A CONTROL-THEORETIC DESIGN AND ANALYSIS FRAMEWORK FOR RESILIENT HARD REAL-TIME SYSTEMS

by

PRADEEP M. HETTIARACHCHI

DISSERTATION

Submitted to the Graduate School

of Wayne State University,

Detroit, Michigan

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

2015

MAJOR: COMPUTER SCIENCE

Advisor Date

ACKNOWLEDGEMENTS

My journey of PhD would not be a success without a few key people.

Prof. Nathan Fisher, my PhD advisor who did not restrict his guidance on academics but extended to a great deal helping me as a person. I am very fortunate and blessed to have him as my advisor. The difference I see in my academic life from the time I joined the PhD program to date is mainly because of the constructive guidance given by Dr. Fisher. I managed to stand on my feet and continue my graduate studies while facing substantial life challenges during the last five years mainly because my advisor's kind support. I convey my heartfelt gratitude and sincere thanks to Dr. Fisher.

During this challenging exercise, I got continuous support and advice from Dr. Le Yi Wang from Electrical Engineering Department. Also, comments and suggestions given by Dr. Weisong Shi and Dr. Monica Brockmeyer were extremely valuable. Furthermore, my CoPaRTS lab mates were always a critical factor for my success. I am very thankful to all of them.

This dissertation was supported by an NSF CAREER Grant (CNS-0953583), an NSF CSR Grant (CNS-1116787), an NSF CRI Grant (CNS-12-5338), and a grant from Wayne State University Office of Vice President of Research. Also, I must thankful for the Wayne State University Computer Science Department for accommodating and facilitating me for six year.

I am always grateful to the key persons who encouraged me to peruse graduate studies and opened the path in the United States, Eng. Samarasiri and Prof. Herath. Furthermore my loving appreciation to my parents, in-laws and sisters who helped and always encouraged me to achieve this goal. My special thanks to Malitha, my wife, and children, Induvari, Vidushani, Dushyanth, and Ravindu who partnered with me in this journey for always being patient, supportive and encouraging.

Finally, I thank anyone if I missed to mention here.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

1.1 Overview

1.1.1 Real-time Systems

Real-time systems include the notions of both *logical* and *temporal correctness* [7]. There are two main type of real-time systems: the soft-real-time systems, such as video decoding, can *tolerate occasional deadline misses;* the hard-real-time systems, such as avionics and radar control systems, can not tolerate deadline misses and have *strict temporal requirements*. Real-time systems work as a functional manager in consumer electronics and cyber-physical systems. Today's standard computing platform [40, 41]–multicore processors–are often the underlying computing architecture of these systems. Furthermore, their correct functionality upon various conditions is extremely important. For example, in the automobile industry, the modern vehicle braking system consists of hard real-time computer and electro mechanical actuators; the accuracy of the vehicle braking action in various environmental conditions determines the safety of the driver. Therefore, the system designer should assure the correct functionality of the real-time computer and mechanical parts for all the operating conditions of the system.

1.1.2 Motivation

Today, multicore processors with advanced capabilities and high clock frequencies are common [60, 38]. Also, many manufacturers offer complete embedded computers, which include a powerful processor, I/O interface, and communication interfaces in a single compact printed circuit board [61]. These modern embedded computers are ideal for contemporary hard-real-time system development as they have higher computing capabilities and reliability, and are less expensive. Despite these advantages, modern real-time systems face many physical challenges and hardware constraints at design and during runtime. For example, the excessive processor thermal dissipation and high power consumption [3] due to higher clock rates, high leakage current [52], and increased chip package density–dark silicon [82, 25] issues–are common problems in modern microprocessor based systems. As a solution to the excess thermal dissipation and larger power consumption, these real-time systems may set to operate at a lower clock frequency or dynamically adjust the processor clock frequency [84, 85]. However, lower clock frequency degrades the performance of the system, arbitrary clock frequency reduction may affect the underlying real-time system and fail to achieve its essential task deadlines. Therefore, the system designer needs to analyze multiple system constraints towards any remedial solution (a tradeoff) for excessive thermal dissipation and should choose the best possible solution. Furthermore, as an example of an embedded system where thermal-stress analysis is essential, consider microprocessors found in implantable medical devices (IMDs). IMDs are increasingly being used to treat various diseases and medical conditions (e.g., pacemakers for heart disease or neural implants to restore hearing/vision). However, recent studies [53, 55] have shown that the heat dissipated from IMDs due to the microprocessor activity is non-negligible. Thus, designing IMDs with minimum thermal dissipation is critical as medical research has shown that a temperature increase of even $1°C$ can have long-term effect on tissue [54] and, in the extreme, death may even result from excessive tissue heating [70]. Complicating the safe thermal design of IMDs, body temperature naturally fluctuates over time and varies depending on location [50]. An IMD designer must balance (under temperature fluctuations) the real-time computational requirements of the device with the non-harmful thermal operating limits. In the presence of an increased surrounding temperature, an IMD will have to reduce its computational load to prevent tissue damage due to heat[1]. However, as the correct and safe functioning of the IMD is an absolute requirement, the system designer requires techniques to formally verify the effect of different body temperatures on the correct operation of the IMD. Similarly, as a less safety-critical example, consider how the quality of audio/video decoding may degrade in a hand-held device as the system reacts to increases in temperature by reducing computational processing (e.g., via instruction fetch toggling). Ideally, a system designer would like to determine how much the performance will degrade under different thermal operating conditions.

Until the last decade, real-time systems were mainly constrained by the CPU processing power, as the CPU processing power was the most scarce resource. Also, the cost of real-time hardware was substantially high, compared to the cost of software design and testing altogether [76]. For the past couple of years, the cost per tera flop has been reduced exponentially [51]. By now, the modern real-time system manufacturers face different challenges; today, real-time software engineering has grown as a different discipline [18]. The real-time software design, development, and testing cost is the highest portion of the

[1]As IMD microprocessors typically do not have DVS capabilities, an IMD may have to reduce non-essential tasks such as communication with other nodes in a body-area network [83].

system development cost due to its size and complex nature. For example, resource, energy, and thermal efficient real-time multicore processor systems design and development process are hard and need many design hours. Lack of proper frameworks and the difficultly involved in streamlining the real-time design process can explain this higher cost. Furthermore, real-time system designers have to concentrate more on the software design and the implementation issues, when systems experience physical and hardware constraints. Thus, designing a reliable system (provisioning) for different hardware and physical constraints has become a central issue in real-time system design paradigm. Furthermore, the optimized CPU resources allocation on each task on a real-time task system is fundamentally hard question. Also, under various physical constraints, the real-time system design needs to gracefully degrade the quality of service, without interfering the essential real-time services on the system.

Modern real-time systems operate on multi-inputs and multi-output environments. They need to adapt to various physical and logical constraints that are dynamic in nature. These systems need to dynamically allocate the resources for the essential activities at a given time. Over the years, control-theoretic based methods have shown to be the best, as they have the capability to analyze and control multiple inputs and outputs precisely at run-time.

There are many important previous results on thermal-aware and power-aware real-time system designs (see Chapter 2). But, none of them addresses the issues of hard-real-time performance guarantee upon multiple unpredictable physical and hardware-constraints for multi-modal system. Therefore, in this thesis, we try to fill this gap. We propose a generalized control-theoretic system-resiliency framework.

In the rest of this chapter, we first discuss the objectives as well as the contributions of our work. Then we will outline the rest of the thesis in the end of this chapter.

1.2 Objectives

In this thesis, we introduce a new metric called *system-resiliency* which characterizes the maximum prior unknown unpredictable external stresses that any hard-real-time performance mode can withstand. Our proposed system-resiliency framework addresses resiliency determination for real-time systems with physical and hardware limitations. Furthermore, our framework advises the system designer about the feasible trade-offs between different system resources. For example, in a solar-powered device, a sunny day benefits the device's battery charging-level, but may negatively affect the performance of the CPU (by raising the environmental temperature and forcing the CPU to run a lower power level to reduce thermal dissipation). Therefore, finding a balanced trade-off between these two external factors for a better system design is essential.

The runtime efficiency is also an important factor to consider during the design process. Therefore, we study the effective scheduling of real-time tasks with intra-task parallelism to reduce the energy-consumption of multiprocessor platforms. Furthermore, we investigate the potential temperature rise of a uniprocessor due to task execution. We show the thermal effect of periodic resource due to different resource capacity and investigate the influences of the peak-temperature of a uniprocessor.

The concept we introduce as system-resiliency closely resembles the stress test in materials engineering. Thus, our design framework and analysis may be classified as a *system-stress-analysis* for real-time systems.

1.2.1 Thesis

The thesis of this document is:

> Our newly introduced metric, *system-resiliency* defines the maximum external stresses that any hard-real-time system mode can withstand before violating timing constraints. A carefully designed control-theoretic framework effectively facilitates the ability of the system designers to quantify the system-resiliency. Furthermore, we consider effective scheduling of real-time tasks with intra-task parallelism to reduce the energy consumption of multiprocessor platforms, to maintain the energy-efficiency of the system design.

1.3 Summary of Contributions

The main contributions of this thesis are listed as the follows:

1. As a proof of concept, we propose a subset of system-resiliency, a new metric, called *thermal-resiliency* which characterizes the maximum external thermal stress that any hard-real-time performance mode can withstand (see Chapter 4). We show how to solve some of the issues and challenges of designing predictable real-time systems that guarantee hard deadlines even under transitions between modes in an unpredictable thermal environment where environmental temperature may dy-

namically change, using our new metric. In our framework, the system designer specifies a set of hard-real-time performance modes under which the system may operate automatically adjusts the real-time performance mode based on the external thermal stress.

2. We extend the derivation of thermal-resiliency, that was proposed for uniprocessor systems to multicore systems and determines the limitations of external thermal stress that any hard-real-time performance mode can withstand (see Chapter 5). Our control-theoretic framework allows system designer to allocate asymmetric processing resource upon a multicore processor and still maintain thermal constraints.

3. We introduce the *generalized system-resiliency* (GSR) design framework that predictably quantifies the real-time performance level attainable under unpredictable dynamic external conditions (see Chapter 6). We show how the designer derives the optimal supported-modes for a multi-constrained system under various processing resources allocations while maintaining maximum system constraints.

4. The fine-grain power and energy consumption of a task system is essential for balanced and optimized task partitioning on a multicore processer. Therefore, we investigate the potential utility of parallelization for meeting real-time constraints and minimizing energy. We consider malleable Gang scheduling of implicit-deadline sporadic tasks upon multiprocessors.

5. Verifying theoretical results is essential in real-time system design. Therefore, we implemented a testbed to verify our theoretical results upon the testbed runs (see Appendix A and Appendix B).

1.4 Organization

The following table gives the details of each chapter:

Table 1.1: The Contribution of each Chapter

Chapter #	Contribution
Chapter 3	Models and definitions used in rest of the thesis
Chapter 4	Thermal-resiliency calculation details on uniprocessor systems
Chapter 5	Thermal-resiliency calculation details on multicore processor systems
Chapter 6	Generalized system-resiliency calculation details
Chapter 7	Conclusion and future work beyond the final dissertation of this thesis
Appendix A	Details on practical implementation, such as temperature calculation, control parameter derivations, and system-identification details
Appendix B	Further details on multicore processor model and multicore processor testbed details

CHAPTER 2: RELATED WORK

In this chapter, we present prior research on thermal and power-aware real-time systems design techniques. Also, we briefly present some of the works related the energy estimation on parallizable workloads and the maximum temperature estimation of a uniprocessor upon real-time task execution.

2.1 Thermal-Aware Design for Uniprocessor Systems

In this thesis, we introduce the system-resiliency framework that addresses resiliency determination for real-time systems with physical and hardware constraints. Toward this final goal, first we present the thermal-resiliency framework for uniprocessor in Chapter 4 and the multiprocessor thermal-resiliency framework details in Chapter 5. Furthermore, to properly understand the importance of our results (explained in the Chapter 4 and Chapter 5), in this subsection, we give a brief, high-level overview of previous research in both general (non-real-time), thermal-aware system design and real-time-specific thermal-aware design.

2.1.1 General (Non-Real-time) Thermal-Aware System Design

For non-real-time systems, Brooks and Martonosi ([12]) investigated major components of any dynamic thermal management scheme and suggested policies and mechanisms for implementing dynamic thermal management for current and future high-end CPUs. They evaluated the benefits of using dynamic thermal management to reduce the cooling system costs of CPUs and developed an architectural-level power modeling tool called Wattch. For the micro-architecture level of thermal modeling, Skadron et al. ([78]) proposed a compact, dynamic, and portable thermal model and a tool called *HotSpot* for use at the architecture level for micro-architectures.

2.1.2 Real-Time Thermal-Aware System Design

For real-time systems in the online setting, Bansal and Pruhs [6] explored algorithms for minimizing both peak-temperature and energy efficiency for online jobs with deadline constraints. In the off-line setting, previous work on scheduling under thermal constraints has followed two main approaches: reactive and proactive schedulers. In a reactive scheduler, the processor speed is reduced in response to a thermal trigger. Wang et al. [89] studied schedulability analysis under the reactive setting. In the proactive setting, the speed schedule for the processor is determined at design time. Chen et al. [16] addressed proactive scheduling for the periodic task model. Quan and Zhang [68] consider feasibility analysis of leakage-aware periodic tasks under temperature constraints. However, previous work on both settings assumed either simple task models or the existence of “ideal” processor speeds. Also, they have not addressed the issues related to multicore processor based systems. Recent dynamic temperature management strategies also exist for multiprocessor real-time systems [15, 13, 31]; however, most of these focus upon static speed-assignment approaches and not a proactive schedule. Thermal analysis has also been studied in the context of web servers [26], but hard deadlines are not guaranteed. The work by Y. Fu et al. [37] and X. Fu et al. [36] address handling unpredictable thermal events; however, the results do not provide any *a priori* guarantees that may be used to equate real-time performance and thermal resiliency.

2.2 Thermal-Aware Design for Multicore Processor Systems

Multicore processor-based systems that operate under external temperature constraints are an important focus in real-time systems research. In this subsection, we overview the prominent work and previous research on thermal-aware, real-time thermal-aware system design, and multicore real-time systems at high-level. Furthermore, we discuss the work on scheduling under thermal constraints, under two main approaches: reactive and proactive schedulers. Although there are many scattered individual works that address the thermal-aware design, power-aware design, and real-time multicore processor systems design, no prior work suggested a generalized design framework for graceful degradation of quality of service of a real-time system.

2.2.1 Control-Theoretic Based Designs

Ghosh et al. [42] proposed a framework for mapping, the level of service and resource requirements for dynamic environmental conditions. They presented an integrated QoS optimization, which is performed using Q-RAM [69]. Lu et al. [62] proposed adaptive utilization based multi-processors real-time design for constrained MIMO systems. Fu et al. [34] proposed a control based solution for simultaneous thermal and timeliness guarantees for distributed real-time embedded systems running in unpredictable environments.

In contrast to these work, that rely on mapping techniques to adopt the varying environmental conditions, we provide the system designer real-time and performance guarantee.

Yao et al. [93] discussed an online adaptive mechanism, based on the utilization control, for multiprocessor real-time settings with online system identification and LQ controllers for a system with multiple constraints. Fu et al. [33] suggested a solution, that integrates core-level feedback control with processor-level optimization to minimize dynamic and leakage power consumption of a multi-core real-time embedded system. Theis research may offer accurate control solutions suitable for soft real-time systems that needs to adjust the utilization by means of task rate adjustments at runtime. However, their methods have limited applicability for systems that need hard real-time and multi-mode capabilities. Seo et al. [71] studied energy-efficient multicore real-time scheduling using processor-wide DVFS; however, their results do not provide any hard real-time or performance guarantee. Further, each of these prior results do not provide a mechanism to specify the graceful degradation of the system's operating modes in an unfavorable environment.

2.2.2 Proactive Methods

In this subsection, we give the previous work on thermal-aware real-time systems design focused on proactive scheduling based techniques. In a reactive scheduler, the processor speed is reduced in response to a thermal trigger (e.g.,the maximum system operating temperature is reached). In the proactive setting, the speed schedule for the processor is determined at design time. Chantem et al. [14] made an interesting observation for maximizing work load under thermal constraints. While working with proactive scheduling, the authors show that the scheduler which maximizes workload under thermal constraints must be a periodic. The authors determined a speed schedule such that the peak temperature constraints were met and total work completed was maximized using a DVFS control policy for processors with discrete speed levels.

Wang et al. [87, 88, 89] studied temperature-constrained real-time systems and performed schedulability analysis of a task system. They computed upper bound on the worst-case delay for tasks with arbitrary job arrivals for both first-in-first-out (FIFO) and static-priority (SP) scheduling algorithms, and also showed that this simple reactive speed control decreased the response time of tasks compared with any constant-speed scheme. However, the previous aforementioned work on proactive and reactive schemes assumes either simple task models or the existence of "ideal" processor speeds which may not be feasible even by using the recent top-of-the-line microprocessors.

There has been extensive research work on finding an optimal feasible speed for minimizing energy consumption as well as temperature minimization. Finding an optimal speed is always possible under the assumption that the processor can run at any speed. Yao et al. [92]. formulated speed scaling problems as scheduling problems to minimize energy consumption. The authors developed an optimal off-line greedy polynomial-time algorithm, and also two online algorithms. Bansal et al. [6] explored algorithms for minimizing both peak temperature and energy efficiency, and proposed an algorithm with constant approximation ratio with respect to the optimal algorithm for managing temperature. Hung et al. [49]. investigated both power-aware and thermal-aware approaches and found a power-aware approach alone is not able to address the temperature challenge; furthermore, many low-power techniques have insufficient impact on chip temperature because they do not directly target the spatial and temporal behavior of the operating temperature. DTM strategies also exist for multiprocessor real-time systems [15, 14, 31]; however, most of these focus upon static speed-assignment approaches and not on a proactive schedule. Real-time thermal analysis has been studied in the context of web servers [26], but hard deadlines are not guaranteed. Real-time scheduling under thermal constraints has also been evaluated for multi-function phased array radar systems via energy minimization [43, 42]. Numerous results exist for the periodic resource model without thermal settings. Shin and Lee [73, 74] obtained linear-time algorithms for determining the capacity of aperiodic resource (given a periodic task system); however, the allocation may potentially over-provision the amount of capacity needed by the task system leading to wasted processor resources. Easwaran [23, 24] devised an exact algorithm for capacity determination and for determining the best period of repetition. While these exact algorithms ensure that no processing resources are wasted, they may require exponential-time in the worst case. Bini et al. [10] studied minimization of energy consumption using periodic-resource considering discrete speed DVFS processor. Fisher and Dewan [28] developed a polynomial-time approximation algorithm which permits a system designer to choose a trade-off between accuracy and response time based on their requirements for non-thermal settings. In a related publication, Fisher [29] proposed a polynomial-time approximation for determining the period of repetition in aperiodic resource.

The Table 2.1 shows the summary of previous results. Note that $\surd$ and x indicate wether a particular research contribute to a given category or not respectively.

The analysis of the previous work does not answer the following: *is there any framework, that quantifies the impact of real-time timing properties upon unpredictable external constraints and advises the designer about the trade-offs between various physical and hardware constraints?* Due to lack of an answer to this question, we introduce a metric called system-resiliency which characterizes the maximum external stresses that any hard-real-time performance mode can withstand. This is a generalized framework for hard-real-time system design process. This proposed framework addresses resiliency determination for real-time systems with physical and hardware limitations. Also, it serves as a design tool that predicts the QoS or mode degradation due to external constraints.

Our system-resiliency framework provides a powerful model of cyber and physical system behavior. Theocratically, the correctness of the system-resiliency is explainable; however, it is too abstract to demonstrate the finer behavior of real-time system under multi-constrained environment. Therefore, we derive the system-resiliency for a sub-domain (lower-dimension) to demonstrate its correctness in practice. Our previous work [46, 48] introduces thermal resiliency – a design metric that quantifies the external thermal constraints. Finally, we introduce the generalized system-resiliency to complete our work.

Table 2.1: Thermal-aware researches and their contributions

Research	Real-Time	Multi-Core	Control-Theoritic	Deployment	Contribution
Brooks et al. [12]	x	x	x	-	Framework (DTM triggers, responses, init policies)
Chen et al. [15]	√	x	x	Proactive	-
Zanini et al. [94]	x	√	√(MPC)	x	-
Cohen et al. [17]	x	x	√(Optimal)	x	-
Fu et al. [35]	√	√	√(Robust)	x	DRE, simultaneous thermal-timeliness optimization (MIMO)
Ma et al. [63]	x	x	x	-	Microcode implementation
Skadron. [79]	x	x	x	CPU architecture	Hybrid architectural DTM
Wang et al. [90]	x	√	√(MPC)	x	Simultaneous power-frequency optimization (MIMO)
Wang et al. [88]	√	x	x	-	Framework (task degradation analysis)
Fu et al. [37]	√	x	√(Classical)	x	Anti-windup
Skadron et al. [78]	x	x	x	-	Framework (Hotspot)
Coskun et al. [19]	x	√	x	O/S	OS level implementation
Ferreira et al. [27]	√	x	x	-	Framework (System wide RC model)
Skadron et al. [77]	x	x	√(Classical)	x	Localized DTM techniques
Hettiarachchi et al. [46]	√	x	√	√	Hard-real-time, thermal-resiliency
Hettiarachchi et al. [48]	√	√	√	√	Hard-real-time, thermal-resiliency

CHAPTER 3: MODELS AND DEFINITIONS

In this chapter, we will introduce a basic overview on notations, concepts, and models used in the thesis. However, we only provide a very concise yet complete overview on the concepts and theories and details can be found in relevant textbooks.

3.1 Real-Time Systems Model

In next sub sections we present real-time models, notations, and definitions.

3.1.1 Jobs and Tasks

Any real-time system can be considered as a set of concurrent *tasks*. Each task generates an infinite number of jobs. The jobs from a same task obey sequential order of execution.

Definition 1 (Job)**.** *A real-time job $j = (A, E, D)$ is characterized by three parameters, an arrival time A, an execution requirement E, and a deadline D, with the interpretation that this job must receive e units of execution over the interval $[A, D)$.*

3.1.1.1 Periodic Task Model

In the periodic task model [7], a task T_i is completely characterized by a 4-tuple (a_i, e_i, d_i, p_i), where

1. the *offset* a_i denotes the instant at which the first job generated by this task becomes available for execution.
2. the *execution requirement* e_i specifies an upper limit on the execution requirement of each job generated by this task.
3. the *relative deadline* d_i denotes the temporal separation between each jobs arrival time and deadline. A job generated by this task arriving at time-instant t has a deadline at time-instant $(t + d_i)$.
4. the *period* p_i denotes the temporal separation between the arrival times of successive jobs generated by the task.

That is, $T_i = (a_i, e_i, d_i, p_i)$ generates a potentially infinite succession of jobs, each with execution-requirement e_i, at each instant (a_i + kp_i) for all integer $k \geq 0$, and the job generated at instant $(a_i + k.p_i)$ has a deadline at instant $(a_i + k\dot{p}_i + d_i)$ [7].

3.1.1.2 Sporadic Task Model

The sporadic task model differs from the periodic task model because of its inability to express job arrival time until the run-time moment. A sporadic task system is comprised of a finite collection of sporadic tasks [7].

Definition 2 (Sporadic Task)**.** *A* **sporadic task** $\tau_i = (e_i, d_i, p_i)$ *is characterized by a* worst-case execution requirement e_i*, a* (relative) deadline d_i*, and a* minimum inter-arrival separation p_i*. Such a sporadic task generates a potentially infinite sequence of jobs, with successive job-arrivals separated by at least* p_i *time units. A* sporadic task system $\tau \overset{def}{=} \{\tau_1, \ldots, \tau_n\}$ *is a collection of* n *such sporadic tasks.*

The utilization indicates the amount of time that system becomes busy due to tasks in a task model. Formally, the the utilization U_i of a periodic or sporadic task T_i is defined to be the ratio of its execution requirement to its period: $U_i = \frac{e_i}{p_i}$. The utilization $U(\tau)$ of a periodic or sporadic task system τ is defined to be the sum of the utilizations of all tasks in τ : $U = \sum_{T_i \in \tau} U_i$. for any task system under any known uni-processor scheduling algorithm, the utilization should not exceed 1 and cannot schedulable otherwise; however, this does not necessarily say that a system is scedulable if the utilization is under 1. This dissertation primarily focuses upon real-time work generated under the more general sporadic task model.

3.1.1.3 Online and Offline Algorithms

In *offline* scheduling algorithms, all scheduling decisions are made before the system begins executing. These scheduling algorithms select jobs to execute by referencing a table describing the pre-determined schedule. Usually, offline schedules are repeated after a least common multiple (LCM) period. The offline schedulers require the full knowledge of the job before the they execute.

In *online* scheduling algorithms, scheduling decisions are made without specific knowledge of jobs that have not yet arrived. These scheduling algorithms select jobs to execute by examining properties of active jobs. Online algorithms can be more flexible than offline algorithms since they can schedule jobs whose behavior cannot be predicted ahead of time.

3.1.2 Scheduling Algorithms

The job execution selection in a CPU is done by the scheduler. The scheduler uses a scheduling algorithm to achieve this task. Typical examples of scheduling algorithms are earliest deadline first (EDF), least laxity first (LLF), rate monotonic (RM), deadline monotonic (DM), etc. In EDF algorithm at each time-instant t schedule the job j active at time-instant t whose deadline parameter is the smallest [7]. Similarly, in LLF algorithm, the job with the smallest laxity has highest priority at all times. One very well-known fixed priority scheduling algorithm is the RM algorithm. In this algorithm, the task period is used to determine priority [58]. Tasks with shorter periods have higher priority. In DM algorithm, the deadline is used to determine the the priority. Tasks with shorter deadline have higher priority [58].

Depending on the requirement, a system might employ *online* or *off-line* scheduling algorithms. For example, if system has a well predefined task set and there are no runtime task modifications expected, then an off-line algorithms might be suitable. The off-line scheduling algorithms are suitable for systems with sufficient resources. However, an off-line scheduling algorithm cannot be used to withstand continuously varying dynamic task/job management. Sometimes, an online scheduling techniques which take the dynamics of the task into account is implemented [58]. Usually online scheduling algorithms of above kind use either control-theoretic or heuristic methods based approaches.

The primary intention of this document is to review work that analyzes the effects of resilient behavior on a real-time system. Therefore, this document describes the outcomes of the research on different scheduling techniques as applicable.

In next subsections, we give details on the system hardware model related to the uniprocessor systems.

3.2 Modeling of Uniprocessor Systems

3.2.1 System Hardware Models

We consider a single processor system with rudimentary DPM capabilities of only *active* and *inactive* power modes. At any time $t > 0$, we denote the instantaneous CPU power as $\mathcal{P}_{\text{cpu}}(t)$. The processor dissipates thermal power at a constant rate $\mathcal{P}_{\text{cpu}}(t) = \mathcal{P}_{\text{act}}$ in the active mode and $\mathcal{P}_{\text{cpu}}(t) = \mathcal{P}_{\text{inc}}$ in the inactive mode and these power dissipation corresponds to the current of the equivalent electrical circuit. While the processor is active, it dissipates heat at a constant rate $\mathcal{P}_{\text{cpu}}(t) = \mathcal{P}_{\text{act}}$. For the inactive mode, we will assume that the processor still continues to dissipate a small amount of power $\mathcal{P}_{\text{cpu}}(t) = \mathcal{P}_{\text{inc}}$. Also, we assume that processor consumes e_{act} amount of energy to activate from inactive mode and e_{inc} amount of energy to deactivate from the active mode. Even though the processor may be minimally active while in the low-power state, we will assume (as a pessimistic assumption for the purpose of schedulability analysis) that the processor is unavailable for task execution during this interval. If the aforementioned assumption does not hold, the system will behave "better" than the analysis and our results will continue to be valid. We believe this model of active/inactive modes is a very general model, applicable to a large number of available embedded processors with rudimentary DPM capabilities[1].

Furthermore, the active/inactive mode setting can model processing platforms that have limited thermal management capabilities. For example, in a processor with a single power-mode, we may say that the processor is active when running normal computational processes and inactive when running a "dummy" idle process. For ideal processors with continuous power modes, $\mathcal{P}_{\text{cpu}}(t)$ may be selected from the range $[0, \mathcal{P}_{\text{act}}]$.

3.2.2 Periodic Resource Model

Our control system for the active/inactive processor will enforce strict periodic mode changes. For this purpose, we employ a recently proposed *thermal-aware periodic resource* [5] model, which is an extension of the well-known periodic resource model proposed by [74] for compositional real-time systems. In the thermal-aware periodic resource model, the processing resource is characterized with a two-tuple (Π, Θ). The parameter Π is called the *resource period* and Θ is called the *resource capacity*. We will assume that Π is a non-negative integer (likely subject to the system tick granularity). The interpretation is that processor will be active for Θ amount of time at the beginning of each successive Π-length intervals[2]. The ratio Θ/Π is called the *resource bandwidth*. Within each processor allocation, an arbitrary uniprocessor scheduling algorithm (e.g., EDF or RM) may be employed to schedule the underlying task system. See

[1]Practically, we implement a system with $\mathcal{P}_{\text{act}}$ and $\mathcal{P}_{\text{inc}}$ power modes from any CPU with clock modulation feature, which is standard for many low end CPUs. Also, if the CPU has DVFS capabilities, $\mathcal{P}_{\text{act}}$ and $\mathcal{P}_{\text{inc}}$ power modes can be emulated by switching the CPU frequency between the highest and lowest available CPU frequencies.

[2]For system with clock modulation or DVFS capabilities, CPU active interval is Θ and inactive for $\Pi - \Theta$ interval.

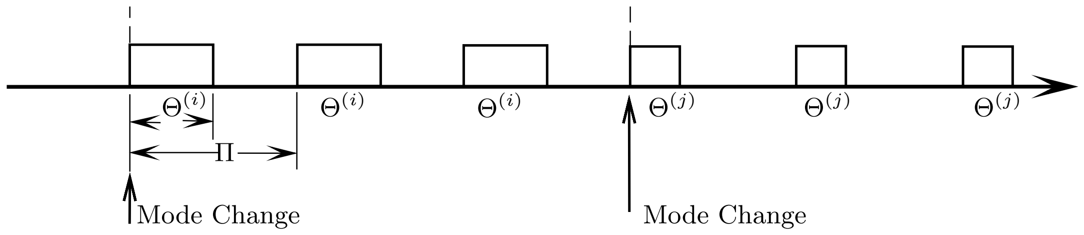


Figure 3.1: The sampling and mode change in the thermal control system (uni-processor systems).

The blocks indicate time periods during with the processor is active under the thermal-aware periodic resource model. Sporadic tasks are scheduled within the activation blocks.

Figure 5.1 for an illustration of the thermal-aware periodic resource.

3.2.3 Software Model

For the uniprocessor case, we assume each performance mode M_i is characterized by a *sporadic task system*[3] [64] with n_i tasks and the resource capacity $\Theta^{(i)}$. That is, $M_i = \left(\left\{\tau_1^{(i)}, \tau_2^{(i)}, \ldots, \tau_{n_i}^{(i)}\right\}, \Theta^{(i)}\right)$ where each $\tau_j^{(i)} \in M_i$ is a sporadic task characterized by a three-tuple $(e_j^{(i)}, d_j^{(i)}, p_j^{(i)})$ and $\Theta^{(i)}$ is the minimum capacity required to meet the deadlines of the tasks of M_i. (Note that we are abusing notation by allowing M_i to represent the set of tasks and the two-tuple of the mode's task system and required resource capacity.) In this three-tuple representation for a task, $e_j^{(i)}$ is the *worst-case execution requirement*, $d_j^{(i)}$ is the *relative deadline*, and $p_j^{(i)}$ is the *minimum inter-arrival separation parameter* (historically called the "period"). A sporadic task $\tau_j^{(i)}$ may produce a (potentially infinite) sequence of jobs, where each job has an execution requirement of $e_j^{(i)}$ time units and must complete $d_j^{(i)}$ time units after its arrival. The first job of $\tau_j^{(i)}$ may arrive at any time after system-start time; however, successive jobs of $\tau_j^{(i)}$ must arrive at least $p_j^{(i)}$ time units apart. Further, throughout this thesis, unless we explicitly mentioned, we assume that the resource period Π is identical in all modes. For mode M_i, a resource capacity of $\Theta^{(i)}$ is provided every resource period. Figure 3.1 illustrates the processing-time allocation in two different modes.

[3]Note, we will be assuming the sporadic task model throughout our objectives, but the results could be extended to other task models without much change.

3.2.4 Performance Modes

We will assume that there is an ordering of real-time performance modes based on their "computational requirements" to meet all of a mode's deadlines. The relation $M_i \succeq M_j$ indicates that M_i is more computationally intensive than M_j. For notational convenience, we will assume that mode M_0 represents the mode with no tasks and $\Theta^{(0)}$ equal to zero. Furthermore, we assume that the modes are well-ordered and have been indexed in increasing order of computational requirements; i.e., $M_0 \preceq M_1 \preceq M_2 \preceq \ldots \preceq M_q$. While there are many possible ways to define the $\preceq$ relation, the only ordering required from the perspective of our thermal control is that $M_i \preceq M_j$, if and only if, $\Theta^{(i)} \leq \Theta^{(j)}$; i.e., to reduce the temperature of the system, we need to decrease the processing-time allocation.

3.2.5 Mode Change Semantics

Our model does not require any particular mode-change semantics to be adopted. Some potential options for dealing with incompletely-executed jobs upon a mode change are:

1. aborting any incomplete jobs;
2. delaying the release of jobs in the new mode until all jobs of the old mode have completed;
3. allowing jobs of the new mode to be released, as soon as legally allowable, while jobs of the old mode are still active.

For the purposes of our hardware testbed and simulations, we assume option 3.

3.2.6 Selection of the Scheduling Algorithm

The scheduling of real-time performance mode M_i upon the thermal-aware periodic resource may be done by any uniprocessor real-time scheduling algorithm (e.g., earliest-deadline-first or rate-monotonic [57]). However, $\Theta^{(i)}$ must be sufficiently large for the scheduling algorithm to correctly schedule all jobs of the task set of M_i (i.e., $\{\tau_1^{(i)}, \tau_2^{(i)}, \ldots, \tau_{n_i}^{(i)}\}$) and (potentially) any jobs from the previous mode that have not completed by the mode change. To obtain a proper resource allocation, $\Theta^{(i)}$, for each mode, we use our recently-developed hard-real-time schedulability test (for EDF scheduling under hardware/software mode changes in the periodic resource model) to search for a safe value of $\Theta^{(i)}$ for each mode [30] to ensure that deadlines are always met. The multi-modal schedulability analysis ensures that for any valid sequence of mode changes and valid set of job arrivals under the sporadic task model that the EDF scheduler will always meet all deadlines. The analysis works by determining the maximum workload carried from one mode to another and testing whether this "carry-in" will cause a deadline miss. Furthermore, any other online scheduling algorithm can also be used (e.g., fixed priority).

3.2.7 Power/Thermal Model

For the uniprocessor case, we use the duality principle in electrical and thermal circuits to describe the dynamics of the power dissipating source using electrical resistance/capacitance (RC) circuits. Therefore, we represent the CPU thermal model with a single RC circuit. Figure 3.2 shows the basic equivalent circuit for the CPU and its surrounding environment. We assume that total dissipated power of the CPU $\mathcal{P}_{\text{cpu}}$ is equal to the sum of the power due to dynamic current $\mathcal{P}^{\text{d}}_{\text{cpu}}$ and power due to leakage current $\mathcal{P}^{\ell}_{\text{cpu}}$. Furthermore, we assume that the temperature-dependant leakage power may be closely approximated by a linear function of CPU temperature [59].

Let $V_{\text{cpu}}(t), V_{\text{env}}(t),$ and $V_{\text{air}}(t)$ represent the equivalent voltages for temperatures of the CPU, environment, and air (room) respectively. Let $\mathcal{T}_{\text{cpu}}$ be the instantaneous relative temperature of the CPU with respect to the immediate environment (e.g., CPU casing), $\mathcal{T}_{\text{env}}$ be the relative temperature of the immediate environment with respect to the room air temperature, and $\mathcal{T}_{\text{air}}$ be the (absolute) room air temperature. For example, if $\mathcal{T}_{\text{air}}$ is $20°C$, $\mathcal{T}_{\text{env}}$ is $10°C$, and $\mathcal{T}_{\text{cpu}}$ is $15°C$, then the absolute temperature of the CPU is $45°C$.

We assume $\mathcal{P}^{\text{d}}_{\text{cpu}}(t)$, $\mathcal{P}^{\ell}_{\text{cpu}}(t)$, and $\mathcal{P}_{\text{env}}(t)$ represent, respectively, the dynamic CPU, leakage CPU, and environment power dissipation. Let $R^{\text{d}}_{\text{cpu}}$, $R^{\text{l}}_{\text{cpu}}$, R_{env}, $C^{\text{d}}_{\text{cpu}}$, $C^{\text{l}}_{\text{cpu}}$, and C_{env} represent the dynamic and leakage thermal resistance, environment resistance, CPU dynamic and leakage capacitance, and environment capacitance. Finally, we let $\sigma_1 \overset{\text{def}}{=} \frac{1}{C^{\text{d}}_{\text{cpu}}+C^{\text{l}}_{\text{cpu}}}$ and k_T and k_C represent processor-dependent constants used in approximating the temperature-dependant leakage current.

Next, we give the details on the system hardware model related to the multicore processor case as follows.

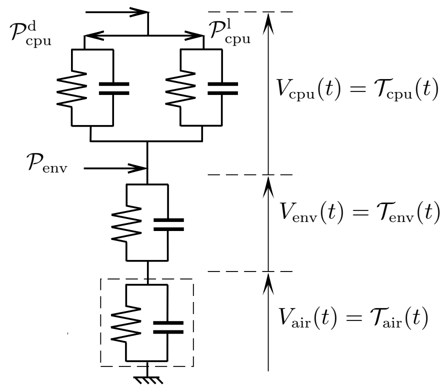

$\mathcal{P}^{\mathrm{d}}_{\mathrm{cpu}}$
$\mathcal{P}^{\mathrm{l}}_{\mathrm{cpu}}$
$V_{\mathrm{cpu}}(t) = \mathcal{T}_{\mathrm{cpu}}(t)$
$\mathcal{P}_{\mathrm{env}}$
$V_{\mathrm{env}}(t) = \mathcal{T}_{\mathrm{env}}(t)$
$V_{\mathrm{air}}(t) = \mathcal{T}_{\mathrm{air}}(t)$

Figure 3.2: The basic equivalent circuit for a working CPU and its working environment

3.3 Modeling of Multicore Processor Systems

Similar to the uniprocessor case, we develop our power model to represent a wide range of embedded processors with minimal amount of power management capabilities.

3.3.1 System Hardware Model

We will assume that the multicore processor consists of m number of active cores and each core has *active* and *inactive* power modes [4]. We denote the instantaneous CPU power of $\mathcal{C}$'th core as $\mathcal{P}^{\mathcal{C}}_{\text{cpu}}(t)$, $(\mathcal{C} \in \{1 \dots m\})$ and assume it dissipates thermal power at a constant rate $\mathcal{P}_{\text{act}}$ and $\mathcal{P}_{\text{inc}}$ in the active and inactive modes, respectively. Also, the power of each core, $\mathcal{P}^{\mathcal{C}}_{\text{cpu}}(t)$ $\forall \mathcal{C} \in \{1 \dots m\}$ may vary from the range $[0, \mathcal{P}_{\text{act}}]$. Further, our model supports the processor to emulate the active and inactive power modes with dynamic voltage and frequency scaling (DVFS) if the processor does not have the real active and inactive implementation. Also, we assume that processor consumes e_{act} and e_{inc} amount of energy to acti-

[4]Most of the modern CPUs do not support DFVS capability at the core granularity [1]. Therefore, in order to emulate active and inactive power level at the core granularity, the resource capacity Θ of each core should be changed along with clock modulation or DVFS.

vate/deactivate from inactive/active modes. Further, during this interval, the processor is inactive while in the low-power state and unavailable for payload task execution. If the processor is minimally active instead of unavailable during inactive interval, the system will behave better than the analysis and our results will continue to be valid.

