Chapter 1

Introduction

EMBEDDED systems are an essential part of our lives and exist in a wide variety. In 2010, more than 15 billion embedded systems were sold globally [5]. The market for embedded systems was $113 billion [17]. This market has exhibited steady growth at a compound annual growth rate of 7% for the past 5 years.

An embedded system [84] is an information processing system embedded into devices, products or other systems, for instance mechanical or electrical systems. Different from servers or desktop PCs, embedded systems are often application domain specific and perform certain specific functions tightly coupled to their environment. Such systems can be hidden inside small and simple entities such as digital watches and traffic lights. They can be also a part of large and complex systems, such as Mars Exploration Rover [9].

Embedded streaming systems are an important class of embedded systems, which are specifically designed to process *streaming applications*. A streaming application [59] is a software program that processes large volume of continuous data streams in short periods of time. Typically, the same operation is performed on large set of data items in the stream. Therefore, there is little control flow between processing different data items. Each data item has short life time and is discarded after being processed. This type of applications is ubiquitous in telecommunications, health-care, transportation, retail, science, security, emergency response, and finance. This thesis focuses on those streaming applications that are commonly used in embedded systems. Figure 1.1 shows three popular streaming applications widely used in our daily lives on mobile phones.

(a) Navigation. (b) Gaming. (c) Video decoding.

Figure 1.1: Three examples of embedded streaming applications.

1.1 Embedded Streaming System Design

Designing embedded streaming systems is definitely a complex process. It involves three main aspects illustrated in the Y-chart [67], namely target applications, underlying platforms, and used design methodologies. This thesis primarily deals with the aspect of design methodology by proposing several novel techniques and a highly automated design framework. The proposed design techniques are highly optimized towards the target applications and platforms with important design requirements in mind. Therefore, first presenting the desired requirements, target applications, and platforms helps understand better the context and contributions of this thesis.

We first show in Section 1.1.1 that the design requirements for embedded streaming systems in general are more strict than general-purpose computing systems. The design techniques proposed in this thesis focus on an important subset of requirements, namely high throughout and hard real-time guarantees. Fortunately, properties of streaming applications can be well exploited with appropriate design techniques, such that the design requirements are satisfied. In addition, application properties show that modern streaming applications exhibit adaptive behavior that has to be explicitly captured in the design methodology. This is one major topic of this thesis. Therefore, we characterize the target applications in Section 1.1.2. Afterwards, we discuss the state-of-the-art hardware platforms in Section 1.1.3 and aim at understanding the architecture capabilities. Ideally, the proper design techniques should exploit application properties in such a way that matches exactly the underlying architecture capabilities. This reinforces the contributions of this thesis. Based on the application properties and selected architecture, we given an overview of a widely acknowledged design methodology in Section 1.1.4. The techniques developed in this thesis significantly extend and strengthen this design methodology.

Application	Resolution	Frame rate	Uncompressed bit rate	Compressed bit rate
HD-DVD	1920x1080	25	607 Mbps	8-20 Mbps
HDTV	1280x720	25	607 Mbps	2-8 Mbps
DVD	720x576	25	121 Mbps	1-2 Mbps
Video conferencing	352x288	25	30 Mbps	128-1000 Kbps
Mobile video	176x144	15	9 Mbps	50-1000 Kbps

Table 1.1: Processing requirements for video decoding (taken from [19]).

Application	Resolution	Uncompressed bit rate	Compressed bit rate
Projection Electronic Cinema	1280x720	350 Mbps	17.5 Mbps
Production HDTV	1920x1080	995 Mbps	140 Mbps
Projection Digital Cinema	4096x2048	6040 Mbps	450 Mbps
Production Digital Cinema	4096x3112	11000 Gbps	2200 Mbps

Table 1.2: Processing requirements for image processing (taken from [43]). All applications are assumed to operate at 24 FpsFrame per second.

1.1.1 Design Requirements

Requirements referred in this section are non-functional ones, such as performance, timing predictability, thermal aspects [60], security [69], and reliability [128]. The functional requirements such as deadlock-free execution are implicit. This thesis addresses the requirements of high performance and timing predictability.

Embedded streaming systems are expected to have high performance. Sometimes high performance is used interchangeably with high throughput. System throughput is a performance metric which denotes the average number of output data produced by the system per time unit. In general, a system with high throughput is referred to be *fast*. A Digital Video Broadcasting-Handheld (DVB-H) receiver found in mobile devices is a typical embedded streaming system with a certain throughput requirement. Unable to satisfy the throughput requirement results in the videos in slow motion and greatly degrades the user experience. In the video processing domain, the requirements of processing power has also increased drastically as screen resolution increases. Table 1.1 shows processing requirements for different resolutions. The state-of-the-art mobile phones, such as Samsung S4 [12], already have screens with the HD-DVD resolution. From the 5*th* column in Table 1.1, it should be clear that designing embedded video streaming system that satisfies the HD-DVD resolution poses a huge challenge. Image processing applications also require high throughput. Table 1.2 shows the processing requirements of different image processing applications used for digital and electronic cinema. The extreme high data rates clearly exceed the processing capacity of conventional embedded streaming systems. For the wireless communication, the requirements have significantly evolved over generations. The 3G standard targets 2 Mbps multimedia service including voice, video, and wireless Internet access. In contrast, it has been proposed in the 4G standard to increase the bandwidth of 100 Mbps or even 1 Gbps.

