Better than Worst-Case Design for Streaming Applications under Process Variation

Better than Worst-Case Design for Streaming Applications under Process Variation

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof.ir. K.Ch.A.M. Luyben,
voorzitter van het College voor Promoties
in het openbaar te verdedigen

op vrijdag 13 december 2013 om 12.30 uur

door

Davit Mirzoyan

Master of Science
Kungliga Tekniska Högskolan, Zweden
geboren te Yerevan, Armenië

Dit proefschrift is goedgekeurd door de promotor:

Prof.dr. K.G.W. Goossens

Copromotor:

Dr. K.B. Åkesson

The work in this thesis is supported by the Dutch government in their STW research program within the NEST project 10346.

Printing by Ipskamp Drukkers.

Dedicated to my family
for their unfailing
encouragement and support

Acknowledgments

I am glad to have this opportunity to express my gratitude to all who directly or indirectly had their contribution in this work. I would like to thank Prof. Koen Bertels at Delft University of Technology for all the practical support during these four years. I extend my gratitude to Ms. Lidwina Tromp, the secretary at the Computer Engineering laboratory, who has always been very helpful with practical matters. I acknowledge fellow Ph.D. students Andrew Nelson, Ashkan Beyranvand Nejad, a former postdoctoral researcher Dr. Anca Molnos, and all the staff at the Computer Engineering laboratory. Being affiliated with Delft University of Technology, my workplace was arranged at Eindhoven University of Technology, where I have always worked. Over the course of these years, I have been fortunate to work in such a friendly environment, as in the Electronic Systems group. My deep appreciation to the group for all the hospitality that I have been shown.

I am sincerely grateful that four years ago Prof. Kees Goossens gave me the opportunity to work with him. I have been fortunate to have a mentor with such a strong work ethic. My deepest gratitude to Dr. Benny Åkesson, my co-supervisor and once a student of Prof. Goossens. Working with him has been a real pleasure. I have had the opportunity to enjoy beautiful Porto, Portugal with Benny on a business travel, which turned out to be very memorable, as we can agree. I would like to acknowledge Dr. Sander Stuijk, who always found time to give me insight into the theory of data flow graphs. Discussions with him have always been fruitful. I extend my gratitude to Prof. José Pineda de Gyvez for the circuit-level discussions that he has provided on multiple occasions. They were always helpful. My special appreciation goes to Ms. Marja de Mol - Regels and Ms. Rian van Gaalen, the truly kind secretaries at the Electronic Systems group. I would like to thank Rian for organizing the "uurtje Nederlands", an hour of weekly delightful dutch lessons, for us and for all the enthusiasm she has shown on different occasions.

I have always enjoyed the working environment in our office, with great office mates Firew Siyoum, Manil Dev Gomony, Cedric Nugteren, Roel Jordans and Luc Vosters. During these four years, we have had all the interesting discussions covering a variety of topics, such as science, technology, work, etc. I would like to add that I feel fortunate to have known Firew Siyoum, Manil Dev Gomony and Karthik Chandrasekar during these years. Last but not the least, I extend my gratitude to all the staff members at the Electronic Systems group, who have made these years more enjoyable.

From the bottom of my heart, I would like to thank my family, my mother Juletta Makaryan, my father Razmik Mirzoyan, and my lovely sister Yevgeniya Harutyunyan, for all the support and encouragement over the course of these years.

I cannot finish writing these acknowledgments without expressing my deepest gratitude and admiration towards my heavenly Father, the Lord Jesus Christ, for his unconditional love and boundless riches for those who love him; his mercies are new every morning. I thank him for the increasing amount of strength he has given me; as it is written "Even youths grow tired and weary, and young men stumble and fall; but those who hope in the Lord will renew their strength. They will soar on wings like eagles; they will run and not grow weary, they will walk and not be faint." Isaiah 40:30-31.

Summary

Modern multimedia and wireless applications require considerable computational power due to an increasing amount of functionality added to these applications. Additionally, many portable consumer electronics impose requirements on low power consumption for long battery life. To meet these requirement, streaming applications are implemented on a multi-processor system-on-chip (MPSoC), where multiple (slower) processing cores exploit task and data-level parallelism to increase performance. The hardware components inside an MPSoC are usually connected to each other by a scalable network on chip (NoC). To be able to analyze the timing of an application implemented on an MPSoC and provide timing guarantees at design time, all MPSoC components must be predictable. To remove the bottleneck of global clock-tree routing and reduce a major source of power consumption, multi-processor systems are implemented with the globally asynchronous, locally synchronous (GALS) design style. GALS architectures enable the use of voltage-frequency islands to additionally reduce power consumption by scaling the frequency (voltage) of individual components in an MPSoC.

To reduce circuit area and thus integrate more functionality on a chip die, CMOS technology has traditionally been scaled down. However, scaling in the nanoscale era has brought significant variability in the manufacturing process. This variability or inability to precisely control the manufacturing process results in significant variation in the maximum supported frequency of hardware components in a multi-processor system. Given this variation, it becomes difficult to provide timing guarantees for an application mapped to a multi-processor system. As a solution, circuits are conventionally implemented with design margins or guard-bands to guarantee the target frequency of hardware components inside an MPSoC. Under this design paradigm, known as worst-case design, an application sees the hardware components in a multi-processor platform with deterministic minimum frequencies, leading to a mapping of the application tasks to the hardware components, such that a certain timing requirement (e.g. throughput or latency) imposed on the application is satisfied. However, worst-case design results in a considerable up-sizing of circuit area and in increased power consumption. Because of this, the benefits of technology scaling can be lost with worst-case design.

This thesis deals with the design of real-time streaming applications constrained by a throughput requirement with reduced design margins, referred to as better than worst-case design. With better than worst-case design, the area, the power consumption and the leakage of a circuit are reduced. Smaller circuit area and thus die size results in a larger number of gross dies on a wafer. However, the target maximum supported frequency of hardware components in a multi-processor system is not guaranteed anymore, and the spread in the maximum supported frequency of hardware components needs to be considered when mapping an application to the platform. The goal of this thesis is to maximize the number of good dies that satisfy the throughput requirement of a real-time streaming application.

To this end, a design flow consisting of three stages is presented in this thesis. In the first stage of the flow, a streaming application is allocated to a NoC-based multiprocessor system with voltage-frequency islands, where each hardware component is characterized by process-induced frequency variations. The goal is to maximize the timing yield, which is a system-level metric quantifying the percentage of manufactured chips that satisfy the timing requirement of the application. Maximizing the timing yield is essential as the number of good dies is given by the product of the number of gross dies and timing yield. Two mapping approaches are possible: single and multiple bindings for manufactured chips. Multiple bindings can exploit the availability of within-die variation-induced faster and slower processing cores on the same chip by adopting the allocation of application tasks. Following the mapping stage, trade-offs between the number of voltage-frequency islands, as well as the number of clock-frequency levels provided to each island, (area and power consumption cost) and timing yield are performed. Having decided on the binding (or a set of bindings), a set of voltage-frequency island configurations and a number of clock-frequency levels per island, the number of good dies on a wafer is evaluated for different guard-band reduction values in the third and final stage of the flow. The reduction in guard-bands providing the most good dies is selected. We show on both synthetic and real applications that the proposed design flow can increase the number of good dies by up to 9.6% and 18.8% for designs with and without fixed SRAM and IO blocks, respectively.

Samenvatting

Moderne multimedia en draadloze applicaties vereisen aanzienlijke rekenkracht als gevolg van hun toenemende functionaliteit. Bovendien eist veel draagbare consumentenelektronica een laag stroomverbruik om een lange levensduur van de batterij te garanderen. Om aan deze eisen te voldoen worden streaming-applicaties geïmplementeerd op een multi-processor system-on-chip (MPSoC), waarbij meerdere (tragere) processorkernen taak en data-parallellisme benutten om de prestaties te verhogen. De hardwarecomponenten in een MPSoC zijn meestal met elkaar verbonden door middel van een schaalbaar network-on-chip (NoC). Alle MPSoC onderdelen moeten voorspelbaar zijn om tijdsanalyse van applicaties te kunnen geven en om tijdsgaranties te kunnen geven tijdens de ontwerpfase. Deze systemen worden ontworpen volgens het globally asynchronous, locally synchronous (GALS) principe om zo de bottleneck van een globale klokfrequentieboom te omzeilen en om een belangrijke bron van vermogen te verlagen. GALS architecturen maken het mogelijk om frequentie-spanningseilanden te vormen die het vermogensgebruik verder verminderen door de frequentie (of spanning) van componenten in een MPSoC individueel te schalen.

Het schalen van CMOS technologie heeft er voor gezorgd dat chip oppervlakte verkleind werd om zodoende meer functionaliteit op een enkele chip te kunnen integreren. Echter, het schalen naar nanometer-technologie heeft aanzienlijke veranderlijkheid gebracht in het fabricageproces. Deze veranderlijkheid (het onvermogen om het productieproces te controleren) resulteert in aanzienlijke variaties in de maximaal ondersteunde frequentie van componenten in een multi-processor systeem. Door deze variaties wordt het moeilijk om tijdsgaranties te geven voor applicaties die afgebeeld zijn op zulke systemen. Als oplossing worden chips gebruikelijk uitgerust met marges (zogenoemde guard-bands) om vooraf vastgestelde frequenties toch te kunnen garanderen. Volgens deze ontwerpstijl (worst-case ontwerp) ziet een applicatie dat alle componenten van een multi-processor platform deterministische frequenties hebben. Dit leidt er toe dat een afbeelding van applicatietaken op de hardwarecomponenten aan vooraf bepaalde tijdsvereisten kan voldoen (bijvoorbeeld doorvoersnelheid of latentie). Echter, de worst-case ontwerpstijl resulteert in aanzienlijk grotere chips en in een verhoogd stroomver-

bruik. Hierdoor kunnen de voordelen van het schalen van CMOS weer verloren gaan.

Dit proefschrift behandelt het ontwerpen van real-time streaming-applicaties die beperkt zijn door een doorvoersnelheidseis met verkleinde marges, bekend als de beter dan worst-case ontwerpstijl. Een dusdanig ontwerp verlaagt het oppervlaktegebruik en vermindert het stroomverbruik en de lekstromen. Een kleiner oppervlaktegebruik resulteert in een groter aantal chips per wafer. Echter worden de vooraf vastgestelde frequenties niet meer gegarandeerd en moet de spreiding van de maximaal ondersteunde frequentie in overweging genomen worden wanneer een applicatie wordt afgebeeld op een platform. Het doel van dit proefschrift is om het aantal goede chips dat aan de doorvoersnelheidseis van een real-time streaming-applicatie voldoet te maximaliseren.

Hiertoe wordt een ontwerp dat bestaat uit drie fases gepresenteerd. In de eerste fase wordt een streaming-applicatie toegewezen aan een NoC-gebaseerd multi-processor systeem met frequentie-spanningseilanden, waarbij elke hardwarecomponent gekarakteriseerd wordt door proces-gestuurde frequentievariaties. Het doel is om de yield te maximaliseren: een systeemniveau metriek die kwantificeert welk percentage van de geproduceerde chips aan de tijdsvereiste van de applicatie voldoen. Het maximaliseren van de yield is essentieel, omdat het aantal goede chips wordt bepaald door het product te nemen van het bruto aantal chips en de yield. Twee afbeeldingstechnieken zijn mogelijk: ofwel een enkele afbeelding ofwel meerdere afbeeldingen. Meerdere afbeeldingen maken de toekenning van applicatietaken adaptief. Hierdoor kan rekening gehouden worden met de aanwezigheid van variaties in snelheden van processorkernen binnen een chip. Volgend op de toekenningsfase worden de yield en de compromissen (oppervlaktegebruik versus vermogensgebruik) van het aantal frequentie-spanningseilanden en frequentieniveaus bepaald. In de derde en laatste fase van het ontwerp wordt het aantal goede chips geëvalueerd voor verschillende guard-band reducties. Uiteindelijk wordt een guard-band gekozen die het grootste aantal goede chips oplevert. We tonen aan dat de gepresenteerde ontwerptechniek voor zowel synthetische als echte applicaties het aantal goede chips kan verhogen met 9.6% of 18.8%, waarbij het eerste resultaat er vanuit gaat dat er vaste SRAM en IO-blokken zijn die niet meeschalen met guard-band reducties.

Content

List of Figures

List of Tables

1 Introduction

1.1 Real-time embedded systems

Embedded systems contribute to most aspects of our daily activities. They surround us and provide us with entertainment, daily information and assist us in completing our every day tasks. Examples of embedded systems are mobile phones, digital cameras, TV sets, global positioning systems, air traffic management, etc. An important portion of embedded systems is represented by embedded multimedia and wireless systems. Applications in these systems work on streams of audio and video data, and are termed *streaming applications*. Examples of streaming applications are video decoding (encoding) [63, 82] from the multimedia domain and software defined radio [59] from the wireless domain. Streaming applications are usually constrained by a throughput requirement associated with user perception. An example throughput requirement for video decoding is the number of frames per second. Timing requirements exist in *firm* and *soft real-time application classes*. In firm real-time applications, such as software-defined radio, violations of the timing requirement are not allowed. Soft real-time applications, such as such as video decoding, are characterized by less stringent timing requirements. In such applications missing a deadline causes only a performance degradation, often evaluated through some quality-of-service parameter. There are also *best-effort applications*, which do not have any timing requirements. An example best-effort application is a graphical user interface. Although there are no timing requirements set on the applications of this class, high performance is preferred by the user.

To enhance user experience, an increasing amount of data must be processed (e.g. to achieve higher video resolutions) and more computationally intensive coding schemes must be implemented. As a result, an increasing computational capability to implement such applications is required. Additionally, many portable consumer electronics, such as mobile phones, digital cameras, and tablets, impose requirements on power consumption for longer battery life. To meet the increasing demand on computational capability and low power consumption, such applications are implemented on a multi-processor system-on-chip (MPSoC) [35, 49, 87, 95], where multiple (slower) processing cores exploit task and data-level parallelism to increase performance without increasing power consumption.

The components inside a multi-processor system were traditionally connected to each other by a bus. However, traditional buses do not provide scalable interconnection. For this reason, a paradigm shift towards network on chips (NoC) based interconnection inside multi-processor systems has been seen in recent years [17]. All hardware components in an MPSoC must be *predictable*, such that the timing of an application implemented on the MPSoC can be formally analyzed and timing guarantees can be given at design time. Present-day MPSoCs are implemented by means of the globally asynchronous, locally synchronous (GALS) design style [48, 60], which was introduced to alleviate the bottleneck of global clock distribution and reduce the related major source of power consumption in multi-processor systems. The GALS architecture is composed of synchronous blocks, communicating with each other on an asynchronous basis. The concept of voltage-frequency islands (VFI), within the GALS design paradigm, enables scaling the frequency (voltage) of each individual hardware component (clusters of components) in a multi-processor system to further reduce power consumption.

Due to rapid technology changes, consumers replace old products with new ones much more frequently. For example, two major products are released annually by mobile phone manufacturers, such as Apple and Samsung. These products offer more and more functionality, leading to a considerable increase in design effort. This imposes requirements on short time to market of products. To address this issue, a *platform-based design* methodology has been proposed [4, 69]. A platform consists of multiple hardware and software components specific to a particular application domain. The software components are application software, middleware and operating systems used for programming the hardware components. A platform, as a result of integration of different components, serves as a starting point in application development. Which components to integrate depends on the requirements imposed on the application [94]. Given a platform, an application, which can be partially specified in software and partially in hardware, is mapped to the multi-processor platform, such that the timing requirements imposed on the application are satisfied.

1.2 Manufacturing process variation

To reduce the power consumption of consumer electronics and decrease the area of integrated circuits, thus enabling integration of more functionality on a chip die, transistor feature sizes have traditionally been scaled down . An observation that the integration density of integrated circuits doubles approximately every two years was made by Gordon Moore in 1956. This observation, known as Moore's law [56, 57], still holds up to this day. However, scaling CMOS technology into nanometer feature size nodes has made it practically impossible to precisely control the manufacturing process. Major sources of manufacturing process variation are random dopant fluctuations and sub-wavelength lithography for patterning transistors [11]. Process variation results in variability in key design parameters, such as transistor channel length and threshold voltage, and interconnect width [12,61]. Parameter variability, in turn, influences circuit speed [13,20], power consumption and leakage [51]. Considerable variability of up to 50% in the longest path delay of a processor is reported in available literature [20,51]. Process variation can be categorized into *die-to-die* and *within-die* variations. Die-to-die variation acts globally on the entire chip die, affecting parameters of all devices (i.e. transistors) on the die identically. Die-to-die variation is seen between dies within a wafer and between dies of different wafers (due to wafer-to-wafer variation). In contrast, within-die variation affects parameters of devices on the same die differently. It can be classified into systematic and random components. Systematic within-die variation exhibits spatial correlation, such that nearby devices possess similar parameter values due to high correlation, which dies out quickly as a function of distance on a die [22]. Random within-die variation is purely random from device to device on a die. The impact of within-die variation is expected to worsen as technology scales. Both die-to-die and within-die variations impact the maximum supported frequency of hardware components inside an MPSoC. Different (identically designed) hardware components on the same die, as well as the same hardware component across different dies, can have different maximum supported frequencies.

As a solution to process-induced variations, circuits are conventionally implemented with conservative design margins, or guard-bands, to guarantee the target frequency of each hardware component in manufactured multi-processor chips. This is known as *worst-case design.* Circuit guard-banding is typically done by using corner-files during the design and verification stages. These files describe the worst-case and best-case delay values of standard-cells, corresponding to *slow and fast process corners*, respectively. Under worst-case design, design synthesis is performed for the slow process corner, while the fast process corner is used in the design verification stage to correct possible set-up timing violations. Corner files lack detailed information on within-die variation. Instead, on-chip timing variation margin is added during the design verification stage to account for within-die variation with a pessimistic assumption that all the devices within a die are performing according to their worst case under slow process conditions.

Figure 1.1 shows a qualitative example of the process-induced spread in frequency of a hardware component. As shown, almost all manufactured instances of the hardware component meet the target frequency f_{tg} under worst-case design (WCD). Therefore, from the perspective of an application, which can be partially defined in software and partially in hardware, the hardware components in a multi-processor platform have deterministic minimum frequencies, leading to a mapping of the application tasks to the hardware components in the multi-processor platform, such that a certain timing requirement (e.g. throughput or latency) imposed on the application is satisfied. However, worst-case design has a considerable impact on circuit area, power consumption and leakage [33, 40]. Consider the qualitative circuit area and clock period trade-off curves for slow, nominal and fast process corners for a generic circuit, illustrated in Figure 1.2. As can be seen, larger circuit up-sizing is performed for a lower clock period (higher performance) under worst-case design. The added timing variation margin to account for within-die process variation also results in increased circuit area, as illustrated in the figure. Due to increased circuit area, power consumption and leakage, the benefits of technology scaling can be lost.

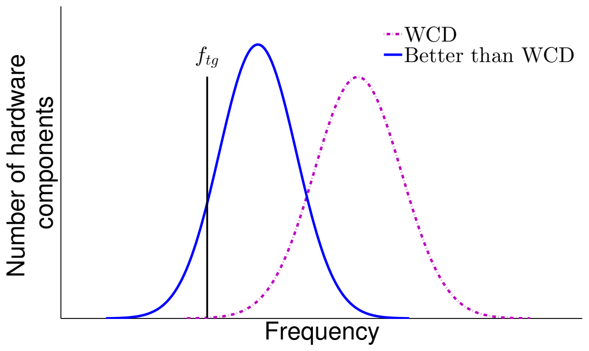


Figure 1.1: Process-induced spread in frequency of a hardware component for worst-case and better than worst-case designs, where f_{tg} is the target frequency.

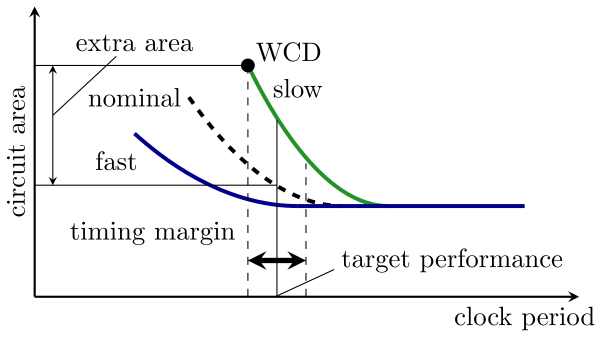
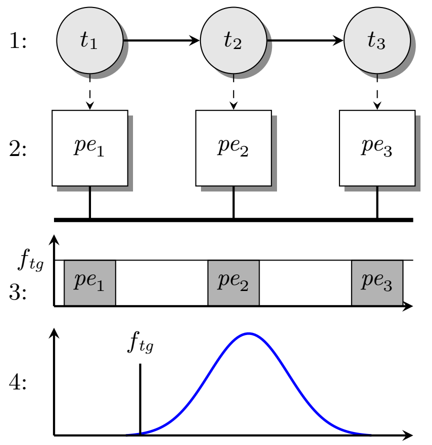


Figure 1.2: Circuit area and clock period trade-off curves for slow, nominal and fast process corners. The timing margin accounting for within-die process variation is shown by an arrow.

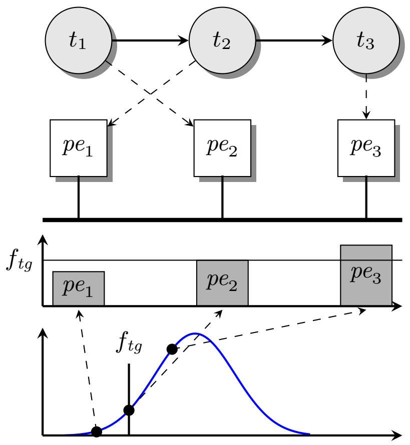
1.3 Problem statement

The trends outlined in Sections 1.1 and 1.2 show that the requirements on computational capability and low energy consumption of multimedia and wireless applications are increasing. Designers use multi-processor systems and platform-based design to deal with the increasing complexity of systems and to shorten time to market. As a solution to increasing process-induced variation in the frequency of hardware components in a multi-processor system, a worst-case design approach is adopted. This results in a considerable increase in circuit area and power consumption, and can lead to a loss in the benefits of technology scaling. This thesis deals with the design of real-time streaming applications constrained by a throughput requirement with reduced design margins, referred to as better than worst-case design. With better than worst-case design, the area, the power consumption and the leakage of a circuit are reduced. Smaller circuit area and thus die size results in a larger number of gross dies on a wafer. However, the target maximum supported frequency of hardware components in a multi-processor system is not guaranteed anymore. The spread in the maximum supported frequency of an example hardware component with respect to its target speed, under better than worst-case design, is illustrated in Figure 1.1. *The goal of this thesis is to maximize the number of good dies that satisfy the throughput requirement of a real-time streaming application.* This goal is accomplished by providing algorithms to exploit process-driven variation in frequency of hardware components in a multi-processor system in the application mapping and voltage-frequency island partitioning stages. The main concept of the work is illustrated in Figure 1.3, where an application with a throughput requirement is mapped to a multi-processor platform under both worst-case and better than worst-case designs. With worst-case design (Figure 1.3a), the target speed of the processing cores (denoted by *pe*) is guaranteed at the cost of large area. All processing cores are operated at their target speeds, providing sufficient computational capability for the application to satisfy its throughput requirement (for a given mapping). With better than worst-case design (Figure 1.3b), the area of the processing cores is reduced, but the target speed is not guaranteed anymore. However, there may be processing cores with higher and lower than the target speeds on the same chip die due to the impact of within-die variation. Operating the processing cores at their corresponding speeds and using the available mapping flexibility, the allocation of the tasks of an application to the processing cores can be tailored for each specific chip, such that the throughput requirement is satisfied whenever possible. This can result in a larger number of good dies, given that a larger number of gross dies are available on a wafer due to smaller die area.

The building blocks of this thesis are a multi-processor platform, a streaming application and a model of computation (MoC) that can capture a streaming multimedia or wireless application along with its mapping to a multi-processor platform. Using the model of computation, timing guarantees can be given by performing design-time analysis. Due to process variation, the frequency of hardware components is given by a probability distribution. A network on chip provides interconnection between hardware components in the multi-processor system. The hardware components are partitioned into voltage-frequency islands. A set of clock-frequency levels are provided to each island. We introduce a metric termed *timing yield* quantifying the percentage of manufactured chips satisfying the throughput requirement imposed on an application. The number of good dies is given by the product of timing yield and the number of gross dies on a wafer. Therefore, the algorithms presented in this thesis aim at designing systems with high timing yield, such that the number of good dies is maximized.



(a) Worst-case design. The processing cores in the platform are operated at their target speeds (3:). For a certain mapping, the throughput requirement is satisfied.



(b) Better than worst-case design. There may be cores with higher and lower than the target speeds on a die (4:). The mapping can be adjusted for each specific chip, such that the throughput requirement is satisfied whenever possible. Given a larger number of dies on a wafer (smaller area (2:)), the number of good dies can be maximized.

Figure 1.3: An application (1:) constrained by a throughput requirement is allocated to a multi-processor platform (2:) under worst-case and better than worst-case designs.

1.4 Overview of solution

To address the problem in Section 1.3, this thesis proposes a design flow, called better than worst-case design. This section provides a brief overview of the flow. The design flow is shown in Figure 1.4. It assumes that a streaming application that has to be implemented on a multi-processor platform is modeled by a synchronous data-flow (SDF) graph [36, 37]. SDF graphs are well-suited for modeling and analysis of streaming applications, and have multiple efficient techniques for throughput computation [79]. The hardware platform is a NoC-based multi-processor platform with voltage-frequency islands, as introduced in Section 1.3. The SDF graph of the application includes information on the number of clock cycles it takes to execute each actor (modeling computation) on each processing element in the multi-processor platform and the amount of data sent between actors. This SDF model of the application is decoupled from the process-induced variation in the frequency of hardware components in the platform.

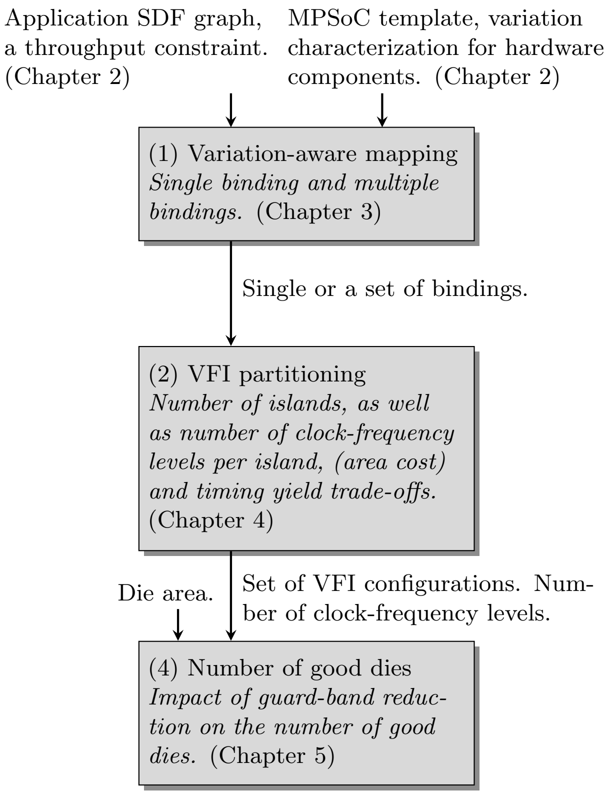

Application SDF graph, a throughput constraint. (Chapter 2)
MPSoC template, variation characterization for hardware components. (Chapter 2)
(1) Variation-aware mapping *Single binding and multiple bindings.* (Chapter 3)
Single or a set of bindings.
(2) VFI partitioning *Number of islands, as well as number of clock-frequency levels per island, (area cost) and timing yield trade-offs.* (Chapter 4)
Die area.
Set of VFI configurations. Number of clock-frequency levels.
(4) Number of good dies *Impact of guard-band reduction on the number of good dies.* (Chapter 5)

Figure 1.4: Design flow.

In Step 1 of the flow, the application is allocated to the platform, such that the timing yield is maximized. For this purpose, both exhaustive and heuristic mapping algorithms are proposed in this thesis. Two different mapping approaches are presented: *single-binding* and *multiple-bindings*. With the single binding mapping approach, only a single binding for all manufactured chips is derived. In contrast, with the multiple-bindings mapping approach, a set of bindings are selected for the manufactured chips. These bindings are stored and based on each manufactured chip, a binding is selected at an initial run-time configuration stage, such that the throughput of the application is maximized. Multiple bindings always result in equal or higher timing yield. To compute the timing yield, a characterization of the hardware multi-processor platform in terms of possible clock-frequency sets for processing elements and the network on chip is performed. The probability that the processing elements and the network on chip are operated at a certain set of clock frequencies is computed. The throughput of the mapped application for the different clock frequency sets is analyzed by constructing another model of the application, which captures the resource allocation for the application on the platform. The actors in this graph are characterized by execution times in *seconds*. Therefore, this model of the application is not decoupled from the variation in frequency of hardware components.

From the first stage of the flow, a single or a set of bindings are derived that result in maximized timing yield. Using these bindings, trade-offs between the number of voltage-frequency islands, as well as the number of clock-frequency levels per island, and timing yield are performed in the second stage of the flow (Step 2). The higher the number of islands and clock-frequency levels per island, the higher the additional area (and power consumption) cost of clock-generation units associated with the islands is. Therefore, by reducing the number of islands, the die size becomes smaller, resulting in more gross dies on the wafer. On the other hand, a reduction in the number of voltage-frequency islands and clock-frequency levels for each island may lead to decreased timing yield. This thesis introduces heuristic algorithms to perform voltage-frequency island partitioning, such that the timing yield is maximized for a given number of islands. Algorithms considering both a single and multiple bindings are proposed. The result of the second stage of the flow is a set of voltage-frequency island configurations and a number of clock-frequency levels for the islands, such that high timing yield is provided.

In the final stage of the flow (Step 3), the change in the number of good dies due to reducing circuit guard-bands is evaluated. In this stage, the mapping of the application to the platform in terms of a single or a set of bindings, the set of voltage-frequency island configurations and the number of clock-frequency levels for each island are known. The number of good dies are evaluated for a set of guard-band reduction values, such that the reduction providing the highest number of good dies is selected. In this stage, characterization of variation in the frequency of hardware components for each guard-band reduction value must be provided. Additionally, the circuit (die) area reduction factor due to reduced guard-bands has to be known.

1.5 Contributions

This thesis makes five contributions to develop real-time streaming applications, constrained by a throughput requirement on a multi-processor system under better than worst-case design (i.e. with reduced design margins).

1. A *formal framework* is presented to estimate the probability distribution of application throughput in a NoC-based multi-processor system with voltage-frequency islands in the presence of process-induced frequency variations of hardware resources (Chapter 2). Both *within-die and die-to-die variations* are considered. Any set of clock-frequency levels can be specified per VFI domain. We use synchronous data-flow (SDF) to model a streaming application mapped to an MPSoC. The novelty of our SDF formulation lies in the *explicit modeling* of software execution in terms of *clock cycles* (which is independent of the frequency variation of hardware components), and in terms of *seconds* (which does depend on the frequency variation of hardware components), which are linked by an explicit *binding.* This modeling allows a system designer to analyze the throughput of an application mapped to an MPSoC in the presence of process-driven variations. An earlier version of this work was published in [52–54].

2. *Optimal and heuristic mapping algorithms* are proposed to map streaming applications to a NoC-based multi-processor system with voltage-frequency islands under within-die and die-to-die process-driven variations (Chapter 3). We differentiate *best-effort, firm real-time and soft real-time* application classes, and define an optimization criteria for each of them. *Single-binding* and *multiple-bindings* mapping approaches are presented. With the single-binding mapping approach, a single binding for all manufactured chips is computed, while a set of bindings are derived with the multiple-bindings approach. At an initial run-time configuration stage, the right binding that maximizes throughput for a particular chip is selected. This work was published in [52, 53].

3. Heuristic algorithms to partition processing elements in a NoC-based multi-processor system into voltage-frequency islands for maximized timing yield in the presence of die-to-die and within-die variations are presented (Chapter 4). Algorithms considering both a single and multiple bindings for manufactured chips are given. An earlier version of this work was published in [54]

4. A demonstration on case-studies is presented showing how the framework proposed in this thesis can be used to estimate the impact of reducing circuit design margins on the number of good dies that satisfy the throughput requirement of a real-time streaming application (Chapter 5). We show for both synthetic and several real applications that the proposed design flow can increase the number of good dies per wafer, compared to conventional worst-case design.

5. All the presented algorithms and the formal models have been implemented in `C++` in the publicly available SDF For Free (SDF^3) tool-kit [72, 81].

2 Formal Modeling

To analyze the performance of an application mapped to a multi-processor platform at design time, a model of computation is required. The model needs to capture the application, the platform and the mapping of the application to the platform. The impact of process variation on the hardware resources (i.e. processing elements, routers, network interfaces and links) in the platform also needs to be captured in the model. This chapter introduces the formal models that are used throughout this thesis. We start by defining a hardware multi-processor platform as a *platform graph*. We present how the modeling of variation in the maximum supported frequency of hardware resources due to process variation is performed. Based on this variation, a characterization of the platform graph in terms of possible clock-frequency sets for processing elements and the interconnect is performed. The methodology to compute the probability that the processing elements and the interconnect are operated at a certain set of clock frequencies is presented. Later, an SDF model of a streaming application, named a *resource-aware application graph*, is introduced. This model is unaware of the binding of application actors to processing elements, and is hence decoupled from hardware variation. Finally, we define another SDF model of the application, coined as a *bound application graph*. This graph captures the binding of a resource-aware application graph to a platform graph. We describe how resource allocation is modeled in a bound application graph. This model is used to perform timing analysis of the mapped application. While a resource-aware application graph describes performance in terms of execution time in cycles, the essence of a bound application graph is that it considers performance in terms of execution time in seconds. This allows us to take process-induced variation in the frequency of processing elements into account. The presented techniques are general and apply to any system that implements the models described in this chapter. Examples of such systems are CoMPSoC [25], CA-MPSoC [73], Daedalus^{RT} [5], MAMPS [34], and systems by NXP [59] and STE / Ericsson [84].

2.1 Platform graph

The template of a hardware multi-processor platform used in this thesis and referred to as a platform graph is illustrated in Figure 2.1. It consists of generic processing elements, such as processors, DSPs, or hardware accelerators, connected to each other by a network on chip (NoC), later referred in this thesis as an interconnect. Processing elements are denoted by *pe*. We assume an arbitrary topology interconnect, which consists of routers, denoted by *rt*, network interfaces, denoted by *ni*, and unidirectional links, denoted by *lk*. Routers are connected to each other and to network interfaces by links. The interconnect provides lossless and ordered data transmission. Each processing element is connected to a single network interface in the interconnect. It is assumed that the network interfaces sit close to processing elements, and that the connections between processing elements and network interfaces do not introduce any delay. The path from a network interface to another network interface in the interconnect is referred to as *a connection*. A connection provides a certain maximum bandwidth (in bytes per cycle) assuming that all resources on the connection are reserved. It also has a certain hop count, given by the number of routers on the path. For constructing a connection model in an application SDF graph, presented later in Section 2.5.1, we require that any arbitration point in a connection can be modeled as a *latency rate server* [78], independent of other connections. Note that any starvation free arbitration can be modeled by a latency rate server. Examples of interconnects that fulfill these requirements are [21, 26, 27, 32, 39, 50, 70, 77, 83, 89, 90, 92]. In this thesis we assume a time-division multiplexing (TDM) arbitration policy (although other arbitration policies can be used). The arbitration for the shared router network is performed in network interfaces, which packetize the transactions from the processing elements and inject them into the router network as *flow control digits (flits)*. All network interfaces in the interconnect have the same slot table size (in number of slots). The injection of flits, regulated by the TDM tables, is done such that no two flits ever arrive at the same link at the same time. Therefore, the flits are forwarded without arbitration in the interconnect. Resource reservation on a connection is performed by allocating a number of slots in the TDM slot table. This provides a certain minimum bandwidth and maximum latency on the connection, as described in more details in Section 2.5.1. Network-on-chips that provide the described properties are Æthereal [26], Aelite [27], dAElite [77], Nostrum [50], SoCBUS [92], SurfNoC [89], and NoCs given in [70], [90], [21] and [83]. We formally define an interconnect in Definition 1. We refer to a processing element, a router, a network interface and a link in a platform graph as *a (hardware) resource* . As such, the union of the sets of processing elements, routers, network interfaces and links represents the set of all resources in the platform graph (Definition 2). The multi-processor platform is given by a globally asynchronous, locally synchronous (GALS) architecture [48], where the processing elements and the interconnect are partitioned into voltage-frequency islands (VFI). The interconnect is placed in an separate VFI, and thus the resources in the interconnect belong to that island. The set of voltage frequency islands is denoted by FI. Communication between the processing elements and the interconnect is accomplished by means of mixed-clock first-in-first-out (FIFO) buffers, which are part of network interfaces. In Figure 2.1, the separation between clock domains is shown by the dotted lines. A clock-generation unit (CGU) that provides a set of discrete clock-frequency levels, is dedicated to each voltage-frequency island. The formal definition of a platform graph gp is given in Definition 3. The set of all platform graphs is denoted by GP. The multi-processor platform depicted in Figure 2.1 is partitioned into three islands, namely fi_1 and fi_2 comprising processing elements pe_1 and pe_2, respectively, and fi_3 consisting of the interconnect.

