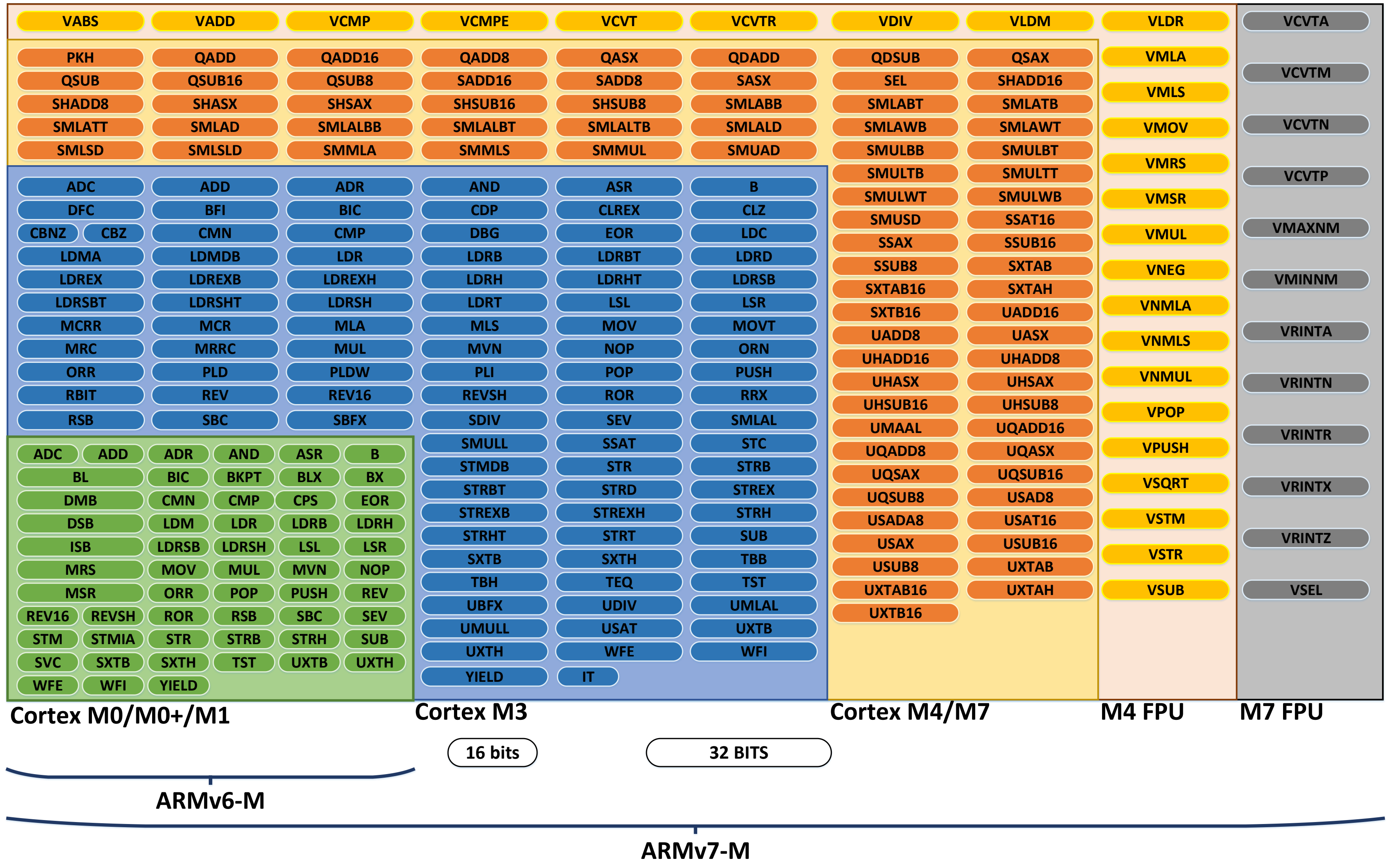
Figure 3.5. PE Architecture.



Source: Author.

One of the contributions of this work is the capability of generating and validating multiprocessor platforms with different processor architectures. In addition, to the literature and aiming to create a commercially attractive platform this work uses the ARM Cortex-M processors family (i.e. ARMv6-M and ARMv7-M) available in OVPsim tool. This processor family was selected due to following features: (i) FreeRTOS supporting, (ii) high use in the industry, (iii) availability of Cortex-M0 processor RTL description. These features does the Cortex-M family a good choice once the project can be validated with different processor architectures in both high level and RTL level simulation. Also, this family brings with the Cortex Microcontroller Software Interface Standard (CMSIS) which improves the software productivity and portability. The ARM CMSIS is a vendor-independent hardware abstraction layer for the Cortex-M processor series and defines generic tool interfaces (ARMDEV 2017). Each processor model variant has different characteristics and capabilities, but the system portability is simple once the FreeRTOS already supports more than thirty (30) processor architectures, including that family. To exemplify the processor architecture variation, Figure 3.6 shows the instruction set for each ARM Cortex-M processor model.

Figure 3.6. ARM Cortex-M Family ISAs.


VABS VADD VCMP VCMPE VCVT VCVTR VDIV VLDM VLDR VCVTA
PKH QADD QADD16 QADD8 QASX QDADD QDSUB QSAX
QSUB QSUB16 QSUB8 SADD16 SADD8 SASX SEL SHADD16
SHADD8 SHASX SHSAX SHSUB16 SHSUB8 SMLABB SMLABT SMLATB
SMLATT SMLAD SMLALBB SMLALBT SMLALTB SMLALD SMLAWB SMLAWT
SMLSD SMLSLD SMMLA SMMLS SMMUL SMUAD SMULBB SMULBT
SMULTB SMULTT
SMULWT SMULWB
SMUSD SSAT16
SSAX SSUB16
SSUB8 SXTAB
SXTAB16 SXTAH
SXTB16 UADD16
UADD8 UASX
UHADD16 UHADD8
UHASX UHSAX
UHSUB16 UHSUB8
UMAAL UQADD16
UQADD8 UQASX
UQSAX UQSUB16
UQSUB8 USAD8
USADA8 USAT16
USAX USUB16
USUB8 UXTAB
UXTAB16 UXTAH
UXTB16
ADC ADD ADR AND ASR B
DFC BFI BIC CDP CLREX CLZ
CBNZ CBZ CMN CMP DBG EOR LDC
LDMA LDMDB LDR LDRB LDRBT LDRD
LDREX LDREXB LDREXH LDRH LDRHT LDRSB
LDRSBT LDRSHT LDRSH LDRT LSL LSR
MCRR MCR MLA MLS MOV MOVT
MRC MRRC MUL MVN NOP ORN
ORR PLD PLDW PLI POP PUSH
RBIT REV REV16 REVSH ROR RRX
RSB SBC SBFX SDIV SEV SMLAL
SMULL SSAT STC
STMDB STR STRB
STRBT STRD STREX
STREXB STREXH STRH
STRHT STRT SUB
SXTB SXTH TBB
TBH TEQ TST
UBFX UDIV UMLAL
UMULL USAT UXTB
UXTH WFE WFI
YIELD IT
ADC ADD ADR AND ASR B
BL BIC BKPT BLX BX
DMB CMN CMP CPS EOR
DSB LDM LDR LDRB LDRH
ISB LDRSB LDRSH LSL LSR
MRS MOV MUL MVN NOP
MSR ORR POP PUSH REV
REV16 REVSH ROR RSB SBC SEV
STM STMIA STR STRB STRH SUB
SVC SXTB SXTH TST UXTB UXTH
WFE WFI YIELD
VMLA VMLS VMOV VMRS VMSR VMUL VNEG VNMLA VNMLS VNMUL VPOP VPUSH VSQRT VSTM VSTR VSUB
VCVTM VCVTN VCVTP VMAXNM VMINNM VRINTA VRINTN VRINTR VRINTX VRINTZ VSEL
Cortex M0/M0+/M1
Cortex M3
Cortex M4/M7
M4 FPU
M7 FPU
16 bits
32 BITS
ARMv6-M
ARMv7-M

Source: Adapted from ARMDEV 2017.

Further, memory size and management are important issues in embedded systems due to their impact on power consumption and area. In the proposed platform, each PE has its private random access memory (RAM) addressing to store the system and applications data. The memory size and its address are defined at design time. The Code/Flash memory address stores FreeRTOS kernel and SRAM address stores the remaining data and tasks. As previously mentioned, the FreeRTOS kernel with the proposed extension has about 16KB size. Thus, this is the minimal size of the required flash memory. However, the minimal SRAM depends on the number of tasks executed in each PE, in general, 16KB is enough to execute multitasking (i.e., two tasks per PE). Although this work uses a small memory, Table II shows the Cortex-M memory model which supports up to 4 GB of memory addressing.

Table II: Cortex-M memory model.

Address Definition	**From**	**To**	**Range**
Code/Flash	0x00000000	0x1FFFFFFF	0.5 GB
SRAM	0x20000000	0x3FFFFFFF	0.5 GB
Peripheral	0x40000000	0x5FFFFFFF	0.5 GB
External RAM	0x60000000	0x9FFFFFFF	1 GB
External Device	0xA0000000	0xDFFFFFFF	0.5 GB
Private Peripheral bus	0xE0000000	0xE00FFFFF	1 MB
Vendor-Specific Memory	0xE01FFFFF	0xFFFFFFFF	511 MB

Source: Adapted from ARMDEV 2017.

The PE local bus controls the on-chip peripheral communication and memory addressing. ARM Cortex-M family processors implement the Advanced Microcontroller Bus Architecture (AMBA) which has the internal Advanced High-performance Bus (AHB) and Advanced Peripheral Bus (APB). Normally the communication peripherals are connected in the APB base system addressing (0x40000000), but aiming to make a trusty PE design reducing the latency by peripheral concurrency the NI is connected in the AHB base system addressing (0x40010000). The NI and DMA modules are made by the OVP tool register callback functions which are executed on every read or write access to a defined address area. Also, the system uses the OVP callbacks to define the end of simulations. In this case, the OVP tool handles these defined addresses and calls functions responsible for executing the operations of the modules.

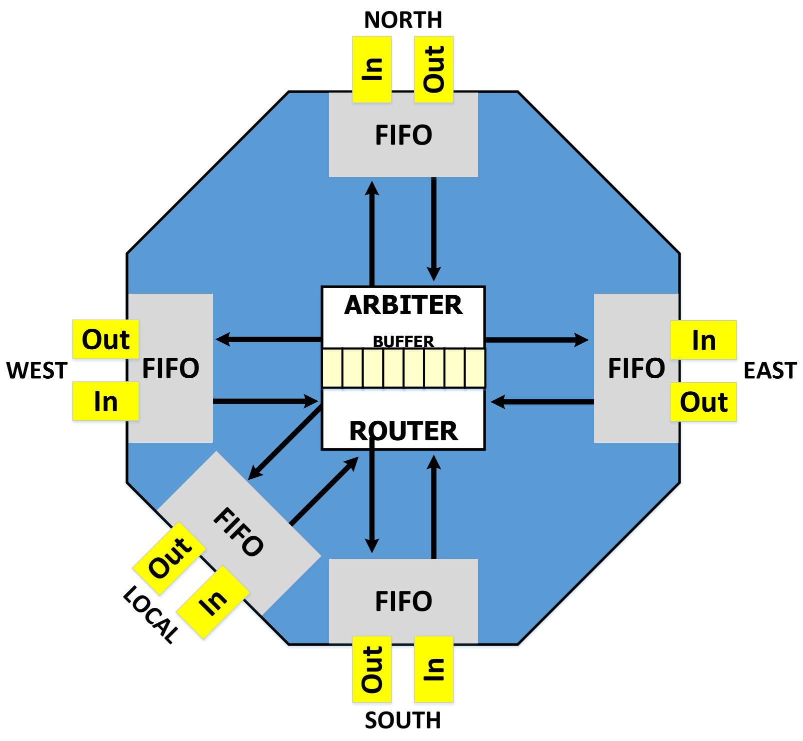
The NI is responsible for sending and receiving messages from/to the NoC. To send messages, the OS creates the message and add it into the NI buffer. The header of this message has two flits size, and each flit is 32 bits wide. The header information is NoC router address (0xA00XXYY0) and the payload size (header size plus payload). If the DMA module is defined, it uses the payload size and its start memory address to feed the NI buffer. Else, it can be software made by feeding the NI buffer flit by flit. When a message comes from the NoC, the NI performs an interruption to the processor, and the system interruption handler treats the incoming message.

3.2.2 NoC Model

The OVPsim tool enables to generate bus-based and NoC-based multiprocessor platform models. This work provides inter-PE communication by the integration of a NoC model developed with OVPsim ppm and bhm APIs (MANDELLI et al., 2015). Figure 3.7 shows the NoC router organization that has five bidirectional ports (input and output data ports), input buffers, and arbiter modules. The local port establishes communication between a router and a PE, and the remaining ports are used to connect a router to its neighbors. The framework generates all router connections which are implemented by using OVP Net ports.

Packets are sent through the NoC divided by flits with 32 bit wide. The arriving flits in an input port trigger a callback arbiter function. The first flit contains the destination NoC address then the callback executes the XY NoC routing algorithm. The routing algorithm selects an output port to send the incoming flit, storing the selected port in the routing FIFO. The following flit has the payload size which defines the buffer allocation to store the message flits while all flits are sent through the selected output port. This OVP NoC module allows the buffers dynamic allocation and deallocation but preserving the wormhole packet switching mode. Also, each output port has an arbiter which adopts a round-robin algorithm to select an input port routing FIFO which has pending packets to its output port. Finally, when the destination router receives an incoming packet, the local port triggers the NI callback which notifies the PE by generating an interruption.

Figure 3.7. NoC router architecture.



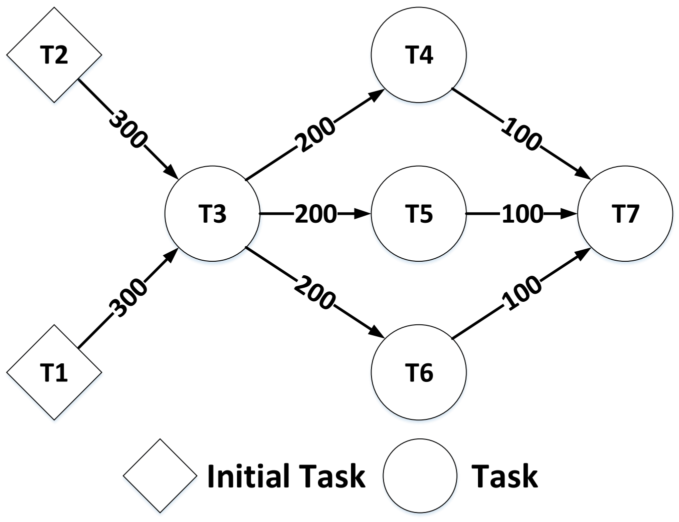
Source: Adapted from MANDELLI et al., 2015.

3.3 Application Set

The system and platform validation was made by executing large scale test cases with different applications with multiple communicating tasks. Each application executes different algorithms based on real application models such as image and video processing, shortest graph path, and temporal sequences, or also synthetic applications just to provide inter-task communication. Applications are modeled as acyclic directed graphs as shown in Figure 3.8, where each node represents an application task, and each directed weighted edge represents a communication dependence. The polygons represent the initial tasks, and the circles represent the remaining tasks of one application.

Each application has one profile with their information about the total number of tasks, the number and ID of initial tasks, tasks interdependency and its inter-task communication volume. Then, the application graph is modeled as *GApp = (T, E)*, where tasks $T_i \in T$ and edge $E_{ij} \in E$ represents the communication between T_i and T_j. The initial tasks initialize the application execution, as the nodes *T1* and *T2* in the figure below, and the remaining are the non-initial tasks.

Figure 3.8. Synthetic application graph.



Source: Author.

With the applications profile the framework generates the application set defined in the configuration file and make the application repository. The application set can have more than one instance of one application type then the repository information is organized as follows:

- Total number of Application;
- Application types;
- Application addresses;
- Application Header;
- Task Header;
- Application Task Codes.