Besides high performance requirements, many embedded streaming systems pose *hard real-time* (HRT) requirements. In a HRT system [33], each application in the system has a deadline to indicate the maximum time within which the application must complete its execution. Missing any deadline may cause catastrophic consequence on the system. As noted in [33], a HRT system does not necessarily need to have high throughput requirements. Instead, the timing predictability is the major concern of the HRT system. That is, e.g., if a video conferencing system is guaranteed to produce a decoded video within 1 hour, this system still can be called a HRT system. Of course, this guarantee may not be useful in practice because the latency of producing an output is beyond being acceptable. For a realistic embedded streaming system, HRT constraints often come together with high throughput requirements. For instance, a collision avoidance system in the automotive or avionics domain is such an example. Processing input frames must be completed within a tight deadline. Missing the deadline will lead to catastrophic consequence for the vehicles, for instance potential collision to obstacles. At the same time, it has been reported in [6] that these algorithms require approximately 170 million calculations for each frame update, with the expectation of being executed on up to 64 processors.

1.1.2 Application Characterization

Although the requirements presented in Section 1.1.1 seem strict, streaming applications often contain ample amount of parallelism which can be exploited to satisfy the requirements. Therefore, a characterization of the application properties is needed, which heavily influences and motivates the solutions proposed in this thesis. In this section, streaming applications from different domains are characterized in terms of availability of parallelism and its different forms, computation and data communication characteristics, and adaptive behavior. The selected application domains contain those that are commonly used in embedded system, including video processing, wireless communication, and image processing/computer vision. Below we start by defining different forms of parallelism.

The type of parallelism is often categorized into three forms as follows:

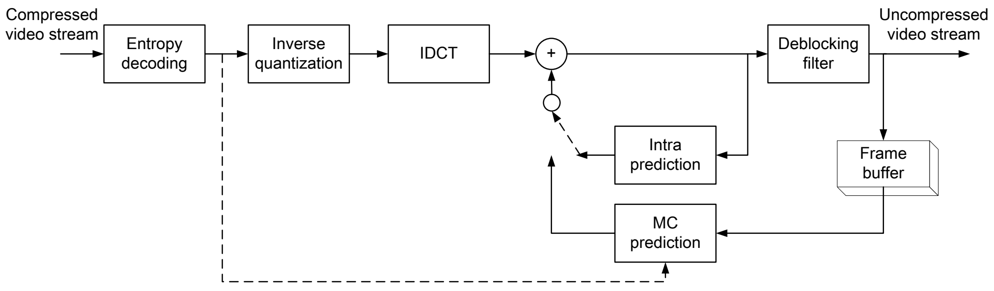


Figure 1.2: Block diagram of a H.264 decoder (taken from [18]). Each task is represented by a rectangular block.

1. Task-Level Parallelism (TLP): TLP refers to running different tasks of an application concurrently.
2. Data-Level Parallelism (DLP): DLP refers to running the same set of operations on multiple datum simultaneously.
3. Pipeline-Level Parallelism (PLP): PLP refers to running different iterations of a pair of producer and consumer tasks simultaneously.

In literature, TLP is often referred as thread level parallelism [58]. For instance, the block diagram of an H.264 decoder is depicted in Figure 1.2. Its computation can be partitioned into several tasks shown as blocks. Some of these tasks can run on different processors concurrently, thereby increasing the performance. DLP can be considered as a special case of instruction level parallelism [58], which was intensively studied in the past. The difference lies in the fact that DLP is explored at coarser level, e.g., at the processor level, whereas instruction level parallelism is exploited at finer level, e.g., using different functional units such as multiple Arithmetic and Logic Unit (ALU), floating point multipliers, *etc*. For instance in case of the H.264 video decoding, executing several video frames simultaneously on different PEs results in performance gain. PLP is an important form to exploit when parallelizing stateful computation (computation with cyclic dependencies) [54].

Video processing applications are in general good candidates for parallelization and demonstrate inherently adaptive behavior. For instance, a H.264 decoder contains major tasks, such as motion estimation, intra prediction, inverse discrete cosine transform, deblocking filter, and entropy coding. The H.264 decoder operates on data as set of Groups of Pictures (GoP). A GoP contains a set of frames. Several slices constitute a frame. Finally, a slice consists of several marcoblocks. Parallel scalability of H.264 video decoding is empirically studied in [85]. Large amount of DLP is shown to exist at different levels. The authors emphasize that DLP at different levels must be explored and especially at frame and marcoblock levels. In addition, the H.264 decoder also exhibits adaptive application behavior, namely three main types of slices/frames: I, P, and B types. For instance, a typical GoP consists of a I-P-B-B-P-B-B sequence of frames. On the one hand, processing an I-frame is independent from other frames. On the other hand, processing a P-frame depends on one or more previous frames, whereas processing B-frames depends on previous and future frames.

Software-Defined Radio (SDR) [88] applications also exhibit high parallelization opportunity and run-time adaptivity. For instance, the authors in [77] show that a 3G protocol, namely Wideband Code Division Multiple Access (WCDMA), demonstrates adaptive application behavior at different levels due to different operation modes and states. In the active mode, all computational tasks are active to process high rate traffic, whereas all tasks process at low rate in the control-hold mode. In the active mode, strict HRT requirements must be guaranteed to avoid buffer overflow, while the timing requirements are much more relaxed in the idle mode. The authors in [77] further characterize the computational workload of the tasks in the WCDMA protocol. Computationally intensive tasks, such as Branch Metric Calculation and Add Compare Select, contain enormous amount of DLP and TLP. This fact can be exploited to achieve an efficient parallel implementation. The authors in [130] study the computational workload of major 4G tasks. The tasks, such as Space Time Block Codes and Vertical Bell Laboratories Layered Space-time, contain abundant amount of DLP.