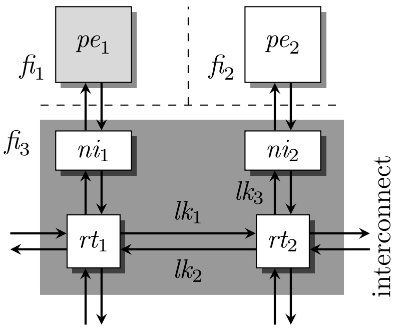


Figure 2.1: The template of a multi-processor platform consisting of processing elements connected to each other by an interconnect. The processing elements and the interconnect are placed in different voltage-frequency islands. The separation between clock domains is shown by the dotted lines.

Definition 1. *(Interconnect) An interconnect noc is a 6-tuple* $\langle RT, NI, LK, \eta, sz_{tb}, sz_{fl}\rangle$ *consisting of a set RT of routers, a set NI of network interfaces, a set LK of links connecting routers and network interfaces in an arbitrary topology, a TDM slot table size* sz_{tb} *(in number of slots) for all network interfaces, a flit size* sz_{fl} *(in bytes), a function* $\eta(ni_i, ni_j)$*, which for a connection from a network interface* $ni_i \in NI$ *to a network interface* $ni_j \in NI$ $(ni_i \neq ni_j)$ *returns a tuple* $\langle\beta, \Psi\rangle$ *with* β *the maximum bandwidth (in bytes per cycle) assuming that all slots in the TDM table are reserved, and* Ψ *the number of hops.*

Definition 2. *(Set of resources) The set R of resources is the union of the sets PE of processing elements, RT of routers, NI of network interfaces, and LK of* *links in a platform graph, and is defined as*

$$R = PE \cup RT \cup NI \cup LK \tag{2.1}$$

Definition 3. *(Platform graph) A platform graph gp is a 5-tuple $\langle PE, noc, FI, \psi, \chi\rangle$ consisting of a set PE of processing elements, an interconnect noc, a set of voltage-frequency islands FI, a function $\psi(fi) : FI \to \mathcal{P}(R)$, which for each voltage-frequency island $fi \in FI$ returns the set $R_{fi} \in R$ of resources belonging to the island, and a function $\chi(pe) : PE \to FI$, which for each processing element $pe \in PE$ returns the voltage-frequency island $fi \in FI$ to which the processing element belongs. Each processing element $pe \in PE$ is connected to a single network interface $ni \in NI$ in the interconnect noc.*

2.2 Variation in hardware resources

In this section, we present the modeling of the impact of manufacturing process variation in the maximum supported frequencies of hardware resources in a platform graph. Manufacturing process variation can be classified into die-to-die and within-die variations. Die-to-die variation, also referred to as global variation, acts globally on the entire chip die, affecting parameters of all devices (i.e. transistors) and wires on the die identically. Global variation is seen between dies within a wafer and between dies of different wafers (due to wafer-to-wafer variation); therefore, overall global variation presumes multiple wafers. In contrast, within-die variation, also known as local variation, affects parameters of devices on the same die differently. It can be classified into systematic and random components. Systematic local variation exhibits spatial correlation, such that nearby devices possess similar parameter values due to high correlation, which dies out quickly at the level of devices on a die, as a function of distance [22]. While the parameter correlation between adjacent devices on a die is high, the correlation between larger adjacent logic blocks on a die, such as a processing element, is typically much lower. Furthermore, measurements performed by Pang *et al.* [64, 65] show no significant spatial correlation at 45 nm technology, in contrast to 90 nm technology. This is partially because random local variation, which is purely random from device to device, has more than doubled at 45 nm technology whereas systematic local variation has decreased. For simplicity, we assume zero correlation between maximum supported frequencies of processing elements, routers, network interfaces and links in a chip due to local variation. An extension of the models presented in this section, such that correlation between maximum supported frequencies can be specified, is one of the possible future works of this thesis. It has been shown that a normal distribution is a good fit for modeling the impact of global and local manufacturing process variations [13, 64]. We, therefore, use normal distributions to model the impact of global and local process variations in the maximum supported frequency of resources in a platform graph. We thus proceed by presenting the models.

2.2.1 Global variation

To model the impact of global variation, we describe the maximum supported frequency of each hardware resource $r \in R$ in a platform graph gp by a random variable f_g^r distributed normally with μ_g^r mean and σ_g^r standard deviation. To denote that f_g^r is normally distributed, the notation $f_g^r = N(\mu_g^r, (\sigma_g^r)^2)$ is used. Global variation affects the maximum supported frequency of all hardware resources on a chip die identically. This results in equally faster or equally slower resources on each manufactured die. Therefore, we can say that the correlation between $(f_g^{r_i}, f_g^{r_j})$ for any $r_i, r_j \in R$ is equal to 1. Additionally, the standard deviation to mean ratio (σ_g^r/μ_g^r) is the same for all resources. The Probability Density Function (PDF) of a normally distributed random variable $x = N(\mu, \sigma^2)$ is given by Equation (2.2). The Cumulative Distribution Function (CDF) of $x = N(\mu, \sigma^2)$ is computed by Equation (2.3), where $\Phi(x)$ is the CDF of the standard normal distribution $N(0, 1^2)$ (Equation 2.4). The CDF $\theta(x_0, \mu, \sigma)$ represents the probability that the random variable x takes on a value less than or equal to x_0. Equations (2.2) and (2.4) can be used to compute the PDF and the CDF of $f_g^r = N(\mu_g^r, (\sigma_g^r)^2)$.

$$\phi(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{(x-\mu)}{2\sigma^2}} \tag{2.2}$$

$$\theta(x, \mu, \sigma) = \Phi\left(\frac{x-\mu}{\sigma}\right) \tag{2.3}$$

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \cdot \int\limits_{-\infty}^{x} e^{(t^2/2)}\,\mathrm{d}t \tag{2.4}$$

2.2.2 Local variation

Let us assume that a hardware resources has a certain maximum supported frequency $f_g^r = f_0$ due to global variation. The impact of local variation on the maximum supported frequency of the resource is overlaid on f_0. We thus introduce a normally distributed random variable $f_l^r = N(f_0 - \delta^r, (\sigma_l^r)^2)$ to model the impact of local variation on the maximum supported frequency of a hardware resource with respect to a global frequency value $f_g^r = f_0$ of the resource. Here, σ_l^r is the standard deviation and δ^r models a reduction in mean frequency of the hardware resource. Processing elements often contain multiple critical paths. The frequency of a processing element is decided by the slowest critical path. The probability that at least one of the critical paths is slowed down due to variation is higher than the probability that a single path is slowed down. This results in a mean frequency reduction, as shown by Bowman *et al.* in [13]. Links contain multiple wires, which has a similar impact on the mean frequency as in processing elements. The reduction for links has been shown experimentally by Hernandez *et al.* in [29]. In the same paper it is shown that the reduction in mean frequency is negligible for routers. We make a similar assumption of a negligible mean frequency reduction for network interfaces. As we assume no spatial correlation between the variation in maximum supported frequencies of hardware resources due to local process variation, the covariance between $(f_l^{r_i}, f_l^{r_j})$ for any $r_i, r_j \in R$ is equal to zero. Figure 2.2 illustrates an example PDF of f_g^r for a processing element with $\mu_g^r = 300$ MHz and $\sigma_g^r = 12$ MHz. The same figure shows the PDFs of f_l^r with respect to $f_0 = (\mu_g^r + k \cdot \sigma_g^r)$ for $(k = -1, 0, 1)$, where $\delta^r = 15$ MHz and $\sigma_l^r = 10$ MHz; these numbers are representative for 45 nm technology nodes, as the measurements in [64] show.

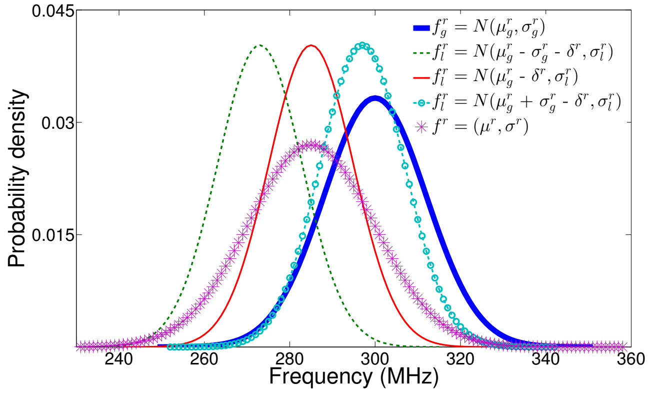


Figure 2.2: f_g^r PDF (due to global variation) for a processing element with $\mu_g^r = 300$ MHz, $\sigma_g^r = 12$ MHz; f_l^r PDFs (due to local variation) with respect to $f_g^r = 273$, 285 and 297 MHz, $\delta^r = 15$ MHz, $\sigma_l^r = 10$ MHz; combined PDF of f^r is the convolution of PDFs of f_g^r and f_l^r.

To describe the maximum supported frequency of a hardware resource by a single distribution, global and local distributions are combined by convolution, as given by Equation (2.5). Using this equation, the convolution for arbitrary distributions can be derived. However, as explained before, global and local variations in the maximum-supported frequency of a resource are modeled by means of normal distributions. It is known that the convolution of two normal distributions is also a normal distribution with added means and variances. Therefore, the maximum supported frequency of a hardware resource due to both global and local variations is described by a normally distributed random variable given by Equation (2.6). The combined distribution for the example described in Figure 2.2 is shown in the figure.

$$\phi_{cv}(f) = \int_{-\infty}^{\infty} \phi(x, \mu_g^r, \sigma_g^r) * \phi(f - x, 0, \sigma_l^r))\,\mathrm{d}x \tag{2.5}$$

$$f^r = N(\mu_g^r - \delta^r, \left(\sigma_g^r\right)^2 + (\sigma_l^r)^2) = N(\mu^r, (\sigma^r)^2) \tag{2.6}$$

2.3 Clock-frequency characterization

From an implementation perspective, all clock-generation units, associated with voltage-frequency islands in a platform graph, provide only a set of discrete clock-frequency levels. The selection of a set of clock-frequency levels for a voltage-frequency island is based on the variation in the maximum supported frequencies of hardware resources belonging to the island. It is performed in the following way. In a general case, a voltage-frequency island is comprised of multiple hardware resources (either processing elements or interconnect resources). Each resource is characterized by a combined distribution of its maximum supported frequency, reflecting both global and local process variations (Equation (2.5)). For the purpose of clock-frequency selection, we consider only the frequency range within three standard deviations from mean (i.e. $\mu^r \pm 3\sigma^r$) in the distributions. The probability of the maximum supported frequency being outside the range of three standard deviations is only 0.3%. Considering the range outside the three standard deviations, and thus providing clock-frequency levels in a wider range, will result in a lower number of clock frequencies in the range of three standard deviations (for the same number of levels). This can result in a performance degradation in manufactured chips, as the gap between the actual maximum supported frequency and the clock frequency a resource is operated will be larger for 99.7% of the resources. Figure 2.3 illustrates example combined distributions for the range of three standard deviations for two hardware resources belonging to the same island. We assume that the combined distributions can be in any arbitrary positioning with respect to each other. The clock frequency of an island is limited by the slowest resource belonging the island. Considering all resources in a voltage-frequency island, we identify the frequency given by the lowest positive three standard deviations from mean (i.e. $\mu^r + 3\sigma^r$) in the combined distributions. In Figure 2.3, this frequency is shown by f_{high}. Similarly, the frequency given by the lowest negative three standard deviations is derived, as shown by f_{low} in Figure 2.3. Once the frequencies f_{low} and f_{high} are identified, the clock-frequency levels are selected in the range given by $(f_{high} - f_{low})$. In principle, clock-frequency levels in the range $(f_{high} - f_{low})$ can be selected in any arbitrary way. The policy of selection does not affect the rest of the methodology in this thesis. We choose to select the clock-frequency levels equidistantly, as formally defined in Definition 4. Figure 2.3 illustrates how five equidistant clock-frequency levels are obtained for the given example.

Definition 4. *(Clock-frequency levels) A set of n equidistant clock-frequency levels available to a voltage-frequency island $fi \in FI$ in a platform graph $gp \in GP$, where $\psi(fi)$ hardware resources belong to fi, is given by $c(gp, fi, n) : GP \times FI \times \mathbb{N} \to \mathcal{P}(\mathbb{R}^+)$, and is defined as*

$$c(gp, fi, n) = \{f_{low} + (k-1) \cdot \frac{(f_{high} - f_{low})}{n} \mid k = 1, 2, .., n\} \tag{2.7}$$

where

$$\begin{aligned} f_{low} &= \min_{r \in \psi(fi)} (\mu^r - 3\sigma^r) \\ f_{high} &= \min_{r \in \psi(fi)} (\mu^r + 3\sigma^r) \end{aligned}$$

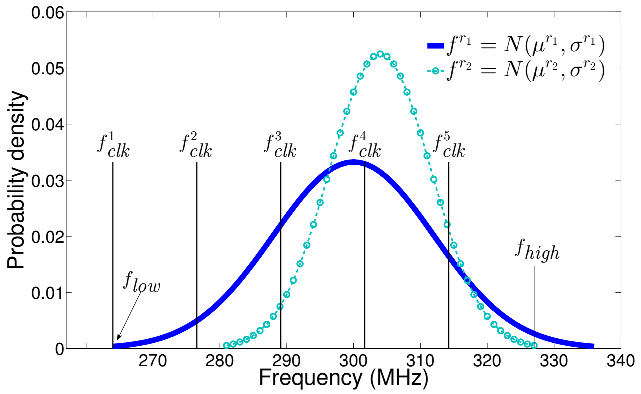


Figure 2.3: An example showing how equidistant clock-frequency levels are selected for a voltage-frequency island comprising two hardware resources.

Given that each voltage-frequency island can be operated at any clock-frequency level in the set $c(gp, fi, n)$, for a set FI of islands in a platform graph, there are multiple possible combinations of clock-frequency levels. An instance of clock-frequency levels for all islands in a platform graph is captured in a *chip-frequency vector*, denoted by fc, and is an M-dimensional vector for M islands (Definition 5). Each element in fc represents a clock-frequency level $f_{clk} \in c(gp, fi, n)$ for a corresponding island $fi \in FI$. The set of all possible chip-frequency vectors is obtained by the Cartesian product of individual sets $c(gp, fi, n)$ (Definition 6).

Definition 5. *(Chip-frequency vector) A chip-frequency vector for a set FI of voltage-frequency islands in a platform graph gp specifies a clock frequency f_{clk} from the set $c(gp, fi, n)$ for every island $fi \in FI$, and is given by $fc(fi) : FI \to \mathbb{R}^+$.*

Definition 6. *(All chip-frequency vectors) The set of all possible chip-frequency vectors for a set FI of voltage-frequency islands in a platform graph gp is given by*

$$FC = \prod_{fi \in FI} c(gp, fi, n) \tag{2.8}$$

Each chip-frequency vector $fc \in FC$ is associated with a probability, which is the probability that voltage-frequency islands in a platform graph are operated at the particular clock-frequency levels specified by fc. From probability theory, it is known that the joint probability of independent events equals the product of their individual probabilities. However, due to the correlated global variation in hardware resources in a platform graph, frequencies described by random variables f^r are not independent. On the other hand, resource frequencies described by random variables f_l^r are independent, as we assume no spatial correlation for local variation. For this reason, the joint probability of a chip-frequency vector fc is represented as a sum of components. Each component is the *joint local probability*, which is the probability that the islands are operated at the particular clock-frequency levels based on local distributions with respect to global frequency values of resources corresponding to a chip-level global variation. To simplify the computation of global frequency values of resources corresponding to a chip-level global variation, we introduce a *base hardware resource*, denoted by r_b, which is chosen arbitrarily among hardware resource in the platform graph. Given an absolute global frequency value $f_g^{r_b} = f_0$ of a base hardware resource, the global frequency values of any resource $r \in R$ is given by $f_0 \cdot (\mu_g^r / \mu_g^{r_b})$, where $\mu_g^{r_b}$ and μ_g^r are the global mean frequencies of hardware resources r_b and r, respectively.

We proceed by explaining how the probability that an island is operated at a clock frequency with respect to a global frequency value of the base resource is computed. In a general case, an island consists of multiple hardware resources. The clock frequency of the island is decided based on the slowest resource in the island. To compute probability that the island is operated at a clock frequency f_{clk}, the probability that the maximum supported frequency of all hardware resources in the island is higher than f_{clk} need to be considered. This is given by the product of probabilities $(1 - \theta(f_{clk}, \mu_l^r, \sigma_l^r)$ for all hardware resources belonging to the island. Definition 7 defines the CDF of the minimum of maximum supported frequencies of hardware resources in an island; the CDF $\theta_m(x_0, vfi, f_0)$ represents the probability that the minimum of the maximum supported frequencies of hardware resources takes on a value lower than x_0 with respect to an absolute global frequency value $f_g^{r_b} = f_0$ of the base hardware resource r_b.

Definition 7. *(Cumulative distribution of minimum) The CDF of the minimum of maximum supported frequencies of hardware resources $r \in \psi(fi)$ belonging to a voltage-frequency island fi in a platform graph $gp \in GP$ is given by $\theta_m(gp, x, fi, f_0) : GP \times \mathbb{R}^+ \times FI \times \mathbb{R}^+ \to \mathbb{R}^+$, and is defined as*

$$\theta_m(gp, x, fi, f_0) = 1 - \prod_{r \in \psi(fi)} (1 - \theta(x, \mu_l^r, \sigma_l^r)) \tag{2.9}$$

where μ_l^r is the mean of the normally distributed random variable f_l^r, and is computed with respect to an absolute global frequency value $f_g^{r_b} = f_0$ of the base *hardware resource $r_b \in R$ by*

$$\mu_l^r = f_0 \cdot \frac{\mu_g^r}{\mu_g^{r_b}} - \delta^r$$

Depending on the maximum supported frequency of hardware resources in a voltage-frequency island, each island is operated at the highest possible clock-frequency level. Let us consider the CDF of the minimum of frequencies of hardware resources in an island, shown in Figure 2.4. The plot is given with respect to an arbitrary global frequency value f_0 of the base hardware resource. For any actual x in the range $(f_{clk}^4,\ f_{clk}^5]$, the island is operated at f_{clk}^4. The probability of x being in the interval $(f_{clk}^4,\ f_{clk}^5]$ is computed by the difference $(\theta_m(gp, f_{clk}^5, fi, f_0) - \theta_m(gp, f_{clk}^4, fi, f_0))$. Similarly, the probability that the island is operated at f_{clk}^5 is given by $(1 - \theta_m(gp, f_{clk}^5, fi, f_0))$. This is formally defined in Definition 8.

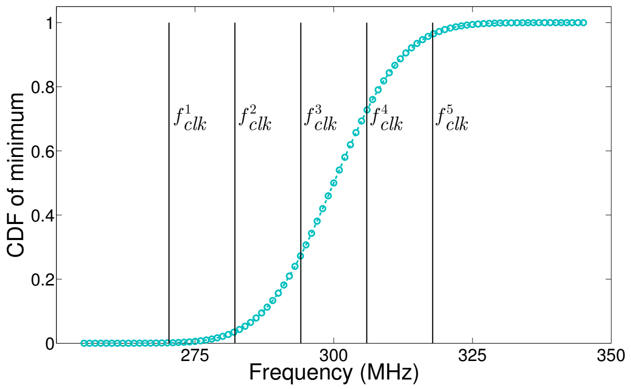


Figure 2.4: An example illustration of how the probability that an island is operated at a particular clock frequency is computed.

Definition 8. *(Probability of clock frequency) The probability that a voltage-frequency island $fi \in FI$ in a platform graph $gp \in GP$ is operated at a clock frequency f_{clk}^i for a set $\{f_{clk}^1, \cdots, f_{clk}^n\}$ of clock-frequency levels, where $f_{clk}^i < f_{clk}^{i+1}$, with respect to a global frequency value $f_g^{r_b} = f_0$ of the base hardware resource $r_b \in R$ is given by $pf(gp, f_{clk}^i, fi, f_0) : GP \times \mathbb{R}^+ \times FI \times \mathbb{R}^+ \to \mathbb{R}^+$, and is defined as*

$$pf(gp, f_{clk}^i, fi, f_0) = \begin{cases} \theta_m(gp, f_{clk}^{i+1}, fi, f_0) - \theta_m(gp, f_{clk}^i, fi, f_0) & i < n \\ 1 - \theta_m(gp, f_{clk}^i, fi, f_0) & i = n \end{cases} \tag{2.10}$$

The probability of a chip-frequency vector $fc \in FC$ for a set FI of voltage-frequency islands, with respect to a global frequency value f_0 of the base hardware resource, is computed by the product of individual probabilities $pf(gp, fc(fi), fi, f_0)$ and the probability of the global frequency value. This is formally defined in Definition 9.

Definition 9. *(Local probability of fc) The probability of a chip-frequency vector $fc \in FC$ for a set FI of voltage-frequency islands in a platform graph $gp \in GP$, with respect to a global frequency value $f_g^{r_b} = f_0$ of the base hardware resource $r_b \in R$, is given by $p(gp, fc, f_0) : GP \times FC \times \mathbb{R}^+ \to \mathbb{R}^+$, and is defined as*

$$p(gp, fc, f_0) = \phi(f_0, \mu_g^{r_b}, \sigma_g^{r_b}) \cdot \prod_{fi \in FI} pf(gp, fc(fi), fi, f_0) \tag{2.11}$$

The overall probability of a chip-frequency vector fc is obtained by adding the joint local probabilities for all global frequency values in the range of three standard deviations from mean for the base hardware resource, as defined in Definition 10.

Definition 10. *(Probability of fc) The probability of a chip-frequency vector $fc \in FC$ in a platform graph $gp \in GP$ is given by $pc(gp, fc) : GP \times FC \to \mathbb{R}^+$, and is defined as*

$$pc(gp, fc) = \sum_{f_0 \in I} p(gp, fc, f_0) \tag{2.12}$$

where

$$I = [\mu_g^{r_b} - 3\sigma_g^{r_b},\ \mu_g^{r_b} + 3\sigma_g^{r_b}]$$

2.4 Resource-aware application graph

We model best-effort and real-time streaming applications by means of Synchronous Data-Flow (SDF) graphs [36]. An SDF graph provides a good compromise between expressiveness, modeling ease, analysis potential and implementation efficiency. With an SDF model, an application is captured by a directed graph, where the nodes, called actors, represent computation, communication or storage; actors communicate with each other by sending streams of data elements, called tokens, over their edges. This is similar to the constructs used in domain-specific languages for streaming applications, such as StreamIt [2]. We denote the set of all actors as A, and formally define an SDF graph in Definition 11.

Definition 11. *(SDF graph) An SDF graph sdfg is a 3-tuple $\langle A, D, P \rangle$ consisting of a set A of actors, a set $D = A^2$ of dependency edges and a set P of ports. Each dependency edge has a number of initial tokens, given by $\xi(d) : D \to \mathbb{N}^0$. Each* *actor $a \in A$ is associated with input and output ports, where each port $pt \in P$ is associated with a rate $rate(pt) : P \rightarrow \mathbb{N}^+$. The source of a dependency edge $d \in D$ is an output port of an actor, and the destination of a dependency edge is an input port of an actor. Each port of each actor is connected to a single edge, and each edge is connected to ports of actors.*

Figure 2.5 illustrates an example SDF model of an H.263 encoder application. It consists of five actors connected to each other by seven dependency edges. Edges d_3, d_6 and d_7 each contain one initial token, illustrated by black dots in the figure. The execution of an actor is called a *firing*. An actor fires when it has sufficient number of tokens on each of its input ports, as specified by port rates $rate(pt)$. The port rates are shown near the channel ends in Figure 2.5. When an actor fires it removes the number of tokens from all its input edges and at the end of the firing (after its execution), it produces a number of tokens on each output port, as given by its output port rates. The sequence of actor firings that restores the initial configuration of the graph (the initial distribution of tokens on dependency edges $d \in D$) is termed an *iteration*. During a single iteration of the graph, each actor can fire multiple times. This is determined by the port rates of actors, and is captured by the repetition vector of the graph (Definition 12).

Definition 12. *(Repetition vector) A repetition vector of an SDF graph sdfg specifies the number of times each actor $a \in A$ fires during a single iteration, and is given by $\kappa : A \rightarrow \mathbb{N}$, such that for each dependency edge $d \in D$ from an actor $a_i \in A$ to actor an $a_j \in A$, $i \neq j$, $rate(pt_d^{sr}) \cdot \kappa(a_i) = rate(pt_d^{ds}) \cdot \kappa(a_j)$, where $pt_d^{sr} \in P$ and $pt_d^{ds} \in P$, are the source and destination ports of the dependency edge, respectively.*

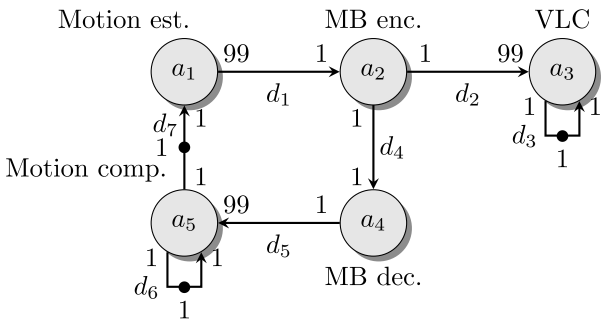


Figure 2.5: An example SDF model of an H.263 encoder.

An SDF graph is called *consistent* if it has a repetition vector such that for all $a \in A$, $\kappa(a) > 0$. If an SDF graph is not consistent, it requires unbounded memory for storing data tokens [37]. Deadlock freedom is also an important property of an

SDF graph. If an SDF graph is not free of deadlock, the actors cannot fire due to insufficient initial tokens on the cycles of the graph. In this thesis, only consistent and deadlock-free SDF graphs are considered. The repetition vector of the SDF graph shown in Figure 2.5 is equal to $\langle 1, 99, 1, 1, 99 \rangle$ for actors $\langle a_1, a_2, a_3, a_4, a_5 \rangle$, respectively. Therefore, the SDF graph is consistent (and deadlock free).

When mapping an application described by an SDF graph to a multi-processor platform, the minimum throughput requirement (in iterations per second) of the application (for real-time applications) and information on resource requirements of the application must be known. Such information includes the number of clock cycles each actor requires to finish its execution on a processing element, for all processing elements to which the actor can be bound (Definition 13), the buffer space (in number of tokens) assigned to each dependency edge for storing the data tokens produced by actors (in a real implementation these buffers can be allocated in local memories of processing elements), and the size (in bytes) of data tokens sent across each dependency edge. After the actors in an application are bound to the processing elements in a multi-processor platform, dependency edges may be allocated to connections in the interconnect (i.e. if two actors connected by a dependency edge are bound to different tiles). To reserve resources on the interconnect, additional information on the minimum bandwidth (in bytes per cycle) required by dependency edges has to be known. We formally define an SDF model of an application, called a *resource-aware application graph* (ga) that includes the described information in Definition 14. The set of all resource-aware application graphs is denoted by GA. Note that, a resource-aware application graph states the resource requirements, but does not include the binding of actors to processing elements and dependency edges to connections in the interconnect.

Definition 13. *(Execution cycles) The number of cycles required to execute an actor $a \in A$ on a processing element $pe \in PE$ to which it can can be bound is given by $ec(a, pe) : A \times PE \to \mathbb{N}$.*

Definition 14. *(Resource-aware application graph) A resource-aware application graph ga is a 4-tuple $\langle sdfg, t_{req}, ec, \omega \rangle$ consisting of an SDF graph $sdfg$, a minimum throughput requirement t_{req} for real-time applications, the function $ec(a, pe)$ that assigns each actor $a \in A$ with execution cycles for the subset of processing elements in the set PE that a can be bound to, and a function $\omega(d)$, which for each dependency edge $d \in D$ returns a 3-tuple $\langle sz_d, \beta_d^{rq}, \alpha_d \rangle$ with sz_d the size of the data token (in bytes) sent across the dependency edge, β_d^{rq} the required bandwidth (in bytes per cycle), and α_d the buffer space (in number of tokens) assigned to the dependency edge.*

In this thesis, the buffer space (in number of tokens) assigned for storing data tokens produced by actors on each dependence edge is fixed throughout the experiments. It is not a relevant parameter for the scope of this thesis, and the impact of it on the throughput of an SDF graph has been previously explored by Stuijk *et al.* in [80]. In this work, we limit the constraining of the throughput due to insufficient buffer space as follows. The buffer space α_d (in number of tokens) assigned to a dependency edge $d \in D$ is given by Equation (2.13), where $rate(pt_d^{sr})$ is the rate of the source port of the edge and $\kappa(a^{sr})$ is the number of times the source actor fires in an iteration. As such, the term $rate(pt_d^{sr}) \cdot \kappa(a^{sr})$ represents the number of tokens produced on the edge in a single iteration. Because of possible overlapping of multiple iterations (i.e. multiple iterations being active at the same point in time), a higher number of tokens than $rate(pt_d^{sr}) \cdot \kappa(a^{sr})$ can be produced on the edge. For this reason, a buffer space twice larger than the term $rate(pt_d^{sr}) \cdot \kappa(a^{sr})$ is assigned to each dependency edge to limit the constraining of the throughput due to insufficient buffer space.

$$\alpha_d = 2 \cdot rate(pt_d^{sr}) \cdot \kappa(a^{sr}) \tag{2.13}$$

The required bandwidth β_d^{rq} (in bytes per cycle) for a dependency edge $d \in D$ in a resource-aware application graph ga is estimated by Equation (2.14), where t_{α_d} is the throughput (in iterations per cycle) of the resource-aware application graph with the specified α_d buffer space for each dependency edge. The first three terms give the number of bytes that are produced on the edge and thus must be sent across a connection in the interconnect during a single iteration. The product of these terms and the throughput t_{α_d} represents the number of bytes produced on the edge in a single cycle. Note that the throughput t_{α_d} is evaluated on the resource-aware application graph assuming that only a single execution of each actor is allowed at the same point in time, as described in more details in Section 2.5.1. The throughput of the application when it is mapped to a multi-processor platform may be lower than t_{α_d} due to resource sharing and communication delays. As such, the throughput t_{α_d} is simply used to derive an estimated required bandwidth (independent of mapping) that is not overly pessimistic. The derived required bandwidth is later used to model the latency for sending data tokens across the interconnect in the bound application SDF graph, as described in Section 2.5.1.

$$\beta_d^{rq} = rate(pt_d^{sr}) \cdot \kappa(a^{sr}) \cdot sz_d \cdot t_{\alpha_d} \tag{2.14}$$

Note that actors are non-blocking pieces of code. Therefore, execution times in cycles can be determined on a processor in isolation, and no complete mapping of the application to the hardware platform is necessary. For real-time applications, execution cycles can be derived using Worst-Case Execution-Time (WCET) estimation tools, such as given in [93]. For best effort applications, typical execution times can be obtained by simulations.

2.5 Bound application graph

In the previous section, we introduced a resource-aware application graph, consisting of an SDF model of an application that includes information on the execution cycles of actors on processing elements, the size of data tokens communicated between actors, the required bandwidth of dependency edges, the buffer space assigned to each dependency edge, and the minimum throughput requirement for real-time applications. When implementing the application on a multi-processor platform, designers need to make choices on mapping of application actors to processing elements and thus dependency edges to connections in the interconnect, reserving resources on the connections, and scheduling of actors on the same processing element. Once these decisions have been made, a new SDF model, coined as a *bound application graph*, that captures these decisions is constructed, such that it can be used to perform timing analysis of the system. While a resource-aware application graph describes performance in terms of execution time in cycles, the essence of a bound application graph is that it considers performance in terms of execution time in seconds. This allows us to take process-induced variation in the frequency of processing elements into account. This section details how a bound application graph is constructed.

2.5.1 Modeling resource allocation

We start by capturing the binding of the resource-aware application graph to the platform graph. We assume that each actor can be bound to a number of processing elements from the set PE, as given by Definition 15. Therefore, there are multiple bindings of a set A of actors to a set PE of processing elements. The set of all possible bindings is obtained by the Cartesian product of the individual sets of possible bindings (Definition 16).

Definition 15. *(Actor bindings) The function $ba(a) : A \to \mathcal{P}(PE) \setminus \emptyset$ returns the set of processing elements to which an actor $a \in A$ can be bound.*

Definition 16. *(Binding) The set B of all possible actor to processing element bindings is given as*

$$B = \prod_{a \in A} ba(a) \tag{2.15}$$

A given binding of actors to processing elements is captured in a *binding vector*, denoted by b, and is an N-dimensional vector for N actors (Definition 17). Each element in b specifies the processing element to which each actor is bound. For example, let us assume that the resource-aware application graph shown in Figure 2.5 is mapped to a multi-processor platform comprising two processing elements; a_2, a_3 are bound to pe_1 and a_1, a_4, a_5 are bound to pe_2. This is given in a binding vector $\langle a_1, a_2, a_3, a_4, a_5\rangle \to \langle pe_2, pe_1, pe_1, pe_2, pe_2\rangle$.

Definition 17. *(Binding vector) A binding vector for a set A of actors specifies the processing element $pe \in PE$ to which each actor $a \in A$ is bound, and is given by $b(a) : A \to PE$.*

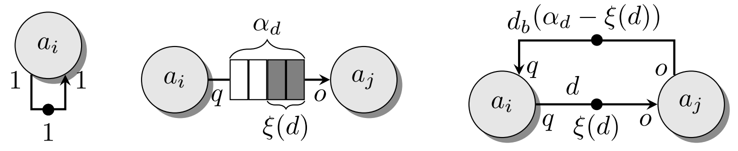
For a given binding vector b, the execution cycles $ec(a, pe)$ of each actor $a \in A$ is known from Definition 13. The execution time $et(gp, a, pe)$ in seconds is obtained by the division of $ec(a, pe)$ by the clock frequency $fc(\chi(b(a)))$ of the voltage-frequency island to which the processing element belongs (Definition 18).

Definition 18. *(Execution time) The execution time (in seconds) of an actor $a \in A$ on a processing element $pe \in PE$, which belongs to a voltage-frequency island $\chi(b(a))$ operated at a clock frequency $fc(\chi(b(a)))$ in a platform graph $gp \in GP$, is given by $et(gp, a, pe) : GP \times A \times PE \rightarrow \mathbb{R}^+$, and is defined as*

$$et(gp, a, pe) = \frac{ec(a, pe)}{fc(\chi(b(a)))} \tag{2.16}$$

In this thesis, we assume that only a single execution of an actor on a processing element at the same point in time is allowed. In an SDF graph, this is modeled by adding a self edge with a single initial token and with rates equal to one on source and destination ports on all actors, as shown in Figure 2.6a. This prohibits an execution to start before the previous execution has finished.

Now we proceed by discussing how buffers with a finite capacity are modeled in an SDF graph. In a real implementation, a buffer space needs to be assigned to each dependency edge in an SDF graph for storing the data tokens produced by actors (tasks). These buffers can be allocated in local memories of processing elements. Thus, an actor can only fire when sufficient space is available in the buffer to which the actor writes data. Figure 2.6b illustrates two actors a_i and a_j, connected to each other by a dependency edge d with port rates $(rate(pt_d^{sr}) = q)$ and $(rate(pt_d^{ds}) = o)$, where pt_d^{sr} and pt_d^{ds} are the source and destination ports of the dependency edge, respectively. The FIFO buffer is illustrated by the dotted rectangle, and is assumed to have a size α_d (in number of tokens), as given by function $\omega(d)$ in the resource-aware application graph. There can also be a number of initial data tokens stored in the buffer, shown as $\xi(d)$. When actor a_i fires, it produces q data tokens, which have to be stored before they are consumed by a_j. We use the approach in [67] to model the buffer space available to dependency edges in an SDF graph. Figure 2.6c shows how the available buffer space to the dependency edge d of the example shown in Figure 2.6b is modeled in the graph. For this, a backward edge d_b from a_j to a_i is added with $(\alpha_d - \xi(d))$ initial tokens on the edge, for which $(rate(pt_{d_b}^{sr}) = rate(pt_d^{ds}))$ and $(rate(pt_{d_b}^{ds}) = rate(pt_d^{sr}))$ hold, where $pt_{d_b}^{sr}$ and $pt_{d_b}^{ds}$ are the source and destination ports of the edge d_b, respectively. The size o the FIFO buffer is α_d, $\xi(d)$ models the number of initial tokens in the buffer and $(\alpha_d - \xi(d))$ models the available space in the buffer. Token consumption from the edge d_b can be seen as claiming space in the buffer for writing the data tokens. Token production on d_b can be seen as freeing space in the buffer. Note that self edges do not require buffer space in a real implementation, as they only model the absence of multiple executions of a task at the same point in time. Modeling constructs similar to the ones described in this paragraph are also used by Stuijk *et al.* in [79].