Our control system for the active/inactive processors will enforce strict periodic mode changes. For this purpose, we employ *thermal-aware periodic resource* [5] model. In the thermal-aware periodic resource model, the processing resource is characterized with a two-tuple $(\Pi, \Theta^{\mathcal{C}})$. The parameter Π is called the *resource period* and $\Theta^{\mathcal{C}}$ is called the *resource capacity* of the $\mathcal{C}$'th core. We will assume that $\Pi(> 0)$ is subject to the system tick granularity. The interpretation is that processor will be active for Θ amount of time at the beginning of each successive Π-length intervals. Within each processor allocation, an arbitrary uniprocessor scheduling algorithm (e.g., EDF or RM) may be employed to schedule the underlying task system. See Figure 5.1 for an illustration of the thermal-aware periodic resource.

3.3.2 Software Model

We will assume that each CPU core has specific number of possible performance modes. Also, the task migration at runtime is not permissable and the tasks are statically partitioned within the available processors. We assume each performance mode $M^{\mathcal{C},i}$ of the $\mathcal{C}$'th core is characterized by a *sporadic task system*[5] [64] with n_i tasks and the resource capacity $\Theta^{\mathcal{C},i}$, where $\Theta^{\mathcal{C},i}$ represents the minimum resource capacity required for i'th mode. That is, $M^{\mathcal{C},i} = \left(\left\{\tau_1^{\mathcal{C},i}, \tau_2^{\mathcal{C},i}, \ldots, \tau_{n_i}^{\mathcal{C},i}\right\}, \Theta^{\mathcal{C},i}\right)$ where each $\tau_j^{\mathcal{C},i} \in M^{\mathcal{C},i}$ is a sporadic task characterized by a three-tuple $(e_j^{\mathcal{C},i}, d_j^{\mathcal{C},i}, p_j^{\mathcal{C},i})$ and $\Theta_j^{\mathcal{C},i}$ is the minimum capacity required to meet the deadlines of the tasks of $M^{\mathcal{C},i}$. (Note that we are abusing notation by allowing $M^{\mathcal{C},i}$ to represent the set of tasks and the two-tuple of the mode's task system and required resource capacity.) In this three-tuple representation for a task, $e_j^{\mathcal{C},i}$ is the *worst-case execution requirement*, $d_j^{\mathcal{C},i}$ is the *relative deadline*, and $p_j^{\mathcal{C},i}$ is the *minimum inter-arrival separation parameter*. A sporadic task $\tau_j^{\mathcal{C},i}$ may produce an infinite sequence of jobs, where each job has an execution requirement of $e_j^{\mathcal{C},i}$ time units and must complete $d_j^{\mathcal{C},i}$ time units after its arrival. The first job of $\tau_j^{\mathcal{C},i}$ may arrive at any time after system-start time; however, successive jobs of $\tau_j^{\mathcal{C},i}$ must arrive at least $p_j^{\mathcal{C},i}$ time units apart. We assume that the resource period Π is identical in all modes. For mode $M^{\mathcal{C},i}$, a resource capacity of $\Theta^{\mathcal{C},i}$ is provided every resource period.

[5]Note, we will be assuming the sporadic task model throughout our objectives, but the results could be extended to other task models without much change.

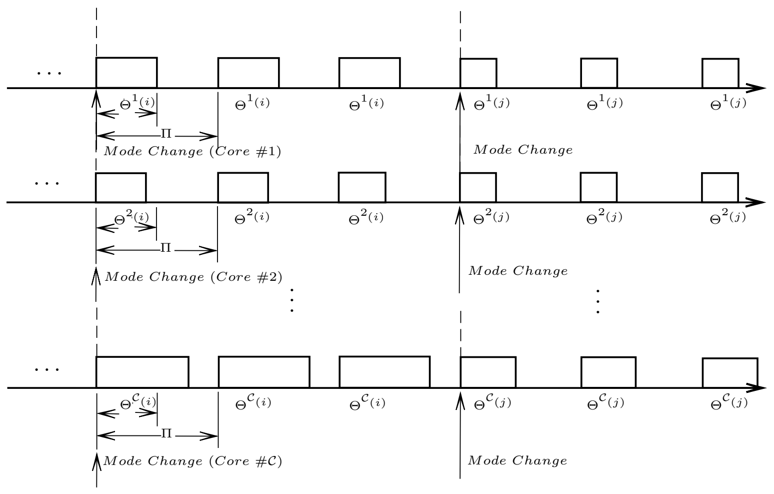


Figure 3.3: The sampling and mode change in the thermal control system (multicore processor system). The blocks indicate time periods during with the processor is active under the thermal-aware periodic resource model. Sporadic tasks are scheduled within the activation blocks .

Figure 3.3 illustrates the processing-time allocation in two different modes.

3.3.3 Performance Modes

Similar to the uniprocessor case, we assume that there is a partial ordering of real-time performance modes based on their "computational requirements" to meet all of a mode's deadlines. The relation $M^{\mathcal{C},i} \succeq M^{\mathcal{C},j}$ indicates that $M^{\mathcal{C},i}$ is more computationally intensive than $M^{\mathcal{C},j}$. For notational convenience, we will assume that mode $M^{\mathcal{C},0}$ represents the mode where with no tasks and $\Theta^{\mathcal{C},0}$ equal to zero. Furthermore, we assume that the modes are well-ordered and have been indexed in increasing order of computational requirements; i.e., $M^{\mathcal{C},0} \preceq M^{\mathcal{C},1} \preceq M^{\mathcal{C},2} \preceq \ldots \preceq M^{\mathcal{C},q}$. While there are many possible ways to define the $\preceq$ relation, the only ordering required from the perspective of our thermal control is that $M^{\mathcal{C},i} \preceq M^{\mathcal{C},j}$, if and only if, $\Theta^{\mathcal{C},i} \leq \Theta^{\mathcal{C},j}$; i.e., to reduce the temperature of the system, we need to decrease the processing-time allocation. Furthermore, we do not define any relationship with modes on different processors. For example, the relationship $M^{\mathcal{C}_p,i}$ to $M^{\mathcal{C}_q,i}$, $\mathcal{C}_p, \mathcal{C}_q \in \{1 \ldots m\}$ is not defined. Our system allows jobs of the new mode to be released, as soon as legally allowable, while jobs of the old mode are still active.

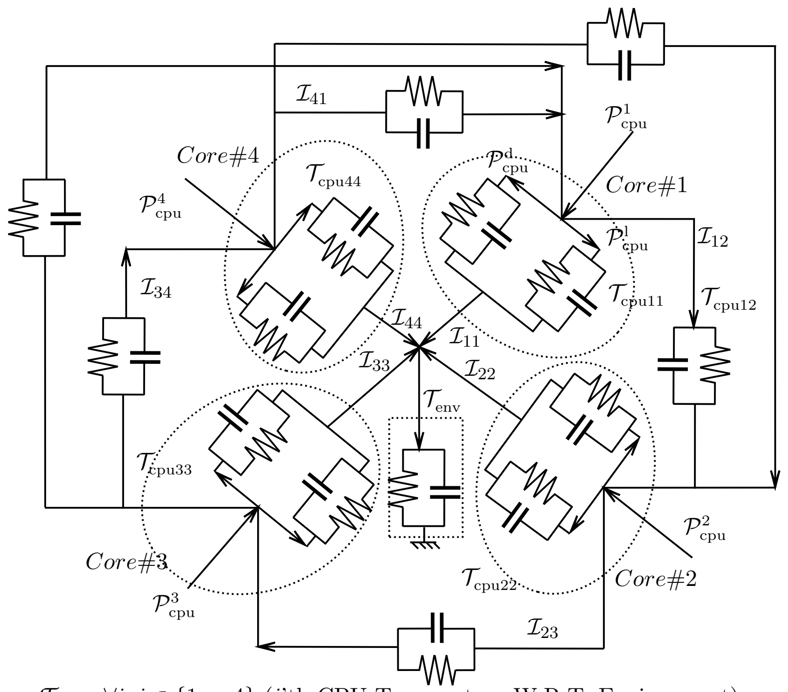
The scheduling of real-time performance mode $M^{\mathcal{C},i}$ upon the thermal-aware periodic resource may be done by any real-time scheduling algorithm (e.g., earliest-deadline-first or rate-monotonic [57]). Note that we do not consider the task migration within cores, thereby we can carry out the schedulability test on each core individually. However, $\Theta^{\mathcal{C},i}$ must be sufficiently large for the scheduling algorithm to correctly schedule all jobs of the task set of $M^{\mathcal{C},i}$ (i.e., $\{\tau_1^{\mathcal{C},i}, \tau_2^{\mathcal{C},i}, \ldots, \tau_{n_i}^{\mathcal{C},i}\}$) and (potentially) any jobs from the previous mode that have not completed by the mode change. To obtain a proper resource allocation, $\Theta^{\mathcal{C},i}$, for each mode, we use our hard-real-time schedulability test (for EDF scheduling under hardware/software mode changes in the periodic resource model) to search for a safe value of $\Theta^{\mathcal{C},i}$ for each mode [30] to ensure that deadlines are <u>always</u> met.

3.3.4 Power/Thermal Model

The thermal architecture (model) of a multicore processor should contain the component to represent each CPU core as well as inter-core thermal effects. For multicore processor thermal model, we extend the RC circuit that we developed for the uniprocessor case. Therefore, we describe the dynamics of the power dissipating source using electrical resistance/capacitance (RC) circuits. Figure 3.4 shows the basic equivalent circuit of a multicore CPU and its surrounding environment. We assume that total dissipated power of $\mathcal{C}$'th CPU core, $\mathcal{P}_{\text{cpu}}^{\mathcal{C}}$ is equal to the sum of the power due to dynamic current ${\mathcal{P}_{\text{cpu}}^{\text{d}}}^{\mathcal{C}}$ and power due to leakage current ${\mathcal{P}_{\text{cpu}}^{\ell}}^{\mathcal{C}}$ of $\mathcal{C}$'th core. Also, we assume that the leakage and the dynamic current parts within different cores are approximated with single component.

Let $V_{\text{cpu}}^{\mathcal{C}}(t)$ and $V_{\text{env}}^{\mathcal{C}}(t)$ represent the equivalent voltages for temperatures of the $\mathcal{C}$'th core of the CPU and environment (room) respectively. Let $\mathcal{T}_{\text{cpu}}^{\mathcal{C}}$ be the instantaneous relative temperature of the $\mathcal{C}$'th core of the CPU with respect to the immediate environment (e.g., CPU casing), $\mathcal{T}_{\text{env}}^{\mathcal{C}}$ be the relative temperature of the immediate environment.

Let $\mathcal{P}_{\text{env}}(t)$ represents, the environment power dissipation. Let $R_{\text{cpu(i,j)}}^{\text{d}}$, $R_{\text{cpu(i,j)}}^{\ell}$, R_{env}, $C_{\text{cpu(i,j)}}^{\text{d}}$, $C_{\text{cpu(i,j)}}^{\ell}$, and C_{env} represent, respectively the dynamic and leakage thermal resistance, environment resistance, CPU dynamic and leakage capacitance, and environment capacitance between i'th and j'th cores, $i, j \in \{1 \ldots m\}$.


$\mathcal{I}_{41}$
$\mathcal{P}^{1}_{\mathrm{cpu}}$
$Core\#4$
$\mathcal{P}^{\mathrm{d}}_{\mathrm{cpu}}$
$\mathcal{T}_{\mathrm{cpu44}}$
$Core\#1$
$\mathcal{P}^{4}_{\mathrm{cpu}}$
$\mathcal{P}^{1}_{\mathrm{cpu}}$
$\mathcal{I}_{12}$
$\mathcal{I}_{34}$
$\mathcal{T}_{\mathrm{cpu11}}$
$\mathcal{T}_{\mathrm{cpu12}}$
$\mathcal{I}_{44}$
$\mathcal{I}_{11}$
$\mathcal{I}_{33}$
$\mathcal{I}_{22}$
$\mathcal{T}_{\mathrm{env}}$
$\mathcal{T}_{\mathrm{cpu33}}$
$\mathcal{P}^{2}_{\mathrm{cpu}}$
$Core\#3$
$\mathcal{T}_{\mathrm{cpu22}}$
$Core\#2$
$\mathcal{P}^{3}_{\mathrm{cpu}}$
$\mathcal{I}_{23}$

$\mathcal{T}_{\mathrm{cpuii}}, \forall i, j \in \{1 \ldots 4\}$ (i'th CPU Temperature W.R.T. Environment)

$$\mathcal{T}_{\mathrm{cpuij}} = \mathcal{T}_{\mathrm{cpujj}} - \mathcal{T}_{\mathrm{cpuii}} \ \forall i, j \in \{1 \ldots 4\}$$

Figure 3.4: The thermal model of the CPU and its working environment.

The basic equivalent electrical circuit of the thermal model of the CPU and its working environment. (for simplicity, the figure shows the structure with 4 adjacent cores) Arrow direction shows the current (A) direction of the equivalent electrical circuit.

3.3.5 Generalized Model

System designers use differential equations to construct mathematical models of physical systems [11]. They are widely used in industry. Furthermore, there are mathematical techniques to convert higher-order differential equations to system of first-order differential equations. Since the system of first-order differential equations characterizes a state-space model, in this research, we use the following state-space model to describe the underlying physical capabilities and dynamics of a generalized system,

$$\begin{aligned} \dot{\mathcal{X}}_{\text{cpu}} &= \mathcal{A}\mathcal{X}_{\text{cpu}} + \mathcal{B}\rho_{\text{cpu}}, \\ \mathcal{Y}_{cpu} &= \mathcal{C}\mathcal{X}_{\text{cpu}}, \end{aligned} \tag{3.1}$$

where, $\mathcal{X}_{\text{cpu}}$, $\dot{\mathcal{X}}_{\text{cpu}}$, $\mathcal{Y}_{cpu}$, and ρ_{cpu} represent the state-space variable, the first derivative of the state-space variable, system output, and the control input, respectively. The dimensions of the these variables depend upon the number of physical capabilities (such as controllable parameters of the system) and observable states (parameters that can be measured or defined according to the designer's need) that are modeled in the system. For a system with ℓ capabilities and o observable states, the state-space variable is a o-dimensional vector and the control input is a ℓ-dimensional vector. Also, $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$ are state-space parameters. A more detailed description about state-space model and its derivations can be found in a standard texts [21][65][66].

3.3.6 Generalized System Software Model

An adaptive real-time system requires a specification of how the underlying real-time software is affected by changes in the processor state due to changing environmental conditions (e.g., how are deadlines affected by reducing processing frequency or durations?). Our software model assumes that each core has specific performance modes and no runtime task migration is allowed – i.e., tasks are statically partitioned within the available processors. In our notation, the i'th performance mode of the $\mathcal{C}$'th core, $M^{\mathcal{C},i}$ is characterized by a *sporadic task system*[6] [64] with n_i tasks and the resource capacity $\Theta^{\mathcal{C},i}$, where $\Theta^{\mathcal{C},i}$ represents the minimum resource capacity required for i'th mode. Therefore, using abused notation,

[6]Note, we will be assuming the sporadic task model throughout our objectives, but the results could be extended to other task models without much change.

$M^{\mathcal{C},i}$ can be used to represent the two-tuple of the mode's task system and required resource capacity, $M^{\mathcal{C},i} = \left(\left\{\tau_1^{\mathcal{C},i}, \tau_2^{\mathcal{C},i}, \ldots, \tau_{n_i}^{\mathcal{C},i}\right\}, \Theta^{\mathcal{C},i}\right)$, where each $\tau_j^{\mathcal{C},i} \in M^{\mathcal{C},i}$ is a sporadic task characterized by a three-tuple $(e_j^{\mathcal{C},i}, d_j^{\mathcal{C},i}, p_j^{\mathcal{C},i})$ and $\Theta^{\mathcal{C},i}$ is the minimum capacity required to meet the deadlines of the tasks of $M^{\mathcal{C},i}$. Furthermore, we use $\mathcal{M}_{rt}$ to denote a mode-vector, $\mathcal{M}_{rt} = \begin{bmatrix} M^{1,i_1} & M^{2,i_2} & \ldots & M^{m,i_m} \end{bmatrix}^T$ that represents the modes of all the cores of the system at a given time, where $M^{\mathcal{C},i_1} \ldots M^{\mathcal{C},i_m}$ represents different possible modes in each core and T denote the matrix transpose operation. (An alternative to resource capacity–for future work–could potentially be CPU core frequency, f^i to represent processing resources for i'th performance mode). In this three-tuple representation for a task, $e_j^{\mathcal{C},i}$ is the *worst-case execution requirement*, $d_j^{\mathcal{C},i}$ is the *relative deadline*, and $p_j^{\mathcal{C},i}$ is the *minimum inter-arrival separation parameter*. A sporadic task $\tau_j^{\mathcal{C},i}$ may produce an infinite sequence of jobs, where each job has an execution requirement of $e_j^{\mathcal{C},i}$ time units and must complete $d_j^{\mathcal{C},i}$ time units after its arrival. The first job of $\tau_j^{\mathcal{C},i}$ may arrive at any time after the start of mode $M^{C,i}$; however, successive jobs of $\tau_j^{\mathcal{C},i}$ must arrive at least $p_j^{\mathcal{C},i}$ time units apart.

Similar to multicore processor case, for the generalized model, we employ the *thermal-aware periodic resource* [5] model. For mode $M^{\mathcal{C},i}$, a resource capacity of $\Theta^{\mathcal{C},i}$ is provided every resource period; each core's resource capacity can be viewed as a variable that controls one of the ℓ physical capabilities of the processor.

3.4 Control Systems Basics

Control theory gives a theoretical guarantee about the stability and the expected behavior of a system. The designer determines timeliness guarantee of a system (e.g., in thermal-aware system, how quickly the temperature settles back to the desired temperature), and an analysis of the maximum deviation of a system from its normal operation using control theory. These are very useful in developing a real-world systems as engineers can design a safe system without risking an actual emergency.

Often real-time systems integrate with control theory to produce sophisticated real-world systems. For example, consider the operation of a computer numerical controlled (CNC) machine that cuts metal in a three dimensional surface. The final metal cutting quality and the safety (of the operator and the machine itself) depend on the calculation and temporal accuracy of the real-time computer as well as the accuracy of the controller. The machine determines its next servo action with respect to multiple calculations and

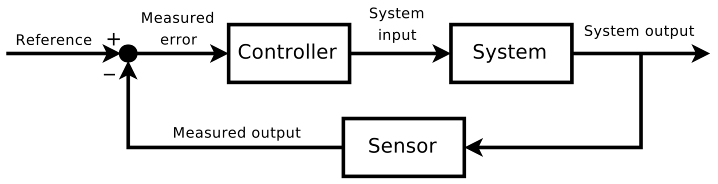


Figure 3.5: A control system with a feedback

actuate multiple servo systems. A typical CNC machine continuous the above operation 2000 times per second. A complex system such as described above will not work as expected if they are not hard real-time control systems. Therefore, studying the integration of the control systems with real-time systems is important. Also, studying correct methodologies to applying the control systems theory within the real-time frame work is a valuable design experience.

There are several classes of controllers available. Depending on the design requirements, the most appropriate class of application can be used. For example, the classical control theory is very mature and it has the ability to analyze the system behavior and the relative stability through phase and gain margins [32, 67] explained below. However, classical theory has a disadvantage that it does not support multi-input, multi-output (MIMO) systems. If any MIMO system has to be analyzed with classical control systems, it has to be done in several iterations, one control loop at a time. To address the limits of classical control theory, a modern control theory techniques were developed. With modern control theory, the MIMO systems can be handled elegantly. However, in modern control theory, some concepts such as the relative stability and gain and phase margin (explained below) are not fully developed.

3.4.1 Basic Control Terminology

In the rest of this chapter, the details of different control theory classes are described. The following definitions and terms are widely used in explaining control systems and their applications. These terms are equally applicable for both classical and modern control systems.

Definition 3 (Control Variable and Manipulated Variable)**.** *The* controlled *variable is the quantity or condition that is measured and controlled. The* manipulated *variable is the quantity or condition that is varied by the controller [67].*

Normally, the controlled variable is the output of the system. When a control system uses its output to investigate the status of the systems, it is commonly known as a *feedback control system*. In a feedback control system, the controlled variable (output) is compared with the desired output (reference) of the system. The feedback control systems are also known as *closed loop* systems. On the other hand, system without a *feedback* is known as *open loop* system. Further, the difference between the reference to the controlled variable is commonly known as the *error*. The desired action of any control system is to reduce the *error*.

Definition 4 (Plant)**.** *A plant may be a piece of equipment, perhaps just a set of machine parts functioning together, the purposes of which is to perform a particular operation [67].*

In this survey, active thermal model of the CPU was considered as the plant.

Definition 5 (Transfer Function)**.** *The transfer function of a linear system is the ratio of the Laplace Transform of the output to the Laplace Transform of the input [67].*

3.4.2 The Phase and Gain Margins

The intuitive meanings of the gain and phase margin [67, 21, 65] are as follows. Gain and phase margins are measures of stability for a feedback system. Gain margin is the difference between unity and *open loop* gain (see Section 3.4.1) when the phase plot is $-180°$. A system with greater gain margin can withstand greater changes in system parameters before becoming unstable in closed loop. Also, the phase margin is equal to the phase difference between *open loop* phase and $-180°$ when $0dB$ is the frequency in magnitude plot [67, 22, 65, 21]. The phase margin also measures the system's tolerance to time delay. The gain and phase margins are the relative stability and robustness indicators[7] of any control systems. Unfortunately, with modern control theory it is not possible to derive these important parameters easily [21, 67]. The following Figure 3.6 shows gain and phase margins in a Bode plot.

[7] A controller is considered as robust if it can withstand a higher different level of tolerance (deviation from the particular set of parameters that was originally designed). High-gain (negative) feedback is an artifact of a basic robust control method. With the help of sufficiently larger gains, the effect of input variations can be nullified in most stable controllers [8].

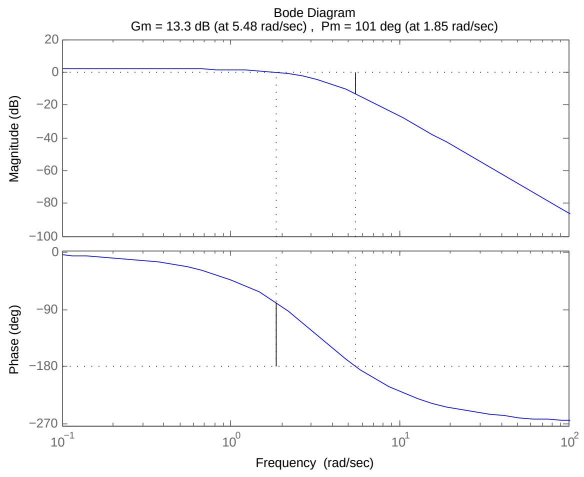


Figure 3.6: The phase margin and the gain margin of a system

3.5 Classical Control Theory

Classical control theory is appropriate if the plant model is (nearly) exactly known. A wealth of theories and common knowledge are particularly helpful for application engineer to solve issues for such plants. In classical control theory, the stability analysis of the system is done mainly with the help of plant transfer function (see Definition 5). The *pole*s and *zeros* of a transfer function are the values for which the output of the transfer function becomes infinity or zero respectively. By analyzing the zeros and poles of the closed loop transfer function, the stability the system can be determined. However, due to computational difficulty of the closed loop transfer function, usually the stability is analyzed using root locus [21] techniques. Further, the relative robustness (refer to Section 3.4.2) is analyzed using frequency plots [21]. The above Figure 3.7 shows the root locus of transfer function $G(s) = \frac{-2.006s^3-4.99s^2-6.69.9s-2.4534}{s^3+3.18S+3.2s+1.07}$.

Following are the lemmas to analyze the stability of a control system under classical methods.

Lemma 1 (from [67])**.** *A linear time invariant (LTI) system is* asymptotically stable *if and only if its all poles lie left half of the complex plane.*

Also, for discrete-time LTI which is given as follows:

Lemma 2 (from [66])**.** *A discrete-time linear time invariant (LTI) system is* asymptotically stable *if and*

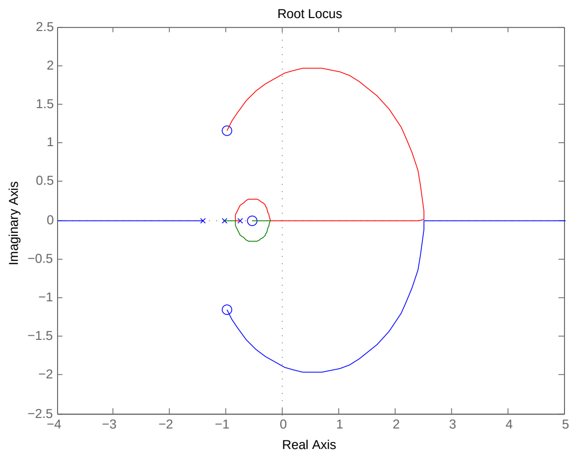


Figure 3.7: The root locus of transfer function $G(s)$

only if its all poles lie strictly inside the unit circle.

3.5.1 Proportional Integral Derivative Controllers

Proportional integral derivative (PID) controllers are considered as the most basic and easy to design and implement in classical control theory. Equation 3.2 below shows the basic PID equation in a typical plant:

$$u(t) = K_P e(t) + K_I \int_0^t e(\tau)d\tau + K_D \frac{d}{dt} e(t), \tag{3.2}$$

where, K_P, K_I, and K_d denotes the proportional, integration, and derivative controller gains. $e(t)$ denotes the the error (the difference between the reference and the output signal), and $u(t)$ denotes the plant input. The PID controller takes the error signal ($e(t)$) as input signal and modify it according to the Equation 3.2, so that the output of the (PID) controller can be used as an input to a given plant.

The PID controllers are easy to adjust to a particular application with its there parameters. For example, if a particular application require a rapid response, then the proportional constant of the controller needs to be increased. Depending on the requirement, sometimes application developers select PI, PD, P controllers. By refereing to the step response plot, the designer can get a clear idea about the overshoot, steady state error, peak time, and other important parameters [44] of the controller dynamics For a detailed treatment of the above parameters can be found in any classical control theory theory text (please refer to [67, 21, 65]).

3.5.2 Anti-Windup Controller

There are situations in classical control systems that should be handled with special care. In any control system, the manipulated variable (plant input) has its limitations and the input value should not exceed given boundary values. For example, when any control system manipulates the utilization as the plant input, it should not exceed the utilization bounds of the underlying scheduling algorithm of the system (say u_{max}). This means, when the controller calculates any input value higher than u_{max}, there should be extra control mechanisms to correct this situation. Similarly, when the input value passes certain lower bound, it should be corrected. The control mechanism that corrects the input saturation situation is called *anti-windup* controller [65, 21].

Therefore, the system input, $u_c(t)$ is modified to $u(t)$ with anti-windup correction as follows:

$$u(t) = \begin{cases} -u_{max}, & \text{for } u_c(t) < -u_{max}; \\ u_c(t), & \text{for } u_c(t) \leq u_{max}; \\ u_{max}, & \text{for } u_c(t) > u_{max}. \end{cases} \tag{3.3}$$

Meanwhile, a similar input saturation situation is handled by the optimal controllers differently. In optimal controllers, the input saturation is defined as a constraint for the controller (input constraints) and therefore, there is no need to modify the control structure. Therefore, optimal controller techniques solves the optimality of the system to meet the constraints. Standard texts in optimal control theory (please refer to [56]) give elaborative explanation about this problem formulation.

3.6 Modern Control Theory

The modern control theory uses a state-space model. This makes it easy to implement and analyze using computers. Because it allows to analyze MIMO systems, larger systems with highly complex nature can be analyzed easily. Also, the modern control theory utilizes the time-domain analysis, which make it more easy to interpret at the design time. In state-space model, the following equations/notation is used to denote the basic control structure [67].

$$\begin{aligned} \dot{x}(t) &= A(t)x(t) + B(t)u(t) + f, \\ y(t) &= Cx(t), \end{aligned} \tag{3.4}$$

where $x(t), u(t)$, and $y(t)$ represent the state vector, the input vector, and the output vector, respectively. $A(t), B(t)$, and C represent the system matrices and f represents a constant vector. All the state matrices can be constant vectors and time-invariant quantities or they can be time-varying quantities. If A and B are linear and time-invariant, the system can be analyzed as linear time invariant (LTI) system. Otherwise, different system theories can be applied to find the system stability (non-linear, optimal control theory etc.).

Also, for discrete time state-space model the following equations are commonly used [32].

$$\begin{aligned} x((k+1)T_s) &= Gx(kT_s) + Hu(kT_s) + \tilde{f}, \\ y(kT_s) &= \tilde{C}x(kT_s), \end{aligned} \tag{3.5}$$

for LTI case we can find the inter-relation of state space matrices [32] as $G = e^{AT_s}$, $H = \int_0^{T_s} e^{At}dtB$, $\tilde{C} = C.\ e^{AT_s}$, and $\tilde{f} = \int_0^{T_s} e^{At}fdt$, can be computed by $\mathcal{L}^{-1}\{(sI-A)^{-1}\}_{t=T_s}$, where $\mathcal{L}^{-1}$ is the inverse Laplace transform. We abuse the notation by representing $x(kT_s)$ as $x(k)$, $x((k+1)T_s)$ as $x(k+1)$, $u(kT_s)$ as $u(k)$, and $y(kT_s)$ as $y(k)$. The above definitions may be found in any textbook on discrete-time control theory [66].

Stability is a broad concept in control systems. Depending on the type of the system, (whether non-linear or time-variant) the correct stability analysis should be used. The most commonly used control systems in thermal-aware scheduling can be analyzed with the following criteria. For the asymptotic stability for a continuous LTI system, the following lemma is used:

Lemma 3 (from [67])**.** *A linear time invariant (LTI) system is* asymptotically stable *if and only if its all eigenvalues of A lie left half of the complex plane.*

Also, for discrete-time LTI, stability is given as follows:

Lemma 4 (from [66])**.** *A discrete-time linear time invariant (LTI) system is* asymptotically stable *if and* *only if its all eigenvalues of G lie strictly inside the unit circle.*

The controllability of a system is a critical factor to consider in any state-space model.

Lemma 5 (from [21])**.** *The system of Equation (3.4) is* completely controllable *if there exists an unconstrained $u(t)$ such that it can control any initial state $x(t_0)$ to any desired final state x_f in a finite time, $t_0 \leq t \leq T$. The property of completely controllable can be determined by examining the algebraic condition*

$$rank[B \quad AB \quad A^2B \quad ... \quad A^{m-1}B] = m, \tag{3.6}$$

where, A is $m \times m$ and B is $m \times r$ matrix.

Generally, the concepts of controllability and observability [66, 32] play an important role in modern control systems design. In many real world situations, one or more state variables might not be possible to measure physically. For example, the output temperature of a plant might be a combination of several state variables. Although it is possible to measure the plant output temperature, the individual state variables might not be separately measurable. To overcome this infeasibility, control theory provides technique called *observer*. The observer provides an estimate of state variables from input and output parameters. Observer design is not possible in every control system. If the plant meets the observability criteria, it is observable [21, 65, 66] and an observer can be designed. Also, the controllability of a plant is a very important factor to analyze. If the plant does not meet the controllability criteria (given in Lemma 5), it is not possible to develop a stable controller to the plant. Therefore, in the controller design, the first step is to inspect the controllability [21]. If controllability fails, it is not recommend to design a controller directly to the plant [32].

Since the full exposition of classical control theory is beyond the scope of this dissertation, proofs of the above results are not provided and they can be found in any standard text on control theory [21, 65, 66].

3.6.1 Optimal Controllers

Optimal control theory is used to handle relatively non-aggressive systems. The control goal is to optimize the performance of the system for a defined time frame. The theoretical derivations for optimality can be done using Pontryagin's maximum principle (a necessary condition), or by solving the Hamilton-Jacobi-Bellman equation (a sufficient condition) [56]. The optimal control solves the problem of finding a control law for a given system while certain physical constrained are met. In optimal control problem, normally a cost function is formed. The cost function consists of state vectors and system inputs. Depending on the importance given by the designer to each state vectors, system inputs, or system outputs, the weight matrices can be decided (adjusted) for each entity. The optimal control problem can be reduced to set of differential equations describing the paths of the control variables that minimize the cost function.

3.6.2 Model Predictive Controllers (MPC)

Model predictive controllers (MPC) are classified as advanced optimal controls, that gives some advantage over non accurate plant model. In MPC, the predictive horizon means the entire time windows that the problem is defined and the control horizon defines the time window that input controls could manipulate. MPC is based on iterative, finite horizon optimization of a given model. In MPC, similar to optimal control system, a cost function is formed. The cost function consists of state vectors and system inputs. Depending on the importance given by the designer to each state vectors, system inputs, or system outputs, the weight matrices can be decided (adjusted) for each entity. In each sampling interval, the cost function is calculated and it is minimized for the entire predictive horizon. This calculation is carried out *online* to find the next position of the plant trajectories. Although in each sampling interval, the complete control trajectory is calculated for the entire control horizon, only the first step of the control strategy is implemented. Then the plant state is sampled again and the calculations are repeated in the next sampling interval. In this process the prediction horizon keeps being shifted forward the prediction horizon. Therefore, sometimes MPCs are also known as receding horizon control.

MPC controller calculates the entire input trajectory *online* at each sampling interval; when the prediction and control horizons are large, the online optimization calculation might require substantial CPU resources. Since the optimization calculation process is repeated for entire predictive horizon, the MPC require a substantial amount of calculation in each sample time. Therefore, a hard real-time system that does not have sufficiently large additional resources for *online* MPC calculation will suffer from delay effects [75, 91] due to calculation delay. On the other hand, if the highest priority is given to the MPC calculations, other hard real-time tasks will suffer. Therefore, MPC should be used for hard real-time systems with certain restrictions [56, 67, 32].

$\square$

3.7 Summary

In this chapter, we briefly explain the models, notations, and theories used in the rest of the chapters. In the next chapter, we will extend our discussion to present the thermal-resiliency on uniprocessor systems.

CHAPTER 4: THERMAL-RESILIENCY ON UNIPROCESSOR SYSTEMS

In this chapter, we introduce a new metric called *thermal-resiliency* which characterizes the maximum external thermal stress that any hard-real-time performance mode can withstand. We show how to solve some of the issues and challenges of designing predictable real-time systems in an unpredictable thermal environment where environmental temperature may dynamically change (e.g., implantable medical devices). Towards this challenge, we propose a control-theoretic design methodology which permits a system designer to specify a set of hard-real-time performance modes under which the system may operate. The system automatically adjusts the real-time performance mode based on the external thermal stress. We show (via analysis, simulations, and a hardware testbed implementation) that our control-design framework is stable and control performance is equivalent to previous real-time thermal approaches, even under dynamic temperature changes. A crucial and novel advantage of our framework over previous real-time control is the ability to guarantee hard deadlines even under transitions between modes. At the end of the chapter, we show how our system design permits the calculation of a new metric called *thermal resiliency* which characterizes the maximum external thermal stress that any hard-real-time performance mode can withstand. Also, we show how our design framework and analysis may be classified as a *thermal stress analysis* for real-time systems.

This chapter presents a methodology for designing and analyzing single core, thermal-resilient hard-real-time systems. Section 4.1 gives a brief introduction and Section 4.2 presents a high-level overview of our methodology. Section 4.3 overviews the hardware, real-time, and thermal models used throughout the chapter. Section 4.4 details the design of our thermal-resilient controller. Section 4.5 derives thermal-resiliency function Λ for control system. Section 4.6 describes the results of our comparison with previous control systems via simulation and implementation upon testbed hardware. In the next chapter, we provide formal derivations and proofs necessary to establish the hard-real-time system guarantees. Finally, Section 4.7 gives a summary of this chapter.

4.1 Introduction

Modern computer-controlled systems are often deployed in dynamic and unpredictable thermal operating environments. From the hardware-design perspective, material scientists and computer engineers use rigorous *thermal-stress analysis* techniques (e.g., see [72]) to determine how the underlying physical hardware will withstand applied internal and external thermodynamic forces. Unfortunately, equivalent analysis does not exist for determining the effects of (unpredictable) thermal stress on the performance of the systems software. While hardware capabilities such as dynamic power management (DPM) permit a computing system to reduce its power dissipation at run-time, many embedded systems have real-time constraints which may be adversely affected by unexpected changes in processor speed.

Unfortunately, no current formal real-time design and analysis framework fully addresses the above setting. Recently-proposed control-theoretic frameworks exist for regulating processor temperature for soft-real-time systems (i.e., systems where jobs are permitted to "occasionally" miss computational deadlines) in an unpredictable thermal environment [37, 36]. While their results successfully show that it is possible to obtain stable and responsive thermal behavior and system utilization control, a system designer cannot use their approaches to *a priori* determine the amount of system-performance degradation due to changes in the thermal environment. Instead, the level of degradation can only be indirectly inferred via simulations of the system for different operating conditions. Furthermore, hard timing guarantees cannot be made in these frameworks. Techniques also already exist for permitting a trade-off between real-time QoS and processing resources (e.g., the *QoS-based resource allocation model* (QRAM) [69]); however, while such techniques may guarantee real-time deadlines under a fixed level of resources, they cannot guarantee deadlines when a system must dynamically switch between real-time modes (due to the uncompleted execution remaining at mode transitions). Furthermore, none of these previously-proposed techniques can be used to obtain a precise, formal quantification of the thermal stress that the system can withstand.

In this chapter, we address the challenge of determining the real-time guarantees in the presence of unpredictable dynamic environmental conditions. Towards this goal, we propose a framework and mechanisms for thermal-stress analysis in real-time systems. Our objective is *to develop techniques that permit a system designer to specify, a priori, a precise quantification of the hard-real-time performance degradation due to external thermal events, via a new system design metric called* **real-time thermal resiliency**. Informally, real-time thermal resiliency is a prediction of the maximum external operating temperature at which a specified *real-time performance mode* (e.g., quality-of-service) may be guaranteed in the system steady-state (i.e., a time at which system properties have converged and do not change). To illustrate, consider a system with q different (system designer-defined) hard-real-time performance modes $M_0, M_1, \ldots, M_q$ where modes are ordered in increasing levels of real-time performance with M_q guaranteeing the highest level and M_0 the lowest. The real-time thermal resiliency of any mode M_i, denoted as $\Lambda(M_i, \mathcal{T}_{\text{ref}})$, is the predicted maximum external operating temperature for which the system will continue to operate (in the steady state) at performance mode M_i or higher and maintain a CPU reference temperature of $\mathcal{T}_{\text{ref}}$. Furthermore, if the external temperature exceeds $\Lambda(M_i, \mathcal{T}_{\text{ref}})$, then the system should automatically degrade to the next lowest performance mode M_{i-1}. The capability to define (at system-design time) thermal-resilient, real-time performance modes allows the system designer to specify how a system will gracefully and predictably degrade under external thermal stress; furthermore, the ability to accurately determine the real-time thermal resiliency of a performance mode provides a real-time system designer with a thermal-stress analysis framework analogous to stress analysis techniques in physical sciences and engineering. In the IMD example above, the thermal-resiliency function Λ may be used to determine (at design time) the body-temperature that a given set of tasks may safely operate at without doing damage to surrounding tissue.

4.2 Methodology Overview

We now describe at a high level the major steps of our thermal-resilient design and analysis methodology.

