# Latency, energy, and schedulability of real-time embedded systems

Liu, D.; Liu D.

# Chapter 1

# Introduction

> Almost all computer systems of the future will utilize real-time scientific principles and technology.
>
> John Stankovic

EMBEDDED systems are computer systems dedicated to a specific functionality. Usually embedded systems are integrated into a large and complex system to control, monitor and assist the operation of the whole system, safely and reliably. In some cases, we may not be aware of the presence of embedded systems, but in our daily life they are prevalent and almost everywhere, affecting and somehow changing our life. From the ARTEMIS report [ART14], 98% of all computing chips are for embedded systems. A smart watch which monitors our health situation is an embedded system; A mouse which we use to control a computer is an embedded system; A thermostat which senses the temperature and humidity of our apartment is also an embedded system. We can give uncountable examples of such embedded systems which are playing an important role in our daily life without notice.

Since embedded systems usually execute control applications which constantly interact with the physical world via sensors and actuators, embedded systems are required to execute not only functionally correctly but also on time. This critical requirement related to time is referred as *real-time constraints* [But11]. Systems are called real-time systems, if the correctness of the system does not only depend on the correctness of the system output but also on whether the output is delivered on time [Sta88]. Based on consequences of missing time deadlines by an application, real-time systems are categorized into two types:

- **Hard-real-time** systems: missing deadlines will lead to failure of the system which in turn causes catastrophic consequences, e.g., loss of human life. Exam-

ples of *hard-real-time* systems are safety control systems in cars and aircrafts, pacemakers, etc;

- **Soft-real-time** systems: missing deadlines will not cause failure of the whole system but will degrade the performance of the system. Examples of *soft-real-time* systems are multimedia applications, on-line service applications, etc.

Embedded systems with real-time constraints are called *real-time embedded systems*.

Besides real-time constraints, many applications of real-time embedded systems feature another important property, called *criticality*. The *criticality* is to denote degrees of importance in guaranteeing the safety of a system. For example, unmanned aerial vehicles (UAVs) have two types of applications, safety-critical applications, such as the flight control, and mission-critical applications, such as surveillance and video streaming. The safety-critical applications (e.g., the flight control) have higher criticality level because they are essentially crucial to the operational safety of the whole system and failure (i.e., violating timing properties) of the safety-critical applications will lead to a catastrophic consequence, such as loss of UAV which may injure a human-being. On the other hand, the mission-critical applications have lower criticality level because they are not coupled to the operational safety of the whole system, so failure of mission-critical applications will not threaten the operational safety of the system but will only affect the system service quality. In different industrial contexts, different standards are deployed to guide the design of systems with different criticality-level applications, such as IEC61508 for electrical/electronic/programmable electronic safety-related systems, ISO26262 for automotive systems, and DO-178B/C for avionic systems [ENNT15]. Table 1.1 defines the classification of criticality levels in standard DO-178B/C [Nor], where 5 criticality levels are present and the criticality levels are classified with respect to the failure consequence on the system safety. Criticality level A is the most critical level and failure of an A-level application leads to catastrophic consequences, whereas failure of an E-level application does not have an effect on the system safety.

Up to this point, we have introduced embedded systems, real-time constraints tightly coupled to embedded systems, and the criticality concept of real-time embedded systems. In the next section, we will discuss three significant development trends in designing real-time embedded systems.

## 1.1 Development Trends in Real-Time Embedded Systems

In the past decade, we have been witnessing some important development trends in the computing world which have profound effects on the design of real-time embedded systems.

| Level | Failure Condition | Failure Consequence |
|---|---|---|
| A | Catastrophic | Failure may cause multiple fatalities, usually with loss of the airplane. |
| B | Hazardous | Failure has a large negative impact on safety or performance, or reduces the ability of the crew to operate the aircraft due to physical distress or a higher workload, or causes serious or fatal injuries among the passengers. |
| C | Major | Failure significantly reduces the safety margin or significantly increases crew workload. May result in passenger discomfort (or even minor injuries). |
| D | Minor | Failure slightly reduces the safety margin or slightly increases crew workload. Examples might include causing passenger inconvenience or a routine flight plan change. |
| E | No Effect | Failure has no impact on safety, aircraft operation, or crew workload. |