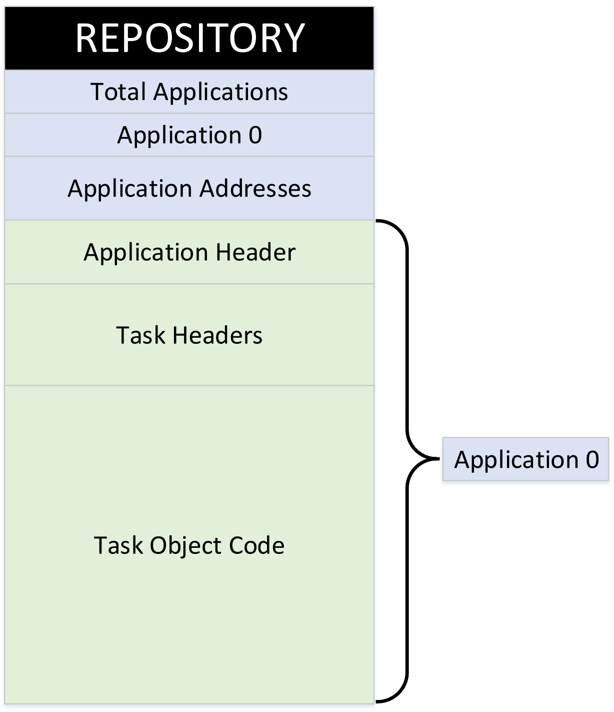
The application repository has all the test case selected applications information divided by the application type to avoid redundant information. Each application type has one application header which has the information about the number of tasks, the application code size, and the initial tasks. Each task header has the information about the task ID, task code size, task uninitialized data size (BSS - *Block Started by Symbol*), task start address, and task dependencies. Figure 3.9 shows an example of the resultant repository generation from a set of applications with its header and the header of one task; the following information was suppressed.

Figure 3.9. Repository example.

```
#ifndef __REPOSITORY_H__
#define __REPOSITORY_H__

#define NUMBER_OF_APPS 1
#define application 0
unsigned int appstype[] = {application};
unsigned int apps_addresses[] = { 0x00000000 };
unsigned int repository[] = {
            0x00000006,     //application id 0
            0x00000C31,     //application size
            0x00000006,     //initial tasks
            0xffffffff,
            0xffffffff,
            0xffffffff,
            0xffffffff,
            0x00000000,     //task_0
            0x000001C4,     //task size
            0x00001054,     //bss size
            0x000000A7,     //task initial address
            0x00000006,     // dependences
            0x00000603,
            0xffffffff,
            0xffffffff,
            ...
            };

#endif  /*__REPOSITORY_H__*/
```

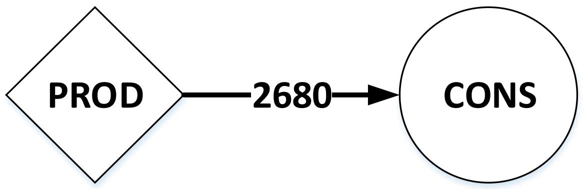


Source: Author.

The application types used to validate the proposed framework and extension are from the available HeMPS framework (HEMPS 2017) and are described below:

- Prod-Cons (*Producer and Consumer*): a simple two (2) task applications with parametrizable communication workload. Figure 3.10 shows the application task graph.

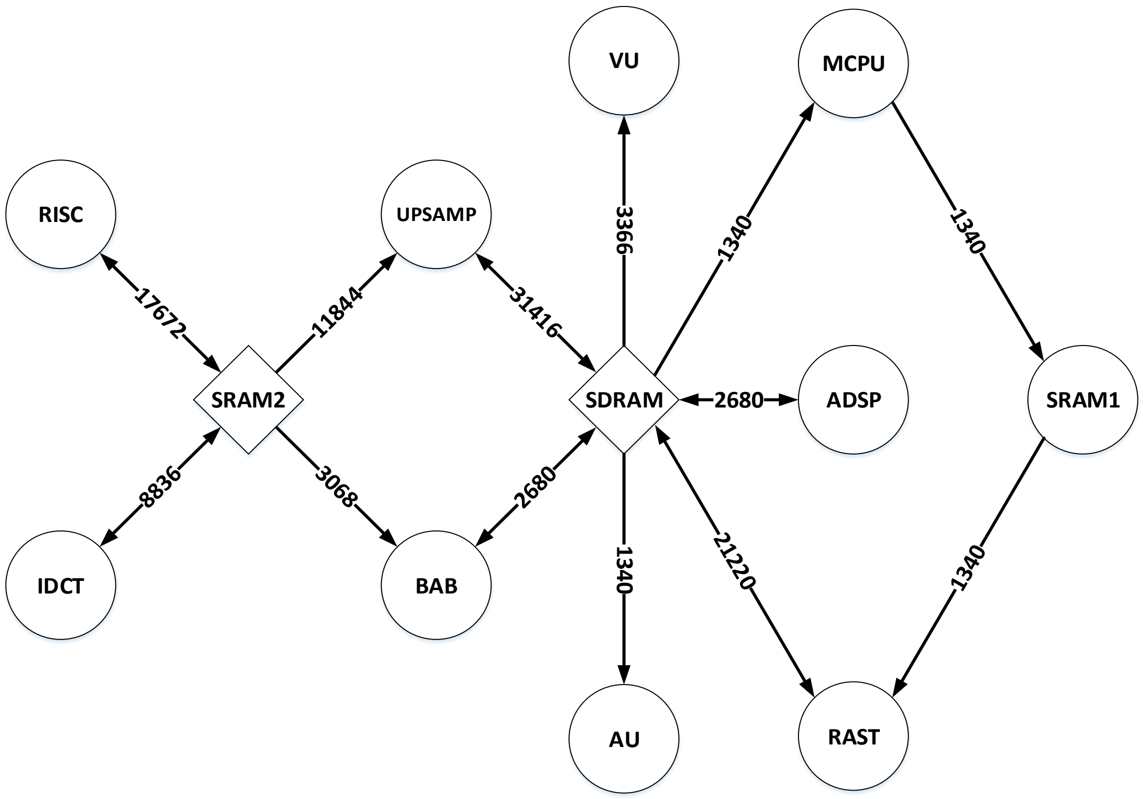
Figure 3.10. Prod-Cons application graph.



Source: Author.

- MPEG4: simulates a full MPEG digital sound and video decoder data iteration with twelve (12) application tasks. Tasks have high communication as shown in Figure 3.11.

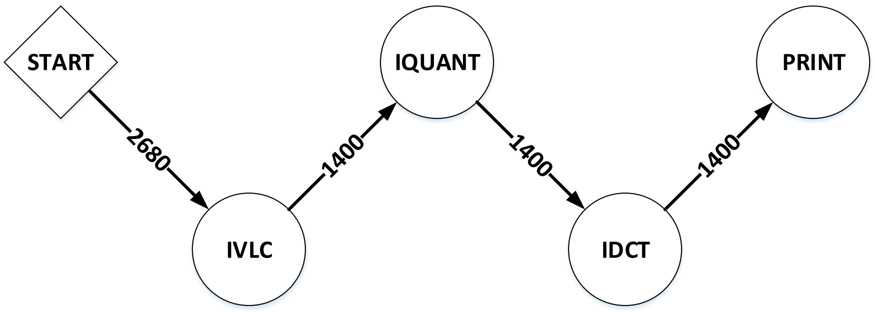
Figure 3.11. MPEG4 application graph.



Source: Author.

- MPEG (*Moving Picture Experts Group*): a partial digital sound and video data decoder with five (5) communication tasks with a high workload, high computational effort, and memory allocation (about 2KB in some tasks). Figure 3.12 shows the application graph with the task communication ordering.

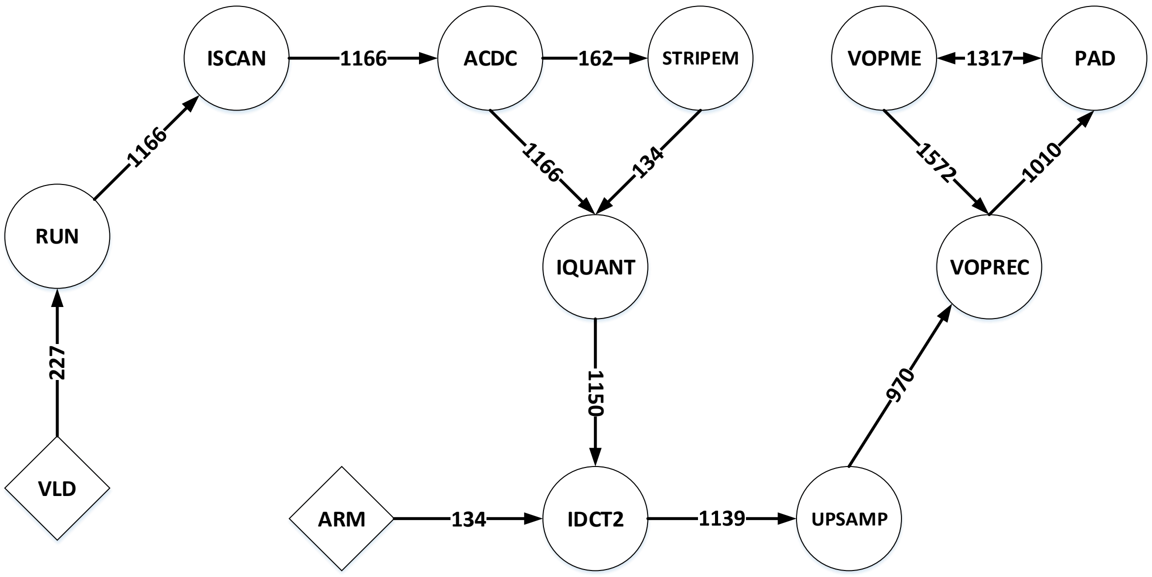
Figure 3.12. MPEG application graph.



Source: Author.

- VOPD (*Video Object Plane Decoder*): simulates the interactions between the hardware modules of a video decoder. Figure 3.13 shows the application graph which has twelve (12) application tasks with high communication workload.

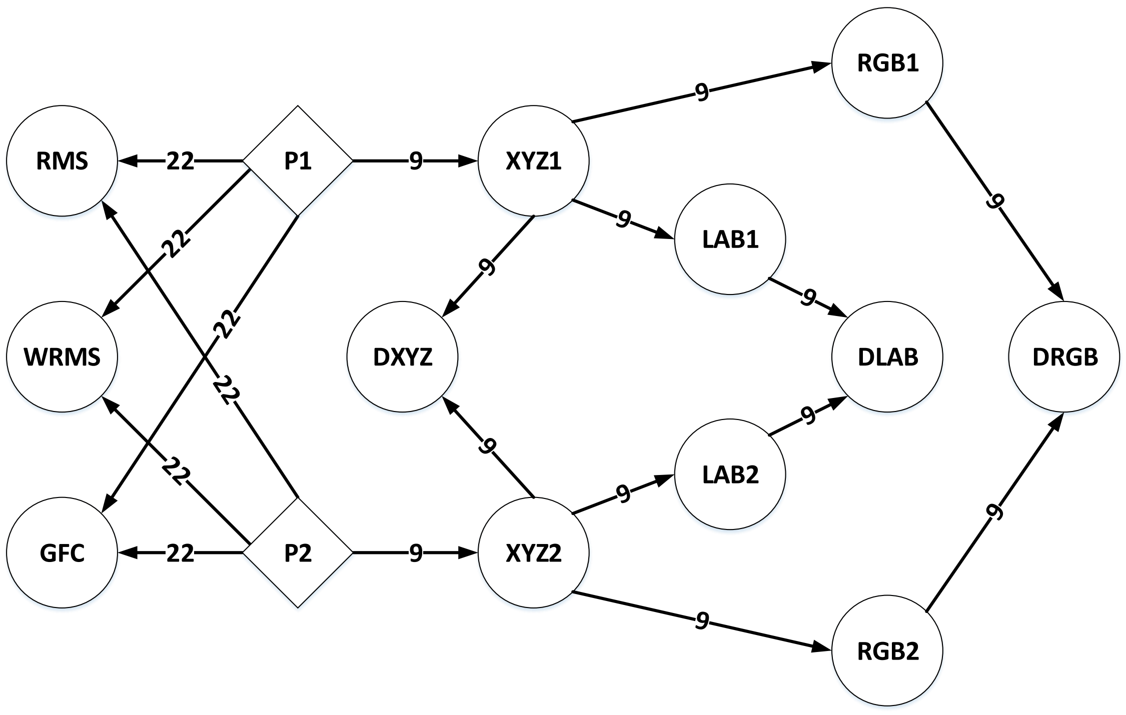
Figure 3.13. VOPD application graph.



Source: Author.

- Fixed-Base Test: compares the similarity of two images. This application has fourteen (14) application tasks with low communication workload but high computational effort. Figure 3.14 shows the application graph.

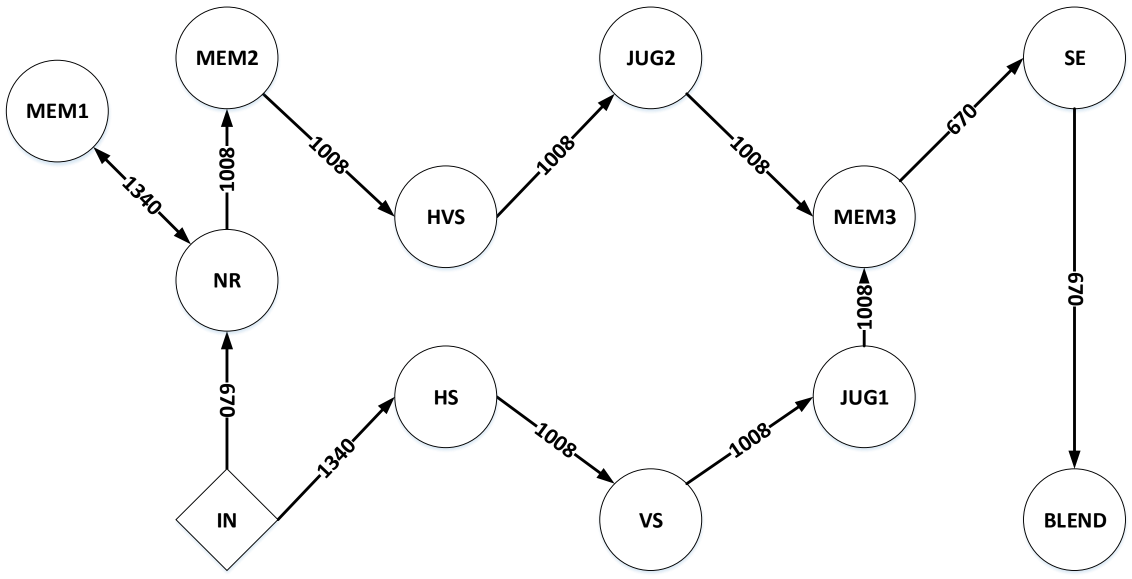
Figure 3.14. Fixed-base Test application graph.



Source: Author.

- MWD (*Multi-Window Display*): a multi-window application control with twelve (12) application tasks with high communication workload. Figure 3.15 shows the application task graph.

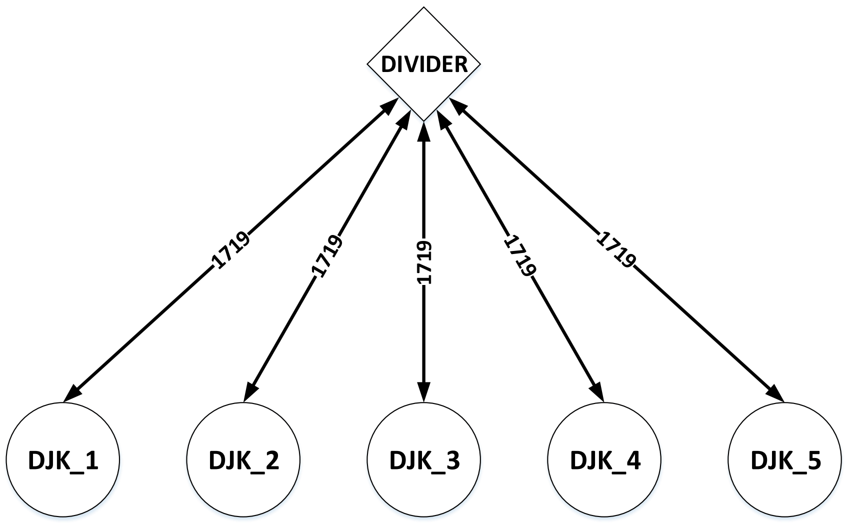
Figure 3.15. MWD application graph.



Source: Author.

- Dijkstra: an algorithm for finding the shortest paths between nodes in a graph. This application has six (6) tasks with high memory usage (about 4KB in some tasks), high computation cost and high communication workload. Figure 3.16 shows the application graph.