(a) actor bound to a processing element.

(b) buffered communication between actors.

(c) model of available buffer space between actors.

Figure 2.6: Modeling resource allocation in an SDF graph.

Connection model

Now we consider how to model latency of data communication across a connection in the interconnect in an SDF graph. In Figure 2.6c, the latency for sending date tokens across the dependency edge d equals zero. We assume that this is the case when two actors are bound to the same processing element, as they communicate through a shared local memory. However, when two actors are bound to different processing elements, the dependency edge that connects the actors is bound to a connection in the interconnect. In this scenario, the interconnect introduces latency for sending a data token across the connection. The following discussion describes how a connection model that considers the interconnect architecture is constructed in an SDF graph. As detailed in Section 2.1, we require that any arbitration point in a connection can be modeled as a latency rate server, independent of other connections. More specifically, we use a TDM-based arbitration with a single arbitration point in a connection at the network interface. Therefore, the connection model includes only a single latency rate component. The connection model used in this thesis is shown in Figure 2.7. It assumes the SDF graph of Figure 2.6c, where actors a_i and a_j are bound to different processing elements, and thus, the dependency edge d is bound to a connection in an interconnect. The execution time of actor a_l captures the maximum latency a data token can ever experience in a connection. The actor does not have a self edge with an initial token on it. This ensures that the latency can be experienced my multiple tokens at the same time; as such, the model captures pipelining between data tokens. The edge d_b^{ds}, with α_d^{ds} initial tokens on it, models the buffer space at the destination tile. As such, actor a_l fires only when sufficient tokens are available on the edge, corresponding to a data transaction being initiated only when buffer space is available at the destination processing element. The execution time of the actor can be broken down to two different components: slot table injection latency and path latency. Data injection into the router network is regulated by the reserved slots in the TDM table of a connection. The slot table injection latency depends on the number of reserved slots. Based on the required bandwidth β_d^{rq} of a dependency edge $d \in D$, a number Λ of slots are reserved in the TDM table (with a size sz_{tb}) of a connection, as shown in Equation (2.17), where β is the maximum bandwidth (in bytes per cycle) of a connection from a network interface ni_i to a network interface ni_j, $i \neq j$, given by the function $\eta(ni_i, ni_j)$ (Definition 3). The bandwidth β assumes that all slots in the TDM table are reserved. The allocated bandwidth β_{al} (in bytes per cycle) is computed by Equation (2.18).

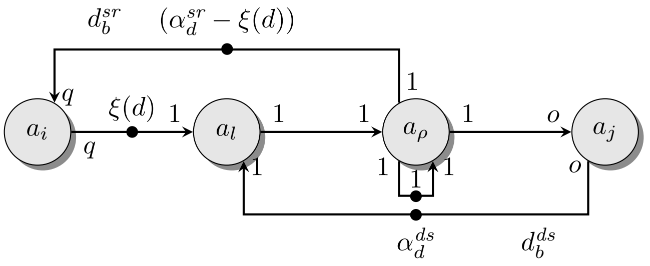


Figure 2.7: SDF model of a connection in the interconnect.

$$\Lambda = \left\lceil \frac{\beta_d^{rq}}{\beta} \cdot sz_{tb} \right\rceil \tag{2.17}$$

$$\beta_{al} = \frac{\Lambda}{sz_{tb}} \cdot \beta \tag{2.18}$$

The slot table injection latency not only depends on the number of reserved slots, but also on the distance between them. In this thesis, we assume a continuous slot reservation strategy due to the simplicity of its analysis and implementation; this strategy is shown in Figure 2.8. With this strategy, in the worst-case situation, a data token arrives right after the reserved Λ slots, resulting in a waiting time of a number $(sz_{tb} - \Lambda)$ of slots. The slot table injection latency (in cycles) is given by Equation (2.19), where sz_{fl}/β represents the latency (in cycles) of a single slot. Note that other reservation strategies, such as distributed slot reservation, are possible. These strategies may give a lower service latency, but are more difficult to analyze and implement.

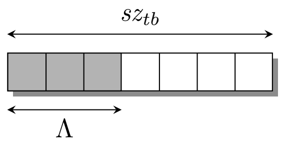


Figure 2.8: Continuous slot reservation strategy in a TDM slot table.

$$\Theta_{tb} = (sz_{tb} - \Lambda) \cdot \frac{sz_{fl}}{\beta} \tag{2.19}$$

The path latency depends on the number Ψ of hops, given by the function η (Definition 3), and the pipelining depth α_{rt} of the routers. In the case of the Æthereal network, the pipelining depth of the routers is three, which is also chosen in this thesis. The path latency (in cycles) is given by Equation (2.20). Combining the path latency and the slot table injection latency, the execution time (in seconds) of actor a_l is given by Equation (2.21).

$$\Theta_{hp} = \Psi \cdot \alpha_{rt} \tag{2.20}$$

$$et_l(a_l) = \frac{(\Theta_{tb} + \Theta_{hp})}{fc(fi_{noc})} \tag{2.21}$$

Actor a_ρ bounds the rate at which data tokens are sent. With a self edge and a single initial token on it, the execution time of the actor captures the time it takes to serve a data token after an initial latency et_l. The execution time (in seconds), given an allocated bandwidth β_{al} and a token size sz_d, is given by Equation (2.22). The edge d_b^{sr} with $(\alpha_d^{sr} - \xi(d))$ initial tokens models the available buffer space at the source processing element. In this thesis, both α_d^{sr} and α_d^{ds} are chosen to be equal to $\alpha_d/2$. The presented connection model is similar to the one given by Hansson *et al.* in [28]. Having presented the modeling of resource allocation in the resource-aware application graph, a bound application graph gb is formally defined in (Definition 19).

$$et_\rho(a_\rho) = \frac{sz_d/\beta_{al}}{fc(fi_{noc})} \tag{2.22}$$

Definition 19. *(Bound application graph) A bound application graph gb is a 5-tuple $\langle sdfg, ga, gp, b, fc, \rangle$ consisting of an SDF graph $sdfg$ that models resource allocation when the resource-aware application graph ga is mapped to the platform graph gp, a binding vector b that specifies the processing element $pe \in PE$ to which each actor $a \in A$ is bound, and a chip-frequency vector $fc \in FC$ that specifies the clock-frequency of each voltage-frequency island $fi \in FI$.*

2.5.2 Throughput computation

Throughput is an important design metric in streaming applications. In the context of an SDF graph, it quantifies the rate at which output data tokens are produced. An example is frames per second for an H.263 decoder. There is a body of work that addresses the problem of throughput computation for SDF graphs. As proposed by Sundararajan *et al.* [75], the throughput of an SDF graph can be computed by performing *Maximum Cycle Mean* (MCM) analysis on an equivalent HSDF graph, which is a special case of an SDF graph with all rates associated to ports equal to 1. This suggests that an SDF graph must be converted to an equivalent HSDF graph. An algorithm for this conversion is given by the same authors in [75]. The throughput of an HSDF graph is then given as 1 over the maximum cycle mean of the graph. The cycle mean of a cycle is the sum of the execution times of actors on the cycle over the number of initial tokens in it. Several algorithms to perform efficient MCM analysis have been proposed, and are compared for run time and additional properties in [19]. However, the drawback of computing the throughput based on an equivalent HSDF graph is that it can be exponentially larger in size than the original SDF graph. This is decided by the repetition vector of the graph. For example, the SDF model of an MP3 Playback application given in [91] consists of only four actors, while the number of actors in an equivalent HSDF graph after conversion becomes 10601. Performing throughput analysis on a large graph results in prohibitively high computation times. For this reason, a different technique to compute the throughput directly on an SDF graph has been proposed by Ghamarian *et al.* in [24]. The authors show that the throughput of an SDF graph computed by their approach is equivalent to 1 over the MCM of the corresponding HSDF graph. This technique is based on the timed execution of an SDF graph, later being constructed with the following principle. All actors that have sufficient data tokens on their input edges fire; the execution of actors is captured by time transitions or increments (e.g. in seconds or cycles). When an actor finishes its firing, it produces data tokens on its output edges. This changes the token distribution on the edges in the graph, enabling other actors (or the same actor) to fire. The distribution of data tokens on the edges of the graph is termed *state*. After each firing of an actors, the state of the graph is saved. The timed execution of the graph continues until a previously visited or a recurrent state (i.e. distribution of tokens on the edges) is reached. At this point, the periodic phase of the timed execution is found. Note that throughput computation is usually restricted to strongly connected SDF graphs, in which all actors can be reached from another actor in the graph. This is the case in this thesis, as the back edges modeling the available buffer space make the SDF graph strongly connected. Following the throughput computation approach of timed execution, the throughput of a bound application graph is given by Definition 20.

Definition 20. *(Throughput) The throughput of a bound application graph gb is defined as*

$$\tau(gb) = \frac{\tau(a)}{\kappa(a)} \tag{2.23}$$

where $\tau(a)$ is the throughput of an arbitrary actor $a \in A$, which is equal to the average number of firings per second in the periodic part of the self-timed execution of bound application graph gb, and κ is the repetition vector.

A tool called SDF For Free (SDF^3) has been developed and is available for use by research community [72, 81]. Besides being able to generate SDF graphs with specified properties with support for visualization, SDF^3 offers a library that can be used to perform throughput computation based on timed execution of an SDF graph. In this thesis, this library is used to perform throughput computation.

2.5.3 Scheduling

Several actors of a resource-aware application graph are often allocated to the same processing element. In this scenario, the firings of actors must be scheduled on shared processing elements. In this thesis, the actors allocated to the same processing element are scheduled with static-order scheduling s_{pe}, which specifies a sequence of actor firings repeated infinitely. For an example binding vector $(a_1, a_2, a_3, a_4, a_5) \to (pe_2, pe_1, pe_1, pe_2, pe_2)$, presented for the graph shown in Figure 2.5, the schedules for two processing elements can be $s_{pe_1} = ((a_2)^{99} a_3)^*$ and $s_{pe_2} = (a_1(a_4)^{99} a_5)^*$, where $(a_i)^j$ specifies that a_i fires j times, and * indicates that the schedule is repeated indefinitely.

Scheduling of SDF graphs on uni-processor systems was addressed by Bhattacharyya *et al.* [6]. They proposed a class of *single appearance schedules*, in which each actor appears only once in the regular expression of a schedule. As the authors demonstrate, the benefit of a single appearance schedule is the minimized program-code size, as no duplication of the software code implementing the functionality of an actor is performed. Sundararajan *et al.* address scheduling of HSDF graphs on multi-processor systems in [75]. Using this approach on an SDF graph requires conversion to an equivalent HSDF graph, which can be exponentially larger in size, resulting in prohibitively high throughput computation times. To limit the increase in the graph size, clustering algorithms may be used, as shown in [66]. With this approach, several actors in an SDF graph are clustered into a single actor before conversion. The actors belonging to the same cluster are then scheduled with uni-processor schedulers on the same processor. A scheduling policy for multi-processor systems working directly on SDF graphs is proposed in [79]. When computing the throughput of an SDF graph mapped to a multi-processor platform, the scheduling can be captured following two approaches. The first approach is to model the schedules in the graph itself. The work in [76] shows how static-order schedules can be modeled in an HSDF graph. It is accomplished by adding additional edges in the graph, such that the schedules (i.e. order of actor firings) is enforced. Following a similar approach, static-order schedules can also be modeled directly in an SDF graph, as addressed by Damavandpeyma *et al.* in [18]. In contrast to modeling schedules in HSDF graphs, with this method, additional edges and also actors with zero execution time are added in the SDF graph to enforce the actor firings according to the schedule on a processor.

When computing the throughput of an SDF graph, the second approach to capture scheduling between actors bound to the same processing element in a multi-processor system is to account for them during the self-timed execution of the SDF graph. This is demonstrated by Stuijk *et al.* in [79]. In this thesis, we use this approach to construct and capture static-order scheduling for all processing elements when computing the throughput of a bound application graph. An intuitive explanation of the method is given as follows. During the construction of the self-timed execution of the SDF graph, each time an actor is enabled for firing, it does not start its firing immediately. Instead, it is added to a ready list of the processing element to which it is bound. When no other actor is firing on the processing element, the actor is added to the schedule of the processing element and starts firing. When a recurrent state is reached, a finite length schedule for each processing element is constructed.

2.6 Summary

Formal models are required to perform design-time analysis and give timing guarantees in a system with voltage-frequency islands, considering the impact of process-induced variations. To this end, this chapter introduced the formal models used throughput this thesis. We presented a multi-processor platform, coined as a platform graph, consisting of multiple processing elements connected to each other by an interconnect. The maximum supported frequency of the hardware resources in a platform graph is given by normally distributed random variables, capturing the impact of process-induced die-to-die and within-die variations. A methodology was presented to select a set of clock-frequency levels provided to each voltage-frequency island based on the frequency variations in hardware resources. We later characterized the platform graph with possible sets of clock frequencies for the voltage-frequency islands. Based on process-induced frequency variations, the probability that the voltage-frequency islands are operated at each set of clock frequencies was derived.

Following the introduction of a platform graph, we presented a general SDF model of computation for performance (timing) analysis. Our main contribution lies in the separation of a resource-aware application graph and a bound application graph. The resource-aware application graph uses clock cycles as the basis for performance specification. Since variation affects the length of the clock cycles, the resource-aware application graph is unaware of process-induced variations. The bound application graph captures the mapping of a resource-aware application graph to a platform graph, where the maximum supported frequency of hardware resources is affected by process variation. Therefore, the bound application graph models the performance of an SDF application mapped on a multi-processor platform considering the impact of process-induced variations. We illustrated how buffers with a finite capacity, used for data communication, and the latency for sending data tokens across a connection in an interconnect are modeled in the bound application graph.

3 Variation-aware mapping

In this chapter, we describe the requirements of *best-effort*, *firm real-time* and *soft real-time* applications, and define a mapping optimization problem for each of them in the presence of process-induced variations in the maximum supported frequency of hardware resources in a multi-processor platform. For best-effort applications, we maximize the average throughput over all manufactured chips. The optimization objective for firm real-time applications is to maximize the timing yield, which is the percentage of manufactured chips that satisfy the minimum throughput requirement of an application. For soft real-time applications, we maximize the timing yield and/or reduce the average throughput degradation, as the chips with a low degradation in the throughput can be used in this class of applications. The optimization criteria for the different application classes are summarized in Table 3.1.

3.1 Optimization problems

For the different application classes, we present *single-binding* and *multiple-bindings* mapping approaches. With the single-binding mapping approach, the objective is to find a binding at design time that results in an optimized objective function for an application class. For example, the objective function for firm real-time applications is the timing yield. With the multiple-bindings mapping approach, a set of bindings are found and stored at design time, and based on the variation in each manufactured chip, the right binding that satisfies the throughput requirement of the application (for real-time applications) or maximizes the throughput (best-effort applications) is selected at a run-time configuration stage for each chip independently. We experimentally compare both mapping approaches in Section 3.5.

Table 3.1: Optimization criteria for application classes

	Best-effort	Firm real-time	Soft real-time
Average throughput	$\surd$	–	–
Timing yield	–	$\surd$	$\surd$
Average throughput degradation	–	–	$\surd$

3.2 Single-binding

With the single-binding mapping approach, a single binding is selected at design time, such that the objective function for an application class (e.g. timing yield for firm real-time applications) is optimized. As explained in Section 2.3, depending on the maximum supported frequency of hardware resources in a platform graph, each voltage-frequency island is operated at a particular clock frequency from the set of frequency levels selected for the island. The clock frequencies for the overall number of islands on the chip is given by a chip-frequency vector fc. The set of all possible chip-frequency vectors for a chip is given by FC. With the single-binding mapping approach, all manufactured chips that can have have different chip-frequency vectors from the set FC have an identical binding. We proceed by presenting the optimization criteria for the different application classes and defining the single-binding optimization problem for each of them.

3.2.1 Best-effort applications

Best-effort applications do not have real-time performance requirements. Although there are no timing requirements set on the applications of this class, high performance is preferred by the user. Our optimization objective for best-effort applications is hence to maximize the average throughput taken over all manufactured chips.

For a given binding $b \in B$, the chips that have different chip-frequency vectors can have different throughputs. Figure 3.1 depicts the throughput $\tau(gb)$ of a bound application graph $gb = \langle sdfg, ga, gp, b, fc \rangle$ for different chip-frequency vectors $fc \in FC$, given a fixed binding $b \in B$. With the single-binding mapping approach, the objective is to find a binding that maximizes the average throughput t_{avg} over all chip-frequency vectors (i.e. average throughput over all manufactured chips). This is illustrated in Figure 3.1. Note that each chip-frequency vector $fc \in FC$, shown on the x-axis in Figure 3.1, is a vector specifying the clock frequency of each voltage-frequency island $fi \in FI$ in the platform graph gp. Therefore, the x-axis does not assume any increasing or decreasing ordering of $fc \in FC$. Furthermore, the figures presented in this section do not represent realistic applications, but are given for illustrative purposes; experiments on real applications are presented in Section 3.5. For clarity, the throughput $\tau(gb)$ is shown as a continuous curve, but is discrete in reality.

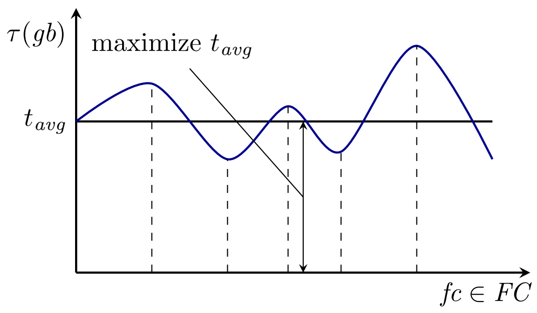


Figure 3.1: Throughput against chip-frequency vector for a fixed binding. The average throughput over all chip-frequency vectors is shown by the dotted line.

The average throughput of a binding $b \in B$ over all chip-frequency vectors $fc \in FC$, where each chip-frequency vector has a probability-weight $pc(gp, fc)$ is given by Definition 21.

Definition 21. *(Average throughput of a binding) The probability-weighted average throughput of a bound application graph $gb = \langle sdfg, ga, gp, b, fc, \rangle$ over all chip-frequency vectors $fc \in FC$, for a specified resource-aware application graph $ga \in GA$, platform graph $gp \in GP$ and binding $b \in B$, is given by $\tau_{avg}(ga, gp, b) : GA \times GP \times B \to \mathbb{R}^+$, and is defined as*

$$\tau_{avg}(ga, gp, b) = \sum_{fc \in FC} \tau(gb) \cdot pc(gp, fc) \tag{3.1}$$

The objective of the single-binding mapping approach for best-effort applications is formulated as: given a resource-aware application graph ga and platform graph gp, find a binding $b_{sb} \in B$ of actors in the resource-aware application graph to processing elements in the platform graph, such that the average throughput $t_{avg} = \tau_{avg}(ga, gp, b_{sb})$ is maximized.

3.2.2 Firm real-time applications

In firm real-time applications, violations of the timing requirements are not allowed. Examples of such applications are software-defined radio, air-traffic control, robotics and military systems. The manufactured chips that have less than the required performance cannot be used in such systems. Providing more than the required performance is not important as long as the performance requirement is satisfied. The optimization for this class of applications aims at maximizing the timing yield, which is the percentage of chips that satisfy the throughput requirement.

With the single-binding mapping approach for firm real-time applications, the objective is to find a binding that maximizes the timing yield. In our modeling framework, the timing yield of a binding $b \in B$ over all chip-frequency vectors $fc \in FC$, where each chip-frequency vector has a probability-weight $pc(gp, fc)$, is given by Definition 22. The timing yield computation of a binding is graphically illustrated in Figure 3.2.

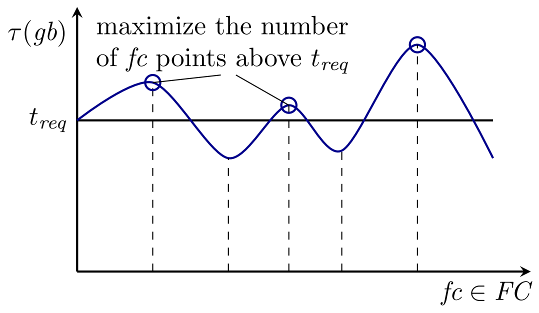


Figure 3.2: Throughput against chip-frequency vector for a fixed binding. The timing yield is given by the percentage of fc points (with associated probabilities $pc(gp, fc)$) above t_{req}.

Definition 22. *(Timing yield of a binding) The percentage of chips, characterized by a bound application graph $gb = \langle sdfg, ga, gp, b, fc, \rangle$, that satisfy the throughput requirement t_{req} over all chip-frequency vectors $fc \in FC$, for a specified resource-aware application graph $ga \in GA$, platform graph $gp \in GP$ and binding $b \in B$, is given by $\gamma(ga, gp, b) : GA \times GP \times B \to \mathbb{R}^+$, and is defined as*

$$\gamma(ga, gp, b) = \sum_{fc \in FC} \begin{cases} pc(gp, fc) & \textit{if } \tau(gb) \geq t_{req} \\ 0 & \textit{otherwise} \end{cases} \tag{3.2}$$

Figure 3.3 shows two bindings b_1 and b_2, where b_2 has a higher average throughput (better for best-effort applications), but a lower timing yield (worse for firm real-time applications) than b_1. This illustrates the benefits of having different optimization criteria for best-effort and firm real-time application classes.

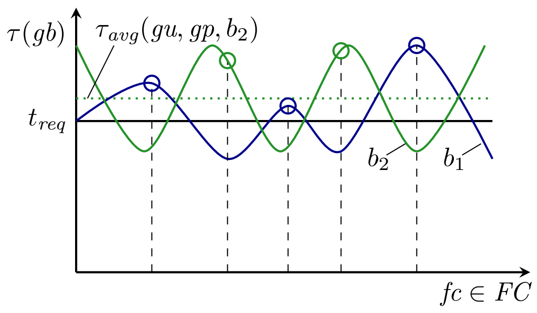


Figure 3.3: Throughput against chip-frequency vector for a fixed binding. Two bindings b_1 and b_2 are shown, where b_2 has a higher average throughput but a lower timing yield than b_1.

The objective of the single-binding mapping approach for firm real-time applications is formulated as: given a resource-aware application graph ga and platform graph gp, find a binding $b_{sb} \in B$ of actors in the resource-aware application graph to processing elements in the platform graph, such that the timing yield $\gamma(ga, gp, b_{sb})$ is maximized.

3.2.3 Soft real-time applications

Compared to firm real-time application, soft real-time applications are characterized by less stringent timing requirements. In such applications, missing a deadline causes only a performance degradation, often evaluated through some quality-of-service parameter. Examples of such applications are multimedia systems, monitoring apparatuses, virtual reality, and interactive computer games. The optimization objective for this class of applications is to improve the timing yield, but also to reduce the average throughput degradation (deviation from the requirement) over all chips, as the chips with a low degradation can be used in this class of applications.

With the single-binding mapping approach, the objective is to find a binding that results in maximized timing yield and/or minimized average throughput degradation over all chip-frequency vectors (i.e. over all manufactured chips). Depending on the particular (timing yield, average throughput degradation) value pairs, one binding may be preferred over another. A binding that has a lower timing yield but also a lower average throughput degradation may be preferred over another binding with a higher timing yield. The trade-off between the timing yield and the average throughput degradation is captured in an objective function that guides the binding selection process (Definition 23). The optimization problem for soft real-time applications is formulated based on the objective function and aims at minimizing it. This is graphically illustrated in Figure 3.4.

Definition 23. *(Probability-weighted sum of throughput degradation) The sum of the probability-weighted difference between the throughput requirement t_{req} and the (lower than the requirement) throughput of a bound application graph $gb = \langle sdfg, ga, gp, b, fc, \rangle$, over all chip-frequency vectors $fc \in FC$, for a specified resource-aware application graph $ga \in GA$, platform graph $gp \in GP$ and binding $b \in B$, is given by $\zeta(ga, gp, b) : GA \times GP \times B \to \mathbb{R}^+$, and is defined as*

$$\zeta(ga, gp, b) = \sum_{fc \in FC} \begin{cases} (t_{req} - \tau(gb)) \cdot pc(gp, fc) & \text{if } \tau(gb) < t_{req} \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

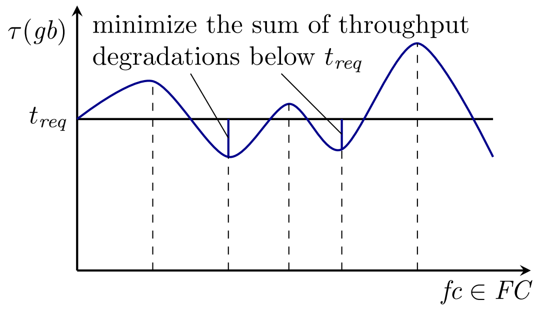


Figure 3.4: Throughput against chip-frequency vector for a fixed binding. Selecting a binding with the lowest objective function ζ is equivalent to minimizing the shaded area below t_{req}.

A binding selected for a soft real-time application, as a result of minimized objective function, provides certain timing yield and average throughput degradation. The timing yield of a binding is estimated by Definition 22. The average throughput degradation of the binding with a certain value of the objective function is computed by Definition 24.

Definition 24. *(Average throughput degradation of a binding) The average throughput degradation of a binding $b \in B$, for a specified resource-aware application graph $ga \in GA$, platform graph $gp \in GP$, is given by $\Delta\tau_{avg}(ga, gp, b) : GA \times GP \times B \to \mathbb{R}^+$, and is defined as*

$$\Delta\tau_{avg}(ga, gp, b) = \frac{\zeta(ga, gp, b)}{1 - \gamma(ga, gp, b)} \tag{3.4}$$

Figure 3.5 shows two bindings b_1 and b_2, where b_2 has a higher timing yield (better for firm real-time applications), but also a much higher average throughput degradation (worse for soft real-time applications) than b_1. This example shows that a good solution for a soft real-time application may not necessarily be preferred for a firm real-time application and vice versa. This is why, different optimization criteria are defined for the application classes.

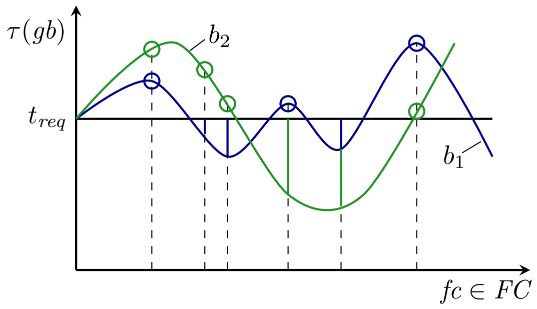


Figure 3.5: Throughput against chip-frequency vector for a fixed binding. Two bindings b_1 and b_2 are shown, where b_2 has higher timing yield, but also a higher average throughput degradation, than b_1.

The objective of the single-binding mapping approach for soft real-time applications is formulated as: given a resource-aware application graph ga and platform graph gp, find a binding $b_{sb} \in B$ of actors in the resource-aware application graph to processing elements in the platform graph, such that the objective function $\zeta(ga, gp, b_{sb})$ is minimized.

3.3 Multiple-bindings

With the multiple-bindings mapping approach, a binding for each chip-frequency vector is selected at design time. Therefore, a set of bindings are found and stored at design time. Based on the chip-frequency vector of each manufactured chip, the right binding that satisfies the application throughput requirement or maximizes the throughput is then selected at a run-time configuration stage. The run-time binding selection for each chip is done only once at a system initial configuration stage through the operating system, and is not detrimental to real-time deadlines. Per chip binding selection always provides better or equal results as compared to the case where a single binding is used for all chips. The downside of the approach is that multiple bindings, together with information on which binding needs to be applied to each chip-frequency vector, has to be stored for the configuration stage. Additionally, diverse application instances are present for the same product, which can complicate the processes of software maintenance and upgrading.

The objective of the multiple-bindings mapping approach for best-effort applications is to find a binding for each chip-frequency vector $fc \in FC$, such that the throughput of the bound application graph at that chip-frequency vector is maximized. For firm real-time applications, a binding for each chip-frequency vector is selected that satisfies the throughput requirement of the bound application graph at that chip-frequency vector. If there is no binding that satisfies the requirement, then the chips with that particular chip-frequency vector cannot be used, reducing the timing yield. For soft real-time applications, the objective is to find a binding for each chip-frequency vector, such that the throughput requirement is satisfied. If there is no binding that satisfies the requirement, then a binding is selected that provides the lowest throughput degradation.

3.4 Implementation algorithms

To implement any of the optimization problems, various bindings of actors in an unbound graph to processing elements in a platform graph have to be explored. In this section, we present two algorithms for the evaluation of bindings, an *exhaustive* and a *heuristic* algorithm. With the exhaustive approach, we evaluate all binding possibilities (as given by Definition 16) to find a single binding for all chips (i.e. chip-frequency vectors) in the case of single-binding approach, or a binding per chip (i.e. for each chip-frequency vector) in the case of multiple-bindings mapping approach. This approach enables us to find the optimum solution. However, it is too computationally expensive for problems of a large size (i.e. large number of actors in an resource-aware application graph and processing elements in a platform graph). To overcome this limitation, we also provide a heuristic algorithm that prunes the search space and obtains results close to the optimum.

The exhaustive and the heuristic algorithms presented in this section are given for the class of firm real-time application, and therefore aim at maximizing the timing yield. The algorithms for best-effort and soft real-time applications are similar to the ones presented and are not given in this section. The differences in the algorithms are explained where necessary.

3.4.1 Exhaustive algorithm

The exhaustive algorithm for the single-binding mapping approach for firm real-time applications is shown in Algorithm 1. As input, the algorithm requires a resource-aware application graph ga, platform graph gp, desired number n of clock-frequency levels for each voltage-frequency island $fi \in FI$, and $\mu_g^r, \sigma_g^r, \delta^r, \sigma_l^r$ values for the random variables describing the maximum supported frequency of each hardware resource $r \in R$ (see Section 2.2). As the algorithm shows, the set of clock-frequency levels for each voltage-frequency island are first computed, followed by the derivation of the set FC of all possible chip-frequency vectors for the set of islands (Definition 6). The algorithm exhaustively evaluates the timing yield (Definition 22) for all possible bindings, and returns the binding b_{sb} that results in the highest timing yield $y_{max} = \gamma(ga, gb, b)$. The algorithms for best-effort and soft real-time applications use the functions $\tau_{avg}(ga, gb, b)$ (Definition 21) and

$\zeta(ga, gb, b)$ (Definition 23) to exhaustively find a binding with the highest average throughput or the lowest objective function for soft real-time applications.

Algorithm 1: Exhaustive mapping algorithm implementing the single-binding mapping approach for firm real-time applications.

```
Require: $ga; gp; n\ \forall fi \in FI; (\mu_g^r, \sigma_g^r, \delta^r, \sigma_l^r)\ \forall r \in R$
 1: Compute $c(gp, fi, n)\ \forall fi \in FI$
 2: Derive $FC$
 3: $y_{max} \leftarrow 0$
 4: for all $b \in B$ do
 5:     if $\gamma(ga, gp, b) > y_{max}$ then
 6:         $b_{sb} \leftarrow b$
 7:         $y_{max} \leftarrow \gamma(ga, gp, b)$
 8:     end if
 9: end for
10: return $b_{sb}$, $y_{max}$
```

Algorithm 2 illustrates the exhaustive algorithm for the multiple-bindings mapping approach for firm real-time applications. As shown, for each chip-frequency vector $fc \in FC$, the algorithm exhaustively evaluates all possible bindings to find a binding that satisfies the requirement t_{req}. The first binding that satisfies the requirement t_{req} is stored for each chip-frequency vector. The algorithm returns a set B_{mb} of bindings that includes a binding for each individual $fc \in FC$ and an estimated timing yield y_{max}. For best-effort applications, the algorithm exhaustively evaluates all possible bindings for each chip-frequency vector $fc \in FC$ to find a binding with a maximized throughput. Note that the break statement on Line 10 in Algorithm 2 does not apply for the best-effort mapping algorithm, as all bindings for each chip-frequency vector have to be explored. For soft real-time applications, the algorithm exhaustively selects a binding for each chip-frequency vector $fc \in FC$, such that the requirement t_{req} is satisfied (the break statement on Line 10 is used) or otherwise the lowest throughput degradation is obtained by evaluating all possible bindings.

The exhaustive algorithms enable us to find the optimum solution. The limitation of the exhaustive algorithms is that they are too computationally expensive for problems of a large size. The total number $|B|$ of bindings to evaluate for timing yield or for throughput for each $fc \in FC$ is given by Equation 3.5, where $|ba(a)|$ is the number of processing elements in a platform graph to which an actor a can be bound, and A is the set of all actors belonging to a resource-aware application graph. Thus, the complexity of the exhaustive algorithms is given by $O(|A| \cdot |PE|)$. To address this problem, we proceed by presenting a heuristic algorithm that prunes the search space to reduce computation time, while still obtaining results close to the optimum.

Algorithm 2: Exhaustive mapping algorithm implementing the multiple-bindings mapping approach for firm real-time applications.

```
Require: $ga; gp; n\ \forall fi \in FI; (\mu_g^r, \sigma_g^r, \delta^r, \sigma_l^r)\ \forall r \in R$
 1: Compute $c(gp, fi, n)\ \forall fi \in FI$
 2: Derive $FC$
 3: $y_{max} \leftarrow 0$
 4: $B_{mb} \leftarrow \emptyset$
 5: for all $fc \in FC$ do
 6:   for all $b \in B$ do
 7:     if $\tau(gb) \geq t_{req}$ then
 8:       $B_{mb} \leftarrow B_{mb} \cup \{b\}$ //save $b$ for current $fc$
 9:       $y_{max} \leftarrow y_{max} + pc(gp, fc)$
10:       break
11:     end if
12:   end for
13: end for
14: return $B_{mb}$, $y_{max}$
```

$$|B| = \prod_{a \in A} |ba(a)| \tag{3.5}$$

3.4.2 Heuristic algorithm

With the heuristic algorithm, only a small number of bindings from the total number of possibilities are explored. The bindings that are evaluated by the heuristic algorithm are generated by a two-phase procedure, *initial resource allocation* and *allocation optimization*. In the initial resource allocation, an initial binding of application actors to processing elements is derived. This initial binding later undergoes an optimization stage where the allocation of each actor is reconsidered to either improve the timing yield (the average throughput or the objective function for soft real-time applications) with the single-binding mapping approach or the throughput for each chip-frequency vector with the multiple-bindings mapping approach.

In the initial resource allocation, the actors whose execution time are likely to have a large impact on the throughput of an application, referred to as *critical actors*, are considered first. The criticality of an actor $a \in A$ is estimated by the product of its repetition vector $\kappa(a)$ (Definition 12) and average execution time (in cycles) (Definition 13) over all processing elements to which the actor can be bound (Definition 15), as formally defined in Definition 25. This is an approximate way of determining the criticality, as it intuitively estimates the average computational demand of an actor. Note that depending on the topology of an SDF graph, an actor a with the highest $ca(a)$, as given by Definition 25, may not necessarily be on the critical cycle of the graph and therefore, may not have the largest impact on the throughput of the graph. For an exact estimation of criticality, all the cycles in the equivalent HSDF graph have to be analyzed, which is computationally more expensive. In this thesis, the criticality estimation given by Definition 25 is chosen for several reasons. First, it is simple to implement and has low computational complexity. Second, mis-predictions in criticality estimation can be tolerated, as the mapping derived in the initial resource allocation stage is later improved in the allocation optimization stage, as described in this chapter. An implementation of a graph-topology aware criticality estimation is a possible future extension to this work.

Definition 25. *(Actor criticality) The criticality of an actor $a \in A$ in a resource-aware application graph ga is given by $ca : A \rightarrow \mathbb{Q}$, and is defined as*

$$\forall a \in A.\ ca(a) = \frac{\kappa(a)}{|ba(a)|} \cdot \sum_{pe \in ba(a)} EC(a, pe) \tag{3.6}$$

When allocating actors in a resource-aware application graph to processing elements in a platform graph, the initial resource allocation tries to balance the load in terms of execution time (in seconds) on the processing elements. The load of a processing element is computed by the sum of products of the repetition vector and the execution time (in seconds) of the actors bound to the processing element (Definition 26).