Table 1.1: Criticality levels in the DO-178B/C standard [Nor]

### 1.1.1 The Era of Multicore/Multiprocessor Systems

When single processor systems were dominating the chip market a decade ago, chip manufacturers used to improve the performance of a processor by scaling up the operational clock frequency. Meanwhile, the fast development of the process technology enables semiconductor manufacturers to produce thiner *transistors*, the fundamental element to implement electronic circuits. Nowadays, some high-end processors, like Samsung Exynos 7 Octa 7420, 7870 and 8890, are implemented with 14 nanometer transistors [Sam16]. However, constantly scaling up the operational clock frequency of thin transistors results in extremely high power consumption [EET04].

As a solution to such high power consumption, chip manufacturers have drastically changed their design scheme from a single processor chip with high operational clock frequency to a chip with multiple cores/processors, but each with lower operational clock frequency. Fabricating more cores on a chip is able to enhance the peak performance of the system and at the same time to reduce the total power consumption in comparison to the single processor design. Throughout this dissertation, we may use the term multicore and multiprocessor interchangeably.

The year 2004 marked the milestone of this significant change in industry, when Intel canceled its single processor design, namely *Tejas*, and moved to a duel-core

design [EET04]. Since then, computing systems including embedded systems have entered the multicore era. Nowadays, multicore systems are the mainstream in computing systems. This trend can be seen on diverse computing systems such as mobile phones, laptops, desktops, etc.

### 1.1.2 The Shift to Heterogeneous Multicore Systems

Multicore systems have been widely adopted to satisfy the increasing computational demands of complicated applications and, in the meantime, to reduce energy consumption. Among all multicore systems, homogeneous multicore systems that consist of identical processing cores are most ubiquitous and widely-used in modern electronic systems spanning from mobile devices to supercomputing systems. However, the rapid development of multicore systems brings a new problem, called the *dark silicon problem* [EBSA+11]. In 1974, Dennard *et al.* [DGR+74] stated that as transistors decrease size, the power density still remains a constant, i.e., the transistors become thinner, and the power consumption also scales down along with the reduced size. This statement is wildly known as the "*Dennard Scaling*". However, when the transistor manufacturing technology enters the era of nanometer, the *Dennard Scaling* fails due to the dramatically increased static power consumption in nanometer-size transistors. Static power is consumed by currents which leak through transistors even when transistors are turned off [KAB+03]. This significant increase in static power consumption in turns leads to an overheating issue for the system. To avoid the overheating, some transistors on a chip have to be inactive (powered-off), i.e., *'dark'*.

Several solutions [CZZ+15, HKPS15] have been proposed in recent years to mitigate the dark silicon problem. Heterogeneous multicore systems [Mit15] have been considered to be one of the promising solutions for the dark silicon problem and a good alternative to homogeneous multicore systems. In contrast to homogeneous systems having identical cores, heterogeneous multicore systems consist of different types of cores. Such variety of cores enable diverse applications to enhance the application performance and/or reduce the power/energy consumption of the application by means of selecting a proper core for execution.

Among all heterogeneous systems, the single-ISA heterogeneous multicore system [KFJ+03] is a special type of heterogeneous multicore system, where the cores on the chip have the same *instruction set architecture* (ISA) but differentiate with each other in terms of power consumption and performance. Typical single-ISA heterogeneous multicore systems usually consist of two types of cores; 'big' cores with complex micro-architecture, e.g., a deep pipeline and wider issue width, designed for high performance computing and 'LITTLE' cores with simple micro-architecture, e.g., a shallow pipeline and narrower issue width, optimized for low power computing. Table 1.2 shows an example of cores implemented on a 'big.LITTLE' architecture

| Core type | pipeline depth | Out-of-order execution | Decode | big.LITTLE role |
|---|---|---|---|---|
| ARM Cortex A57 | 15 | Yes | 3-wide-issue | 'big' |
| ARM Cortex A53 | 8 | No | 2-wide-issue | 'LITTLE' |