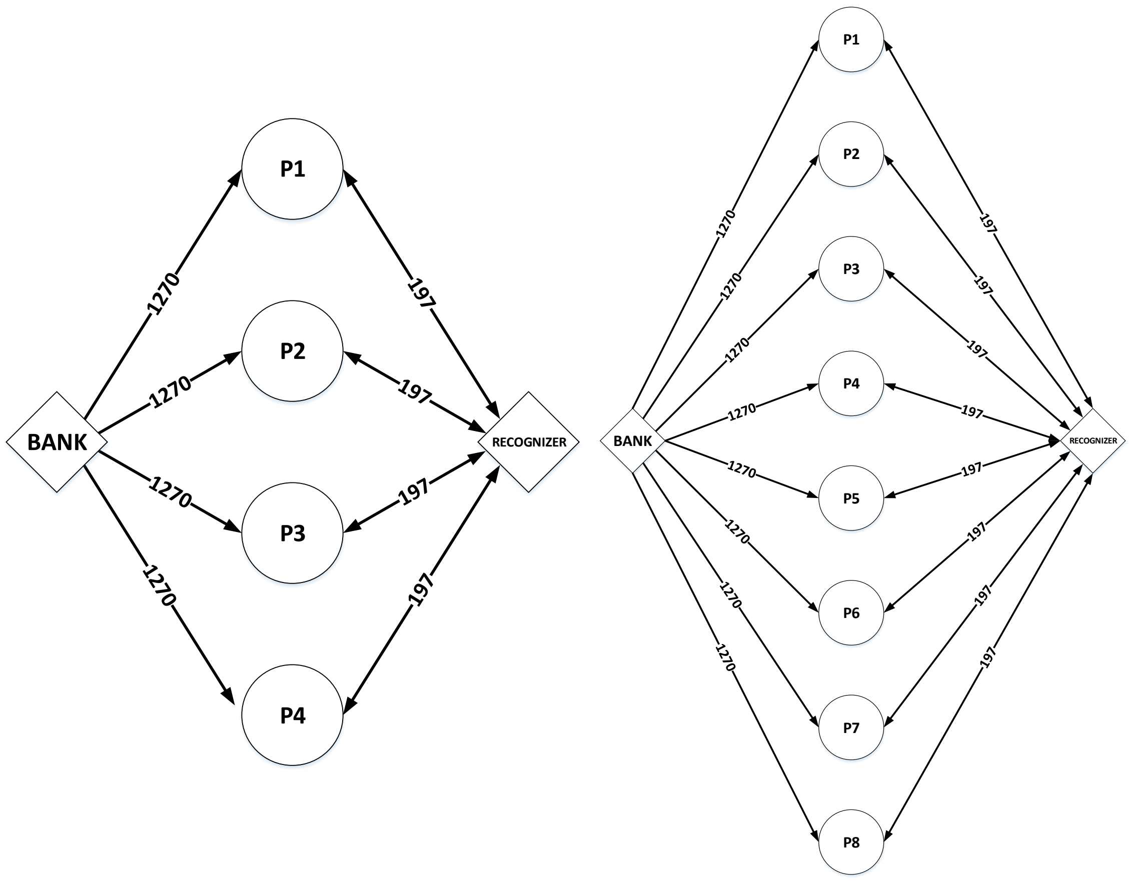
Figure 3.16. Dijkstra application graph.



Source: Author.

- DTW (*Dynamic Time Wrapping*): an algorithm for measuring the similarity between two temporal sequences which may vary in speed. This application is parametrizable, and this work presents two cases with six (6) and ten (10) tasks. Figure 3.17 shows the application graph which has multiple task dependencies with high communication workload.

Figure 3.17. DTW application graphs with six (a) and ten (b) tasks.



(a) (b)

Source: Author.

Aforementioned applications have very different profiles regarding communication workload, computation cost, the number of tasks, and task sizes. These characteristics are needed to evaluate the system performance under different workload situations. Once the framework generates the platform and the application set, the system simulation starts and the simulation results can be extracted as described below.

3.4 Resulted Information and Debugging

The OVPsim tool model allows extracting data during the simulation improving debug and evaluation metrics. The metrics used to evaluate and validate the proposed extension are the communication volume through the NoC, the energy spent in task communication, the execution time for each processor, and the task mapping workload distribution. This framework integrates some high-level models to extract these metrics, which are described below.

The extraction of the communication volume is made whenever one PE sent messages through the NoC, differing between the total communication and the inter-task communication. The total volume communication considers all the packets sent through the NoC while the task communication considers just inter-task communication services. Also, the energy spent in communication is characterized by the method proposed by HU et al., 2010 which considers the packet size and the number of hops to determine the total energy spent. The amount of energy spent for each flit was extracted by the ST/IBM CMOS 65 nm technology at 1.0V, considering clock gating and a 100 MHz clock frequency. The equation below presents the total communication energy spent in the NoC.

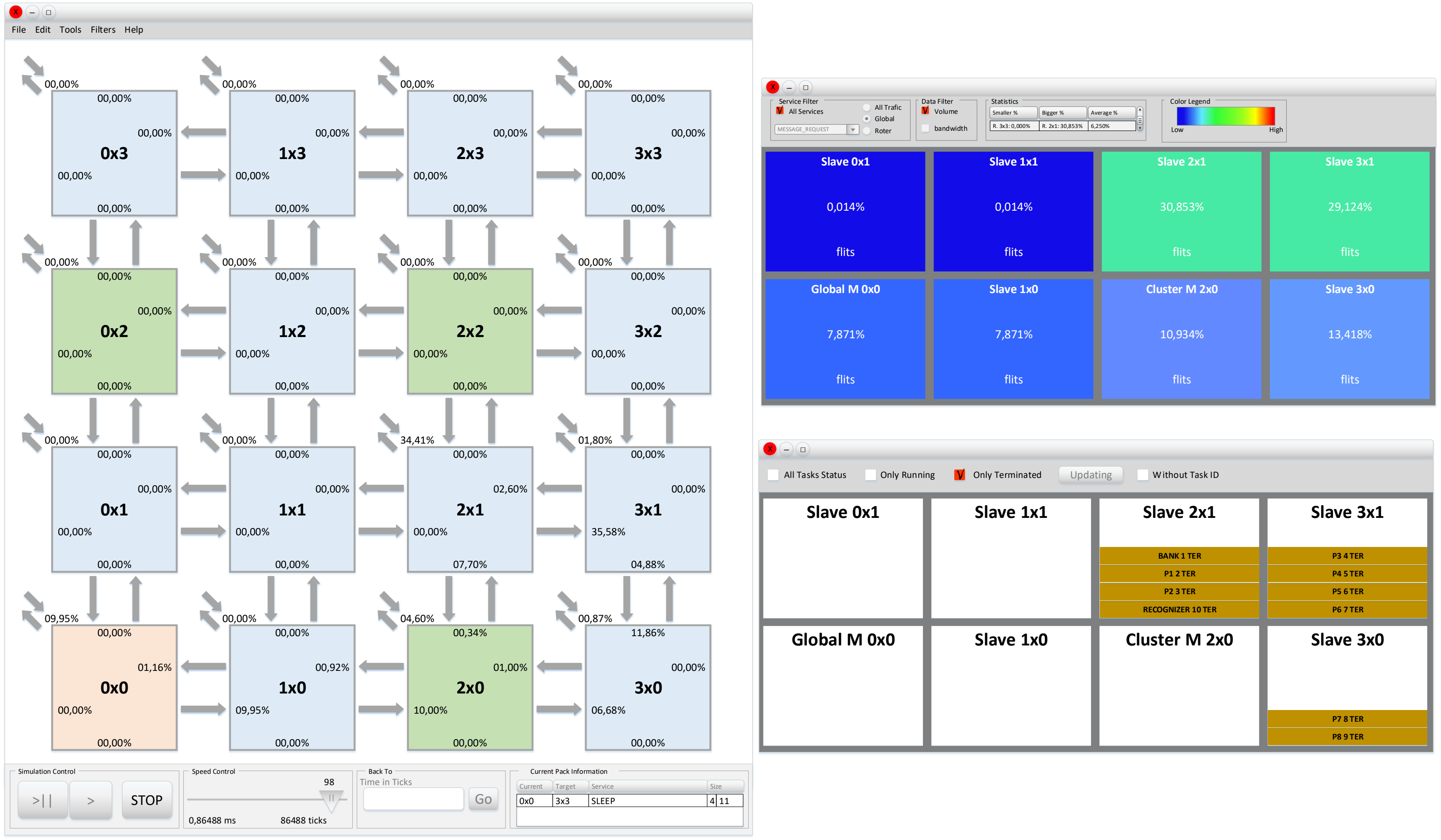
$$toal_{energy} = E_{flit} * \sum flits * hops$$

The execution time for each processor is characterized by the model proposed by ROSA et al., 2013 based on experiments executed on STM32F4-Discovery board with an ARM Cortex-M4F processor. This model executes a watchdog which executes a callback for each executed instruction from a given processor. Then, defines a group for each similar behavioral instruction and estimate the required number of clock cycles to run. The average mismatch between OVPsim CPU timing model and the real board platform is below 5%.

Regarding system debuggability and data extraction, this framework integrates the multiprocessor platform debugger tool developed by RUARO et al., 2014. The framework generates the input configuration files with the information about the platform, system services, and scheduling. With this information, the tool uses the extracted data from NoC and PE simulation events to generate the visual platform and the test case behavioral. Then, allowing to show the workload distribution by the task mapping, and both global and service communication volume.

Figure 3.18 shows a test case visualization example with the visualization of startup platform, task mapping, and communication volume. This test case has one DTW application running on a 4x4 platform distributed by 2x2 cluster where each PE can execute four (4) tasks. On the left of the figure, the startup view shows an animation with the inter-PE NoC communication. On right bottom of the figure, the task mapping overview of the ten (10) application tasks, where two (2) PEs executed four (4) tasks and one executed the two (2) remaining application tasks. The debugging shows each task information such as task name, ID, and state. On right top of the figure, the communication workload distribution where the information is the workload by percent value and by PE color from low (blue cold) to high (red hot). Also, there are the option to choose one service and its distribution in relation to all traffic, global and per router.

Figure 3.18. Startup simulation screen of the Platform Debugger Tool, communication analysis, and task mapping analysis.



Speed Control: 98

0,86488 ms 86488 ticks

Time in Ticks

Current	Target	Service	Size
0x0	3x3	SLEEP	4 11

Service Filter: [✓ All Services] MESSAGE_REQUEST; All Trafic [✓ Global] Roter

Data Filter: [✓ Volume] bandwidth

Smaller %	Bigger %	Average %
R. 3x3: 0,000%	R. 2x1: 30,853%	6,250%

All Tasks Status; Only Running; [✓ Only Terminated]; Updating; Without Task ID

Source: Adapted from RUARO et al., 2014.

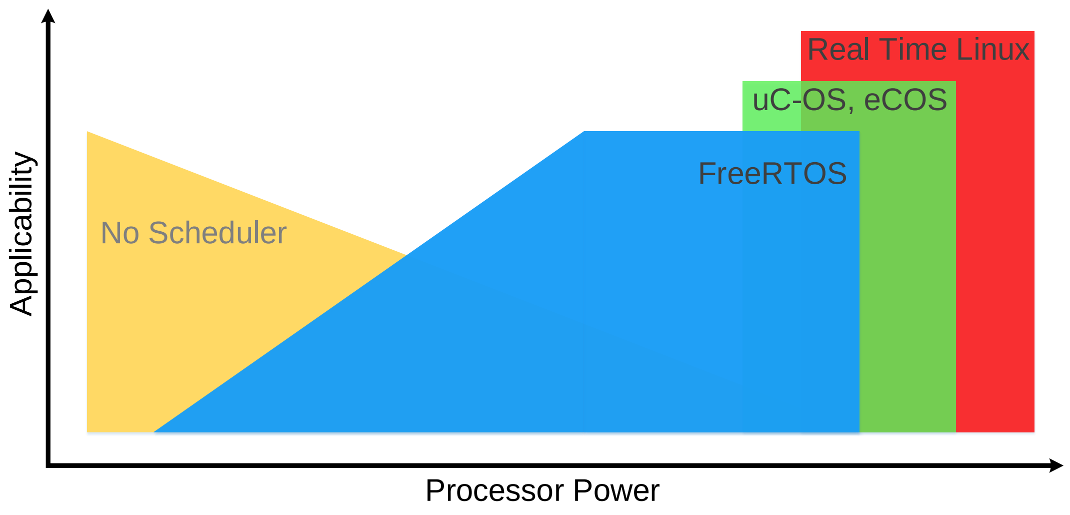
4 PROPOSED FREERTOS EXTENSION

This chapter presents the FreeRTOS real time operating system source kernel and then describes the proposed extension. First, Section 4.1 presents the FreeRTOS kernel and describes the default features and the API reference included in the source code. The following sections describe the proposed extension organization. Section 4.2 describes the architecture and application portability. Section 4.3 describes the developed MPI-like NoC communication control. Finally, Section 4.4 presents the distributed resource management and the integrated task mapping techniques.

4.1 FreeRTOS Source

FreeRTOS is an open-source real-time operating system, widely used and market leading in embedded system projects. FreeRTOS has an active development community in partnership with the world's leading chip companies and has been validated over more than 30 different processor architectures. Although its kernel footprint varies in the region of 6K to 12K bytes, the source project provides several API facilities and functions.

Figure 4.1. OS processor architectures portability.



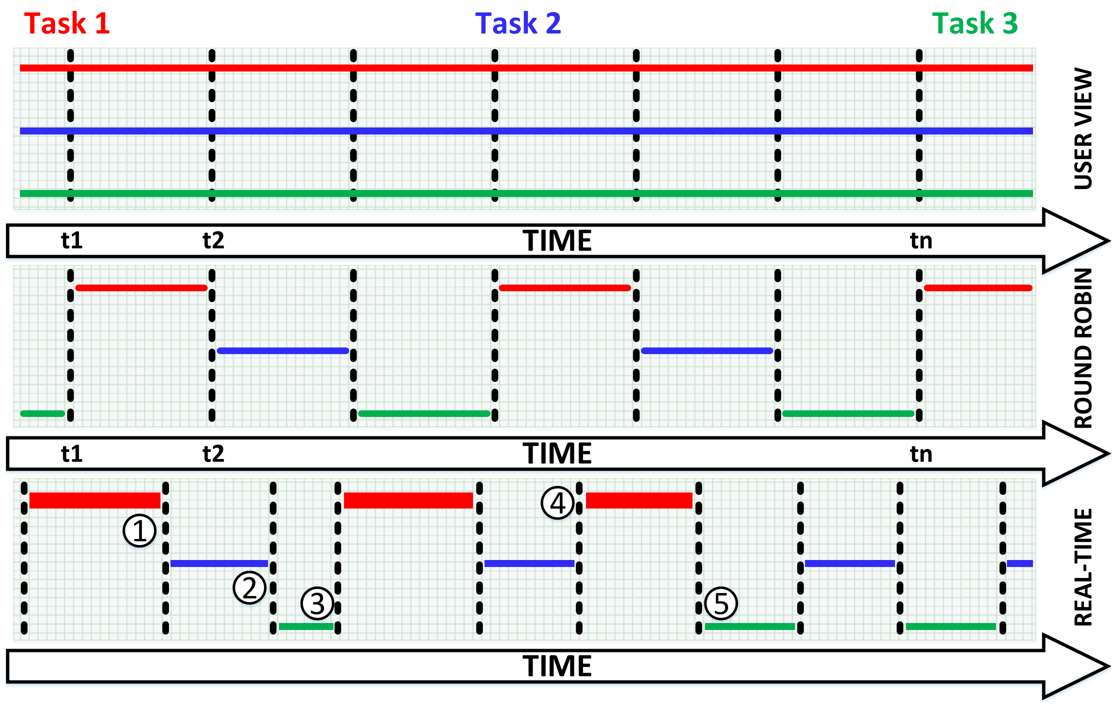
Source: Adapted from BARRY 2013.

Figure 4.1 is a diagram which shows where FreeRTOS fits and the comparison with other operating systems applicability. The applicability axis considers from a non-scheduling application to a full operating system, and processor power axis considers from a 4-bit microcontroller to a high-performance processor. FreeRTOS applicability covers from very simple microcontroller architectures with adequate RAM and simple scheduling to more powerful architectures and applications. Therefore, FreeRTOS have more applicability and portability in both software and hardware levels.