Definition 26. *(Load of a processing element) The load of a processing element $pe \in PE$ is given by $lpe : PE \rightarrow \mathbb{Q}$, and is defined as*

$$\forall pe \in PE.\ lpe(pe) = \sum_{\substack{a \in A \\ b(a) = pe}} \kappa(a) \cdot et(a, pe) \tag{3.7}$$

Algorithm 3 shows the heuristic algorithm for the single-binding mapping approach for firm real-time applications. In the first part of the algorithm (Lines 5–10), initial resource allocation is performed. The actors in a resource-aware application graph are sorted in decreasing order of criticality and are allocated to processing elements in a platform graph, such that the load on the processing elements is balanced. Each time an actor is to be bound to a processing element, the processing element with the lowest load is selected (Line 8). If several processing elements are not yet allocated or have equal loads, an actor is bound to the processing element with the highest mean frequency μ_g^{pe} of the normally distributed random variable f_g^{pe} describing the maximum supported frequency of the processing element. This is done to ensure that the actors with high criticality are allocated to processing elements with high computational power (processing elements with high mean frequencies μ_g^{pe} are faster on average). In the second part of the algorithm (Lines 15–27), allocation optimization is performed. The allocation of each actor, in increasing order of criticality, is changed from the processing element to which it is bound to all other processing elements (to which it can be bound) in sequence (Lines 15–19). Each time the allocation of an actor is changed, the new binding is evaluated for timing yield. If the timing yield is improved, the new allocation of the actor is kept, otherwise the actor is moved back to the processing element to which it was bound (Lines 20–24). This process continues until the allocation of all actors has been reconsidered. The algorithm returns a binding b_{sb} that has the highest timing yield $y_{max} = \gamma(ga, gb, b_{sb})$ among the bindings that have been explored. For best-effort and soft real-time applications, the functions $\tau_{avg}(ga, gb, b_{sb})$ and $\zeta(ga, gb, b_{sb})$ are used instead of $\gamma(ga, gb, b_{sb})$ to find a binding with an improved average throughput and reduced objective function, respectively.

Algorithm 3: Heuristic mapping algorithm implementing the single-binding mapping approach for firm real-time applications.

```
Require: $ga; gp; n \ \forall fi \in FI; (\mu_g^r, \sigma_g^r, \delta^r, \sigma_l^r) \ \forall r \in R$
 1: Compute $c(gp, fi, n) \ \forall fi \in FI$
 2: Derive $FC$
 3: $y_{max} \leftarrow 0$
 4: //Initial resource allocation
 5: $\forall pe \in PE$. initialize loads
 6: Sort $A$ in decreasing $ca(a)$
 7: for all $a \in A$ do
 8:    Bind $a$ to $pe$ with lowest $lpe(pe)$ or highest $\mu_g^{pe}$ (if equal loads)
 9:    Update the load of $pe$
10: end for
11: //initial binding $b_{sb}$ retrieved
12: $y_{max} = \gamma(ga, gp, b_{sb})$
13:
14: //Allocation optimization
15: for all $a \in A$ do
16:    $pe' = b_{sb}(a)$
17:    for all $pe \in ba(a)$ do
18:       if $pe \neq pe'$ then
19:          Bind $a$ to $pe$ //new binding $b_{sb}$ retrieved
20:          if $\gamma(ga, gp, b_{sb}) > y_{max}$ then
21:             $y_{max} \leftarrow \gamma(ga, gp, b_{sb})$
22:          else
23:             Revert the allocation of $a$
24:          end if
25:       end if
26:    end for
27: end for
28: return $b_{sb}$, $y_{max}$
```

The heuristic algorithm for the multiple-bindings mapping approach for firm real-time applications is illustrated in Algorithm 4. For each chip-frequency vector $fc \in FC$, the algorithm performs initial resource allocation and allocation optimization, such that a binding is found that satisfies the requirement t_{req}. The initial resource allocation approach (Lines 8–12) is similar to the one described for Algorithm 3. The difference lies in binding an actor to the processing element with the highest clock frequency $fc(pe)$, as specified by current chip-frequency vector, in case several processing elements are not yet allocated or have equal loads (Line 10). In the allocation optimization stage (Lines 16–35), the allocation of each actor in increasing order of criticality is reconsidered, as described before for Algorithm 3. Whenever a binding is found that satisfies the requirement t_{req}, the binding is stored in a set B_{mb} of output bindings and the timing yield y_{max} is incremented by the probability of the chip-frequency vector (Lines 18–21, 25–29). Note that the break statement on Line 27 is used to stop the optimization process when the throughput requirement is satisfied. The algorithm returns the set of output bindings and an estimated timing yield. For best-effort applications, the initial resource allocation and allocation optimization are performed for each chip-frequency vector to find a binding with a maximized throughput. For soft real-time applications, a binding is found for each chip-frequency vector, such that the requirement t_{req} is satisfied or otherwise the throughput degradation is minimized.

With the heuristic algorithm, the number n_b of bindings to evaluate for timing yield or for throughput for each $fc \in FC$ is given by Equation (3.8). The complexity of the algorithm is $O(|A|)$. Given a large problem, n_b is considerably lower than the total number $|B|$ of bindings explored by the exhaustive algorithm (Equation (3.5)). For example, one of our test applications that has sixteen actors is allocated to a platform consisting of three processing elements. With the exhaustive algorithm, the overall number of bindings to be evaluated is $|B| = 3^{16} = 43046721$ (given that each actor can be bound to any processing element), while only $16 \cdot 2 = 32$ bindings are explored by the heuristic algorithm. The proposed heuristic algorithm thus addresses the scalability problem of the exhaustive mapping algorithms from the perspective of the number of bindings that must be evaluated. However, given a large number of voltage-frequency islands $fi \in FI$ in the presence of a large number processing elements, and a large number of clock-frequency levels $c(gp, fi, n)$ (Definition 4) per voltage-frequency island, the number of chip-frequency vectors FC (Definition 6) becomes very large, resulting in a very large number of throughput evaluations. This limits the scalability of the proposed mapping algorithms. The treatment of this scalability issue is possible future work for this thesis.

$$n_b = \sum_{a \in A} (|ba(a)| - 1) \tag{3.8}$$

Algorithm 4: Heuristic mapping algorithm implementing the multiple-bindings mapping approach for firm real-time applications.

```
Require: $ga; gp; n\ \forall fi \in FI; (\mu_g^r, \sigma_g^r, \delta^r, \sigma_l^r)\ \forall r \in R$
 1: Compute $c(gp, fi, n)\ \forall fi \in FI$
 2: Derive $FC$
 3: $y_{max} \leftarrow 0$
 4: $B_{mb} \leftarrow \emptyset$
 5: Sort $A$ in decreasing $ca(a)$
 6: for all $fc \in FC$ do
 7:    //Initial resource allocation
 8:    $\forall pe \in PE$. initialize loads
 9:    for all $a \in A$ do
10:       Bind $a$ to $pe$ with lowest $lpe(pe)$ or highest $fc(pe)$ (if equal loads)
11:       Update the load of $pe$
12:    end for
13:    //initial binding $b$ retrieved
14:
15:    //Allocation optimization
16:    for all $a \in A$ do
17:       $pe' = b(a)$
18:       if $\tau(gb) \geq t_{req}$ then
19:          $B_{mb} \leftarrow B_{mb} \cup \{b\}$ //save $b$
20:          $y_{max} \leftarrow y_{max} + pc(gp, fc)$
21:       else
22:          for all $pe \in ba(a)$ do
23:             if $pe \neq pe'$ then
24:                Bind $a$ to $pe$ //new binding $b$ retrieved
25:                if $\tau(gb) \geq t_{req}$ then
26:                   $B_{mb} \leftarrow B_{mb} \cup \{b\}$ //save $b$
27:                   $y_{max} \leftarrow y_{max} + pc(gp, fc)$
28:                   break
29:                else
30:                   Revert the allocation of $a$
31:                end if
32:             end if
33:          end for
34:       end if
35:    end for
36: end for
37: return $B_{mb}$, $y_{max}$
```

3.5 Experimental results

In this section, we use the presented algorithms to map SDF graphs modeling real multimedia and DSP applications to a NoC-based MPSoC platform considering the impact of process variation on the hardware resources. We demonstrate the average throughput, the timing yield, and the average throughput degradation, as a result of the exhaustive and heuristic mapping algorithms implementing the single-binding and multiple-bindings mapping approaches for best effort and real-time applications. We analyze how well the heuristic algorithms perform, compared to the optimum results provided by the exhaustive algorithms. Additionally, we illustrate how allocating more resources in the interconnect (NoC) and thus providing higher bandwidth for communication impacts the timing yield considering firm real-time applications. We proceed by describing the experimental setup.

3.5.1 Experimental setup

The presented variation-aware mapping algorithms for best-effort and real-time applications are evaluated on a set of six SDF graphs that model real DSP and multimedia applications. From the DSP domain, the set contains a Sample rate converter and a Modem [7], and from the multimedia domain an H.263 encoder [63], an H.263 decoder [82], an MP3 playback [91] and an MP3 decoder [82]. An overview of the SDF graphs is shown in Table 3.2. The table reports the number of actors, the number of cycles (feedback loops) in each graph and the available parallelism in terms of the minimum number of processing elements that can fully exploit it. The topologies of the applications, together with the execution times (in cycles) of actors and the size of data tokens (in bytes) sent across dependency edges, are given in Appendix B. These application SDF graphs are the resource-aware application graphs in our formal framework. For simplicity, we interpret the same six applications as best effort, firm real-time and soft real-time when evaluating the mapping algorithms for the different application classes.

Table 3.2: Application SDFG overview.

SDF graph	Nr. actors	Nr. cycles	Parallelism (Nr. PEs)
H.263 decoder	4	0	3
H.263 encoder	5	1	3
MP3 playback	4	1	2
Sample rate	6	0	3
Modem	16	5	3
MP3 decoder	14	0	3

The described applications are allocated to a NoC-based MPSoC platform consisting of three homogeneous processing elements (most of the applications have a parallelism of three processing elements). The MPSoC platform, described by a platform graph in our modeling framework, is shown in Figure 3.6. The three processing elements and the interconnect (NoC) are placed in separate voltage-frequency islands; as such, there are four islands. The number of clock-frequency levels provided by clock-generation units (not shown in Figure 3.6) to each of the four voltage-frequency islands is chosen to be five. The interconnect consists of two routers, three network interfaces and multiple links, which connect the processing elements to each other. Table 3.3 shows the parameters assumed for the interconnect. The net bandwidth (i.e. the overall bandwidth capacity) of each connection in the interconnect is assumed to be four bytes per cycle. The flit size is set to twelve bytes, and a TDM table size of twenty slots for each network interface is assumed. Note that the net bandwidth may be affected by possible overheads, such as inserted headers. To account for this, we assume a worst-case scenario, where four out of twelve bytes in a flit is a header. Therefore, the net bandwidth used for transferring data becomes two-thirds of the original bandwidth.

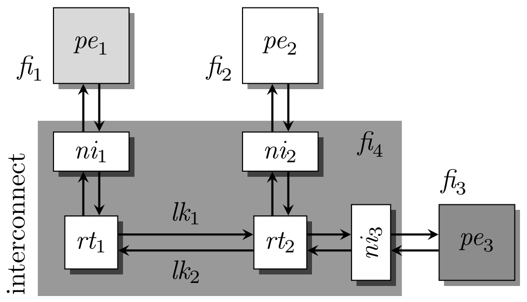


Figure 3.6: The multi-processor platform used in the experiments. It consists of three processing elements connected to each other by an interconnect. The processing elements and the interconnect are placed in separate voltage-frequency islands.

Table 3.3: Parameters assumed for connection bandwidth, TDM slot-table size and flit size for the interconnect.

β_c (bytes/cycle)	sz_{tb}	sz_{fl} (bytes)
(2/3)·4	20	12

Table 3.4 illustrates the parameters assumed for the hardware resources. The mean of the random variable describing the maximum supported frequency of each processing element due to global variation is 300 MHz. For routers and network interfaces, a 500 MHz mean is chosen. The choice of the mean frequency for links, compared to the mean for routers, is made according to a NoC implementation given in [29], where the authors design a link slightly faster than the routers not to

limit the performance of the NoC and to preserve power. We select a mean of 560 MHz for links. As reported in [20], at high computing frequencies (in the order of GHz), the variation is large. In our work, we target embedded systems running streaming applications, where the typical frequencies are in the order of hundreds of MHz. Measurements at 45 nm technology in the frequency range of interest are provided by Pang *et al.* [64]. Based on this data, we assume standard deviation to mean ratios of 4% and 3.3% for within-die and die-to-die variations, respectively (Table 3.4). We assume a 5% reduction in mean frequency for processing elements and links due to multiple critical paths and wires, respectively. For routers and network interfaces, no reduction in mean frequency is assumed. These choices correspond to the discussion presented in Chapter 2.2.

Table 3.4: Parameters assumed for random variables modeling the maximum supported frequency of hardware resources.

	pe	*rt*	*ni*	*lk*
μ_g^r (MHz)	300	500	500	560
σ_g^r (% of μ_g^r)	4	4	4	4
δ^r (% of μ_g^r)	5	0	0	5
σ_l^r (% of μ_g^r)	3.3	3.3	3.3	3.3

3.5.2 Evaluation results

Below, we demonstrate how the proposed algorithms, implementing the single-binding and multiple-bindings mapping approaches, perform when allocating the set of applications given in Table 3.2 to the multi-processor platform shown in Figure 3.6. The results are presented for best-effort, soft real-time and firm real-time application classes with corresponding optimization criteria. We show that the multiple-bindings mapping approach provides better results, compared to the single-binding approach. Both exhaustive and heuristic mapping algorithms are evaluated. The exhaustive algorithms are applied to a subset of small to medium size applications, as they are are too computationally expensive for applications having a large number of actors (Modem, MP3 decoder). The heuristic mapping algorithms are evaluated on the complete set of applications. For the set of small to medium size applications, we analyze how the heuristic mapping algorithms perform, as compared to the optimum results provided by the exhaustive algorithms. We additionally compare the algorithms for the single-binding mapping approach to a mean-frequency based mapping, where the binding of actors to processing elements is derived based on the mean frequencies μ_g^r of the processing elements. The purpose is to see if the lower complexity mean-frequency based mapping can provide similar results as the single-binding mapping algorithm without considering the variation in the mapping process.

Firm real-time applications

Figure 3.7 illustrates the timing yield for the H.263 decoder, H.263 encoder, MP3 playback and Sample rate converter applications, as a result of mean frequency-based, single-binding and multiple-bindings exhaustive mapping algorithms for the class of firm real-time applications. As shown in Table 3.2, these applications have a small to medium number of actors (6 actors the largest), enabling the use of the exhaustive mapping algorithms. The exhaustive algorithms for the single-binding and multiple-bindings mapping approaches are denoted by SBE and MBE, respectively. The lower complexity exhaustive mean-frequency based mapping algorithm is denoted by MFBE. With this mapping algorithm, all mapping possibilities are evaluated with the assumption that the processing elements possess mean frequencies given by μ_g^r. There can be multiple bindings of actors to processing elements that satisfy the throughput requirement. A binding that just satisfies the requirement can potentially result in a low timing yield, as any negative deviation from the mean frequency of the processing elements can lead to a violation. For this reason, a binding that provides the highest throughput among all bindings is chosen. Later, the timing yield of the selected binding is estimated by the formula given in Definition 22. Note that, there can be multiple bindings that provide the same highest throughput. Evaluating them all for timing yield and selecting the one that provides the highest timing yield would increase the complexity and the run time of the algorithm, which is not the intention of introducing the algorithm (instead single-binding heuristic mapping algorithms can be used). Therefore, one of the bindings that provide the highest throughput is arbitrarily selected and evaluated for timing yield.

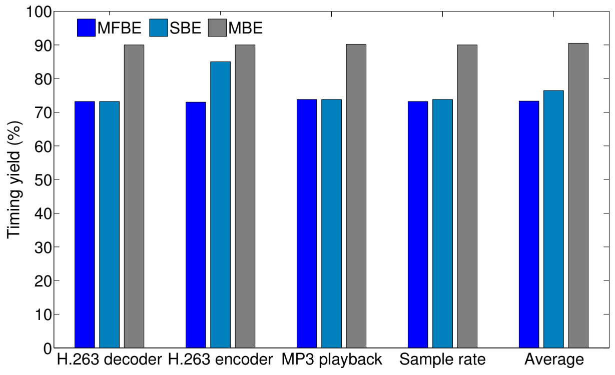


Figure 3.7: Timing yield of applications using the exhaustive MFBE, SBE and MBE mapping algorithms for the class of firm real-time applications.

Figure 3.7 shows that SBE provides better results than MFBE for the H.263 encoder and Sample rate converter applications, providing a 12% higher timing yield for H.263 encoder. For Sample rate converter, the improvement is negligible. For the rest of the applications, MFBE results in an identical timing yield as SBE. Therefore, we observe that the bindings providing a high throughput for mean frequencies of processing elements typically result in high timing yield. However, as shown with the H.263 encoder application, it can be necessary to consider the variation in the frequencies of processing elements in the binding process for finding a better solution. Moreover, processing elements may have identical mean frequencies, but different frequency distributions (in our setup, we assume identical frequency distributions for processing elements). Without considering the variation in frequencies, mean-frequency based mapping algorithms can provide low timing yield. As expected, MBE performs better than SBE, resulting in improvements for all applications. Timing yield improvements of up to 17% compared to SBE are reported. Figure 3.7 additionally illustrates the average timing yield for the set of applications. The exhaustive mean-frequency based mapping algorithm (MFBE) results in a 73% average timing yield, which is improved to 76% and 90% by SBE and MBE, respectively. These results show that having multiple bindings for the chip population provides higher timing yield in the presence of process variation, compared to the case where only a single binding is used. We also observed that lower-complexity mean-frequency based mapping, which does not consider the probability distribution of frequency, on average provides results close to SBE. However, without the knowledge of variation, it can result in a lower timing yield, as demonstrated with one of our benchmark applications in Figure 3.7. The run time of the exhaustive algorithms on a 3.07 GHz machine is reported in Table 3.5. As can be seen, the mean-frequency based algorithm (MFBE) is considerably faster than the variation-aware algorithms (SBE and MBE). The run time of the algorithm for multiple-bindings mapping approach is lower than the one for the single-binding approach, as not all bindings are evaluated in the allocation optimization stage (Algorithm 2). The optimization stops whenever the throughput requirement is satisfied.

Table 3.5: The run time of exhaustive MFBE, SBE and MBE mapping algorithms.

	MFBE (sec.)	SBE (sec.)	MBE (sec.)
H.263 decoder	13	870	224
H.263 encoder	4	640	63
MP3 playback	136	9694	2716
Sample rate	181	20716	7097

The timing yield achieved by the variation-aware and mean-frequency based heuristic mapping algorithms for the complete set of applications is shown in Figure 3.8. The mean-frequency based mapping algorithm, denoted by MFBH, is implemented by the same heuristic algorithm presented in Section 3.4.2. With

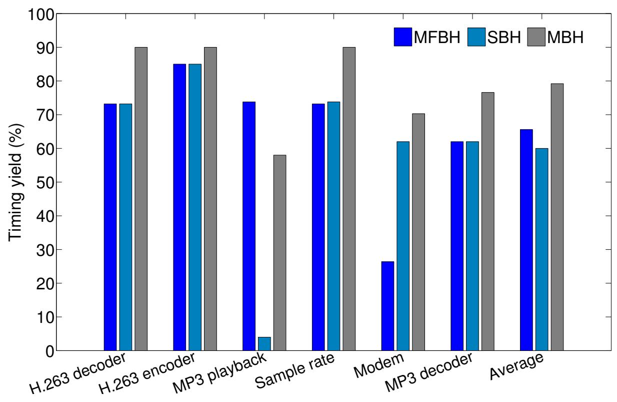


Figure 3.8: Timing yield of applications using the heuristic MFBH, SBH and MBH mapping algorithms for the class of firm real-time applications.

this mapping algorithm, application actors, in decreasing order of criticality (Definition 25), are initially allocated to processing elements based on their mean frequencies. The derived initial binding is later optimized, in the allocation optimization stage, for higher throughput. The heuristic MFBH mapping algorithm results in the same timing yield, as the exhaustive MFBE mapping algorithm (Figure 3.7) for the H.263 decoder, H.263 encoder and Sample rate applications. For H.263 encoder, the provided timing yield is higher by 12%. The reason is that twelve bindings are found by the exhaustive MFBE algorithm that provide the highest throughput for mean frequencies, and only one of the bindings is selected arbitrarily. The binding which is selected provides 73% timing yield, while there are other bindings that provide 85% timing yield. In principle, all bindings found by MFBE can be evaluated for timing yield, such that the binding providing the highest timing yield can be selected. However, by doing so, we loose the benefit of having a fast mean-frequency based mapping algorithm (instead SBH may be used). The heuristic MFBH algorithm finds a single binding that results in 85% timing yield. For the H.263 decoder, H.263 encoder and Sample rate applications, the heuristic mapping algorithm implementing the single-binding mapping approach, denoted by SBH, provides the optimum results found by the computationally expensive exhaustive SBE mapping algorithm, the results of which are shown in Figure 3.7. With MP3 playback, SBH results in only 4% timing yield, compared to the 74% provided by SBE. With this application, SBH is not able to find the binding selected by SBE, resulting in a significant reduction in timing yield. Similar to SBH, the heuristic algorithm for the multiple-bindings mapping approach, denoted by MBH, gives the optimum result with the H.263 decoder, H.263 encoder and Sample rate applications. With MP3 playback, the provided timing yield is 58%, compared to the 90% given by the exhaustive MBE algorithm. For this application and MBH algorithm given in Algorithm 4, we observed that the bindings that provide high throughput for some of the chip-frequency vectors (Algorithm 4), but are not found for these chip-frequency vectors (reducing the yield), are in fact located for other chip frequency vectors. We combined all the bindings found by the algorithm and evaluated the timing yield for the precomputed bindings. This provided the same 90% timing yield. As such, although the reported timing yield with MBH is only 58%, in reality, the selected set of bindings provide 90% timing yield. For the rest of the applications, the reported timing yield with MBH and the provided timing yield with the selected set of bindings are identical. These results show that the heuristic algorithm performs well on average. It is able to find the optimum for all applications in question, except MP3 playback.

Figure 3.8 additionally illustrates the timing yield achieved by the heuristic mapping algorithms for the Modem and MP3 decoder applications of a larger size (16 actors the largest). For Modem, the timing yield with the variation-unaware MFBH algorithm is only 26%. In contrast, SBE provides a considerably higher timing yield of 62%. This is further improved to 70% by MBH. For the MP3 decoder application, both MFBH and SBH provide the same 62% timing yield, while 77% is given with MBH. The figure also shows the average timing yield for the complete set of applications. Table 3.6 shows the run time of the heuristic mapping algorithms. Significant reductions, compared to the run time of the exhaustive algorithms can be seen. Note that the exhaustive mapping algorithms for Modem and MP3 decoder applications are unfeasible and do not finish in reasonable time. These results show that the heuristic algorithms can be efficiently applied to problems of a medium to large size.

Table 3.6: The run time of heuristic MFBH, SBH and MBH mapping algorithms.

	MFBH (sec.)	SBH (sec.)	MBH (sec.)
H.263 decoder	11	157	57
H.263 encoder	3	47	9
MP3 playback	121	1128	841
Sample rate	29	855	124
Modem	9	285	138
MP3 decoder	75	2088	51

In Table 3.7, we analyze the number of bindings selected by the heuristic algorithm implementing the multiple-bindings mapping approach. This is relevant to observe, as these bindings must be stored to be used in a configuration stage, where one of the stored bindings is selected based on the chip. As can be seen a low number of (different) bindings are required to be stored. For the Modem applications, 106 non-identical bindings are returned. However, there are only 23 bindings that provide the highest throughput for all chip-frequency vectors;

we refer to these bindings as dominant. The rest of the bindings do not provide a higher throughput for any of the chip-frequency vector. Therefore, only 23 bindings need to be stored for Modem. This shows the applicability of the multiple-bindings mapping approach as it provides significant improvements with a low impact on storage requirements.

Table 3.7: The number of bindings selected by MBH.

	Total	Dominating bindings (if less than total #)
H.263 decoder	17	–
H.263 encoder	12	–
MP3 playback	9	–
Sample rate	6	–
Modem	106	23
MP3 decoder	6	–

Lowering communication latency

As detailed in Section 2.5, the number of slots allocated in the TDM table of a connection in the interconnect for each dependence edge mapped on this connection depends on the required bandwidth of the edge, as given by Equation (2.14). In the experiments above, only a single TDM slot in the table of a connection, for each dependency edge mapped on the connection is allocated. By allocating more slots and thus providing lower communication latency in the interconnect, a higher timing yield may be achieved. For the class of firm real-time applications, we analyze the impact of allocating more interconnect resources on the timing yield. Figure 3.9 illustrates the impact of increasing the number of allocated TDM slots, for each dependency edge mapped on a connection in the interconnect, on the timing yield for the H.263 decoder, H.263 encoder and MP3 decoder applications. The graphs given by H.263 decoder (a), H.263 encoder (a) and MP3 decoder are derived based on the bindings that are selected by the variation-aware heuristic SBH mapping algorithm. As these graphs show, allocating more slots has no effect on the timing yield. This is due to the bindings derived by SBH. Application actors which communicate large amounts of data in a single iteration of the application graph are bound to the same processing element. Binding these actors to different processing elements and thus the dependency edge connecting the to a connection in the interconnect reduces the timing yield. As the variation-aware mapping algorithms aim at maximizing the timing yield, they rule out the bindings where these actors are bound to different connections. We conclude that for the graphs given by H.263 decoder (a), H.263 encoder (a) and

MP3 decoder, the actors modeling the latency and rate of communication are not on the critical cycle of the graph, and thus reducing the latency does not improve the timing yield. To strengthen this observation, Figure 3.9 additionally shows how the timing yield for H.263 decoder and H.263 encoder changes when allocating a higher number of TDM slots, given a binding where dependency edges that incur large amounts of data communication are allocated to a connection in the interconnect. These graphs are shown by H.263 decoder (b), H.263 encoder (b). With a single allocated slot, these bindings provide 30% and 6% timing yield for H.263 decoder and H.263 encoder, respectively. The increase in the number of allocated TDM slots provides considerable improvements in the timing yield, such that 73% and 75% timing yield is achieved with three and five slots for H.263 decoder and H.263 encoder, respectively. Note that depending on the application, not always a binding can be derived by the variation-aware mapping algorithm, such that no dependency edge with associated with large data amounts is bound to a connection in the interconnect. In this scenario, allocating more resources on connections in the interconnect can help to increase the timing yield.

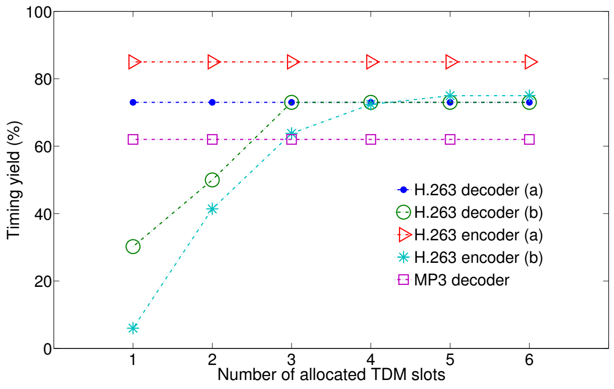


Figure 3.9: Timing yield against the number of allocated TDM slots for each dependency edge mapped to a connection in the interconnect for the H.263 decoder, H.263 encoder and MP3 decoder applications.

Best-effort applications

Figure 3.10 illustrates the normalized average throughput for the H.263 decoder, H.263 encoder, MP3 playback and Sample rate converter applications provided by the exhaustive MFBE, SBE and MBE algorithms for the class of best-effort applications. Both MFBE and SBE perform equal to each other and result in the same average throughput. The algorithm implementing the multiple-bindings mapping approach provides a 3% higher average throughput for all applications. We thus observe that the benefits of having multiple bindings for manufactured chips, in the presence of process variation, are lower for best-effort applications, compared to firm real-time application. For this class of applications, a single binding may be preferred over multiple bindings if the cost of storing the multiple bindings, together with the information on which binding needs to be applied to each chip-frequency vector, is high. To experimentally illustrate the benefits of having different optimization criteria for firm real-time and best effort applications, consider the H.263 encoder application. A binding found by the SBE algorithm targeting to maximize the average throughput (Definition 21) provides the highest average throughput for the H.263 encoder application. This result is reported in Figure 3.10. For the same application, the same binding provides only 73% timing yield (this is shown in Figure 3.7 by MFBE). However, the SBE algorithm targeting to maximize the timing yield (Definition 22) provides a higher timing yield of 85% (see SBE in Figure 3.7). This experimentally shows that a binding preferred for a best-effort application may not be good for a firm real-time application.

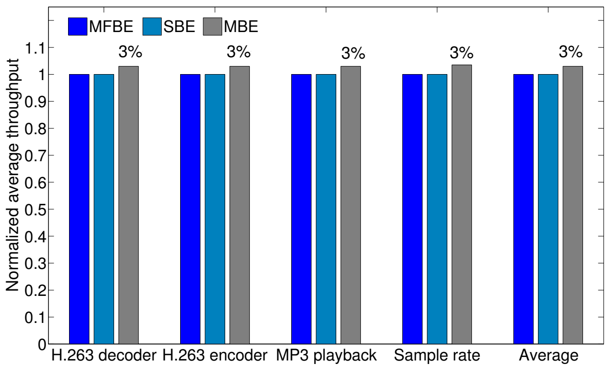


Figure 3.10: Normalized average throughput of applications using the exhaustive MFBE, SBE and MBE mapping algorithms for the class of best-effort applications.

The normalized average throughput with the heuristic mapping algorithms for the complete set of applications is illustrated in Figure 3.11. The results are normalized to the results of MFBE in Figure 3.10, such that the difference between exhaustive and heuristic algorithms can be captured. For all small to medium size applications, except the MP3 playback, the optimum results are provided by both MFBH and SBH algorithms. With MP3 playback, SBH is unable to find a binding that results in the highest average throughput, resulting in a 14% lower average throughput. The MBH algorithm provides the optimum result for only H.263 decoder. For H.263 encoder, MP3 playback and Sample rate converter, 2%, 6% and 1% reductions in the average throughput are seen, compared to the result of MBE. Similar results are also reported for larger in size Modem and MP3 decoder applications.

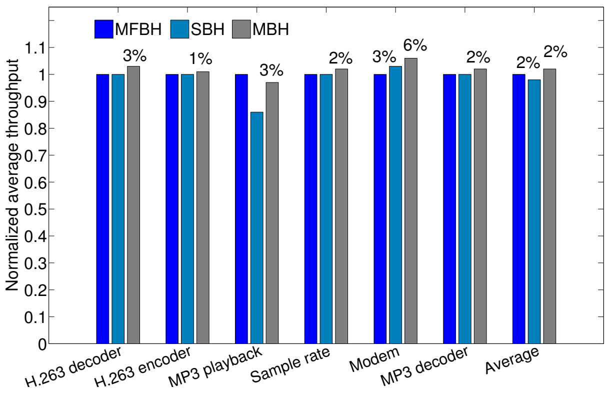


Figure 3.11: Normalized average throughput of applications using the heuristic MFBH, SBH and MBH mapping algorithms for the class of best-effort applications.

Soft real-time applications

As explained in Section 3.2.3, the optimization objective for soft real-time applications is to improve the yield and/or reduce the average throughput degradation, as the chips with a low degradation can be used in this class of applications. Note that, with the single-binding mapping approach, a binding with a particular (yield, average throughput degradation) value pair is selected based on minimized objective function, as presented in Section 3.2.3. In this section, we illustrate the yield and the average throughput degradation (measured in percents of the throughput requirement) for the set of applications as a result of mean-frequency based and variation-aware mapping algorithms. With the exhaustive MFBE, SBE and MBH mapping algorithms, the yield results for (the small to medium size) soft real-time applications are identical to the ones given for firm real-time applications in Figure 3.7. In fact, identical bindings are selected in both cases. We thus observe that, for the given SBE and MBE mapping algorithms and the set of applications, the same bindings provide the best results for both soft and firm real-time applications. The average throughput degradation as a result of MFBE, SBE and MBH for the subset of applications is shown in Figures 3.12. It can be seen that the mean-frequency based MBE and variation-aware SBE mapping algorithms provide identical average throughput degradation for all applications, except for H.263 encoder. Although the binding selected by MFBE has a slightly lower average throughput degradation, it has a 12% lower timing yield. Compared to SBE, MBE provides improvements in the average throughput degradation of only 1% on average, whereas the timing yield with MBE is improved by 14% on average (see Figue 3.7).

Figures 3.13 and 3.14 show the yield and the average throughput degradation for the heuristic MFBH, SBH and MBH algorithms and the complete set

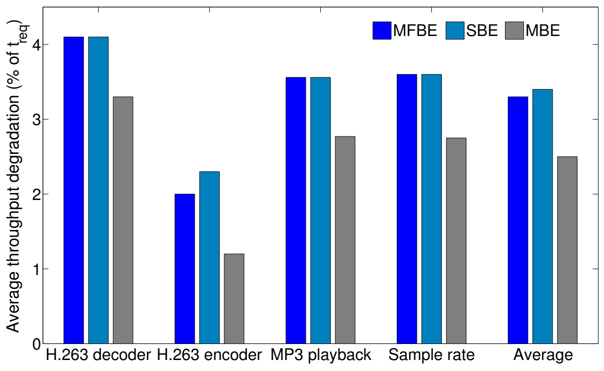


Figure 3.12: Average throughput degradation of applications using the MFBE, SBE and MBE heuristic mapping algorithms for the class of soft real-time applications.

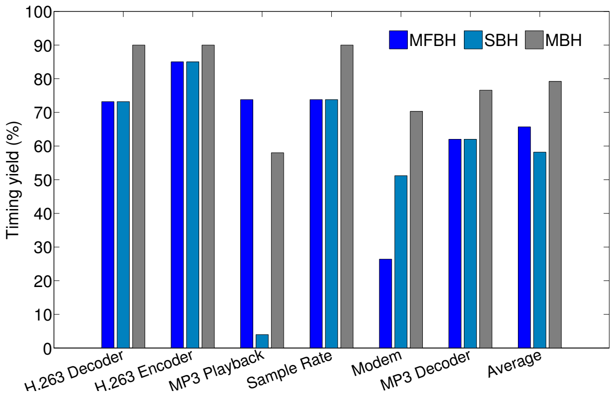


Figure 3.13: Timing yield of applications using the MFBE, SBE and MBE heuristic mapping algorithms for the class of soft real-time applications.

of applications. The yield results are similar to the ones given for firm real-time applications in Figure 3.8. One of the difference is seen for the Sample rate converter application, where the algorithm for soft real-time optimization provides a slightly higher timing yield. For the Modem application, SBE results in a 51% timing yield. As such, it is unable to find the binding selected by the same algorithm for firm real-time optimization that gives 62% timing yield. On average, MBH improves the timing yield of MFBE by 21%. The heuristic mapping algorithms result a negligible increase in the average throughput degradation for the subset of applications, compared to the optimum results given by MBE. The only exception is the MP3 playback application. As can be seen from Figure 3.14, the selected binding that provides a low yield of 4% for the MP3 playback application, results also in a high average throughput degradation of 18%. For the same application, MBH provides a 3% higher average throughput degradation, compared to MBE.

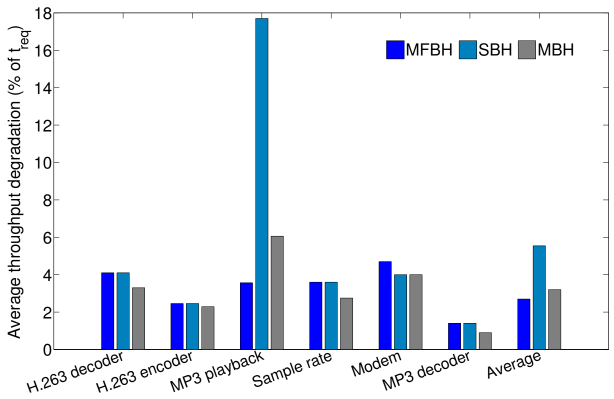


Figure 3.14: Average throughput degradation of applications using the MFBE, SBE and MBE heuristic mapping algorithms for the class of soft real-time applications.

3.6 Summary

In this chapter, we described the requirements of best-effort and real-time applications. In the presence of process-induced variations, we formalized a mapping optimization problem for each of them. Two mapping approaches were presented. The first approach is to have a single binding for all manufactured multi-processor chips, which possess different process-induced variations in the maximum-supported frequency of the hardware resources. With the second mapping approach, multiple different bindings of the actors in an application to the processing elements in the multi-processor platform in the manufactured chips are present. This mapping approach makes use of the faster and slower hardware resources on the same chip die due to within-die variation, which affects the maximum-supported frequency of resources on the same die differently. Thus, the mapping of the application to the platform can be tailored based on process-induced variations in each manufactured chip, resulting in better performance. To implement the mapping optimization problems with the defined objective functions (e.g. maximizing timing yield for firm real-time applications), the chapter also presented exhaustive and heuristic algorithms. From the perspective of the number of bindings that are evaluated, the heuristic mapping algorithm effectively addresses the scalability problem of the exhaustive algorithm. These algorithms are used to derive a binding or a set of bindings at design time, such that the corresponding objective function is optimized. In the case of multiple bindings, a one-time system configuration through the operating system takes place for each manufactured chip at run time, such that the right binding from the set of computed bindings is selected based on the process-induced variations in a chip.