Computer vision is another important target application domain of embedded streaming systems. The applications in this domain are widely used in the fields of automotive, robotics, medicine, etc. Disparity Map [82] is such an example application that is used for adaptive cruise control on robotics or vehicles. It continuously processes a pair of images taken at slightly different positions. A disparity map is then computed in which depth information of all objects is represented. Since image processing kernels are often used, computer vision and image processing applications are categorized together in this thesis. In general, the applications in this domain contain large amount of DLP and TLP [121]. Typically, the same operations are performed repeatedly on all pixels in each image. At a higher level, there exists a few data dependencies between images in many applications. In this case, DLP at the image level can be also exploited. Next to DLP, different tasks of an application can execute normally in a feedforward pipeline fashion. Thus, there also exits a large amount of TLP and PLP to explore. In some applications, adaptive application behavior is an inherent part. For example, Feature Tracking [81] aims at extracting motion information from a set of consecutively captured images. During its execution at run-time, the features are first extracted. The number of extracted features and their width are expressed as parameters. The parameter values cannot be completely determined at compile-time and their values must be updated at run-time.

Finally, a collection of 65 real-life streaming applications is characterized in the StreamIT benchmark suit [116], to study their impact on language and compiler design. The applications are from different domains including video/audio processing, graphics rendering, DSP, and encryption. An important finding is that DLP should be considered as the first class citizen for performance optimization. In another important finding, the authors emphasize that cyclic data dependencies are uncommon in the application specifications. Around 90% of the studied benchmarks does not have cyclic data dependencies.

1.1.3 Platform Implications

Traditionally, the solution to achieve higher performance always involves the design of a system with higher frequency. However, as the technology node reaches below 100 nm, a single processor running at high frequency leads to extremely high power consumption [73]. Using Multi-Processor System-on-Chip (MPSoC) platforms partially addresses this problem by running processors at a lower frequency, which reduces power consumption. An MPSoC [132] is a very large scale integration system that incorporates most or all the components, including multiple programmable Processing Elements (PE) [1], peripheries, and memories, necessary for an application. It is widely acknowledged that MPSoC platforms are the best candidate to cope with various increasing requirements for embedded streaming systems. This thesis focuses on two important components, namely multiple PEs and the interconnection transferring data between them.

As the technology node further shrinks, chips with the same size of die is capable of accommodating more PEs. Together with the increasing performance requirements as motivated previously in Section 1.1.1, it can be expected that the number of PEs on a single chip will continue to increase. The processing part of an MPSoC platform for mobile devices is shown in Figure 1.3. Normally, the platform is equipped with multi-core CPUs which handle high-level applications, such as rendering Web pages and user interface functionalities. Next to the multi-core CPU, the multi-core GPU contains a set of PEs, which performs 2D/3D graphical processing. For instance, the Nvidia Tegra 4 [10] platform offers a quad-core CPU and a 72-core GPU. In addition to CPU and GPU, there are other programmable processors dedicated to certain class of functionalities. For example, a dedicated processor is often used to handle wireless communication protocols. To

[1]The term “PE” is used interchangeably with “core” or “processor” in this thesis.

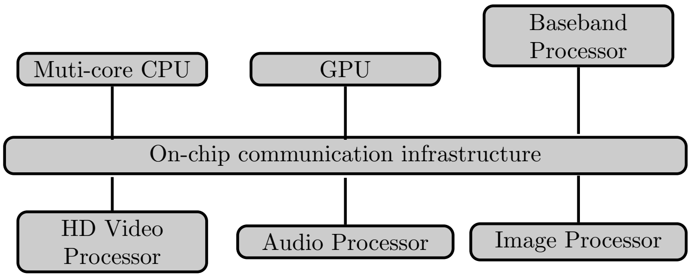


Figure 1.3: Processing part of an MPSoC platform for mobile devices from Nvidia (taken and simplified from [11]).

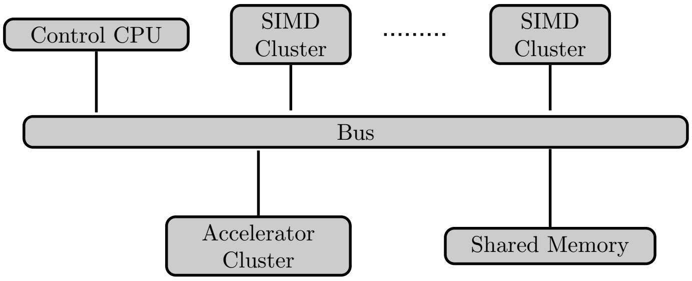


Figure 1.4: Template of a baseband processor (taken and simplified from [103]). All interfaces and peripheries are omitted.

be able to support multiple protocols, a programmable solution at the physical layer has emerged. For a baseband processor using the SDR technique, its template is illustrated in Figure 1.4. It consists of a control CPU for the processing protocol stack and hosting OS to orchestrate computation on other parts of the platform. For computationally intensive parts of applications, several Single Instruction Multiple Data (SIMD) clusters are used to support different algorithms in various wireless protocols. For instance, the Ardbeg [131] architecture has two SIMD clusters with one PE in each cluster. A PE is mainly a SIMD core with local memory. X-GOLD [103] is another instance of a baseband processor. It mainly differs from Ardbeg in the number of SIMD cores and size of local memory.