Regarding the system features, the basic system execution includes interruption handlers, task scheduling, and memory management. The memory management includes five (5) sample memory allocation (i.e. static and dynamic) implementations which allow controlling allocation and deallocation of RAM regions. This work uses the FreeRTOS *heap_4* memory allocation implementation which allows to allocate and deallocate memory; also it does combine adjacent free blocks to avoid fragmentation. The interruption handler controls the software and hardware interruptions by the interruption vector table. The interruption registers vary according to the processor architecture and can be configured due to the necessity. FreeRTOS execution needs some basic software interruptions such as system calls, context switching, hard fault, and reset. However, as a real-time operating system, FreeRTOS needs at least one timer interruption to apply its real-time scheduling constraints.

FreeRTOS scheduler considers threads as tasks, allows to improve multitasking and have three (3) possible configurations: preemptive, cooperative, and hybrid. In multitasking scheduling, all available tasks appear to be executing in parallel, but only one task is executing at any time. Figure 4.2 illustrates the timelines to explain how does the scheduler appear to be working to the users and how it really works. First, the timeline shows the users view where all tasks seem to be executing at the same time. The second timeline shows the round-robin co-operative scheduling where each task will execute at any time, and the scheduler executes the context switching. The last timeline shows the preemptive priority-based scheduling where *Task 1* have higher priority, and the remaining tasks have the same lower priority. In ‘1’ and ‘2’, the scheduler executes a normal context switching, note that *Task 1* execution time is longer than others. In ‘3’, *Task 3* tries to access an occupied processor peripheral, finding it locked it cannot continue, so it suspends itself and switches the context to *Task 1*. In ‘4’, the context is switching between *Task 1* and *Task 2* while *Task 3* is suspended. In ‘5’, *Task 1* ends, releasing the processor peripheral requested by *Task 3*, then it resumes. Finally, *Task 2* and *Task 3* have the same priority; then they have the same execution time.

Figure 4.2. FreeRTOS scheduling in different perspectives of view.



Source: Adapted from FREERTOS 2017.

The FreeRTOS set of API facilities are validated at single PE architectures and includes inter-task communication and queue control, semaphores, co-routines, timers, mutexes, trace macros, and also tick less low power features. The FreeRTOS defines and functions used to provide the proposed extension are described as follows:

- *xTaskCreate*: creates a new task, allocates the stack size, and add it to the list of tasks that are ready to run;
- *vTaskSuspend*: suspends a ready task removing it from the ready list and saving its context;
- *vTaskResume*: resumes a suspended task restoring its context and adding it to ready list;
- *vTaskDelete*: remove a task from the RTOS kernels management and deallocate the stack size;

- *vTaskDelay*: delay a task for a relative number of ticks;
- *vTaskDelayUntil*: delay a task for an absolute number of ticks;
- *vTaskStartScheduler*: starts the RTOS scheduler and the Idle control task which controls features such as memory deallocation, memory coalescence, and task context switching;
- *vTaskEndScheduler*: finishes the scheduler and also the system execution;
- *pvPortMalloc*: a FreeRTOS provided memory allocation function;
- *vPortFree*: a FreeRTOS provided memory freeing function;
- *configSUPPORT_DYNAMIC_ALLOCATION*: defined to ‘1’ allows to use dynamic memory allocation;
- *configTICK_RATE_HZ*: sets the SYSTICK timer interruption rate (i.e. clock divider);
- *configUSE_PREEMPTION*: defined to ‘1’ uses the real-time preemptive scheduler;
- *configMINIMAL_STACK_SIZE*: defines the minimal stack size used to system default task such as Idle Task;
- *configTOTAL_HEAP_SIZE*: defines the heap size, addressing the memory to be managed.

Aiming at keeping FreeRTOS modularity and flexibility, its original structure was maintained and the promoted extensions were developed to operate in a non-intrusive manner. The proposed extension includes the FreeRTOS features regarding processor architecture portability, memory management, interruption handlers, scheduling and multitasking management. Note that this work uses the FreeRTOS preemptive real-time scheduling but does not objective the real-time performance exploration and analysis. Therefore, all those features and functionalities allow and justify the choice of using FreeRTOS in this work. Underlying extensions were developed targeting NoC-based multiprocessor architectures, and they are described in the following sections.

4.2 System and Application Portability

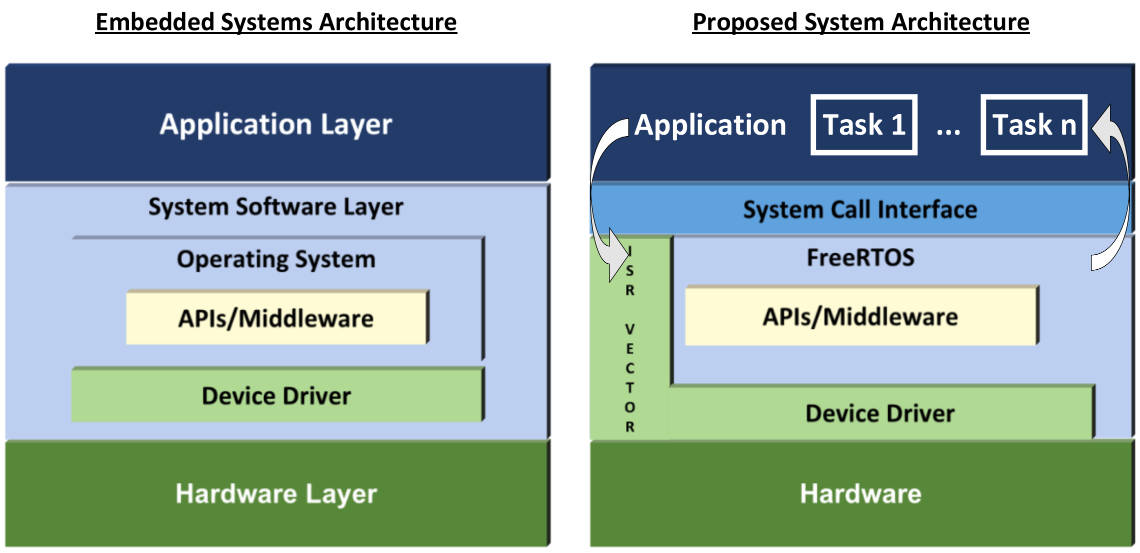
Modern embedded OSs and processors support the interruption control. This feature allows handling high-priority interruptions during the system execution, improving software and hardware control and portability. As shown in Figure 3.1, the proposed non-intrusive extension does not change the FreeRTOS source project. The developed extension has the same FreeRTOS folder and files organization model. Once the FreeRTOS supports multiple processor architectures, it provides the interface between hardware and software. There are a set of defines and functions to enable the system needed interruptions for each processor architecture, such as the CMSIS for ARM Cortex-M family. The extension underlies the processor architecture portability by defining the interruption vector configuration. The system startup points to the processor interruption vector which has the default FreeRTOS functions and the extension API services. The extension project is organized as follows:

- *Startup*: is the system startup file which has interruption vector configuration and the startup *Reset* function. This method enables interruptions, handlers, and then executes to the main function.
- *Main*: has the *main* function which initializes the required data structures depending on the PE function on the MPSoC and then starts the FreeRTOS scheduler.
- *System_Call*: implements the system call handler function which treats the system call interruption and then executes the API function required by the application task.
- *Network_Interface*: has the NI handler function which treats incoming NoC packets and executes services required by any other PE.
- *Communication*: has the developed MPI-like API with the send and receive primitive functions.
- *Mapping*: has the integrated task mapping algorithms which are responsible for defining where the applications and the application tasks will be executed.
- *Distributed Management*: has the implemented distributed resource and application management functions.

- *Misc*: any other miscellaneous auxiliary functions.

Furthermore, there are two folders with the portable and the extension libraries. The *portable* folder (Figure 3.1) has the architecture specific libraries of the processors which the extensions are validated (i.e. ARMv6-M and ARMv7-M). For each processor architecture, there is a library folder, and future extensions libraries also can be included. Then, the project loads both extension and FreeRTOS libraries compile the proposed embedded multiprocessor system.

Figure 4.3. Software portability under embedded system architecture abstraction layers.

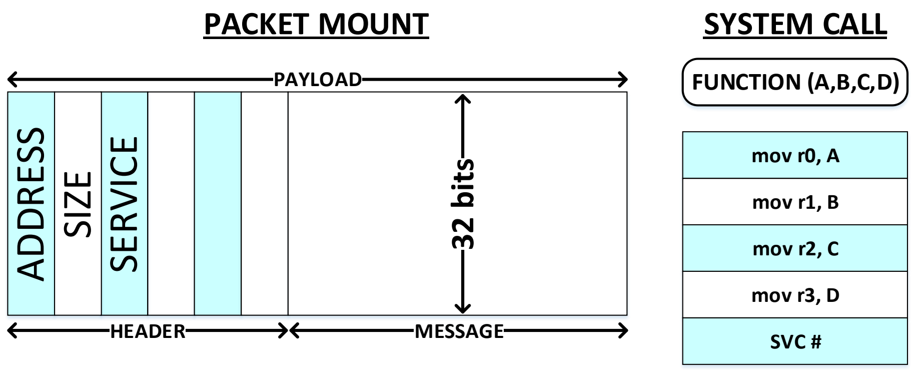


Source: Adapted from NOERGAARD 2012.

Figure 4.3 shows the typical embedded systems architecture and the proposed embedded system architecture divided by layers. In general, the embedded systems are composed of application, system software, and hardware layers. Inside the proposed system software layer there is the kernel with the interruption vector (*ISR VECTOR)* where the interruptions are configured to point their function handlers. The proposed extension use two interruptions to provide the proposed system extension under NoC-based multiprocessors: the system call and the NI interruption. The system call interruption is triggered by the system call instruction (i.e. *svc* in ARM Cortex-M) to perform the interface between the application tasks requests and the kernel APIs. The external NI interruption is triggered by incoming packets through the NoC to perform the inter-PE communication. For each of those interruptions, there are specific functions to treat the task level (system call handler) and system level (NI handler) requests.

Embedded systems normally implement the application portability by using shared libraries which make the applications depending on kernel and APIs addressing. That means that any kernel modification involves recompiling the entire kernel and all the applications. The proposed non-intrusive extension modifies the System Call Handler to treat the application level requests (e.g. send and receive) in a privileged mode, isolating privileged operations and system resources. The system call functions are: MPI send, MPI receive, Task Delete, and an output debug function. The MPI primitives implement the task communication requests, and the task delete finishes the task execution. Each System Call instruction carries an embedded number, which is associated with a given service, Figure 4.4 explains these services arguments stacked into registers r0-r3. These features make the kernel and applications compilation independent, reducing the software development time.

Figure 4.4. NoC communication packet and system call interface function.



Source: Author.

Inter-PE and inter-task communication is an issue in multiprocessor systems once it involves data dependency through different processor resources to continues task and system execution. The proposed extension integrates the NI interruption handler which treats the incoming packets as services. Figure 4.4 shows NoC packet mount details. In this work, a packet consists of a six 32 bits-wide flits header followed by the payload. The header contains the destination address, the payload size (header plus message size), the service request, and three flits to service parameters. The NI handler services are:

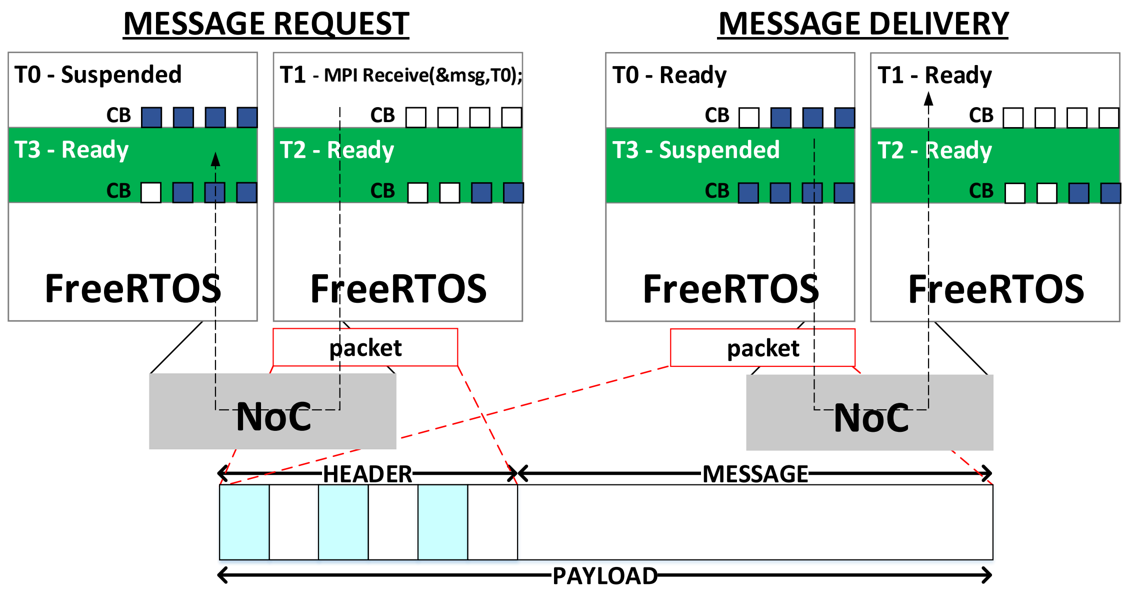
message request, message delivery, task finished, application finished, mapping request, task request, task allocation, application handler, update task location buffer, and sleep PE. The message request and delivery services treat task communication requests. The remaining services are related distributed mapping management. Following sections detail the task management and communication flow and the distributed management with the integrated task mapping algorithms.

4.3 Task Management and NoC Communication

On the proposed extension even the PEs have the same kernel they can assume two functions in the distributed organization: manager or slave. The manager processors are responsible for mapping and allocating application tasks. The slave processors allocate and execute the mapped application tasks. That involves memory allocation, task managing, and scheduling. To the management of the tasks allocation and communication, a Task Manager (TM) with a Task Management Structure (TMS), a Communication Buffer (CB), and a Task Location Buffer (TLB) were incorporated into the FreeRTOS kernel. The TMS contains for each task the local ID, the global application ID, the task relationship ID, and the CB. While CB stores the outgoing task messages, the TLB stores the PE physical address where application tasks are allocated. Whenever an application task is mapped in a slave processor, it allocates the required memory (*pvPortMalloc*) to store the TMS, CB, the application task code, and data. Then, calls the FreeRTOS function *xTaskCreate* to allocate the task stack and add it to the list of tasks that are ready to run.

To enable data transfers between communicating tasks and PEs, an MPI-like API was developed. Underlying API includes two communication primitives: MPI Send and MPI Receive, which are used to transfer data and management control packets devoted to inter-task communication and system management. Whenever a task has to communicate with another, a System Call instruction is invoked, in user mode, triggering the System Call Handler that executes the communication primitives. Once the outgoing messages do not prevent the task execution, the CB stores the message whenever an MPI Send is invoked, and it suspends the sender task when the CB overflows. However, on MPI Receive the task needs the information to continue execution then suspends the task until the data are available.

Figure 4.5. MPI-like task communication.



Source: Author.