The presented algorithms implementing both mapping approaches with a single and multiple bindings were evaluated on a set of SDF models of real applications. As expected, multiple bindings make use of fast and slow processing elements on a chip die due to within-die variation and provide better results than a single binding for the manufactured chips. An average of 19% higher timing yield is reported for the set of applications considering them as firm real time. We observed that the benefits of multiple bindings are much lower for best-effort applications, providing only a 4% improvement in the average throughput. The results for soft real-time applications are similar to the results for firm real-time and best-effort applications with respect to timing yield and average throughput degradation, respectively. Interestingly, a small number of bindings are required to be stored for an on-line selection with the multiple-bindings mapping approach. At most 23 bindings are derived considering all the applications. This shows that the multiple-bindings mapping approach imposes low storage requirements, while providing significant improvements in timing yield, compared to the single-binding mapping approach.

Allocating more resources on the interconnect in a multi-processor platform, and thus providing lower communication latency may increase the timing yield and/or the average throughput. Analysis of this is provided in the experimental section for firm real-time applications. An observation was made that the single-binding mapping algorithm derives a binding, where the application's actors communicating large amounts of data are allocated to the same processing element. This is the result of the algorithm, which aims at maximizing the timing yield. For this reason, allocating more interconnect resources does not increase the timing yield for the given set of applications. However, depending on an application, a binding may not always be derived where no two actors communicating large amounts of data are bound to different processing elements. In this scenario, allocating more resources on the interconnect will improve the timing yield, as experimentally shown in this chapter.

The same chapter also provided an evaluation of the heuristic mapping algorithms, compared to the exhaustive mapping algorithms, which provide the optimum results. This evaluation is performed on a set of four small to medium size applications, which enable the use of the exhaustive mapping algorithms. For the MP3 playback application, the heuristic mapping algorithm implementing the single-binding mapping approach leads to a considerable 70% lower timing yield for firm real-time applications. For the rest of the applications, the optimum result is provided. Similarly, considering the multiple-bindings mapping algorithms, optimum results are found by the heuristic algorithms for three out of the four applications. Similar accuracy of the heuristic mapping algorithm is observed considering best-effort and soft real-time applications. Concerning the run time of the algorithms, the heuristic mapping algorithms provide improvements of up to 24 and 57 times for single-binding and multiple-bindings mapping algorithms, respectively.

4 Voltage-frequency island partitioning

In a multi-processor platform, a clock-generation unit is associated with each voltage-frequency island. Each clock-generation unit provides a set of clock-frequency levels. The cost of a CGU in terms of area and power consumption depends on its implementation, which in turn is dependent on the desired number of clock-frequency levels and the step size between them. A large number of clock-frequency levels within a frequency range provided by clock-generation units to corresponding voltage-frequency islands in a multi-processor platform translates into a higher cost hardware (in terms of area and power consumption). Note that the actual maximum supported frequency of a hardware resource in a manufactured chip can be any value within the process-induced frequency spread. Thus, a large number of clock-frequency levels available to islands can better match the actual maximum supported frequencies of hardware resources. This leads to higher throughput values of the application mapped to the hardware platform in manufactured chips, which in turn result in higher timing yield. Similarly, the number of voltage-frequency islands in a multi-processor platform can be reduced, resulting in a smaller number of clock-generation units and thus a lower area (power consumption) cost. However, with a reduced number of voltage-frequency islands, multiple processing elements are placed in a single island. If one of the processing elements is slow due to within-die process variation, the whole island is operated at the low clock frequency supported by the slow processing element. This may reduce the throughput of the application mapped to the platform in manufactured chips, resulting in reduced timing yield. Therefore, it is important to partition the processing elements into voltage-frequency islands, such that a minimal reduction in the timing yield is obtained. In this chapter, we consider only firm real-time applications constrained by a throughput requirement, which cannot be violated, as described in Section 3.2.2. We first outline the possible architectures of a clock-generation unit based on the desired number of clock-frequency levels. Later, we introduce algorithms to partition the processing elements in a platform graph into multiple voltage-frequency islands, such that a high timing yield for the given number of islands is obtained. Note that the interconnect is always placed in a separate voltage-frequency island, as described in Section 2.1.

4.1 Outline of CGU architectures

Process variation results in a spread in the maximum supported frequency of hardware resources in a multi-processor platform. The actual maximum supported frequency of a hardware resource in a manufactured chip can be any value within this spread. With a large number of clock-frequency levels provided by clock-generation units to each voltage-frequency island in a platform graph, a better matching of the actual maximum supported frequencies of the hardware resources can be performed. This provides higher throughput values in manufactured chips, resulting in higher timing yield. On the other hand, the finer the granularity of clock-frequency levels, the higher the complexity and cost in terms of area (power consumption) of a clock-generation unit can be. In this section, we outline possible architectures for implementing a clock-generation unit depending on the number of clock-frequency levels. Note that we do not provide actual data on hardware cost, but rather show that a larger number of clock-frequency levels translates into a higher complexity and hardware cost. We proceed by presenting three kinds of clock frequency generation.

Frequency Division

The simplest way to generate clock frequencies is to use frequency dividers. With this approach, it is only possible to generate frequencies of $f_{clk}^{ref}/\mathbb{N}$, where f_{clk}^{ref} is the reference clock frequency. However, the process-induced spread in the maximum supported frequency of a hardware resource is not as large (e.g. 16% three standard deviations from the mean for 45 nm technology considering combined die-to-die and within-die variations) to accommodate multiple f_{clk}^{ref}, $f_{clk}^{ref}/2$, $f_{clk}^{ref}/3$, etc, clock frequencies with frequency division.

Programmable Ring Oscillators

Another approach to generate clock frequencies is to use programmable ring oscillators. A simple ring oscillator is composed of an odd number of inverters. The clock frequencies are generated by programming the length of the ring oscillator [46]. For a ring oscillator consisting of a chain of $m = 2 \cdot k + 1; k \in \mathbb{N}$ inverters, the frequency is given by Equation (4.1), where t_d is the inverter delay. For two adjacent ring oscillator lengths, $2 \cdot k + 1$ and $2 \cdot (k+1) + 1$, the step in the oscillating frequencies is relatively large [46]. For example, assuming an inverter delay of 100 ps (this is a representative number for 0.13 μm technology), the oscillating frequencies assuming 15, 17 and 19 inverters are 333, 294, and 263 MHz, respectively, resulting in step sizes of 39 and 31 MHz. Thus, this approach of clock-frequency generation can provide a small number of clock-frequency levels within the range of process-induced variation in the maximum supported frequency of a hardware resource.

$$f_{osc} = \frac{1}{2 \cdot m \cdot t_d} \tag{4.1}$$

Voltage Control

To generate clock frequencies with a higher resolution, the voltage supply of the ring oscillator is controlled. The relation between gate supply voltage (v_{dd}) and gate propagation delay [3] is given by Equation (4.2), where c_l is the load capacitance at the gate's output, v_{th} is the threshold voltage, and α_v is the velocity saturation index.

$$t_d \approx \frac{c_l \cdot v_{dd}}{(v_{dd} - v_{th})^{\alpha_v}} \tag{4.2}$$

By controlling the supply voltage of the gates of the ring oscillator, we can adjust the oscillating frequency. Voltage control is achieved by using on-chip or off-chip voltage regulators. Off-chip voltage regulators are separate board-level components, which utilize bulky filter components [96] (i.e. large inductors and capacitors). Therefore, it is impractical and cost prohibitive to have many off-chip voltage regulators. A fully-digital on-chip voltage regulator is proposed in [44]. The supply voltage regulation is achieved by creating a voltage drop on power-transistor segments, which are connected to the circuit of the ring oscillator sequentially. By carefully choosing the number and the size of power-transistor segments, different voltage drops on the transistor segments and therefore on the circuit of the ring oscillator is achieved. The disadvantage of using on-chip voltage regulators is the reduced power-conversion efficiency (power dissipation on transistor segments) compared to off-chip regulators. The power transistors are of high threshold-voltage type. In the implementation presented in [44], the transistors have a large width of 10 μm. Therefore, a large number of clock-frequency levels comes at the cost of a large number of large transistor segments.

4.2 Heuristic Partitioning Algorithms

In this section, we present heuristic algorithms to partition the processing elements in a platform graph into multiple voltage-frequency islands. We consider only the class of firm real-time applications. Therefore, the objective of the partitioning algorithms is to derive a partitioning that results in high timing yield, which represents the percentage of manufactured chips satisfying the throughput requirement of an application. The algorithms for soft real-time and best-effort applications are similar to the ones presented in this section, and can be derived based on the discussions presented in Section 3.4. The heuristic algorithms in this section are given assuming a single and multiple bindings for all chips. As the presented model of a multi-processor platform suggests, the interconnect is never grouped with a processing element in the same voltage-frequency island, but is always placed in an unshared voltage-frequency island (see Section 2.1). Thus, the partitioning is only performed for the processing elements in a platform graph. This choice is motivated as follows. The interconnect is usually operated at higher clock frequencies, so that the latency for sending data across the connections in the interconnect is low, thus avoiding the bottleneck of communication. Placing the interconnect in a voltage-frequency island with other processing elements will slow down the interconnect. Low latency of communication can also be achieved by reserving large amount of resources (slots in the TDM slot table) on a connection. However, this may deprive other dependency edges bound to the same connection of the same or a different application from getting sufficient resources, resulting in long communication latency.

4.2.1 Single binding

To guide the process of merging multiple voltage-frequency islands into a single island, such that fewer partitions are created, we define a metric called *island criticality*, which assumes a given binding of the actors in a resource-aware application graph to processing elements in a platform graph. This metric quantifies the sensitivity of the throughput of a bound application graph to the clock frequency of a voltage-frequency island. The higher the island criticality of a voltage-frequency island, the higher the reduction in the throughput of a bound application graph is when the clock frequency of the island is reduced. A voltage-frequency island with a low criticality value suggests that the actors, which are bound to the processing elements in the island, are either not on the critical cycle or have low execution times on the cycles of the bound application graph. Island criticality of a voltage-frequency island for a given binding is defined as the reduction in the throughput of a bound application graph as a result of reducing the clock frequency of the island from the highest to the lowest value among its clock-frequency levels, while the clock frequency for the rest of the voltage-frequency islands in the platform graph is set to their maximum values (Definition 27). Merging a voltage-frequency island with a high criticality value with other islands into a single partition can result in high reductions in the throughput of manufactured chips. This is because the clock frequency of the voltage-frequency island is reduced due to the presence of other processing elements in the island (the clock frequency of the island is decided by the slowest processing element in it). This in turn can result in low timing yield.

Definition 27. *(Island criticality) The criticality of a voltage-frequency island $fi \in FI$ in a platform graph $gp \in GP$, for a given resource-aware application graph $ga \in GA$ and a binding $b \in B$, is given by $ci(ga, gp, b, fi) : GA \times GP \times B \times FI \rightarrow \mathbb{R}^+$, and is defined as*

$$ci(ga, gp, b, fi) = \frac{\tau(gb_1) - \tau(gb_2)}{\tau(gb_1)} \tag{4.3}$$

where $gb_1 = \langle sdfg, ga, gp, b, fc_1 \rangle$, $gb_2 = \langle sdfg, ga, gp, b, fc_2 \rangle$; chip-frequency vectors fc_1 and fc_2 are given as

$$\begin{aligned} \forall fi_i \in FI.\ fc_1(fi_i) &= \max_{f_{clk} \in c(gp, fi_i, n)} f_{clk} \\ \forall fi_i \in FI.\ fc_2(fi_i) &= \begin{cases} \max\limits_{f_{clk} \in c(gp, fi_i, n)} f_{clk} & fi_i \neq fi \\ \min\limits_{f_{clk} \in c(gp, fi_i, n)} f_{clk} & fi_i = fi \end{cases} \end{aligned}$$

Algorithm 5 demonstrates our proposed heuristic algorithm for partitioning processing elements in a platform graph into voltage-frequency islands. The algorithm requires as input a resource-aware application graph ga, a platform graph gp, a binding b_{sb}, a specified number n of clock-frequency levels for each voltage-frequency island, and $\mu_g^r, \sigma_g^r, \delta^r, \sigma_l^r$ values for the random variables describing the maximum supported frequency of each hardware resources $r \in R$. Initially, all processing elements in the platform graph are placed in separate voltage-frequency islands. The binding b_{sb} is derived using the variation-aware heuristic algorithm implementing the single-binding mapping approach (Algorithm 3) with each processing element placed in a separate VFI. In each of its iterations, the algorithm sorts the voltage-frequency islands FI_{pe} comprising all processing elements (except for the interconnect) in increasing order based on their criticality (Line 7). Two islands are merged together, forming a single partition. This process continues until all processing elements are placed in a single partition; thus, two voltage-frequency islands remain considering the interconnect that is in a separate island. As such, the algorithm evaluates all possible granularities of partitions.

Intuitively, islands having the lowest criticality values, as given by the order of islands based on criticality, shall be merged in each iteration. This may allow processing elements with higher criticality values, which are more likely to limit the throughput of a bound application graph, to remain in unshared voltage-frequency islands, resulting in high timing yield. However, merging two adjacent islands (adjacent in the ordered list of islands based on criticality) with higher criticality values may result in lower throughput reductions (e.g. when these islands have equal or close criticality values). For this reason, the timing yield given the binding b_{sb} as a result of merging each pair of adjacent voltage-frequency islands is evaluated in each iteration. This is shown on Lines 8–20 in Algorithm 5. Each adjacent pair of islands fi_x and fi_y are merged on Lines 9–10, where fi_x and fi_y are grouped into fi_y, and fi_x is removed from the list of islands FI. Each merge is reverted on Lines 18–19, such that the next pair of islands can be merged and evaluated for timing yield. Once the timing yield for merging all adjacent pairs of voltage-frequency islands FI_{pe} is evaluated, the grouping providing the highest timing yield is chosen, and the corresponding islands are merged permanently in FI. This is performed on Lines 21–22, where fi_x^{tmp} and fi_y^{tmp} are the voltage-frequency islands, the grouping of which provides the highest timing yield (derived on Lines 14–15). The algorithm performs an overall number of $(M/2 \cdot (M-1)-1)$ evaluations of timing yield, M being the number of processing elements. Note that an exhaustive evaluation of all possible groupings of processing elements into voltage-frequency islands requires $(2M)!/((M+1)! \cdot n!)$ (given by the Catalan number) evaluations, resulting in prohibitively long computation times for a large number of processing elements. Algorithm 5 returns a set FI_v of pairs $\{FI, y_{max}\}$ of voltage-frequency island configurations with different number of partitions and associated timing yield values.

Algorithm 5: Heuristic VFI partitioning algorithm considering a single binding for all chips

```
Require: $ga; gp; b_{sb}; n \ \forall fi \in FI; (\mu_g^r, \sigma_g^r, \delta^r, \sigma_l^r) \ \forall r \in R$
 1: $FI_v \leftarrow \emptyset$
 2: Compute $c(gp, fi, n) \ \forall fi \in FI$
 3: while $|FI| > 2$ do
 4:    $y_{max} \leftarrow 0$
 5:    $FI_{pe} \leftarrow FI \setminus fi_{noc}$
 6:    Sort $FI_{pe}$ in increasing $ci(ga, gp, b_{sb}, fi)$
 7:    for all adjacent $fi_x, fi_y \in FI_{pe}$ do
 8:       $fi_y \leftarrow fi_y \cup fi_x$
 9:       $FI \leftarrow FI \setminus fi_x$
10:       Compute $c(gp, fi_y, n)$
11:       Derive $FC$
12:       if $\gamma(ga, gp, b_{sb}) \geq y_{max}$ then
13:          $fi_x^{tmp} \leftarrow fi_x$
14:          $fi_y^{tmp} \leftarrow fi_y$
15:          $y_{max} \leftarrow \gamma(ga, gp, b_{sb})$
16:       end if
17:       $fi_y \leftarrow fi_y \setminus fi_x$
18:       $FI \leftarrow FI \cup fi_x$
19:    end for
20:    $fi_y^{tmp} \leftarrow fi_y^{tmp} \cup fi_x^{tmp}$
21:    $FI \leftarrow FI \setminus fi_x^{tmp}$
22:    $FI_v \leftarrow FI_v \cup \{FI, y_{max}\}$
23: end while
24: return $FI_v$
```

4.2.2 Multiple bindings

This section presents a heuristic algorithm to partition processing elements in a platform graph into multiple voltage-frequency islands considering that multiple bindings of actors in a resource-aware application graph to the processing elements in a platform graph are present. Compared to the case with only a single binding, the availability of multiple bindings can improve the timing yield, as demonstrated in the following example. Consider the example platform graph shown in Figure 4.1. It consists of three voltage-frequency islands, namely fi_1, fi_2 and fi_3. The interconnect is placed separately in voltage-frequency island fi_3. There are three processing elements pe_1, pe_2 and pe_3, which are partitioned into two voltage-frequency islands, such that processing element pe_1 is in one island and processing elements pe_2 and pe_3 are in another. Multiple clock-frequency levels are provided to each of the voltage-frequency islands. As processing elements pe_2 and pe_3 are in a single voltage-frequency island, the clock frequency of the island is decided by the slowest processing element belonging to the island. Let us assume that a resource-aware application graph comprising three actors a_1, a_2 and a_3 is mapped to the platform. Actor a_1 has the highest computational demands, and its execution time has the largest impact on the application throughput. Depending on the clock frequencies of islands fi_1 and fi_2, given by the chip frequency vector $fc \in FC$, different bindings of the actors to the processing elements may provide high throughput. In manufactured chips where fi_1 is operated at a higher clock frequency than fi_2, allocating actor a_1 to processing element pe_1, a_2 to pe_2 and a_3 to pe_3 provides higher throughput, as actor a_1 is given higher computational power (Table 4.1). In contrast, in manufactured chips where fi_2 is operated at a higher clock frequency than fi_1 (due to pe_2 and pe_3 both being faster than pe_1), allocating a_1 to pe_2, a_2 to pe_1 and a_3 to pe_3 provides higher throughput. In this example, we do not consider the latency of communication across the interconnect, and assume that it is infinitely fast. This demonstrates how multiple bindings can improve the timing yield, given a certain partitioning of processing elements into voltage-frequency islands. We proceed by introducing the partitioning algorithm considering multiple bindings.

Algorithm 6 specifies the heuristic algorithm to partition processing elements in a platform graph into multiple voltage-frequency islands, given a set of bindings of the actors in a resource-aware application graph to the processing elements in the platform graph. The algorithm requires as input a resource-aware application graph ga, a platform graph gp, a set B_{mb} of bindings, a specified number n of clock-frequency levels for each voltage-frequency island, and $\mu_g^r, \sigma_g^r, \delta^r, \sigma_l^r$ values for the random variables describing the maximum supported frequency of each hardware resources $r \in R$. The set B_{mb} of bindings are derived using the variation-aware heuristic algorithm implementing the multiple-bindings mapping approach, given in Algorithm 4, with each processing element placed in a separate island. Algorithm 6 is similar to Algorithm 5, which considers only a single binding for all chips. In each of its iterations, two islands are grouped forming

Algorithm 6: Heuristic VFI partitioning algorithm considering multiple bindings for all chips

```
**Require:** $ga; gp; B_{mb}; n\ \forall fi \in FI; (\mu_g^r, \sigma_g^r, \delta^r, \sigma_l^r)\ \forall r \in R$
 1: $b_p \in B_{mb}$
 2: $FI_v \leftarrow \emptyset$
 3: Compute $c(gp, fi, n)\ \forall fi \in FI$
 4: while $|FI| > 2$ do
 5:   $y_{max} \leftarrow 0$
 6:   $y_{tmp} \leftarrow 0$
 7:   $FI_{pe} \leftarrow FI \setminus fi_{noc}$
 8:   Sort $FI_{pe}$ in increasing $ci(ga, gp, b_p, fi)$
 9:   for all adjacent $fi_x, fi_y \in FI_{pe}$ do
10:     $fi_y \leftarrow fi_y \cup fi_x$
11:     $FI \leftarrow FI \setminus fi_x$
12:     Compute $c(gp, fi_y, n)$
13:     Derive $FC$
14:     for all $fc \in FC$ do
15:       for all $b \in B_{mb}$ do
16:         if $\tau(gb) \geq t_{req}$ then
17:           $y_{tmp} = y_{tmp} + pc(gp, fc)$
18:           break
19:         end if
20:       end for
21:     end for
22:     if $y_{tmp} \geq y_{max}$ then
23:       $fi_x^{tmp} \leftarrow fi_x$
24:       $fi_y^{tmp} \leftarrow fi_y$
25:       $y_{max} \leftarrow y_{tmp}$
26:     end if
27:     $fi_y \leftarrow fi_y \setminus fi_x$
28:     $FI \leftarrow FI \cup fi_x$
29:   end for
30:   $fi_y^{tmp} \leftarrow fi_y^{tmp} \cup fi_x^{tmp}$
31:   $FI \leftarrow FI \setminus fi_x^{tmp}$
32:   $FI_v \leftarrow FI_v \cup \{FI, y_{max}\}$
33: end while
34: return $FI_v$
```

a single partition (Lines 5–32). Note that an arbitrary binding $b_p \in B_{mb}$ (Line 1) is used for sorting the voltage frequency islands on criticality (Line 8). Similar to Algorithm 5, all adjacent pairs of voltage-frequency islands given by the ordered list of criticality are merged and are evaluated for timing yield (Lines 9–28), such that the grouping that provides the highest timing yield is found. In contrast to the algorithm considering only a single binding, the timing yield is computed based on the set B_{mb} of bindings (Lines 14–21). The algorithm evaluates all granularities of partitions, until all processing elements are placed in a single voltage-frequency island. It returns a set FI_v of pairs $\{FI, y_{max}\}$ of voltage-frequency island configurations with different number of partitions and associated timing yield values.

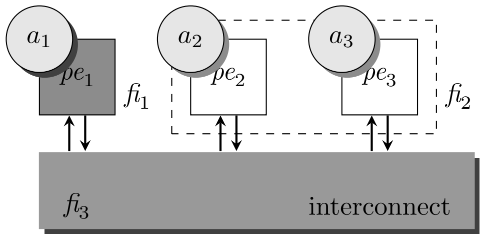


Figure 4.1: Example platform graph comprising three voltage-frequency islands. Depending on the clock frequencies of islands, one or another binding of the three actors belonging to a resource-aware application graph to the processing elements, which are placed in the islands, is required to achieve high throughput.

Table 4.1: Different mappings of the actors in an application to the processing elements in a multi-processor platform shown in Figure 4.1 based on the clock frequencies of voltage-frequency islands.

Actors	a_1	a_2	a_3
$fc(fi_1) > fc(fi_2)$	pe_1	pe_3	pe_2
$fc(fi_1) < fc(fi_2)$	pe_2	pe_1	pe_3

4.3 Experimental results

This section demonstrates, on both synthetic and realistic applications, how the proposed framework can be used by system designers to make trade-offs between the number of voltage-frequency islands (area cost) and timing yield. We illustrate that the proposed VFI partitioning algorithm effectively groups resources into islands for maximized timing yield. Additionally, the impact of the number of clock-frequency levels provided to voltage-frequency islands on timing yield is analyzed. We proceed to describing our experimental setup.

4.3.1 Experimental setup

The experimental setup in this chapter is similar to the one presented in Section 3.5.1. The same H.263 decoder, H.263 encoder, MP3 playback, Modem and MP3 decoder (except the Sample Rate Converter) applications are used. These applications are mapped to a multi-processor platform consisting of three processing elements, as most of the applications have a parallelism of three processing elements. Five clock-frequency levels are provided to each voltage-frequency island. Detailed information on the applications and the platform can be found in Section 3.5.1 and in Appendix B. In addition to the set of real applications described above, a single synthetic application is included for experimentation in this chapter. The reason we include a synthetic application is that it has a higher level of parallelism, compared to the real applications, and is mapped to a multi-processor platform comprised of a larger number of processing elements. This allows us to better demonstrate the important concepts of the proposed voltage-frequency island partitioning methodology. The SDF For Free (SDF^3) tool was used to generate the synthetic SDF graph [72, 81]. An overview of the synthetic application, showing the number of actors and cycles, as well as the available parallelism in terms of the minimum number of processing elements that can fully exploit it, is shown in Table 4.2. The topology of the application, together with the execution times of the actors and the size of data tokens sent across dependency edges, can be found in Appendix B. Note that mapping an application to a platform consisting of a higher than the minimum number of processing elements needed to exploit the parallelism of the application will result in larger slacks in some of the processing elements. Grouping these processing elements in a single voltage-frequency island may have a low impact on timing yield, resulting in unfair trade-offs between the number of voltage-frequency islands and timing yield. As Table 4.2 shows, the synthetic application has a parallelism that can be fully exploited by a minimum number of seven processing elements. This application is mapped to a multi-processor platform consisting of seven homogeneous processing elements. The platform-graph that describes the multi-processor platform is shown in Figure 4.2. Each processing element and the interconnect are placed in separate voltage-frequency islands; as such there are eight islands in total. Five clock-frequency levels are provided to each of the eight islands. The parameters of the interconnect and the random variables describing the maximum supported frequency of hardware resources are the same as those presented for the platform used for the real applications described above. These parameters are given in Tables 3.3 and 3.4.

Table 4.2: Overview of the synthetic application

SDF graph	Nr. actors	Nr. cycles	Parallelism (Nr. PEs)
Synthetic cyclic	17	3	7

Note that single and multiple bindings must be provided to the voltage-frequency partitioning algorithms given by Algorithms 5 and 6, respectively.

These bindings are determined using the variation-aware heuristic mapping algorithms illustrated in Algorithms 3 and 4.

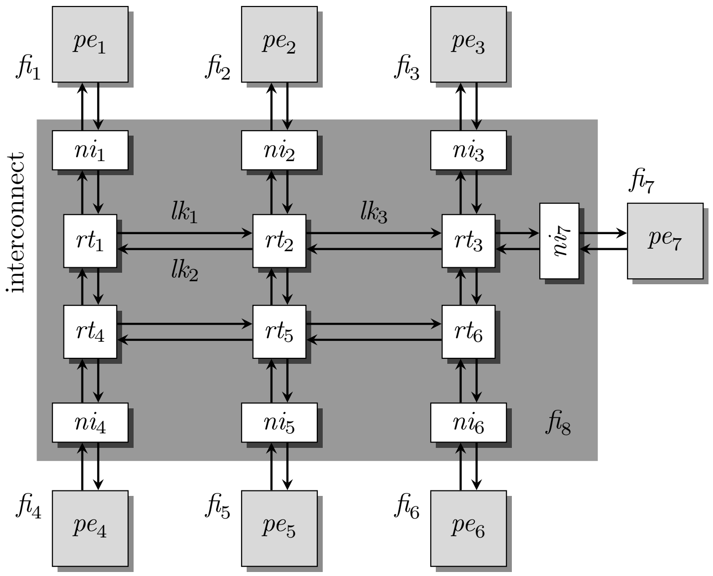


Figure 4.2: The multi-processor platform used for the synthetic application. It consists of seven processing elements connected to each other by an interconnect. The processing elements and the interconnect are placed in separate voltage-frequency islands.

4.3.2 Evaluation results

Below, we demonstrate the trade-offs between the number of voltage-frequency islands and timing yield for the set of real and the synthetic applications presented above. The trade-offs are given for both single and multiple bindings of the actors in a resource-aware application graph to the processing elements in a platform graph, which is a single and multiple bindings for the manufactured chips. In addition to this, the effectiveness of the proposed variation-aware voltage-frequency island partitioning algorithm is evaluated based on a comparison with a baseline deterministic partitioning algorithm. We also analyze the impact of changing the number of clock-frequency levels provided to each voltage-frequency island on the timing yield for each application.

Single binding

In this section, the results are obtained considering a single binding of the actors in an application to the processing elements in a multi-processor platform. The throughput requirements of the applications are determined arbitrarily. As shown in Algorithm 5, the proposed heuristic VFI partitioning algorithm returns a set FI_v of pairs $\{FI, y_{max}\}$ of voltage-frequency island configurations with different number of partitions and associated timing yield values. To better illustrate the results in this chapter, we show not only the single timing yield value y_{max} for each voltage-frequency island configuration *FI*, but the Cumulative Distribution Function (CDF) of the throughput of an application. This allows us to determine the timing yield for any throughput requirement. Consider the CDF of the inverse of the throughput of the synthetic application illustrated in Figure 4.3. The distributions are presented for four different system implementations, namely VFI-8, VFI-5, VFI-3 and VFI-2. VFI-8 is an eight voltage-frequency domain architecture, where each processing element and the interconnect are in separate voltage-frequency islands. VFI-5 and VFI-3 are architectures, where the processing elements are partitioned into four and two voltage-frequency islands, respectively. The partitioning is decided by the proposed heuristic VFI partitioning algorithm given in Algorithm 5. For the given throughput requirement t_{req}, the inverse of which is shown by the vertical dashed line, the grouping of the processing elements and the interconnect into islands for VFI-5 and VFI-2 is shown in Table 4.3. Finally, with VFI-2, all processing elements are placed in a single voltage-frequency island (together with the interconnect, there are two islands). For any throughput value t_0 within the throughput spread t, the CDF of $1/t_0$ represents the probability that the inverse of the throughput is less than or equal to $1/t_0$ (i.e. $1/t \leq 1/t_0$), and thus the throughput is greater than or equal to t_0 (i.e. $t \geq t_0$). Therefore, for any throughput requirement t_0, the CDF of the inverse of throughput determines the timing yield.

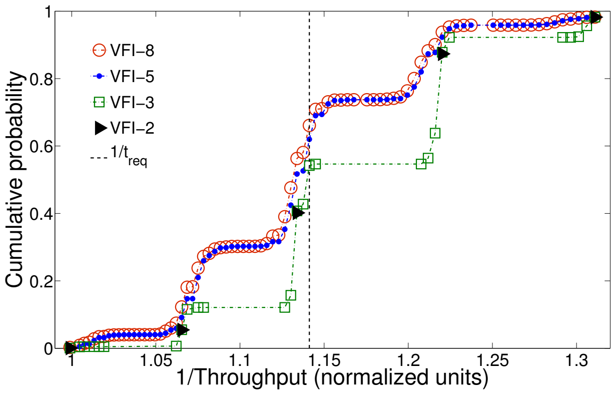


Figure 4.3: Throughput CDF for VFI-8, VFI-5, VFI-3 and VFI-2 architectures for the synthetic application. The results presume a single binding of the application to a NoC-based multi-processor platform consisting of seven processing elements.

Figure 4.3 shows that the throughput distribution for VFI-5 closely tracks the one for VFI-8. This is because the processing elements pe_6, pe_7, pe_3 and pe_2

have low criticality values, and grouping them in a single voltage-frequency island results in low throughput reductions. For the given throughput requirement, VFI-8 provides a 66% timing yield, while a 62% timing yield is provided by VFI-5. In contrast, the VFI-3 architecture results in considerable reductions in throughput, providing a 12% lower timing yield compared to VFI-8 for the given throughput requirement. Note that for a different throughput requirement, the reduction in timing yield can be much higher. For example, for a requirement resulting in an 88% timing yield with VFI-8, the reduction with VFI-3 is 32%. Note that the partitioning of the processing elements into VFIs is based on the requirement. Therefore, we can argue that a different grouping for VFI-3 may be derived given a different requirement. The VFI-2 architecture results in a sub-sampled throughput distribution, and can lead to considerable reductions in the timing yield. For the given throughput requirement, VFI-2 gives only a 40% timing yield compared to the 66% with VFI-8. This information can be used by system designers to make informed trade-offs between the granularity of VFIs and thus area cost and timing yield.

Table 4.3: Grouping of processing elements and the interconnect into VFIs, as a result of both variation-aware (VFI-5, VFI-3) and deterministic (DVFI-5, DVFI-3) partitioning algorithms considering a single binding.

Architecture	grouping
VFI-5	$\{pe_6, pe_7, pe_3, pe_2\}$ $\{pe_5\}$ $\{pe_4\}$ $\{pe_1\}$ $\{noc\}$
VFI-3	$\{pe_6, pe_7, pe_3, pe_2\}$ $\{pe_5, pe_4, pe_1\}$ $\{noc\}$
DVFI-5	$\{pe_1, pe_2\}$ $\{pe_3, pe_4\}$ $\{pe_5, pe_6\}$ $\{pe_7\}$ $\{noc\}$
DVFI-3	$\{pe_1, pe_2, pe_3, pe_4\}$ $\{pe_5, pe_6, pe_7\}$ $\{noc\}$

To show the effectiveness of the proposed variation-aware VFI partitioning algorithm, we compare it to a baseline deterministic partitioning approach. From the perspective of clock-tree routing, it is desirable to have a low number of processing elements in each voltage-frequency island. This can reduce the complexity of local clock wiring in the partitions. Therefore, for a specified number of voltage-frequency islands, the deterministic partitioning tries to equally distribute the processing elements into voltage-frequency islands, thus reducing the maximum number of processing elements in a single island. Figure 4.4 shows the CDF of the inverse of the throughput for five and three domain architectures based on groupings of processing elements to islands derived by both variation-aware and deterministic partitioning algorithms. The grouping of the processing elements into islands for both variation-aware and deterministic partitioning algorithms is given in Table 4.3. As the table shows, with the deterministic partitioning (DVFI-5 and DVFI-3), the processing elements are equally distributed in the islands (except one island, as there is an odd number of processing elements). The choice of processing elements in the voltage-frequency islands is based on the adjacent index values of processing elements. For the given throughput requirement, the deterministic five-domain partitioning (DVFI-5) results in an 11% lower timing yield than the variation-aware partitioning (VFI-5). Similarly, a 14% lower yield is achieved by DVFI-3, compared to VFI-3. Note that for a throughput requirement resulting in a 64% timing yield with DVFI-3, the timing yield is reduced with VFI-3. However, it can be argued that with this throughput requirement specified, the partitioning of the processing elements into voltage-frequency islands with VFI-3 may be different, leading to improved timing yield (the current grouping is derived based on the throughput requirement shown in Figure 4.4). These results show the importance of variation-aware voltage-frequency island partitioning for improved timing yield.

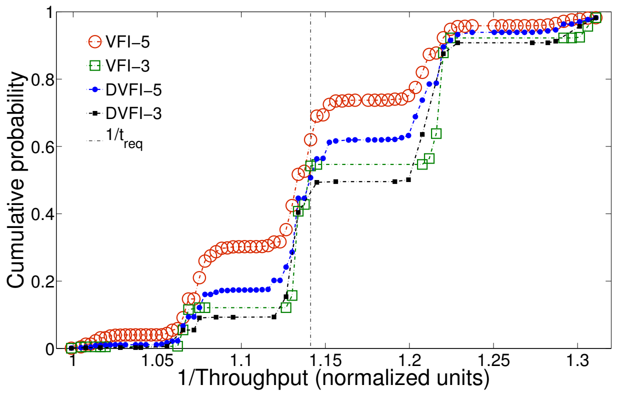
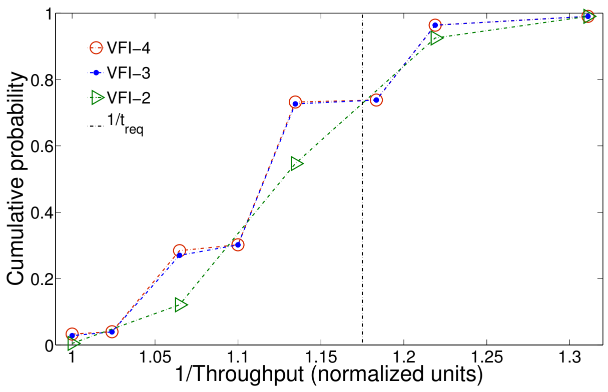
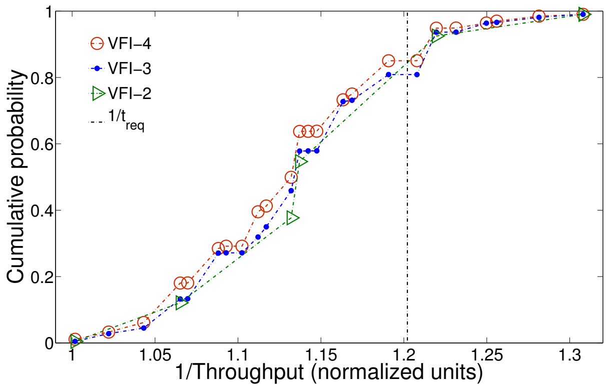


Figure 4.4: Throughput CDF for five and three VFI architectures, based on both variation-aware (VFI-4, VFI-2) and deterministic partitions (DVFI-4, DVFI-2).