1. **System Hardware Specification:** In the first step, the system designer must specify the processing and DPM capabilities of the system. Throughout this paper, we will be illustrating and validating our methodology upon an Intel Pentium IV 3.0 GHz single-core processor testbed. To match the rudimentary DPM capabilities often present in embedded processors, our testbed possesses the ability to only modulate the power modes of the system between active and inactive states. Section 3.3.1 gives more detail on the hardware model and our testbed implementation details.

2. **System Software Specification:** The system designer must specify the set of valid software modes $M_0, M_1, \ldots, M_q$ for the system. In Section 3.3.2, we discuss using the sporadic task model [64] as a model for real-time workload of each software mode.

3. **Real-Time Mode Resource Allocation:** After the HW/SW specification steps, the designer must determine the minimum resource allocation under which the multi-mode system is schedulable. We discuss in Section 3.3.2 how recent techniques for schedulability analysis of hard-real-time systems where both the hardware and software change modes may be used in allocating sufficient processing time to each mode.

4. **Power/Thermal Model Evaluation:** Given the processing platform, we need an accurate power model in order to derive formal guarantees on the thermal resiliency of the system. Due to the duality between electrical and thermal circuits, we model the thermodynamics of our processing system using the resistance/capacitance (RC) circuits. We use *system identification* (SI) to identify the system parameters and evaluate the efficacy of our power-model choice. The details on the derived parameters for our hardware testbed are explained in the Section A.2 of the appendix.

5. **Control System Design:** We design a control structure based on optimal control theory. In this process, we use the SI parameters (determined in the previous step) to design the feedback gain parameters. We present details on our controller design in Section 4.4.

6. **System Simulation:** We build a system simulator which implements the real-time scheduling algorithm and control algorithm and simulates the real-time and thermal behavior of the system based on the resource allocations and power model derived in Steps 3 and 4. The details of our simulator are provided in Section 4.6.

7. **Thermal-Resiliency Function Calculation:** Given the real-time mode resource allocation, power model, controller, and simulator observations obtained from Steps 3, 4, 5, and 6 we can obtain a quantification of the thermal-resiliency function Λ. We give details on the derivation of this function in Section 4.5.

8. **System Validation:** We finally validate our system simulator and thermal-resiliency calculations in Section 4.6 by comparing directly with observations from our hardware testbed. Our comparison shows that the system simulator closely models the actual testbed behavior. Furthermore, we validate that our predicted thermal-resiliency Λ function is accurate by observing that it closely tracks the actual hardware testbed behavior.

While most of the steps above are standard practice in control system design, we would like to emphasize that our ability to ensure the hard-real-time schedulability of each mode in Step 3 and obtain *a priori* guarantees on thermal resiliency in Step 7 distinguishes our approach from previous thermal control for real-time systems.

Importantly, our proposed control framework may be considered a proactive scheduler; however, we attempt to remove some ideal assumptions by working with only two power modes and the more general sporadic task model. Also, we consider the ambient temperature changes and analyze the effects on the task system due to its variation.

4.3 Models

We consider a single processor system with rudimentary DPM capabilities of only *active* and *inactive* power modes as explained in the Subsection 3.2.1.

In the Introduction Section (4.1) of this chapter, we proposed a system model of *real-time performance modes* $M_1, \ldots, M_q$. We will assume that there is an ordering of real-time performance modes based on their "computational requirements" to meet all of a mode's deadlines. The relation $M_i \succeq M_j$ indicates that M_i is more computationally intensive than M_j. As we defined in the Subsection 3.2.3, for the purpose of this uniprocessor case, we will assume each performance mode M_i is characterized by a *sporadic task system*. Figure 3.1 illustrates the processing-time allocation in two different modes.

In our uniprocessor case, the scheduling of real-time performance mode M_i upon the thermal-aware periodic resource may be done by any uniprocessor real-time scheduling algorithm (e.g., earliest-deadline-first or rate-monotonic [57]). However, $\Theta^{(i)}$ must be sufficiently large for the scheduling algorithm to correctly schedule all jobs of the task set of M_i (i.e., $\{\tau_1^{(i)}, \tau_2^{(i)}, \ldots, \tau_{n_i}^{(i)}\}$) and (potentially) any jobs from the previous mode that have not completed by the mode change.

To obtain a proper resource allocation, $\Theta^{(i)}$, for each mode, we use our recently-developed hard-real-time schedulability test (for EDF scheduling under hardware/software mode changes in the periodic resource model) to search for a safe value of $\Theta^{(i)}$ for each mode [30] to ensure that deadlines are <u>always</u> met. The multi-modal schedulability analysis ensures that for any valid sequence of mode changes and valid set of job arrivals under the sporadic task model that the EDF scheduler will always meet all deadlines. The analysis works by determining the maximum workload carried from one mode to another and testing whether this "carry-in" will cause a deadline miss.

We use the thermal/power model defined in the Subsection 3.2.7. As we previously mentioned, our model uses the duality principle in electrical and thermal circuits to describe the dynamics of the power dissipating source using electrical resistance/capacitance (RC) circuits. Figure 4.3 shows the basic equivalent circuit for the CPU and its surrounding environment. We assume that total dissipated power of the CPU $\mathcal{P}_{\text{cpu}}$ is equal to the sum of the power due to dynamic current $\mathcal{P}_{\text{cpu}}^{\text{d}}$ and power due to leakage current $\mathcal{P}_{\text{cpu}}^{\ell}$. Furthermore, we assume that the temperature-dependant leakage power may be closely approximated by a linear function of CPU temperature [59].

4.3.1 The Testbed

As a case study of our methodology, we have built a hardware testbed using an Intel Pentium IV 3.0 GHz single core processor running a modified Linux kernel (2.6.33.7.2-rt30 PREEMPT_RT). The low power CPU on our testbench does not have a System Developer Interface to measure the on-die temperature [1] directly. We follow the procedure given in the Intel Documentation [4] and install a T-type thermocouple on the CPU die[2]. We use Phidgets 4-port temperature sensor board to measure the environment, air, and the on-die temperature through the USB driver and allows us to directly interface the sensors with the testbed software.

We develop a loadable kernel module to activate and vary the frequency modulation level at run-time. We use Model Specific Registers (MSR) to control the frequency modulation ratio in the clock and select the higher and the lowest frequency modulation indices to emulate the low and the higher power levels. We use 12.5% and 87.5% modulation ratios in the $IA32\_CLOCK\_MODULATION$ MSR for active and inactive power mode emulation.

We develop a multi-threaded application using Linux native posix thread libraries (NTPL). Our application consists of a scheduler simulator and a thread activator where the schedule simulator selects the EDF based jobs from the local ready-queue and dispatches them into a thread activator. The thread activator consists of a very high priority thread (priority is set to higher than the threaded IRQ handlers), emulates the schedule tick in the Linux kernel in higher level abstraction. Similar to the Linux kernel

[1]The Intel documentation says that a on-die sensor is present, however, they have not provided a system developer interface to measure the on-die temperature by means of software methods as opposed to latest CPU families.

[2]We mount a T-type thermocouple on the CPU die using a small penetration made by a precise milling machine as recommended by the Intel.

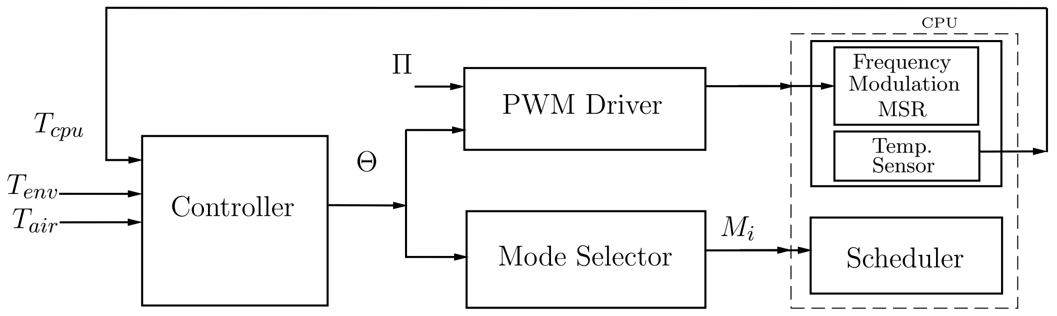


Figure 4.1: The implementation details of the testbed.

Note that the scheduler is responsible for EDF selection of jobs, activation of task threads to fill Θ and activation of idle thread during $\Pi - \Theta$, and thereby emulating the PWM cycle.

scheduler tick, the thread activator sleeps until it wakes up accurately in the scheduling boundaries. Our thread activator wakes up in unequal tick intervals to schedule jobs, raises the appropriate thread which should have the priority, and goes back to the sleeps. The jobs are selected by the schedule simulator according to EDF. This process repeats and the amount of time allocates to each job depends on EDF and the total time depends on the Θ given by the optimal controller.

Figure 4.1 provides a high-level overview of the workflow for the different components of our framework. The controller after sampling the temperatures determines the capacity. The capacity is given to the PWM controller and the real-time performance mode selector. The PWM modulates the frequency of the CPU via the MSR and an OS Scheduler (EDF) determines how to schedule the selected performance mode within the PWM duty cycle. Our temperature sensors sample the temperatures and the process iterates ad infinitum.

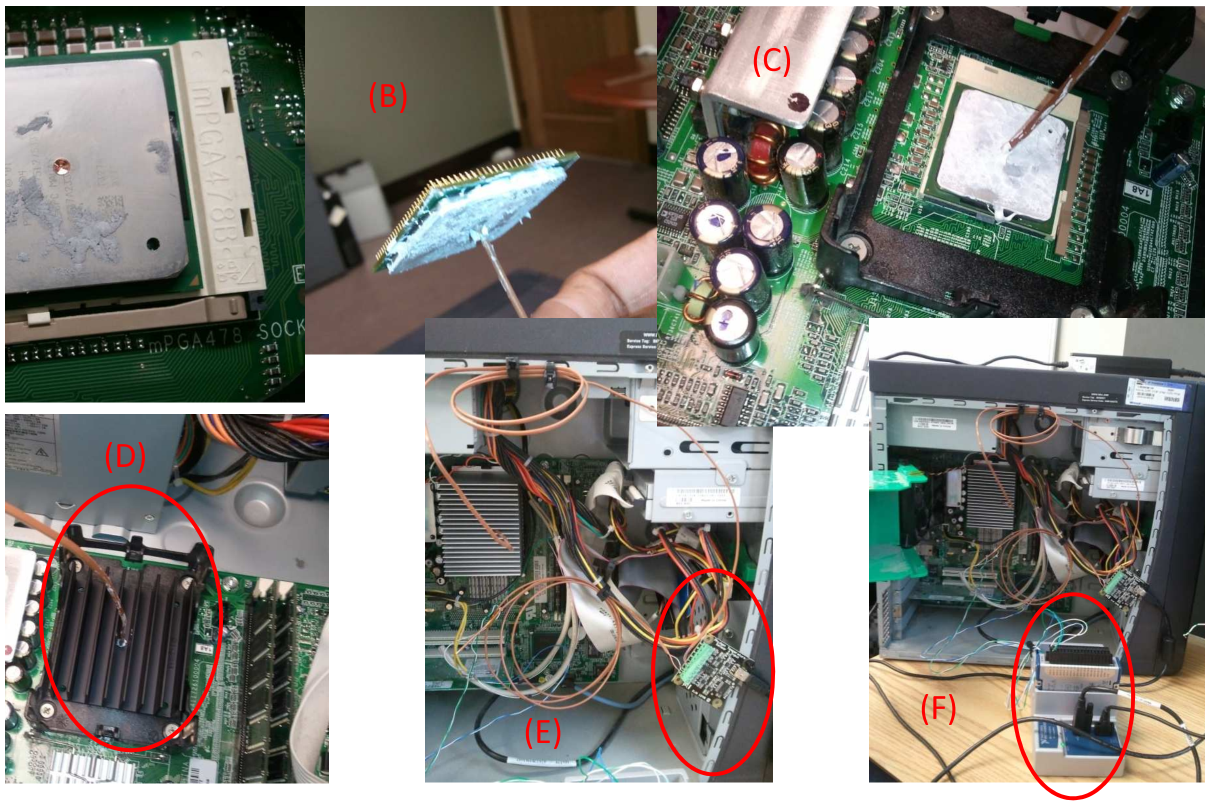


Figure 4.2: The physical testbed preparation steps.

We selected low power Intel Pentium P4 processor that has minimal power saving features. In the figure, Part A) shows the tiny impression (a hole) made on the processor heat spreader, B) shows T-type thermocouple has been mounted on the processor with heat resistant adhesive (silicon grease is shown in white color on the processor heat spreader), C) shows the processor is mounted in ZIF socket, D) shows the minimal heat-zink that we initially used to run the experiment (but failed to withstand the heat dissipation of the processor and later we changed to the regular heat sink), E) shows the Phidget 4 port thermal sensor board along with the environment temperature sensor of the testbed, and F) shows the testbed and the NI DAQ board in action.

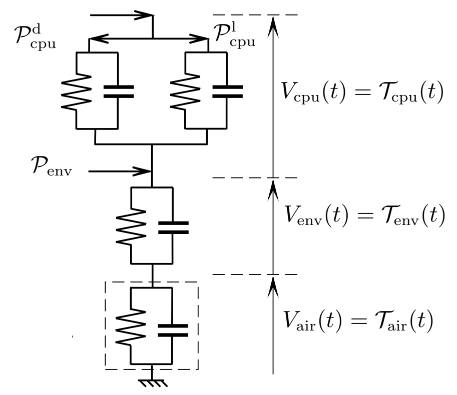

$\mathcal{P}^{\mathrm{d}}_{\mathrm{cpu}}$
$\mathcal{P}^{\mathrm{l}}_{\mathrm{cpu}}$
$V_{\mathrm{cpu}}(t) = \mathcal{T}_{\mathrm{cpu}}(t)$
$\mathcal{P}_{\mathrm{env}}$
$V_{\mathrm{env}}(t) = \mathcal{T}_{\mathrm{env}}(t)$
$V_{\mathrm{air}}(t) = \mathcal{T}_{\mathrm{air}}(t)$

Figure 4.3: The basic equivalent circuit for a working CPU and its working environment

4.3.2 Power/Thermal Derivations

We apply the Kirchhoff's circuit laws for our RC thermal model (in Figure 4.3) and get the following equations for $\mathcal{T}_{\mathrm{cpu}}(t)$,

$$\begin{aligned}\frac{\mathcal{T}_{\mathrm{cpu}}(t)}{R^{\mathrm{d}}_{\mathrm{cpu}}} + C^{\mathrm{d}}_{\mathrm{cpu}}\frac{d}{dt}\mathcal{T}_{\mathrm{cpu}}(t) &= \mathcal{P}^{\mathrm{d}}_{\mathrm{cpu}}(t) &(4.1)\\ \frac{\mathcal{T}_{\mathrm{cpu}}(t)}{R^{\mathrm{l}}_{\mathrm{cpu}}} + C^{\mathrm{l}}_{\mathrm{cpu}}\frac{d}{dt}\mathcal{T}_{\mathrm{cpu}}(t) &= \mathcal{P}^{\ell}_{\mathrm{cpu}}(t) &(4.2)\\ &= k_T\big(\mathcal{T}_{\mathrm{cpu}}(t) + \mathcal{T}_{\mathrm{env}}(t)\big) + k_C.\end{aligned}$$

Adding (4.1) and (4.2), and solving for $\frac{d}{dt}\mathcal{T}_{\mathrm{cpu}}(t)$,

$$\frac{d}{dt}\mathcal{T}_{\mathrm{cpu}}(t) = \sigma_1\Big(k_T - \frac{1}{R^{\mathrm{l}}_{\mathrm{cpu}}} - \frac{1}{R^{\mathrm{d}}_{\mathrm{cpu}}}\Big)\mathcal{T}_{\mathrm{cpu}}(t) + k_T\sigma_1\mathcal{T}_{\mathrm{env}}(t) + \sigma_1\mathcal{P}^{\mathrm{d}}_{\mathrm{cpu}}(t) + \sigma_1 k_C. \qquad (4.3)$$

We obtain the following equation for $\mathcal{T}_{\mathrm{env}}(t)$,

$$\frac{\mathcal{T}_{\mathrm{env}}(t)}{R_{\mathrm{env}}} + C_{\mathrm{env}}\frac{d}{dt}\mathcal{T}_{\mathrm{env}}(t) = \mathcal{P}_{\mathrm{cpu}}(t) + \mathcal{P}_{\mathrm{env}}(t) = \mathcal{P}^{\mathrm{d}}_{\mathrm{cpu}}(t) + \mathcal{P}^{\ell}_{\mathrm{cpu}}(t) + \mathcal{P}_{\mathrm{env}}(t). \qquad (4.4)$$

Solving (4.4) for $\frac{d}{dt}\mathcal{T}_{\text{env}}(t)$,

$$\frac{d}{dt}\mathcal{T}_{\text{env}}(t) = \frac{k_T}{C_{\text{env}}}\mathcal{T}_{\text{cpu}}(t) + \frac{1}{C_{\text{env}}}\mathcal{P}^{\text{d}}_{\text{cpu}}(t) + \frac{1}{C_{\text{env}}}\mathcal{P}_{\text{env}}(t) + \Big(\frac{k_T}{C_{\text{env}}} - \frac{1}{R_{\text{env}}C_{\text{env}}}\Big)\mathcal{T}_{\text{env}}(t) + \frac{k_C}{C_{\text{env}}} \quad (4.5)$$

If we know the temperature of the environment and CPU at some initial time $t_0 \leq t$, then we can derive following Equations[3] from (4.3) and (4.5):

$$\mathcal{T}_{\text{cpu}}(t) = \int_{t_0}^{t} \sigma_1 \mathcal{P}_{\text{cpu}}(s)e^{-(t-s)\beta_1}ds + \mathcal{T}_{\text{cpu}}(t_0)e^{-(t-t_0)\beta_1}, \quad (4.6)$$

$$\mathcal{T}_{\text{env}}(t) = \int_{t_0}^{t} \sigma_2\Big(\mathcal{P}_{\text{env}}(s) + \mathcal{P}_{\text{cpu}}(s)\Big)e^{-(t-s)\beta_2}ds + \mathcal{T}_{\text{env}}(t_0)e^{-(t-t_0)\beta_2}. \quad (4.7)$$

where

$$\begin{aligned} \beta_1 &\overset{\text{def}}{=} \Big(\frac{1}{R^{\text{d}}_{\text{cpu}}} + \frac{1}{R^{\text{l}}_{\text{cpu}}} - k_T\Big) \cdot \frac{1}{(C^{\text{d}}_{\text{cpu}} + C^{\text{l}}_{\text{cpu}})}, \\ \beta_2 &\overset{\text{def}}{=} \frac{1}{R_{\text{env}}C_{\text{env}}} - \frac{k_T}{C_{\text{env}}}, \text{ and} \\ \sigma_2 &\overset{\text{def}}{=} \frac{1}{C_{\text{env}}}. \end{aligned}$$

According to the Figure 4.3 shown above, the absolute CPU temperature can be calculated as $\mathcal{T}_{\text{cpu}}(t) + \mathcal{T}_{\text{env}}(t) + \mathcal{T}_{\text{air}}(t)$.

4.4 Controller Design

In this research, we use the standard state-space model to represent continuous-time (ideal) system introduced in the Section 3.6. Our design process is two fold. In Section 4.4.1, we design a thermal controller assuming that an ideal system with continuous power modes; this assumption is not practically viable. Therefore, in Section 4.4.2, we will extend the controller design to a processor with only active/inactive power modes, which is more practical and can be implemented in a real-world system.

[3]Assume that the leakage current mostly depends on the $\mathcal{T}_{\text{cpu}}$ and $\mathcal{T}_{\text{env}}$ effect on the leakage current is negligible.

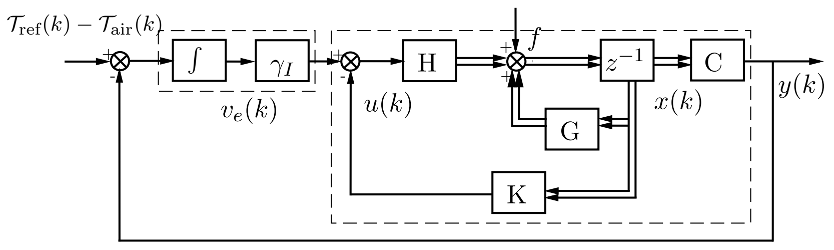

$\mathcal{T}_{\mathrm{ref}}(k) - \mathcal{T}_{\mathrm{air}}(k)$
$\int$
γ_I
$v_e(k)$
$u(k)$
H
f
z^{-1}
C
$x(k)$
$y(k)$
G
K

Figure 4.4: The thermal control design with state feedback and integral actuator

4.4.1 Continuous Power Modes

As a first step towards our goal of designing a control-theoretic framework for thermal stress analysis, we employ *linear quadratic (LQ) optimal control* for real-time thermal management. Our design consists of an optimal state feedback and an integrator that regulates the dynamics of the system. An LQ controller enables us to design an efficient and low-overhead controller, derive the feedback parameters before run-time (used in thermal-resiliency analysis), and smoothly track our reference input. In the future, we plan on applying more complex and robust controllers (e.g., $\mathcal{H}_\infty$ controllers) to decrease the controller's sensitivity to modeling inaccuracy and noise. However, as observed in the simulations and experiments of Section 4.6, our current LQ design is appropriately responsive to changes in environmental temperature.

In our system model, we specify the thermal power of the CPU as the control to the system. The controller is designed to follow the temperature reference, $\mathcal{T}_{\mathrm{ref}}$. In our design, we consider $\mathcal{T}_{\mathrm{cpu}}(t)$ as one of the variable to be controlled and $\mathcal{P}^{\mathrm{d}}_{\mathrm{cpu}}(t)$ as a manipulated variable (equivalent to $y(t)$ and $u(t)$, respectively, in continuous state-space model). The basic control structure is given in Figure 5.3.

From Equations (4.3) and (4.5), the continuous-time state space model can be written as

$$\begin{bmatrix} \dot{\mathcal{T}}_{\mathrm{cpu}}(t) \\ \dot{\mathcal{T}}_{\mathrm{env}}(t) \end{bmatrix} = \begin{bmatrix} -\beta_1 & k_T\sigma_1 \\ k_T\sigma_2 & -\beta_2 \end{bmatrix} \begin{bmatrix} \mathcal{T}_{\mathrm{cpu}}(t) \\ \mathcal{T}_{\mathrm{env}}(t) \end{bmatrix} + \begin{bmatrix} \sigma_1 \\ \sigma_2 \end{bmatrix} \mathcal{P}^{\mathrm{d}}_{\mathrm{cpu}}(t) + \begin{bmatrix} 0 \\ \sigma_2 \end{bmatrix} \mathcal{P}_{\mathrm{env}}(t). \tag{4.8}$$

While our analysis below is in the continuous-time domain, a discrete-time control system approach would be applied in an actual computer implementation. Therefore, we now note that we may easily convert the continuous-state space model to the discrete-time sampled system, $x(k+1) = Gx(k) + Hu(k) + f$ from the continuous-time state matrices $A = \begin{bmatrix} -\beta_1 & k_T\sigma_1 \\ k_T\sigma_2 & -\beta_2 \end{bmatrix}$ and $B = \begin{bmatrix} \sigma_1 \\ \sigma_2 \end{bmatrix}$ where k is the sampling index, T_s is sampling interval, and G and H can be calculated as described in Section 3.6. Furthermore, the matrices A and B satisfy the condition of Lemma 5 implying that the continuous system is completely controllable. For our given system, $x(k) \equiv \begin{bmatrix} \mathcal{T}_{\text{cpu}}(k) \\ \mathcal{T}_{\text{env}}(k) \end{bmatrix}$ and $u(k) \equiv \begin{bmatrix} \mathcal{P}_{\text{cpu}}(k) \end{bmatrix}$ where we are again abusing notation for the $\mathcal{T}$ and $\mathcal{P}$ functions.

To eliminate steady state tracking error, we design our control system with an integrator. Define an additional error vector $v_e(t)$ in continuous time as,

$$\begin{aligned} v_e(t) &\stackrel{\text{def}}{=} \int_0^t (\mathcal{T}_{\text{ref}} - \mathcal{T}(t) - \mathcal{T}_{\text{air}}(t))dt \\ \dot{v}_e(t) &\stackrel{\text{def}}{=} \mathcal{T}_{\text{ref}} - \mathcal{T}_{\text{air}}(t) - \mathcal{T}(t) \\ &= -C \begin{bmatrix} \mathcal{T}_{\text{cpu}}(t) \\ \mathcal{T}_{\text{env}}(t) \end{bmatrix} + \mathcal{T}_{\text{ref}} - \mathcal{T}_{\text{air}}(t) \end{aligned} \tag{4.9}$$

where $C = [1, 1]$.

Then, the system input is calculated with a gain $K_o = [\gamma_1, \gamma_2]$ and integral constant γ_I in the following equation.

$$\begin{aligned} \mathcal{P}^{\text{d}}_{\text{cpu}}(t) &= -K_o \begin{bmatrix} \mathcal{T}_{\text{cpu}}(t) \\ \mathcal{T}_{\text{env}}(t) \end{bmatrix} + \gamma_I v_e(t) \\ &= -\Big((\gamma_1)\mathcal{T}_{\text{cpu}}(t) + (\gamma_2)\mathcal{T}_{\text{env}}(t)\Big) + \gamma_I \int_0^t (\mathcal{T}_{\text{ref}} - \mathcal{T}_{\text{air}}(t) - \mathcal{T}_{\text{cpu}}(t) - \mathcal{T}_{\text{env}}(t))dt. \end{aligned} \tag{4.10}$$

We employ standard techniques from optimal control theory to derive K_o and γ_I and prove stability. In our derivation of system stability, we use the following two results which can be found in any standard text on control theory [21, 65, 66].

We derive the augmented model that is used to obtain the optimality of the system. Consider an instance where system is completely stable and has reached steady state. We denote the input, states, and the integrator error (described in Equation (4.9)) of this special instance of the system by $\mathcal{P}_{\text{cpu}}(t_\infty))$, $\mathcal{T}_{\text{cpu}}(t_\infty), \mathcal{T}_{\text{env}}(t_\infty)$ and $v_e(t)$ respectively. Therefore,

$$\begin{bmatrix} \dot{\mathcal{T}}_{\text{cpu}}(t_\infty) \\ \dot{\mathcal{T}}_{\text{env}}(t_\infty) \end{bmatrix} = \begin{bmatrix} -\beta_1 & k_T\sigma_1 \\ k_T\sigma_2 & -\beta_2 \end{bmatrix} \begin{bmatrix} \mathcal{T}_{\text{cpu}}(t_\infty) \\ \mathcal{T}_{\text{env}}(t_\infty) \end{bmatrix} + \begin{bmatrix} \sigma_1 \\ \sigma_2 \end{bmatrix} \mathcal{P}^{\text{d}}_{\text{cpu}}(t_\infty) + \begin{bmatrix} 0 \\ \sigma_2 \end{bmatrix} \mathcal{P}_{\text{env}}(t_\infty). \tag{4.11}$$

From the Equation (4.8) and Equation (4.11) we get,

$$\begin{bmatrix} \dot{\mathcal{T}}_{\text{cpu}}(t) - \dot{\mathcal{T}}_{\text{cpu}}(t_\infty) \\ \dot{\mathcal{T}}_{\text{env}}(t) - \dot{\mathcal{T}}_{\text{env}}(t_\infty) \end{bmatrix} = \begin{bmatrix} -\beta_1 & k_T\sigma_1 \\ k_T\sigma_2 & -\beta_2 \end{bmatrix} \begin{bmatrix} \mathcal{T}_{\text{cpu}}(t) - \mathcal{T}_{\text{cpu}}(t_\infty) \\ \mathcal{T}_{\text{env}}(t) - \mathcal{T}_{\text{env}}(t_\infty) \end{bmatrix} + \begin{bmatrix} \sigma_1 \\ \sigma_2 \end{bmatrix} (\mathcal{P}^{\text{d}}_{\text{cpu}}(t) - \mathcal{P}^{\text{d}}_{\text{cpu}}(t_\infty)). \tag{4.12}$$

Also, from the Equation (4.9), we get,

$$\dot{v}_e(t) - \dot{v}_e(t_\infty) = -C \begin{bmatrix} \mathcal{T}_{\text{cpu}}(t) - \mathcal{T}_{\text{cpu}}(t_\infty) \\ \mathcal{T}_{\text{env}}(t) - \mathcal{T}_{\text{env}}(t_\infty) \end{bmatrix}. \tag{4.13}$$

Now, combining the Equation (4.12) and (4.13), we define our higher order system as,

$$\dot{e}(t) = \hat{A}e(t) + \hat{B}u_e(t), \tag{4.14}$$

where,

$$\begin{aligned} e(t) &= \begin{bmatrix} \mathcal{T}_{\text{cpu}}(t) - \mathcal{T}_{\text{cpu}}(t_\infty) \\ \mathcal{T}_{\text{env}}(t) - \mathcal{T}_{\text{env}}(t_\infty) \\ v_e(t) - v_e(t_\infty) \end{bmatrix}, \\ u_e(t) &= \mathcal{P}^{\text{d}}_{\text{cpu}}(t) - \mathcal{P}^{\text{d}}_{\text{cpu}}(t_\infty), \\ \hat{A} &= \begin{bmatrix} A & 0 \\ -C & 0 \end{bmatrix}, \text{ and} \\ \hat{B} &= \begin{bmatrix} B & 0 \end{bmatrix}^T. \end{aligned}$$

We select the feedback gain $\hat{\gamma}$ such that,

$$u_e(t) \quad = \quad -\hat{K}e(t), \tag{4.15}$$

where,

$$\hat{K} = \begin{bmatrix} K_o \\ -\gamma_I \end{bmatrix}^T. \tag{4.16}$$

The above state-space and the control gain parameters are valid for a continuous-time controller. So, we may obtain the discrete-time state-space matrices for the augmented model (i.e, G and H) from $\hat{A}$ and $\hat{B}$ via the transformation described after Equation (3.5). In LQ optimal control, the objective is to design the controller to minimize some performance index. A standard LQ performance index is given by

$$J \overset{\text{def}}{=} \frac{1}{2}\sum_{k=0}^{\infty}\big(e(k)^TQe(k) + u_e^T(k)Ru_e(k)\big), \tag{4.17}$$

where Q and R are arbitrary symmetric matrices of size $m \times m$ and $r \times r$ such that $Q \geq 0$ (positive semi definite), $R > 0$ (positive definite). (In our system given in Equation (4.8), m is two and r is one). It is easy to show that for a Linear Time Invariant (LTI) system, (Refer to [66]), the optimal state feedback can be obtained as,

$$u_e(k) = -\hat{K}e(k), \tag{4.18}$$

where $\hat{K}$ is the feedback gain defined as

$$\hat{K} = (R + H^TPH)^{-1}H^TPG, \tag{4.19}$$

and where P is the positive definite solution of the algebraic Riccati equation below,

$$P = Q + G^TPG - G^TPH(R + H^TPH)^{-1}H^TPG.$$

From the above, it may be shown [66] that the optimal performance index can be calculated as $J_{min} = \frac{1}{2}e^T(0)Pe(0)$.

It is well known [66] that the feedback control (i.e., $\hat{K}$) results in an asymptotically stable closed-loop system according to Lemma 2. Obviously, stable choices of K_o and γ_I for the original (non-augmented) system can be immediately obtained from the derived $\hat{K}$.

4.4.2 Active/Inactive Power Modes

Since the CPU power cannot be varied continuously, the controller designed in the previous section cannot be directly applied to the setting of discrete active/inactive power modes. In this section, we extend the design of the continuous power modes controller described in the previous section to the active/inactive power mode setting by applying pulse-width modulation (PWM) techniques. Recall in Section 4.3 that we stated the active/inactive power modes will be modeled via the thermal-aware periodic resource model with parameters Π and Θ. Thus, to control the system via this model, we must choose the appropriate values of Π and Θ. The Π value is a design parameter which may be chosen at controller design-time and will be assumed fixed throughout controller execution. Typically, a smaller value of Π will increase the system schedulability; however, a larger value of Π will decrease the overhead potentially incurred by switching between the active and inactive power modes. (See [5] for algorithms for determining Π in the thermal setting). The only constraint that our framework places on the chosen value of Π is that it must evenly divide the sampling interval length T_s (i.e., $T_s = \kappa\Pi$ for some $\kappa \in \mathbb{N}^+$).

Since we have only two power modes, we cannot arbitrarily set the power level. However, we may change the assigned resource capacity between sampling periods to approximate arbitrary power levels. Therefore, the assigned resource capacity will be the manipulated variable in our PWM system. The periodic resource capacity ($\Theta(k)$) and resource period (Π) can be respectively viewed as the pulse duration and duty cycle of the PWM. Let $\Theta(k)$ denote the value of the resource capacity over the k'th sampling period. For determining the $\Theta(k)$ value, we use a method based on the *principle of equivalent areas* (PEA) for converting any arbitrary input signal into an equivalent PWM signal [39]. First, note that in a discrete-time system using *zero-order hold* (ZOH), the input signal is held constant over the sampling period. Specifically, for the k'th sampling interval, the input $\mathcal{P}^{\text{d}}_{\text{cpu}}(k)$ is held over the T_s-length interval, resulting in a total energy dissipation of $T_s \cdot \mathcal{P}^{\text{d}}_{\text{cpu}}(k)$ over the interval. To get the equivalent area (i.e., energy) as the (ideal) system with continuous power modes, we must set $\Theta(k)$ such that the periodic modulations between the power modes of $\mathcal{P}_{\text{act}}$ and $\mathcal{P}_{\text{inc}}$ dissipate the equivalent amount of energy over the T_s-length interval. Figure 4.5 illustrates the area equivalence between the continuous and PWM controllers. It is

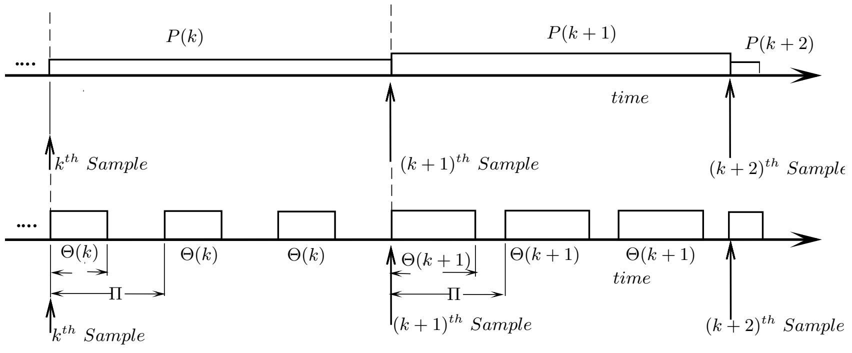


Figure 4.5: The simplified power and modulation relationship

easy to see that a smaller T_s gives a better PWM approximation. However, our controller needs to follow a system with relatively slower (thermal) dynamics. Thus, for efficiency, we select relatively larger T_s and higher κ value. Also, even under varying air and environmental conditions, the the resource capacity, Θ does not change rapidly due to slower system dynamic. Therefore, the same mode will continue to hold over several sampling periods before change occurs. Furthermore, in the steady state (when the environment or air temperature does not change much), the system will change modes very infrequently.

More formally, we may derive the following relationship between $\mathcal{P}^{\text{d}}_{\text{cpu}}(k)$ and $\Theta(k)$,

$$\begin{aligned}
\kappa\Pi\mathcal{P}^{\text{d}}_{\text{cpu}}(k) &= \kappa\Big(e_{\text{act}} + \int_0^{\Theta(k)} \mathcal{P}_{\text{act}}dt + e_{\text{inc}} + \int_{\Theta(k)}^{\Pi} \mathcal{P}_{\text{inc}}dt\Big) \\
\Rightarrow \mathcal{P}^{\text{d}}_{\text{cpu}}(k) &= \left(\frac{\mathcal{P}_{\text{act}} - \mathcal{P}_{\text{inc}}}{\Pi}\right)\Theta(k) + \mathcal{P}_{\text{inc}} + \frac{1}{\Pi}(e_{\text{act}} + e_{\text{inc}}). \qquad (4.20)
\end{aligned}$$

The PWM controller pseudocode is presented in Algorithm 1. The controller proposed here consists of two integrated operations: the thermal controller and the PWM modulator. The first step is to obtain the CPU temperature at t_ℓ (Line 3 of Algorithm 1). The error is then calculated by taking the difference between the reference temperature and the CPU temperature (Line 4). The error is integrated into the error vector and added to vector sum of the integrated error in the next line (Line 5). After which, the power

Algorithm 1 Control Algorithm

```
Require: Reference Temperature $\mathcal{T}_{\text{ref}}$; Feedback Gain $K \equiv [\gamma_1, \gamma_2]$; Integral Constant $\gamma_I$; PWM Period $\Pi$; Number of PWM periods in a sampling period $\kappa$.
1:  while At beginning of sampling period $[t_\ell, t_{\ell+1}) : t_\ell \equiv \kappa\ell\Pi$ do
2:      Sample $\mathcal{T}_{\text{cpu}}(t_\ell) + \mathcal{T}_{\text{env}}(t_\ell) + \mathcal{T}_{\text{air}}(t_\ell)$.
3:      $\dot{v}_e(t_\ell) = \mathcal{T}_{\text{ref}} - (\mathcal{T}_{\text{cpu}}(t_\ell) + \mathcal{T}_{\text{env}}(t_\ell) + \mathcal{T}_{\text{air}}(t_\ell))$
4:      $Tot\_\dot{v}_e(t_\ell) = Tot\_\dot{v}_e(t_{\ell-1}) + \gamma_I \kappa \Pi \frac{\left(\dot{v}_e(t_\ell) + \dot{v}_e(t_{\ell-1})\right)}{2}$
5:      $\mathcal{P}_{\text{cpu}}(t_\ell) = \left(Tot\_\dot{v}_e(t_\ell) - \left(\gamma_1 \mathcal{T}_{\text{cpu}}(t_\ell) + \gamma_2 \mathcal{T}_{\text{env}}(t_\ell)\right)\right)$
6:      $\Theta(t_\ell) = \min\left(\Pi \times \frac{(\mathcal{P}_{\text{cpu}}(t_\ell) - \mathcal{P}_{\text{inc}})}{\mathcal{P}_{\text{act}} - \mathcal{P}_{\text{inc}}}, \Pi\right)$
7:      $i = \max\{j \in \mathbb{Z}_{q+1} \mid \Theta^{(j)} \leq \Theta(t_\ell)\}$
8:      Update real-time performance mode to $M_i$.
9:      Set PWM to operate at period of $\Pi$ and width of $\Theta(t_\ell)$.
10: end while
```

input is calculated (Line 6) and the equivalent Θ is calculated from the property of Equation (4.20) (Line 8). Finally, the appropriate mode is selected (Line 9), the mode change is performed (Line 11), and the pulse-width modulator is invoked for the next κ Π-length intervals (Line 12). It is important to note that $\Theta(t_\ell)$ calculated in Line 8 does not have to be equal the $\Theta^{(j)}$ for the selected mode; we must only select the highest mode with $\Theta^{(j)} \leq \Theta(t_\ell)$. (If $\Theta(t_\ell)$ is larger, we are only giving the mode more processing than it requires.) It should also be observed that all operations, except for finding the appropriate mode, may be done in $O(1)$ time. Finding the highest real-time performance mode that may execute can be done in $O(\lg q)$ time (via binary search) where q is the number of real-time performance modes.