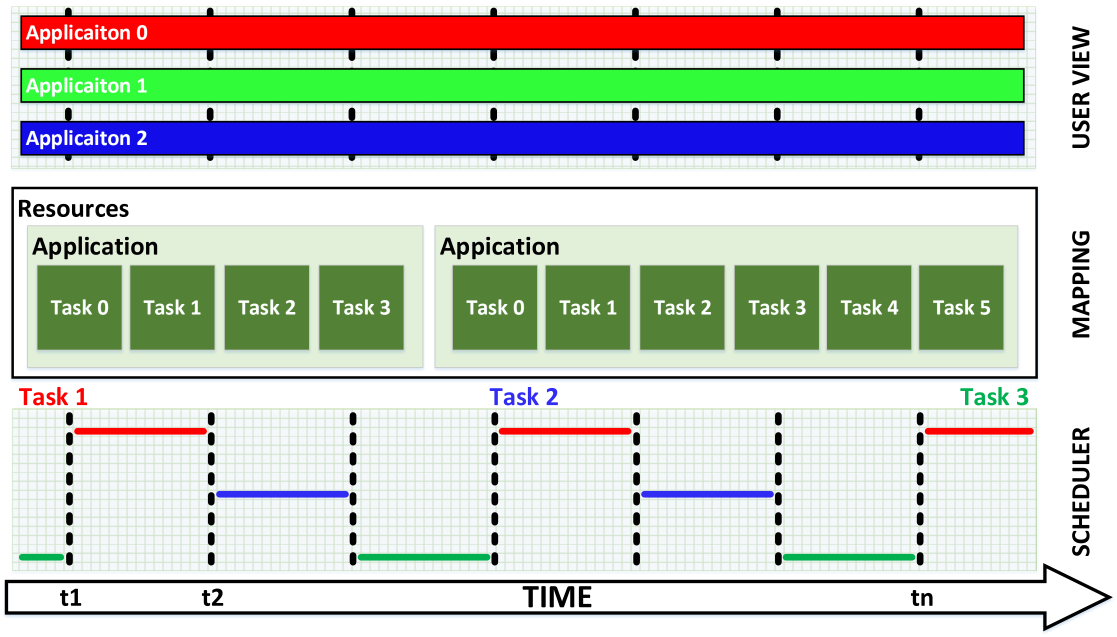
The TM determines the sender processor address by checking the tasks allocation. In this case, there are two possibilities: (i) the requested message originates from a task mapped into the same processor, thus the TM retrieves the message data from the local task CB and delivers it; (ii) the requested message needs to be fetched in another processor. Figure 4.5 shows how the task communication flow occurs through the NoC. An outgoing message first allocates a header descriptor, acquires the receiver physical address from the TLB, and configures the DMA module. This module uses the payload address to transfer the information between the local memory to the NI buffer. If the DMA module was not defined, the kernel feeds the NI buffer.

Further, for each communication, the NI triggers the interruption handler, and the NI Handler manages the message request and delivery services. First, the requesting TM sends to the target TM a service message to fetch the message data (*message request*). The target TM identifies the requesting task ID and removes the message from the sender CB. Further, it resumes the task if it is suspended, then it allocates the header descriptor and configures the DMA module with the message to be sent. Then, the packet is delivered (*message delivery*) and the requesting task is resumed. Note that MPI Receive is task blocking while MPI Send is task non-blocking unless no CB space is available. In both cases the system and any other tasks continues executing, thus they are scheduler non-blocking. During the tasks execution time, the kernel's extensions provide the required functions to their communication and scheduling control. When a task finishes, it calls the system call delete which executes the FreeRTOS *vTaskDelete* function to remove it from scheduling list deallocating the task stack size, and also deallocate the task code and data required memory (*vPortFree*). Even the task has finished, the CB can contain any information and task finishing report will be sent only when the CB is empty.

4.4 Distributed Mapping

In distributed mapping, many parallel applications are executed at the same time independent of the location of the resources (CASTILHOS et al., 2013). Although mapping algorithms map applications in different resources and the system schedule different tasks for execution at any time, in users view all applications are executing in one MPSoC.

Figure 4.6. Distributed mapping application execution in different perspectives of view.

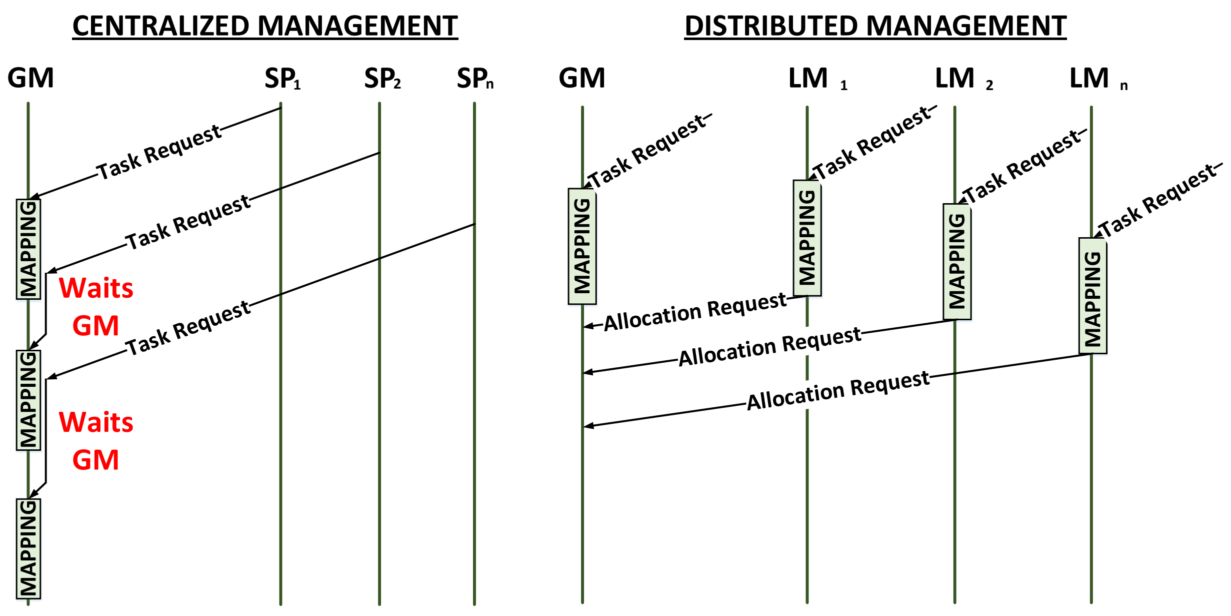


Source: Author.

Aiming to improve gains of performance, reliability, and scalability, the proposed extension integrates distributed mapping techniques which share the MPSoC resources into cluster regions. Figure 4.6 shows the different perspectives of view during the application execution. For the user, once it requests the applications execution, all the applications execute at the same time (user view in Figure 4.6). The distributed mapping considers the information about the applications, the tasks, and the available resources while determining each PE will execute each task (mapping view in Figure 4.6). In each PE, the FreeRTOS scheduler is responsible for defining which task will be execute at any time (scheduler view in Figure 4.6). At the system startup, each PE assumes one of the following roles:

- Local Manager (LM) - responsible for cluster control, executing functions such as task mapping within the cluster.
- Global Manager (GM) - in addition to the local manager functions, is responsible for the overall system management, such as defining an application-to-cluster mapping and controlling external devices accesses (e.g. application repository).
- Slave PE (SP) - responsible for executing user applications with the possibility of parametrization of a single task or multitasking.

Figure 4.7. Centralized Management vs. Distributed Management.



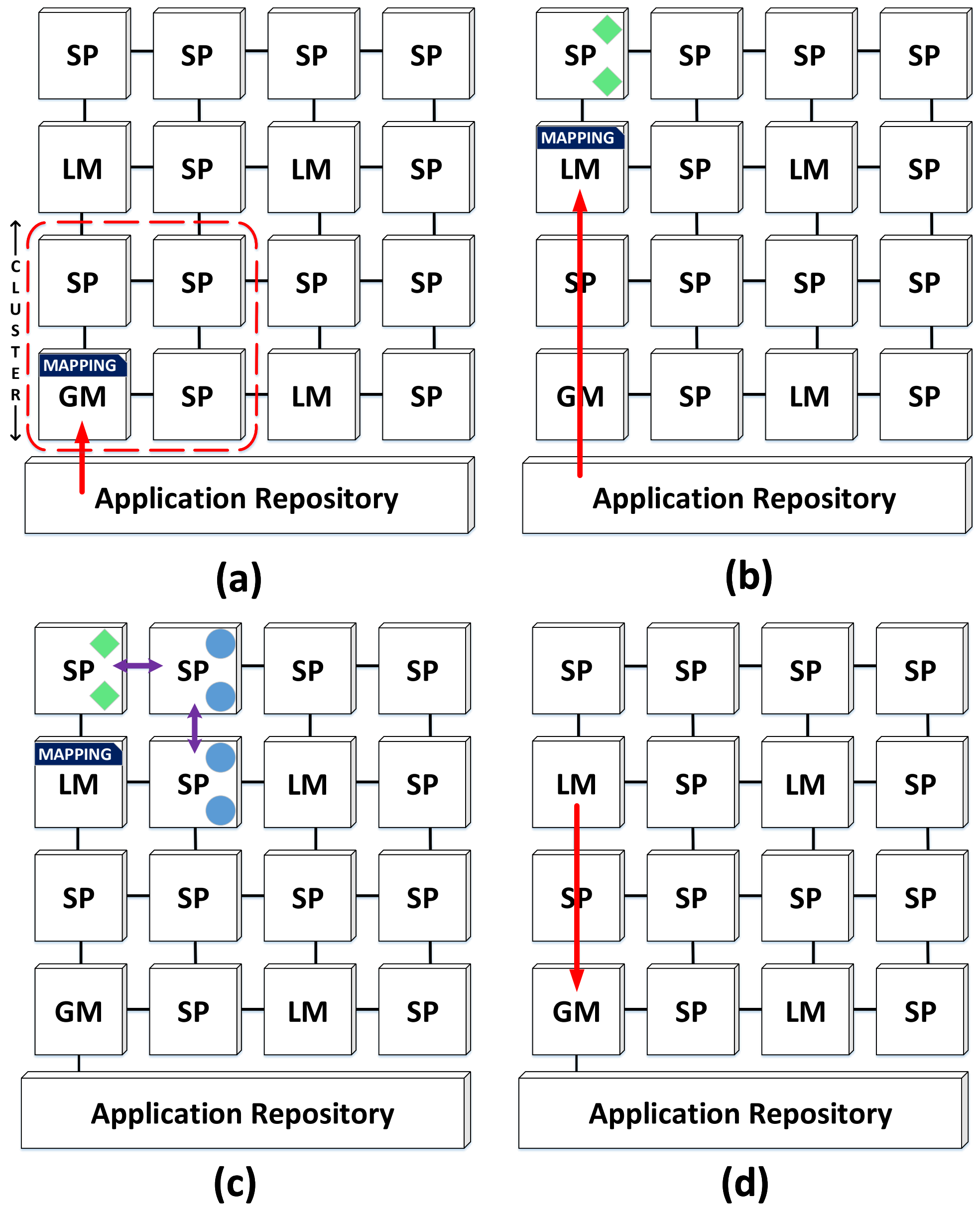
Source: Adapted from CASTILHOS et al., 2013.

The developed extension is also parametrizable, being possible to define: the platform size, the GM position, the cluster size, the maximum number of tasks per PE, the CB size and the application set to execute. The CB size is also parametrizable, allowing to limit the amount of memory to be allocated for inter-task communication. The hardware and the software of all PEs are the same, and this enables to assign the management task to different PEs. Figure 4.7 shows the sequence diagram which compares the centralized and the distributed management. On centralized management, all the task requests are sent to the GM leading delay and overload of services. However, on distributed management, each LMs only handles the incoming task requests from their clusters. This feature increases the reliability of the system since faults in manager processors do not halt the system.

At system startup, the distributed mapping structure is created according to pre-defined parameters. The structure contains the hardware platform size, clusters sizes, their available resources and the manager's addresses. At this time, all processors know the GM, the LM, and the cluster limit (all processors within a cluster have this information). The GM receives requests from the application repository to map new applications. This request contains the number of required resources to execute the application. If the available resources are smaller than the required by the application, the application is scheduled to execute later.

Figure 4.8 illustrates a 4x4 NoC-based platform, with 2x2 clusters, the application path, and the tasks execution flow with message passing interface. It details the application and task management flow starting from the application request in the system startup and finishing at the end of the application execution. The application information is loaded to GM, at runtime, from the repository. The mapping begins with *SearchCluster* function, which analyses the application information and clusters available resources to send the application header to the destination cluster LM (Figure 4.8 a). The target LM receives the application header and executes a mapping heuristic for the initial tasks. After that, it sends an update TLB message to the selected PEs and the GM, then requests to the GM to send the task code to be allocated in destination SP (Figure 4.8 b). The destination PE receives the task code, and the function *xTaskCreate* creates a new task. Then the required RAM is automatically allocated from the FreeRTOS heap, while the underlying task is included in the list of tasks that are ready to run. During its execution, the initial tasks request to the local manager to map the remaining tasks (same task mapping flow) (Figure 4.8 c). At the end of its execution, the task calls system call delete performing *vTaskDelete*, removes the task from the scheduler and frees the allocated space. The kernel only notifies the LM that the task is finished after the message buffer is empty. In turn, when the application is finished the LM reports the availability of new resources to GM (Figure 4.8 d). Finally, the GM maps other incoming applications or finishes system execution.

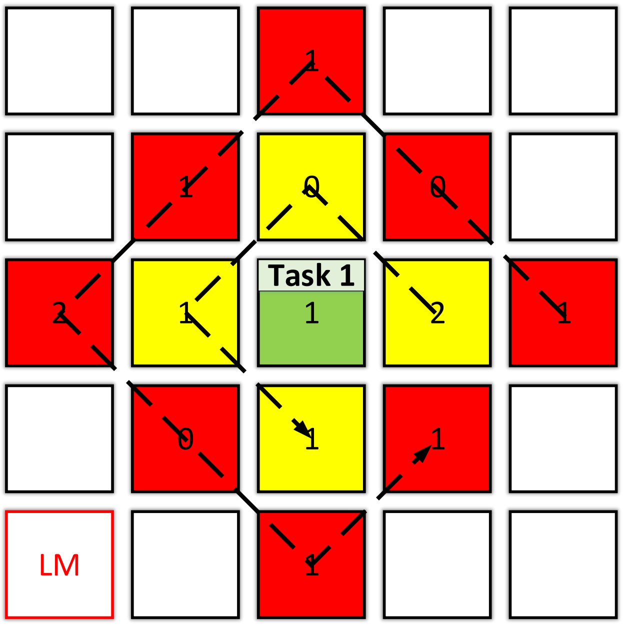
Figure 4.8. Application and task management flow under distributed MPSoC.



Source: Author.

The task mapping algorithms define where each application and application task will be executed. The heuristics algorithm represents the applications as task graphs based on its ID and characteristics (e.g. load, communication), then select the best resource to allocate the task based on its heuristics. This work deploys two task mapping heuristics: Nearest Neighbor (NN) and Low Energy Communication based on Dependencies Neighbor (LEC-DN) (MANDELLI et al., 2015). The initial task mapping evaluates all SPs inside the selected cluster, selecting the SP with the largest number of free SPs around it. In the NN heuristic, the requesting PE ID is the input parameter to define where the next task will be allocated. This heuristic is a simple dynamic task mapping and it does not consider the application tasks characteristics. Then, the algorithm only searches the nearest available resource to define the task allocation. Figure 4.9 shows the mapping heuristic and the search order. In this case, the PEs are represented as squares and the numbers defines its available resources. Also, the color represents the distance in hops between the map requesting and the targeting PE. The search algorithm tests all n-hop neighbors, n varying between 1 and the NoC limits in a spiral way, stopping when the first free PE is found.

Figure 4.9. Runtime dynamic task mapping search heuristic.



Source: Adapted from MANDELLI at al., 2015.

The LEC-DN heuristic also uses the same search order. However, it reduces the communication volume through the NoC by nearing communicating tasks that exchange a high communication volume. The heuristic evaluates the best resource by task interdependencies considering the communication and loads task profiles. Moreover, the heuristic task execution performance by reducing the hops between interdependent communicating tasks.

5 EXPERIMENTAL SETUPS AND RESULTS

This chapter describes the test cases used to validate the proposed framework and the proposed FreeRTOS extension. The test cases vary regarding MPSoC size, clusters size, number of clusters, number of applications, tasks per PE, PE architecture, platform abstraction level, and task mapping techniques. All the experiments have been performed with the Imperas OVPsim version 20160323 on Intel i7-4790K 4.00 GHz machine, with 32GB RAM, running Ubuntu 16.04 64 bits. The chapter is organized as follows: Section 5.1 describes the validation of the proposed system extension by executing all the application types. Section 5.2 describes the proposed framework and system validation under test cases with different MPSoC sizes, cluster sizes, number of clusters and application sets. Section 5.3 describes the system extension evaluation by high different application workloads under large scale platforms. Section 5.5 shows the system validation under Cortex-M0-based RTL level hardware platform. Finally, section 5.4 shows the system portability by using publisher-subscriber protocol.

5.1 System Validation Through Different Applications with Time-Varying Workloads

To validate the proposed FreeRTOS extension, the first test cases execute all the applications available in the proposed framework. Then, for each application type and task mapping heuristic, there is a test case to be executed. Also, to stress the system, there is one test case which executes all the application types. Thus, the FreeRTOS extension needs to deal with the different time-varying workloads of the applications.