In addition to the PEs on an MPSoC platform, another important architectural element is the on-chip communication infrastructure. Network-on-Chip (NoC) [27] as the communication paradigm has emerged to alleviate the problem of platform scalability and its design has been one of the hottest research topics in the past decade. Æthereal [53] and Xpipes [28] are two prominent examples of NoC developed in academia. Æthereal provides bandwidth guarantees and thus it is more suitable

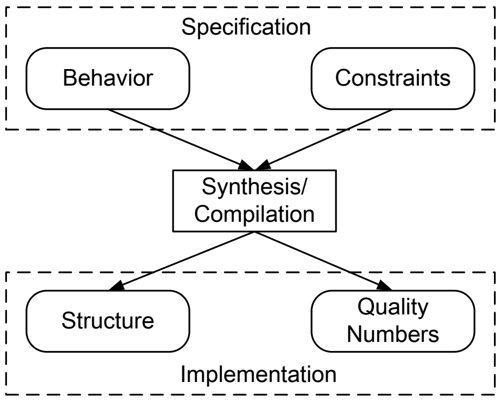


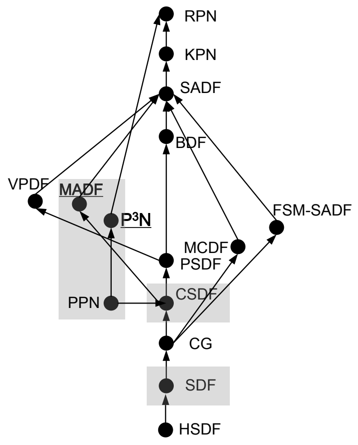
Figure 1.5: X-chart: a general design process (adopted from [50]).

for real-time systems due to bounded communication latency. Commercial NoC solutions [3] also have been integrated into the state-of-art MPSoCs for mobile phones.

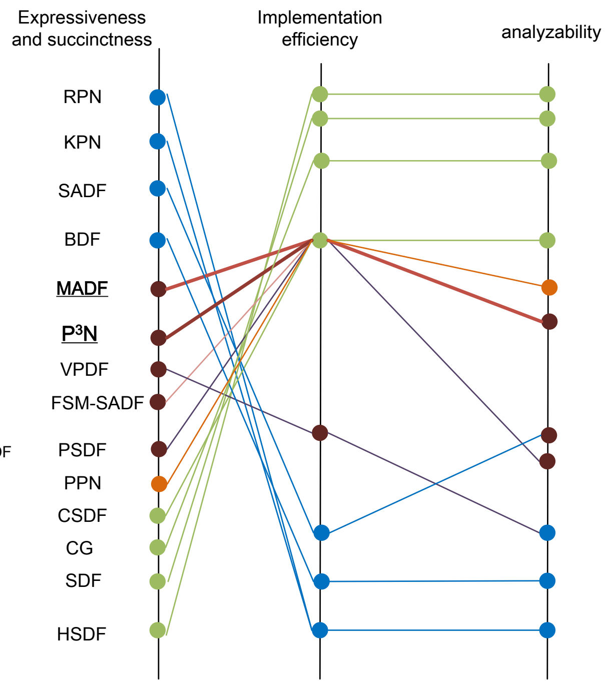
1.1.4 Model-based Design Methodology

The high system requirements presented in Section 1.1.1 and platform complexity presented in Section 1.1.3 impose huge design challenges for designers to develop an efficient system manually. The traditional design process at a low-level of abstraction becomes very error-prone and time-consuming. It is widely recognized in the research community that rising the level of abstraction to Electronic System Level (ESL) [50] seems inevitable to increase the design productivity.

A complete design flow defined in [50] is shown in Figure 1.5. For the *specification* layer sitting on the top, an important component is called *behavioral model*. The behavioral model is specified either in certain programming language, such as C/C++/SystemC, CAL [38], StreamIT [117], Verilog/VHDL, or graphical representations, such as LabVIEW-G [20] and Simulink [8]. Different from general programming, a behavioral specification used for embedded system design normally complies with the underlying *Model of Computation* (MoC). A MoC [74] defines components and communication protocols that constraint the mechanism by which components can interact. A MoC is a formal model of how computation works. Consequently, adopting MoCs during the design process allows automated tools to reason about both functional and non-functional properties of an application. In the context of this thesis, only *concurrent* MoCs are considered because they are the natural way to express parallelism in streaming applications in an explicit way. Normally, a concurrent MoC describes an application by a directed graph



(a) Expressive hierarchy of MoCs. The MoCs considered in this thesis are highlighted by the boxes.



(b) Three aspects when comparing MoCs.

Figure 1.6: Comparison of dataflow MoCs for streaming applications (taken and extended from [112]). The MoCs underlined are proposed in this thesis.

where nodes are application tasks representing computation and the arcs represent communication. Consequently, MoCs greatly facilitate parallelizing compilers to perform aggressive optimizations. Therefore, both industrial and academic design flows extensively adopt different MoCs.