Table 1.2: Comparison between ARM Cortex A57 and ARM Cortex A53 [ARM16]

system and gives a comparison of the two types of ARM cores in terms of microarchitecture [ARM16]. Several leading semiconductor companies have mass-produced their own single-ISA heterogeneous multicore systems for commodity products, e.g., Qualcomm Snapdargon 810 and 808, Samsung's Exynos 5 Octa series [Sam16], and Nvidia's Tegra X1[Gil15]. In the remainder of this dissertation, when we refer to heterogeneous multicore/multiprocessor systems, we mean single-ISA heterogeneous multicore/multiprocessor systems.

### 1.1.3 The Emergence of Mixed-Criticality Systems

Real-time systems which execute applications with different criticality are becoming prevalent, e.g, automotive vehicles, unmanned aerial vehicles, aircrafts, etc. To ensure the safety guarantee of systems with different critical-level applications, the old paradigm of designing such safety-critical systems was to physically isolate applications with different criticality level, i.e., critical applications and non-critical applications are executed separately on different processing units. Such complete spatial isolation enables critical applications to avoid the interference from non-critical applications, thereby guaranteeing system safety. However, with the rapid development of complex and sophisticated real-time systems, increasing number of applications with different criticality and complex functionality are incorporated into a system, leading to a huge growing number of processing units. For instance, modern premium cars typical contain around 70-100 computers, around 100 electronic motors and 2 km of wire [Tho12]. This complicated and sometimes redundant hardware leads to a system with large system size and very high power consumption. Therefore, to reduce Size, Weight, and Power (SWaP), the emerging trend in the development of safety-critical systems is to integrate applications with different criticality into a shared computing platform. We call such systems *mixed-criticality systems*. A formal definition of a *mixed-criticality system* is given as follows:

**Definition 1.1.1** ([BBB+09])**.** A mixed-criticality system is an integrated suite of hardware, operating system and middleware services, and application software that supports the execution of safety-critical, mission-critical, and non-critical software within a single, secure compute platform.

## 1.2 Problem Statement

The important development trends, described in Section 1.1, bring new opportunities to develop embedded systems, but they also arise several challenges when designing real-time embedded systems. In this dissertation, we address challenges arisen by the above-mentioned development trends in the contexts of system resource optimization, system energy optimization, and system schedulability analysis. The specific problems, we address in this dissertation, are formulated as follows.

**Problem 1: Resource Optimization for hard-real-time streaming applications.**

Streaming applications, such as video/audio processing and digital signal processing, have become prevalent in embedded systems. These applications contain ample amount of parallelism which perfectly matches the processing power of Multi-Processor System-on-Chip (MPSoC) platforms. To efficiently program MPSoC platforms, Models-of-Computation (MoCs) are usually used to specify streaming applications. Prominent examples of MoCs include Synchronous Data Flow (SDF) [LM87] and its generalization Cyclo-Static Dataflow (CSDF) [BELP96], in which actors representing computation are executed concurrently, thereby naturally exposing parallelism.

Traditionally, self-timed scheduling [MB07] is considered to be the most proper scheduling paradigm to schedule Data-Flow modeled applications. However, hard-real-time constraints are increasingly imposed to streaming applications and self-timed scheduling cannot guarantee such rigorous constraints. In addition, self-timed scheduling suffers from complex analysis techniques, making its design procedure really time-consuming. Recently, Bamakhrama and Stefanov in [BS11][BS12][BS13] proposed a scheduling framework that schedules acyclic CSDF graphs by using hard-real-time theories. In this scheduling framework, each CSDF actor executes strictly periodically and meets a given deadline. The periodic execution of actors guarantees a certain throughput and latency. Additionally the well-defined analytical techniques of real-time theories significantly reduce the design time when designing embedded multiprocessor streaming systems [BZNS12]. When CSDF actors are scheduled as strictly periodic tasks, the deadline of each actor can be varied in a well-defined bounded interval (see Section 2.3), thereby controlling the application latency and the number of processors needed to schedule the application. *This means that selecting a proper deadline value for each actor is an important issue for reducing the latency and minimizing the number of processors in this framework.* Although, in [BS12], the authors give a method to select deadlines of actors to reduce the application latency, their method is not optimal in terms of the required number of processors. **The problem, we address in the scheduling framework proposed in [BS11][BS12][BS13], is how to select deadlines of actors of hard-real-time streaming applications in**