Table III shows the proposed test cases for validating the system extension with different application types and mapping heuristics. Due the different task number and different workloads, the test cases with single application types were executed in a 4x4 MPSoC platform with defined multitasking of two (2) tasks per PE. Also, the test case with all the applications was executed in an 8x8 MPSoC size with 4x4 cluster size, also with multitasking. Both hardware platforms considered use Cortex-M4F processor architecture.

Table III: Test cases used to validate the proposed system extension with time-varying application workloads.

Test Case	Applications	Application #	Task #
A	Producer Consumer	1	2
B	MPEG	1	5
C	MPEG4	1	12
D	Fixed-Base Test	1	14
E	VOPD	1	12
F	MWD	1	12
G	DTW6	1	6
H	DTW10	1	10
I	Dijkstra	1	6
J	Producer Consumer, MPEG, MPEG4, Fixe-Base Test, VOPD, MWD, DTW6, DTW10, Dijkstra	9	79

Source: Author.

Thus, Table IV shows the extracted results from the validation test cases. The results from the NoC communication are the volume in some flits and the energy spent in nanojoules extracted by the integrated energy model. The execution time in clock cycles was extracted from the integrated timing model. Finally, the simulation time and the simulated instructions were extracted from the OVPsim.

The results show that the system extension provides all the required features to execute different workload applications in MPSoCs. In the test cases with one application type, the results show that the higher workload applications such as test cases C and H, and applications with high computing effort such as B, G, I, and H, have high execution and simulation time. For example, the applications in F and I have similar communication volume, but the execution time in 1K clock cycles is 4.24 times higher and the simulation time is 3.53 times higher. This shows that both characteristics, communication volume, and computing effort are important issues while considering workload distribution.

Table IV: Extracted results from the proposed test cases in Table III.

Mapping	Test case	NoC Communication Volume (flits)	NoC Communication Energy(nJ)	Execution Time (1K Clock Cycles)	Simulation Time (s)	Simulated Instructions
NN	A	227	2.454	797,88	2.84	5559056
	B	5558	42.37	1663,34	5.63	11536640
	C	107800	393.817	9287,38	29.87	63953008
	D	**8046**	**96.118**	901,44	3.38	6509168
	E	8483	52.56	2059,39	7.05	14379904
	F	11212	61.021	1553,81	5.64	10889920
	G	8034	43.43	2741,81	8.67	18987280
	H	16039	80.399	2881,28	8.85	20017600
	I	11354	70.713	5595,35	17.41	38520400
	J	181832	1161.057	9221,59	180.44	256039680
LEC-DN	A	227	2.454	797,83	2.82	5559056
	B	5558	42.37	1676,59	5.62	11536640
	C	107800	**392.209**	9285,98	30.11	63953072
	D	8064	98.203	896,81	3.36	6494992
	E	**7419**	**43.494**	2057,43	6.92	14379904
	F	**10120**	**59.22**	1354,91	5.00	9532416
	G	8034	**36.438**	2741,44	8.54	18986592
	H	16039	**78.601**	2880,88	8.90	20017600
	I	11354	**70.713**	5595,09	17.66	38520400
	J	**181630**	**1113.079**	9217,92	175.16	256039680

Source: Author.

To evaluate the impact of the two task mapping heuristics in the workload distribution the better results are underlined. Regarding to the communication volume, only the test case D present the NN heuristics was better. This difference occurs when more communicating tasks are in different processors. The remaining experiments show the communication volume is equal in both heuristics or better in LEC-DN. Even if the communication volume is equal in some cases, the communication energy spent is lower in most test cases with LEC-DN heuristics. That occurs because the LEC-DN task mapping algorithm approximates communicating tasks and reduces the number of hops of task communication packets. Therefore, the reduction of hops and NoC communication increases the system performance which reflects on the applications execution time.

While the most test cases execute only one application in a single cluster, the test case J executes all the application types distributed by clusters. In this test case, the distributed task mapping needs to allocate all the different application tasks and deals with their workloads. Nevertheless, the results show that on LEC-DN the communication volume and also the communication energy was reduced.

5.2 System Validation Through Different Platform Organizations

The proposed framework allows the design space exploration by generating different test cases, varying regarding MPSoC size, cluster size, cluster number, and number of applications. Proposed FreeRTOS extension needs to be able to adapt to this variation and handle with the application workloads. Then, to validate the system adaptability under different platform sizes, cluster distribution and application number. Table V shows the test cases used to validate these features. All the test cases are executing with Cortex-M4F processor architecture with the LEC-DN task mapping algorithm considering multitasking. The MPSoC sizes vary from 2x2 to 20x20 hardware platforms with up to four hundred PEs. Also, the clusters organization varies regarding size from 2x2 to 5x5 and quantity from a single cluster to one hundred clusters.

Table V: Design space exploration and system scalability evaluation test cases.

Test case	**Size**		**Number**		**Application #**	**Task #**
	MPSoC	**Cluster**	**PEs**	**Clusters**		
K	2x2	2x2	4	1	1	5
L	3x3	3x3	9	1	1	5
M	4x4	4x4	16	1	1	5
N	4x4	2x2	16	4	4	20
O	6x6	3x3	36	4	4	20
P	8x8	4x4	64	4	4	20
Q	8x8	2x2	64	16	16	80
R	10x10	5x5	100	4	4	20
S	10x10	2x2	100	25	25	125
T	20x20	5x5	400	16	16	80
U	20x20	2x2	400	100	100	500

Source: Author

Table VI: Extracted results from the proposed test cases in Table V.

Test case	NoC Communication		Simulated Instructions	Simulation Time(s)
	Volume (flits)	Energy(nJ)		
K	5486	23.877	2880396	1.21
L	5516	33.215	6673743	3
M	5558	42.37	11536640	5.55
N	22676	164.903	11889360	5,72
O	22796	238.214	26837208	15,2
P	22964	312.312	47907200	34.11
Q	91436	1214.8	48432704	34.47
R	23180	390.285	75046200	61.17
S	143006	2331.838	76211300	64.38
T	93452	3019.824	383073600	569.91
U	572756	18005.078	464039600	783.84

Source: Author.

The application set for each test case have a same number of clusters and applications just to guarantee that each cluster executes at least one application. Table VI shows the extracted results from the test cases execution. The results show that the system allows the design space exploration while it can handle all the variations in terms of MPSoC size, application set, and workload distribution.

5.3 Workload Distribution in Large-Scale MPSoCs

To validate the proposed work, the platform need to execute high workload test cases considering two tasks mapping heuristics under large scale hardware platforms. For this purpose, three applications with a high workload and also high computing effort are used as benchmarks: DTW, with ten tasks; MPEG, with five tasks; Dijkstra, with six tasks. Table II presents the six evaluated test cases. Such test cases use a 10x10 multiprocessor platform instance with a 5x5 cluster size and with different applications. For the evaluated test cases, was considered that a PE is executing two tasks at a time.

Table VII: Experimental test cases to evaluate the distribution of high workload application sets in large scale MPSoCs.

Test case	Applications	Application #	Task #
W1	120 x MPEG	120	600
W2	100 x DJK	100	600
W3	15 x DTW, 35 x MPEG	50	325
W4	65 x MPEG, 35 x DJK	100	535
W5	10 x DTW, 25 x MPEG, 25 x DJK	60	375
W6	15 x DTW, 5 x MPEG, 40 x DJK	60	415

Source: Author.

Table VIII: Extracted results from the proposed test cases in Table VII.

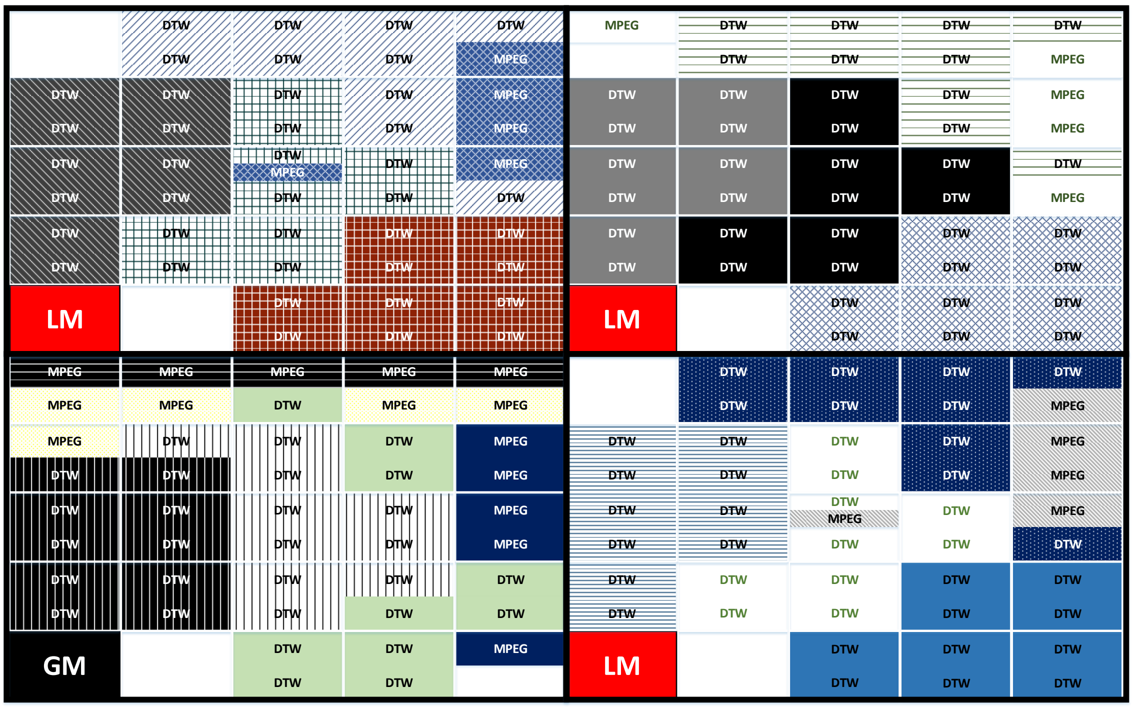
Test case	Communication Energy (µJ)		Execution Time (1K clock cycles)	
	NN	LEC-DN	NN	LEC-DN
W1	16.04	15.74	36493	36806
W2	87.90	87.27	36554	37621
W3	10.93	10.20	27875	27858
W4	39.59	38.95	40036	35554
W5	28.42	27.80	32117	31456
W6	40.20	39.21	41857	39918

Source: Author.

Table VIII presents a comparison of the two evaluated heuristics, concerning two different metrics: communication energy, and execution time. The communication energy presents the cost to transfer the communication volume through the NoC. For this purpose, it considers the distance in hops between each pair of communicating tasks.

LEC-DN reduces the communication energy for all scenarios compared to NN. This is explained since LEC-DN approximates all communicating tasks and NN approximates only pairs of communicating tasks. The execution time varies according to the scenario, depending on the communication volume, NoC contention, mapping algorithm computation and shared execution of tasks in the system.

Figure 5.1. Task mapping example in test case W3 when all resources are occupied.


DTW DTW DTW DTW MPEG DTW DTW DTW DTW
DTW DTW DTW MPEG DTW DTW DTW MPEG
DTW DTW DTW DTW MPEG DTW DTW DTW DTW MPEG
DTW DTW DTW DTW MPEG DTW DTW DTW DTW MPEG
DTW DTW DTW MPEG DTW MPEG DTW DTW DTW DTW DTW
DTW DTW DTW DTW DTW DTW DTW DTW DTW MPEG
DTW DTW DTW DTW DTW DTW DTW DTW DTW DTW
DTW DTW DTW DTW DTW DTW DTW DTW DTW DTW
LM DTW DTW DTW LM DTW DTW DTW
DTW DTW DTW DTW DTW DTW
MPEG MPEG MPEG MPEG MPEG DTW DTW DTW DTW
MPEG MPEG DTW MPEG MPEG DTW DTW DTW MPEG
MPEG DTW DTW DTW MPEG DTW DTW DTW DTW MPEG
DTW DTW DTW DTW MPEG DTW DTW DTW DTW MPEG
DTW DTW DTW DTW MPEG DTW DTW DTW MPEG DTW MPEG
DTW DTW DTW DTW MPEG DTW DTW DTW DTW DTW
DTW DTW DTW DTW DTW DTW DTW DTW DTW DTW
DTW DTW DTW DTW DTW DTW DTW DTW DTW DTW
GM DTW DTW MPEG LM DTW DTW DTW
DTW DTW DTW DTW DTW

Source: Author.

Figure 5.1 illustrates an example of LEC-DN mapping (test case W3), where each color corresponds to a different application. Note the locality of the applications, where tasks belonging to the same application are mapped in the same region, with a small distance regarding a number of hops.

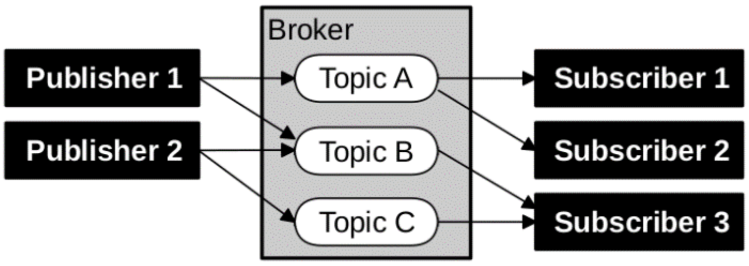
5.4 Integrating a Publisher-Subscriber Protocol into NoC-based MPSoCs

Middleware is an abstraction layer generally used on embedded systems with two or more applications in order to provide flexibility, security, portability, connectivity, intercommunication, and/or interoperability mechanisms between applications (NOERGAARD 2012). This section describes a case study developed by a partner (HAMESRKY et al., 2017), which integrates a Publisher-subscriber protocol in the proposed FreeRTOS extension. The publish-subscribe (PUB-SUB) programming model has been used in middleware for highly distributed domains, such as: MQTT 1 (Message

Queuing Telemetry Transport) for sensors networks and mobile devices domains; DDS 2 (Data Distribution Service) for real-time systems domains; and ROS 3 (Robot Operating System) for robotics domains. All these middlewares evolved to provide properties, such as reliability, security, low power consumption, and QoS.

Figure 5.2 shows a general scheme with two publishers, three subscribers, three topics, and one broker. A publisher can publish data in more than one topic. A subscriber can receive data about same or different topics from one or more publishers.

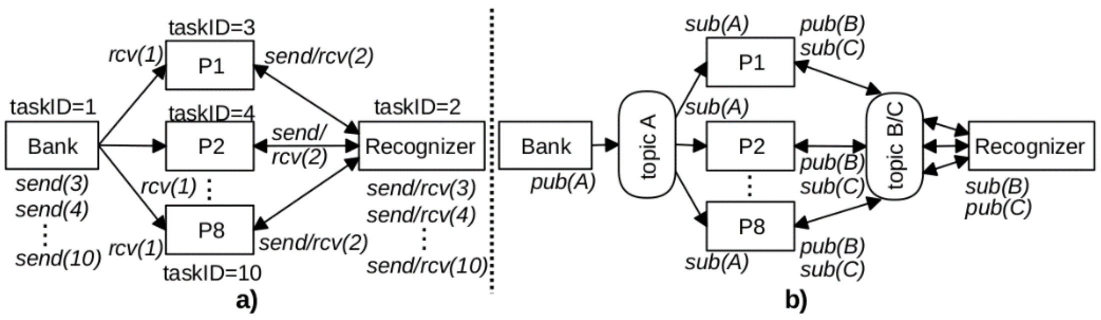
Figure 5.2. The general scheme of a publish-subscribe system.



Source: HAMERSKY et al., 2017.

In MPSoC domain, publisher nodes are those which want to produce data, while subscribers are nodes that want to consume them. The topics represent atomic data shared among the nodes. As an example, Figure 5.3 shows the DTW application with ten tasks (Bank, P1-P8 workers, and Recognizer) represented through a task graph. A directed arrow between two tasks (blocks in Figure 5.2) means that the first task sends data to the second one.

Figure 5.3. DTW task graph in (a) MPI and (b) PUB-SUB models.



Source: HAMERSKY et al., 2017.