4.5 Thermal-Resiliency Calculation

In this section, we explain how to derive the real-time thermal resiliency $\Lambda(M_i, \mathcal{T}_{\text{ref}})$ for a given real-time performance mode M_i and reference temperature $\mathcal{T}_{\text{ref}}$. Assuming a steady-state error of zero, we will now briefly outline how to obtain a solution for $\Lambda(M_i, \mathcal{T}_{\text{ref}})$.[4] Assume that we have reached the steady-state by the $(k-1)$'th sampling period. Therefore, $\mathcal{T}_{\text{cpu}}(k) = \mathcal{T}_{\text{cpu}}(k-1)$, $\mathcal{T}_{\text{env}}(k) = \mathcal{T}_{\text{env}}(k-1)$, $\mathcal{T}_{\text{air}}(k) = \mathcal{T}_{\text{air}}(k-1)$, and $\Theta(k) = \Theta(k-1)$. Substituting the temperature equalities into Equations (4.6) and (4.7) allows us to solve for $\mathcal{T}_{\text{cpu}}(k)$ and $\mathcal{T}_{\text{env}}(k)$ to obtain a function of $\mathcal{T}_{\text{air}}(k)$, $\mathcal{T}_{\text{ref}}$, and $\Theta(k)$. Since

[4]The approach may be generalized when there is bounded steady-state error. However, the approach will be similar, and we omit the details due to space.

we are interested in obtaining $\Lambda(M_i, \mathcal{T}_{\text{ref}})$, we may fix $\mathcal{T}_{\text{ref}}$ and $\Theta(k) = \Theta^{(i)}$ Since the steady-state error is zero, we also have

$$\mathcal{T}_{\text{ref}} \quad = \quad \mathcal{T}_{\text{cpu}}(k) + \mathcal{T}_{\text{env}}(k) + \mathcal{T}_{\text{air}}(k). \tag{4.21}$$

Combining Equation 4.21 with the function of $\mathcal{T}_{\text{air}}(k)$ obtained from $\mathcal{T}_{\text{cpu}}(k)$ and $\mathcal{T}_{\text{env}}(k)$ allows us to solve for $\mathcal{T}_{\text{air}}(k)$. Thus, solving the entire system results in a value for $\mathcal{T}_{\text{env}}(k) + \mathcal{T}_{\text{air}}(k)$ (i.e., value of $\Lambda(M_i, \mathcal{T}_{\text{ref}})$). The resulting expression is quite complicated as it requires solutions to second-order inhomogeneous equations.

We first calculate the $\mathcal{T}_{\text{cpu}}$ as follows,

$$\begin{aligned}\mathcal{T}_{\text{cpu}}((\zeta\kappa + \kappa)\Pi) \quad &= \quad \mathcal{T}_{\text{cpu}}(\zeta\kappa\Pi) + \sum_{i=0}^{\kappa-1}\sum_{j=1}^{2}\Big(\mathcal{C}_{(j)_{inc}}((\zeta\kappa + i)\Pi + \Theta)(e^{r_{(j)}(\Pi-\Theta)} - 1) \\ &+ \quad \mathcal{C}_{(j)_{act}}((\zeta\kappa + i)\Pi)(e^{r_{(j)}\Theta} - 1)\Big).\end{aligned} \tag{4.22}$$

At the stability, $\mathcal{T}_{\text{cpu}}(\zeta\kappa\Pi)$ stays at a steady value and therefore, $\mathcal{T}_{\text{cpu}}((\zeta\kappa + \kappa)\Pi)$ and $\mathcal{T}_{\text{cpu}}(\zeta\kappa\Pi)$ are the same. Further, if the CPU does not vary the temperature within a single sampling period, the CPU should maintain the same temperature in each resource period Π intervals (For same Θ, same $\mathcal{T}_{\text{cpu}}$ and $\mathcal{T}_{\text{env}}$ temperature at successive stages). Therefore, we consider the CPU temperature for two adjacent resource periods and conclude,

$$\sum_{j=1}^{2}\Big(\mathcal{C}_{(j)_{inc}}(\zeta\kappa\Pi + \Theta)(e^{r_{(j)}(\Pi-\Theta)} - 1) + \mathcal{C}_{(j)_{act}}(\zeta\kappa\Pi)(e^{r_{(j)}\Theta} - 1)\Big) = 0, \tag{4.23}$$

because, $\mathcal{T}^{\text{inc}}_{\text{cpu}}((\zeta\kappa + 1)\Pi) = \mathcal{T}^{\text{act}}_{\text{cpu}}(\zeta\kappa\Pi)$ as per to the above argument. Then we further simplify the Equation (4.23) as follows,

$$\begin{aligned}&\Rightarrow \quad \mathcal{T}_{\text{env}}(\zeta\kappa\Pi + \Theta)\Big(\mathcal{P}_4(\Theta)\Big) + \mathcal{T}_{\text{cpu}}(\zeta\kappa\Pi + \Theta)\Big(\mathcal{P}_3(\Theta)\Big) + \mathcal{T}_{\text{cpu}}(\zeta\kappa\Pi)\Big(\mathcal{P}_1(\Theta)\Big) \\ &+ \quad \mathcal{T}_{\text{env}}(\zeta\kappa\Pi)\Big(\mathcal{P}_2(\Theta)\Big) + \mathcal{P}_A(\Theta) = 0\end{aligned} \tag{4.24}$$

where,

$$\begin{aligned}
\mathcal{P}_4(\Theta) &= \big(\mathcal{G}_4(e^{r_1(\Pi-\Theta)}-1)+\mathcal{G}_8(e^{r_2(\Pi-\Theta)}-1)\big),\\
\mathcal{P}_3(\Theta) &= \mathcal{T}_{\text{cpu}}(\zeta\kappa\Pi+\Theta)\big(-\mathcal{G}_3(e^{r_1(\Pi-\Theta)}-1)-\mathcal{G}_7(e^{r_2(\Pi-\Theta)}-1)\big),\\
\mathcal{P}_1(\Theta) &= \mathcal{T}_{\text{cpu}}(\zeta\kappa\Pi)\big(-\mathcal{G}_1(e^{r_1(\Theta)}-1)-\mathcal{G}_5(e^{r_2(\Theta)}-1)\big),\\
\mathcal{P}_2(\Theta) &= \mathcal{T}_{\text{env}}(\zeta\kappa\Pi)\big(\mathcal{G}_2(e^{r_1(\Pi-\Theta)}-1)+\mathcal{G}_6(e^{r_2(\Theta)}-1)\big),\ \text{and}\\
\mathcal{P}_A(\Theta) &= \mathcal{G}_A(e^{r_1\Theta}-1)+\mathcal{G}_B(e^{r_1(\Pi-\Theta)}-1)+\mathcal{G}_C(e^{r_2\Theta}-1)+\mathcal{G}_D(e^{r_2(\Pi-\Theta)}-1).
\end{aligned}$$

In Equation (4.24), we use the definitions of $\mathcal{C}$ for $\zeta\kappa\Pi$ and $(\zeta\kappa\Pi+\Theta)$ time instances as shown below for $i\in\{1,2\}$. Define $\bar{i}=3-i$ (equal two if i equals one and one if i equals two).

$$\begin{aligned}
\mathcal{C}_{i_{act}}(\zeta\kappa\Pi) &= \frac{(-1)^{(i)}}{r_2-r_1}\left(\begin{array}{c} r_{\bar{i}}\mathcal{C}_{3_{inc}}(\zeta\kappa\Pi)\\ +\sigma_1\left(\mathcal{P}_{\text{act}}+k_C+k_T\mathcal{T}_{\text{env}}(\zeta\kappa\Pi)\right)\\ -\left(\beta_1+r_{\bar{i}}\right)\mathcal{T}_{\text{cpu}}(\zeta\kappa\Pi)\end{array}\right)\\
&= \frac{(-1)^{(i)}}{r_2-r_1}\left(\ \mathcal{G}_{A_i}+\mathcal{G}_B\mathcal{T}_{\text{env}}(\zeta\kappa\Pi)-\mathcal{G}_{C_i}\mathcal{T}_{\text{cpu}}(\zeta\kappa\Pi)\ \right),\\
\mathcal{C}_{i_{inc}}(\zeta\kappa\Pi+\Theta) &= \frac{(-1)^{i}}{r_2-r_1}\left(\begin{array}{c} r_{\bar{i}}\mathcal{C}_{3_{inc}}(\zeta\kappa\Pi+\Theta)\\ +\sigma_1\left(\mathcal{P}_{\text{inc}}+k_C+k_T\mathcal{T}_{\text{env}}(\zeta\kappa\Pi+\Theta)\right)\\ -\left(\beta_1+r_{\bar{i}}\right)\mathcal{T}_{\text{cpu}}(\zeta\kappa\Pi+\Theta)\end{array}\right)\\
&= \frac{(-1)^{i}}{r_2-r_1}\left(\ \mathcal{G}_{A_i}+\mathcal{G}_B\mathcal{T}_{\text{env}}(\zeta\kappa\Pi+\Theta)-\mathcal{G}_{C_i}\mathcal{T}_{\text{cpu}}(\zeta\kappa\Pi+\Theta)\ \right), \qquad (4.25)
\end{aligned}$$

where,

$$\begin{aligned}
\mathcal{G}_{A_i} &= r_{\bar{i}}\mathcal{C}_{3_{inc}}(\zeta\kappa\Pi)+\sigma_1\left(\mathcal{P}_{\text{act}}+k_C\right),\\
\mathcal{G}_B &= \sigma_1 k_T,\ \text{and}\\
\mathcal{G}_{C_i} &= \beta_1+r_{\bar{i}}.
\end{aligned}$$

Similarly, from the Equation (A.10) in Appendix A, we can show that [5],

[5]Derivation of the Equation A.10 is not shown here, given in the appendix to keep the focus of the discussion.

$$\begin{aligned}\Rightarrow\quad & \mathcal{T}_{\text{env}}(\zeta\kappa\Pi+\Theta)\Big(\mathcal{J}_4(\Theta)\Big)+\mathcal{T}_{\text{cpu}}(\zeta\kappa\Pi+\Theta)\Big(\mathcal{J}_3(\Theta)\Big)+\mathcal{T}_{\text{cpu}}(\zeta\kappa\Pi)\Big(\mathcal{J}_1(\Theta)\Big)\\ +\quad & \mathcal{T}_{\text{env}}(\zeta\kappa\Pi)\Big(\mathcal{J}_2(\Theta)\Big)+\mathcal{J}_A(\Theta)=0 \qquad (4.26)\end{aligned}$$

where,

$$\begin{aligned}\mathcal{J}_4(\Theta) &= \mathcal{G}_B\big((e^{r_1(\Pi-\Theta)}-1)(\frac{r_1+\beta_1}{k_T\sigma_1})+(e^{r_2(\Pi-\Theta)}-1)(\frac{r_2+\beta_1}{k_T\sigma_1})\big),\\ \mathcal{J}_3(\Theta) &= \big(-\mathcal{G}_{C_1}(e^{r_1(\Pi-\Theta)}-1)(\frac{r_1+\beta_1}{k_T\sigma_1})-\mathcal{G}_{C_2}(e^{r_2(\Pi-\Theta)}-1)(\frac{r_2+\beta_1}{k_T\sigma_1})\big),\\ \mathcal{J}_1(\Theta) &= \big(-\mathcal{G}_{C_1}(e^{r_1(\Theta)}-1)(\frac{r_1+\beta_1}{k_T\sigma_1})-\mathcal{G}_{C_2}(e^{r_2(\Theta)}-1)(\frac{r_2+\beta_1}{k_T\sigma_1})\big),\\ \mathcal{J}_2(\Theta) &= \mathcal{G}_B\big((e^{r_1(\Pi-\Theta)}-1)(\frac{r_1+\beta_1}{k_T\sigma_1})+(e^{r_2(\Theta)}-1)(\frac{r_2+\beta_1}{k_T\sigma_1})\big), \text{ and}\\ \mathcal{J}_A(\Theta) &= \mathcal{G}_{A_1}(\frac{r_1+\beta_1}{k_T\sigma_1})(e^{r_1\Theta}+e^{r_1(\Pi-\Theta)}-2)+\mathcal{G}_{A_2}(\frac{r_2+\beta_1}{k_T\sigma_1})(e^{r_2\Theta}+e^{r_2(\Pi-\Theta)}-2).\end{aligned}$$

Furthermore, we consider a CPU temperature for $(\zeta\kappa\Pi, \zeta\kappa\Pi+\Theta]$ within the stability region and find the following relationship from the Equation (A.7) in Appendix A[6],

$$\mathcal{T}^{\text{act}}_{\text{cpu}}(\zeta\kappa\Pi+\Theta) = \mathcal{T}^{\text{act}}_{\text{cpu}}(\zeta\kappa\Pi)+\mathcal{C}_{1_{act}}(\zeta\kappa\Pi)(e^{r_1\Theta}-1)+\mathcal{C}_{2_{act}}(\zeta\kappa\Pi)(e^{r_2\Theta}-1) \qquad (4.27)$$

Substituting values for the constants from Equation (4.25), we get,

$$\Rightarrow \mathcal{T}_{\text{cpu}}(\zeta\kappa\Pi+\Theta) = \mathcal{T}^{\text{act}}_{\text{cpu}}(\zeta\kappa\Pi)\Big(\mathcal{P}_7(\Theta)\Big)+\mathcal{T}_{\text{env}}(\zeta\kappa\Pi)\Big(\mathcal{P}_8(\Theta)\Big)+\mathcal{P}_9(\Theta),$$

where,

$$\begin{aligned}\mathcal{P}_7(\Theta) &= 1-(e^{r_1\Theta}-1)\mathcal{G}_{C_1}-(e^{r_2\Theta}-1)\mathcal{G}_{C_2},\\ \mathcal{P}_8(\Theta) &= (e^{r_2\Theta}-1)\mathcal{G}_B+(e^{r_1\Theta}-1)\mathcal{G}_B, \text{ and}\\ \mathcal{P}_9(\Theta) &= (e^{r_1\Theta}-1)\mathcal{G}_{A_1}+(e^{r_2\Theta}-1)\mathcal{G}_{A_2}.\end{aligned}$$

[6]Derivation of the Equation A.7 is not shown here, given in the appendix to keep the focus of the discussion.

Also, considering the environment thermal behavior and substituting values for the constants from Equation (4.25), we get,

$$\Rightarrow \quad \mathcal{T}_{\text{env}}^{\text{act}}(\zeta\kappa\Pi + \Theta) = \mathcal{T}_{\text{cpu}}(\zeta\kappa\Pi)\Big(\mathcal{P}_{10}(\Theta)\Big) + \mathcal{T}_{\text{env}}(\zeta\kappa\Pi)\Big(\mathcal{P}_{11}(\Theta)\Big) + \Big(\mathcal{P}_{12}(\Theta)\Big), \tag{4.28}$$

where,

$$\begin{aligned}
\mathcal{P}_{10}(\Theta) &= -\frac{r_1+\beta_1}{k_T\sigma_1}\mathcal{G}_{C_1}(e^{r_1\Theta}-1) - \frac{r_2+\beta_1}{k_T\sigma_1}\mathcal{G}_{C_2}(e^{r_2\Theta}-1)\\
\mathcal{P}_{11}(\Theta) &= 1 + \frac{r_1+\beta_1}{k_T\sigma_1}\mathcal{G}_{B}(e^{r_1\Theta}-1) + \frac{r_2+\beta_1}{k_T\sigma_1}\mathcal{G}_{B}(e^{r_2\Theta}-1)\\
\mathcal{P}_{12}(\Theta) &= \frac{r_1+\beta_1}{k_T\sigma_1}(\mathcal{G}_{A_1}(e^{r_1\Theta}-1) + \frac{r_2+\beta_1}{k_T\sigma_1}\mathcal{G}_{A_2}(e^{r_2\Theta}-1).
\end{aligned}$$

Therefore, applying the Equations (4.24), (4.26), (4.28), and (4.28), in Equation (4.21), we may finally express our thermal-resiliency function in terms of the fixed thermal constants and input $\mathcal{T}_{\text{ref}}$ and $\Theta^{(i)}$ (which comes from the input mode M_i) as follows,

$$\Lambda(M_i, \mathcal{T}_{\text{ref}}) = \mathcal{T}_{\text{ref}} - \frac{\mathcal{E}_1(\Theta^{(i)})}{\mathcal{E}_N(\Theta^{(i)})} - \frac{\mathcal{E}_2(\Theta^{(i)})}{\mathcal{E}_N(\Theta^{(i)})}, \tag{4.29}$$

where,

$$
\begin{aligned}
\mathcal{E}_1(\Theta) &= \mathcal{J}_A(\Theta)\mathcal{P}_2(\Theta) + \mathcal{J}_4(\Theta)\mathcal{P}_{12}(\Theta)\mathcal{P}_2(\Theta) + \mathcal{J}_A(\Theta)\mathcal{P}_{11}(\Theta)\mathcal{P}_4(\Theta) - \mathcal{J}_2(\Theta)\mathcal{P}_{12}(\Theta)\mathcal{P}_4(\Theta) \\
&+ \mathcal{J}_A(\Theta)\mathcal{P}_3(\Theta)\mathcal{P}_8(\Theta) + \mathcal{J}_4(\Theta)\mathcal{P}_{12}(\Theta)\mathcal{P}_3(\Theta)\mathcal{P}_8(\Theta) - \mathcal{J}_3(\Theta)\mathcal{P}_{12}(\Theta)\mathcal{P}_4(\Theta)\mathcal{P}_8(\Theta) \\
&- \mathcal{J}_2(\Theta)\mathcal{P}_3(\Theta)\mathcal{P}_9(\Theta) - \mathcal{J}_4(\Theta)\mathcal{P}_{11}(\Theta)\mathcal{P}_3(\Theta)\mathcal{P}_9(\Theta) + \mathcal{J}_3(\Theta)\mathcal{P}_{11}(\Theta)\mathcal{P}_4(\Theta)\mathcal{P}_9(\Theta) \\
&- \mathcal{J}_4(\Theta)\mathcal{P}_{11}(\Theta)\mathcal{P}_A(\Theta) - \mathcal{J}_3(\Theta)\mathcal{P}_8(\Theta)\mathcal{P}_A(\Theta) + \mathcal{J}_3(\Theta)\mathcal{P}_2(\Theta)\mathcal{P}_9(\Theta) - \mathcal{J}_2(\Theta)\mathcal{P}_A(\Theta), \\
\mathcal{E}_2(\Theta) &= -\mathcal{J}_A(\Theta)\mathcal{P}_1(\Theta) - \mathcal{J}_4(\Theta)\mathcal{P}_1(\Theta)\mathcal{P}_{12}(\Theta) - \mathcal{J}_A(\Theta)\mathcal{P}_{10}(\Theta)\mathcal{P}_4(\Theta) + \mathcal{J}_1(\Theta)\mathcal{P}_{12}(\Theta)\mathcal{P}_4(\Theta) \\
&- \mathcal{J}_A(\Theta)\mathcal{P}_3(\Theta)\mathcal{P}_7(\Theta) - \mathcal{J}_4(\Theta)\mathcal{P}_{12}(\Theta)\mathcal{P}_3(\Theta)\mathcal{P}_7(\Theta) + \mathcal{J}_3(\Theta)\mathcal{P}_{12}(\Theta)\mathcal{P}_4(\Theta)\mathcal{P}_7(\Theta) \\
&+ \mathcal{J}_1(\Theta)\mathcal{P}_3(\Theta)\mathcal{P}_9(\Theta) + \mathcal{J}_4(\Theta)\mathcal{P}_{10}(\Theta)\mathcal{P}_3(\Theta)\mathcal{P}_9(\Theta) - \mathcal{J}_3(\Theta)\mathcal{P}_{10}(\Theta)\mathcal{P}_4(\Theta)\mathcal{P}_9(\Theta) \\
&+ \mathcal{J}_4(\Theta)\mathcal{P}_{10}(\Theta)\mathcal{P}_A(\Theta) + \mathcal{J}_3(\Theta)\mathcal{P}_7(\Theta)\mathcal{P}_A(\Theta) - \mathcal{J}_3(\Theta)\mathcal{P}_1(\Theta)\mathcal{P}_9(\Theta) + \mathcal{J}_1(\Theta)\mathcal{P}_A(\Theta), \\
\mathcal{E}_N(\Theta) &= \mathcal{J}_2(\Theta)\mathcal{P}_1(\Theta) + \mathcal{J}_4(\Theta)\mathcal{P}_1(\Theta)\mathcal{P}_{11}(\Theta) - \mathcal{J}_1(\Theta)\mathcal{P}_2(\Theta) - \mathcal{J}_4(\Theta)\mathcal{P}_{10}(\Theta)\mathcal{P}_2(\Theta) \\
&+ \mathcal{J}_2(\Theta)\mathcal{P}_{10}(\Theta)\mathcal{P}_4(\Theta) - \mathcal{J}_1(\Theta)\mathcal{P}_{11}(\Theta)\mathcal{P}_4(\Theta) - \mathcal{J}_3(\Theta)\mathcal{P}_2(\Theta)\mathcal{P}_7(\Theta) + \mathcal{J}_2(\Theta)\mathcal{P}_3(\Theta)\mathcal{P}_7(\Theta) \\
&+ \mathcal{J}_4(\Theta)\mathcal{P}_{11}(\Theta)\mathcal{P}_3(\Theta)\mathcal{P}_7(\Theta) - \mathcal{J}_3(\Theta)\mathcal{P}_{11}(\Theta)\mathcal{P}_4(\Theta)\mathcal{P}_7(\Theta) \\
&+ \mathcal{J}_3(\Theta)\mathcal{P}_1(\Theta)\mathcal{P}_8(\Theta) - \mathcal{J}_1(\Theta)\mathcal{P}_3(\Theta)\mathcal{P}_8(\Theta) - \mathcal{J}_4(\Theta)\mathcal{P}_{10}(\Theta)\mathcal{P}_3(\Theta)\mathcal{P}_8(\Theta) \\
&+ \mathcal{J}_3(\Theta)\mathcal{P}_{10}(\Theta)\mathcal{P}_4(\Theta)\mathcal{P}_8(\Theta).
\end{aligned}
$$

4.6 Validation

In this section, we evaluate our control framework both in simulations and upon an experimental hardware testbed.

4.6.1 Simulations

In the simulations, we simulate the execution of a single-core processor which consists of a thermal controller, PWM frequency controller loop, and scheduling algorithm. The following task parameters are used in our simulations:

- Each sporadic task $\tau_j = (e_j, d_j, p_j)$ has a period p_j uniformly drawn from the interval $[5, 15]$. (A small period range is used to keep LCM of periods from becoming too large). The execution time requirement e_j set to the task utilization times p_j, where task utilization is calculated using the

UUnifast algorithm[9]. For each task, d_j equals p_j. The tasks are scheduled by EDF.

- The total number of tasks is eight; each task τ_j has three different real-time performance modes where $\tau_j^{(2)} = (e_j, d_j, p_j)$; $\tau_j^{(1)} = (.2e_j, d_j, p_j)$; and $\tau_j^{(0)}$ means that task is not selected. From set of all possible combinations of tasks, we have selected fifteen modes with utilizations ranging from zero to one.

We refer to the controller described in Algorithm 1 as *Temperature Regulated Capacity Bound* (TRCB). In our simulations, we closely compare the performance of our proposed method with [37] referred to as *Thermal Control Utilization Bound* (TCUB). TCUB has been chosen due to its low controller time complexity of $O(1)$. TCUB works by attempting to track a reference temperature and adjusting system utilization as needed by changing task modes via a mode assignment heuristic. The major difference between TCUB and TRCB is that TCUB does not have predefined modes. Therefore, TCUB may differ in the assigned modes from run to run for the same system temperature. Furthermore, TCUB does not use multiple power levels. TRCB on the other hand has predefined modes which permit the derivation of thermal resiliency for each mode. TCRB also utilizes a low-power mode (if available).

In our simulation, we use the same system parameters as our testbed (Intel Pentium IV 3.0 GHz). The pertinent power and control parameters are given in Table 4.1. Extensive testbed runs were carried out to generate the remaining system parameters using SI. We use the SI tools provided by Matlab to derive the system state-space parameters. Also we use the system parameters, generated from our testbed as the simulation parameters. We observe a matching of our testbed readings and the simulation. More details on this process are contained in the technical report [45].

In Figure 4.6, the system response and the utilization has been shown for both TRCB (right graphs) and TCUB (left graphs) given a stable air temperature $\mathcal{T}_{\text{air}}$ temperature equal to $5°C$. The behavior of both controllers in this stable environment is nearly identical for thermal and utilization behavior. (The difference is due to the fact that TRCB uses EDF and TCUB uses RM scheduling). For TRCB, we also display the achieved modes at any given time in the simulation in the lower right graph.

Figure 4.7 shows the behavior of both TRCB and TCUB when $\mathcal{T}_{\text{air}}$ is dynamically changed over time. In the top two graphs of the figure, the absolute CPU temperatures over time obtained by TCUB and TRCB, respectively, are plotted along with the $\mathcal{T}_{\text{air}}$. The two bottom graphs of Figure 4.7 present the achieved utilization for each controller; additionally, the bottom right graph displays the active mode at

Table 4.1: Testbed parameters for uni-processor simulations

Parameter	Variable	Value
CPU Active Power	$\mathcal{P}_{\text{act}}$	73 W
CPU Idle Power	$\mathcal{P}_{\text{inc}}$	20 W
Server Period	Π	20 ms
Sampling Time	T_s	100 ms
Optimal Feedback	K_o	$\begin{bmatrix} .5725 & 0 \end{bmatrix}$
Q matrix in Performance Index	Q	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
R matrix in Performance Index	R	$\begin{bmatrix} 1 \end{bmatrix}$
Integral Gain	γ_I	0.00042

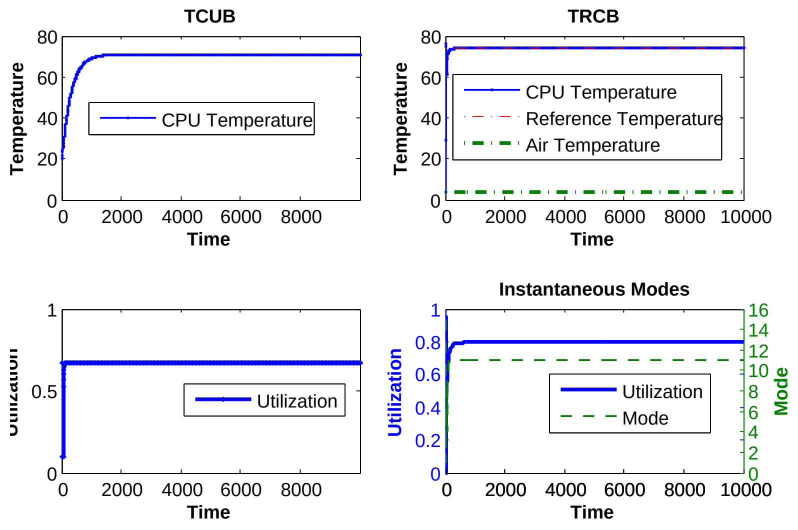


Figure 4.6: Fixed $\mathcal{T}_{\text{air}}$ for Simulation. Left plots represent TCUB and right plots represent TCRB.

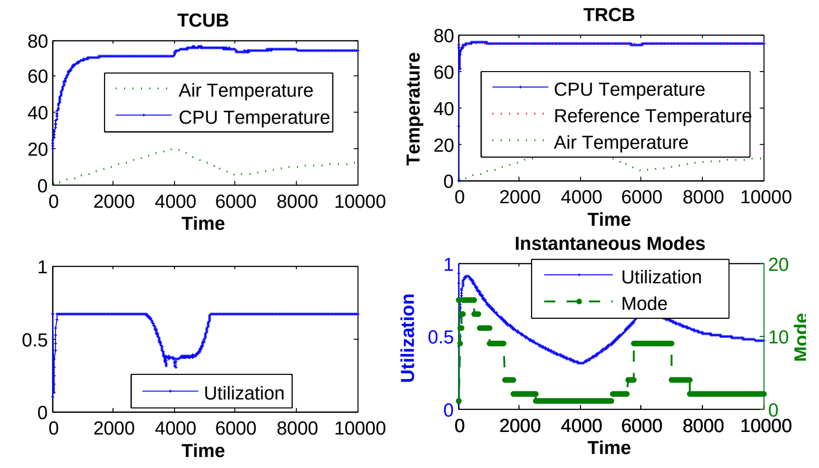


Figure 4.7: Simulation comparison under varying T_{air}.

Dynamically varying T_{air} for simulation. Left plots represent TCUB and right plots represent TCRB.

any point in time for TRCB. Observe that both controllers are able to track the reference temperature $\mathcal{T}_{\text{ref}}$ despite the sharp changes in $\mathcal{T}_{\text{air}}$. For both controllers, the utilization appropriately tracks the changes in air temperature. When the air temperature increases, both controllers decrease the system utilization and increase the utilization again when the air temperature drops. Similarly, the mode plot in the lower right graph tracks the temperature changes.

Regarding the real-time performance, figures displaying deadline miss ratios have been omitted as no deadline miss was experienced for either controller in all the simulations. TCUB uses a safe utilization bound of approximately 67% to make deadline misses improbably for rate-monotonic scheduling [57]. However, TCRB guarantees that no deadlines are ever missed due to verification using a multi-modal schedulability test [30] as described in Section 3.3.2.

Thus far, the empirical performance of TRCB and TCUB may appear similar. However, we believe the distinguishing feature of TRCB is the ability to guarantee hard deadlines and to calculate thermal resiliency levels during design time. Thermal resiliency calculation provides a *non-destructive* thermal stress analysis for real-time performance modes in an unpredictable operating environment. Our approach has achieved the ability to calculate the thermal resiliency by forcing the system to execute in a very

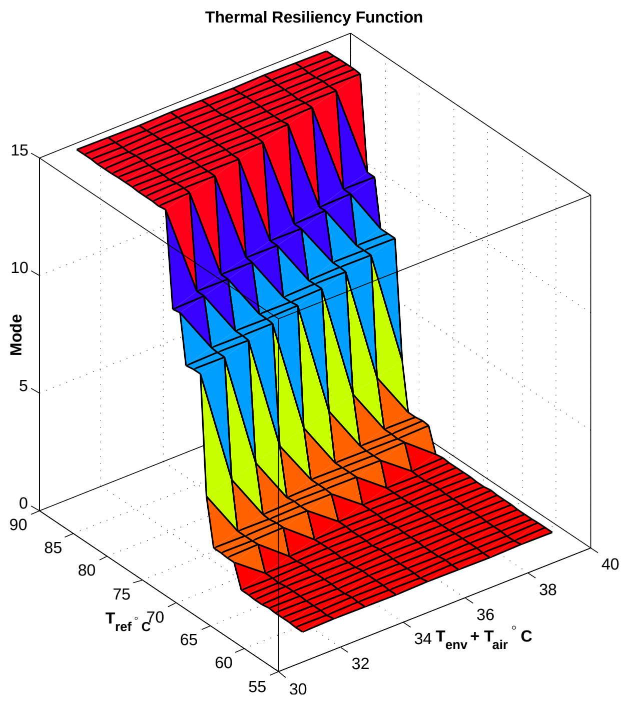


Figure 4.8: Thermal resiliency over modes and $\mathcal{T}_{\text{ref}}$.

predictable manner (i.e., periodic executions from PWM). To evaluate and illustrate our thermal resiliency calculation, we have used the technique in Section 4.5 to calculate the thermal resiliency levels for our randomly-generated multi-mode system. Figure 4.8 displays the thermal resilience $\Lambda(M_i, \mathcal{T}_{\text{ref}})$ for a range of modes and reference temperatures. Observe that the thermal resiliency increases with decreasing modes or increasing $\mathcal{T}_{\text{ref}}$.

4.6.2 Experiments upon Hardware Testbed

To further confirm the validity of the theoretical results, we have run a task system with eight tasks, each with three modes (identical to the simulation setting), on our hardware testbed. Each task performs numerical calculations while executing on the system. Our hardware testbed behaves similar to the simulations of the previous subsection. Figure 4.9 presents testbed runs for a fixed air and environment temperature.

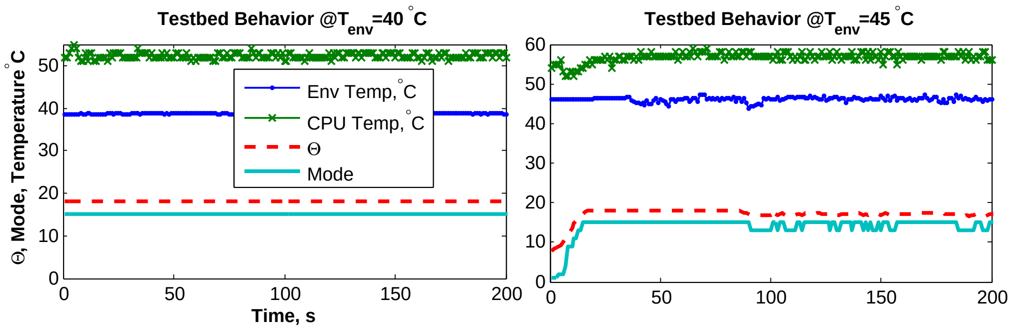


Figure 4.9: The testbed behavior under different $\mathcal{T}_{\text{env}}$ values.

The testbed running at different $\mathcal{T}_{\text{env}}$ values and $\mathcal{T}_{\text{ref}} = 55°C$ showing the Θ and Mode change over the time.

Finally, we validate our thermal resiliency calculation. Figure 4.10 shows the comparison of the thermal resiliency. The upper left and right graph show the simulated results and the actual testbed observations of the resiliency respectively. The lower figure shows comparison of the resiliency of the system. Figure shows the calculated thermal resiliency tracks the actual behavior of the testbed and provides a safe upper bound on $\mathcal{T}_{\text{ref}}$ in a large majority of the cases which validates the effectiveness of the resiliency function. Our equipment supports $1°C$ accuracy in temperature measurements. Furthermore, for a better accurate experiments, the environmental temperature chamber (Thermal Cabinet) needs an extensive insulation methods. However, even under given conditions, using our framework, the designer can accurately calculate thermal constraint for a given mode.

4.7 Summary

In this chapter, we discussed the problem of obtaining performance guarantees in an unpredictable thermal environment. Towards this challenge we presented a control-theoretic framework for thermal stress

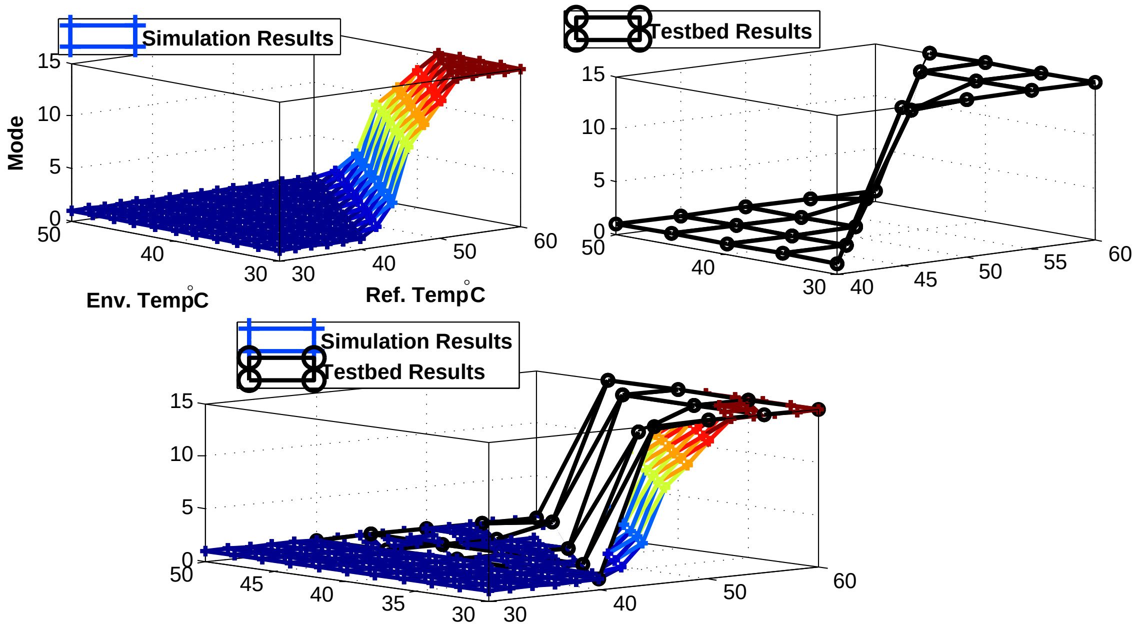


Figure 4.10: Simulated Thermal Resiliency Comparison with Testbed Data.

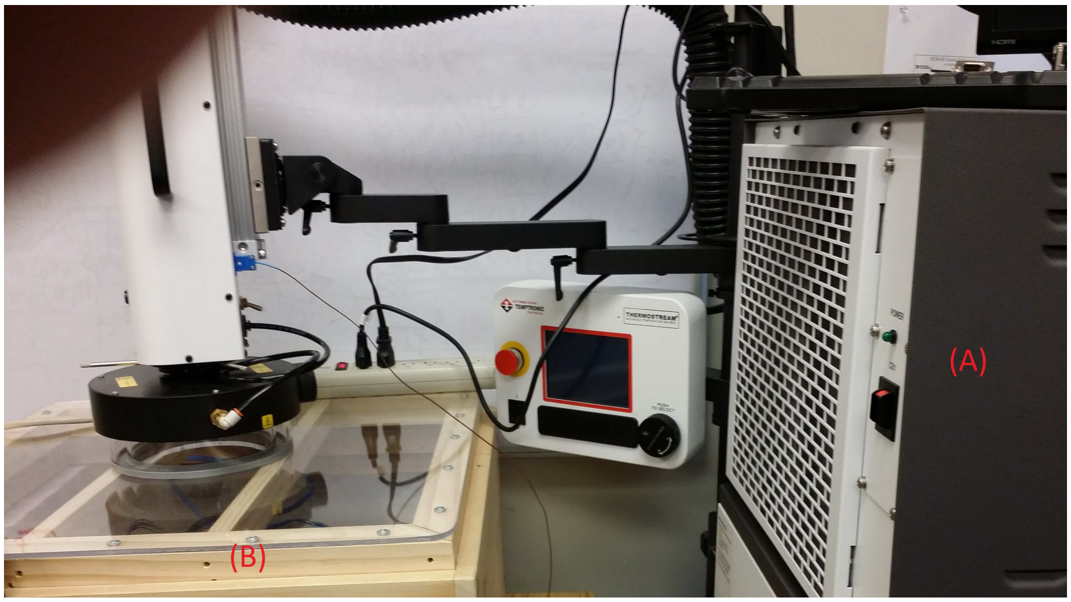


Figure 4.11: The Thermal Forcing System (A) and the Thermal Cabinet (B).

analysis in real-time systems. Our proposed method employs a nested feedback control system, which is based on optimum control theory. Our thermal controller nests with a faster PWM based CPU-modulation driver; the thermal control system dynamically controls both processor temperature and CPU utilization through an underlying periodic resource. We have shown this design has bounded deviation from ideal system with continuous power modes. Furthermore, for our system, we derive strong thermal-resiliency and hard-real-time guarantees for any real-time performance mode. Our method has the distinct advantage of being able to verify the real-time thermal resiliency of a system before it is put into operation. In addition, we show via simulations that our framework performs as well as previous approaches which have no formal guarantee on the thermal resiliency. Our implementation upon a hardware testbed validates our proposed model and control framework.

In future work, we plan to extend our framework to control designs that are more robust to model inaccuracies (e.g., $\mathcal{H}_\infty$ or model-predictive controllers). As a initial step in designing a framework for thermal stress analysis, our current design uses two RC circuits (for dynamic and leakage currents) to model the CPU temperature. We plan on extending our model to permit multiple RC circuits for heterogeneous thermal distributions and generalizing our thermal equations for more complex RC circuit layouts. We hope to derive a general-theoretic design framework that captures "resiliency" metrics for other system properties (e.g., energy, noise, etc.) and extend our analysis to other hardware settings (e.g., multicore, DVS).

CHAPTER 5: THERMAL-RESILIENCY ON MULTICORE PROCESSOR SYSTEMS

In the previous chapter, we introduced the thermal resiliency for uniprocessor systems. In this chapter, we extend the thermal resiliency for multicore processor systems, which characterizes the maximum external thermal stress that any hard-real-time performance mode can withstand.