**a proper way such that the resources (the number of processors) is minimized while meeting their latency constraints.**

**Problem 2: Energy-efficient mapping and scheduling of hard-real-time applications on "big.LITTLE" heterogeneous multicore systems.**

Energy/power consumption has gradually become a critical design criterion for a system, especially for embedded systems which are mostly battery-powered. Voltage/frequency scaling (VFS) is the most common technique for power reduction and can be seen on many modern processors. Due to its prevalence, VFS is also applied to real-time embedded systems for energy minimization.

Basically, for energy-efficient real-time application mapping, an algorithm with consideration of energy efficiency is deployed first to map real-time applications on multicore systems and then the VFS technique is used on the system to scale down the operational clock frequency/voltage to a proper level that is able to guarantee the time deadlines of all real-time applications and to minimize the energy consumption at the same time. However, the energy-efficient mapping and scheduling problem has been proven to be $\mathbf{NP}$-hard in the strong sense on homogeneous multiprocessor systems [AY03] as well as on heterogeneous multiprocessor systems [CT08]. Thus, heuristic or approximate algorithms are required to deal with the problem in a reasonable time. Many approaches have been proposed to effectively and efficiently map real-time applications in an energy-efficient manner. Two surveys in [CK07, BMAB16] comprehensively review existing works concerning energy-efficient mapping and scheduling of real-time applications.

However, the existing approaches cannot effectively handle the new heterogeneous "big.LITTLE" multicore systems discussed in Section 1.1.2, because some important features of these emerging heterogeneous multicore systems were not considered. Hence, this fact motivates us to revisit the existing mapping and scheduling approaches and consider the features of the new heterogeneous multicore systems. **The problem, we address, is how to map hard-real-time applications on the emerging "big.LITTLE" heterogeneous multicore systems in an energy-efficient manner while satisfying real-time constraints and performance requirement when considering hard-real-time streaming applications.**

**Problem 3: Schedulability of imprecise mixed-criticality systems.**

To ensure the correctness of a mixed-criticality (MC) system, highly critical applications are subject to certification by Certification Authorities (CAs), such as the Civil Aviation Authority [Civ16] and Federal Aviation Administration [Fed16], and usually the certifications are done under extremely rigorous and pessimistic assumptions [Ves07]. As a consequence, this pessimism generally causes large worst-case execution time (WCET) over-estimation for highly critical applications and in turn it

results in underutilization of the hardware resource.

To deal with this overestimation, Vestal proposed in [Ves07] to characterize a highly critical application with different WCETs corresponding to different criticality levels. Besides the large WCET determined by the CAs, each highly critical application is specified with several smaller WCETs which are determined by system designers at lower assurance levels, i.e., considering less pessimistic situations. Since CAs only certify highly critical applications, all less critical applications are only validated by system designers who generally do this under less pessimistic situations, thereby having only lower assurance and smaller WCETs. When scheduling an MC system modeled as described above, all applications (highly critical and lowly critical) are initially scheduled using their low assurance WCETs. This can better utilize hardware resources, and in most cases all applications can be safely and successfully scheduled with their low assurance WCETs. Then, if a rare case occurs, i.e., any highly critical application cannot complete its execution within its low assurance WCET, the system discards all less critical applications and dedicates the whole system to schedule only highly critical applications with their certified (very pessimistic, large) WCETs. Throughout this dissertation, we call the MC model discussed above the *classical* MC model.