Figure 1.6 shows different MoCs widely used for modeling streaming applications. They differ in *expressiveness*, implementation efficiency, and compile-time *analyzability*[2]. Figure 1.6(a) shows the expressive hierarchy of different MoCs. The expressiveness and succinctness [112] of a MoC indicate which system can be modeled and how compact the models in these MoCs are. In most of cases, an arrow from MoC *A* to MoC *B* indicates that that a model in MoC *A* can be transformed to an input-output equivalent model in MoC *B*. In general, the MoCs with high expressiveness exhibit low compile-time analyzability. Similarly, the MoCs with

[2]Analyzability is referred as *decidability* in [55].

high expressiveness generally have lower implementation efficiency. The analyzability of a MoC [112] is determined by the availability of analysis and synthesis algorithms at compile-time and the run-time need for an algorithm on a graph with a given number of nodes and edges. The third aspect, implementation efficiency of a MoC [112] is decided by the complexity of the run-time scheduling algorithm problem and the (code) size of the resulting schedules. When comparing adaptive MoCs, we also consider the incurred performance overhead during run-time reconfiguration. As shown in Figure 1.6(b), Reactive Process Network (RPN) [46], Kahn Process Network (KPN) [64], Scenario-Aware Data Flow (SADF) [114], and Boolean Data Flow (BDF) [32] are Turing-complete MoCs, thereby being highly expressive. That is, this type of MoC is able to perform any computation that any other computer is capable of. However, these MoCs do not offer many possibilities of analysis at compile-time. At the bottom part of Figure 1.6(b), the MoCs, such as Synchronous Data Flow (SDF) [76], Cyclo-Static Data Flow (CSDF) [30], and Polyhedral Process Network (PPN) [125], exhibit high compile-time analyzability. They are discussed in detail in Chapter 2. For these MoCs, various powerful analysis and compilation/synthesis methods have been developed over the past twenty years, e.g., to compute throughput [52, 87], buffer sizes [110], efficient static schedules for software compilation [91, 107, 124], and hardware synthesis [63, 120]. However, these MoCs are restricted to static application behavior. Modern streaming applications with adaptive behavior as explained in Section 1.1.2 cannot be expressed using these MoCs. To model adaptive behavior while having certain degree of compile-time analyzability, different adaptive MoCs, such as Mode-controlled Data Flow (MCDF) [89], Finite State Machine (FSM)-based Scenario-Aware Data Flow (FSM-SADF) [47], Parameterized SDF (PSDF) [29], and Variable-rate Phased Data Flow (VPDF) [129], have been proposed. For these MoCs, functional properties of the adaptive MoCs can only be partially decided at compile-time, and run-time verification is thus needed. For SADF, it is even possible to statically analyze functional properties at compile-time.

To take advantage of different properties of MoCs, some design flows separate the *analysis* model from the *implementation* model. Here the implementation model is the one that is close to the final implementation to be executed on the real MPSoC platform, whereas the analysis model is primarily used for analyzing non-functional properties. In this thesis, the timing property is of particular interest. For instance in the current industrial practice, a disciplined version [70] of C is used as the implementation model to program embedded radio applications, including code generation for communication and/or synchronization. On the other hand, analysis of real-time guarantees, required buffer sizes, etc., is performed on the SDF MoC, which serves as the analysis model.

Next to the behavioral model, the specification layer of the design flow shown in Figure 1.5 may contain *platform constraints* that explicitly specify the platform model. As explained in Section 1.1.3, that is, e.g., the type and number of PEs, the memory type and capability, and the interconnection between PEs. In addition to the platform constraints, other constraints can be used as input to the design flow in this thesis, such as timing constraints. In particular, the timing constraints are the essential property of a real-time streaming system.

With the behavior model, namely MoCs, and constraints in place, they are transformed in a step, called *synthesis* or *compilation* (in case of software models). This step normally determines e.g., allocation of PEs and necessary buffers if not given before hand, spatial mapping[3] of application tasks on PEs, temporal scheduling of all tasks on a PE, etc. Obtaining an efficient solution for these problems is certainly very challenging. In most cases, all possible combinations of PE allocation and assignment of tasks to PEs constitute an enormous design space with different conflicting objectives. For example, maximum throughput should be achieved while resource usage needs to be minimized. To efficiently search the design space and find an optimum solution, various Design Space Exploration (DSE) approaches proposed in the literature try to find a solution that is called *Pareto-optimal* point in the design space if, e.g., higher throughput cannot be achieved with fewer PEs. Currently, existing DSE approaches search the design space using different algorithms, e.g., stepwise refinement in [51], heuristics in [109] and [111], evolutionary algorithms in [100, 115], branch-and-bound in [34], and constraint programming in [139]. The synthesis/compilation step outputs a *structure* model as shown in Figure 1.5. Here the structure model is (or closer than the behavioral model to) the final, executable implementation. It may be in the form of pin-accurate netlists or Transaction-Level Models (TLM). As an output next to the structure model, *quality numbers* represent non-functional properties, e.g,. throughput, end-to-end latency, etc.

An Incarnation: Daedalus^{RT} Design Flow

The Daedalus^{RT} [23] design flow is based on the initial Daedalus [96, 97] framework, which covers all three layers in Figure 1.5, namely system-level DSE, synthesis, and prototyping of MPSoCs. Daedalus^{RT} has been recently proposed, as the name suggests, to address HRT requirements (see Section 1.1.1), The research work of this thesis has been performed in the context of the Daedalus^{RT} design flow and an overview of Daedalus^{RT} is shown in Figure 1.7. The grey boxes highlight the contributions of this thesis, which are explained in details in Section 1.3.

[3]Task mapping is often also referred as task allocation in literature and both are used interchangeably in this thesis.