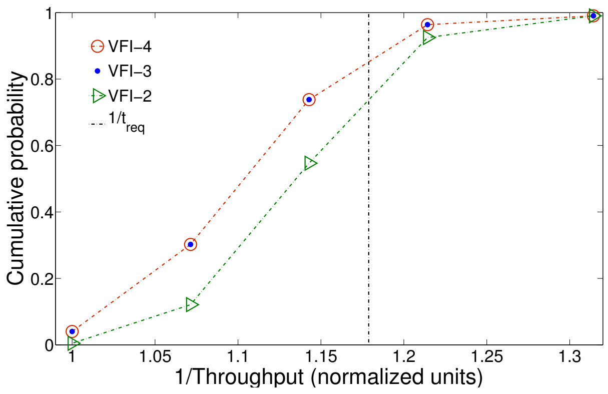
Figure 4.5 illustrates the CDFs of the inverse of the throughput for the H.263 decoder, H.263 encoder and MP3 playback applications. For all applications, architectures with four (VFI-4), three (VFI-3) and two (VFI-2) voltage-frequency islands, are considered. VFI-4 is an architecture where each processing element is in a separate voltage-frequency island. On the other hand, all processing elements are placed in a single partition with VFI-2, and thus together with the interconnect, there are two VFIs. The grouping of the processing elements into islands for VFI-3 is performed by the proposed variation-aware heuristic partitioning algorithm. Figure 4.5 shows that for all three applications, the reduction in the timing yield when moving from VFI-4 to VFI-2 is considerable. More precisely, there is an 18%, 29% and 19% reduction for the H.263 decoder, H.263 encoder and MP3 playback applications, respectively. On the other hand, VFI-3 may or may not reduce the timing yield. For H.263 decoder, there is a negligible decrease in the timing yield, compared to VFI-4. For MP3 playback, VFI-4 and VFI-3 result in identical throughput distributions. This is due to two processing elements having



(a) H.263 decoder

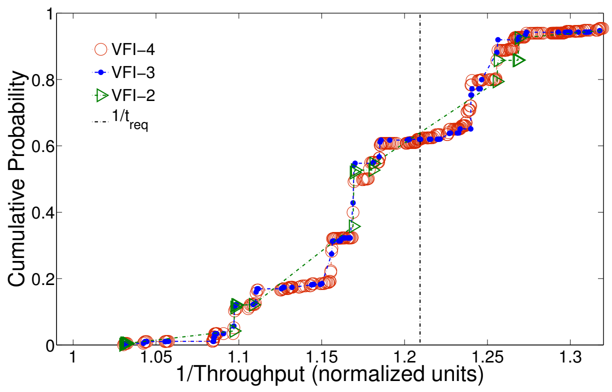


(b) H.263 encoder

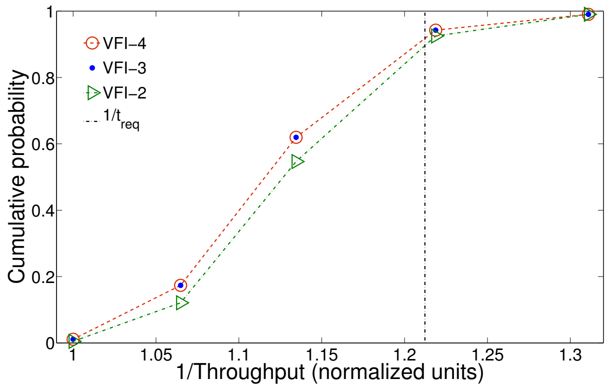


(c) MP3 playback

Figure 4.5: Throughput CDF for VFI-4, VFI-3 and VFI-2 architectures for the H.263 decoder, H.263 encoder and MP3 playback applications. The results presume a single binding of the application to a NoC-based multi-processor platform consisting of three processing elements.



(a) Modem



(b) MP3 decoder

Figure 4.6: Throughput CDF for VFI-4, VFI-3 and VFI-2 architectures for the Modem and MP3 decoder applications. The results presume a single binding of the application to a NoC-based multi-processor platform consisting of three processing elements.

equal criticality values. Grouping them in a single voltage-frequency island does not result in throughput reductions. In contrast, the timing yield is lower by 4% than the 85% provided by VFI-4 fro the H.263 encoder application. Similar results are illustrated for the Modem and MP3 decoder applications in Figure 4.6. For both applications, VFI-4 and VFI-3 both provide the same timing yield, while VFI-2 results in a 7% lower timing yield compared to VFI-4 for both Modem and MP2 decoder. Note that for the Modem application, there are much more different throughput points in the distribution, compared to the MP3 decoder application. This is dependent on the topology and the mapping of the application. There is a large number of chip-frequency vectors (sets of clock frequencies for the voltage-frequency islands) (Definition 5). For the Modem application, many of these chip-frequency vectors provide different throughput values, while identical throughput values are obtained for most of them for the MP3 decoder application due to its topology and mapping (e.g. only the frequency change in one processing element affects the throughput).

The discussion presented in this section demonstrates on both a synthetic SDF graph and SDF models of real applications how the proposed framework can be used by system designers to make trade-offs between the number of voltage-frequency islands, and thus cost (in terms of area and power consumption), and timing yield. Now, we proceed to demonstrating the results of VFI partitioning given multiple bindings of the actors in a resource-aware application graph to the processing elements in a platform graph.

Multiple bindings

In this section the results are presented considering multiple bindings of the actors in an application to the processing elements in a multi-processor platform. The set of bindings is derived by the variation-aware heuristic algorithm implementing the multiple-bindings mapping approach given in Algorithm 4. Figure 4.7 shows the CDF of the inverse of the throughput of the synthetic application for VFI-8, VFI-5, VFI-3 and VFI-2 architectures. Initially, 125 non-identical bindings are selected by the algorithm, out of which only 38 bindings are dominant, and thus they provide the highest throughput for all chip-frequency vectors. However, from the set of 38 bindings, only eight bindings are manually chosen, as the rest are dominant only for a few chip-frequency vectors. The grouping of the processing elements and the interconnect into voltage-frequency islands for the five and three domain architectures is given in Table 4.4. As can be seen, a different partitioning for both VFI-5 and VFI-3 is derived compared to the partitioning considering a single binding, given in Table 4.3. In Table 4.4, the voltage-frequency islands are given in increasing order of criticality (from left to right). Note that with VFI-5 islands consisting of processing elements pe_7 and pe_6 are merged into a single island, even though the island with pe_1 has a lower criticality value. This happens as merging pe_7 and pe_6 into a single partition provides higher timing yield (the islands with these processing elements have close criticality values). This illustrates the benefits of evaluating the timing yield, as a result of merging each pair of adjacent voltage-frequency islands in each iteration of the proposed heuristic partitioning algorithms given in Algorithms 5 and Algorithm 6.

Figure 4.7 shows significant improvements in the timing yield, compared to the case where only a single binding of the actors in the resource-aware synthetic application graph to the processing elements in the platform graph is present. More specifically, the eight domain architecture (VFI-8) considering a single binding provides a 66% timing yield, as given by Figure 4.3. The same eight domain implementation considering multiple (eight) bindings provides an 84% timing yield, resulting in an 18% improvement. Similar observations are made for the VFI-5 and VFI-3 architectures. With VFI-5, the timing yield is increased from 62% (single binding) to 77% (multiple bindings) with an improvement of 15%. An increase in the timing yield of 10% is observed in the case of the VFI-3 implementation, where a single and multiple bindings result in a timing yield of 54% and 64%, respectively. From these results, we make an observation that the benefits of having multiple bindings (in terms of increased timing yield) are lower for a lower number of voltage-frequency islands. Note that the throughput distributions with VFI-2 in the case of both single and multiple bindings are identical. With the VFI-2 architecture all processing elements are operated at the clock frequency the slowest processing element in the platform can support. Therefore, multiple bindings do not result in throughput improvements in this case. As suggested by Figure 4.3, which presumes a single binding, the timing yield decreases by 4% and 8% when moving from VFI-8 to VFI-5 and from VFI-5 to VFI-3, respectively. On the other hand, in Figure 4.7, higher reductions of 7% and 13% are seen between VFI-8, VFI-5 and VFI-5, VFI-3, respectively. We thus make another observation that the loss in timing yield when moving to a smaller number of voltage-frequency islands is larger with multiple bindings than with a single binding.

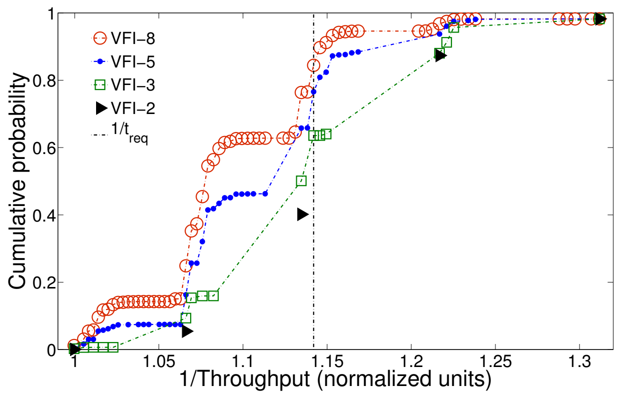


Figure 4.7: Throughput CDF for VFI-8, VFI-5, VFI-3 and VFI-2 architectures for the synthetic application. The results presume multiple bindings (eight) of the application to a NoC-based multi-processor platform consisting of seven processing elements.

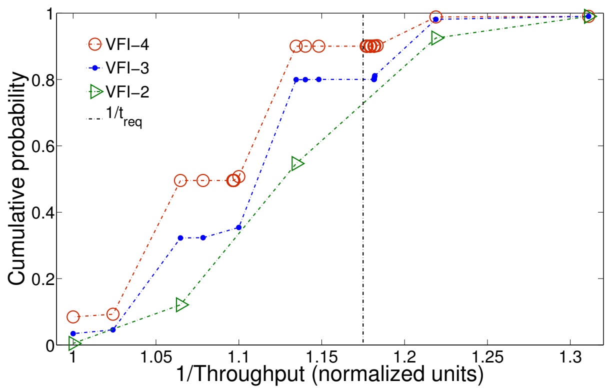
Table 4.4: Grouping of processing elements and the interconnect into VFIs, as a result of the variation-aware partitioning algorithm considering multiple bindings.

Architecture	grouping
VFI-5	$\{pe_5, pe_3, pe_2\}$ $\{pe_1\}$ $\{pe_7, pe_6\}$ $\{pe_4\}$ $\{noc\}$
VFI-3	$\{pe_1, pe_5, pe_3, pe_2\}$ $\{pe_4, pe_7, pe_6\}$ $\{noc\}$

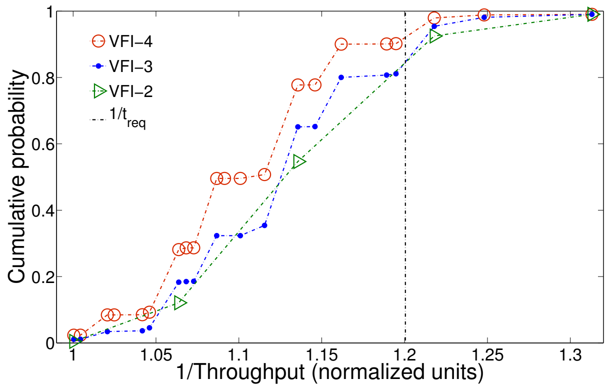
Figure 4.8 illustrates the CDFs of the inverse of the throughput of the H.263

decoder, H.263 encoder and MP3 playback applications for VFI-4, VFI-3 and VFI-2 architectures. The number of bindings selected for each application can be found in Table 3.7. Figure 4.8 shows significant improvements in timing yield with both VFI-4 and VFI-3, compared to the case where only a single binding is present. For the H.263 decoder, the improvements are 7% and 17% with VFI-3 and VFI-4, respectively. Similarly, a 7% and a 16% higher timing yield is provided with the VFI-3 and VFI-4 architectures for the MP3 playback application. In the case of the H.263 encoder application, the timing yield is the same for both single and multiple bindings with VFI-3, while the VFI-4 architecture considering multiple bindings increases the timing yield by 5%. Similar to the case with the synthetic application, we observe that the benefits of having multiple bindings (in terms of increased timing yield) are lower for a lower number of voltage-frequency islands.

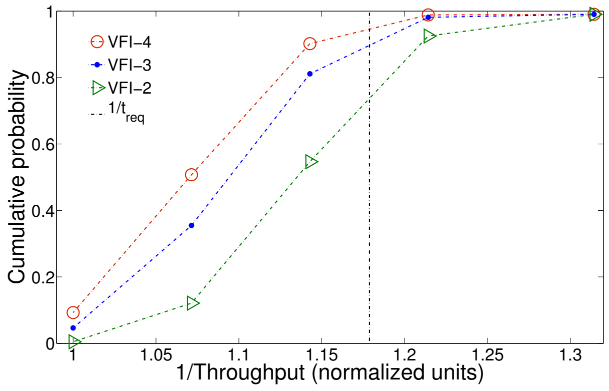
Similar results are seen for the Modem and MP3 decoder applications in Figure 4.9. For MP3 decoder, multiple bindings do not benefit in increasing the timing yield considering a three voltage-frequency island architecture (VFI-3). With both single and multiple bindings, the timing yield is 62%. In contrast, an increase of 15% with VFI-4, where each processing element is in a separate island, is provided by multiple bindings. For the Modem application, there is a considerable gain in timing yield with both VFI-4 and VFI-3 when moving to multiple bindings. With a four domain architecture, the timing yield is improved by 23% (from 62% to 85%), compared to the case with a single binding. Note that the timing yield for the Modem application reported by the heuristic algorithm implementing the multiple-bindings mapping approach (MBH) in Figure 3.8 is 70%. This result is given for the same four domain architecture, and the same set of bindings are also used in the experiment, the results of which are shown in Figure 4.9. The reason of this difference is that, with the MBH algorithm, some bindings that provide high throughput for certain chip-frequency vectors, but are not found for these chip-frequency vectors (reducing the timing yield), are in fact located for other chip frequency vectors. However, with the VFI partitioning algorithm given in Algorithm 6, the complete set of bindings are evaluated for each chip-frequency vector, providing a higher 85% timing yield. As such, although the reported timing yield with MBH is 70%, in reality, the selected set of bindings provide 85% timing yield for the Modem application. Similar to the improvement with VFI-4, multiple bindings with the VFI-3 architecture for the Modem application provide an 18% increase in the timing yield. Note that the reduction in the timing yield when moving from VFI-4 to VFI-3 is 5% (Figure 4.9). The results and the discussion presented in this section illustrate that multiple bindings result in considerable improvements in the timing yield, compared to the case with a single binding. We also showed that the reduction in the timing yield when decreasing the number of VFI partitions is higher with multiple bindings. This information can be used by system-designer to make informed trade-offs between single and multiple bindings, as well as the number of VFI partitions and timing yield. We proceed to analyzing the impact of changing the number of clock-frequency levels provided to islands on timing yield.



(a) H.263 decoder

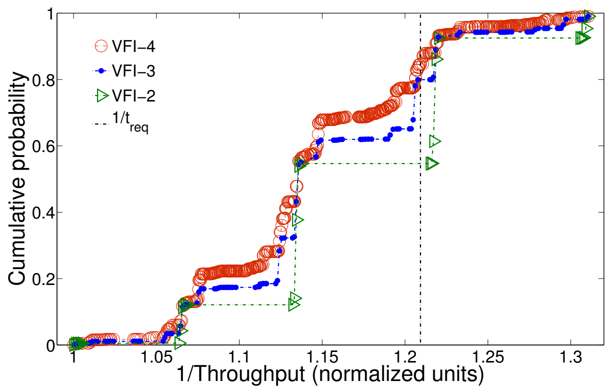


(b) H.263 encoder

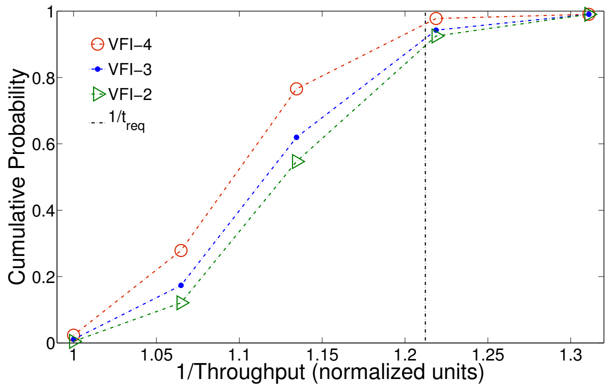


(c) MP3 playback

Figure 4.8: Throughput CDF for VFI-4, VFI-3 and VFI-2 architectures for the H.263 decoder, H.263 encoder and MP3 playback applications. The results presume multiple binding of the application to a NoC-based multi-processor platform consisting of three processing elements.



(a) Modem



(b) MP3 decoder

Figure 4.9: Throughput CDF for VFI-4, VFI-3 and VFI-2 architectures for the Modem and MP3 decoder applications. The results presume multiple binding of the application to a NoC-based multi-processor platform consisting of three processing elements.

Number of clock-frequency levels

The results in this section presume a single binding of the actors in a resource-aware application graph to the processing elements in a platform graph. Figure 4.10 shows the CDF of the throughput of the synthetic application for a five voltage-frequency island architecture. The graphs are presented for eight, five, three, two and a single clock-frequency levels provided to the voltage-frequency islands in the platform graph. The figure shows that the number of clock-frequency levels has a large impact on the throughput distribution and hence on the timing yield. The higher the number of levels is the finer the distributions (i.e. more points are present in them). It can be seen that a larger number of clock-frequency levels does not necessarily result in higher timing yield. For example, two clock-frequency levels provide a much higher timing yield of 52%, compared to the 17% given with three levels. Similarly, five levels provide a 3% higher timing yield, compared to the case where the number of levels is eight. A larger number of clock-frequency levels can reduce the timing yield, as the timing yield depends not only on the number, but also on the specific clock frequencies. In this work, equidistant clock-frequency levels are selected (in the same range), as detailed in Section 2.3. Therefore, the set of a certain number of clock-frequency levels (e.g. $n = 2$, and the set of levels is $c(gp, fi, 2)$) is not always a subset of a larger number of clock-frequency levels (i.e. $c(gp, fi, 2) \not\subset c(gp, fi, 3)$). For this reason, the timing yield may be reduced with an increase in the number of levels. Note that the timing yield is never reduced when going from two to eight clock-frequency levels, as shown in Figure 4.10. This is because, with the equidistant clock-frequency selection policy, the set of two clock-frequency levels is a subset of the set of eight levels (i.e. $c(gp, fi, 2) \subset c(gp, fi, 8)$). Note that for a different throughput requirement, two and five clock-frequency levels can result in up to 44% and 20% reduction in the timing yield, compared to the case with eight levels. Depending on the requirement, we observe that it is more likely that the timing yield is higher for a larger number of clock-frequency levels. Finally, a single clock-frequency level results in a single low throughput point and a 0% timing yield.

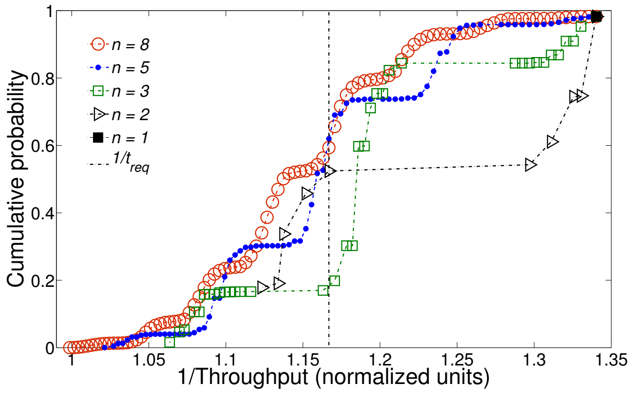
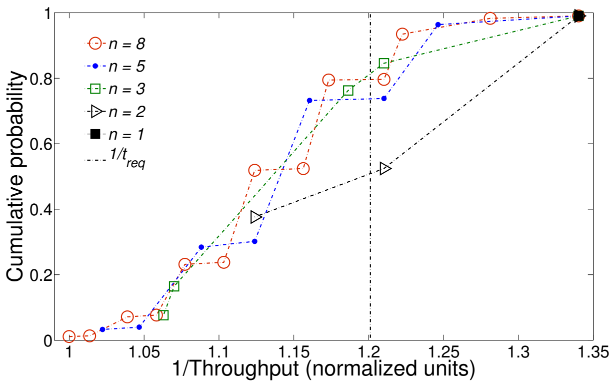
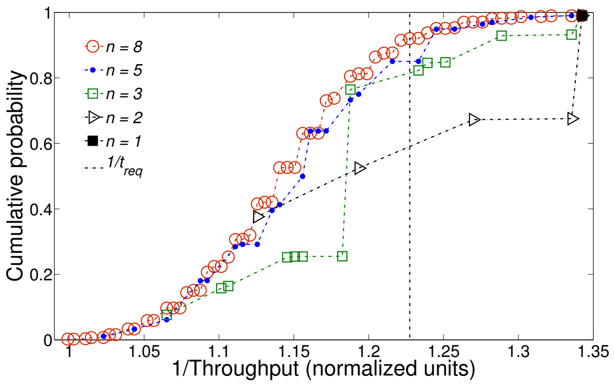


Figure 4.10: Throughput CDF for a VFI-5 architecture for the synthetic application. The results presume a single binding of the application to a NoC-based multi-processor platform consisting of seven processing elements. The graphs are presented for eight, five, three, two and a single clock-frequency levels provided to voltage-frequency islands in the platform graph.

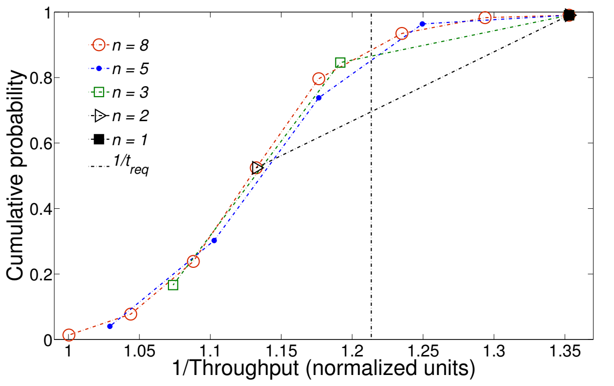
Figure 4.11 shows similar results for the H.263 decoder, H.263 encoder and MP3 playback applications. The graphs are shown for a single binding and for eight, five, three, two and a single clock-frequency levels provided to the voltage-frequency islands in the platform graph. For the H.263 decoder application, eight, five, three, two and a single clock-frequency levels result in 79%, 73%, 76%, 38% and 0% timing yield, respectively. As in the case of the synthetic application, here



(a) H.263 decoder



(b) H.263 encoder



(c) MP3 playback

Figure 4.11: Throughput CDF for a VFI-4 architecture for the H.263 decoder, H.263 encoder and MP3 playback applications. The results presume a single binding of the application to a NoC-based multi-processor platform. The graphs are presented for eight, five, three, two and a single clock-frequency levels provided to voltage-frequency islands in the platform graph.

too, the timing yield is reduced when moving from a smaller to a larger number number of clock-frequency levels (e.g. from three to five levels). With the H.263 encoder, 92%, 85%, 76% and 52% timing yield values are provided with eight, five, three and two clock-frequency levels, respectively. This illustrates the possible benefits in terms of improved timing yield when increasing the number of clock-frequency levels. Similar results are shown for the MP3 playback application. Note that for this application the timing yield provided by three clock frequency levels is higher by 5% than the timing yield of eight clock-frequency levels.

The discussion presented in this section demonstrates on both a synthetic SDF graph and SDF models of real applications how the proposed framework can be used by system designers to make informed trade-offs between single and multiple bindings, the number of voltage-frequency islands, as well as the number of clock-frequency levels provided to islands, (i.e. cost in terms of area and power consumption) and timing yield.

4.4 Summary

Reducing the number of voltage-frequency islands in a multi-processor platform results in a smaller number of clock-generation units, which provide a set of clock-frequency levels to the islands. This in turn reduces the cost in terms of area and power consumption. On the other hand, with a reduced number of voltage-frequency levels, multiple processing elements share a single island, the clock-frequency of which is decided based on the slowest processing element in the island. If one of the processing elements is slow due to within-die process variation, the whole island is operated at a reduced clock frequency. This may lower the throughput of the application mapped to the platform in manufactured chips, negatively affecting the timing yield. Additionally, the area and power consumption costs of a clock-generation unit are decided based on its implementation, which in turn depends on the desired number of clock-frequency levels and the step size between them. Thus, a reduction in the number of levels may lead to a lower area cost. On the other hand, a smaller number of levels provide a worse matching of the actual maximum supported frequency, which can be any value within the process-induced frequency spread, of hardware resources. This in turn lowers the throughput in manufactured chips and as a consequence the timing yield. This chapter introduced a heuristic algorithm, for each case with a single and multiple bindings for the manufactured chips, to partition the processing elements into voltage-frequency islands with a minimal reduction in timing yield. A new metric, coined as island criticality, was proposed to guide the partitioning process. The metric quantifies the sensitivity of the throughput of an application mapped to a multi-processor platform to the clock frequency of a voltage-frequency island in the platform. An island with a high criticality value presumes a high reduction in the throughput when the clock frequency of the island is reduced. Thus, merging an island with a high criticality value with other islands may result in high reductions in the throughput in manufactured chips, as the clock frequency of the created island is lowered due to the presence of slower processing elements in the island. In addition to this, an outline of possible architectures of a clock-generation unit based on the desired number of clock-frequency levels was presented, showing that the finer the granularity of the clock-frequency levels, the higher the area cost can be.

The proposed partitioning algorithms were evaluated on a synthetic and a set of SDF graphs modeling real applications. As such, the chapter illustrates the trade-offs that can be performed by system designers between the number of voltage-frequency islands (i.e. area and power consumption costs) and timing yield. Given a single binding for the manufactured chips, we experimentally showed that the number of islands can be reduced with a negligible reduction in timing yield. More specifically, for the synthetic application, a 4% lower timing yield is provided with a five-island architecture, compared to an architecture with eight islands in a platform consisting of seven processing elements. Meanwhile, there is no reduction in the timing yield for a six-domain architecture. This is a positive observation, as it allows us to reduce the area and power consumption costs with a negligible or no impact on timing yield. The reduction in timing yield increases when the number of islands is further reduced. For example, 12% and 23% lower values of timing yield are reported for three and two VFI architectures, respectively. Similar results are reported for the SDF models of real applications for the same case of a single binding for the manufactured chips. For four out of five applications, going from a four to a three voltage-frequency island architecture negligibly lowers the timing yield in a platform with three processing elements. On the other hand, up to a 29% reduction is seen when placing all processing elements in a single island (i.e. two islands in total considering one for the interconnect).

For the case of multiple bindings for the manufactured chips, the following results are experimentally reported. An observation is made that the benefits of having multiple bindings (in terms of increased timing yield) reduce together with a reduction in the number of voltage-frequency islands. More specifically, for the synthetic application, the increase in timing yield due to having multiple bindings for eight, five and three island architectures are 18%, 15% and 10%, respectively. The reported results suggest that the loss in timing yield when moving to a smaller number of voltage-frequency islands is larger for multiple bindings than for a single binding. With the synthetic application example, reductions of 7% and 13% in timing yield when moving from eight to five and from five to three island architectures are obtained. These numbers are larger than the 4% and 8% values that presume a single binding. Similar results are reported for the SDF graphs of real applications.

The same chapter additionally illustrates the effectiveness of the proposed variation-aware heuristic partitioning algorithm by comparing it to a baseline deterministic partitioning approach. With this approach, the processing elements are equally distributed among voltage-frequency islands to reduce the complexity of local clock tree routing within partitions. We have observed that the proposed variation-aware heuristic partitioning algorithm provides 11% and 14% higher values of timing yield, compared to the deterministic partitioning.

Lastly, experimental results of timing yield for varying the number of clock-frequency levels provided to voltage-frequency islands are provided in the chapter. The results show that the number of clock-frequency levels has a large impact on timing yield. However, a larger number of clock-frequency levels does not necessarily result in higher timing yield. For the synthetic application example, a 35% lower timing yield is reported with three clock-frequency levels, compared to two clock-frequency levels. Similarly, five clock-frequency levels improve the result provided by eight levels by 3%. A larger number of clock-frequency levels can reduce the timing yield, as the timing yield depends not only on the number, but also on the specific clock frequencies. In this work, equidistant clock-frequency levels are selected within the same frequency range. Therefore, the set of a certain number of clock-frequency levels (e.g. $n = 2$, and the set of levels is $c(gp, fi, 2)$) is not always a subset of a larger number of clock-frequency levels (i.e. $c(gp, fi, 2) \not\subset c(gp, fi, 3)$). For this reason, the timing yield may be reduced with an increase in the number of levels. As such the timing yield is never reduced when going from two to eight clock-frequency levels, as the set of two clock-frequency levels is a subset of the set of eight levels with the equidistant clock-frequency selection policy. With the rest of the applications, the timing yield is mostly increased due to increasing the number of clock-frequency levels. Improvements of up to 41% when going from two to eight levels are reported. We thus observe that it is more likely that the timing yield is improved when increasing the number of clock-frequency levels.

5 Better than worst-case design

Reducing the design margins or guard-bands when implementing a circuit provides the benefit of decreased circuit area, resulting in a higher number of dies on a wafer. This in turn may provide a higher number of good dies that satisfy the throughput requirement imposed on the system. In this chapter, we demonstrate on case studies, how our framework is used to estimate the impact of guard-band reduction on the number of good dies on a wafer. The number of good dies is given by the product of the number of gross dies and timing yield. The timing yield corresponding to a guard-band reduction value is estimated by the methodology proposed in this thesis. We consider both cases where a single and multiple bindings of the actors in a resource-aware application graph to the processing elements in a platform graph are present. We additionally illustrate the impact of reducing the number of voltage-frequency islands in a platform graph, and thus cost in terms of area, on the number of good dies.

5.1 Number of good dies

Circuit guard-banding is typically done by using corner-files during the design and verification stages; these files describe the worst-case and best-case delay values of standard-cells, corresponding to slow and fast process corners, respectively. The change in circuit area when reducing these guard-bands (i.e. implementing the circuit with reduced WC and increased BC delay values) is assessed by Jeong *et al.* in [33]. They use open-source cores and an industrial embedded processor core with target clock frequencies ranging from 300 to 600 MHz; the cores are synthesized using 90, 65 and 45 nm technology model libraries. Based on measured data, the authors provide a linear regression model for circuit-area reduction versus guard-band reduction. Equation (5.1) shows the model, where v is the area reduction factor and u is the guard-band reduction in percent.

$$v = 1 - 0.0033 \cdot u \tag{5.1}$$

The number of gross dies on a wafer is given by Equation (5.2), where l is the radius of the wafer and V is the die area [33]; the second term in the equation accounts for wasted area around the edges of the circular wafer. Using Equation (5.1), the number of gross dies corresponding to a u% guard-band reduction can be computed.

$$N_{\text{gross}} = \pi \cdot \left(\frac{l^2}{V} - \frac{2l}{\sqrt{2V}} \right) \tag{5.2}$$

We are interested in the number of good dies that provide throughput at least equal to the minimum throughput requirement of an application. As the timing yield metric quantifies the percentage of manufactured chips that satisfy the throughput requirement of an application, the number of good dies is given by the product of timing yield and the number of gross dies (Equation (5.3)). Note that random defect yield can also be taken into account in the equation, as shown by Jeong *et al.* [33]. However, due to low error density values, the random defect yield is high, and has a negligible impact on the number of good dies [33]. Furthermore, guard-band reduction has almost no impact on random defect yield, as shown in [33]. For these reasons, we do not take random defect yield into consideration.

$$N_{\text{good}} = y \cdot N_{\text{gross}} \tag{5.3}$$

5.2 Variation characterization

Each guard-band value results in a particular circuit implementation with a certain area, after the design and verification stages. By performing statistical characterization (Monte Carlo simulations) on each circuit implementation, the PDFs of the maximum supported frequency of the hardware resources (processing elements, routers, network interfaces and links) in a platform graph is obtained. A statistical characterization flow is proposed by Miranda *et al.* in [51]. However, in this chapter, we do not perform statistical characterization to obtain the frequency distributions for hardware resource. Instead, we make the following intuitive assumptions. With the original guard-band (i.e. 0% guard-band reduction), we assume that manufactured hardware resource instances within the range of three standard deviations from mean ($\approx 99.7\%$) satisfy the target frequency f^r_{tg}. This corresponds to a combined normal distribution of maximum supported frequency, where $(\mu^r_{0\%} - 3\sigma^r_{0\%} \cdot (\mu^r_{0\%}/100)) = f^r_{tg}$, where $\sigma^r_{0\%}$ is the standard deviation in percents of mean frequency. Therefore, the mean frequency $\mu^r_{0\%}$ is computed by

Equation (5.4). For a resource with $f_{tg}^r = 300$ MHz target frequency, $\sigma_g^r = 4\%$, $\sigma_l^r = 3.3\%$, and thus $\sigma_{0\%}^r \approx 5.186\%$ of mean frequency, the combined distribution $f^r = N(\mu_{0\%}^r, \sigma_{0\%}^r)$ is shown in Figure 5.1. The mean frequency $\mu_{0\%}^r$ for this example is equal to ≈ 355 MHz. These numbers are based on the available data at 45nm technology [64].

$$\mu_{0\%}^r = \frac{f_{tg}^r}{1 - 3\sigma_{0\%}^r/100} \tag{5.4}$$

With no guard-band (i.e. 100% guard-band reduction), we assume that half of manufactured hardware resource instances satisfy the target frequency f_{tg} considering only global variation. This corresponds to a global normal distribution of maximum supported frequency with f_{tg}^r mean. Therefore, the mean of the combined distribution is given by $\mu_{100\%}^r = (f_{tg}^r - \delta^r \cdot (f_{tg}^r/100))$, where δ^r is the reduction in mean frequency (in percents) due to local variation (Equation (5.5)). For the example described above, where $f_{tg}^r = 300$ MHz, $\sigma_g^r = 4\%$, $\sigma_l^r = 3.3\%$, and $\sigma_{100\%}^r \approx 5.186\%$ of mean frequency, the distribution $f^r = N(\mu_{100\%}^r, \sigma_{100\%}^r)$ is shown in Figure 5.1.

$$\mu_{100\%}^r = f_{tg}^r \cdot (1 - \frac{\delta^r}{100}) \tag{5.5}$$

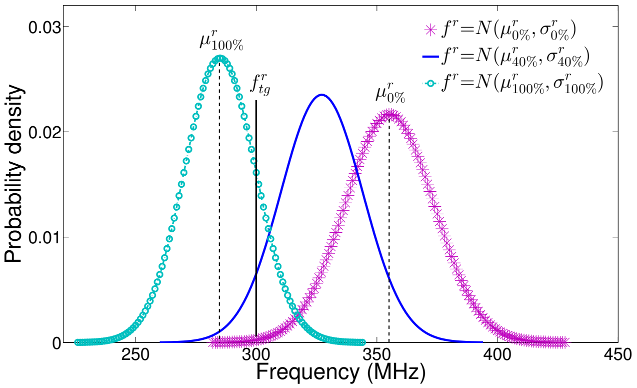


Figure 5.1: Combined f^r PDF of a hardware resource due to a 0%, 40%, 100% guard-band reduction. The target frequency f_{tg}^r is 300 MHz, $\sigma_g^r = 4\%$, $\sigma_l^r = 3.3\%$, and thus $\sigma_{u\%}^r \approx 5.186\%$ of mean frequency.