Although the *classical* MC model captures the core features of MC systems, it also receives some criticisms from system engineers because completely discarding less critical applications is too pessimistic and in some cases unacceptable [BB13]. To address these criticisms, Burns and Baruah in [BB13] proposed a more general MC model. In this general MC model, besides the normal WCET validated by system designers, a reduced WCET is given to each less critical application. Then, if a rare situation occurs, i.e., highly critical applications overrun their low assurance WCETs, instead of discarding all less critical applications, this general MC model keeps less critical applications running with their reduced WCETs. Reducing WCETs to keep less critical applications running is conceptually similar to the *imprecise computation model* [LLS$^+$91][LSL$^+$94]. In the *imprecise computation model*, the output quality of a real-time application depends on its execution time. The longer a task executes, the better quality results it produces. Then, if there is an overload in the system, tasks can trade off the quality of the produced results to ensure their timing correctness. In [RKKK14a], Ravindran et al. give several real-life applications with this imprecise feature in different domains, e.g., video encoding, robotic control, cyber-physical systems, and planetary rovers. Considering the conceptual analogy between the general MC model proposed in [BB13] and the *imprecise model*, we call this general MC model *imprecise mixed-criticality* (IMC) model in this dissertation.

For MC real-time systems or even non-MC real-time systems (all applications have the same criticality level), the most important problem is to analyze the *schedu-*

*lability* (i.e., feasibility) of the system, i.e., whether under a certain scheduling algorithm, a set of real-time applications can run on a platform without violating any deadline, even in the worst case. To analyze the *schedulability* of a real-time system under a certain scheduling algorithm, a *schedulability test* is needed.

**Definition 1.2.1.** Given a scheduling algorithm, a hardware platform and a real-time application set, a *schedulability test* decides whether the application set is schedulable by the scheduling algorithm on the hardware platform.

In [BB13], Burns and Baruah presented a test based on Adaptive Mixed-Criticality (AMC) [BBD11] to check the schedulability of the IMC model under the fixed-priority scheduling algorithm [LL73]. However, a schedulability test of the IMC model under dynamic-priority scheduling algorithm, e.g., earliest deadline first (EDF) with virtual deadline (EDF-VD) [BBD+12], has not been addressed. Here, **the problem, we address, is how to ensure and test the schedulability of an IMC system under the EDF-VD scheduling algorithm.**

## 1.3 Contributions of This Dissertation

Below, we summarize our novel contributions to the problems outlined in Section 1.2.

**Contribution 1: Novel approach for Resource Optimization of CSDF-modeled Streaming Applications with Latency Constraints**
To address Problem 1 in Section 1.2, in the context of CSDF-modeled streaming applications and the *hard-real-time scheduling framework* proposed in [BS11][BS12] [BS13], we propose a new method to optimally select the deadlines of actors in CSDF graphs so that the required resources (i.e, the number of processors) are minimized while the latency requirement is satisfied. Our novel contributions are twofold: 1) we propose a new method to interpret the precedence relation between actors so that the parameters of actors, i.e., starting times and deadlines, can be formalized in a mathematical form; 2) based on our first contribution, we formulate our resource optimization problem as an *integer convex programing* (ICP) problem. Convex programming is a mathematical optimization problem which can be optimally solved in polynomial time [BV04]. The formulated ICP problem enables us to use an off-the-shelf convex programming solver, e.g., CVX [GB14][GB08] to solve our problem and obtain the optimal solution for our resource optimization problem. Compared to the existing approach [BS12], our ICP-based approach can effectively reduce the resource requirements for the hard-real-time streaming applications while guaranteeing the latency constraints. This contribution and experimental results are presented in Chapter 3.

**Contribution 2: Novel Algorithm for Energy-Efficient Mapping of Real-Time Streaming Applications on Heterogeneous Multiprocessor System-on-Chip (MPSoC)**

We address Problem 2 in Section 1.2 by proposing a novel polynomial time algorithm, called Frequency Driven Mapping (FDM), to map real-time streaming applications onto a heterogeneous multiprocessor system with the aim of reducing the energy consumption and guaranteeing the latency and throughput constraints. The main novelty in this algorithm is twofold: 1) By using the *hard-real-time scheduling framework* for CSDF graphs in [BS11][BS12][BS13], we propose an efficient way to determine a suitable processor type for each actor/task in the CSDF graph, where the energy consumption is minimized and throughput and latency constraints are met; 2) Then, based on the obtained processor type assignment, we propose a new approach to map actors of the streaming application specified as a CSDF graph to the given platform and then further reduce the energy consumption by using VFS. Experimental results show the effectiveness of our proposed algorithm in terms of energy efficiency in comparison to the related work. This contribution and experimental results are presented in Chapter 4.