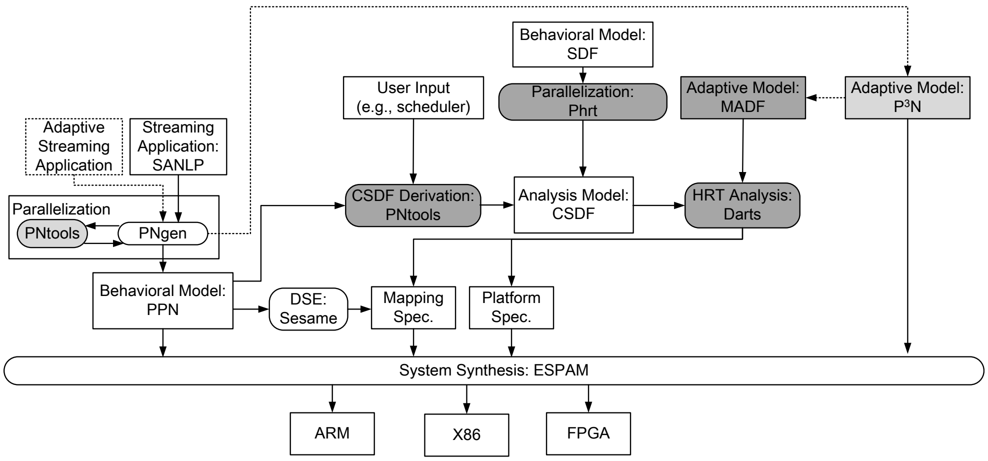


Figure 1.7: Daedalus^{RT} design flow. The grey boxes highlight the contributions of this thesis. The dashed box and lines denote the parts that are currently not fully implemented.

The input to Daedalus^{RT} is a streaming application specified as a sequential C code with restrictions, called Static Affine Nested Loop Program (SANLP) [125] (see Section 2.1). Many streaming applications are amenable to this restricted form [26]. Moreover, an early study [106] has shown that, out of 100,000 lines of loops, 53% of them can be converted to SANLPs. In the Parallelization step, a SANLP is automatically translated to its equivalent behavioral model, the PPN MoC using the PNgen compiler [125]. The resulting PPN exposes certain form of parallelism, specifically TLP of the initial SANLP. Currently, the PNgen compiler also extracts DLP from a SANLP in a particular way using a combination of transformations [86]. The formal definition of SANLP and the PPN MoC is detailed later in Section 2.1. Alternatively, application designers also have the flexibility to specify streaming applications as (C)SDF graphs directly. It is sometimes more convenient to do so using tools based on graphical interfaces. For adaptive streaming applications, they are specified as two new MoCs proposed in this thesis. Their details can be found in Chapter 6 and Chapter 7, respectively.

In the initial Daedalus framework, the second step, namely DSE, is realized using the Sesame [100] tool, which takes a PPN as input and generates a Pareto-optimal set of design points. A design point consists of a platform and mapping specifications. For HRT streaming systems, an analysis model, the CSDF MoC, is required. In Daedalus^{RT}, a PPN derived from a SANLP needs to be converted to its equivalent CSDF graph. Subsequently, the Darts tool replaces time-consuming DSE and performs the HRT analysis [22] on the resulting CSDF graph. The main advantage of the HRT analysis is the fast, yet accurate determination of the minimum number of PEs needed to schedule the CSDF graph and leveraging well-known HRT multiprocessor scheduling algorithms. The HRT analysis on the CSDF MoC is detailed in Section 2.3.

Finally in the third step, namely System Synthesis, the ESPAM [95, 96] tool takes a PPN with the platform and mapping specifications, and produces an executable implementation on various platforms. The platform consists of several tiles interconnected via certain communication infrastructure. On the FPGA-based platform, each tile consists of a PE in the form of the MicroBlaze [13] softcore from Xilinx with its local program and data memories. A communication memory resides in each tile and it is used as data storage for communication between application tasks mapped to different tiles. The interconnection between all tiles, the DDR off-chip memory, and peripheries is an AXI crossbar switch [2]. In principle, the crossbar switch can be replaced by e.g., the Æthereal [53] NoC, to provide guaranteed communication latency. For the PEs, ARM Cortex A9 [1] cores can be instantiated on the Xilinx Zynq [16] platform instead of the MicroBlaze softcore. In Daedalus, a static schedule [124] is used on each PE to temporally schedule all tasks allocated on the PE. Alternatively, a light-weight and multi-threaded OS, Xilkernel [14], is built on top of a PE to perform run-time scheduling. Later, support for the x86 platform has been added to the ESPAM backend [39]. The target is normally desktop multi-core platform, such as Intel i7-920 processor. For the x86 platform, application tasks implemented as threads can be dynamically scheduled by OS, such as Windows or Linux. In this case, OS either determines allocation and temporal schedule of all threads at run-time. Alternatively, the threads are statically bound to a PE by assigning core affinity. In the latter case, no run-time migration of threads is required, thereby reducing performance penalty. In Daedalus$^{\mathrm{RT}}$, a RTOS, specifically FreeRTOS [7], is chosen to run on each PE. FreeRTOS implements fixed-priority scheduling and supports Xilinx FPGAs. The hardware and software architecture explained here is extensively used later throughout case studies and experiments.

1.1.5 Summary

Here, we summarize the key insights that can be drawn from the discussion in previous sections.

From the design requirements point of view, the following are the most significant requirements.

- Embedded streaming applications pose ever increasing throughput requirements.