Multicore hard real-time systems have been widely using in an increasing number of real-time and embedded systems. These systems need to operate under various physical and design constraints, including ensuring that the system is operating within safe thermal constraints. In this chapter, we discuss a control-theoretic framework to ensure hard-real-time deadlines on a multiprocessor platform in a dynamic thermal environment. We use real-time performance modes to permit the system to adapt to changing conditions. Also, we show how the system designer can use our framework to allocate asymmetric processing resource upon a multicore CPU and still maintain thermal constrains. We show the simulations and physical testbed results at the end of the chapter to confirm that our algorithm predicts how a system will gracefully and predictably degrade under external thermal stress.

This chapter presents a methodology for designing and analyzing thermal-resilient multicore hard-real-time systems. Section 5.1 presents brief introduction and overview of this research. Section 5.2 overview the hardware, real-time, and thermal models used throughout the chapter. Section 5.3 details the design of our thermal-resilient controller. Section 5.4 derives thermal-resiliency function Λ for control system. Section 5.5 describes the results of our simulations and implementation upon testbed hardware. In the next chapter, we provides formal hard-real-time system guarantees through formal derivations and proofs by extending the model that we developed in the previous chapter earlier. Finally, Section 5.6 gives a chapter summary.

5.1 Introduction

Designing a real-time and thermal-aware multicore system is not straightforward; the CPU cores are closely packed inside the CPU die, and each core potentially has different thermal characteristics. Therefore, each core needs separate thermal model and interaction between cores also needs to be considered. Further, if the system is operating in an environment with changing temperatures, the thermal-control needs to be designed very carefully to take into account the system's timing constraints. Modern embedded systems are extremely complex and therefore, identification of some of the system limitations through testing are limited. Furthermore, due to pressure to minimize the size and cost of the system, a system designer is often compelled to consider the trade-offs between performance and the physical constraints of the system (e.g., temperature). For a smooth, robust design phase and for a reliable final product, the system designer must be equipped with appropriate design-frameworks to carefully consider the implication of the tradeoff. The proposed methods in this paper help the system designer to analyze the trade-offs between the real-time performance and environmental constraints of a hard real-time system.

Although many modern CPUs are manufactured with built-in dynamic thermal management (DTM) capabilities, they cannot be freely utilized in hard real-time system designs; a complicated hard real-time system might compromise its timing constraints if DTM features are utilized improperly. Therefore, it is essential to develop a framework that considers all external system capabilities such as DTM and real-time requirements. Further, a useful framework should also permit the system designer to predict the behavior of the system even if the operating environment is not a priori known or predictable.

There are already available frameworks for permitting a trade-off between a real-time level of service and resource requirements. For example, Ghosh et al. [42] proposed a framework for mapping the level of service and resource requirements for dynamic environmental conditions. These techniques might guarantee the real-time deadlines for a previously-determined level of resources, but they do not address the scenario where requirements can dynamically change (resulting in a real-time mode change). They cannot guarantee hard real-time deadlines when a system may dynamically switch between real-time modes. Furthermore, none of these previously-proposed techniques can be used to obtain a precise, formal quantification of the thermal stress that the muticore system can withstand.

In this chapter, we extend the concept of *real-time thermal resiliency* to multicore platforms. Real-time thermal resiliency is a system design metric that quantifies the maximum external operating temperature that the system can withstand for a specified *real-time performance mode*. To define thermal resiliency for a multicore processor, let us assume the $\mathcal{C}$'th core of a multicore system with q different hard-real-time performance modes $M^{\mathcal{C},0}, M^{\mathcal{C},1}, \ldots, M^{\mathcal{C},q}$ where modes are ordered in increasing levels of real-time performance with $M^{\mathcal{C},q}$ guaranteeing the highest level and $M^{\mathcal{C},0}$ the lowest. For this system, the real-time thermal resiliency for the i'th mode of the $\mathcal{C}$'th processor $M^{\mathcal{C},i}$, denoted as $\Lambda(M^{\mathcal{C},i}, \mathcal{T}_{\text{ref}})$, represents the predicted maximum external operating temperature for which the system will continue to operate at performance mode $M^{\mathcal{C},i}$ or higher and maintain a CPU reference temperature of $\mathcal{T}_{\text{ref}}$.

In recent years, we find many real-time researches on multicore systems. They address important issues related to thermal-aware and power-aware design. Further, each of these prior results do not provide a mechanism to specify the graceful degradation of the system's operating modes in an unfavorable environment. In contrast to these work, that rely on mapping techniques to adopt the varying environmental conditions, we provide the system designer real-time and performance guarantee. While no previously-proposed techniques exist for obtaining a formal quantification on a muticore system, in this chapter, we proposed the idea of thermal resiliency multicore processor systems.

Towards this goal, we design a framework to calculate and verify the real-time guarantees of a multicore system in the presence of unpredictable dynamic environmental conditions. Thus, we are proposing a thermal-stress analysis mechanism for multicore real-time systems. Using our proposed method, the system designer can specify, a priori, a precise quantification of the hard-real-time performance degradation of multicore system due to external thermal events. Furthermore, our thermal analysis framework explains how the modes degrade gracefully and predictably under externalthermal stress. Therefore, using our framework, the system designer can predict the thermal resiliency of a performance mode. Finally, our framework addresses the issue of allocating asymmetric mode request from different cores in a multicore CPU.

This chapter discusses the the following important contributions:

- We propose a controller for a multicore system with rudimentary power-control mechanisms (e.g., cores can be active or inactive). Our objective is to ensure that the maximum core temperature tracks a given reference temperature $\mathcal{T}_{\text{ref}}$. We show that our controller is able to maintain system stability and controllability. Furthermore, using the system specification, we can obtain a closed-form quantification of the thermal-resiliency for each operating mode for each core of the system.

- The proposed mechanism is also helpful to analyze multicore system under certain capacity constraints. Often times system designer assigns a mission-critical hard-real-time task to a specific core and resources for other real-time tasks allocated from rest of the cores in the system. For example, assume the designer wants a fixed resource capacity from a specific core that needs to run the crucial real-time task in a system and other cores contribute to the system with multi-mode capabilities; under this situation, the designer needs to know the the modes that the system could expect from the other cores under various external thermal conditions. We provide support and analysis for this setting.

- We empirically evaluate the efficacy of our control algorithm and associated analysis upon a multicore CPU. We show (through simulations and testbed runs) that our model and characterization of thermal resiliency closely predicts the system's thermal behavior even in a dynamic operating environment.

5.2 Models

We develop our power model to represent a wide range of embedded processors with minimal amount of power management capabilities. As explained in the Section 3.3, we assume a multicore processor system with *active* and *inactive* power modes. We denote the instantaneous CPU power of $\mathcal{C}$'th core as $\mathcal{P}_{\text{cpu}}^{\mathcal{C}}(t), (\mathcal{C} \in \{1 \ldots m\})$ and assume it dissipates thermal power at a constant rate $\mathcal{P}_{\text{act}}$ and $\mathcal{P}_{\text{inc}}$ in the active and inactive modes, respectively. Also, we assume that processor consumes e_{act} and e_{inc} amount of energy to activate/deactivate from inactive/active modes. The complete multicore model is given in the Section 3.3.

As we explained in the introduction, we consider each core of the multicore processor has a specific number of possible performance modes. Also, the task migration at runtime is not permissable and the tasks are statically partitioned within the available processors. Further, the complete description is given in the Section 3.3.

We furthermore assume that within each processor allocation, an arbitrary uniprocessor scheduling algorithm (e.g., EDF or RM) may be employed to schedule the underlying task system. See Figure 5.1 for an illustration of the thermal-aware periodic resource.

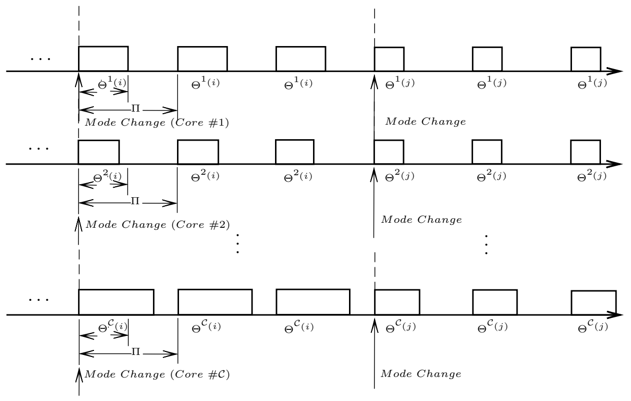


Figure 5.1: The sampling and mode change in our thermal control system.

The blocks indicate time periods during with the processor is active under the thermal-aware periodic resource model. Sporadic tasks are scheduled within the activation blocks .

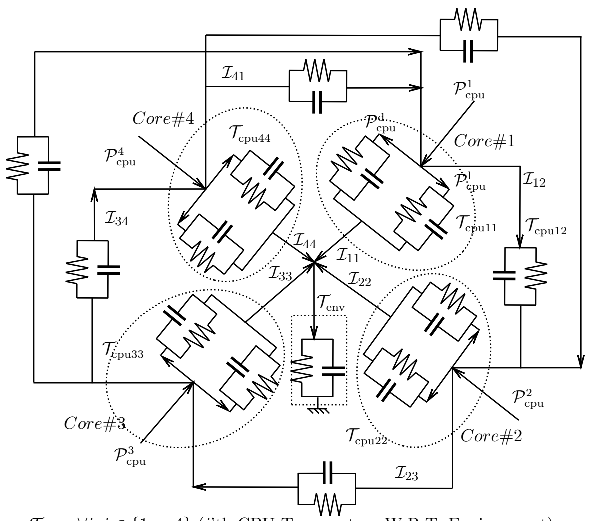
5.2.1 Power/Thermal Derivations

Our thermal architecture is based on the model we explained in the Section 3.3. We explained how the thermal architectural model of a multicore processor is organized and showed the arrangement of different RC components to represent individual cores of the multicore processor as well as inter-core thermal effects.

Figure 5.2 shows the basic equivalent circuit of a multicore CPU and its surrounding environment. For clarification, we have shown a multicore processor with four cores.
We use the following state-space thermal model for the rest of the design process,

$$\dot{\mathcal{T}}_{\text{cpu}}(t) = \mathcal{A}\mathcal{T}_{\text{cpu}}(t) + \mathcal{B}\mathcal{P}_{\text{cpu}}(t), \tag{5.1}$$

where, $\dot{\mathcal{T}}_{\text{cpu}}(t) = \left(\dot{\mathcal{T}}^{\mathcal{C}}_{\text{cpu}}(t)\right)_{m\times 1}$, $\mathcal{T}_{\text{cpu}}(t) = \left(\mathcal{T}^{\mathcal{C}}_{\text{cpu}}(t)\right)_{m\times 1}$, $\mathcal{P}_{\text{cpu}}(t) = \left(\mathcal{P}^{\mathcal{C}}_{\text{cpu}}(t)\right)_{m\times 1}$, and $\mathcal{A} = (\mathcal{A})_{m\times m}, \mathcal{B} = (\mathcal{B})_{m\times m}$ are state-space parameters. The detailed derivations definitions of these matrices are given in the Appendix of the thesis.


$\mathcal{I}_{41}$
$\mathcal{P}^{1}_{\text{cpu}}$
$Core\#4$
$\mathcal{T}_{\text{cpu44}}$
$\mathcal{P}^{\text{d}}_{\text{cpu}}$
$Core\#1$
$\mathcal{P}^{4}_{\text{cpu}}$
$\mathcal{P}^{1}_{\text{cpu}}$
$\mathcal{I}_{12}$
$\mathcal{I}_{34}$
$\mathcal{T}_{\text{cpu11}}$
$\mathcal{T}_{\text{cpu12}}$
$\mathcal{I}_{44}$
$\mathcal{I}_{11}$
$\mathcal{I}_{33}$
$\mathcal{I}_{22}$
$\mathcal{T}_{\text{env}}$
$\mathcal{T}_{\text{cpu33}}$
$Core\#3$
$\mathcal{P}^{3}_{\text{cpu}}$
$\mathcal{T}_{\text{cpu22}}$
$Core\#2$
$\mathcal{P}^{2}_{\text{cpu}}$
$\mathcal{I}_{23}$

$\mathcal{T}_{\text{cpuii}}, \forall i, j \in \{1 \ldots 4\}$ (i'th CPU Temperature W.R.T. Environment)

$$\mathcal{T}_{\text{cpuij}} = \mathcal{T}_{\text{cpujj}} - \mathcal{T}_{\text{cpuii}} \ \forall i, j \in \{1 \ldots 4\}$$

Figure 5.2: The multicore equivalent electrical circuit.

The basic equivalent electrical circuit of the thermal model of the CPU and its working environment is shown. (for simplicity, the figure shows the structure with 4 adjacent cores) Arrow direction shows the current (A) direction of the equivalent electrical circuit.

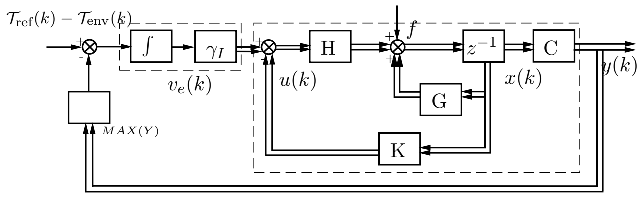


Figure 5.3: The thermal control design with state feedback and integral actuator

5.3 Controller Design

First, we design a thermal controller assuming that an ideal system with continuous power modes. In Section 5.3.4, we will extend the controller design to a processor with only active/inactive power modes.

We need our controller to accurately follow the reference temperature, $\mathcal{T}_{\text{ref}}$. Also, to eliminate steady state tracking error, we design our system as a servo with an integrator. Define an additional error vector $v_e(t)$ in continuous time as,

$$\begin{aligned} v_e(t) &\overset{\text{def}}{=} \int_0^t (\mathcal{T}_{\text{ref}} - \mathcal{T}_{\text{cpu}}(t) - \mathcal{T}_{\text{env}}(t))dt \\ \dot{v}_e(t) &\overset{\text{def}}{=} \mathcal{T}_{\text{ref}} - \mathcal{T}_{\text{cpu}}(t) - \mathcal{T}_{\text{env}}(t) \\ &= \mathcal{T}_{\text{ref}} - \max\left(\left[\mathcal{T}_{\text{cpu}}(t)\right]\right) - \mathcal{T}_{\text{env}}(t). \end{aligned} \tag{5.2}$$

Then, the system input is calculated with a gain K_o and integral constant γ_I in the following equation.

$$\mathcal{P}_{\text{cpu}}(t) = -K_o\left[\mathcal{T}_{\text{cpu}}(t)\right] + \gamma_I v_e(t), \tag{5.3}$$

where, $K_0 = (\mathcal{K}_{ij})_{m\times m}$. We employ standard techniques from optimal control theory to derive K_o and prove stability. Details are presented in an appendix of an extended version of this paper [48].

5.3.1 State-Space Controller Details

We use the standard state-space model to represent continuous-time (ideal) system, and the state matrices and constant vector are time-invariant quantities as explained in the Section 3.6.

Since we have a computer-controlled discrete-time system, we use the state-space mode for the discrete-time controller for active/inactive modes as given in the Section 3.6.

5.3.2 Continuous Power Modes

As a first step towards our goal of designing a control-theoretic framework for thermal stress analysis, we employ *linear quadratic (LQ) optimal control* for real-time thermal management. Our design consists of an optimal state feedback and a servo that regulates the dynamics of the system. An LQ controller enables us to design an efficient and low-overhead controller, derive the feedback parameters before runtime (used in thermal-resiliency analysis), and smoothly track our reference input. In the future, we plan on applying more complex and robust controllers (e.g., $\mathcal{H}_\infty$ controllers) to decrease the controller's sensitivity to modeling inaccuracy and noise. However, as observed in the simulations and experiments of Section 5.5, our current LQ design is appropriately responsive to changes in environmental temperature.

In our system model, we specify the thermal power of the CPU as the control to the system. The controller is required to work as a servo and should follow the temperature reference, $\mathcal{T}_{\text{ref}}$. In our design, we consider $\mathcal{T}_{\text{cpu}}(t)$ as one of the variable to be controlled and $\mathcal{P}^{\text{d}}_{\text{cpu}}(t)$ as a manipulated variable. The basic control structure is given in Figure 5.3.

5.3.3 Stability Analysis and Optimal State Feedback

In our derivation of stability for our system, we will use the Lemma 5 and Lemma 4 in the Section 3.6, which two results also can be found in any standard text on control theory [21, 65, 66].

Now we derive the augmented system model that is used to obtain the optimality of the system. Equation (B.11) can be used to describe the system dynamics at any time instance. Consider an instance where system is completely stable and has attained to the steady state. We denote the system input, system states, and the servo error (described in Equation (5.2)) of this special instance of the system by $\mathcal{P}_{\text{cpu}}(t_\infty))$, $\mathcal{T}_{\text{cpu}}(t_\infty)$, $\mathcal{T}_{\text{env}}(t_\infty)$ and $v_e(t)$ respectively. Therefore, we get,

$$\begin{bmatrix}\dot{\mathcal{T}}_{\text{cpu}}(t_\infty)\end{bmatrix} = \begin{bmatrix}\mathcal{A}\end{bmatrix}\begin{bmatrix}\mathcal{T}_{\text{cpu}}(t_\infty)\end{bmatrix} + \begin{bmatrix}\mathcal{B}\end{bmatrix}\mathcal{P}_{\text{cpu}}(t_\infty). \tag{5.4}$$

Then, from the Equations (B.11) and (5.4) we get,

$$\begin{aligned}\left[\dot{\mathcal{T}}_{\text{cpu}}(t) - \dot{\mathcal{T}}_{\text{cpu}}(t_\infty)\right] &= \left[\mathcal{A}\right]\left[\mathcal{T}_{\text{cpu}}(t) - \mathcal{T}_{\text{cpu}}(t_\infty)\right] \\ &+ \left[\mathcal{B}\right](\mathcal{P}_{\text{cpu}}(t) - \mathcal{P}_{\text{cpu}}(t_\infty)).\end{aligned} \tag{5.5}$$

Define $e(t)$,

$$e(t) = \left[\mathcal{T}_{\text{cpu}}(t) - \mathcal{T}_{\text{cpu}}(t_\infty)\right], \tag{5.6}$$

then we get,

$$\dot{e}(t) = Ae(t) + Bu_e(t). \tag{5.7}$$

We select the feedback gain $\hat{\gamma}$ such that,

$$\begin{aligned} u_e(t) &= \mathcal{P}_{\text{cpu}}(t) - \mathcal{P}_{\text{cpu}}(t_\infty) && (5.8) \\ &= -K_0 e(t). && (5.9)\end{aligned}$$

The above state-space and the control gain parameters are valid for a continuous-time controller. So, we may obtain the discrete-time state-space matrices for the augmented model (i.e, G and H) from $\hat{A}$ and $\hat{B}$ via the transformation described after Equation (3.5). In LQ optimal control, the objective is to design the controller to minimize some performance index. A standard LQ performance index is given by

$$J \overset{\text{def}}{=} \frac{1}{2}\sum_{k=0}^{\infty}\big(e(k)^T Q e(k) + u_e^T(k) R u_e(k)\big), \tag{5.10}$$

where Q and R are arbitrary symmetric matrices of size $m \times m$ and $r \times r$ such that $Q \geq 0$ (positive semi definite), $R > 0$ (positive definite). The $u_e(k)$ is the difference of the feedback value and it becomes zero at the steady state (when $\mathcal{T}_{\text{cpu}}(t)$ becomes $\mathcal{T}_{\text{cpu}}(t_\infty)$ at the steady state).

It is easy to show that for a Linear Time Invariant (LTI) system, (Refer to Ogata [66]), the optimal state feedback can be obtained as,

$$u_e(k) = -\hat{K}e(k), \tag{5.11}$$

where $\hat{K}$ is the feedback gain defined as

$$\hat{K} = (R + H^T P H)^{-1} H^T P G, \tag{5.12}$$

and where P is the positive definite solution of the algebraic Riccati equation below,

$$P = Q + G^T P G - G^T P H (R + H^T P H)^{-1} H^T P G.$$

From the above, it may be shown [66] that the optimal performance index can be calculated as

$$J_{min} = \frac{1}{2} e^T(0) P e(0). \tag{5.13}$$

It is well known [66] that the feedback control (i.e., K) results in an asymptotically stable closed-loop system according to Lemma 2. Obviously, stable choices of K_o for the system can be derived.

5.3.4 Continuous Power Emulation with Active/Inactive Power Modes

In this section, we explain how the resource capacity, $\Theta^{\mathcal{C}}$ is manipulated to produce the control input value $\mathcal{P}^{\mathcal{C}}_{\text{cpu}}$. Any modern CPU has a discrete set of operating frequencies. The power dissipation (consumption) for each operating frequency is not a variable and it is fixed.[1] However, the control design we proposed requires a continuous input, $\mathcal{P}_{\text{cpu}}(t)$ for its proper functionality. Since the CPU power cannot be varied continuously, the controller designed in the previous section cannot be directly applied to the setting of discrete active/inactive power modes. Therefore, the design of the continuous power modes controller described in the previous section needs to be modified to accomodate the active/inactive power mode setting by applying pulse-width modulation (PWM) techniques. As we explained earlier in the Section 5.2 (and the details in the Section 3.3 of Chapter 3), the active/inactive power modes will be modeled via the thermal-aware periodic resource model with parameters Π and Θ. This could be achievable via choosing the appropriate values of Π and Θ. The Π value is a design parameter which may be chosen at controller design-time and will be assumed fixed throughout controller execution. The only constraint that our framework places on the chosen value of Π is that it must evenly divide the sampling interval length T_s (i.e.,

[1]Under experimental conditions and due to the variation of the workload, we may find slight variation on the power consumption value for a specific operating frequency for different workload executions; however, the average value of the power consumption is very closer.

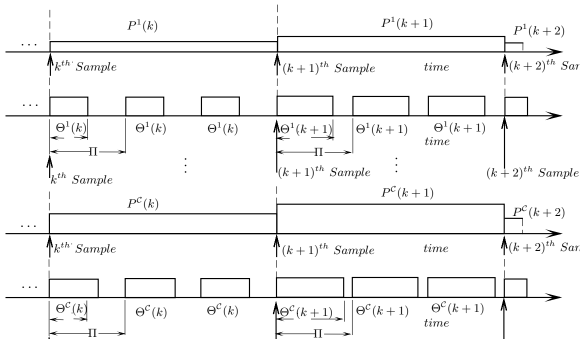


Figure 5.4: The simplified power and modulation relationship

$T_s = \kappa\Pi$ for some $\kappa \in \mathbb{N}^+$).

Let $\Theta^{\mathcal{C}}(k)$ denote the value of the resource capacity over the k'th sampling period on $\mathcal{C}$'th core. For determining the $\Theta^{\mathcal{C}}(k)$ value, we use a method based on the *principle of equivalent areas* (PEA) for converting any arbitrary input signal ($\mathcal{P}^{\mathcal{C}}_{\text{cpu}}(k)$) into an equivalent PWM signal [39] and assume the *zero-order hold* (ZOH), the input signal is held constant over the sampling period. Specifically, for the k'th sampling interval, the input $\mathcal{P}^{\mathcal{C}}_{\text{cpu}}(k)$ is held over the T_s-length interval, resulting in a total energy dissipation of $T_s \cdot \mathcal{P}^{\mathcal{C}}_{\text{cpu}}(k)$ over the interval. To get the equivalent area (i.e., energy) as the (ideal) system with continuous power modes, we must set $\Theta^{\mathcal{C}}(k)$ such that the periodic modulations between the power modes of $\mathcal{P}_{\text{act}}$ and $\mathcal{P}_{\text{inc}}$ dissipate the equivalent amount of energy over the T_s-length interval. Figure 5.4 illustrates the area equivalence between the continuous and PWM controllers.

More formally, we may derive the following relationship between $\mathcal{P}^{\mathcal{C}}_{\text{cpu}}(k)$ and $\Theta^{\mathcal{C}}(k)$,

$$\begin{aligned}
\kappa\Pi\mathcal{P}^{\mathcal{C}}_{\text{cpu}}(k) &= \kappa\Big(e_{\text{act}} + \int_0^{\Theta^{\mathcal{C},i}(k)} \mathcal{P}_{\text{act}}dt \\
&+ e_{\text{inc}} + \int_{\Theta^{\mathcal{C},i}(k)}^{\Pi} \mathcal{P}_{\text{inc}}dt\Big) \\
\Rightarrow \mathcal{P}^{\mathcal{C}}_{\text{cpu}}(k) &= \left(\frac{\mathcal{P}_{\text{act}} - \mathcal{P}_{\text{inc}}}{\Pi}\right)\Theta^{\mathcal{C},i}(k) \\
&+ \mathcal{P}_{\text{inc}} + \frac{1}{\Pi}(e_{\text{act}} + e_{\text{inc}}),
\end{aligned} \tag{5.14}$$

where $\Theta^{\mathcal{C},i}(k)$ the resource capacity of the $\mathcal{C}$'th CPU for i'th mode and $\mathcal{P}^{\mathcal{C}}_{\text{cpu}}(k)$ denotes the relevant power of $\mathcal{C}$'th CPU, $\mathcal{C} \in \{1 \ldots m\}$. Note that, the above Equation 5.14 gives us a way to calculate the total power consumption of any real-time task by using the time it takes to complete the task and the Θ value.

Algorithm 2 Control Algorithm

```
Require: Reference Temperature $\mathcal{T}_{\text{ref}}$; Feedback Gain $K$; Integral Constant $\gamma_I$; PWM Period $\Pi$; Number
    of PWM periods in a sampling period $\kappa$.
 1: while At beginning of sampling period $[t_\ell, t_{\ell+1}) : t_\ell \equiv \kappa\ell\Pi$ do
 2:     Sample $\mathcal{T}_{\text{cpu}}(t_\ell) + \mathcal{T}_{\text{env}}(t_\ell)$.
 3:     $\dot{v}_e(t_\ell) = \mathcal{T}_{\text{ref}} - \mathcal{T}_{\text{env}}(t_\ell) - \max(\mathcal{T}_{\text{cpu}}(t_\ell))$
 4:     $Tot\_\dot{v}_e(t_\ell) = Tot\_\dot{v}_e(t_{\ell-1}) + \gamma_I \kappa \Pi \frac{\left(\dot{v}_e(t_\ell) + \dot{v}_e(t_{\ell-1})\right)}{2}$
 5:     for $\mathcal{C} = 1$ to $m$ do
 6:         $\mathcal{P}^{\mathcal{C}}_{\text{cpu}}(t_\ell) = \left(Tot\_\dot{v}_e(t_\ell)\gamma_A - \left([K_{\mathcal{C},r}]\mathcal{T}_{\text{cpu}}(t_\ell)\right)\right)$ ; $r \in \{1 \ldots m\}$
 7:         $\Theta^{\mathcal{C},i}(t_\ell) = \min\left(\Pi \times \frac{(\mathcal{P}^{\mathcal{C}}_{\text{cpu}}(t_\ell) - \mathcal{P}_{\text{inc}})}{\mathcal{P}_{\text{act}} - \mathcal{P}_{\text{inc}}}, \Pi\right)$
 8:         $i = \max\{j \in \mathbb{Z}_{q+1} \mid \Theta^{\mathcal{C},j} \leq \Theta^{\mathcal{C},i}(t_\ell)\}$
 9:         Update real-time performance mode to $M^{\mathcal{C},i}$.
10:         Set PWM to operate at period of $\Pi$ and width of $\Theta^{\mathcal{C},i}(t_\ell)$.
11:     end for
12: end while
```

The PWM controller pseudocode is presented in Algorithm 2. The controller proposed here consists of two integrated operations: the thermal controller and the PWM modulator. The first step is to obtain the sample CPU temperature (Line 3 of Algorithm 2). Then, we select maximum temperature from the $\mathcal{T}_{\text{cpu}}$ vector and calculate the error for the integrator (Line 4).

Next, we calculate control signal for each input of the system as follows: we calculate the target state feedback gain for each input and subtract it from the integrator error we calculated in the previous step (Line 6). Next, we calculate the equivalent $\Theta^{\mathcal{C}}$ from the property of Equation (5.14) (Line 8). Finally, the appropriate mode is selected (Line 9), the mode change is performed (Line 11), and the pulse-width modulator is invoked for the next κ Π-length intervals (Line 12). We repeat these steps (Line 6 to 12) for each CPU core (which is corresponding to the number of control inputs, according to our model).

It is important to note that $\Theta^{\mathcal{C}}(t_\ell)$ calculated in Line 8 for a $\mathcal{C}$'th Core does not have to be equal the $\Theta^{\mathcal{C},j}$ for the selected mode; we must only select the highest mode with $\Theta^{\mathcal{C},j} \leq \Theta^{\mathcal{C}}(t_\ell)$. (If $\Theta^{\mathcal{C}}(t_\ell)$ is larger, we are only giving the mode more processing than it requires.) It should also be observed that all operations, except for finding the appropriate mode, may be done in $O(m)$ time. Finding the highest real-time performance mode that may execute can be done in $O(m \lg q)$ time (via binary search) where q is the number of real-time performance modes.

5.4 Thermal-Resiliency Calculation

In this section, we explain how to derive the real-time thermal resiliency $\Lambda(M^{\mathcal{C},i}, \mathcal{T}_{\text{ref}})$ for a given real-time performance mode $M^{\mathcal{C},i}$ and reference temperature $\mathcal{T}_{\text{ref}}$. Assume the system is in the steady-state. Therefore, the error value from the integrator output become zero. Then we get the following:

$$\mathcal{T}_{\text{ref}} = \max(\mathcal{T}_{\text{cpu}}(k)) + \mathcal{T}_{\text{env}}(k). \tag{5.15}$$

The the maximum allowable $\mathcal{T}_{\text{cpu}}(k)$ value is processor-specific and in Intel processors, it is $100\ ^{\circ}C$. Furthermore, at the steady state, we do not observe any temperature increment from the CPU. Assume that the system has reached the steady-state by the $(k-1)$'th sampling period. Therefore, $\mathcal{T}^{\mathcal{C}}_{\text{cpu}}(k) = \mathcal{T}^{\mathcal{C}}_{\text{cpu}}(k-1)$, $\mathcal{T}_{\text{env}}(k) = \mathcal{T}_{\text{env}}(k-1)$, and $\Theta^{\mathcal{C}}(k) = \Theta^{\mathcal{C}}(k-1)$. We can calculate the $\Theta^{\mathcal{C}}(k)$ at steady state. Further, in Chapter B we show that we can calculate the CPU temperature, $\mathcal{T}^{\mathcal{C}}_{\text{cpu}}(k)$ for a given $\Theta^{\mathcal{C}}(k)$ values. Since we are interested in obtaining $\Lambda(M^{\mathcal{C},i}, \mathcal{T}_{\text{ref}})$, we may fix $\mathcal{T}_{\text{ref}}$ and $\Theta^{\mathcal{C}}(k) = \Theta^{\mathcal{C},i}$.

Now now briefly outline how to obtain a solution for $\Lambda(M^{\mathcal{C},i}, \mathcal{T}_{\text{ref}})$.[2] We can calculate the steady-state CPU temperature using feedback gain equation of the controller. As we know that the steady-state $\Theta^{\mathcal{C}}$, $\mathcal{C} \in \{1 \dots m\}$ for a given external temperature is fixed, we calculate the $\Theta^{\mathcal{C}}$ for various modes and will derive the thermal-resiliency. Therefore, from our schedulability analysis, we calculate the $(\Theta^{i})_{m\times 1}$ vector, for a given mode set and the we can derive the power of the system from Equation 5.14. Our extended version of this paper shows that the feedback value can be calculated as K_0 and

$$\left(\mathcal{P}_{\text{cpu}}(t)^{\mathcal{C}}\right)_{m\times 1} = -K_0\left(\mathcal{T}^{\mathcal{C}}_{\text{cpu}}(t)\right)_{m\times 1} + \gamma_I v_e(t),$$

[2]The approach may be generalized when there is bounded steady-state error. However, the approach will be similar, and we omit the details due to space.

and therefore,

$$\begin{aligned}\Rightarrow \left(\mathcal{T}^{\mathcal{C}}_{\text{cpu}}(t)\right)_{m\times 1} &= -{K_0}^{-1}(\left(\frac{\mathcal{P}_{\text{act}}-\mathcal{P}_{\text{inc}}}{\Pi}\right)\left(\Theta^{\mathcal{C}}(k)\right)_{m\times 1} \\ &+ \mathcal{P}_{\text{inc}} + \frac{1}{\Pi}(e_{\text{act}}+e_{\text{inc}})) + {K_0}^{-1}\gamma_I v_e(t),\end{aligned} \tag{5.16}$$

is obtained and it can be easily solvable. Then, we use the Equation 5.15 to calculate the thermal resiliency.

$$\begin{aligned}\Rightarrow \left[\mathcal{T}_{\text{env}}(t)\right] &= \mathcal{T}_{\text{ref}}(t) + K^{-1}(\left(\frac{\mathcal{P}_{\text{act}}-\mathcal{P}_{\text{inc}}}{\Pi}\right)\left(\Theta^{i}(\mathcal{C})\right)_{m\times 1} \\ &- \mathcal{P}_{\text{inc}} + \frac{1}{\Pi}(e_{\text{act}}+e_{\text{inc}})) - {K_0}^{-1}\gamma_I v_e(t),\end{aligned} \tag{5.17}$$

and therefore,

$$\begin{aligned}\begin{bmatrix}\Lambda(M^{1,i},\mathcal{T}_{\text{ref}})\\ \vdots \\ \Lambda(M^{m,i},\mathcal{T}_{\text{ref}})\end{bmatrix} &= \mathcal{T}_{\text{ref}} + K^{-1}(\left(\frac{\mathcal{P}_{\text{act}}-\mathcal{P}_{\text{inc}}}{\Pi}\right)\begin{bmatrix}\Theta^{1,i}(k)\\ \vdots \\ \Theta^{m,i}(k)\end{bmatrix} \\ &- \mathcal{P}_{\text{inc}} + \frac{1}{\Pi}(e_{\text{act}}+e_{\text{inc}})) - {K_0}^{-1}\gamma_I v_e(t).\end{aligned} \tag{5.18}$$

The $\Theta^{\mathcal{C},i}(k)$ represents the minimum capacity of i'th mode on $\mathcal{C}$'th core. We will see later, from our simulations and the testbench runs, that the thermal resiliency of different cores on the same CPU does not vary too much; this is self explainable: due to the closely-coupled thermal architecture of the CPU, there is no provision on one CPU to exhibit substantially higher temperate values from rest of the cores. Therefore, we cannot necessarily expect the thermal resiliency to be a variable value for different cores.

Further, assume that we assign a predetermined capacity value to one of the cores, $\Theta^{\mathcal{C}_a}$ $\forall \mathcal{C}_a \in \{1 \ldots m\}$, $0 < \Theta^{\mathcal{C}_a} < \Pi$ and calculate the rest of the $\Theta^{\mathcal{C}}$ values. Assume the final capacity vector as $\Theta^{\mathcal{C}_f}$. As explained earlier, we can calculate the $\mathcal{T}_{\text{cpu}}$ values using $\Theta^{\mathcal{C}_f}$, and the the final resiliency value vector also can be calculated. In this case, we generate the resiliency value for the system while a fixed capacity was assigned (by design) to a specific core.

5.5 Validation

Our evaluation of the proposed method is carried out in two steps. We first generate actual system parameters from the testbed and use them in our simulations. Also, we implement our algorithm in the experimental testbed and compare the simulation results with testbed observations.

5.5.1 Simulations

We use simulations to demonstrate the validity of our proposed framework. In our simulations, we consider a system with 8 CPU cores. In our simulation, we calculate the temperature vector of cores, $\mathcal{T}_{\text{cpu}}$ and select the peak temperature from the core CPU temperature vector. Then, we calculate the control input of the system as described in our algorithm. The following task parameters are used in our simulations:

We use a partitioned scheduling. The total tasks of the system are divided into cores statically. During runtime, we consider the task migration is not possible within the cores.

- Each sporadic task $\tau_j^{\mathcal{C},i} = (e_j^{\mathcal{C},i}, d_j^{\mathcal{C},i}, p_j^{\mathcal{C},i})$ has a period $p_j^{\mathcal{C},i}$ uniformly drawn from the interval $[5, 15]$. The execution time requirement $e_j^{\mathcal{C},i}$ set to the task utilization times $p_j^{\mathcal{C},i}$, where task utilization is calculated using the UUnifast algorithm [9]. For each task, $d_j^{\mathcal{C},i}$ equals $p_j^{\mathcal{C},i}$. The tasks are scheduled by EDF in each core.

- The total number of tasks is eight for each core; each task $\tau_j^{\mathcal{C}}$ has three different real-time performance modes where $\tau_j^{\mathcal{C},2} = (e_j^{\mathcal{C},2}, d_j^{\mathcal{C},2}, p_j^{\mathcal{C},2})$; $\tau_j^{\mathcal{C},1} = (.2e_j^{\mathcal{C},1}, d_j^{\mathcal{C},1}, p_j^{\mathcal{C},1})$; and $\tau_j^{\mathcal{C},0}$ means that task is not selected. From set of all possible combinations of tasks, we have selected fifteen modes with utilizations ranging from zero to one. In order to accurately distinguish hotspots on the processor, the same tasks/modes are present on all cores.

We refer to the controller described in Algorithm 3 as *Multicore Temperature Regulated Optimized Capacity* (M-TROC). In our simulations, we do not compare the performance of our proposed method with any other algorithm, as we do not know of any other research that considers the adaptive thermal-aware scheduling on multicore hard real-time systems.

Table 5.1: Testbed parameters for multicore processor simulations

Parameter	Variable	Value
CPU Active Power	$\mathcal{P}_{\text{act}}(\Theta_{max})$	65 W
CPU Idle Power	$\mathcal{P}_{\text{inc}}(\Theta_{min})$	20 W
Server Period	Π	20 ms
Sampling Time	T_s	200 ms
Integral Gain	γ_I	0.1

The power and control parameters are given in Table 5.1. For G, H, Optimal Feedback, K_o, and Q and R matrix (in performance Index) refer to the Chapter B. We primarily use the testbed to generate the system parameters with the help of System Identification (SI). We use the SI tools provided by Matlab to derive the system state-space parameters. We use the Predictive Error Method (PEM) in SI toolbox, as it found to be better in MIMO system parameter generation process. Also we use the system parameters, generated from our testbed as the simulation parameters. We observe a matching of our testbed readings and the simulation. More details on this process are contained in the extended version of the paper [48].

5.5.2 Testbed Details

To prove our theoretical results through experiments, we have built a hardware testbed using an Intel i7-950 multicore processor running a modified Linux kernel (2.6.33.7.2-rt30 PREEMPT_RT). Our testbed consists of 8 CPU cores, 4 of them are physical cores and each physical core consists of 2 hyper threads[3]. The on-die temperature of the testbench CPU is measured through model specific registers (MSR) direly. Our software interact with the Phidgets 4-port temperature sensor board to measure the environment temperature. We develop a loadable kernel module to activate and vary the frequency modulation level of the CPU at run-time. We use to control the frequency modulation ratio in the clock and select the higher and the lowest frequency modulation indices to emulate the low and the higher power levels. We use 12.5%

[3]We consider this system as 8 core processor because each processor core has their own individual MSR and allow us to measure the individual temperature values.

and 87.5% modulation ratios in the IA32_CLOCK_MODULATION_MSR for active and inactive power mode emulation.