Given the values $\mu_{0\%}^r$ and $\mu_{100\%}^r$, a $u\%$ guard-band reduction results in a new combined normal distribution with a mean frequency $\mu_{u\%}^r$ given by Equation (5.6). Figure 5.1 shows the combined distribution $f^r = N(\mu_{40\%}^r, \sigma_{40\%}^r)$ for a 40% guard-band reduction. We assume that the standard deviation to mean ratio of the combined normal distribution for any $u\%$ guard-band reduction is constant. For the example described above this ratio is $\sigma^r_{u\%}/\mu^r_{u\%} \approx 5.186\%$. In reality, it may change due to local variation that depends on circuit implementation, which is different for each $u\%$ guard-band reduction (global variation does not depend on circuit implementation). However, only small differences in a close range of mean frequency are expected (e.g. the ratio difference for 300 MHz and 355 MHz mean frequency circuit implementations is much lower than for 300 MHz and 2 GHz circuit implementations). For a given a mean frequency $\mu^r_{u\%}$, the frequency μ^r_g corresponding to a guard-band reduction vale can be computed by Equation 5.7, where δ^r is the mean reduction in percents of μ^r_g.

$$\mu^r_{u\%} = \mu^r_{0\%} - u \cdot \frac{\mu^r_{0\%} - \mu^r_{100\%}}{100} \tag{5.6}$$

$$\mu^r_g = \frac{\mu^r_{u\%}}{(1 - \frac{\delta^r}{100})} \tag{5.7}$$

5.3 Experimental setup

The results in this section are presented for the synthetic and the set of real applications used in Sections 3.5.1 and 4.3.1. This set contains an H.263 decoder, an H.263 encoder, an MP3 playback, a Modem and an MP3 decoder. The set of real applications are mapped to a NoC-based multi-processor platform consisting of three homogeneous processing elements. As the synthetic application has a higher level of parallelism, it is mapped to a platform consisting of seven homogeneous processing elements. For both platform graphs with three and seven processing elements, the number of clock-frequency levels provided to each of the voltage-frequency islands is chosen to be five. More information on the applications and multi-processor platforms can be found in Sections 3.5.1, 4.3.1 and in Appendix B. The target frequency and the variation-related parameters for the random variables describing the maximum supported frequency of hardware resources in the multi-processor platforms are shown in Table 5.1. An explanation on the choice of these numbers is presented in Section 3.5.1. The single and multiple bindings of the actors in each resource-aware application graph to the processing elements in a platform graph are derived using the variation-aware heuristic mapping algorithms given in Algorithms 3 and 4. Similarly, the partitioning of the processing elements in a platform graph into voltage-frequency islands, given a single and multiple bindings, is determined by Algorithms 5 and 6, respectively.

We assume that die area consists of standard logic cells and embedded SRAM together with IO cells. Two scenarios are distinguished: guard-band reduction results in an overall decrease in die area; only the area of logic cells is reduced when reducing guard-bands. For the scenario, where overall die area is reduced, the formula for area computation is given in Equation (5.8). In the formula, v is the area reduction factor due to a $u\%$ guard-band reduction (Equation (5.1)); n_{pe} and n_{cgu}

Table 5.1: Target frequency and variation-related parameters assumed for hardware resources.

	pe	*rt*	*ni*	*lk*
f_{tg}^r (MHz)	300	500	500	560
σ_g^r (% of μ_g^r)	4	4	4	4
δ^r (% of μ_g^r)	5	0	0	5
σ_l^r (% of μ_g^r)	3.3	3.3	3.3	3.3

are the number of processing elements and clock-generation units, respectively; and V_{pe}, V_{noc} and V_{cgu} are the area of a processing element, the interconnect and the clock-generation unit, respectively. Note that an assumption that the area of clock-generation units does not change due to guard-band reduction is made. The area of a processing element is assumed to be 0.7 mm^2, based on the area of an ARM Cortex-A5 processor at 45 nm technology. The area of the interconnect is 3.1 mm^2. We assume that a typical fine-grained clock-generation unit has area of 0.03 mm^2 [68]. Note that, with 0% guard-band reduction, all processing elements and the interconnect can be clocked at their target clock frequencies. This requires a simple clock-generation unit providing a fixed clock frequency for all voltage-frequency islands. In contrast, a higher area cost clock-generation unit per island is necessary with reduced guard-bands. For a fair comparison, the area of clock-generation units with 0% guard-band reduction is neglected.

$$V = v \cdot (n_{pe} \cdot V_{pe} + V_{noc}) + n_{cgu} \cdot V_{cgu} \tag{5.8}$$

For the scenario, where only the area of logic cells is reduced, the are is computed by Equation (5.9), where ($V_{logic} + V_{fixed} = n_{pe} \cdot V_{pe} + V_{noc}$). We assume that 70% of the area of processing elements and the interconnect ($n_{pe} \cdot V_{pe}$) consists of standard logic cells, and 30% of embedded SRAM and IO cells. Equations (5.8) and (5.9) are used to compute the die area corresponding to a u% guard-band reduction. The number of gross dies on a wafer is derived by (5.2).

$$V = v \cdot V_{logic} + V_{fixed} + n_{cgu} \cdot V_{cgu} \tag{5.9}$$

The timing yield is evaluated by means of the proposed framework, using the mean frequency μ_g^r of hardware resources corresponding to a u% guard-band reduction. The mean frequency μ_g^r is computed by Equation (5.7). The throughput requirement of the cyclic synthetic application is set based on the target clock frequency of the hardware resources (Table 5.1). With the specified target frequencies, we assume that the application just satisfies its throughput requirement. This assumption enables us to have fair results when estimating the impact of guard-band reduction on timing yield (a relaxed throughput requirement creates a large slack in performance). To determine the throughput requirement, each application is mapped to the platform with the specified target frequencies, such that the throughput is maximized. The heuristic mapping algorithm presented in Section 3.4.2 is used for finding a mapping with maximized throughput. The result of the mapping is a throughput value, which is taken as the requirement for each application.

5.3.1 Evaluation results

Figure 5.2 illustrates the change in the number of good dies per wafer (normalized units) against guard-band reduction for the synthetic application with an eight voltage-frequency domain architecture (VFI-8), and for two different assumptions: a design with fixed blocks, where the area of embedded SRAM and IO cells does not change with guard-band reduction; and a design without fixed blocks (i.e. hard macros are newly designed corresponding to the guard-band reduction). The graphs are given for both a single and multiple bindings of the actors in the resource-aware synthetic application graph to the processing elements in the platform graph consisting of seven processing elements (Figure 4.2). The timing yield corresponding to different $u\%$ guard-band reduction values for both single and multiple bindings is shown in Table 5.2. Figure 5.2 shows that the number of good dies is maximized at a 20% guard-band reduction with a single binding and fixed blocks. This results in a 0.6% more dies that satisfy the throughput requirement imposed on the application. When there are no fixed blocks, a 30% reduction in the guard-band leads to a 3.1% increase in the number of good dies. With a single binding, the number of good dies beyond 30% guard-band reduction gradually decreases. This is because the reduction in the timing yield becomes considerable (see Table 5.2). It can be seen that a larger number of good dies is provided with multiple bindings. This is due to high timing yield, as reported in Table 5.2. Assuming a design with fixed blocks, the number of good dies is maximized at a 50% guard-band reduction, resulting in a 6.1% more good dies. A design without fixed blocks provides 12.3% more good dies with the same 50% guard-band reduction. Note that a 6.1% increase in the number of good dies is significant. For example, if 4 K wafers are required to produce 30 M good dies, a 6.1% larger number of good dies per wafer translates into 230 fewer wafers for the same 30 M good dies. For a wafer cost of €3000, the cost saving is €690,000.

Figure 5.3 shows the normalized change in the number of good dies per wafer against guard-band reduction for the same synthetic application, but with a five voltage-frequency domain architecture (VFI-5). A design with fixed blocks assuming a single and multiple bindings are considered. The timing yield for guard-band reduction values is given in Table 5.2. With a single binding, a 0.8% larger number of good dies are provided for a 20% guard-band reduction. This is slightly larger than the 0.6% given by the eight domain architecture due to the lower area cost of clock-generation units with VFI-5 (approximately the same timing yield is achieved by both VFI-8 and VFI-5 at the 20% reduction in guard-band, as shown in Table 5.2). On the other hand, multiple bindings with VFI-5 provide a much lower 3.25% improvement in the number of good dies, compared to the 6.1% with VFI-8. This is due to the considerably lower timing yield provided by the five domain architecture. Note that guard-band reduction also benefits in reduced dynamic and leakage power. These results show that a higher number of good dies with reduced guard-bands is obtained, increasing profit.

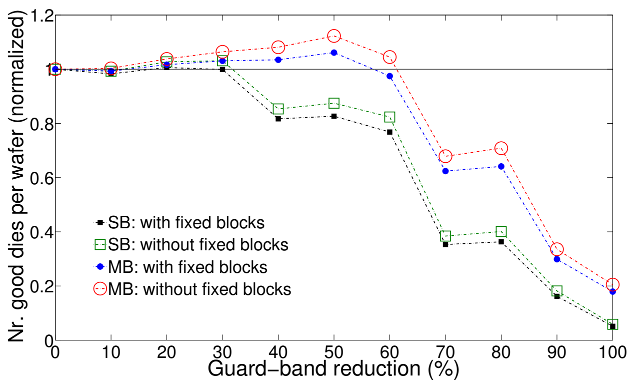


Figure 5.2: Number of good dies per wafer against reduced guard-band for the synthetic application and a VFI-8 architecture. The graphs are given for both a single and multiple bindings. Designs with and without fixed blocks are considered.

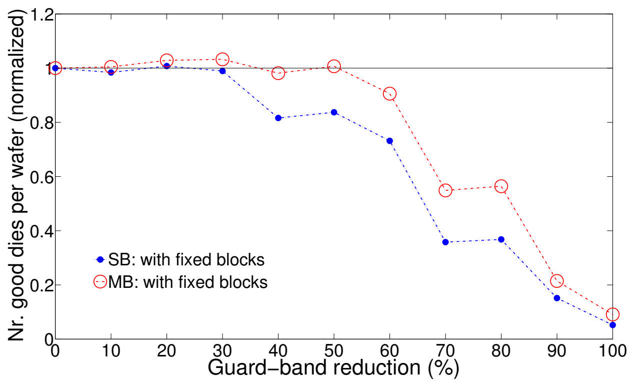


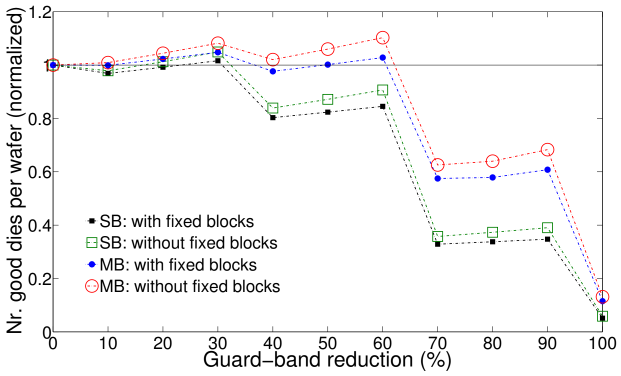
Figure 5.3: Number of good dies per wafer against reduced guard-band for the synthetic application a VFI-5 architecture. The graphs are given for both a single and multiple bindings. Only a design with fixed blocks is considered.

Figure 5.4 illustrates the number of good dies per wafer (normalized units) as a result of reducing guard-bands for the H.263 decoder, H.263 encoder and MP3 playback applications with a four voltage-frequency domain architecture (VFI-4). Both designs with and without fixed blocks are considered. The graphs are given for both a single and multiple bindings of the actors in each resource-aware application graph to the processing elements in the platform graph consisting of three processing elements. Similar results are seen for the H.263 decoder and MP3 playback applications. Considering the design with fixed blocks, the number of good dies is maximized at a 30% reduction in guard-bands, resulting in 1.6% and 4.8% more good dies for single and multiple bindings, respectively. For a design without fixed blocks, a single binding results in a 4.8% increase in the number of good dies for the same 30% guard-band reduction for both applications. With multiple bindings and without fixed blocks, a 30% guard-band reduction provides 10.3% and 11.5% more good dies for H.263 decoder and MP3 playback, respectively. For the H.263 encoder application, 0.6% and 3.7% increase in the number of good dies is observed with a single binding and for designs with and without fixed blocks. With multiple bindings, the improvements are 3.7% and 7% (30% reduction in guard-bands) with and without fixed blocks, respectively.

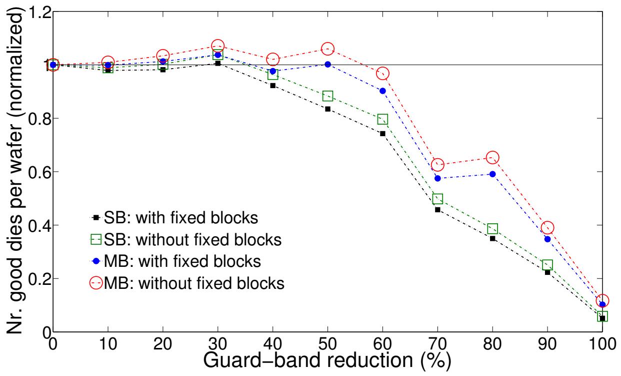
Table 5.2: Timing yield for u% reduced guard-bands.

u%	0	10	20	30	40	50	60	70	80	90	100
y% (SB, VFI-8)	99	97	97	94	75	74	67	30	30	13	4
y% (SB, VFI-5)	99	96	96	92	74	74	63	30	30	12	4
y% (MB, VFI-8)	99	98	98	97	95	95	85	53	53	24	14
y% (MB, VFI-5)	99	98	98	96	89	89	78	46	46	17	7

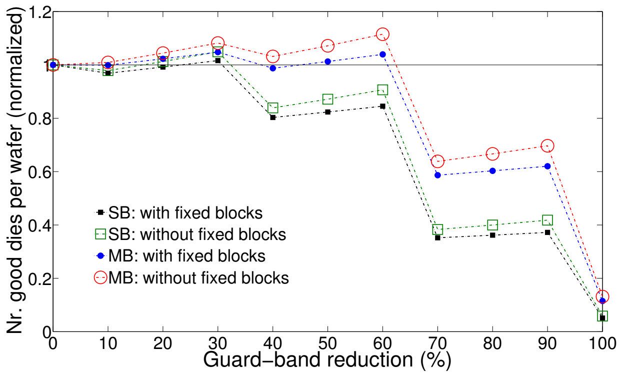
The change in the number of good dies due to reducing guard-bands for the Modem and MP3 decoder applications are shown in Figure 5.5. Significant improvements in the number of good dies are seen for the Modem application. More specifically, with a single binding and with a design with fixed blocks, a 50% guard-band reduction leads to an increase of 3.5% in the number of good dies. This is due to relatively high timing provided at a 50% reduction in guard-bands. When fixed blocks are not present, the improvement goes up to 9.5% with the same single binding. With multiple bindings, the guard-bands can be reduced by 70%, resulting in an increase of 9.6% and 18.8% for designs with and without fixed blocks, respectively. The number of good dies is maximized at the 70% guard-band reduction, as high timing yield is provided for guard-band reduction values up until 70%. Beyond 70%, the timing yield reduces, leading to a decrease in the number of good dies. For the MP3 decoder application, the number of good dies is in fact reduced by 0.5% at a 30% guard-band reduction with a single binding and a design with fixed blocks. This is due to a relatively low timing yield for the 30% reduced guard-bands. However, multiple bindings provide a higher timing yield and result in a 3.7% more good dies. With a design without fixed blocks, an increase of 2.7% and 7% in the number of good dies is observed with



(a) H.263 decoder



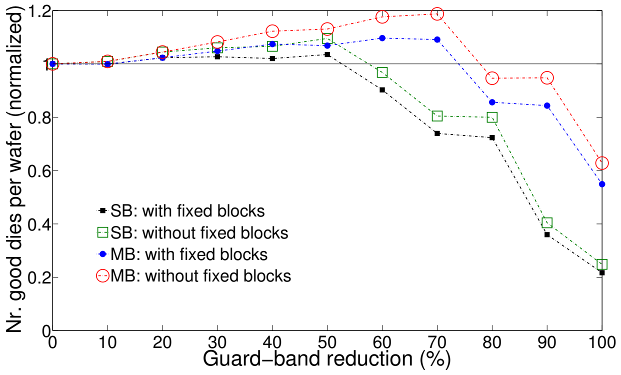
(b) H.263 encoder



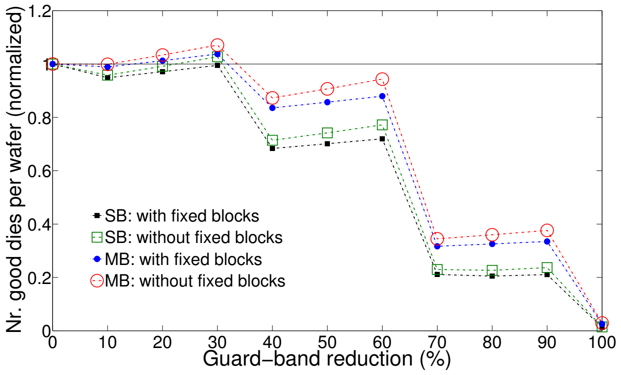
(c) MP3 playback

Figure 5.4: Number of good dies per wafer against reduced guard-band for the H.263 decoder, H.263 encoder and MP3 playback applications and a VFI-4 architecture. The graphs are given for both a single and multiple bindings. Designs with and without fixed blocks are considered.

a single and multiple bindings, respectively.



(a) Modem



(b) MP3 decoder

Figure 5.5: Number of good dies per wafer against reduced guard-band for the Modem and MP3 decoder applications and a VFI-4 architecture. The graphs are given for both a single and multiple bindings. Designs with and without fixed blocks are considered.

Note that the numbers for the SDF graphs modeling real applications are presented for an architecture where each processing element is in a separate voltage-frequency domain. As experimentally illustrated in Chapter 4, for most of the applications the number of islands can be reduced without affecting the timing yield (with a single binding). This leads to a reduction in the are cost due to clock-generation units, and in turn to more good dies per wafer. Similarly, the results in this section are given for an architecture, where five clock-frequency levels are provided to each of the voltage-frequency islands. It has been shown in Chapter 4 that more clock-frequency levels are more likely to provide a higher timing yield. Therefore, increasing the number of clock-frequency levels to each island is more likely to result in an increased number of good dies.

5.4 Summary

Designing circuits with reduced margins or guard-bands, referred to as better than worst-case design, provides the benefits of reduced circuit area and power consumption. Due to a smaller circuit and thus die size, a larger number of dies are placed on a wafer. This in turn may provide a larger number of good dies that satisfy the throughput requirement imposed on the system. The number of good dies is given by the product of the timing yield and the overall number of dies. In this chapter, we demonstrated on the synthetic and the set of SDF graphs modeling real applications how the proposed framework can be used to estimate the change in the number of good dies due to guard-band reduction. The results are presented for both cases with a single and multiple bindings for manufactured chips, and two different assumptions: a design with fixed blocks, where the area of embedded SRAM and IO cells does not change with guard-band reduction; and a design without fixed blocks, where the the overall area is reduced. Additionally, the impact of reducing the number of voltage-frequency islands, and thus circuit area (die size), on the number of good dies is analyzed.

For the case of a single binding for manufactured chips, an average of 1.4% increase in the number of good dies is observed due to 20%–50% guard-band reductions assuming a design with fixed blocks. The maximum improvement is 3.5% for the Modem application for a 50% reduction in guard-bands. For the MP3 decoder application, a 0.5% lower number of good dies is actually provided at a 30% guard-band reduction due to a relatively low timing yield. Considering a design without fixed blocks and a single binding for manufactured chips, the average improvement increases from the 1.4% to 4.8%. This increase is because of a larger reduction in circuit area when guard-bands are reduced, resulting in more dies on a wafer. The maximum improvement of 9.5% is still with the Modem application at a 50% guard-band reduction. This improvement is due to a relatively high timing yield value at the 50% reduction in the guard-bands. Further reducing the guard-bands considerably lowers the timing yield, and although the area is further reduced (i.e. more dies on a wafer), this results in a lower number of good dies.

With multiple bindings for manufactured chips, larger improvements in the number of good dies are obtained. This is due to higher timing yield values at larger guard-band reductions, compared to the case with a single binding. The larger the reduction in the guard-bands, the smaller the circuit is, resulting in more dies on a wafer. Therefore, if the timing yield is relatively high at a large guard-band reduction value, large improvements in the number of good dies are obtained. On average, the number of good dies is increased by 5.5% for 30%–70% guard-band reduction assuming a design with fixed blocks. The largest increase of 9.6% is for the Modem application, for which a high timing yield is provided given a large 70% reduction in guard-bands. The lowest improvement is for the H.263 encoder and MP3 playback, being 3.7%. Note that the average 5.5% increase in the number of good dies is significant. For example, if 4 K wafers are required to produce 30 M good dies, a 5.5% larger number of good dies per wafer translates into 208 fewer wafers for the same 30 M good dies. For a wafer cost of €3000, the cost saving is €624,000. For a design without fixed blocks, the average improvement in the number of good dies is 11.2% (18.8% the highest and 7% the lowest) for the same 30%–70% guard-band reduction. Additionally, the impact of reducing the number of voltage-frequency islands on the number of good dies was analyzed on the synthetic application. A five island architecture resulted in a negligible improvement in the number of good dies for the case with a single binding for manufactured chips, compared to an eight voltage-frequency islands (the timing yield for both five and eight island architectures is approximately the same). Another benefit of a reduced number of islands is the reduced power consumption, the evaluation of which is out of the scope of this thesis. However, with multiple bindings for manufactured chips, the five island architecture leads to a 2.8% smaller number of good dies with respect to eight islands. This is due to the considerably lower timing yield provided by the five island architecture.

6 Related work

This thesis proposes better than worst-case design or a design with reduced margins for real-time streaming applications, constrained by a throughput requirement, on a NoC-based multi-processor system with voltage-frequency islands. In this chapter, we position our work with respect to related work. In Section 6.1, we present techniques to mitigate the impact of process variation at the circuit level, and discuss the advantages and disadvantages of them. Later, we position the proposed mapping algorithms with respect to existing task allocation and scheduling techniques for MPSoCs in Section 6.2. Existing voltage-frequency island partitioning methods for multi-processor systems are briefly discussed in Section 6.3. Finally, Section 6.4 presents existing solutions for performing variation-aware throughput analysis of an application mapped to a multi-processor system.

6.1 Mitigating variation at the circuit level

The main ideas for reducing the impact of process variation at the circuit level are related to adaptive supply voltage and body biasing approaches [8, 15, 43, 45, 47, 55, 85, 86]. With voltage scaling, the supply voltage of an integrated circuit is changed, while body biasing refers to the adaptation of the threshold voltage of transistors. With both approaches, the current of transistors is changed, affecting the speed and power consumption of an integrated circuit. By increasing the supply voltage and applying *forward body bias*, the performance is increased. On the other hand, reverse body biasing together with reducing the supply voltage results in decreased power consumption. As shown in [47] by Meijer *et al.*, supply voltage upscaling and forward body biasing are effective for process-dependent performance compensation, and provide a large range of frequency upscaling. The disadvantage of the approach is the considerable increase in power consumption [47]. The better than worst-case design approach proposed in this thesis can be used in combination with voltage scaling and body biasing methods to further reduce design margins and increase the number of good dies.

6.2 Task allocation for MPSoC

There has been extensive research in the area of task allocation and scheduling for MPSoC [1, 9, 10, 14, 16, 30, 74, 79, 88]. The researchers in [1, 9, 10, 79] proposed methods to map throughput-constrained applications modeled as SDF graphs to resources in an MPSoC. However, none of them considers the impact of process variation. With a variation-unaware mapping approaches, the impact of process variation cannot be reflected by having different resources with different frequencies as the availability of a resource with a specific frequency is a matter of probability.

Wang *et al.* [88] introduced a new design metric called *performance yield*, defined as the probability of an assigned schedule meeting a predefined performance constraint. They proposed a variation-aware scheduling algorithm that allocates and schedules tasks with latency requirements modeled as an acyclic task graph to MPSoC, such that the performance yield is maximized. Resource sharing in task allocation and scheduling under process variation has been studied by Chon and Kim [16]. They proposed a statistical static timing analysis technique, which schedules and binds tasks in an acyclic task graph to the resources in an MPSoC in the presence of resource sharing, such that the performance yield is maximized. Singhal and Bozorgzadeh [74] introduced the problem of stochastically optimal task allocation, which tries to minimize the overall execution time of tasks in sequence and in parallel under process variation. Huang and Xu [30] took into account the spatial correlation characteristics of systematic within-die variation and presented a scheduling algorithm that schedules tasks with latency constraints in an acyclic task graph, such that the performance yield is maximized. With their solution, a set of schedules is synthesized off-line and based on the variation in each chip, a run-time scheduler selects the right one, such that the latency constraint is satisfied whenever possible.

All the solutions above that account for process variation use acyclic task graphs for application modeling and are based on latency requirements. Acyclic task graphs are not able to capture the iterative and overlapping execution of real-time streaming applications, which are primarily constrained by throughput requirements. Several real-life streaming applications, such as our H.263 Encoder, MP3 Playback and Modem benchmark applications, presented in Section 3.5 and in Appendix B, include cyclic data dependencies. For these applications, none of the solutions in the above work apply, as they cannot capture the cyclic data dependencies in the applications. In contrast, we allow arbitrary task graphs that may include cyclic data dependencies. Our solutions are primarily based on throughput requirements, but can be extended to cover latency requirements [58]. We additionally distinguish best-effort, firm real-time and soft real-time application classes, which benefit from having different optimization criteria. To the best of our knowledge, this is the first work that addresses the problem of variation-aware task allocation for cyclic task graphs.

6.3 VFI partitioning

The works in [38] and [31] address the problem of partitioning a tile-based network on chip architecture into voltage-frequency islands for minimized energy consumption, subject to performance constraints. The authors in [62] solve a similar problem, and propose a methodology for a run-time energy management through voltage (frequency) scaling, given workload and technology-related variations. None of these works consider process variation in the partitioning process. Majzoub *et al.* include process, voltage and temperature variations in the VFI partitioning process to minimize energy consumption [41]. They estimate expected voltage and temperature variations, and assume a given core-frequency map across a chip due to within-die process variation. In contrast, we consider both within-die and die-to-die variations, such that different chips have different core-frequency maps with associated probabilities; this is how the variation in reality behaves, as demonstrated by measurements in [64]. Moreover, the authors in [41] perform voltage-frequency island partitioning to minimize energy consumption, while we maximize timing yield in a probabilistic setting. We are not aware of any other work addressing the problem of voltage-frequency island partitioning for improved timing yield, considering process-driven variations.

6.4 Variation-aware throughput analysis

Marculescu *et al.* analyze the probability distribution of *latency* of systems with multiple voltage-frequency islands, considering within-die variation [42]. Their approach is only applicable to systems specified as acyclic task graphs, which are not able to capture the iterative and overlapping execution of many real-life streaming applications primarily constrained by a throughput requirement. In contrast, we allow arbitrary task graphs that may include cyclic data dependencies. We model a system by means of an SDF graph, which is well-suited for modeling and analysis of real-time streaming applications with throughput requirements. A methodology to perform system-level throughput analysis of multiple VFI designs, considering process variation, is presented in [23] by Garg *et al.* However, they only account for within-die variation, while we consider both within-die and die-to-die variations. Their approach is based on Homogeneous SDF (HSDF) graphs, which is a special case of an SDF graph, where all token rates associated with edges are equal to 1. We use an SDF graph for system modeling. An SDF graph provides much more compact application models, which is why many real-time streaming applications are modeled in an SDF formulation. To be able to use the approach in [23] for an application specified as an SDF graph, a conversion from the SDF graph to an equivalent HSDF graph is required [75]. This can lead to an exponential increase in the graph size (in terms of the number of actors and edges), as compared to the original SDF graph. For example, the SDF graph of our MP3 Playback benchmark application consists of only four actors, while the number of actors in an equivalent HSDF graph after conversion becomes 10601. Performing throughput analysis on such a large graph results in prohibitively high computation times, making the approach in [23] unsuitable for many applications. Additionally, the work in [23] assumes a one-to-one mapping of tasks to processing elements, while we allow resource sharing and assume static-order scheduling among tasks of an application allocated to the same core.

7 Conclusions and future work

This chapter provides concluding remarks on the work presented in this thesis. The limitations of the proposed approaches are briefly discussed and possible future directions for the work are proposed.

7.1 Conclusions

This thesis addresses the problem of designing real-time streaming applications, constrained by a throughput requirement, on a multi-processor system under better than worst-case design (i.e. with reduced design margins). With better than worst-case design, circuit area is reduced, resulting in more dies on a wafer. However, with reduced guard-bands, the target maximum supported frequency of hardware components in a multi-processor system is not guaranteed anymore. The goal of the work is to provide more good dies, which satisfy the throughput requirement of a real-time streaming application. To this end, a design flow is proposed in this thesis, consisting of several steps. Firstly, an application is allocated to a multi-processor platform, such that the timing yield, quantifying the percentage of chips that satisfy the throughput requirement of the application, is maximized. Maximizing the timing yield is essential, as the number of good dies is given by the product of the overall number of dies and timing yield. In this stage, we take advantage of having faster and slower processing elements in a multi-processor system due to process-induced within-die variation, which affects different hardware components on a chip die differently. The binding of the actors in the application to the processing elements is adapted based on the frequency of the processing elements in a chip instance. This is accomplished by deriving a set of bindings at design time, and based on each manufactured chip, the right binding that maximizes the throughput of the application is selected. Heuristic algorithms to derive a single or a set of bindings for manufactured chips, such that the timing yield is maximized are presented in this thesis. Once the binding (or set of bindings) are derived, a system designer can perform trade-offs between the number of voltage-frequency islands, as well as the number of clock-frequency levels provided to each island, and timing yield. Both the number of voltage-frequency islands and clock-frequency levels per island have an impact on the area cost of the clock-generation units associated with voltage-frequency islands. By reducing the number of islands and clock-frequency levels, circuit area is reduced and more gross dies on a wafer are provided. However, the timing yield can also decrease due to reducing the number of VFI partitions and clock-frequency levels. This may in the end provide less good dies. This is why careful trade-offs must be made. Heuristic algorithms are proposed to perform voltage-frequency island partitioning, such that the minimal loss in timing yield is achieved.

Having selected the binding (set of bindings), the voltage-frequency island configuration (set of configurations) and the number of clock-frequency levels, the number of good dies on a wafer is evaluated for different guard-band reduction values in the final stage of the flow. The reduction in guard-bands providing the most good dies is selected. All the steps of the design flow are evaluated on both synthetic and real applications. We show that the proposed better than worst-case design methodology can increase the number of good dies by up to 9.6% and 18.8% for designs with and without fixed SRAM and IO blocks, respectively, compared to worst-case design. Intuitively, the design without fixed blocks, where the overall die area is scaled down with guard-band reduction, results in more good dies, as smaller circuit area and thus more gross dies on a wafer are provided. Note that a 9.6% increase in the number of good dies is significant. For example, if 4 K wafers are required to produce 30 M good dies, a 9.6% larger number of good dies per wafer translates into 350 fewer wafers for the same 30 M good dies. For a wafer cost of €3000, the cost saving is €1,050,000.

7.2 Future directions

The work in this thesis can be extended in multiple ways. This section briefly outlines possible future directions.

- Systematic within-die variation exhibits spatial correlation. At the level of gates, this correlation is high, such that the parameters of nearby gates are affected similarly. However, it dies out quickly as a function of distance between gates on a die, resulting in much lower correlation between larger adjacent logic blocks. For simplicity, zero correlation between hardware components, such as processing elements, routers, network interfaces and links, is assumed in this thesis, as presented in Section 2.2. As an extension, the models presented in Section 2.2 can be extended such that correlation between hardware components can be specified. This will provide a modeling framework that more accurately captures the physical phenomenon of process-induced variation. The impact of different correlation maps on the number of good dies can also be analyzed.

- To evaluate the timing yield for an application, given a single or multiple bindings of it to a multi-processor platform, the throughput of a bound application graph must be computed for each chip-frequency vector. This can be seen in Algorithms 3, 4, 5 and 6. Given a large number of voltage-frequency islands in a platform graph and a large number of clock-frequency levels per island, the number of chip-frequency vectors (Definition 6) becomes very large, resulting in a very large number of throughput evaluations. This limits the scalability of the proposed mapping and voltage-frequency island partitioning algorithms. The scalability can be improved by pruning the combinations of chip-frequency vectors or by sacrificing accuracy. The treatment of this scalability problem will enable experimentation for future embedded systems with larger applications (in number of actors), platform graphs (in number of processing elements) and a larger number of clock-frequency levels.

- In the mapping stage, it is important to derive a binding or a set of bindings providing high timing yield. This ultimately translates into more good dies at the end of the flow. As illustrated in Section 3.5, the heuristic mapping algorithm is unable to find the binding that provides high timing yield for the MP3 playback application, resulting in a 70% reduction in the timing yield. An improvement of the heuristic algorithm can be a future extension to this work. A possible place to start is to implement a topology-aware actor criticality estimation (Section 3.4.2), such that a better initial resource allocation is performed.

- With reduced design margins, the power consumption of a circuit is lowered due to smaller area. It is interesting to evaluate the impact of the proposed better than worst-case design methodology on power consumption, compared to worst-case design. To this end, algorithms to minimize dynamic power consumption subject to throughput constraints can be devised considering the impact of process variation. This requires implementing necessary models to capture power consumption.

- In the proposed design-flow (including variation-aware mapping, voltage-frequency island partitioning and estimation of the number of good dies with reduced guard-bands), a single application is considered. In practice, multiple applications are mapped to a multi-processor platform. Possible future work is to extend the proposed design-flow, such that multiple applications are considered. For example, this can be achieved by TDM virtualization, where each application is given a portion of a multi-processor platform, such that there is no temporal interference between applications. In such a case, the presented mapping approaches for each application on a virtual platform may require no changes. However, as there are multiple applications mapped on the same platform, the metric of island criticality in the voltage-frequency island partitioning step of the flow must be reconsidered.

Bibliography

[1] H. Ali, L. M. Pinho, and B. Akesson. Critical-path-first based allocation of real-time streaming applications on 2D mesh-type multi-cores. In *Proc. Int'l Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2013.

[2] S. Amarasinghe, M. I. Gordon, M. Karczmarek, J. Lin, D. Maze, R. M. Rabbah, and W. Thies. Language and compiler design for streaming applications. *Int'l Journal of Parallel Programming*, 33(2/3):261–278, June 2005.

[3] M. Anis, M. Mahmoud, M. Elmasry, and S. Areibi. Dynamic and leakage power reduction in MTCMOS circuits using an automated efficient gate clustering technique. In *Proc. Design Automation Conference (DAC)*, pages 480–485, 2002.

[4] How reusable IP helps reduce product design cycles. http://www.eetimes.com/author.asp?section_id=36&doc_id=1319583, 2013.

[5] M. A. Bamakhrama, J. T. Zhai, H. Nikolov, and T. Stefanov. A methodology for automated design of hard-real-time embedded streaming systems. In *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 941–946, 2012.

[6] S. S. Bhattacharyya. *Compiling Dataflow Programs for Digital Signal Processing*. PhD thesis, EECS Department, University of California, Berkeley, 1994.

[7] S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee. Synthesis of embedded software from synchronous dataflow specifications. *Journal of VLSI Signal Processing Systems (IJVSPA)*, 21:151–166, 1999.

[8] D. Blaauw, S. Kalaiselvan, K. Lai, W.-H. Ma, S. Pant, C. Tokunaga, S. Das, and D. Bull. Razor II: In situ error detection and correction for PVT and SER tolerance. In *Proc. Int'l Solid-State Circuits Conference (ISSCC)*, pages 400–622, 2008.

[9] A. Bonfietti, L. Benini, M. Lombardi, and M. Milano. An efficient and complete approach for throughput-maximal SDF allocation and scheduling on multi-core platforms. In *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 897–902, 2010.

[10] A. Bonfietti, M. Lombardi, M. Milano, and L. Benini. Throughput constraint for synchronous data flow graphs. In *Proc. Int'l Conference on Integration of AI and OR Techniques in Constraint Programming (CPAIOR)*, pages 26–40, 2009.

[11] S. Borkar. Designing reliable systems from unreliable components: The challenges of transistor variability and degradation. *Proc. Microarchitecture (MICRO)*, 25(6):10–16, Nov. 2005.

[12] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De. Parameter variations and impact on circuits and microarchitecture. In *Proc. Design Automation Conference (DAC)*, pages 338–342, 2003.

[13] K. Bowman, S. Duvall, and J. Meindl. Impact of die-to-die and within-die parameter fluctuations on the maximum clock frequency distribution for gigascale integration. *Journal of Solid-State Circuits (JSSC)*, 37(2):183–190, Feb. 2002.

[14] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, 61:810–837, June 2001.

[15] T. Chen and S. Naffziger. Comparison of adaptive body bias (ABB) and adaptive supply voltage (ASV) for improving delay and leakage under the presence of process variation. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 11(5):888–899, 2003.

[16] H. Chon and T. Kim. Timing variation-aware task scheduling and binding for MPSoC. In *Proc. Design Automation Conference. Asia and South Pacific (ASPDAC)*, pages 137–142, Jan. 2009.

[17] W. Dally and B. Towles. Route packets, not wires: on-chip interconnection networks. In *Proc. Design Automation Conference (DAC)*, pages 684–689, 2001.

[18] M. Damavandpeyma, S. Stuijk, T. Basten, M. Geilen, and H. Corporaal. Modeling static-order schedules in synchronous dataflow graphs. In *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 775–780, 2012.

[19] A. Dasdan. Experimental analysis of the fastest optimum cycle ratio and mean algorithms. *Transactions on Design Automation of Electronic Systems (TODAES)*, 9(4):385–418, Oct. 2004.