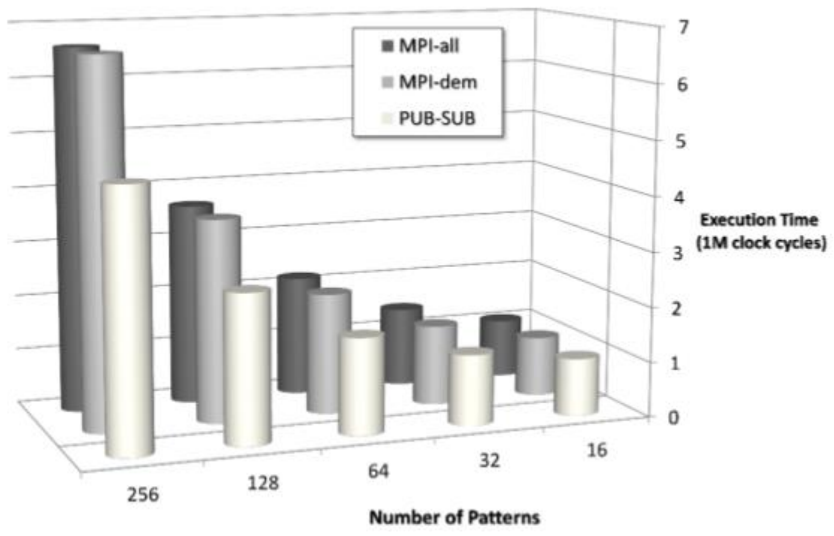
The PUB-SUB version replaces the MPI primitives by PUB-SUB primitives. The sender side must define a topic ID for the flow, register itself in the system as publisher of that topic and publish the data. The receiver side must register itself as a subscriber of that topic, setting a callback function to handle the incoming data.

The middleware is integrated on FreeRTOS between the kernel and the application level. While system calls provide the PUB-SUB API primitives to the user application(s), the network interruptions (NIs) are used to manage the API services at the system level. The management structure remains itself with the incorporation of the PUB-SUB middleware, with PEs assuming new management functions. Both LM and GM assume the role of brokers, but they can also be publishers and subscribers. The SPs can only be publishers and/or subscribers.

The experiments are based on the DTW application. This application has been chosen because it uses a communication pattern of 1:N and N:1 (N is the number of workers). This experiment uses eight workers. Three scenarios were analyzed: MPI-all, MPI-dem, and PUB-SUB. The first two scenarios use MPI primitives with all tasks mapped at the beginning of the execution (MPI-all), or tasks mapped on demand (MPI-dem), where only the initial tasks are mapped at the beginning of the execution and the other tasks are mapped as soon as there is communication among them. The PUB-SUB scenario uses the proposed publish-subscribe primitives and middleware, with all the tasks mapped at the beginning of the execution. All scenarios use a single-cluster 5x5 MPSoC, with each PE executing a single task to stimulate the NoC communication between the tasks and evaluate the middleware protocol.

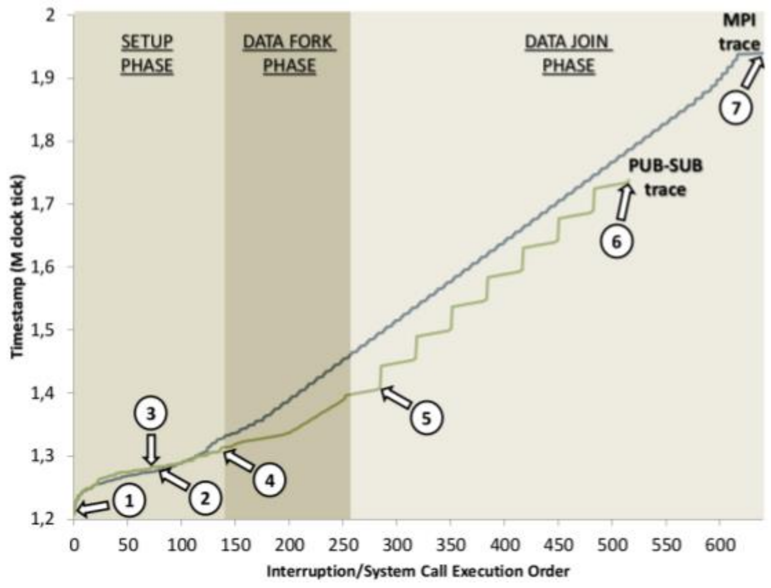
Figure 5.4 shows the results of the DTW application execution time, using a model that captures the executed instructions for each PE, generating an execution time of the total executed instructions. PUB-SUB reduces the execution time from 2.6% to 29.9% as the number of patterns is increased, respectively, from 16 to 256. Compared to MPI, the PUB-SUB model requires an initial setup time to advertise the topics. Besides, the PUB-SUB application object code is lightly bigger, taking more time to finish the task mapping. Therefore, the MPI has an advantage for small communication volumes. However, the MPI model presents the drawback of generating more System Calls and NIs caused by the messages, as detailed next.

Figure 5.4. DTW execution time using MPI and PUB-SUB.



Source: HAMERSKY et al., 2017.

Figure 5.5. MPI vs PUB-SUB time spent in System Calls and NIs.



Source: HAMERSKY et al., 2017.

Figure 5.5 is a detailed view of the results obtained for 64 patterns, presented in Fig. 5. The X axis represents the order of System Calls or NIs generated in the system, and the Y axis represents the instant of time (timestamp) in which each one of them was executed. The figure also presents two lines representing the MPI and the PUB-SUB execution trace. Since both MPI-all and MPI-dem had the same behavior, only one is illustrated. The figure is divided into the three main phases of the DTW application: setup, data fork, and data join.

5.5 RTL System Validation

The OVPsim MPSoC platform provides high software development, high debuggability, high performance and low simulation. The high level platform can be calibrated from data extracted from different low-level models. However, to really validate the proposed FreeRTOS extension the system needs to be executed on an real RTL platform. In this case, a NoC-based RTL MPSoC platform with the Cortex-M0 processor architecture was provided by a partner. The proposed FreeRTOS extension was executed in both OVPsim and RTL platforms to compare the simulation results. Table IX shows the proposed test cases comparing the proposed framework generating the OVPsim platform with the RTL platform. The test cases inculdes simple application sets with one and four applications just to validate the proposed system extension in different architectures and different abstration levels. The platform sizes varies from 4x4 to 8x8 considering up to four clusters.

Table IX: Test cases to evaluate the FreeRTOS extension running under RTL Cortex-M0 platform.

Test case	**Size**		**Applications**	**Application #**	**Task #**
	MPSoC	**Cluster**			
R1	4x4	4x4	MPEG	1	5
R2	4x4	4x4	DJK	1	6
R3	4x4	4x4	DTW10	1	10
R4	4x4	4x4	Fixed-base Test	1	12
R5	8x8	4x4	MPEG x 4	4	20
R6	8x8	4x4	DJK x 4	4	24
R7	8x8	4x4	DTW10 x 4	4	40
R8	8x8	4x4	Fixed-base Test x 4	4	48

Source: Author.

Table X: Comparison between the results extracted from the proposed test cases in Table IX executed on OVPsim and RTL platforms.

Test case	Communication		Simulation Time (s)	
	Volume (flits)	Energy (nJ)	RTL	OVP
R1	9878	68.817	26745	5.3
R2	13343	81.251	26082	9.05
R3	16051	105.125	60221	6.97
R4	7928	91.435	23985	3.27
R5	40244	418.101	112112	31.77
R6	54251	514.451	115878	53.19
R7	65731	578.663	245350	42.65
R8	33947	731.425	102480	19.77

Source: Author.

Table X shows the extracted results from both RTL and OVPsim platform models. The resultant information shows the OVPsim system validation under high level simulation can be easely applied to real platforms. Even the RTL having more accurance, the test case simulation time is much higher when compared with the OVPsim simulation. In addition, the data extracted from RTL platform description can be used to calibrate or create high level models allowing a more accurate simulation. Therefore, both approaches are important while reducing the design time of complex MPSoCs.

6 CONCLUSIONS

This Dissertation proposes a non-intrusive FreeRTOS extension to support dynamic and distributed task mapping in Multiprocessor Systems. The extension enables FreeRTOS adoption in large-scale NoC-based MPSoC architectures with distributed-memory organization together with runtime distributed mapping heuristics. A Framework was also proposed to design space exploration of high level MPSoC platforms. The framework allows to automatic generate large scale NoC-based MPSoCs with an application set to be executed into the OVPsim tool.

The Framework and FreeRTOS extension were validated in several test cases with many variations regarding MPSoC size, clusters size, application number, task per PEs, PEs architecture, platform abstraction level and task mapping. All these features were executed in a RTL synthesizable platform with different processor architecture.

Results showed the effectiveness of the included mapping heuristics, which allocated communicating tasks near to each other, reducing the NoC congestion and the communication energy. Additionally, the OVPsim is an efficient tool to validate software and the portability to different levels are also simple.

The proposed extension also provided a case study using distributed systems publisher-subscriber middleware applied to MPSoCs. Results show that the proposed middleware is a worthwhile alternative to programming NoC-based multiprocessor platforms, with low footprint overhead, and lower execution time when compared with MPI-based FreeRTOS kernel implementation.

All the proposed extensions and framework proves to be highly portable regarding software development and processor architectures. Therefore, the use of a largely used OS ported to different ISAs facilitates the system design and validation, reducing software design cost and time-to-market. The platform applicability can be extended to fault detection, system security, and IoT applications.

6.1 Future Works

Considering the current technological scenario and the applications of MPSoCs in the literature, there are some important gaps which can be filled regarding multiple input and multiple output data, secure computing and encryption, and fault tolerance. As processors seek more resource efficiency, they increasingly need to target multiple goals at the same time, such as a level of performance, power consumption, and average utilization (POTHUKUCHI et al., 2016). With the concept of multiple inputs and multiple outputs the MPSoC needs to handle multiple workload inputs and balance through to its resources.

At the same time, the system needs to be secure while receiving and transmitting information from and to external means. To ensure the input and output information are correct and true, the system must encrypt and decrypt it, which demands performance and computing time, both available in MPSoCs. The system must do a trade-off between application execution and security handling the information and allocating resources to handle these applications. This issue is directly related to the system reliability once the system handling unrequired information may induce the system to unrecoverable fault or leak private information.

Concerning reliability in NoC-based systems, the literature presents some works proposing alternatives for multiprocessor systems: secure communication between the various devices of a SoC (LUCAS et al., 2009; COTA et al., 2012); Fault-tolerant routing (MARCON et al., 2013, LIU et al., 2013); Monitoring of chip aging (KERKHOFF et al., 2014); Power management in large scale circuits using NoCs (BOKHARI et al., 2015; ZHAN et al., 2014); Failure tolerance in NoCs design (PARK et al., 2006; RADETZKI et al., 2013). However, these features are little explored at the embedded system level for large-scale multiprocessors targeting commercial kernels. Therefore, the contributions of this dissertation provide the required resources to explore those issues in the future works.

REFERENCES

ABICH, G., MANDELLI, M. G., ROSA, F. R., et al. “Extending FreeRTOS to Support Dynamic and Distributed Mapping in Multiprocessor Systems.” In: 2016 IEEE INTERNATIONAL CONFERENCE ON ELECTRONICS, CIRCUITS AND SYSTEMS (ICECS), Monte Carlo, December 2016. **Proceedings...** pp. 712-715. Available in: <https://doi.org/10.1109/ICECS.2016.7841301>.

AGUIAR, A., SÉRGIO FILHO, J., MAGALHÃES, F. G., et al. “Hellfire: A design framework for critical embedded systems' applications.” In: 2010 11th INTERNATIONAL SYMPOSIUM ON QUALITY ELECTRONIC DESIGN (ISQED), San Jose, CA, 2010. **Proceedings...** pp. 730-737. Available in: <https://doi.org/10.1109/ISQED.2010.5450495>.

AGUIAR, A., JOHANN FILHO, S., MAGALHAES, F., et al. “On the design space exploration through the Hellfire Framework.” In: **Journal of Systems Architecture**, Volume 60, Issue 1, January 2014, pp 94-107, ISSN 1383-7621. Available in: <https://doi.org/10.1016/j.sysarc.2013.10.011>.

(ARMDEV 2017) “**ARM Developer Infocenter**” Available in: <http://infocenter.arm.com> Access: February 2017.

(ARMTECH 2017) “**ARM the Architecture for the Digital World**” Available: <http://www.arm.com> Access: February 2017.

BAKLOUTI, M., KRICHENE, H., & ABID, M. “Synchronous Communication-Based Many-Core SoC.” In: **Arabian Journal for Science and Engineering**, Volume 42, Issue 2, February 2017, pp 845–857, ISSN 2191-4281. Available in: <https://doi.org/10.1007/s13369-016-2373-2>.

BENINI, L. & DE MICHELI, G. “Networks on Chip: A New SoC paradigm.” In: **IEEE Computer**, vol. 35, no. 1, Jan 2002, pp. 70-78, ISSN 0018-9162. Available in: <https://doi.org/10.1109/2.976921>.

BINKERT, N., SARDASHTI, S., SEN, R., SEWELL, K., et al. “The gem5 simulator.” In: **ACM SIGARCH Computer Architecture News**, Volume 39, no. 2, pp. 1-7, 2011. Available in: <https://doi.org/10.1145/2024716.2024718>.

BOHNENSTIEHL, B., STILLMAKER, A., PIMENTEL, J., et al. “A 5.8 pJ/Op 115 billion ops/sec, to 1.78 trillion ops/sec 32nm 1000-processor array.”. In: 2016 IEEE SYMPOSIUM ON VLSI CIRCUITS (VLSI-CIRCUITS), Honolulu, HI, 2016. **Proceedings…** pp. 1-2. Available in: <https://doi.org/10.1109/VLSIC.2016.7573511>.

BOHNENSTIEHL, B., STILLMAKER, A., PIMENTEL, J., et al. “KiloCore: A Fine-Grained 1,000-Processor Array for Task-Parallel Applications.” In: **IEEE Micro**, March 2017, volume 37, no. 2, pp. 63-69, ISSN 0272-1732. Available in: <https://doi.org/10.1109/MM.2017.34>.

BOKHARI, H., JAVAID, H., SHAFIQUE, M., et al. “Malleable NoC: dark silicon inspired adaptable Network-On-Chip.” In: PROCEEDINGS OF THE 2015 DESIGN, AUTOMATION & TEST IN EUROPE (DATE) Conference & Exhibition, Grenoble, France. **Proceedings...** Publisher EDA Consortium, pp. 1245-1248.

BORKAR, S. "Microarchitecture and design challenges for gigascale integration." In: 37th INTERNATIONAL SYMPOSIUM ON MICROARCHITECTURE, 2004. **Proceedings…** Volume 37, 2004, pp. 3-3. Available in: <https://doi.org/10.1109/MICRO.2004.24>.

BOSE, P. “Technical Perspective: Is Dark Silicon Real?” In: **Communications of the ACM**, 2013, Volume 56, no. 2, pp. 92-92, ISSN 0001-0782. Available in: <https://doi.org/10.1145/2408776.2408796>.

BURGIO, P., DANILO, R., MARONGIU, A., et al. “A tightly-coupled Hardware Controller to improve scalability and programmability of shared-memory heterogeneous clusters.” In: PROCEEDINGS OF THE CONFERENCE ON DESIGN, AUTOMATION & TEST IN EUROPE (DATE), 2014. **Proceedings...** Publisher European Design and Automation Association, pp. 25:1-25:4.

BUSSEUIL, R., BARTHE, L., ALMEIDA, G. M., et al. “Open-scale: A scalable, open-source NoC-based MPSoC for design space exploration.” In: 2011 INTERNATIONAL CONFERENCE ON RECONFIGURABLE COMPUTING AND FPGAs (ReConFig), IEEE, November 2011. **Proceedings...** pp. 357-362. Available in: <https://doi.org/10.1109/ReConFig.2011.66>.

CASTILHOS, G., MANDELLI, M., MADALOZZO, G., et al. “Distributed resource management in NoC-based MPSoCs with dynamic cluster sizes.” In: 2013 IEEE COMPUTER SOCIETY ANNUAL SYMPOSIUM ON VLSI (ISVLSI), Natal, 2013. **Proceedings...** pp. 153-158. Available in: <https://doi.org/10.1109/ISVLSI.2013.6654651>.

CASTILHOS, G., MANDELLI, M., OST, L., et al. “Hierarchical energy monitoring for task mapping in many-core systems.” In: **Journal of Systems Architecture**, Volume 63, February 2016, Pages 80-92, ISSN 1383-7621, Available in: <https://doi.org/10.1016/j.sysarc.2016.01.005>.