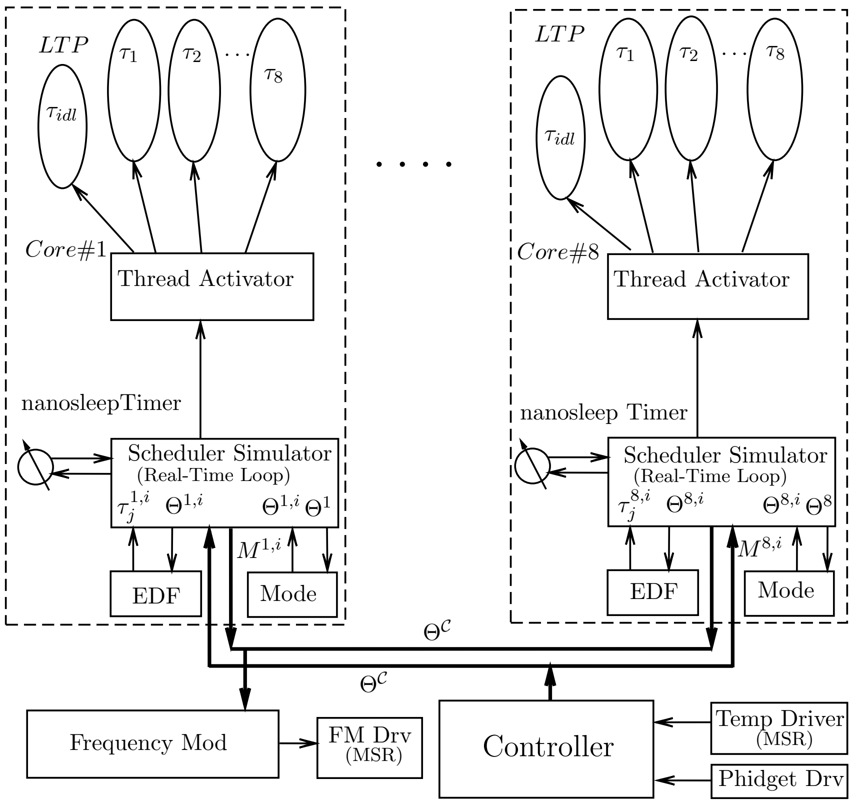


Figure 5.5: The block diagram of the testbed implementation

We develop a multi-threaded application using Linux native posix thread libraries (NTPL). Our application consists of a parallel scheduler simulator (PSS) and a thread activator. Our PSS is such that it has multiple instances of a scheduler simulator code; further, a scheduler simulator only manages a specific CPU core. Each scheduler simulator is in-charge for its own task threads (equal to number of tasks running in that core) and can access and controls these task threads; we call these task threads as local thread pool (LTP). Further, these task threads in LTP are allowed to run on a specified CPU core only. Figure 5.5 shows essential components of the implementation of our testbed.

The real-time loop runs in a very high-priority thread; the priority is set to higher than the threaded IRQ handlers. A single iteration of the real-time loop as works follows: the scheduler simulator invokes the optimal controller. Then, the optimal controller reads all the CPU core temperatures, environment temperature, and calculates the optimal $\left(\Theta^{i}\right)_{m\times 1}$ vector for the next period. The calculated $\Theta^{\mathcal{C}}$, $\mathcal{C} \in \{1 \ldots m\}$ values are applied to each scheduler simulator to select and activate their corresponding job threads from LTP. First, the scheduler simulator selects the appropriate mode corresponding to the $\Theta^{\mathcal{C}}$ value; then, schedule simulator selects the jobs (threads) based on EDF from its LTP (to sufficiently fill the Θ) and dispatches them into a thread activator contiguously. During this the entire Θ period, the CPU core is set to the highest power level. If no job (thread) is available to dispatch during Θ time interval (according to EDF), the idle task (thread) is selected. During the $\Pi - \Theta$ period, the idle job (thread) is executed, and the CPU core is set to the low power level.

The scheduler simulator emulates the schedule tick functionality in the Linux kernel in a higher level abstraction. Similar to the Linux kernel scheduler tick, the scheduler simulator, with the help of thread activator, sleeps (uses $nanosleep()$ in Linux) until it wakes up accurately in the scheduling boundaries. Our thread activator wakes up in unequal tick intervals to schedule jobs, raises the appropriate thread (from LTP) which should have the priority, and goes back to the sleeps. This process repeats and the amount of time allocates to each job depends on EDF and the total time depends on the $\Theta^{\mathcal{C}}$ given by the optimal controller. Also, the scheduler simulator (with the help of a thread activator) in each core complete the above process in parallel. When $\kappa \times \Pi$ is passed, the scheduler simulator invokes the optimal controller to calculate the $\Theta^{\mathcal{C}}$ value again.

5.5.3 Results

In Figure 5.6, the M-TROC thermal response, the utilization, and the instantaneous modes have been shown for a given fixed reference temperature $\mathcal{T}_{\text{ref}}$ temperature (equal to $83°C$) and linearly varying environment temperature, $\mathcal{T}_{\text{env}}$, from $20°C$ to $32°C$. The behavior of controller in this environment is stable and with a minimum effort the controller achieves its goal. (note the enlarged windows of the third graph) Our control algorithm selects the maximum core temperature from all the CPU cores to regulate to the reference temperature, $\mathcal{T}_{\text{ref}}$. However, the temperature graph of the simulation (and the testbed results as well) shows that the temperature values of all the CPU cores try to converge to a (nearly) same temperature value. As we mentioned early, the normal operation of a CPU does not allow to formulate substantially larger hotspots and always core temperatures normalize.

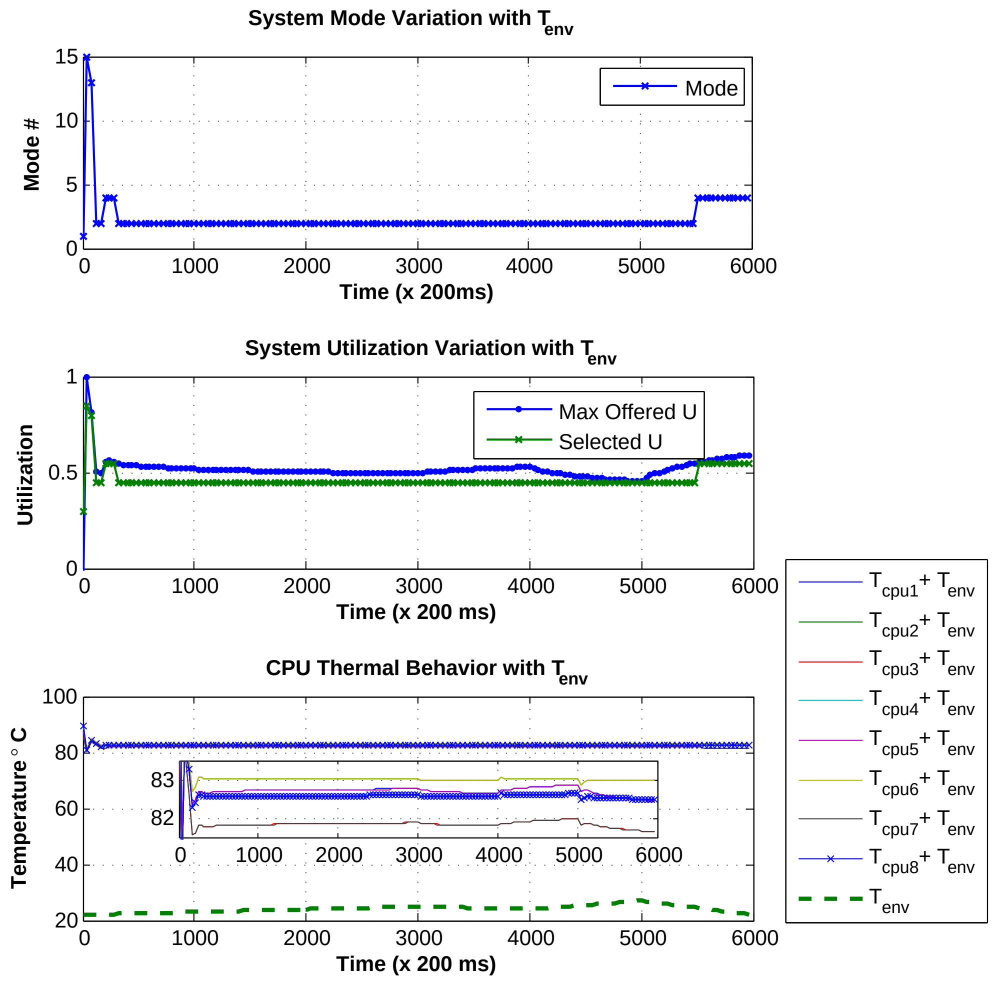


Figure 5.6: The controller capability over $\mathcal{T}_{\text{env}}$ temperature variation.

The controller is capable to maintain the $\mathcal{T}_{\text{ref}}$ over considerable $\mathcal{T}_{\text{env}}$ temperature variation. This simulation shows the $\mathcal{T}_{\text{ref}} = 83°C$ stays fixed and the $\mathcal{T}_{\text{env}}$ varies over 22 to 32 $°C$ linearly. The utilization and the modes are shown for a single core and found to be the same for all other cores (The utilization values are also the same for all the processor for last two decimal places).

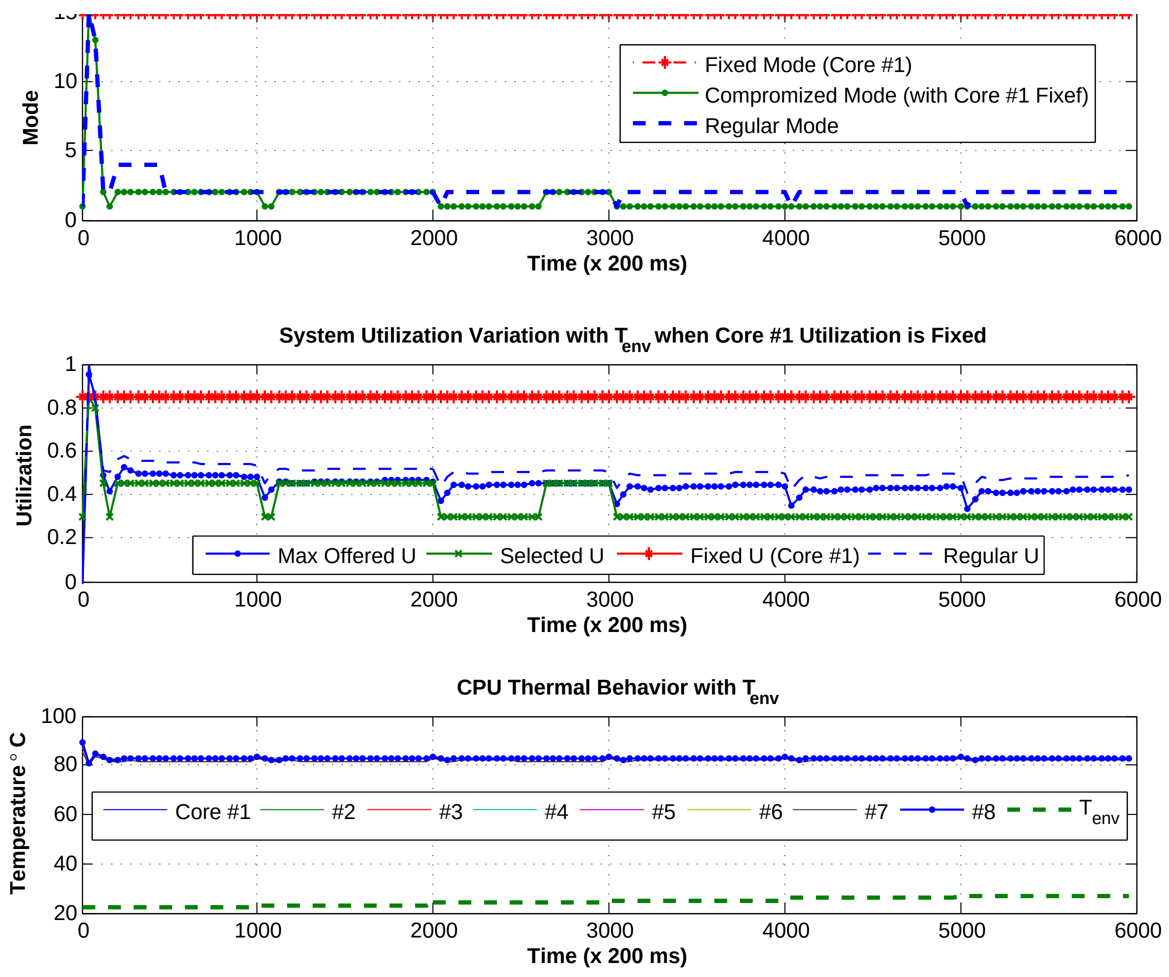


Figure 5.7: The mode variation of cores under different environment temperatures.

The mode variation of cores at $\mathcal{T}_{\text{ref}} = 83°C$ with respect to the environment temperature is shown. We have fixed a mode on core #1 and rest of the cores were controlled to seek the best possible mode The utilization and the modes are shown for a single core and found to be the same for all other cores (The utilization values are also the same for all the processor for last two decimal places).

Figure 5.7 shows the behavior of M-TROC when $\mathcal{T}_{\text{env}}$ is dynamically changed over time. The difference of this graph over the Figure 5.6 is that the $\mathcal{T}_{\text{env}}$ varies stepwise. Further notice that, we have assigned a fixed mode to the core #1. Observe that controller is able to track the reference temperature $\mathcal{T}_{\text{ref}}$ despite the sharp changes in $\mathcal{T}_{\text{env}}$. For both controllers, the utilization appropriately tracks the changes in environment temperature. When the environment temperature increases, controller decreases the system utilization and increase the utilization again when the environment temperature drops.

Regarding the real-time performance, figures displaying deadline miss ratios have been omitted as no deadline miss was experienced for controller in all the simulations. This is because we select the possible mode with available Θ so that the deadline missing does not occur. We have develop techniques to obtain the minimum safe resource capacity in the presence of mode changes and used here in calculating the anticipated mode at each mode change instance. Also, M-TROC guarantees that no deadlines are ever missed due to verification using a multi-modal schedulability test [30] as described in Section 3.3.2.

5.5.4 Experiments upon Hardware Testbed

To further confirm the validity of the theoretical results, we have run a task system with eight tasks, each with three modes in each CPU core (identical to the simulation setting), on our hardware testbed. Regarding our real-time tasks, we assign each task to perform heavy numerical calculations while executing on the system. Our hardware testbed behaves similar to the simulations of the previous subsection. Figure 5.9 presents testbed runs for a fixed environment and and varying reference temperature, $\mathcal{T}_{\text{ref}}$ settings. As suggested from the simulations, we clearly see the ability of our testbed to reach the stable stage in a shorter period. Figure 5.8 shows how the testbed behaves when the $\mathcal{T}_{\text{ref}}$ is varied comparatively slowly. Also, note that the control signal shown below shows that it reaches a steady state, which is corresponding to the Θ of a corresponding CPU core. Observe that there is a momentary drop in performance mode; however, the system soon stabilizes.

Finally, we validate our thermal resiliency calculation. Figure 5.10 shows the predicted thermal-resiliency function for our system, Figure 5.11 shows the function when Core #1 is assigned a fixed mode of 15. Notice that when either $\mathcal{T}_{\text{ref}}$ decreases or $\mathcal{T}_{\text{env}}$ increases, the predicted mode decreases as the system is becoming more thermally constrained. Unfortunately, we do not have test equipment to accurately vary the environment temperature. Thus, we consider the environment temperature to be fixed at the room temperature, $\mathcal{T}_{\text{env}} = 24°C$. Instead, we indirectly analyze the thermal resiliency function via the inverse of the thermal resiliency function $\Lambda^{-1}(M^{\mathcal{C},i}, \mathcal{T}_{\text{env}}) = \min\{\mathcal{T}_{\text{ref}} \mid \mathcal{T}_{\text{env}} \leq \Lambda(M^{\mathcal{C},i}, \mathcal{T}_{\text{ref}})\}$. Intuitively, a lower value of $\Lambda^{-1}(M^{\mathcal{C},i}, \mathcal{T}_{\text{env}})$ means the system can operate at a lower temperature and thus is more resilient than a higher value of the function. We have calculated this function multiple runs of the hardware testbed (to ensure that minor fluctuations of the air temperature do not affect the system). Figure 5.12 shows a plot of the thermal resiliency of the testbed runs when the $\mathcal{T}_{\text{ref}}$ is changed. The upper figure shows that the calculated inverse resiliency of the system increases with increasing operating mode. Most importantly, the calculated thermal resiliency tracks the actual behavior of the testbed and provides a safe upper bound on $\mathcal{T}_{\text{ref}}$ in a large majority of the cases which validates the effectiveness of the resiliency function. The lower figure shows that the calculated inverse resiliency of the system increases with increasing operating mode while a fixed mode is assigned to core $\#1$.

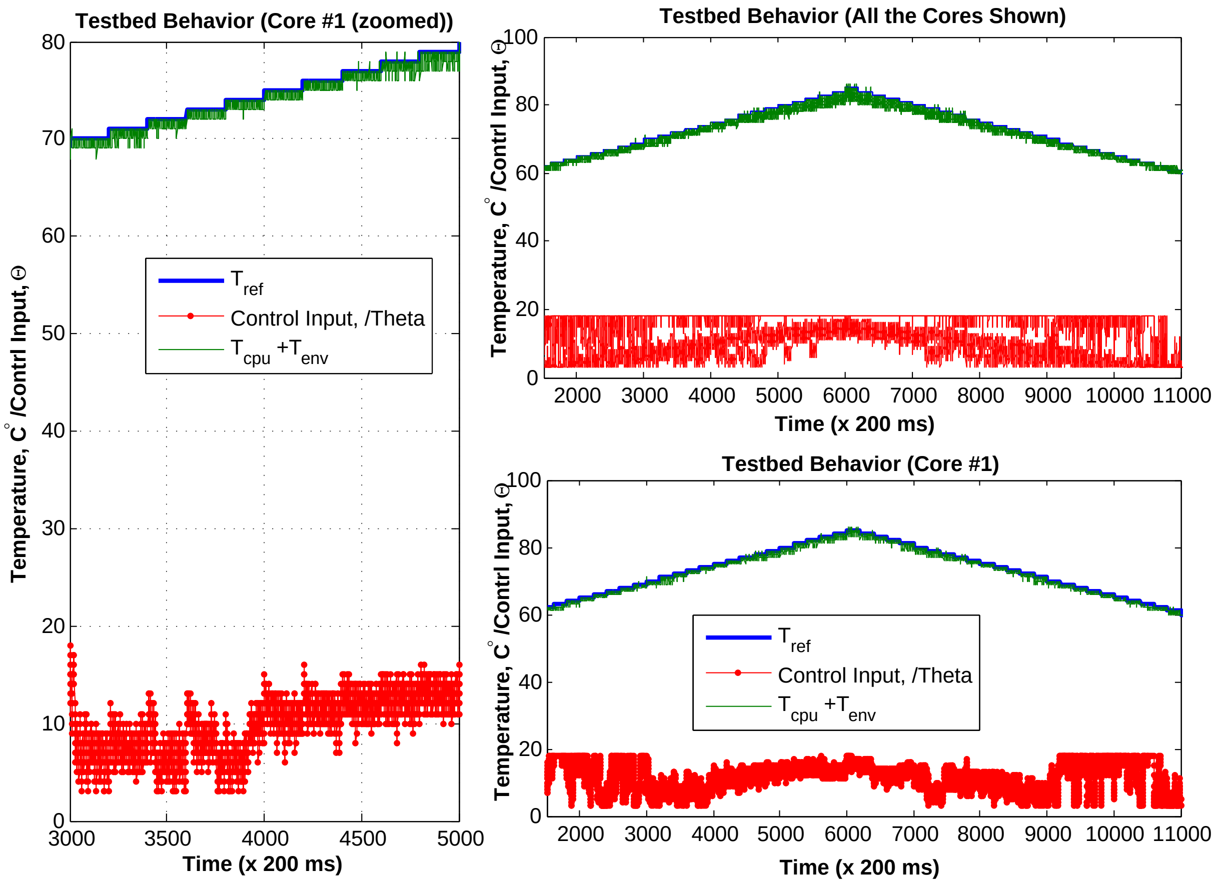


Figure 5.8: The controller tracking capability.

The controller is capable to track the $\mathcal{T}_{\text{ref}}$ over considerable range. This testbed result shows how the controller behaves when the reference temperature, $\mathcal{T}_{\text{ref}}$ varies from $60°C$ to $80°C$. The $\mathcal{T}_{\text{env}}$ stays stable over the test run.

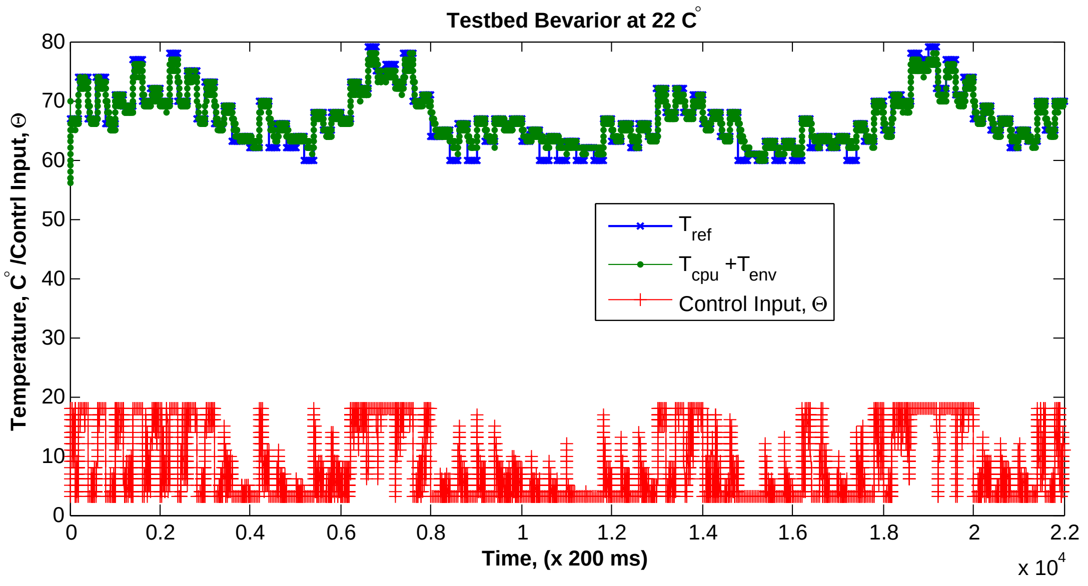


Figure 5.9: The controller behavior under $\mathcal{T}_{\text{env}}$ temperature variations.

The controller is capable to maintain the $\mathcal{T}_{\text{ref}}$ over considerable $\mathcal{T}_{\text{env}}$ temperature variation. The $\mathcal{T}_{\text{ref}} = 80°C$ stays fixed and the $\mathcal{T}_{\text{env}}$ varies over 22 to 32 $°C$ randomly.

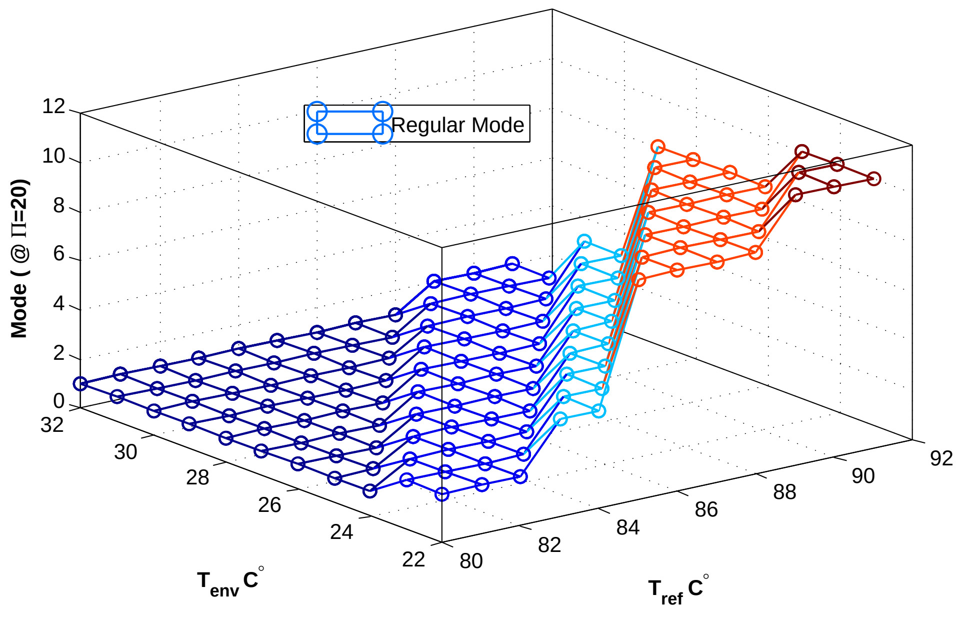


Figure 5.10: Thermal Resiliency for the system for all the possible $\mathcal{T}_{\text{env}}$ and $\mathcal{T}_{\text{ref}}$ values.

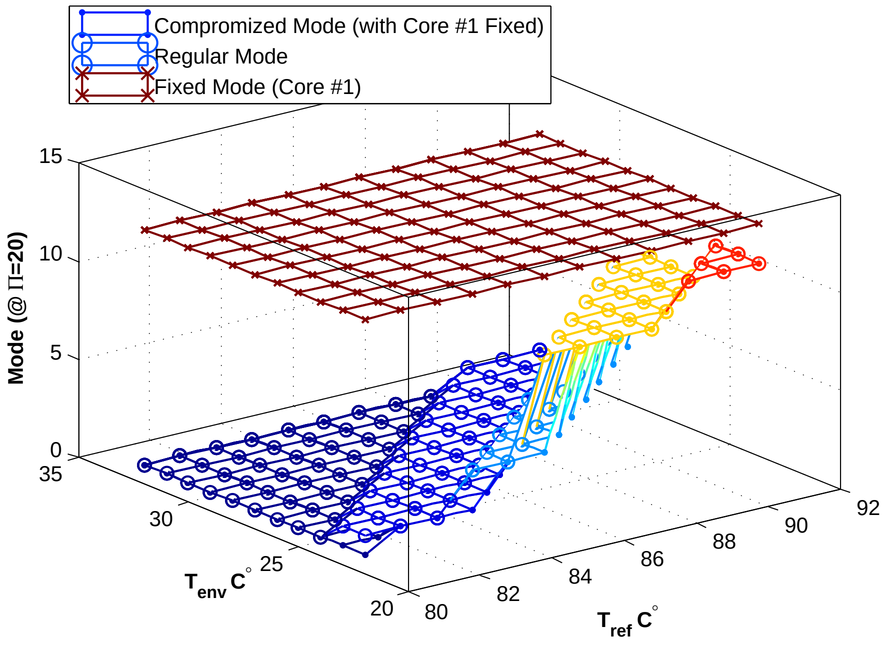


Figure 5.11: Thermal resiliency when a task is statically pinned to a core.

The available Thermal Resiliency of the system when Core #1 is assigned a fixed mode is compared with the regular thermal resiliency values for various $\mathcal{T}_{\text{env}}$ and $\mathcal{T}_{\text{ref}}$ values. This is generated from simulation results.

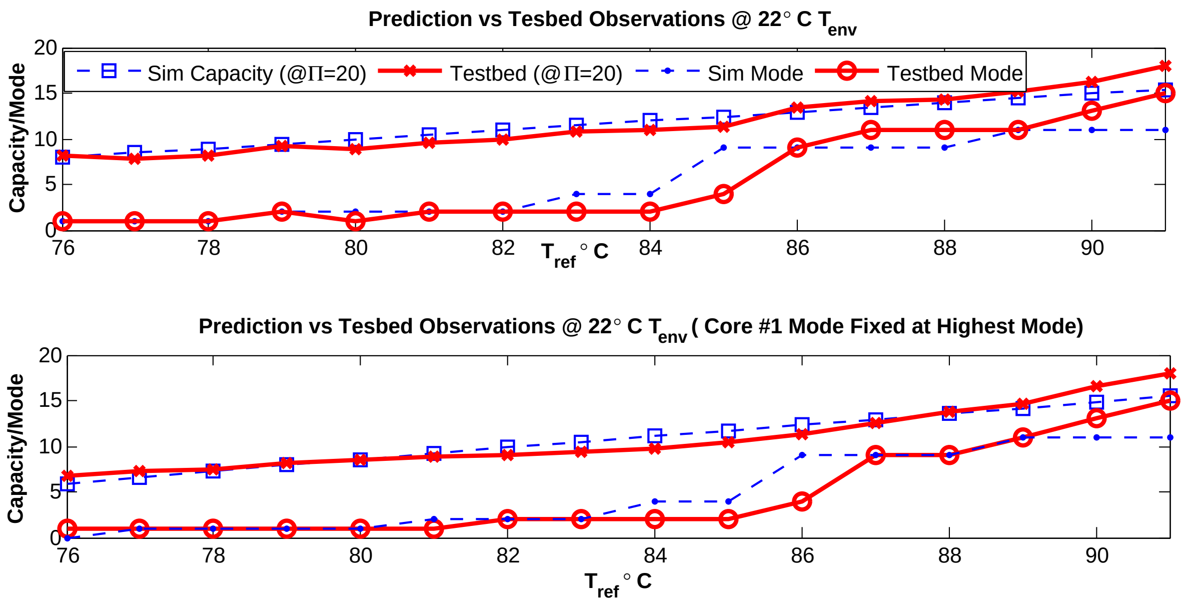


Figure 5.12: The inverse resiliency.

Due to practical difficulty to control the environment temperature accurately, we calculate $\Lambda^{-1}(M^{\mathcal{C},i}, \mathcal{T}_{\text{env}}) = \min\{\mathcal{T}_{\text{ref}} \mid \mathcal{T}_{\text{air}} \leq \Lambda(M^{\mathcal{C},i}, \mathcal{T}_{\text{ref}})\}$, which is the inverse of the thermal resiliency function. The upper figure shows the available Thermal Resiliency of the system when no restriction on core mode/capacity is enforced. The lower figure shows the available Thermal Resiliency of the system when Core #1 is assigned a fixed mode. Each figure shows a comparison of our testbed observations with simulations results.

5.6 Summary

In this chapter, we have addressed the problem of obtaining performance guarantees of multicore systems in an unpredictable thermal environment. Towards this challenge we have presented a control-theoretic thermal-stress framework using nested feedback control system, which is based on optimum control theory. The proposed framework is ideal to validate readiness of the modern hard-real-time systems for a wide range of uncertainties in system and environmental conditions.

For our system, we derive strong thermal-resiliency and hard-real-time guarantees for any real-time performance mode. Our method has the distinct advantage of being able to verify the real-time thermal resiliency of a system before it is put into operation as previous approaches which have no formal guarantee on the thermal resiliency. Further, our proposed mechanism also quantifies the thermal resiliency on multicore systems under certain capacity constraints on selected cores.

In future work, we plan to extend our framework to control designs that are more robust to model inaccuracies (e.g., $\mathcal{H}_\infty$ or model-predictive controllers). As a initial step in designing a framework for thermal stress analysis, our current design uses two RC circuits (for dynamic and leakage currents) to model the CPU temperature. We plan on extending our model to permit multiple RC circuits for heterogeneous thermal distributions and generalizing our thermal equations for more complex RC circuit layouts. We hope to derive a general-theoretic design framework that captures “resiliency” metrics for other system properties (e.g., energy, noise, etc.) and extend our analysis to other hardware settings (e.g., DVFS).

CHAPTER 6: A GENERALIZED DESIGN FRAMEWORK FOR ADAPTIVE REAL-TIME SYSTEM RESILIENCY

In the previous chapter, we explained the thermal-resiliency framework for multicore processor systems. This chapter, introduces the *generalized system-resiliency* (GSR) design framework that predictably quantifies the real-time performance level attainable under unpredictable dynamic external conditions. Our hard-real-time-ready control-theoretic framework predicts the performance-level in terms of system operating-modes. We show how the designer derives the optimal supported-modes for a multi-constrained system under various processing resources allocations while maintaining maximum system constraints.

The *system resiliency*, an extension of our previously-proposed thermal-resiliency concept, determines the maximum-withstanding limits of external stresses for a hard-real-time performance mode. This proposed framework is based on rigorous control-theoretic model and theoretically quantifies the system's graceful and predictable degradation under external stress. Finally, at the end of the chapter, we confirm our results with simulations and extensive hardware testbed runs.

This chapter presents a methodology for designing and analyzing system-resilient hard-real-time systems. Section 6.1 gives a brief introduction and Section 6.2 presents a methodology overview. Section 6.3 presents the hardware and real-time models used throughout the chapter. Section 6.4 details the design of our controller. Section 6.5 derives system resiliency function Λ for our control system. Section 6.6 gives the details of our framework applied to a case study. Section 6.7 gives the simulation and testbed results and details of the experimental setup. Finally, Section 6.8 concludes this chapter. An appendix contains supplemental information on the details of our case study and its system-resiliency calculation.

6.1 Introduction

Modern hard-real-time systems are faced with various external physical constraints. Despite hardware, functional, and design complexities, the proper system design should be temporally predictable even under varying external dynamic conditions; that is, a system designer should be able to predict the timing properties of the system as a function of the current state of the environment (e.g., what tasks can meet their deadlines might be determined by the current environmental temperature [46, 48]). Furthermore, often the system designer must balance system performance and the physical constraints for optimized design needs. For example, in a solar-powered device, a sunny day benefits the device's battery charging-level, but may negatively affect the performance of the CPU (by raising the environmental temperature and forcing the CPU to operate a lower power level to reduce thermal dissipation). Therefore, finding a balanced trade-off between these two external factors for a better system design is essential. Unfortunately, no design framework exists to support a smooth, robust design-phase and a reliable final product, while simultaneously carefully utilizing the system management capabilities and evaluating the implication of the trade-offs. In this work, we introduce a framework to predict the behavior of the system for prior unknown and unpredictable operating environment.

Our previous research (Hettiarachchi et al. [46, 48]) quantifies the trade-off between a real-time level-of-service and resource requirements, and formally defines the *thermal-resiliency*–the maximum external temperature for a given level-of-service–for both uniprocessor and multicore processor systems. However, our previous work lacks the support for a system under multiple dynamic external physical constraints. In this chapter, we propose a method for a system designer to analyze the multiple trade-offs between the performance and external constraints of a hard-real-time system. The framework we propose is a generalization of our previously-proposed thermal-resiliency framework [46][48] to a broader scope.

As an example where system-resiliency analysis is essential, consider a mission-critical real-time, solar and battery-powered surveillance robot. These systems are regularly deployed in many applications, such as surveillance, data collection, and safety-critical tasks too difficult or dangerous for humans. While a battery-powered robot has many payloads and servicing tasks, the robot's functionalities largely depend on the battery state (available energy) and the charging rate of the battery. Therefore, the battery state, the charging rate, decides the feasible processor speed and thereby the best-mode-offered by the system at any time. Therefore, system designers may consider battery state along with the schedulability analysis for predictable system design. During the design process, the system developer analyzes the real-time capability degradation with respect to the available instantaneous charging rate along with the other external factors and defines the system resiliency. In this example, the system resiliency might return the minimum charging rate required to support the most safety-critical tasks.

The abstract concept we introduce in this chapter, the *real-time system resiliency* is a system-design metric that quantifies the maximum external forces (constraints) that the system can withstand for a specified *real-time performance mode*. Towards this goal, we develop the generalized framework that calculates the real-time guarantees of a system in the presence of unpredictable external dynamic conditions. We highlight the sufficient characteristics of any system model for the system-resiliency derivation. Using our proposed method, the system designer can specify, predict *a priori*, and precisely quantify the graceful degradation of hard-real-time performance of a system due to external constraints.

The outcome of this research is orthogonal to robust control design, and it is fundamentally different. Robust controllers address the problem of designing an accurate controller in the presence of significant plant uncertainties, robust stabilization, sensitivity optimization, or other aspect of controller improvements [20]. Our control-theoretic design framework guides the controller to bring the system to a desired operating point upon unpredictable external conditions. Furthermore, our framework decides the best operating mode for the system under that external conditions. Although, the control system robustness does not affect the best operating mode selection decision process (which is purely based on the unpredictable external condition); the robustness of the controller helps to eliminate design-time errors.

This chapter gives the following contributions:

- We formally provide the conditions and requirements for system-resiliency for a system with multiple constraints.
- The proposed generic controller framework ensures the tracking of the given primary reference condition x_{ref} and does not violate system constraints. We derive the quantification of the system resiliency.
- Our framework guides the system designer the options and parameters for a predictable hard-real-time system design to achieve the performance goals.
- We used the proposed framework to design a case study and implemented this system upon our thermal/power hardware testbed.

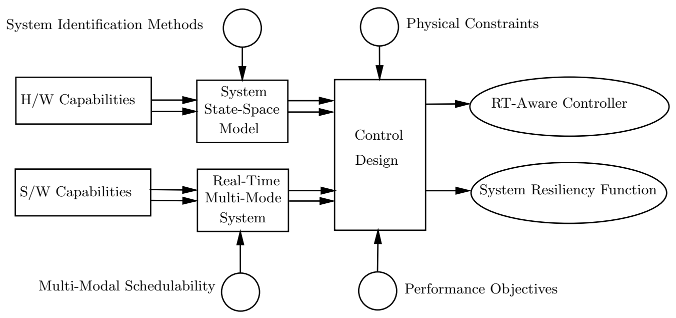


Figure 6.1: GSR Framework Methodology.

6.2 Methodology Overview

In this section, we outline the hard-real-time system design process to show the benefits of the proposed framework.

1. **System Hardware Capabilities Specification:** In this stage, the designer precisely specifies the processor and other system component capabilities (e.g., the processor's ability to adjust its frequency or voltage). The designer selects entities from the system capabilities and forms them into system input and output metrics to obtain a system state-space that describes the physical dynamics of the system. Next, the designer uses model-based design techniques such as system identification (SI) to derive the system parameters of the system. Section 6.3.1 gives more details on this step.

2. **System Software Capabilities Specification:** In this stage, the system designer determines the processing requirements of each task of the system, ranks the criticality of each task towards the system functionality, and classifies tasks into valid software modes. Next, the designer allocates resources for real-time modes and determines the minimum resource allocation under which the multi-modal system is schedulable. In a multi-modal system design, the allocation for each mode depends on potential mode-change overhead–such as carry-in processing power–as well as tasks in each mode. The research by Fisher et al. [30] details the sufficient schedulability analysis for such a multi-modal real-time system. Section 6.3.2 gives more details about this step.

3. **The Controller Design:** The system developer designs the controller upon completing the previous steps. First, the system designer defines the constraints faced by the system. These constraints characterizes the boundary conditions of the system. Section 6.4.1 gives more details on this step.

4. **The System Resiliency Calculation:** Given the real-time mode resource allocation, system model, and controller design, the designer can obtain a quantification of the system resiliency function Λ as given in the Section 6.5.

Figure 6.1 illustrates the overall design process. Our proposed multi-mode system design ensures the hard-real-time schedulability of each mode and obtain *a priori* guarantees on system resiliency by-design, that distinguishes our approach from previous system control for real-time systems.

6.3 System Models

6.3.1 System Hardware Models

In this chapter, we use the state-space model introduced in Chapter 3.3.5 to describe the underlying physical capabilities and dynamics of the system. The designer selects the system properties (control-input and output) according to the system resiliency requirements. For example, if the designer wants to derive the system resiliency in terms of external temperature and external load-current, then she needs to select the corresponding system parameters: the CPU temperature and the CPU load-current as outputs and processor active and inactive durations or CPU frequency as the control-input.

When the details of the dynamics of the system are not available, the state-space parameters can be determined using grey-box or black-box modeling techniques (system identification methods [2]). We illustrate this process in our GSR case study (Section 6.6). Also, the system designer might derive single or several such models (to represent each factor of the system resiliency) and combine them into a single state-space model. While many physical systems are non-linear in nature, a designer can often identify a piecewise linear operating regions [80] in any physical system. Thus, in this research, our focus is designing a controller for a such a linear region as evident by many research. In the future, we will focus on hybrid systems [86] that covers the complete operating space.