- Embedded streaming systems require Hard Real Time (HRT) guarantees. Furthermore, it is not uncommon to have both HRT constraints and high throughput requirements at the same time.

From the application characteristics point of view:

- Data Level Parallelism (DLP) and Task Level Parallelism (TLP) are the most important forms of parallelism to exploit, which result in an efficient parallel implementation to achieve high throughput requirements.
- Embedded streaming applications commonly exhibit adaptive behavior in the form of parameter reconfigurations at run-time. This behavior should be explicitly captured in the application specification.

From the architectural perspective:

- An increasing number of Processing Elements (PE) on MPSoC platform is deployed to meet stringent performance requirements. The key question is thus how to utilize them efficiently.
- Network-on-Chip (NoC) emerges and is expected to become the standard communication infrastructure of an MPSoC platform in the near future. A corresponding design methodology is desired to program applications on NoC-based MPSoC platforms to manage communication latency.

From the design methodology perspective:

- Raising the abstraction level to ESL seems inevitable to cope with ever increasing complexity. To fully leverage the benefit of ESL, highly automated tools are needed.
- A central component of an ESL solution is the Model-of-Computation (MoC). Various MoCs, such as (C)SDF, PPN, SADF, PSDF, and VPDF, are extensively adopted to program and/or analyze embedded streaming applications.

1.2 Problem Statement

As motivated in Section 1.1.5, a de-facto solution to the problem of designing complex embedded streaming systems is the adoption of an ESL methodology and highly automated tools. In this thesis, we choose the Daedalus^{RT} design flow as a particular instance. We see several components missing in Daedalus^{RT} to address the requirements outlined in Section 1.1.5 and to efficiently exploit the proper application characteristics and emerging architectural features. Therefore, we address three main problems in this thesis as described below.

We first observe that the current MoC, namely the PPN MoC, used in the Daedalus^{RT} design flow works well as an implementation model. It is possible to efficiently generate code [95] automatically from the PPN MoC for task execution, communication, and synchronization. However, analysis on the PPN MoC, such as for timing guarantees, is rather difficult if not impossible. Both in Daedalus^{RT} and the current industrial practice [90], a more analyzable MoC, such as (C)SDF MoC, is adopted. So far, this analysis model is created manually. However, creating analysis model from an implementation model manually may introduce disparity between both types of models. It is thus hard to guarantee correctness of the analysis model. Based on the discussion above, we formulate the first problem addressed in this thesis: **derive automatically a CSDF graph as the analysis model from an equivalent PPN used as the implementation model**.

Generally, in the Synthesis step shown in Figure 1.5, the traditional DSE approaches like Sesame consider only different mapping and architectural alternatives. With respect to the behavior model, only a single application specification is considered during DSE. This single application specification is normally given by the application designer. Or, the PNgen compiler generates one instance of a PPN that exposes TLP. However, this application specification may not be the most appropriate one for the considered MPSoC platform. That is, the specification may not expose enough parallelism, particularly in the form of DLP, to satisfy the required performance. This is because application designers mainly focus on realizing certain application behavior, including the identification of the functionality of application tasks and the synchronization/communication between these tasks. Moreover, the computational capacity and communication cost of the MPSoC platform are often not taken into account when developing a parallel application specification. In particular, as mentioned in Section 1.1.3, the MPSoC platform is becoming more communication-centric with NoC as the interconnection. As a consequence, overwhelming communication between application tasks may cancel out the expected performance improvement when the application tasks are executed concurrently. Therefore, the second problem addressed in this thesis aims at effectively exploiting DLP in a streaming application. The second problem consists of two sub-problems. We formulate the first sub-problem in the context of Daedalus^{RT} as: **for an initial PPN, investigate an approach to derive an alternative PPN that contains only independent and load-balanced application tasks, if such an alternative PPN exists.**

On the other hand, if more parallelism is revealed than needed when selecting an alternative application specification, it will overload the underlying MPSoC platform.

The overwhelming parallelism leads to an inefficient task allocation. That is, the excessive number of tasks cannot be efficiently allocated and temporally scheduled on the available PEs. Moreover, the excessive number of tasks introduces significant memory overhead for both code and data. When a streaming application is initially modeled using the SDF MoC and requires to meet HRT constraints, we exploit DLP and TLP simultaneously by actor (i.e., tasks) unfolding and transform the initial SDF graph to its equivalent CSDF graph. Therefore, we formulate the second sub-problem in the context of Daedalus^{RT} as: **for an initial SDF graph, derive an alternative CSDF graph that exhibits just-enough parallelism to fully utilize the available PEs, such that HRT constraints are met**.

The third problem addressed in this thesis relates to adaptive application behavior as explained in Section 1.1.2. Such behavior is usually expressed by using parameters whose values need to be reconfigured and updated at run-time. We call such parameters dynamic parameters and their values are not known at design-time. Models such as (C)SDF or PPN used in the Daedalus^{RT} design flow have the limitation of allowing only static parameters. The values of the static parameters are fixed at design-time and they can not be changed at run-time. As a consequence, the adaptive behavior is not amenable to the models such as SDF/CSDF and PPN. Therefore, more expressive MoCs are needed. The MoCs such as BDF and KPN shown in Figure 1.6 provide capability of modeling adaptive application behavior. However, these general MoCs are not analyzable at design-time. Therefore, we are interested in an adaptive MoC which is able to capture adaptive/dynamic behavior in applications while allowing design-time analyzability to some extent. Furthermore, if an adaptive streaming application has HRT requirements, the existing methods lack the ability to efficiently reason about timing behavior based on the chosen adaptive MoC. Moreover, a feasible and efficient way of implementing such an adaptive MoC on MPSoC platforms has not been taken into consideration. Therefore, as the third problem, we **investigate new adaptive MoCs to model adaptive streaming applications and techniques to schedule those adaptive MoCs under HRT constraints**.

