Chapter 7

Summary and Conclusions

STREAMING applications have become prevalent in embedded systems in several application domains, such as image processing, video/audio processing, and digital signal processing. These applications usually have high computational demands and tight timing requirements, such as throughput requirements. To handle the ever-increasing computational demands and satisfy tight timing requirements, Multi-Processor System-on-Chip (MPSoC) has become a standard platform that is widely adopted in the design of embedded streaming systems to benefit from parallel execution. To efficiently exploit the computational capacity of such MPSoCs, however, streaming applications must be expressed primarily in a parallel fashion. To do so, the behavior of streaming applications is usually specified using a parallel Model of Computation (MoC), in which the application is represented as parallel executing and communicating tasks. Although parallel MoCs resolve the problem of explicitly exposing the available parallelism in an application, the design of embedded streaming systems imposes two major challenges: 1) how to execute the application tasks spatially, i.e., task mapping, and temporally, i.e., task scheduling, on an MPSoC platform such that timing requirements are satisfied while making efficient utilization of available resources (e.g, processors, memory, energy, etc.) on the platform, and 2) how to implement and run the mapped and scheduled application tasks on the MPSoC platform. In this thesis, we have addressed several research questions related to the aforementioned challenges in the design of embedded streaming systems. The research questions and the logical connection between them are illustrated in the design flow shown in Figure 1.2. Below, we provide a summary of the presented research work in this thesis along with some conclusions.

To address the first aforementioned challenge in the design of embedded streaming systems, the strictly periodic scheduling (SPS) framework is proposed in [8] which establishes a bridge between the data flow models and the real-time theories, thereby enabling the designers to directly apply the classical hard real-time scheduling theory to applications modeled as *acyclic* CSDF graphs. In Chapter 3, we have extended the SPS framework and have proposed a scheduling framework, namely Generalized Strictly Periodic Scheduling (GSPS), that can handle **cyclic** CSDF graphs. The GSPS framework converts each actor in a cyclic CSDF graph to a real-time periodic task. This conversion enables the utilization of many hard real-time scheduling algorithms that offer properties such as temporal isolation and fast calculation of the number of processors needed to satisfy a throughput requirement. Based on experimental evaluations, using a set of real-life streaming applications, modeled as cyclic CSDF graphs, we conclude that our GSPS framework can deliver an equal or comparable throughput to related scheduling approaches for the majority of the applications, we experimented with. However, enabling the utilization of scheduling algorithms from the classical hard real-time theory on streaming applications by using our GSPS framework comes at the costs of increasing the latency and buffer sizes of the data communication channels for the applications by up to 3.8X and 1.4X when compared with related scheduling approaches.

In Chapter 4, we have addressed the problem of efficiently exploiting the computational capacity of processors when mapping a streaming application, modeled as an acyclic SDF graph, on an MPSoC platform to reduce the number of needed processors under a given throughput requirement. Given the fact that an initial SDF application specification is often not the most suitable one for the given MPSoC platform, we have explored an alternative application specification, using an SDF graph transformation technique, which closely matches the given MPSoC platform. In this regard, in Chapter 4, we have proposed a novel algorithm to find a proper replication factor for each task/actor in an initial SDF application specification such that by distributing the workloads among more parallel task/actor in the obtained transformed graph, the computational capacity of the processors can be efficiently exploited and a smaller number of processors is then required. Based on experimental evaluations, using a set of real-life streaming applications, we conclude that our proposed algorithm can reduce the number of needed processors by up to 7 processors while increasing the memory requirements and application latency by 24.2% and 17.2% on average compared to FFD task mapping heuristic algorithms while satisfying the same throughput requirement. The experimental evaluations also show that our proposed algorithm can still reduce the number of needed processors by up to 2 processors and considerably improve the memory requirements and application latency by up to 31.43% and 44.09% on average compared to the other related approaches while satisfying the same throughput requirement.

As embedded streaming systems operate very often using stand-alone power supply such as batteries, energy efficiency has become an important design requirement of such embedded streaming systems in order to prolong their operational time without replacing/recharging the batteries. In this regard, in Chapter 5, we have addressed the problem of energy-efficient scheduling of streaming applications, modeled as CSDF graphs, with throughput requirements on MPSoC platforms with voltage and frequency scaling (VFS) capability. In particular, we have proposed a novel periodic scheduling approach which switches the execution of streaming applications periodically between a few energy-efficient schedules, referred as modes, at run-time in order to satisfy a given throughput requirement at a long run. Using such specific switching scheme, we can benefit from adopting a dynamic voltage and frequency scaling (DVFS) mechanism to efficiently exploit available idle time in an application schedule. Based on experimental evaluations, using a set of real-life streaming applications, we conclude that our novel scheduling approach can achieve up to 68% energy reduction compared to related approaches depending on the application while satisfying the given throughput requirement.

Finally, in Chapter 6, we have addressed the second aforementioned challenge in the design of embedded streaming systems, namely, how to implement and run a mapped and scheduled adaptive streaming application, modeled and analyzed with the MADF MoC, on an MPSoC platform such that the properties of the analysis model are preserved. In particular, we have proposed a generic parallel implementation and execution approach for adaptive streaming applications modeled with MADF. Our approach can be easily realized on top of existing operating systems while supporting the utilization of a wider range of schedules. We have demonstrated our approach on LITMUS^{RT} which is one of the existing real-time extensions of the Linux kernel. Based on a case study using a real-life adaptive streaming application, we conclude that our approach is practically applicable on a real hardware platform and conforms to the analysis model. In addition, another case study, using a real-life streaming application, has shown that our proposed energy-efficient periodic scheduling approach presented in Chapter 5, which adopts the MOO protocol of the MADF MoC for switching the application mode, is also practically applicable on a real hardware platform by using our generic parallel implementation and execution approach presented in Chapter 6.