6.3.2 System Software Model

An adaptive real-time system requires a specification of how the underlying real-time software is affected by changes in the processor state due to changing environmental conditions. In this chapter, we use the system software model described in Chapter 3.3.6. Furthermore, we use the performance mode definition as described in Chapter 3.3.3.

6.4 Controller Model and the Design

In this research, we use the state-space model introduced in the Chapter 3.3.5 by Equation 3.1 to represent the controller model. We may choose more than one physical model to characterize the properties of the system. However, for the control design, we select only one model and calculate the feedback values accordingly. Furthermore, this controller tracks x_{ref}, the control objective. Therefore, the control-input is calculated as,

$$\begin{aligned} \rho_{\text{cpu}} &= -\mathcal{K}_{fb}\mathcal{X}_{\text{cpu}} + \mathcal{V}, \\ \Rightarrow \mathcal{X}_{\text{cpu}} &= -\mathcal{K}_{fb}^{-1}(\rho_{\text{cpu}} - \mathcal{V}), \end{aligned} \tag{6.1}$$

where, $\mathcal{K}_{fb}$ is the state-feedback matrix and $\mathcal{V}$ is a vector representing the independent terms from $\mathcal{X}_{\text{cpu}}$ of the feedback equation. Furthermore, the controller must also determine how the real-time performance modes change as the system changes. Therefore, we assume that there is a surjective function $\phi : \wp \mapsto \mathcal{M}$ that maps a given control-input ρ_{cpu} to a real-time performance mode $\mathcal{M}_{rt}$ (where $\wp$ is the set of possible control-input vectors and $\mathcal{M}$ the set of specified real-time performance modes).

6.4.1 System Constraints

Before we go into more details of the system design, we introduce a realistic assumption on the controller model that is required for our GSR framework. In our system, $\mathcal{X}_{\text{ref}}$ is an o-dimensional vector of reference values for each observable system state. The variable x_{ref} represents the primary system constraint/objective (i.e., main dimension that we wish to optimize with respect to); x_{ref} is assumed for the sake of convenience to be the first element in vector $\mathcal{X}_{\text{ref}}$. For our framework, we require that the system constraints be linearly and additively related to the real-time state-variables. This assumption holds for many realistic physical constraints (e.g., temperature, electrical current, etc.) Furthermore, we have shown the validity of this assumption for thermal constraints in our previous research [47]. The variable $\mathcal{X}_{\text{cpu}}$ (e.g., the CPU temperature, $\mathcal{T}_{\text{cpu}}$) and to a vector of external factors $\mathcal{X}_{\text{ext}}$ (e.g., the environment temperature, $\mathcal{T}_{\text{env}}$) are related to $\mathcal{X}_{\text{ref}}$ as follows:

$$\mathcal{X}_{\text{ref}} \quad - \quad \alpha\mathcal{X}_{\text{cpu}} - \beta\mathcal{X}_{\text{ext}} - \gamma \geq 0, \tag{6.2}$$

where, α and β are o-dimensional square matrices and γ is o-dimensional vector of positive constants (determined by the physical dynamics of the system).

6.4.2 Control Integration of Design Objectives

Our controller design should meet the following goals: *We design a controller that automatically adjusts the ρ_{cpu}, the control-input, ensuring the non-violation of the constraints–factors of the system-resiliency.* In systems where the primary target objective, x_{ref} is related to other physical constraints (e.g., if CPU current is the primary target dimension, this is also related to other potential secondary constraints such as CPU temperature), we may include the following addition to the controller. We define the following error vector: $\mathcal{X}_{error} \overset{\text{def}}{=} \mathcal{X}_{\text{ref}} - \alpha\mathcal{X}_{\text{cpu}} - \beta\mathcal{X}_{\text{ext}} - \gamma$. Here $\mathcal{X}_{error}[j]$ represents the error with respect to the j'th system constraint (i.e., the j'th entry in the error vector). If the integrated error value $\int K_j\mathcal{X}_{error}[j]dt$ is negative (where K_j is an integration constant), then the current system state has exceeded the j'th constraint and we may have to adjust the primary objective to bring the system into a state where the j'th constraint is satisfied. The primary objective (x_{ref}) reduction also reduces the control-input, ρ_{cpu} and adjusts the system-states accordingly. We denote this adjusted constraint as $\overline{x_{\text{ref}}}$ and calculate it as follows:

$$\overline{x_{\text{ref}}} \quad = \quad x_{\text{ref}} + \bigoplus_{j=2}^{o}\left[\min(0, \int K_1[j]\mathcal{X}_{error}[j]dt)\right], \tag{6.3}$$

where $\bigoplus$ is a mathematical operator that determines the reduction in the primary reference for a system with multiple constraints (e.g., $\bigoplus$ could be the $\max$ operator or weighted summation). The mathematical operation $\bigoplus$ is determined by the control design and applied for all the constraints other than primary system constraint/objective.

6.5 System Resiliency Calculation

We are now prepared to define the real-time system-resiliency metric and outline the process by which it may be calculated.

Definition 6 (System-Resiliency)**.** *The* **system-resiliency***,*

$$\mathcal{X}_{\text{ext}} \overset{def}{=} \Lambda(\mathcal{M}_{rt}, \mathcal{X}_{\text{ref}}), \tag{6.4}$$

is the maximum external stress that a given hard-real-time performance mode, $\mathcal{M}_{rt}$ can withstand on the processor for a given reference vector, $\mathcal{X}_{\text{ref}}$. The variable, $\mathcal{X}_{\text{ext}}$ represents the maximum external forces.

System resiliency can be calculated from the state-space equations and system constraint inequalities. Unfortunately, due to the complexity of the systems, it is not always possible to solve a multiple-constraints system to a closed-form. In such cases, we use the difference-form of the equations to solve the system for an exact answer. At the steady-state, control input or state variables remain stationary. (We may also relax this assumption to account for a small steady-state error). Assume that the system reached the steady-state by the $(k-1)$'th sampling period, then instantaneous CPU states becomes, $\mathcal{X}_{\text{cpu}}(k) = \mathcal{X}_{\text{cpu}}(k-1)$. Also, from the difference-form of the Equation 3.1, we get, $\rho_{\text{crl}}(k) = \rho_{\text{crl}}(k-1)$. Therefore, we may calculate $\mathcal{X}_{\text{cpu}}(k)$ for a given $\rho_{\text{crl}}(k)$ value from Equations 3.1 and 6.1. At this stable state, the $\Lambda(\mathcal{M}_{rt}^{\mathcal{C},i}, \mathcal{X}_{\text{ref}})$ is solved as follows:

1. We fix $\mathcal{X}_{\text{ref}}$, the reference constraints, as specified in the system resiliency function parameter.
2. We fix control-input, $\rho_{\text{crl}}(k)$ by inverting the ϕ function for the given mode defined in Section 6.4. (Here we use the minimum value of ρ_{crl} that maps to the mode $\mathcal{M}_{rt}$).
3. After fixing the above terms, the resulting system of difference equations (explained above) may be solved using standard techniques and tests for uniqueness of solutions [81].

This completes the GSR framework description. The next section will illustrate how to apply some of the abstract concepts introduced in the first half of this paper to an actual hardware implementation and illustrate how to use difference equations to obtain an exact solution for system resiliency.

6.6 GSR Framework Case Study

In this section, we design and implement on hardware a predictable system (utilizing our proposed GSR framework) with multiple physical constraints executing under varying external conditions. The physical constraints upon our system are the battery load and external temperature denoted as $\mathcal{I}_{ext}$ and $\mathcal{T}_{\text{env}}$ respectively. Our goal is to evaluate the system's ability to track the primary system objective and satisfy the secondary constraints. Furthermore, we also validate our framework's ability to accurately predict the real-time performance degradation with respect to different external load conditions and temperature conditions (i.e., the system resiliency). Our hardware implementation and the associated constraints might potentially be applicable to settings such as the surveillance-robot example in the introduction where the device has to operate under changing battery (i.e., change the load to prolong battery life) and changing temperature conditions.

6.6.1 System Hardware Model

In our case study, we leverage our previously-developed platform for thermal-resiliency calculation [46, 48] and extend it to support the primary objective of tracking a given load reference point $\mathcal{I}_{ref}$ subject to thermal constraint $\mathcal{T}_{\text{ref}}$. Our platform utilizes only two power modes $\mathcal{P}_{\text{act}}$ and $\mathcal{P}_{\text{inc}}$ (active and inactive power, respectively) and uses pulse-width modulation (PWM) to approximate any specified power level. The PWM uses period of length Π with a power-active period (duty cycle) of Θ (i.e., resource capacity) and power-inactive period of $\Pi - \Theta$ for each cycle. Throughout the paper, we assume that Θ and Π are expressed in seconds. We also set Π equal to 20 seconds. In the following, we show how to formulate the resource capacity, Θ, as the control input. We denote the $\mathcal{I}_{cpu}(t)$ as CPU load current, $\mathcal{P}_{\text{cpu}}(t)$ as the instantaneous CPU power, $\mathcal{X}_{\bar{I}}(t)$ as a state variable to represent CPU-load state at time t. Also, the CPU power can be used to control the CPU current. Therefore,

$$\begin{aligned} \dot{\mathcal{X}}_{\bar{I}}(t) &= \mathcal{A}_I \mathcal{X}_{\bar{I}}(t) + \mathcal{B}_{\bar{I}} \mathcal{P}_{\text{cpu}}(t), \\ \mathcal{I}_{c\bar{p}u} &= \mathcal{C}_I \mathcal{X}_{\bar{I}}. \end{aligned} \tag{6.5}$$

Since, the instantaneous power can be represented by means of resource capacity, Θ and resource period, Π, We get,

$$\mathcal{P}_{\text{cpu}}(t) \quad = \quad \frac{\mathcal{P}_{\text{act}}\Theta(t) + \mathcal{P}_{\text{inc}}(\Pi - \Theta(t))}{\Pi}. \tag{6.6}$$

Therefore, from Equation 6.5, we get,

$$\dot{\mathcal{X}}_{\bar{I}}(t) \quad = \quad \mathcal{A}_I \mathcal{X}_{\bar{I}}(t) + \mathcal{B}_{\bar{I}}(\frac{\mathcal{P}_{\text{act}}\Theta(t) + \mathcal{P}_{\text{inc}}(\Pi - \Theta(t))}{\Pi}). \tag{6.7}$$

To simplify the state-space expression, we wish to reformulate the system dynamics to remove the power constants and have the control input depend only upon Θ. Towards this goal, we observe the system at a special time instance, t_∞, which denotes a stable point of the system. As we have showed in our previous work [47], we derive the feedback parameters for an stable optimal controller. From Equation 6.7,

$$\begin{aligned}
\dot{\mathcal{X}}_{\bar{I}}(t_\infty) &= \mathcal{A}_I \mathcal{X}_{\bar{I}}(t_\infty) \\
&+ \mathcal{B}_{\bar{I}}(\frac{\mathcal{P}_{\text{act}}\Theta(t_\infty) + \mathcal{P}_{\text{inc}}(\Pi - \Theta(t_\infty))}{\Pi}), \\
\mathcal{I}_{c\bar{p}u}(t_\infty) &= \mathcal{C}_I \mathcal{X}_{\bar{I}(t_\infty)}.
\end{aligned} \tag{6.8}$$

We then redefine our system equations in reference to the system load state at time t_∞. Thus, define $\mathcal{X}_I(t) \stackrel{\text{def}}{=} \mathcal{X}_{\bar{I}}(t) - \mathcal{X}_{\bar{I}}(t_\infty)$, $\Theta(t) \stackrel{\text{def}}{=} \Theta(t) - \Theta(t_\infty)$, $\mathcal{I}_{cpu} \stackrel{\text{def}}{=} \mathcal{I}_{c\bar{p}u}(t) - \mathcal{I}_{c\bar{p}u}(t_\infty)$, and $\mathcal{B}_I \stackrel{\text{def}}{=} \mathcal{B}_{\bar{I}}\frac{(\mathcal{P}_{\text{act}} - \mathcal{P}_{\text{inc}})}{\Pi}$. Then Equations 6.7 and 6.8 give,

$$\begin{aligned}
\dot{\mathcal{X}}_I(t) &= \mathcal{A}_I \mathcal{X}_I(t) + \mathcal{B}_I \Theta(t). \\
\mathcal{I}_{cpu}(t) &= \mathcal{C}_I \mathcal{X}_I(t).
\end{aligned} \tag{6.9}$$

We use the above load-current model as the main controller. We apply the same normalization for the temperature subsystem[1],

[1] We have shown in our previous work [46] that the temperature can be controlled by means of power consumption of the system.

$$\begin{aligned} \dot{\mathcal{X}}_T(t) &= \mathcal{A}_T\mathcal{X}_T(t) + \mathcal{B}_T\Theta(t), \\ \mathcal{T}_{\text{cpu}}(t) &= \mathcal{C}_T\mathcal{X}_T(t), \end{aligned} \tag{6.10}$$

where, $\mathcal{X}_{I/T}$ represents the current/temperature state-variables and $\mathcal{A}_{I/T}, \mathcal{B}_{I/T}$, and $\mathcal{C}_{I/T}$ represent state-space parameters for the current and thermal models.

Therefore,

$$\begin{aligned} \begin{bmatrix} \dot{\mathcal{X}}_I(t) \\ \dot{\mathcal{X}}_T(t) \end{bmatrix} &= \begin{bmatrix} \mathcal{A}_I & 0 \\ 0 & \mathcal{A}_T \end{bmatrix} \begin{bmatrix} \mathcal{X}_I(t) \\ \mathcal{X}_T(t) \end{bmatrix} + \begin{bmatrix} \mathcal{B}_I & 0 \\ 0 & \mathcal{B}_T \end{bmatrix} \Theta(t), \\ \Rightarrow \dot{\mathcal{X}}_{\text{cpu}} &= \mathcal{A}\mathcal{X}_{\text{cpu}} + \mathcal{B}\rho_{\text{crl}}, \end{aligned} \tag{6.11}$$

.

where, $\mathcal{X}_{\text{cpu}} \overset{\text{def}}{=} \begin{bmatrix} \mathcal{X}_I(t) \\ \mathcal{X}_T(t) \end{bmatrix}$, $\rho_{\text{crl}} \overset{\text{def}}{=} \begin{bmatrix} \Theta(t) \\ \Theta(t) \end{bmatrix}$, $\mathcal{A} \overset{\text{def}}{=} \begin{bmatrix} \mathcal{A}_I & 0 \\ 0 & \mathcal{A}_T \end{bmatrix}$, and $\mathcal{B} \overset{\text{def}}{=} \begin{bmatrix} \mathcal{B}_I & 0 \\ 0 & \mathcal{B}_T \end{bmatrix}$ can be derived.

Furthermore,

$$\begin{aligned} \begin{bmatrix} \mathcal{I}_{cpu}(t) \\ \mathcal{T}_{\text{cpu}}(t) \end{bmatrix} &= \begin{bmatrix} \mathcal{C}_I & 0 \\ 0 & \mathcal{C}_T \end{bmatrix} \begin{bmatrix} \mathcal{X}_I(t) \\ \mathcal{X}_T(t) \end{bmatrix}, \\ \Rightarrow \mathcal{Y}_{cpu} &= \mathcal{C}\mathcal{X}_{cpu}, \end{aligned} \tag{6.12}$$

where, $\mathcal{Y}_{cpu} \overset{\text{def}}{=} \begin{bmatrix} \mathcal{I}_{cpu}(t) \\ \mathcal{T}_{\text{cpu}}(t) \end{bmatrix}$ and $\mathcal{C} \overset{\text{def}}{=} \begin{bmatrix} \mathcal{C}_I & 0 \\ 0 & \mathcal{C}_T \end{bmatrix}$.

We assume that no further details of the dynamics of the system are available; thus, the state-space parameters are determined using system identification methods, as briefly described in Section 6.7.1. Furthermore, we have shown the thermal model derivation for the processor using first principles (white-box modeling) in our previous work [47].

6.6.2 Hardware Testbed Details

The implementation of the model described in the previous subsection and used in the experiments of Section 6.7 is upon our custom-built hardware testbed based on Intel i3-4130 processor with modified Linux kernel (PREEMPT_RT). Unlike our previous testbed implementations, in this research we measure and control processor temperature and current using our control algorithm. We use model specific registers (MSR) to measure the processor core temperatures and Phidgets temperature sensor board to measure the environment temperature. We control the frequency scaling level and select the higher and the lowest frequency scaling indices to emulate the low and the higher power levels of the CPU; this was done by dynamically updating the MSR with correct frequency scaling parameters for the active and inactive power mode emulation. We measure the CPU power consumption using MSR_PKG_ENERGY_STATUS MSR and external load power and total power consumption by NI PXIe-1082 (6363) data acquisition system. We use the 3, 10Ω resistors in parallel as our load; this load represents some external current draw $\mathcal{I}_{ext}$ that may reduce the amount of current that our CPU may draw (and thus reduce the mode we are able to support). We are unable to measure the individual core power consumption in our testbed. Therefore, for simplicity, we configure our testbed to emulate a single-core system by executing same resource capacity, $\Theta(t)$, on all cores simultaneously at any given time. Further testbed details are available in the appendix of this paper.

6.6.3 System Software Model

We allocate resources for real-time modes and determines the minimum resource allocation under which the multi-modal system is schedulable as explained in Section 3.3.2. Furthermore, once we define the modes and their criticality, we also find the relation of the modes of the system, $\mathcal{M}_{rt}$–the modes vector– and the resource capacity vector. We can then derive

$$\begin{aligned} \mathcal{M}_{rt} &= \phi(\rho_{\text{crl}}) = \phi(\Theta), \\ \Rightarrow \Theta &= \phi^{-1}(\mathcal{M}_{rt}), \end{aligned} \tag{6.13}$$

where, ϕ function maps different control-inputs (in this case Θ) to modes. Section 6.7.1 gives the parameters of the task systems that comprise $\mathcal{M}$. For the remainder of this section, we just need to know

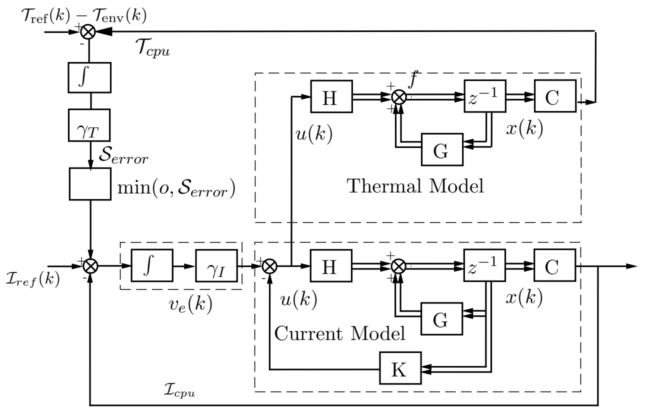


Figure 6.2: The RAS controller.

In this control design, load-current-states are in the main controller and thermal-states are used to adjust the main reference as auxiliary adjuster.

that ϕ essentially partitions the range $[0, \Pi]$ into intervals. For instance, ϕ assigns $M^{\mathcal{C},i_1}$ to the interval $[\Theta^{\mathcal{C},i_1}, \Theta^{\mathcal{C},i_2})$, $M^{\mathcal{C},i_2}$ to the interval $[\Theta^{\mathcal{C},i_2}, \Theta^{\mathcal{C},i_3})$, and so on.

6.6.4 Controller Design

As shown in Figure 6.2, our controller should be able to reduce its reference value, $\mathcal{I}_{ref}$ upon the system's attempt to violate constraint thermal constraint, $\mathcal{T}_{\text{ref}}$. Therefore, we design a structure with this characteristic and call it a *reference self-adjust* (RSA) controller. First, we define thermal error, $\mathcal{T}_{error}$ and accumulated thermal error, $\mathcal{S}_{error}$ as follows.

$$\begin{aligned} \mathcal{T}_{error}(t) &\overset{\text{def}}{=} \mathcal{T}_{\text{ref}}(t) - \mathcal{T}_{\text{cpu}}(t) - \mathcal{T}_{\text{env}}(t), \\ \mathcal{S}_{error} &\overset{\text{def}}{=} \gamma_T \int \mathcal{T}_{error}(t)dt, \end{aligned} \tag{6.14}$$

where, $\mathcal{T}_{\text{cpu}}$ denotes the CPU temperature. In this design, to maintain the non-violating reference points, we adjust the current reference, $\mathcal{I}_{ref}$ by adding $\min(0, \mathcal{S}_{error})$ when it is negative. We can then calculate the feedback of the system as,

$$\begin{aligned}\Theta(t) &= -\mathcal{K}_{sfb}\mathcal{X}_I(t) \\ &+ \int \mathcal{K}_{ifb}(-\mathcal{C}_I\mathcal{X}_I(t) + \mathcal{I}_{ref} + \min(0, \mathcal{S}_{error}))dt,\end{aligned} \tag{6.15}$$

where, $\mathcal{K}_{sfb}$ denotes the state feedback and $\mathcal{K}_{ifb}$ denotes the integral feedback values of the primary system. (The detailed derivations of the above state feedback parameters satisfying the system stability can be obtained by following techniques used in our previous paper [48]).

6.6.5 System Constraints

We now identify the constraints of the system. First, from Kirchhoff's current law, we get (load-current constraint),

$$\mathcal{I}_{ext} = \mathcal{I}_{ref} - \mathcal{I}_{cpu}, \tag{6.16}$$

where $\mathcal{I}_{ref}$ is the total battery current.

Next, we denote the instantaneous CPU power consumption by $\mathcal{W}_{cpu}$, the equivalent CPU impedance by r_{cpu}, and external load impedance by r_{ext}. The CPU resistance, r_{cpu} and power consumption, $\mathcal{W}_{cpu}$ are dependent upon the resource capacity, and we determine their correspondence using $cftool$ in Matlab and given in the appendix. We select sinusoidal approximation for processor power and higher order polynomial approximation for processor resistance to optimize the matching function selection. Furthermore, we confirm the relationship of CPU impedance with resource capacity, Θ, through our testbed observations. Figure B.2 in the appendix shows the CPU impedance variation over the resource capacity. Observe that when Θ is increased, the CPU impedance decreases. This result is justifiable, as for a system to consume a larger power, under fixed voltage, the CPU internal impedance should be reduced. Similarly, Figure 6.3 shows the power and Θ variation of the testbed. We use this CPU impedance values in our simulations. Furthermore, the CPU impedance and the external load impedance are arranged in parallel. Therefore, we derive the battery-load-resiliency, the maximum external load, $\mathcal{I}_{ext}$ that can be tolerated by the system for a given mode as the following Equation 6.17:

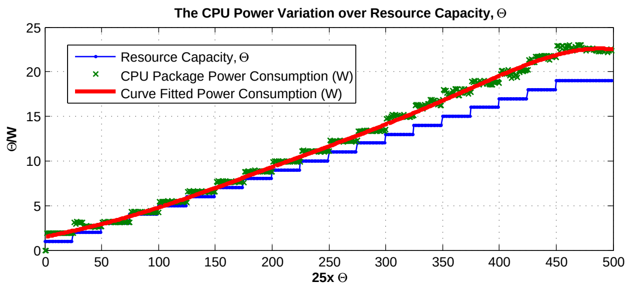


Figure 6.3: The testbed CPU power variation.

The power variation of the testbed CPU over the resource capacity, Θ for fixed 3.33Ω external load, r_{ext}.

$$\begin{aligned} \mathcal{I}_{ext} &= \frac{\mathcal{W}_{cpu}}{\mathcal{V}_{bat}} \cdot \frac{r_{ext} + r_{cpu}}{r_{ext}} - \frac{\mathcal{W}_{cpu}}{\mathcal{V}_{bat}}, \\ &= \frac{1}{\mathcal{V}_{bat} \cdot r_{ext}} \cdot r_{cpu} \cdot \mathcal{W}_{cpu}, \\ &= \frac{1}{\mathcal{V}_{bat} \cdot r_{ext}} \cdot r_{cpu}(\Theta) \cdot \mathcal{W}_{cpu}(\Theta). \end{aligned} \tag{6.17}$$

We proceed to the thermal constraints analysis to complete the system resiliency. At stability, the system meets the thermal-constraint:

$$\mathcal{T}_{\text{ref}} \geq \mathcal{T}_{\text{cpu}} + \mathcal{T}_{\text{env}}. \tag{6.18}$$

To calculate the system temperature, $\mathcal{T}_{\text{cpu}}$, we have to calculate CPU state variable, $\mathcal{X}_T$. Due to the complexity of the differential equations, we cannot derive a closed-form solution for the $\mathcal{X}_T$. However, a system of difference equations will lead to a solution. Therefore, we start with following equivalent difference equations from the previously-derived equations. We assume sampling time as T_s and used abused notation to represent kT_s interval by k. Also, we denote $\mathcal{G}_{T/I}$ and $\mathcal{H}_{T/I}$ as discrete state-space parameters. From Equation 6.9 and Equation 6.10, we obtain

$$\begin{aligned}
\mathcal{X}_I(k+1) &= \mathcal{G}_I\mathcal{X}_I(k) + \mathcal{H}_I\Theta(k),\\
\mathcal{I}_{cpu}(k) &= \mathcal{C}_I\mathcal{X}_I(k),\\
\mathcal{X}_T(k+1) &= \mathcal{G}_T\mathcal{X}_T(k) + \mathcal{H}_T\Theta(k),\\
\mathcal{T}_{\text{cpu}}(k) &= \mathcal{C}_T\mathcal{X}_T(k). \qquad (6.19)
\end{aligned}$$

Furthermore, from Equation 6.14, we obtain

$$\begin{aligned}
\mathcal{T}_{error}(k) &= \mathcal{T}_{\text{ref}}(k) - \mathcal{T}_{\text{cpu}}(k) - \mathcal{T}_{\text{env}}(k),\\
\mathcal{T}_{error}(k) &= \mathcal{T}_{\text{ref}}(k) - \mathcal{C}_T\mathcal{X}_T(k) - \mathcal{T}_{\text{env}}(k),\\
\mathcal{S}_{error}(k) &= \mathcal{S}_{error}(k-1) + \gamma_T T_s \mathcal{T}_{error}(k), \qquad (6.20)
\end{aligned}$$

and, from Equation 6.15,

$$\begin{aligned}
\mathcal{I}_{error}(k) &= -\mathcal{C}_I\mathcal{X}_I(k) + \mathcal{I}_{ref}(k) + \min(0, \mathcal{S}_{error}(k)),\\
\Theta_{error}(k) &= \Theta_{error}(k-1) + T_s\mathcal{K}_{ifb}\mathcal{I}_{error}(k),\\
\Theta(k) &= \Theta_{error}(k) - \mathcal{K}_{sfb}\mathcal{X}_I(k). \qquad (6.21)
\end{aligned}$$

From the above difference equations, we may obtain $\mathcal{T}_{\text{env}}$ by fixing $\mathcal{T}_{\text{ref}}$, $\mathcal{I}_{ref}$, and Θ. Let $\psi_{cpu}(\mathcal{X}_{\text{ref}}, \Theta)$ represent this solution.

6.6.6 Resiliency Calculation

The following expression for system resiliency is derived from Equation 6.17 and the solution to the system of Equations 6.19, 6.20, and 6.21.

$$
\begin{aligned}
&\Lambda(\mathcal{M}_{rt}, \mathcal{X}_{\text{ref}}) \\
&\quad = \begin{bmatrix} \mathcal{I}_{ext} \\ \mathcal{T}_{\text{env}} \end{bmatrix}, \\
&\quad = \begin{bmatrix} \frac{1}{\mathcal{V}_{bat} \cdot r_{ext}} \cdot r_{cpu}(\Theta_{rt}) \cdot \mathcal{W}_{cpu}(\Theta_{rt}) \\ \psi_{cpu}(\mathcal{X}_{\text{ref}}, \Theta_{rt}) \end{bmatrix}, \\
&\quad = \begin{bmatrix} \frac{1}{\mathcal{V}_{bat} \cdot r_{ext}} \cdot r_{cpu}(\phi^{-1}(\mathcal{M}_{rt})) \cdot \mathcal{W}_{cpu}(\phi^{-1}(\mathcal{M}_{rt})) \\ \psi_{cpu}(\mathcal{X}_{\text{ref}}, \phi^{-1}(\mathcal{M}_{rt})) \end{bmatrix}.
\end{aligned} \tag{6.22}
$$

6.6.7 Discrete-Time Controller

The controller pseudocode is presented in Algorithm 3. The controller proposed here performs servo action of the load-current controller and load-current reference adjustment action on behalf of the thermal model. At the beginning, the controller initializes the state-variable and integrated error values once. The first step is to obtain the sample CPU temperature and CPU load-currant (Line 3 of Algorithm 3). The error is then calculated by taking the difference between the reference temperature and the CPU temperature plus environment temperature (Line 4). This error is integrated into the error vector and added to vector sum of the integrated error in the next line (Line 5). Next, if the previously calculated integrated error is negative, the load-current reference is lowered by adding this error value and load-current error is formulated (Line 6). This load-current error values is integrated into the error vector and added to vector sum of the integrated error to formulate the load-current integrated error (Line 7). Finally the Θ value is calculated with previously calculated integrated error (in Line 7) and with the current state-feedback value (Line 8). Next, the appropriate mode is selected (Line 9), the mode change is performed (Line 11), and the resource capacity is set for the next Π-length intervals (Line 12). It is important to note that $\Theta(t_\ell)$ calculated in Line 8 does not have to be equal the $\Theta^{(j)}$ for the selected mode; we must only select the highest mode with $\Theta^{(j)} \le \Theta(t_\ell)$. (If $\Theta(t_\ell)$ is larger, we are only giving the mode more processing than it requires.) It should also be observed that all operations, except for finding the appropriate mode, may be done in $O(1)$ time. Finding the highest real-time performance mode that may execute can be done in $O(\lg q)$ time (via binary search) where q is the number of real-time performance modes.

Algorithm 3 Control Algorithm

```
Require: Reference Current $\mathcal{I}_{ref}$; Reference Temperature $\mathcal{T}_{\text{ref}}$; Feedback Gain $\mathcal{K}_{sfb}$; Integral Constant $\mathcal{K}_{ifb}, \mathcal{K}_i$; PWM Period $\Pi$.
1: Initialize $\mathcal{X}_I, Tot\_\dot{v}_e, Tot\_\dot{I}_e$
2: while At beginning of sampling period $[t_\ell, t_{\ell+1}) : t_\ell \equiv \ell\Pi$ do
3:     Sample $\mathcal{T}_{\text{cpu}}(t_\ell) + \mathcal{T}_{\text{env}}(t_\ell), \mathcal{I}_{cpu}(\ell)$.
4:     $\dot{v}_e(t_\ell) = \mathcal{T}_{\text{ref}} - (\mathcal{T}_{\text{cpu}}(t_\ell) + \mathcal{T}_{\text{env}}(t_\ell))$
5:     $Tot\_\dot{v}_e(t_\ell) = Tot\_\dot{v}_e(t_{\ell-1}) + \mathcal{K}_i\Pi\dot{v}_e(t_\ell)$
6:     $\dot{I}_e(t_\ell) = \mathcal{I}_{ref} - (\mathcal{I}_{cpu}(t_\ell) - \min(0, Tot\_\dot{v}_e(t_\ell)))$
7:     $Tot\_\dot{I}_e(t_\ell) = Tot\_\dot{I}_e(t_{\ell-1}) + \mathcal{K}_{ifb}\Pi\dot{I}_e(t_\ell)$
8:     $\Theta(t_\ell) = Tot\_\dot{I}_e(t_\ell) - \mathcal{K}_{sfb}\mathcal{X}_I(t_\ell)$
9:     $i = \max\{j \in \mathbb{Z}_{q+1} \mid \Theta^{(j)} \le \Theta(t_\ell)\}$
10:    $\mathcal{X}_I(t_{\ell+1}) = \mathcal{G}_I\mathcal{X}_I(t_\ell) + \mathcal{H}_T\Theta(t_\ell)$
11:    Update real-time performance mode to $M_i$.
12:    Set resource period of $\Pi$ and resource capacity of $\Theta(t_\ell)$.
13: end while
```

6.7 Validation

In this research, we evaluate the proposed method using simulations and verified the results on our hardware testbed.

6.7.1 Simulations

We implemented a discrete-event simulation in Matlab that uses the system parameters/model derived from our testbed. At each time tick, the simulator calculates the CPU load-current consumption, $\mathcal{I}_{cpu}$, the feedback, the target resource capacity, Θ (control-input), and the $\mathcal{T}_{\text{cpu}}$ value. If the temperature of the system, $\mathcal{T}_{\text{cpu}}$ causes a violation of the reference temperature, $\mathcal{T}_{\text{ref}}$, then an error value is calculated, as explained earlier in Equation 6.14. This error value is added to the CPU load-current reference, $\mathcal{I}_{ref}$ if the error is negative. This action effectively brings down the load-current into feasible lower value that does not cause a $\mathcal{T}_{\text{ref}}$ violation. After a couple of control cycles, the system become stable at a point that does not violate any constraints.

The simulation methodology used in this experiment is the same as the previous. During simulation, the following task parameters are used:

- Each sporadic task $\tau_j^{\mathcal{C},i} = (e_j^{\mathcal{C},i}, d_j^{\mathcal{C},i}, p_j^{\mathcal{C},i})$ has a period $p_j^{\mathcal{C},i}$ uniformly drawn from the interval $[5, 15]$. The execution time requirement $e_j^{\mathcal{C},i}$ set to the task utilization times $p_j^{\mathcal{C},i}$, where task utilization is calculated using the UUnifast algorithm [9]. For each task, $d_j^{\mathcal{C},i}$ equals $p_j^{\mathcal{C},i}$. The tasks are scheduled by EDF.

- The total number of tasks is eight; each task $\tau_j^{\mathcal{C}}$ has three different real-time performance modes where $\tau_j^{\mathcal{C},2} = (e_j^{\mathcal{C},2}, d_j^{\mathcal{C},2}, p_j^{\mathcal{C},2})$; $\tau_j^{\mathcal{C},1} = (.2e_j^{\mathcal{C},1}, d_j^{\mathcal{C},1}, p_j^{\mathcal{C},1})$; and $\tau_j^{\mathcal{C},0}$ means that task is not selected. From set of all possible combinations of tasks, we have selected fifteen modes with utilizations ranging from zero to one. Each mode has a schedulable Θ^i value set by running the multi-modal schedulability analysis in [30].

We refer to the controller described in algorithm as Generalized System Resiliency (GSR). We do not compare the performance of our proposed method with other algorithms as we are unaware of any research that addresses combined load/thermal constraints for hard-real-time systems.

The load-current and control parameters, $\mathcal{G}_{I/T}$, $\mathcal{H}_{I/T}$ are all generated from the testbed using Matlab System Identification (SI) tools with the Predictive Error Method (PEM). The state feedback, $\mathcal{K}_{sfb}$, integrator values, $\mathcal{K}_{ifb}, \mathcal{K}_i$, Q, and R are derived for system optimality using Riccati Equation as shown in our previous work [46, 48]. These testbed parameters are also used as the simulation parameters for the controller. The values of these parameters are shown in a table in the appendix.

6.7.2 Results

Figure 6.4 shows the GSR behavior over different reference temperature ($\mathcal{T}_{\text{ref}}$) values generated from simulations. As $\mathcal{T}_{\text{ref}}$ is lowered, the $\mathcal{I}_{cpu}$ is also lowered accordingly. Furthermore, when the external temperature is increased, $\mathcal{I}_{cpu}$ reaches its full value, $1.66\ A$.

Figure 6.5 shows the mode variation (i.e., the resiliency function Λ) over external temperature, $\mathcal{T}_{\text{env}}$ and external current, $\mathcal{I}_{ref}$. Clearly when $\mathcal{I}_{ref}$ is increased, the mode is decreased to maintain fixed load-resistance value; a reverse trend is observable for $\mathcal{T}_{\text{ref}}$.

Figure 6.6 shows the mode variation over reference temperature, $\mathcal{T}_{\text{ref}}$ and CPU current, $\mathcal{I}_{cpu}$ while external temperature, $\mathcal{T}_{\text{env}}$ is fixed at $19^\circ\ C$. This figure gives the system designer an indication as how modes will degrade with changing environmental conditions. Clearly, as either $\mathcal{T}_{\text{ref}}$ or $\mathcal{I}_{cpu}$ is increased, a larger Θ can be supported (i.e., the system is less constrained with respect to the power) and a higher mode can be supported. Finally, Figure 6.7 shows the load resiliency comparison of the system, load-current variation of the load, I_{ext} with respect to the resource capacity (mode) at $\mathcal{T}_{\text{ref}} = 45^\circ\ C$ and $\mathcal{T}_{\text{env}} = 19^\circ\ C$.

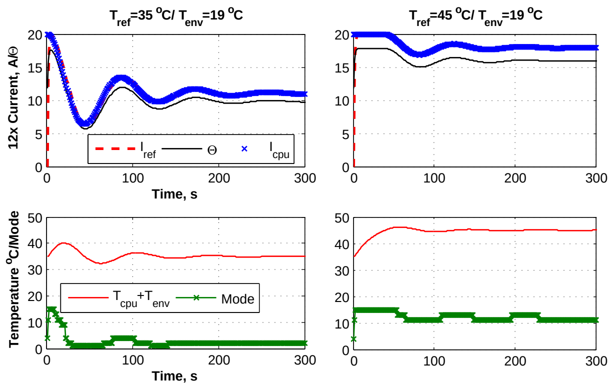


Figure 6.4: The effect of the reference temperature, $\mathcal{T}_{\text{ref}}$, on CPU current $\mathcal{I}_{cpu}$.

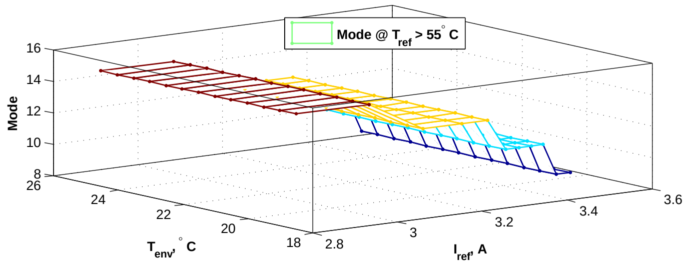


Figure 6.5: The mode variation over external temperature,$\mathcal{T}_{\text{env}}$ and reference current, I_{ref} .

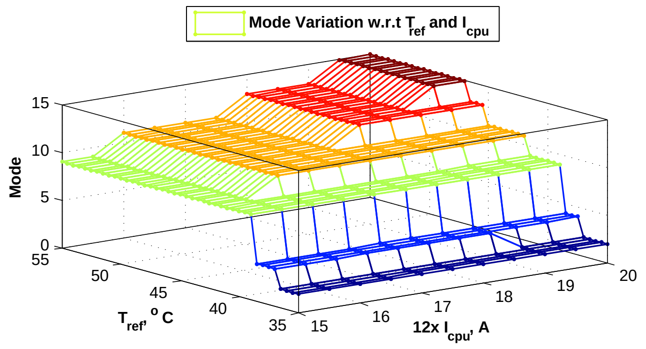


Figure 6.6: The mode variation over reference temperature,$\mathcal{T}_{\text{ref}}$ and CPU current, I_{cpu} .

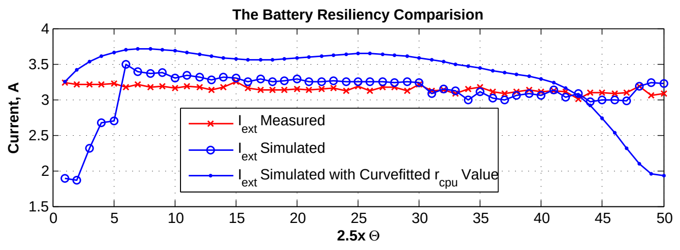


Figure 6.7: The calculated and measured external current comparison.