COTA, E., AMORY, A., & LUBASZEWSKI, M. S. “Reliability, Availability and Serviceability of Networks-on-chip.” In: **Springer Science & Business Media**, Publisher Springer US, 2011, pp. 209. Available in: <https://doi.org/10.1007/978-1-4614-0791-1>.

DAS, A., KUMAR, A., & VEERAVALLI, B. “Reliability-driven task mapping for lifetime extension of networks-on-chip based multiprocessor systems.” In: PROCEEDINGS OF THE CONFERENCE ON DESIGN, AUTOMATION AND TEST IN EUROPE (DATE), Grenoble, France. **Proceedings...** Publisher EDA Consortium, 2013, pp. 689-694.

DAS, A., KUMAR, A., & VEERAVALLI, B. “Temperature aware energy-reliability trade-offs for mapping of throughput-constrained applications on multimedia MPSoCs.” In: 2014 DESIGN, AUTOMATION & TEST IN EUROPE CONFERENCE &

EXHIBITION (DATE), Dresden, 2014. **Proceedings...** pp. 1-6. Available in: <https://doi.org/10.7873/DATE.2014.115>.

DAS, A., KUMAR, A., & VEERAVALLI, B. "Reliability and energy-aware mapping and scheduling of multimedia applications on multiprocessor systems." In: **IEEE Transactions on Parallel and Distributed Systems**, March 2016, vol. 27, no. 3, pp. 869-884, ISSN 1045-9219. Available in: <https://doi.org/10.1109/TPDS.2015.2412137>.

ESMAEILZADEH, H., BLEM, E., ST. AMANT, R., et al. "Dark silicon and the end of multicore scaling." In: Proceedings of the 38th ANNUAL INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURE (ISCA), 2011. **Proceedings...** Vol. 39, No. 3, pp. 365-376. Available in: <https://doi.org/10.1145/2000064.2000108>.

(FREERTOS 2017) "**FreeRTOS - Real Time Operating System reference manual.**" In: 2017. Available in: <http://www.freertos.org/>. Access: February 2017.

GARIBOTTI, R., OST, L., BUSSEUIL, R., et al. "Simultaneous multithreading support in embedded distributed memory MPSoCs." In: 2013 50th ACM/EDAC/IEEE DESIGN AUTOMATION CONFERENCE (DAC), Austin, TX, 2013. **Proceedings...** pp. 1-7. Available in: <https://doi.org/10.1145/2463209.2488836>.

HAGHBAYAN, M. H., MIELE, A., RAHMANI, A. M., et al. "A lifetime-aware runtime mapping approach for many-core systems in the dark silicon era." In: 2016 DESIGN, AUTOMATION & TEST IN EUROPE CONFERENCE & EXHIBITION (DATE), Dresden, 2016. **Proceedings...** pp. 854-857.

HAMERSKI, J.C., ABICH, G., REIS, R., et al. "Publish-Subscribe Programming for a NoC-based Multiprocessor System-on-Chip." In: 2017 INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS (ISCAS), IEEE, 2017. **Proceedings...** (Accepted Paper).

(HEMPS 2017) "**HeMPS: Hermes Multiprocessor System on Chip.**" Available in: <http://www.inf.pucrs.br/hemps/>. Access: February 2017.

HOLT, J., AGARWAL, A., BREHMER, S., et al. "Software Standards for the Multicore Era," in **IEEE Micro**, vol. 29, no. 3, pp. 40-51, May-June 2009. Available in: <https://doi.org/10.1109/MM.2009.48>.

HU, W., TANG, X., XIE, B., et al. "An Efficient Power- Aware Optimization for Task Scheduling on NoC-based Many-core System." In: 2010 10th IEEE INTERNATIONAL CONFERENCE ON COMPUTER AND INFORMATION TECHNOLOGY (CIT), Bradford, 2010. **Proceedings...** pp. 171-178. Available in: <https://doi.org/10.1109/CIT.2010.67>.

(IMPERAS 2017) "**Imperas Revolutionizing Embedded Software Development.**" Available in: <http://www.imperas.com/>. Access: February 2017.

(ITRS 2015). "**International Technology Roadmap for Semiconductors.**" Available in: <http://www.itrs2.net/> Access: February 2017.

KERKHOFF, H. G., WAN, J., & ZHAO, Y. "Linking aging measurements of health-monitors and specifications for multi-processor SoCs." In: 2014 9th IEEE

INTERNATIONAL CONFERENCE ON DESIGN & TECHNOLOGY OF INTEGRATED SYSTEMS IN NANOSCALE ERA (DTIS), Santorini, 2014. **Proceedings...** pp. 1-6. Available in: <https://doi.org/10.1109/DTIS.2014.6850656>.

LORENZON, A. F., CERA, M. C., & BECK, A. C. S. "Investigating different general-purpose and embedded multicores to achieve optimal trade-offs between performance and energy." In: **Journal of Parallel and Distributed Computing**, Volume 95, September 2016, pp. 107-123, ISSN 0743-7315. Available in: <https://doi.org/10.1016/j.jpdc.2016.04.003>.

LUCAS, A. H., AMORY, A. M., & MORAES, F. G. "*Crosstalk Fault Tolerant NoC: Design and Evaluation*." In: 17th IFIP/IEEE INTERNATIONAL CONFERENCE ON VERY LARGE-SCALE INTEGRATION-SYSTEM ON A CHIP (VLSI-SoC), Springer Berlin Heidelberg, October 2009. **Proceedings...** pp. 81-93. Available in: <https://doi.org/10.1007/978-3-642-23120-9_5>.

MA, J., FU, F., LIU, Z., et al. "μC-OS II-based Operating System design for cluster in NoC-based MPSoC." In: 2013 IEEE INTERNATIONAL CONFERENCE ON SIGNAL PROCESSING, COMMUNICATION AND COMPUTING (ICSPCC), Kun Ming, 2013, IEEE. **Proceedings...** pp. 1-5. Available in: <https://doi.org/10.1109/ICSPCC.2013.6664028>.

MADALOZZO, G., DUENHA, L., AZEVEDO, R., et al. "Scalability evaluation in many-core systems due to the memory organization." In: 2016 IEEE INTERNATIONAL CONFERENCE ON ELECTRONICS, CIRCUITS AND SYSTEMS (ICECS), Monte Carlo, 2016. **Proceedings...** pp. 396-399. Available in: <https://doi.org/10.1109/ICECS.2016.7841216>.

MAHESHWARI, A., BURLESON, W., & TESSIER, R. "Trading off transient fault tolerance and power consumption in deep submicron (DSM) VLSI circuits." In: **IEEE Transactions on Very Large-Scale Integration (VLSI) Systems**, Volume 12, no. 3, pp. 299-311, March 2004. Available in: <https://doi.org/10.1109/TVLSI.2004.824302>.

MANDELLI, M. G., DA ROSA, F. R., OST, L., et al. "Multi-level MPSoC modeling for reducing software development cycle." In: 2013 IEEE 20th INTERNATIONAL CONFERENCE ON ELECTRONICS, CIRCUITS, AND SYSTEMS (ICECS), Abu Dhabi, 2013. **Proceedings...** pp. 489-492. Available in: <https://doi.org/10.1109/ICECS.2013.6815460>.

MANDELLI, M., CASTILHOS, G., SASSATELLI, G., et al. "A Distributed Energy-aware Task Mapping to Achieve Thermal Balancing and Improve Reliability of Many-core Systems." In: 2015 28th Symposium on Integrated Circuits and Systems Design (SBCCI), Salvador, Brazil, 2015. **Proceedings...** no. 13, pp. 13:1-13:7. Available in: <https://doi.org/10.1145/2800986.2800992>.

MANDELLI, M., OST, L., SASSATELLI, G., et al. "Trading-off system load and communication in mapping heuristics for improving NoC-based MPSoCs reliability." In: 16th INTERNATIONAL SYMPOSIUM ON QUALITY ELECTRONIC DESIGN (ISQED), Santa Clara, CA-USA, 2015. **Proceedings...** pp. 392-396. Available in: <https://doi.org/10.1109/ISQED.2015.7085457>.

MANDELLI, M. G. "**Exploration of Runtime Distributed Mapping Techniques for Emerging Large Scale MPSoCs.**" In: 2015, p. 134, Thesis (Doctorate), Faculdade de Informática – PUCRS, July 13 2015. Available in: <http://tede2.pucrs.br/tede2/handle/tede/6317>. Access: February 2017.

MARCON, C., AMORY, A., WEBBER, T., et al. "Phoenix NoC: A distributed fault tolerant architecture." In: 2013 IEEE 31st INTERNATIONAL CONFERENCE ON COMPUTER DESIGN (ICCD), Asheville, NC-USA, 2013. **Proceedings...** pp. 7-12. Available in: <https://doi.org/10.1109/ICCD.2013.6657018>.

MARTINS, A. L., SILVA, D. R., CASTILHOS, G. M., et al. "A method for NoC-based MPSoC energy consumption estimation." In: 2014 21st IEEE INTERNATIONAL CONFERENCE ON ELECTRONICS, CIRCUITS AND SYSTEMS (ICECS), Marseille, 2014. **Proceedings...** pp. 427-430. Available in: <https://doi.org/10.1109/ICECS.2014.7050013>.

MAYER-SCHÖNBERGER, V., & CUKIER, K. "**Big data**: A revolution that will transform how we live, work, and think." Publisher Houghton Mifflin Harcourt, Boston, New York, 2013, p. 244. ISBN 978-0-544-00269-2.

(MELLANOX 2016) "**Mellanox Multicore Processors.**" Available in: <http://www.mellanox.com/>. Access: February 2017.

MORAES, F., CALAZANS, N., MELLO, A., et al. "HERMES: an infrastructure for low area overhead packet-switching networks on chip." In: **INTEGRATION, the VLSI journal**, Elsevier 2004, volume 38, no. 1, pp. 69-93. https://doi.org/10.1016/j.vlsi.2004.03.003.

NOERGAARD, T. "**Embedded systems architecture**: a comprehensive guide for engineers and programmers." Publisher Elsevier (Newnes), Second Edition, Waltham, MA-USA, 2013, p. 652, ISBN 978-0-12-382196-6.

(OVP 2017) "**OVPsim Simulator 2017.**" Available in: <http://www.ovpworld.org/>. Access: February 2017.

PARK, D., NICOPOULOS, C., KIM, J., et al. "Exploring fault-tolerant network-on-chip architectures." In: INTERNATIONAL CONFERENCE ON DEPENDABLE SYSTEMS AND NETWORKS (DSN), Philadelphia, PA, 2006. **Proceedings...** pp. 93-104. Available in: <https://doi.org/10.1109/DSN.2006.35>.

POTHUKUCHI, R. P., ANSARI, A., VOULGARIS, P., et al. "Using multiple input, multiple output formal control to maximize resource efficiency in architectures." In: 2016 ACM/IEEE 43rd ANNUAL INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURE (ISCA), Seoul, 2016. **Proceedings...** pp. 658-670. Available in: <https://doi.org/10.1109/ISCA.2016.63>.

RADETZKI, M., FENG, C., ZHAO, X., et al. "Methods for fault tolerance in networks-on-chip." In: **ACM Computing Surveys**, July 2013, Volume 46, pp. 8:1-8:38, ISSN 0360-0300. Available in: <https://doi.org/10.1145/2522968.2522976>.

RUARO, M., CARARA, E. A., & MORAES, F. "Tool-set for NoC-based MPSoC debugging - A protocol view perspective." In: 2014 IEEE INTERNATIONAL

SYMPOSIUM ON CIRCUITS AND SYSTEMS (ISCAS), Melbourne VIC, 2014. **Proceedings...** pp. 2531-2534. Available in: <https://doi.org/10.1109/ISCAS.2014.6865688>.

RUARO, M., CHAMORRA, H., RUBIN, F., et al. "A data extraction and debugging framework for large-scale MPSoCs." In: 2016 IEEE INTERNATIONAL CONFERENCE ON ELECTRONICS, CIRCUITS AND SYSTEMS (ICECS), Monte Carlo, 2016. **Proceedings...** pp. 616-619. Available in: <https://doi.org/10.1109/ICECS.2016.7841277>.

SAHOO, S. S., KUMAR, A., & VEERAVALLI, B. "Design and evaluation of reliability-oriented task re-mapping in MPSoCs using time-series analysis of intermittent faults." In: 2016 DESIGN, AUTOMATION & TEST IN EUROPE CONFERENCE & EXHIBITION (DATE), EDA Consortium, Dresden, Germany, 2016. **Proceedings...** pp. 798-803.

SHAFIQUE, M., GARG, S., HENKEL, J., et al. “The EDA Challenges in the Dark Silicon Era: Temperature, Reliability, and Variability Perspectives.” In: PROCEEDINGS OF THE 51st ANNUAL DESIGN AUTOMATION CONFERENCE (DAC), ACM, San Francisco, CA-USA, 2014. **Proceedings...** pp. 185:1-185:6. Available in: <https://doi.org/10.1145/2593069.2593229>.

(SIMICS 2017) “**Simics Full System Simulator.**” Available in: <https://www.windriver.com/products/simics/>. Access: February 2017.

SINGH, A. K., SHAFIQUE, M., KUMAR, A., et al. “Mapping on multi/many-core systems: survey of current and emerging trends.” In: PROCEEDINGS OF THE 50th ANNUAL DESIGN AUTOMATION CONFERENCE(DAC), ACM, Austin, TX-USA, 2013. **Proceedings...** pp. 1:1-1:10. Available in: <https://doi.org/10.1145/2463209.2488734>.

TAYLOR, M. B. “Is dark silicon useful? Harnessing the four horsemen of the coming dark silicon apocalypse.” In: 2012 49th ACM/EDAC/IEEE DESIGN AUTOMATION CONFERENCE (DAC), San Francisco, CA-USA, 2012. **Proceedings...** pp. 1131-1136. ISSN 0738-100X.

ZENG, H., YOURST, M., GHOSE, K., et al. “MPTLsim: a cycle-accurate, full-system simulator for x86-64 multicore architectures with coherent caches.” In: **ACM SIGARCH Computer Architecture News**, 2009, volume 37, no. 2, pp. 2-9, ISSN 0163-5964. Available in: <https://doi.org/10.1145/1577129.1577132>.

Zhan, J., Xie, Y., & Sun, G. “NoC-Sprinting: Interconnect for fine-grained sprinting in the dark silicon era.” In: 2014 51st ACM/EDAC/IEEE DESIGN AUTOMATION CONFERENCE (DAC), San Francisco, CA-USA, 2014. **Proceedings...** pp. 1-6. Available in: <https://doi.org/10.1145/2593069.2593165>.

Zhang, Y., Peng, L., Fu, X., et al. “Lighting the dark silicon by exploiting heterogeneity on future processors.” In: PROCEEDINGS OF THE 50th ANNUAL DESIGN AUTOMATION CONFERENCE (DAC), ACM, Austin, TX-USA, 2013. **Proceedings...** pp. 82:1-82:7. Available in: <https://doi.org/10.1145/2463209.2488835>.

(µC-OS 2017) "**µC-OS II The Real-Time Operating System.**" Available in: <https://www.micrium.com/>. Access: February 2017.

APPENDIX A – PUBLICATIONS OF THE AUTHOR

The results presented in this dissertation were published in two conferences with international academic recognition. Table X presents the papers, with the respective conferences name and the year of publication. The first paper underlies the validation of the proposed FreeRTOS extension under large scale MPSoCs with the proposed framework (ABICH et al., 2016). The second exploits the Publisher-Subscriber protocol developed and validated with the proposed extension and framework (HAMERSKY et al., 2017).

Table XI: List of Publications related to this dissertation results.

	Title	**Description**
1	*Extending FreeRTOS to Support Dynamic and Distributed Mapping in Multiprocessor Systems*. **Abich, G.**, Mandelli, M. G., Rosa, F. R., Moraes, F., Ost, L. & Reis, R. In: IEEE International Conference on Electronics, Circuits and Systems (ICECS), 2016. (ABICH et al., 2016)	Proposed Framework and FreeRTOS Extension presented in Chapters 3 and 4
2	*Publish-Subscribe Programming for a NoC-based Multiprocessor System-on-Chip.* Hamerski, J.C., **Abich, G.**, Reis, R., Ost, L. & Amory, A. In: IEEE International Symposium on Circuits and Systems (ISCAS), 2017. (HAMERSKY et al., 2017)	Integration of Publisher-Subscriber Protocol presented in Chapter 5 Section 5.4

Source: Author