**Contribution 3: Novel Algorithm for Energy-Efficient Mapping of Real-Time Tasks on Heterogeneous Multicores Using Task Splitting**

To address Problem 2 in Section 1.2, in the context of independent real-time tasks (applications), we are inspired by the latest C=D task-splitting technique [BDWZ12] and propose a novel algorithm, called *Allocation and Split on Heterogeneous Multicore systems* (ASHM), to energy-efficiently map real-time tasks on heterogeneous multicore systems by using the C=D task-splitting technique. To the best of our knowledge, our ASHM algorithm is the first work to consider the C=D task-splitting technique on heterogeneous multicore systems for energy efficiency. In this work, we investigate the application of the C=D task-splitting on real-time heterogeneous multicore systems to reduce energy consumption. The concepts regarding the task-splitting approach and the C=D approach will be explained later in Section 5.2.4. We analyze the properties of the C=D task-splitting and extend it for heterogeneous multicore systems. The analysis and extension of the C=D task-splitting on heterogeneous multicore systems serves as the foundation of the proposed ASHM algorithm. Experimental results show that our ASHM outperforms the existing approaches in terms of energy efficiency. This contribution and experimental results are presented in Chapter 5.

**Contribution 4: The first Schedulability Test for Imprecise Mixed-Criticality Systems Under EDF-VD Scheduling**

The last contribution in this dissertation addresses Problem 3 discussed in Section 1.2. We propose the first test to check the schedulability of the IMC model under the

scheduling and prove the correctness of the proposed schedulability test. A brief introduction of the EDF-VD scheduling algorithm is given in Section 6.2.3. Moreover, with the proposed schedulability test, a special metric for scheduling algorithms, namely *speedup factor*, is used to quantify the optimality of the IMC model under EDF-VD scheduling algorithm. The experimental results show that EDF-VD can schedule more IMC task sets than the existing approach AMC [BB13]. This contribution and its experimental results are presented in Chapter 6.

## 1.4 Dissertation Outline

The remainder of this dissertation is organized in a self-contained manner. **Chapter 2** introduces some common, fundamental, and relevant knowledge about the data-flow model, considered in this dissertation, and the real-time system models and their corresponding analysis techniques that are deployed in this dissertation are also presented in order to facilitate the understanding of the contributions afterwards.

**Chapter 3 - 6** present in details the dissertation contributions briefly introduced in Section 1.3. Each chapter is organized in a self-contained way. That is, each chapter has its specific

- brief introduction and detailed contributions;
- related work and a system model;
- proposed algorithm/approach; and
- experimental results.

Finally, **Chapter 7** summarizes this dissertation and points out some directions which deserve further investigation. The detailed organization of this dissertation is as follows:

1. **Chapter 2** presents some common background information pertaining to the CSDF model, real-time systems, hard-real-time scheduling framework proposed in [BS11][BS12][BS13].
2. **Chapter 3** presents the new approach to optimize resource requirements of hard-real-time streaming applications subject to latency constraints in the scheduling framework proposed in [BS11][BS12][BS13].
3. **Chapter 4** presents the FDM algorithm to energy-efficiently map hard-real-time streaming applications onto cluster heterogeneous multicore systems.

4. **Chapter 5** presents the ASHM algorithm to energy-efficiently map real-time tasks onto heterogeneous multicore systems by using the task-splitting technique.

5. **Chapter 6** presents the schedulability test for the IMC model under EDF-VD and the proof of the speed-up factor.

6. **Chapter 7** summarizes the dissertation and discusses possible future work.