1.3 Research Contributions

To address the problems outlined in Section 1.2, this thesis provides several contributions highlighted using the grey boxes in Figure 1.7.

To address the first problem, we develop a step, called *CSDF Derivation*, in this thesis as shown in Figure 1.7. This step primarily contains an **algorithm**, published as a major part of [23], to derive the analysis MoC, i.e., the CSDF MoC, from the implementation model, i.e., the PPN MoC. We present such an algorithm in

Chapter 3. This algorithm is a key enabler of a highly automated design flow, namely Daedalus^{RT} [23], for designing embedded streaming systems with hard real-time constraints. The automated CSDF derivation avoids manual creation of analysis models, thereby greatly improving the productivity of designing such complex embedded streaming systems. Beyond the above-mentioned advantage, automated CSDF derivation can be applied together with other compilation frameworks in which CSDF is adopted as the intermediate representation, e.g., the compilation toolchain [21] for the $\sum$C language, the MAMPS [62] design flow, and the CompSoC [57] framework.

Our second contribution consists of the two *Parallelization* steps shown in Figure 1.7 addressing the second problem stated in Section 1.2. First, we propose in Chapter 4 a parallelization approach next to the PNgen compiler for the PPN MoC, called **communication free partitioning**, published in [138] and [137]. Our approach analytically determines the maximum amount of DLP in the form of a set of communication-free partitions from a given PPN specification. When mapping theses partitions onto different PEs, the communication between PEs is completely eliminated. This parallelization approach is thus highly relevant to emerging NoC-based MPSoC platforms as mentioned in Section 1.1.3, where communication latency may play a significant role on the total execution time of an application. Our approach also can be applied to applications with cyclic dependences, which are traditionally considered as performance bottleneck and hard to parallelize. Second, we propose in Chapter 5 a *Parallelization* step for the SDF MoC, published in [135], to exploit **just-enough parallelism** by task unfolding that fully utilizes the underlying MPSoC platforms, while meeting hard real-time constraints. More specifically, our solution determines simultaneously which SDF actors (i.e., tasks) to unfold by what factor, and the allocation of unfolded actors onto PEs. We show that the solution space of the problem is bounded and derive its upper bounds. We then propose an efficient algorithm to find a solution to the problem, while the obtained solution meets a pre-defined quality.

To address the third problem in Section 1.2, we introduce in Chapter 6 and Chapter 7 two new MoCs, **Parameterized Polyhedral Process Networks** (P^{3}N), published in [136], and **Mode-Aware Data Flow** (MADF), for modeling adaptive streaming applications. We further define the operational semantics of both MoCs, which allows flexible update of parameter values at run-time. In addition, we propose a consistency check approach for P^{3}N, which is applied at both, compile-time and run-time. Based on the P^{3}N semantics, we devise a compile-time approach to extract relations between parameters if they are dependent. This leads to a consistent parameterization of the P^{3}N MoC and moreover, it simplifies the run-time consistency check. The simplification reduces the run-time overhead. Subsequently, we **extend the capability of the hard real-time scheduling framework** used in Daedalus^{RT}

for CSDF to handle MADF. We propose a novel protocol that allows efficient mode transitions, i.e., parameter reconfiguration. As a result, the transition protocol enables us to show an efficient analysis technique to reason about guaranteed timing behavior, particularly during mode transitions.

All contributions mentioned above are implemented either in Daedalus or in Daedalus^{RT}. Furthermore, both Daedalus and Daedalus^{RT} are publicly available [4] for further research. A detailed user manual [24] including an installation guideline and step-by-step tutorial is also available for the benefit of the research community.

1.4 Thesis Organization

The remaining part of this thesis is organized in a self-contained way. That is, every chapter starts with more elaborated introduction and scope of work. More importantly, each chapter has its own related work.

In Chapter 2, we first introduce different MoCs considered in this thesis, particularly (C)SDF and PPN, to better understand our research contributions in later chapters.

In Chapter 3, we present the algorithm to derive the CSDF MoC from its equivalent PPN MoC. The benefit of the proposed algorithm is demonstrated in the context of the Daedalus^{RT} real-time extension.

In Chapter 4, we present the analytical approach to determine the number of communication-free partitions of a PPN. Subsequently, we present the procedure to transform the initial PPN to an alternative PPN that has only set of communication-free partitions, if possible.

In Chapter 5, we present the approach to simultaneously unfold an acyclic SDF graph to its functionally equivalent CSDF graph and allocate all unfolded actors onto PEs, such that HRT constraints are met.

In Chapter 6, we introduce a new adaptive MoC, called Parameterized Polyhedral Process Networks (P^{3}N) and its operational semantics. Subsequently, we show how consistency check can be performed for P^{3}N at compile-time and run-time.

In Chapter 7, we present the hard real-time scheduling approach for another adaptive MoC, which we propose and call Mode-Aware Data Flow (MADF). The approach contains a novel protocol to change scenarios. Based on the protocol, we derive an efficient analysis to reason about timing guarantees, not only within individual scenarios, but also during scenario transitions.

Finally, we conclude this thesis with a summary and some suggestions for future work.