[20] S. Dighe, S. Vangal, P. Aseron, S. Kumar, T. Jacob, K. Bowman, J. Howard, J. Tschanz, V. Erraguntla, N. Borkar, V. De, and S. Borkar. Within-die variation-aware dynamic-voltage-frequency-scaling with optimal core allocation and thread hopping for the 80-core TeraFLOPS processor. *Journal of Solid-State Circuits (JSSC)*, 46(1):184–193, 2011.

[21] S. Evain, J.-P. Diguet, and D. Houzet. NoC design flow for TDMA and QoS management in a GALS context. *Journal on Embedded Systems (EURASIP)*, 2006(1):4–4, Jan. 2006.

[22] P. Friedberg, Y. Cao, J. Cain, R. Wang, J. Rabaey, and C. Spanos. Modeling within-die spatial correlation effects for process-design co-optimization. In *Proc. Quality of Electronic Design (ISQED)*, pages 516–521, 2005.

[23] S. Garg and D. Marculescu. System-level throughput analysis for process variation aware multiple voltage-frequency island designs. *Transactions on Design Automation of Electronic Systems (TODAES)*, 13(4):1–25, Oct. 2008.

[24] A. Ghamarian, M. Geilen, S. Stuijk, T. Basten, A. Moonen, M. Bekooij, B. Theelen, and M. Mousavi. Throughput analysis of synchronous data flow graphs. In *Proc. Int'l Conference on Application of Concurrency to System Design (ACSD)*, pages 25–36, june 2006.

[25] K. Goossens, A. Azevedo, K. Chandrasekar, M. D. Gomony, S. Goossens, M. Koedam, Y. Li, D. Mirzoyan, A. Molnos, A. Beyranvand Nejad, A. Nelson, and S. Sinha. Virtual execution platforms for mixed-time-criticality systems: The CompSOC architecture and design flow. *To appear in Special Interest Group on Embedded Systems (SIGBED) Review, 2013.*

[26] K. Goossens, J. Dielissen, and A. Radulescu. Æthereal network on chip: Concepts, architectures, and implementations. *Design & Test of Computers*, 22(5):414–421, Sept. 2005.

[27] A. Hansson, M. Subburaman, and K. Goossens. Aelite: A flit-synchronous network on chip with composable and predictable services. In *Design, Automation Test in Europe Conference Exhibition*, pages 250–255, 2009.

[28] A. Hansson, M. Wiggers, A. Moonen, K. Goossens, and M. Bekooij. Enabling application-level performance guarantees in network-based systems on chip by applying dataflow analysis. *IET Computers & Digital*, 3(5):398–412, 2009.

[29] C. Hernandez, A. Roca, F. Silla, J. Flich, and J. Duato. On the impact of within-die process variation in GALS-based NoC performance. *Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 31(2):294–307, 2012.

[30] L. Huang and Q. Xu. Performance yield-driven task allocation and scheduling for MPSoCs under process variation. In *Proc. Design Automation Conference (DAC)*, pages 326–331, June 2010.

[31] W. Jang and D. Pan. A voltage-frequency island aware energy optimization framework for networks-on-chip. *Journal on Emerging and Selected Topics in Circuits and Systems (JETCAS)*, 1(3):420–432, sept. 2011.

[32] A. Jantsch. Models of computation for networks on chip. In *Int'l Conference on Application of Concurrency to System Design (ACSD)*, pages 165–178, 2006.

[33] K. Jeong, A. Kahng, and K. Samadi. Impact of guardband reduction on design outcomes: A quantitative approach. *Transactions on Semiconductor Manufacturing (SM)*, 22(4):552–565, 2009.

[34] R. Jordans, F. Siyoum, S. Stuijk, A. Kumar, and H. Corporaal. An automated flow to map throughput constrained applications to a MPSoC. In *Bringing Theory to Practice: Predictability and Performance in Embedded Systems (PPES)*, volume 18, pages 47–58, 2011.

[35] P. Kollig, C. Osborne, and T. Henriksson. Heterogeneous multi-core platform for consumer multimedia applications. In *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 1254–1259, 2009.

[36] E. Lee and D. Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75(9):1235–1245, 1987.

[37] E. A. Lee and D. G. Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. *Transactions on Computers*, 36(1):24–35, Jan. 1987.

[38] L.-F. Leung and C.-Y. Tsui. Energy-aware synthesis of networks-on-chip implemented with voltage islands. In *Proc. Design Automation Conference (DAC)*, pages 128–131, june 2007.

[39] J. Liang, S. Swaminathan, and R. Tessier. aSOC: A scalable, single-chip communications architecture. In *Proc. Int'l Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 37–, 2000.

[40] S. Londono and J. de Gyvez. A better-than-worst-case circuit design methodology using timing-error speculation and frequency adaptation. In *Proc. Int'l SOC Conference (SoCC)*, pages 15–20, Sept. 2012.

[41] S. Majzoub, R. Saleh, and R. Ward. PVT variation impact on voltage island formation in MPSoC design. In *Proc. Quality of Electronic Design (ISQED)*, pages 814 –819, march 2009.

[42] D. Marculescu and S. Garg. Process-driven variability analysis of single and multiple voltage frequency island latency-constrained systems. *Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 27(5):893–905, may 2008.

[43] M. Meijer and J. de Gyvez. Body-bias-driven design strategy for area- and performance-efficient CMOS circuits. *Transactions on Very Large Scale Integration (VLSI) Systems*, 20(1):42–51, 2012.

[44] M. Meijer, J. de Gyvez, and R. Otten. On-chip digital power supply control for system-on-chip applications. In *Proc. Int'l Symposium on Low Power Electronics and Design (ISLPED)*, pages 311 – 314, Aug. 2005.

[45] M. Meijer, B. Liu, R. Van Veen, and J. de Gyvez. Post-silicon tuning capabilities of 45nm low-power CMOS digital circuits. In *Symposium on VLSI Circuits*, pages 110–111, 2009.

[46] M. Meijer, F. Pessolano, and J. de Gyvez. Glitch-free discretely programmable clock generation on chip. In *Proc. Int'l Symposium on Circuits and Systems (ISCAS)*, volume 2, pages 1839 – 1842, May 2005.

[47] M. Meijer, F. Pessolano, and J. de Gyvez. Limits to performance spread tuning using adaptive voltage and body biasing. In *Proc. Int'l Symposium on Circuits and Systems (ISCAS)*, pages 5–8 Vol. 1, 2005.

[48] T. Meincke, A. Hemani, S. Kumar, P. Ellervee, J. Oberg, T. Olsson, P. Nilsson, D. Lindqvist, and H. Tenhunen. Globally asynchronous locally synchronous architecture for large high-performance ASICs. In *Proc. Int'l Symposium on Circuits and Systems (ISCAS)*, volume 2, pages 512–515 vol.2, 1999.

[49] D. Melpignano, L. Benini, E. Flamand, B. Jego, T. Lepley, G. Haugou, F. Clermidy, and D. Dutoit. Platform 2012, a many-core computing accelerator for embedded SoCs: performance evaluation of visual analytics applications. In *Proc. Design Automation Conference (DAC)*, pages 1137–1142, 2012.

[50] M. Millberg, E. Nilsson, R. Thid, S. Kumar, and A. Jantsch. The Nostrum backbone - a communication protocol stack for networks on chip. In *Proceedings of the International Conference on VLSI Design (VLSID)*, pages 693–, 2004.

[51] M. Miranda, B. Dierickx, P. Zuber, P. Dobrovoln, and F. Kutscherauer. Variability aware modeling of SoCs: From device variations to manufactured system yield. In *Proc. Quality of Electronic Design (ISQED)*, pages 547–553, March 2009.

[52] D. Mirzoyan, B. Akesson, and K. Goossens. Process-variation aware mapping of real-time streaming applications to MPSoCs for improved yield. In *Proc. Quality of Electronic Design (ISQED)*, pages 41–48, 2012.

[53] D. Mirzoyan, B. Akesson, and K. Goossens. Process-variation aware mapping of best-effort and real-time streaming applications to MPSoCs. *To appear in Transactions in Embedded Computing Systems (TECS)*, 2013.

[54] D. Mirzoyan, B. Akesson, S. Stuijk, and K. Goossens. Throughput analysis and voltage-frequency island partitioning for streaming applications under process variation. In *Proc. Embedded Systems for Real-Time Multimedia (ESTIMedia)*, 2013.

[55] M. Miyazaki, H. Mizuno, and K. Ishibashi. A delay distribution squeezing scheme with speed-adaptive threshold-voltage CMOS (SA-Vt CMOS) for low voltage LSls. In *Proc. Int'l Symposium on Low Power Electronics and Design (ISLPED)*, pages 48–53, 1998.

[56] G. Moore. Progress in digital integrated electronics. In *Int'l Electron Devices Meeting (IEDM)*, volume 21, pages 11–13, 1975.

[57] G. Moore. Cramming more components onto integrated circuits. *Proceedings of the IEEE*, 86(1):82–85, 1998.

[58] O. Moreira and M. Bekooij. Self-timed scheduling analysis for real-time applications. *Journal on Advances in Signal Processing (EURASIP)*, June 2007.

[59] O. Moreira, F. Valente, and M. Bekooij. Scheduling multiple independent hard-real-time jobs on a heterogeneous multiprocessor. In *Proc. Int'l Conference on Embedded software (EMSOFT)*, pages 57–66, 2007.

[60] J. Muttersbach, T. Villiger, and W. Fichtner. Practical design of globally-asynchronous locally-synchronous systems. In *Proc. Int'l Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 52–59, 2000.

[61] S. Nassif. Design for variability in DSM technologies [deep submicron technologies]. In *Proc. Quality of Electronic Design (ISQED)*, pages 451–454, 2000.

[62] U. Ogras, R. Marculescu, D. Marculescu, and E. G. Jung. Design and management of voltage-frequency island partitioned networks-on-chip. *Transactions on Very Large Scale Integration (VLSI) Systems*, 17(3):330–341, march 2009.

[63] H. Oh and S. Ha. Fractional rate dataflow model for efficient code synthesis. *Journal of VLSI Signal Processing Systems*, 37(1):41–51, May 2004.

[64] L.-T. Pang and B. Nikolic. Measurement and analysis of variability in 45nm strained-Si CMOS technology. In *Custom Integrated Circuits Conference (CICC)*, pages 129–132, 2008.

[65] L.-T. Pang, K. Qian, C. J. Spanos, and B. Nikolic. Measurement and analysis of variability in 45 nm strained-Si CMOS technology. *Journal of Solid-State Circuits*, 44(8):2233–2243, 2009.

[66] J. L. Pino, E. A. Lee, and S. S. Bhattacharyya. A hierarchical multiprocessor scheduling system for DSP applications. In *Proc. of the Asilomar Conference on Signals, Systems and Computers (ASILOMAR)*, pages 122–, 1995.

[67] P. Poplavko, T. Basten, M. Bekooij, J. van Meerbergen, and B. Mesman. Task-level timing models for guaranteed performance in multiprocessor networks-on-chip. In *Proc. Int'l conference on Compilers, architecture and synthesis for embedded systems (CASES)*, pages 63–72, 2003.

[68] A. Rylyakov, J. Tierno, G. English, M. Sperling, and D. Friedman. A wide tuning range (1 GHz-to-15 GHz) fractional-N all-digital PLL in 45nm SOI. In *Custom Integrated Circuits Conference (CICC)*, pages 431–434, 2008.

[69] R. Saleh, S. Wilton, S. Mirabbasi, A. Hu, M. Greenstreet, G. Lemieux, P. Pande, C. Grecu, and A. Ivanov. System-on-chip: Reuse and integration. *Proceedings of the IEEE*, 94(6):1050–1069, 2006.

[70] M. Schoeberl, F. Brandner, J. Spars, and E. Kasapaki. A statically scheduled time-division-multiplexed network-on-chip for real-time systems. In *Proc. Int'l Symposium on Networks on Chip (NOCS)*, pages 152–160, 2012.

[71] SDF^3 example applications. www.es.ele.tue.nl/sdf3/download/examples.

[72] SDF^3: SDF For Free. http://www.es.ele.tue.nl/sdf3/, 2009.

[73] A. Shabbir, A. Kumar, S. Stuijk, B. Mesman, and H. Corporaal. CA-MPSoC: An automated design flow for predictable multi-processor architectures for multiple applications. *Journal of Systems Architecture (JSA)*, 56(7):265–277, July 2010.

[74] L. Singhal and E. Bozorgzadeh. Process variation aware system-level task allocation using stochastic ordering of delay distributions. In *Proc. Int'l Conference on Computer Aided Design (ICCAD)*, pages 570–574, Nov. 2008.

[75] S. Sriram and S. S. Bhattacharyya. *Embedded Multiprocessors: Scheduling and Synchronization.* 1st edition, 2000.

[76] S. Sriram and E. A. Lee. Determining the order of processor transactions in statically scheduled multiprocessors. *Journal VLSI Signal Processing Systems*, 15(3):207–220, Mar. 1997.

[77] R. Stefan, A. Molnos, and K. Goossens. dAElite: A TDM NoC supporting QoS, multicast, and fast connection set-up. *Transactions on Computers*, 99, 2012.

[78] D. Stiliadis and A. Varma. Latency-rate servers: a general model for analysis of traffic scheduling algorithms. *Transactions on Networking (TON)*, 6(5):611–624, Oct. 1998.

[79] S. Stuijk, T. Basten, M. C. W. Geilen, and H. Corporaal. Multiprocessor resource allocation for throughput-constrained synchronous dataflow graphs. In *Proc. Design Automation Conference (DAC)*, pages 777–782, 2007.

[80] S. Stuijk, M. Geilen, and T. Basten. Exploring trade-offs in buffer requirements and throughput constraints for synchronous dataflow graphs. In *Proc. Design Automation Conference (DAC)*, pages 899–904, 2006.

[81] S. Stuijk, M. Geilen, and T. Basten. SDF^3: SDF For Free. In *Proc. Int'l Conference on Application of Concurrency to System Design (ACSD)*, pages 276–278, 2006.

[82] S. Stuijk, M. Geilen, and T. Basten. Throughput-buffering trade-off exploration for cyclo-static and synchronous dataflow graphs. *Transactions on Computers (TC)*, 57(10):1331–1345, 2008.

[83] M. B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, J.-W. Lee, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, and A. Agarwal. The Raw microprocessor: A computational fabric for software circuits and general-purpose programs. *Micro*, 22(2):25–35, Mar. 2002.

[84] W. Tong, O. Moreira, R. Nas, and K. van Berkel. Hard-real-time scheduling on a weakly programmable multi-core processor with application to multi-standard channel decoding. In *Proc. Real Time and Embedded Technology and Applications Symposium (RTAS)*, pages 151–160, 2012.

[85] J. Tschanz, J. Kao, S. Narendra, R. Nair, D. Antoniadis, A. Chandrakasan, and V. De. Adaptive body bias for reducing impacts of die-to-die and within-die parameter variations on microprocessor frequency and leakage. *Journal Solid-State Circuits (JSSC)*, 37(11):1396–1402, Nov 2002.

[86] O. Unsal, J. Tschanz, K. Bowman, V. De, X. Vera, A. Gonzalez, and O. Ergin. Impact of parameter variations on circuits and microarchitecture. *Micro*, 26(6):30–39, Nov. 2006.

[87] C. H. K. van Berkel. Multi-core for mobile phones. In *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 1260–1265, 2009.

[88] F. Wang, C. Nicopoulos, X. Wu, Y. Xie, and N. Vijaykrishnan. Variation-aware task allocation and scheduling for MPSoC. In *Proc. Int'l Conference on Computer Aided Design (ICCAD)*, pages 598 –603, Nov. 2007.

[89] H. M. G. Wassel, Y. Gao, J. K. Oberg, T. Huffmire, R. Kastner, F. T. Chong, and T. Sherwood. SurfNoC: a low latency and provably non-interfering approach to secure networks-on-chip. *SIGARCH Computer Architecture News*, 41(3):583–594, June 2013.

[90] Tilera corporation. http://www.tilera.com/.

[91] M. H. Wiggers, M. J. G. Bekooij, and G. J. M. Smit. Efficient computation of buffer capacities for cyclo-static dataflow graphs. In *Proc. Design Automation Conference (DAC)*, pages 658–663, 2007.

[92] D. Wiklund and D. Liu. SoCBUS: Switched network on chip for hard real time embedded systems. In *Proc. Int'l Parallel and Distributed Processing Symposium (IPDPS)*, page 8, 2003.

[93] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström. The worst-case execution-time problem overview of methods and survey of tools. *Transactions on Embedded Computing Systems (TECS)*, 7(3):36:1–36:53, May 2008.

[94] N. Wingen. What if you could design tomorrow's system today? In *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 1–6, 2007.

[95] W. Wolf, A. Jerraya, and G. Martin. Multiprocessor system-on-chip (MPSoC) technology. *Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 27(10):1701–1713, 2008.

[96] W. Wu, N.-C. Lee, and G. Schuellein. Multi-phase buck converter design with two-phase coupled inductors. In *Applied Power Electronics Conference and Exposition (APEC)*, page 6, March 2006.

A
Glossary

This chapter provides a guide to the abbreviations and the symbols used in this thesis. Section A.1 contains the list of abbreviations, while the list of symbols are given in Section A.2.

A.1 List of abbreviations

The abbreviations used in this thesis are listed below.

MPSoC	Multi-processor system-on-chip
NoC	Network on chip
TDM	Time-division multiplexing
GALS	Globally asynchronous, locally synchronous
VFI	Voltage-frequency island
CGU	Clock-generation unit
FIFO	First-in-first-out
SDF	Synchronous data-flow
HSDF	Homogeneous synchronous data-flow
MCM	Maximum cycle mean
PDF	Probability density function
CDF	Cumulative distribution function

A.2 List of symbols

Table A.1 contains most of the symbols used throughput this thesis. For each symbol, a description and the page number in which the symbol is introduced are given. The symbols are sorted in alphabetical order, where the Greek symbols precede the Latin.

Table A.1: List of symbols.

Symbol	Description	Page
α_d	Buffer space (in tokens) of a dependency edge d	23
β	Maximum bandwidth (in bytes per cycle) of a connection in an interconnect	13
β_d^{rq}	Bandwidth requirement (in bytes per cycle) of a dependency edge d	23
$\gamma(ga, gp, b)$	Timing yield of a bound application graph for a resource-aware application graph ga, a platform graph gp and a binding b	36
δ^r	Mean-frequency reduction of a hardware resource r	15
$\zeta(ga, gp, b)$	Sum of throughput degradations of a bound application graph for a resource-aware application graph ga, a platform graph gp and a binding b	38
$\theta(x, \mu, \sigma)$	CDF of a normal random variable x with mean μ and standard deviation σ	15
$\theta_m(gp, x, fi, f_0)$	CDF of the minimum of the maximum supported frequencies of hardware resources belonging to a VFI fi in a platform graph gp for a random variable x with respect to a global frequency value f_0	19
κ	Repetition vector of an SDF graph	22
μ^r	Mean of a normal random variable f^r	17
μ_g^r	Mean of a normal random variable f_g^r	15
σ^r	Standard deviation of a normal random variable f^r	17
σ_g^r	Standard deviation of a normal random variable f_g^r	15
σ_l^r	Standard deviation of a normal random variable f_l^r	15
$\tau(gb)$	Throughput of a bound application graph gb	30
$\tau_{avg}(ga, gp, b)$	Average throughput of a bound application graph for a resource-aware application graph ga, a platform graph gp and a binding b	35
$\Delta\tau_{avg}(ga, gp, b)$	Average throughput degradation of a bound application graph for a resource-aware application graph ga, a platform graph gp and a binding b	38
$\phi(x, \mu, \sigma)$	PDF of a normal random variable x with mean μ and standard deviation σ	15

$\Phi(x)$	CDF of the standard normal distribution	15
$\chi(pe)$	VFI to which a processing element pe belongs	14
$\psi(fi)$	Set of resources belonging to a VFI fi	14
Ψ	Number of hops of a connection in an interconnect	13
a	Actor	21
A	Set of actors	21
b	Binding vector	25
B	Set of all bindings	25
$c(gp, fi, n)$	n equidistant clock-frequency levels provided to a VFI fi in a platform graph gp	18
$ca(a)$	Criticality of an actor a	43
$ci(ga, gp, fi)$	Criticality of a VFI fi in a platform graph gp for a resource-aware application graph ga	67
d	Dependency edge	21
D	Set of dependency edges	21
$ec(a, pe)$	Execution time (in cycles) of an actor a on a processing element pe	23
$et(gp, a, pe)$	Execution time (in seconds) of an actor a on a processing elements pe	26
fc	Chip-frequency vector	18
FC	Set of all chip-frequency vectors	18
f^r	Normal random variable of the maximum supported frequency of a hardware resource modeling both die-to-die and within-die variations	17
f^r_g	Normal random variable of the maximum supported frequency of a hardware resource modeling die-to-die variation	15
f^r_l	Normal random variable of the maximum supported frequency of a hardware resource modeling within-die variation	15
fi	Voltage-frequency island	14
FI	Set of VFIs in a platform graph	14
ga	Resource-aware application graph	23
gb	Bound application graph	29
gp	Platform graph	14
ni	Network interface	13
NI	Set of network interfaces in an interconnect	13

noc	Interconnect	13
lk	Link	13
LK	Set of links in an interconnect	13
$lpe(pe)$	Load of a processing element pe	43
MBE	Exhaustive algorithm implementing the multiple-bindings mapping approach	50
MBH	Heuristic algorithm implementing the multiple-bindings mapping approach	52
MFBE	Mean-frequency based exhaustive mapping algorithm	50
MFBH	Mean-frequency based heuristic mapping algorithm	51
$p(gp, fc, f_0)$	Local probability of a chip-frequency vector fc in a platform graph gp with respect to a global frequency value f_0	21
$pc(gp, fc)$	Probability of a chip-frequency vector fc in a platform graph gp	21
pe	Processing element	13
PE	Set of processing elements in a platform graph	13
$pf(gp, f^i_{clk}, fi, f_0)$	Probability that a VFI fi in a platform graph gp is operated at a clock-frequency f^i_{clk} with respect to a global frequency value f_0	20
r	Hardware resource	13
R	Set of hardware resources in a platform graph	13
rt	Router	13
RT	Set of routers in an interconnect	13
$sdfg$	SDF graph	21
SBE	Exhaustive algorithm implementing the single-binding mapping approach	50
SBH	Heuristic algorithm implementing the single-binding mapping approach	52
sz_d	Token size (in bytes)	23
sz_{fl}	Flit size (in bytes)	13
sz_{tb}	TDM slot table size (in slots)	13
t_{req}	Throughput requirement of a resource-aware application graph	23

B

Application SDF graphs

This chapter presents the SDF models of the H.263 decoder [82], H.263 encoder [63], MP3 playback [91], Sample rate converter [7], Modem [7], MP3 decoder [82] and synthetic applications [71]. For each SDF model, the (worst-case) execution time (in cycles) and the size of data tokens (in bytes) communicated between actors across dependency edges are reported. The (worst-case) execution times of the actors in the H.263 decoder, H.263 encoder and MP3 decoder applications are given for a processor architecture similar to ARM7. The size of data tokens for the MP3 playback, Sample rate and Modem applications was not available at the provided references. For these applications, a size of four bytes is used. Similarly, no profiled or simulated data on execution times of actors for the Modem application is available. For this application, arbitrary execution times are used.

H.263 decoder

Figure B.1 illustrates the SDF graph of the H.263 decoder application. The (worst-case) execution time of the actors, together with the size of data tokens on the dependency edges, is given in Table B.1.

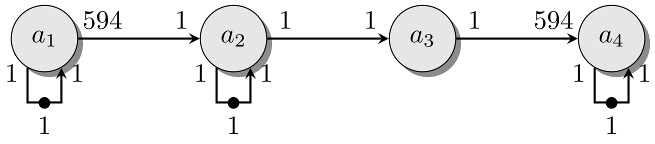


Figure B.1: An SDF model of an H.263 decoder.

The throughput requirement used in the experimental sections throughout this thesis is 858 iterations per second.

Table B.1: The execution time (in cycles) of actors and the size of data tokens (in bytes) sent across the dependence edges for the H.263 decoder.

Actor	Execution time	Edge	Token size
a_1 (VLD)	26018	a_1 self-edge	1024
a_2 (IQ)	559	a_4 self-edge	38016
a_3 (IDCT)	486	Others	64
a_4 (Motion comp.)	10958		

H.263 encoder

The SDF graph of the H.263 encoder application is presented in Figure B.2. The execution time of the actors and the size of data tokens are given in Table B.2. The throughput requirement is set to 230 iterations per second.

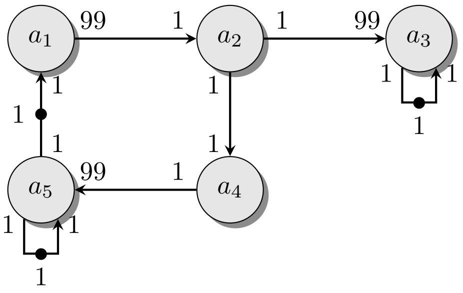


Figure B.2: An SDF model of an H.263 encoder.

Table B.2: The execution time (in cycles) of actors and the size of data tokens (in bytes) sent across the dependence edges for the H.263 encoder.

Actor	Execution time	Edge	Token size
a_1 (Motion est.)	382419	a_3 self-edge	1024
a_2 (MB enc.)	8409	a_5 self-edge	38016
a_3 (VLC)	26018	a_5 to a_1	38016
a_4 (MB dec.)	6264	Others	384
a_5 (Motion comp.)	11356		

MP3 playback

Figure B.3 illustrates the SDF model of the MP3 playback application. Each actor includes a self-edge with a single initial token, which are not shown in the figure. The execution time (in cycles) of the actors a_1, a_2, a_3 and a_4 are equal to 7510, 10000, 22 and 22 cycles, respectively. The size of the data tokens is chosen to be four bytes for all dependency edges. A requirement of 1227 iterations per second is imposed on the throughput for this application.

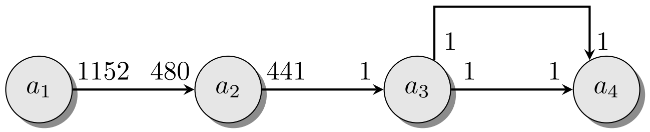


Figure B.3: An SDF model of an MP3 playback.

Sample rate converter

The SDF graph for the Sample rate converter application is given in Figure B.4. The self-edges with a single initial token for all actors are not shown in the figure. The execution times of the actors a_1, a_2, a_3, a_4, a_5 and a_6 are 11, 4, 6, 2, 9 12 cycles. The token size is chosen to be four bytes for all dependency edges. The requirement on throughput is 140000 iterations per second, as used in the experimental sections in this thesis.

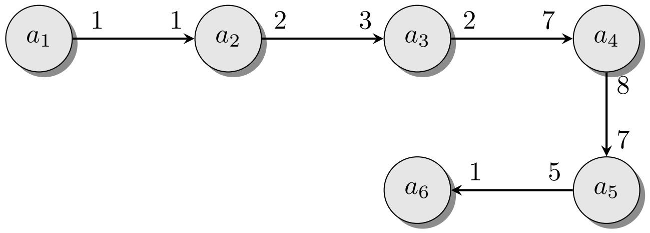


Figure B.4: An SDF model of a Sample rate converter.

Modem

Figure B.5 shows the SDF graph of the modem application. Each actor has a self-edge with a single initial token on it (not shown in the figure). The execution times of the actors a_1, a_5 and a_{16} are 130, 7000 and 1200 cycles, respectively. The rest of the actors take 1000 cycles to finish their execution. For all dependency edges (also the self-edges), the size of data tokens is four bytes. The throughput requirement used in the experimental section throughout the thesis is 12180 iterations per second.

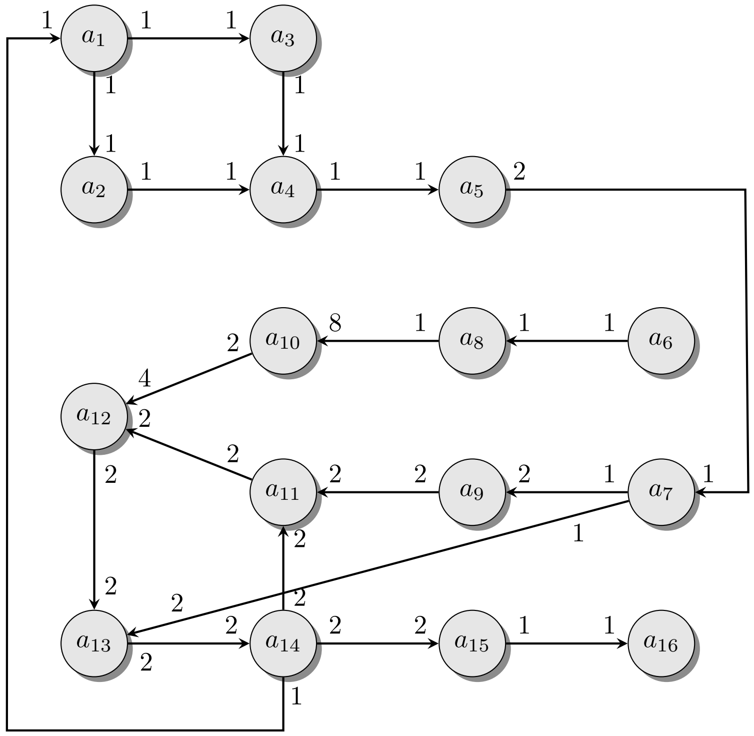


Figure B.5: An SDF model of a Modem.

MP3 decoder

The model of the MP3 decoder application is illustrated in Figure B.6. Table B.3 contains the (worst-case) execution times (in cycles) of all actors and the size of data tokens being communicated between actors across dependency edges. In all the experiments throughout this thesis, a throughput requirement of 76 iterations per second is used.

Table B.3: The execution time (in cycles) of actors and the size of data tokens (in bytes) sent across the dependence edges for the MP3 decoder.

Actor	Execution time	Edge	Token size
a_1 (Huffman)	151977	a_1 self-edge	1024
a_2, a_3 (Req.)	72695	a_2, a_3 self-edge	64
a_3, a_5 (Reorder)	34684	Others	576
a_6 (Stereo)	53602		

a_7, a_8 (Antialias)	409	
a_9, a_{10} (Hybrid synth.)	7414	
a_{11}, a_{12} (Freq. inv.)	4912	
a_{13}, a_{14} (Subb. inv.)	1865001	

Continued from previous page

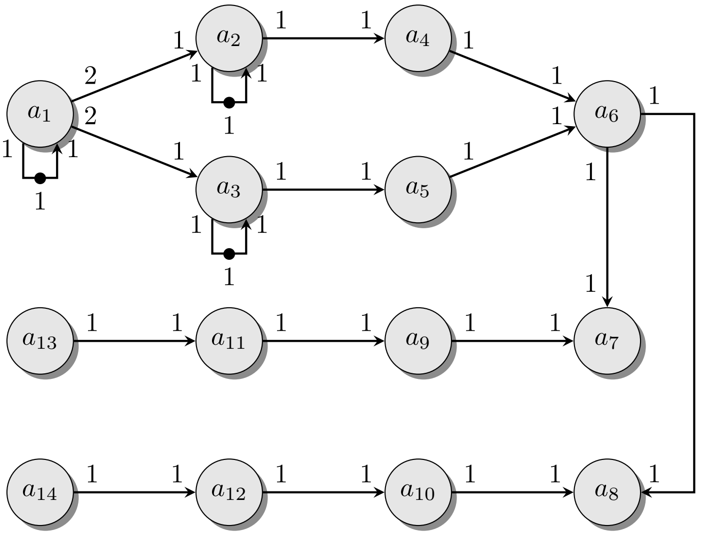


Figure B.6: An SDF model of an MP3 decoder.

Synthetic

Figure B.7 illustrates the topology of the synthetic application. The execution time of actors and the size of data tokens sent across the dependency edges are given in Table B.4. The requirement on throughput for this application is set to 322 iterations per second.

Table B.4: The execution time (in cycles) of actors and the size of data tokens (in bytes) sent across the dependence edges for the synthetic application.

Actor	Execution time	Actor	Execution time	Edge	Token size
a_1	62018	a_{10}	78280	a_{12} to a_{13}	20
a_2	82208	a_{11}	50228	a_{15} to a_{16}	10

a_3	62544	a_{12}	61122	a_{11} to a_{12}	100
a_4	52720	a_{13}	89112	a_1 to a_2	128
a_5	72352	a_{14}	35852	a_7 to a_8	8
a_6	78244	a_{15}	89012	a_2 to a_5	30
a_7	85628	a_{16}	89012	a_6 to a_7	40
a_8	59422	a_{17}	89012	a_2 to a_3	50
a_9	82302			Others	4

Continued from previous page

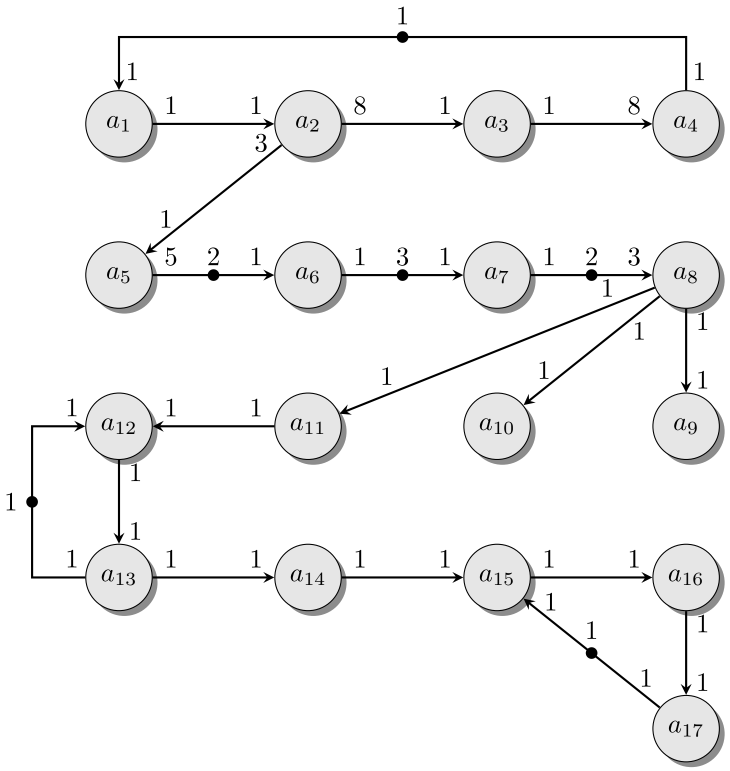


Figure B.7: An SDF model of a synthetic application.

C

About the author

Davit Mirzoyan was born in Yerevan, Armenia, in 1985. He acquired his B.Sc. degree in cybernetics from State Engineering University of Armenia (SEUA), located in Yerevan. He received his M.Sc. degree in Information and Communication Technologies from Royal Institute of Technology (KTH), Stockholm, Sweden. His graduation project for the M.Sc. program, on the topic of fault-tolerant memories, was carried out in Ikerlan, a research center located in Arrasate, Spain. In February 2010, Mr. Mirzoyan started his Ph.D. program in Computer Engineering at Delft University of Technology (TU Delft). His research interests include process-variation-aware design, system-level design and performance analysis.

D

List of publications

Journal articles

- Process-Variation Aware Mapping of Best-Effort and Real-Time Streaming Applications to MPSoCs. Davit Mirzoyan, Benny Akesson and Kees Goossens. *To appear in Transactions in Embedded Computing Systems (TECS), 2013.*
- Virtual Execution Platforms for Mixed-Time-Criticality Systems: The CompSOC Architecture and Design Flow. Kees Goossens, Arnaldo Azevedo, Karthik Chandrasekar, Manil Dev Gomony, Sven Goossens, Martijn Koedam, Yonghui Li, Davit Mirzoyan, Anca Molnos, Ashkan Nejad Beyranvand, Andrew Nelson and Shubhendu Sinha. *To appear in Special Interest Group on Embedded Systems (SIGBED) Review, 2013.*

Conference and workshop papers

- Process-variation aware mapping of real-time streaming applications to MPSoCs for improved yield. Davit Mirzoyan, Benny Akesson and Kees Goossens. In *Proc. Quality Electronic Design (ISQED), 2012.*
- Throughput Analysis and Voltage-Frequency Island Partitioning for Streaming Applications under Process Variation. Davit Mirzoyan, Benny Akesson, Sander Stuijk and Kees Goossens. In *Proc. Embedded Systems for Real-Time Multimedia (ESTIMedia), 2013.*
- Impact of Process Variations on the Throughput of Real-Time Applications in Multiprocessor Systems-on-Chip. Davit Mirzoyan, Benny Akesson and

Kees Goossens. *Workshop on PROGram for Research on Embedded Systems & Software (PROGRESS), 2010.*

Posters

- Impact of Process Variation on QoS in SRT Applications. Davit Mirzoyan, Benny Akesson and Kees Goossens. *Workshop on PROGram for Research on Embedded Systems & Software (PROGRESS), 2011.* (*Best poster award*).