The external current I_{ext} comparison on calculated and testbed measured values when $\mathcal{T}_{\text{ref}} = 45^{\circ}\ C$ and $\mathcal{T}_{\text{env}} = 19^{\circ}\ C$.

In this graph, x-axis shows the 2.5 times scaled resource capacity, Θ. We plot the $\mathcal{I}_{ext}$ measured load testbed-values versus the load resiliency calculated from Equation 6.17. In this experiment, we calculate the r_{cpu} in discrete points and later use these values to calculate a function (using curve fitting techniques) of r_{cpu} in terms of resource capacity, Θ as shown in the appendix. The figure shows that the calculated load resiliency is a close match with the observed values from the testbed; the differences at high and low values of Θ are likely due to modeling inaccuracies.

6.8 Conclusions

This chapter addresses the problem of obtaining performance guarantees of multi-constrained real-time systems in an unpredictable external conditions. Our control-theoretic generalized system-resiliency (GSR) framework provides the system designer with a general methodology for obtaining predictable and stable hard-real-time control systems. In particular, the GSR framework is able to derive strong guarantees on the physical conditions under which a hard-real-time for any real-time performance mode may operate. Our framework provides the designer a means to verify the real-time system resiliency of a system before it is put into operation–as previous approaches which have no formal guarantee on the system resiliency and can only be verified through extensive experimentation. Furthermore, we have validated our GSR framework and its techniques on a hardware testbed with current as the primary control objective and temperature as a secondary constraint.

In future work, we plan to extend our case study to multiple secondary constraints. Furthermore, we will focus on the generation of automated design tools for the GSR framework. In such tools, the system designer might need to only express the basic physical properties/capabilities of the system and a high-level specification of the controller objectives/constraints; the tools would ideally then automatically derive the appropriate control models, system parameter, and resiliency calculation. We also hope to validate on other "resiliency" non-power-related system properties (e.g., noise level of a device in a hospital setting).

CHAPTER 7: CONCLUSION AND FUTURE WORK

In this thesis, we introduce a new metric called *system-resiliency* which characterizes the maximum external system stresses that any hard-real-time performance mode can withstand by given constraints. This framework addresses resiliency determination for real-time systems with physical and hardware limitations. Our method advocates the system designer about the feasible trade offs between different system resources for a multi parametric resiliency. The system-resiliency concept we introduced closely resembles the stress test in materials engineering. Thus, our design framework and analysis may be classified as a *system stress analysis* for real-time systems.

As a proof of concept, we introduce a new metric called thermal-resiliency, a subset of system resiliency which characterizes the maximum external thermal stress that any hard-real-time performance mode can withstand. We show how to solve some of the issues and challenges of designing predictable real-time systems that guarantee hard deadlines even under transitions between modes in an unpredictable thermal environment where environmental temperature may dynamically change using our new metric. In our framework, the system designer specifies a set of hard-real-time performance modes under which the system may operate automatically adjusts the real-time performance mode based on the external thermal stress. We extend the derivation of thermal resiliency to multicore systems and determines the limitations of external thermal stress that any hard-real-time performance mode can withstand. Also, we show how asymmetric processing resource allocation upon a multicore processor still maintain thermal constraints.

We investigate the peak temperature elevation of a hard-real-time system due to various task systems on uniprocessor systems. The system designer can use these temperature elevation information upon task allocation for various cores of multicore based system to avoid the peak temperature violation on a system that the periodic resource with proper capacity allocation on each core. Furthermore, we contribute to develop an efficient schedulability analysis framework for multi-modal real-time systems that are scheduled by Fixed Priority algorithm. We investigate the potential utility of parallelization for meeting real-time constraints and minimizing energy by considering the malleable Gang scheduling of implicit-deadline sporadic tasks upon multiprocessors. We use our real-time thermal-aware testbed to verify all these theoretical results upon the testbed runs.

7.1 Future Works Beyond the Thesis

Real-time system design upon multiple unpredictable physical and hardware constraints is a well-recognized research area. There are many growing number of career opportunities in cyber-physical systems design that need the broader understanding on real-time systems. Our research involves on thermal-aware and power-aware real-time system design. Towards our main research, we investigated many less-documented and hidden details of the processor architecture. Furthermore, we developed several system software modules and could gather many invaluable system development experiences. Most importantly, our research results along with theoretical modeling experience give us the strength and the training to handle many industry challenges. Therefore, we hope to pursue a research-oriented career in a closely related area with great confidence.

In the future, I will address following unsolved questions.

External Constraints with Dynamic Priorities: The dynamic external conditions of a real-time systems might have different priorities on various occasions. Furthermore, a multicore system design–with a dynamic external constraints, criticality of the external conditions are dynamically variable–is an open problem. In my future research, I will propose a framework to analyzes and predicts best modes of operation upon multiple dynamic constraints where external constraints priorities are dynamically variable.

Dynamic Core Activation: The multicore processors are becoming the industry standard for the embedded and real-time system development. However, dark silicon issue prevents activating a larger, fix number of cores all the time. Therefore, the processor forces to activate variable number of cores conditionally, thus the dynamic task allocation to cores with possible dynamic task migration is essential. However, task migration is costly and dynamic task allocation has many associated issues in a real-time design. In my future research, I address this issue with real-time control-theoretic dynamic core activation framework.

Robust Design: My proposed system-resiliency framework provides the predication and quantification of system degradation upon unpredictable external conditions. At this time, it does not safeguard the system against modeling and uncertainty errors. Therefore, I will broaden my framework to cover this aspect: the robust system design issue. Furthermore, in the future, the framework will analyze the system operations under noise and environmental disturbances as well. Finally, I will extend my research on learning-based control-theoretic framework to automate system design process.

Efficient Thermal and Power Plan: Multicore processors are increasing in cores per processor and are also becoming increasingly power hungry. On the other hand, they face the dark and dim silicon issues and majority simultaneous core activation is impossible. Furthermore, each activated-core acts as a thermal dissipating agent. Therefore, the active core floor plan and number of active cores at a given time is important to design an energy and thermal efficient real-time system. In the future, I will enhance my design framework to address these multicore design issues.

7.1.1 Multi-Mode Physiological Control-Systems

If opportunity permits, we would like to pursue research on the following areas: the idea is essentially design the tools to identify the multi-mode behavior of a physiological system of a human with a medical conditions under different constraints.

Often time, scientists try to understand physical systems through mathematical models. In the past, a little or no research exists in the area of physiological control systems. However, in recent years, scientist find many important research areas in physiology that can be govern/control by control systems. The human body is essentially a real-time system with larger time constants. Furthermore, it's time constants, the response times (for various activities, medicines etc) are varying; therefore, the human body dynamics can not be easily modeled unlike most of the physically-made systems that we have studied so far. On the other hand, the human body can be considered as a system–plant–with many input and output parameters. Intuition suggests, the human physiology can be mathematically modeled by taking all these factors into an account. In such research, first, the human body parameters variation should be modeled; the variable time constants should be identified. Although these objectives are not trivial, after formal design and analysis process (that may take many years and might need substantial amount of data for model verification), we should be able to answer a question such as follows: what would be the minimum respiratory rate to maintain the 95% Oxygen Saturation of a particular person (with X_a medical condition, say), thus what amount of exercise, he should not exceed? or, when a particular person is on X_b medication, what should be the maximum amount of sugar intake that he can tolerate without unconscious? or when a particular person is on $X_i,\ i \in \{1 \ldots n\}$ medications (medical-resiliency), what should be the maximum amount of sugar, exercise, alcohol, or $Y_j,\ j \in \{1 \ldots m\}$ medicines (external constraints) intake that he can tolerate at his marginal-unconscious, partially-active, or wide-awake states (different models)?

LIST OF PUBLICATIONS

JOURNAL

Published

(a) Pradeep Hettiarachchi, Nathan Fisher, Masud Ahmed, Le Yi Wang, Shinan Wang, and Weisong Shi. *The Design and Analysis of Thermal-Resilient Hard-Real-Time Systems*. ACM Transactions on Embedded Computing Systems, ACM TECS., 13(5s):146:1-146:25, July 2014.

(b) Masud Ahmed, Nathan W. Fisher, and Sangquan Wang, and Pradeep Hettiarachchi. *Minimizing Peak Temperature in Embedded Real-Time Systems via Thermal-Aware Periodic Resources* Sustainable Computing: Informatics and Systems . 1 (13), pp. 226-240, 2011.

CONFERENCE & WORKSHOP

Published

(a) Masud Ahmed, Pradeep Hettiarachchi, and Nathan Fisher. *Real-Time Multi-Modal Analysis for Fixed-Priority Scheduled Systems with Non-Preemptible Regions*. IEEE Real-Time and Embedded Technology and Application Symposium (RTAS 2015)

(b) Nathan Fisher, Masud Ahmed, and P. Hettiarachchi. *Open Problems in Multi-Modal Scheduling Theory for Thermal-Resilient Multicore Systems* 5th Real-Time Scheduling Open Problems Seminar (RTSOPS), Spain, 2014. (The most wanted problem award)

(c) Antonio Paolillo, Joel Goossens, Pradeep Hettiarachchi, and Nathan Fisher. *Power Minimization for Parallel Real-Time Systems with Malleable Jobs and Homogeneous Frequencies*. The 20th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2014).

(d) Pradeep Hettiarachchi, Nathan Fisher, Masud Ahmed, Le Yi Wang, Shinan Wang, and Weisong Shi. *The Design and Analysis of Thermal-Resilient Hard-Real-Time Systems*. Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium, Beijing, China. April 2012. (23%; 30 full papers accepted out of 127)

(e) Pradeep Hettiarachchi, Nathan Fisher, and Le Yi Wang. *Achieving Thermal-Resiliency for Multicore Hard-Real-Time Systems*. 25th Euromicro Conference on Real-Time Systems (ECRTS13) July 9-12, 2013 Paris, ECE, France.

Papers to be Submitted

1. Pradeep M. Hettiarachchi and Nathan Fisher. *A Generalized Design Framework for Adaptive Real-Time System Resiliency* IEEE Real-Time and Embedded Technology and Application Symposium (RTAS 2016).

APPENDIX A: DETAILS OF PRACTICAL SYSTEM IMPLEMENTATION ON TESTBEDS

A.1 The Temperature Calculations

In this section, we calculate the CPU temperature precisely. Therefore, we need to consider the CPU leakage current effects on temperature calculation as well. Using our thermal model with the leakage current effect, we calculate the CPU temperature, which is based on the solution of second order differential equation. From Equation (4.3), we get the first derivative of $\mathcal{T}_{\text{env}}(t)$ as follows,

$$\begin{aligned}\frac{d}{dt}\mathcal{T}_{\text{env}}(t) &= \frac{1}{k_T\sigma_1}\Big(\frac{d^2}{dt^2}\mathcal{T}_{\text{cpu}}(t) + \beta_1\frac{d}{dt}\mathcal{T}_{\text{cpu}}(t) - \sigma_1\frac{d}{dt}\mathcal{P}^{\text{d}}_{\text{cpu}}(t)\Big)\\ &= \frac{1}{k_T\sigma_1}\Big(\frac{d^2}{dt^2}\mathcal{T}_{\text{cpu}}(t) + \beta_1\frac{d}{dt}\mathcal{T}_{\text{cpu}}(t)\Big).\end{aligned} \tag{A.1}$$

In this analysis, we consider a system that can be described according to the model shown in the Section 4.3. Therefore, in the above Equation (A.1), we consider the system behavior for discrete time intervals and the input is considered to be constant in each sampling interval (the input value at the sampling time continue to hold for the rest of the period, until the next sampling time). This assumption is realistic because we implement our system as a discrete-time control system, in which the ZOH functionality means for holding the input value for inter-sampling times periods. Let us consider any such general time period where the input is held constant; therefore, for time instant t in this range, $\frac{d}{dt}\mathcal{P}^{\text{d}}_{\text{cpu}}(t)$ can be considered as zero. Thus, we can substitute Equation (4.2) and (A.1) to the Equation (4.5) to get the following,

$$\frac{d^2}{dt^2}\mathcal{T}_{\text{cpu}}(t) + \mathcal{V}\frac{d}{dt}\mathcal{T}_{\text{cpu}}(t) + \mathcal{B}\mathcal{T}_{\text{cpu}}(t) = \mathcal{F}_{act/inc/cont}, \tag{A.2}$$

where,

$$\begin{aligned}
\mathcal{V} &\stackrel{\text{def}}{=} (\beta_1 + \beta_2), \\
\mathcal{B} &\stackrel{\text{def}}{=} (\beta_1\beta_2 - k_T^2\sigma_1\sigma_2), \\
\mathcal{F}_{act/inc/cont} &\stackrel{\text{def}}{=} \Big(\beta_2\sigma_1 + \sigma_1\sigma_2 k_T\Big)(\mathcal{P}_{\text{act/inc/cont}} + k_C) + \sigma_1\sigma_2 k_T \mathcal{P}_{\text{env}}(t), \\
\mathcal{P}_{\text{act/inc/cont}} &= \begin{cases} \mathcal{P}_{\text{act}}, & \textit{active}; \\ \mathcal{P}_{\text{inc}}, & \textit{inactive}; \\ (\mathcal{P}_{\text{act}} - \mathcal{P}_{\text{inc}})\frac{\Theta}{\Pi} + \mathcal{P}_{\text{inc}}, & \textit{continuous}. \end{cases}
\end{aligned} \tag{A.3}$$

The Equation (A.2) is a second-order inhomogeneous equation and $\mathcal{F}$ is a constant ($\mathcal{P}^{\text{d}}_{\text{cpu}}(t)$ and $\mathcal{P}_{\text{env}}(t)$ are unchanged over two sampling periods). As we already discuss, the CPU can operate in two power modes. Depending on the operating mode of the system (active or inactive CPU operation), we can derive two different $\mathcal{F}$ values. Also, we assume, when the CPU power is represented in terms of the resource capacity, Θ, the corresponding $\mathcal{F}$ is denoted by $\mathcal{F}_{cont}$.[1]. Therefore, the complete solution for $\mathcal{T}_{\text{cpu}}$ and $\mathcal{T}_{\text{env}}$ over any continuous interval is given by,

$$\mathcal{T}^{\text{act/inc}}_{\text{cpu}}(t) = \mathcal{C}_{1_{act/inc}} e^{r_1 t} + \mathcal{C}_{2_{act/inc}} e^{r_2 t} + \mathcal{C}_{3_{act/inc}}, \tag{A.4}$$

where,

$$\begin{aligned}
r_{1/2} &= -\frac{1}{2}\Big(\mathcal{V} \mp \sqrt{\mathcal{V}^2 - 4\mathcal{B}}\Big), \text{ and} \\
\mathcal{C}_{3_{act/inc/cont}} &= \frac{\mathcal{F}_{act/inc/cont}}{\mathcal{B}}.
\end{aligned}$$

In the Equation (A.4), the r_1 and r_2 terms are negative because $\sqrt{\mathcal{V}^2 - 4\mathcal{B}}$ is positive and less than $\mathcal{V}$.

From Equation (4.5) and (A.4), we can find the $\mathcal{T}_{\text{env}}(t)$ for active and inactive CPU operations as follows,

[1] We justify the need for $\mathcal{F}_{cont}$ in the Tech report) under PWM Error Calculation

$$\begin{aligned}\mathcal{T}^{\text{act/inc}}_{\text{env}}(t) &= \frac{1}{k_T\sigma_1}\Big(\mathcal{C}_{1_{act/inc}}r_1e^{r_1t}+\mathcal{C}_{2_{act/inc}}r_2e^{r_2t}-\sigma_1(\mathcal{P}_{\text{act/inc}}+k_c)\Big)\\ &+ \frac{\beta_1}{k_T\sigma_1}\Big(\mathcal{C}_{1_{act/inc}}e^{r_1t}+\mathcal{C}_{2_{act/inc}}e^{r_2t}+\mathcal{C}_{3_{act/inc}}\Big). \qquad (A.5)\end{aligned}$$

We consider the system operates in interleaved active and inactive power modes over given interval size; the initial temperature of given period is the final temperature of the previous period. Given $\mathcal{T}_{\text{cpu}}(t_b)$ and $\mathcal{T}_{\text{env}}(t_b)$, fixed $\mathcal{P}^{\text{d}}_{\text{cpu}}$ and $\mathcal{P}_{\text{env}}$, we may obtain $\mathcal{C}_1$, $\mathcal{C}_2$ by solving Equations (A.4) and (A.5), where t_b is the initial time of the interval. Further, we derive the $\mathcal{C}$ as follows,

$$\begin{aligned}\mathcal{C}_{1_{act/inc/cont}}(t_b) &= \frac{1}{r_1-r_2}\Big(r_2\mathcal{C}_{3_{act/inc}}+\sigma_1\left(\mathcal{P}_{\text{act/inc/cont}}+k_C+k_T\mathcal{T}_{\text{env}}(t_b)\right)-\left(\beta_1+r_2\right)\mathcal{T}_{\text{cpu}}(t_b)\Big),\\ \mathcal{C}_{2_{act/inc/cont}}(t_b) &= \frac{1}{r_2-r_1}\Big(r_1\mathcal{C}_{3_{act/inc}}+\sigma_1\left(\mathcal{P}_{\text{act/inc/cont}}+k_C+k_T\mathcal{T}_{\text{env}}(t_b)\right)-\left(\beta_1+r_1\right)\mathcal{T}_{\text{cpu}}(t_b)\Big),\\ \mathcal{C}_{3_{act/inc/cont}}(t_b) &= \frac{\left(\beta_2\sigma_1+\sigma_1\sigma_2k_T\right)\left(\mathcal{P}_{\text{act/inc/cont}}+k_C\right)+\sigma_1\sigma_2k_T\mathcal{P}_{\text{env}}(t_b)}{\beta_1\beta_2-k_T^2\sigma_1\sigma_2}. \qquad (A.6)\end{aligned}$$

Note that, here we replace the initial power settings $\mathcal{P}^{\text{d}}_{\text{cpu}}(t)$ with $\mathcal{P}_{\text{act}}$. We use the Equation (A.4) and (A.5) to derive the temperature of the system at the end of each period. Therefore, consider the CPU temperature at any active period, $(n\Pi, n\Pi+\Theta]$ and adjacent inactive $(n\Pi+\Theta,(n+1)\Pi]$ period (details given in the tech report).

$$\mathcal{T}^{\text{act}}_{\text{cpu}}(n\Pi+\Theta) = \mathcal{T}^{\text{act}}_{\text{cpu}}(n\Pi)+\mathcal{C}_{1_{act}}(n\Pi)(e^{r_1\Theta}-1)+\mathcal{C}_{2_{act}}(n\Pi)(e^{r_2\Theta}-1) \qquad (A.7)$$

$$\begin{aligned}\mathcal{T}^{\text{inc}}_{\text{cpu}}((n+1)\Pi) &= \mathcal{T}^{\text{act}}_{\text{cpu}}(n\Pi)+\mathcal{C}_{1_{inc}}(n\Pi+\Theta)(e^{r_1(\Pi-\Theta)}-1)+\mathcal{C}_{2_{inc}}(n\Pi+\Theta)(e^{r_2(\Pi-\Theta)}-1)\\ &+ \mathcal{C}_{1_{act}}(n\Pi)(e^{r_1\Theta}-1)+\mathcal{C}_{2_{act}}(n\Pi)(e^{r_2\Theta}-1) \qquad (A.8)\end{aligned}$$

Therefore, we can derive the equation for the period $(n\Pi, (n+\varsigma)\Pi]$ is as follows.

$$\begin{aligned}
\mathcal{T}^{\text{inc}}_{\text{cpu}}((n+\varsigma)\Pi) &= \sum_{i=0}^{\varsigma-1} \mathcal{C}_{1_{inc}}((n+i)\Pi+\Theta)(e^{r_1(\Pi-\Theta)}-1) + \sum_{i=0}^{\varsigma-1} \mathcal{C}_{2_{inc}}((n+i)\Pi+\Theta)(e^{r_2(\Pi-\Theta)}-1) \\
&+ \mathcal{T}^{\text{act}}_{\text{cpu}}(n\Pi) + \sum_{i=0}^{\varsigma-1} \mathcal{C}_{1_{act}}((n+i)\Pi)(e^{r_1\Theta}-1) + \sum_{i=0}^{\varsigma-1} \mathcal{C}_{2_{act}}((n+i)\Pi)(e^{r_2\Theta}-1), \\
&= \mathcal{T}^{\text{act}}_{\text{cpu}}(n\Pi) \\
&+ \sum_{i=0}^{\varsigma-1}\sum_{j=1}^{2} \Big(\mathcal{C}_{(j)_{inc}}((n+i)\Pi+\Theta)(e^{r(j)(\Pi-\Theta)}-1) + \mathcal{C}_{(j)_{act}}((n+i)\Pi)(e^{r_{(j)}\Theta}-1) \Big).
\end{aligned} \tag{A.9}$$

Please note that the above Equation (A.9) is inductively defined, as the constants for the boundary conditions can be derived from the Equation (A.6) which are in terms of previous values of $\mathcal{T}_{\text{cpu}}$ and $\mathcal{T}_{\text{env}}$.

Now we use the same approach to derive the environment temperature $\mathcal{T}_{\text{env}}$ (the details given in the tech report) and derive the following for the period $(n\Pi, n\Pi+\varsigma]$ is as follows,

$$\begin{aligned}
\mathcal{T}^{\text{inc}}_{\text{env}}((n+\varsigma)\Pi) &= \mathcal{T}^{\text{act}}_{\text{env}}(n\Pi) + \sum_{i=0}^{\varsigma-1} \mathcal{C}_{1_{inc}}((n+i)\Pi+\Theta)(e^{r_1(\Pi-\Theta)}-1)\frac{r_1+\beta_1}{k_T\sigma_1} \\
& \sum_{i=0}^{\varsigma-1} \mathcal{C}_{2_{inc}}((n+i)\Pi+\Theta)(e^{r_2(\Pi-\Theta)}-1)\frac{r_2+\beta_1}{k_T\sigma_1} + \sum_{i=0}^{\varsigma-1} \mathcal{C}_{1_{act}}((n+i)\Pi)(e^{r_1\Theta}-1)\frac{r_1+\beta_1}{k_T\sigma_1} \\
&+ \sum_{i=0}^{\varsigma-1} \mathcal{C}_{2_{act}}((n+i)\Pi)(e^{r_2\Theta}-1)\frac{r_2+\beta_1}{k_T\sigma_1} \\
&= \mathcal{T}^{\text{act}}_{\text{env}}(n\Pi) + \sum_{i=0}^{\varsigma-1}\sum_{j=1}^{2} \Big(\mathcal{C}_{(j)_{inc}}((n+i)\Pi+\Theta)(e^{r_{(j)}(\Pi-\Theta)}-1)\frac{r_{(j)}+\beta_1}{k_T\sigma_1} \\
&+ \mathcal{C}_{(j)_{act}}((n+i)\Pi)(e^{r_{(j)}\Theta}-1)\frac{r_{(j)}+\beta_1}{k_T\sigma_1} \Big).
\end{aligned} \tag{A.10}$$

In the Equation (A.10), the constants for the boundary conditions can be derived from the Equation (A.6).

The CPU and environment temperature calculation equations (Equation (A.9) and (A.10)) gives a possible way to calculate the temperature states of the system, provided that we know a single boundary condition. Therefore, when we calculate the thermal resiliency and the PWM error for second-order thermal model, Equation (A.9) and (A.10) are used.

A.2 Calculation of State-Space Parameters Using Testbed Results

The state-space parameter generation process needs input and output data collected over sufficiently larger period. Unlike the testbench output ($\mathcal{T}_{\text{cpu}} + \mathcal{T}_{\text{env}}$ reading–measured using T-type thermocouple as explained in the Section 3.3.1), the testbed input, the equivalent CPU thermal input power cannot be measured directly. Instead, we measure the CPU input power, the closest measurable parameter. We assume the electrical power consumed by the CPU totally converts to thermal energy and measure the CPU input power and consider it as the equivalent thermal power[2].

We install two shunt resisters in series with the 4-pin ATX power connector and measure the voltage (and calculate the current drawn) drop across it using National Instrument data acquisition interface, NI 9205. Since The NI 9205 does not have a Linux USB driver, we create an application interface in a Windows computer to connect with the testbed using the Ethernet. The testbed measures the CPU and environment temperature and sends a sync signal to Windows computer with NI 9205 interface to record the ATX current readings. We calculate the the total power fed to the CPU, as the current drawn by CPU (through the NI measurements) and the voltage of the 4 wire ATX interface are known.

We run a random workload for a longer time period to generate thermal effects on the CPU and record input (power) and output (CPU temperature) data. We collect two sets of data from the testbed, one set to generate the model parameters and the other set validates them. We use standard tools provided by system-identification toolbox in Matlab to derive the state-space parameters (SSP) with the test data. [3] We use these SSP in the rest of the simulations and in the controller design.

We observe that when we do the SI process, the thermal output of the CPU is not sufficient enough to make a accurately measurable temperature difference in the environment. Therefore, for the parameter generation purpose, we consider the following: we use a first order CPU thermal-model, for the parameter generation, considering that the system environmental temperature stays stable and the the thermal model of the system is considered as a differential model. In other words, the leakage power of the testbed is a constant for a given temperature and, therefore when we consider the differential model (the difference between any steady point to the current point), the leakage power component need not to be considered

[2]This assumption is realistic because in the CPU (any electrical circuit) the desired objective is to operate its switches. However each gate (in switches) consume energy and generate heat. There is no any other energy transformation in an ordinary electrical circuit.

[3]We use Predictive Error Method (PEM) algorithm implementation in Matlab.

for closer operational points.

When we consider the environment temperature is nearly stable over a sufficiently larger time period, we may get a normalized thermal model of the CPU as follows,

$$\frac{d}{dt}\mathcal{T}_{\text{cpu}}(t) = \sigma_1\Big(k_T - \frac{1}{R^{\text{l}}_{\text{cpu}}} - \frac{1}{R^{\text{d}}_{\text{cpu}}}\Big)\mathcal{T}_{\text{cpu}}(t) + k_T\sigma_1\mathcal{T}_{\text{env}}(t) + \sigma_1\mathcal{P}^{\text{d}}_{\text{cpu}}(t) + \sigma_1 k_C. \tag{A.11}$$

Consider an another test-point (at t_E) during our SI process and assume the same environmental temperature, then we get, the following differential system model,

$$\frac{d}{dt}\bar{\mathcal{T}}_{\text{cpu}}(t) = A\bar{\mathcal{T}}_{\text{cpu}}(t) + B\bar{\mathcal{P}}^{\text{d}}_{\text{cpu}}(t), \tag{A.12}$$

where, $\bar{\mathcal{T}}_{\text{cpu}}(t) = \mathcal{T}_{\text{cpu}}(t) - \mathcal{T}_{\text{cpu}}(t_E)$, and $\bar{\mathcal{P}}^{\text{d}}_{\text{cpu}}(t) = \mathcal{P}^{\text{d}}_{\text{cpu}}(t) - \mathcal{P}^{\text{d}}_{\text{cpu}}(t_E)$. Therefore, the final system that we used in the controller design and the parameter generation may be considered as the above model.

In our parameter generation process, we use the discrete form of the above state-space Equation (A.12). As we shown earlier, the continuous-time state-space model can converted to discrete-time state-space model and the following discrete model is obtained,

$$\bar{\mathcal{T}}_{\text{cpu}}(k+1) = G\bar{\mathcal{T}}_{\text{cpu}}(k) + H\bar{\mathcal{P}}^{\text{d}}_{\text{cpu}}(k). \tag{A.13}$$

This parameter generation can be considered as linearization of our model at the operating points (at a particular environment temperature point). In our future work, we will generate linearized system parameters for a smooth operating regions and will implement a gain scheduled controller.

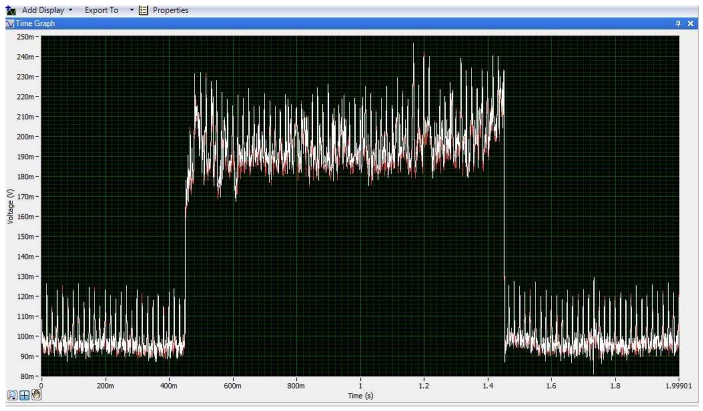


Figure A.1: The voltage variation of shunt resister for random workload.

The voltage variation of shunt resister for random workload in our testbed, at high and low frequency modulation values during the SI process shown by NI LabView (prominent high and low voltage values (square wave) correspond to high and low frequency modulation coefficient and the short voltage spikes, within the square wave correspond to the software workload variation. Also, note that the white and red color voltage traces correspond to two ATX channels).

APPENDIX B:
FURTHER DETAILS ON MULTICORE PROCESSOR MODEL

B.1 Multicore Thermal Model

To simplify the analysis, we consider a processor with 4 adjacent cores. We apply the Kirchhoff's circuit laws and get the following equations. We assume that the heat (power dissipation in the equivalent system) distribution from core i to core j as $\mathcal{I}_{ij}$. It is easy to see that $\mathcal{P}^{\mathcal{C}}_{\text{cpu}}(t), \mathcal{C} \in \{1 \dots m\}$ (in this case $m = 4$) the individual core power dissipation, distributes in the own core and to the adjacent cores as follows:

$$\begin{aligned}
\mathcal{P}^{1}_{\text{cpu}}(t) &= \mathcal{I}_{11} + \mathcal{I}_{12} + \mathcal{I}_{13} + \mathcal{I}_{14}, \\
\mathcal{P}^{2}_{\text{cpu}}(t) &= \mathcal{I}_{21} + \mathcal{I}_{22} + \mathcal{I}_{23} + \mathcal{I}_{24}, \\
\mathcal{P}^{3}_{\text{cpu}}(t) &= \mathcal{I}_{31} + \mathcal{I}_{32} + \mathcal{I}_{33} + \mathcal{I}_{34}, \\
\mathcal{P}^{4}_{\text{cpu}}(t) &= \mathcal{I}_{41} + \mathcal{I}_{42} + \mathcal{I}_{43} + \mathcal{I}_{44}.
\end{aligned} \tag{B.1}$$

Therefore, we get the thermal distribution for the 4 core system as,

$$
\begin{aligned}
\mathcal{I}_{11} &= C^{\mathrm{d}}_{\mathrm{cpu11}}(t)\frac{d}{dt}\mathcal{T}_{\mathrm{cpu11}}(t) + \frac{\mathcal{T}_{\mathrm{cpu11}}(t)}{R^{\mathrm{d}}_{\mathrm{cpu11}}} \\
&+ C^{\ell}_{\mathrm{cpu11}}(t)\frac{d}{dt}\mathcal{T}_{\mathrm{cpu11}}(t) + \frac{\mathcal{T}_{\mathrm{cpu11}}(t)}{R^{\ell}_{\mathrm{cpu11}}}, \\
\mathcal{I}_{22} &= C^{\mathrm{d}}_{\mathrm{cpu22}}(t)\frac{d}{dt}\mathcal{T}_{\mathrm{cpu22}}(t) + \frac{\mathcal{T}_{\mathrm{cpu22}}(t)}{R^{\mathrm{d}}_{\mathrm{cpu22}}} \\
&+ C^{\ell}_{\mathrm{cpu22}}(t)\frac{d}{dt}\mathcal{T}_{\mathrm{cpu22}}(t) + \frac{\mathcal{T}_{\mathrm{cpu22}}(t)}{R^{\ell}_{\mathrm{cpu22}}}, \\
\mathcal{I}_{33} &= C^{\mathrm{d}}_{\mathrm{cpu33}}(t)\frac{d}{dt}\mathcal{T}_{\mathrm{cpu33}}(t) + \frac{\mathcal{T}_{\mathrm{cpu33}}(t)}{R^{\mathrm{d}}_{\mathrm{cpu33}}} \\
&+ C^{\ell}_{\mathrm{cpu33}}(t)\frac{d}{dt}\mathcal{T}_{\mathrm{cpu33}}(t) + \frac{\mathcal{T}_{\mathrm{cpu33}}(t)}{R^{\ell}_{\mathrm{cpu33}}}, \\
\mathcal{I}_{44} &= C^{\mathrm{d}}_{\mathrm{cpu44}}(t)\frac{d}{dt}\mathcal{T}_{\mathrm{cpu44}}(t) + \frac{\mathcal{T}_{\mathrm{cpu44}}(t)}{R^{\mathrm{d}}_{\mathrm{cpu44}}} \\
&+ C^{\ell}_{\mathrm{cpu44}}(t)\frac{d}{dt}\mathcal{T}_{\mathrm{cpu44}}(t) + \frac{\mathcal{T}_{\mathrm{cpu44}}(t)}{R^{\ell}_{\mathrm{cpu44}}}.
\end{aligned}
\tag{B.2}
$$

Also, we can show that the thermal distribution between cores as,

$$
\begin{aligned}
\mathcal{I}_{21} &= C_{\mathrm{cpu12}}(t)\frac{d}{dt}(\mathcal{T}_{\mathrm{cpu22}}(t) - \mathcal{T}_{\mathrm{cpu11}}(t)) + \frac{(\mathcal{T}_{\mathrm{cpu22}}(t) - \mathcal{T}_{\mathrm{cpu11}}(t))}{R_{\mathrm{cpu12}}}, \\
\mathcal{I}_{31} &= C_{\mathrm{cpu13}}(t)\frac{d}{dt}(\mathcal{T}_{\mathrm{cpu33}}(t) - \mathcal{T}_{\mathrm{cpu11}}(t)) + \frac{(\mathcal{T}_{\mathrm{cpu33}}(t) - \mathcal{T}_{\mathrm{cpu11}}(t))}{R_{\mathrm{cpu13}}}, \\
\mathcal{I}_{41} &= C_{\mathrm{cpu14}}(t)\frac{d}{dt}(\mathcal{T}_{\mathrm{cpu44}}(t) - \mathcal{T}_{\mathrm{cpu11}}(t)) + \frac{(\mathcal{T}_{\mathrm{cpu44}}(t) - \mathcal{T}_{\mathrm{cpu11}}(t))}{R_{\mathrm{cpu14}}}, \\
\mathcal{I}_{32} &= C_{\mathrm{cpu23}}(t)\frac{d}{dt}(\mathcal{T}_{\mathrm{cpu33}}(t) - \mathcal{T}_{\mathrm{cpu22}}(t)) + \frac{(\mathcal{T}_{\mathrm{cpu33}}(t) - \mathcal{T}_{\mathrm{cpu22}}(t))}{R_{\mathrm{cpu23}}}, \\
\mathcal{I}_{42} &= C_{\mathrm{cpu24}}(t)\frac{d}{dt}(\mathcal{T}_{\mathrm{cpu44}}(t) - \mathcal{T}_{\mathrm{cpu22}}(t)) + \frac{(\mathcal{T}_{\mathrm{cpu44}}(t) - \mathcal{T}_{\mathrm{cpu22}}(t))}{R_{\mathrm{cpu24}}}, \\
\mathcal{I}_{43} &= C_{\mathrm{cpu34}}(t)\frac{d}{dt}(\mathcal{T}_{\mathrm{cpu44}}(t) - \mathcal{T}_{\mathrm{cpu33}}(t)) + \frac{(\mathcal{T}_{\mathrm{cpu44}}(t) - \mathcal{T}_{\mathrm{cpu33}}(t))}{R_{\mathrm{cpu34}}}.
\end{aligned}
\tag{B.3}
$$

Therefore, we get systems of first order differential equations,

$$\begin{aligned}
& (C_{\text{cpu12}} + C_{\text{cpu13}} + C_{\text{cpu14}} + C^{\text{d}}_{\text{cpu11}} + C^{\ell}_{\text{cpu11}})\frac{d}{dt}\mathcal{T}_{\text{cpu11}}(t) \\
- \; & C_{\text{cpu12}}\frac{d}{dt}\mathcal{T}_{\text{cpu22}}(t) - C_{\text{cpu13}}\frac{d}{dt}\mathcal{T}_{\text{cpu33}}(t) - C_{\text{cpu14}}\frac{d}{dt}\mathcal{T}_{\text{cpu44}}(t) \\
+ \; & (\frac{1}{R_{\text{cpu12}}} + \frac{1}{R_{\text{cpu13}}} + \frac{1}{R_{\text{cpu14}}} + \frac{1}{R^{\text{d}}_{\text{cpu11}}} + \frac{1}{R^{\ell}_{\text{cpu11}}})\mathcal{T}_{\text{cpu11}}(t) \\
- \; & \frac{\mathcal{T}_{\text{cpu22}}(t)}{R_{\text{cpu12}}} - \frac{\mathcal{T}_{\text{cpu33}}(t)}{R_{\text{cpu13}}} - \frac{\mathcal{T}_{\text{cpu44}}(t)}{R_{\text{cpu14}}} = \mathcal{P}^{1}_{\text{cpu}}(t),
\end{aligned} \tag{B.4}$$

$$\begin{aligned}
- \; & C_{\text{cpu12}}\frac{d}{dt}\mathcal{T}_{\text{cpu11}}(t) \\
+ \; & (C_{\text{cpu12}} + C_{\text{cpu23}} + C_{\text{cpu24}} + C^{\text{d}}_{\text{cpu22}} + C^{\ell}_{\text{cpu22}})\frac{d}{dt}\mathcal{T}_{\text{cpu22}}(t) \\
- \; & C_{\text{cpu23}}\frac{d}{dt}\mathcal{T}_{\text{cpu33}}(t) - C_{\text{cpu24}}\frac{d}{dt}\mathcal{T}_{\text{cpu44}}(t) - \frac{\mathcal{T}_{\text{cpu11}}(t)}{R_{\text{cpu12}}} \\
+ \; & (\frac{1}{R_{\text{cpu12}}} + \frac{1}{R_{\text{cpu23}}} + \frac{1}{R_{\text{cpu34}}} + \frac{1}{R^{\text{d}}_{\text{cpu22}}} + \frac{1}{R^{\ell}_{\text{cpu22}}})\mathcal{T}_{\text{cpu22}}(t) \\
- \; & \frac{\mathcal{T}_{\text{cpu33}}(t)}{R_{\text{cpu23}}} - \frac{\mathcal{T}_{\text{cpu44}}(t)}{R_{\text{cpu24}}} = \mathcal{P}^{2}_{\text{cpu}}(t),
\end{aligned} \tag{B.5}$$

$$\begin{aligned}
- \; & C_{\text{cpu13}}\frac{d}{dt}\mathcal{T}_{\text{cpu11}}(t) - C_{\text{cpu23}}\frac{d}{dt}\mathcal{T}_{\text{cpu22}} \\
+ \; & (C_{\text{cpu13}} + C_{\text{cpu23}} + C_{\text{cpu34}} + C^{\text{d}}_{\text{cpu33}} + C^{\ell}_{\text{cpu33}})\frac{d}{dt}\mathcal{T}_{\text{cpu33}}(t) \\
- \; & C_{\text{cpu34}}\frac{d}{dt}\mathcal{T}_{\text{cpu44}} - \frac{\mathcal{T}_{\text{cpu11}}(t)}{R_{\text{cpu13}}} - \frac{\mathcal{T}_{\text{cpu22}}(t)}{R_{\text{cpu23}}} \\
+ \; & (\frac{1}{R_{\text{cpu13}}} + \frac{1}{R_{\text{cpu23}}} + \frac{1}{R_{\text{cpu34}}} + \frac{1}{R^{\text{d}}_{\text{cpu33}}} + \frac{1}{R^{\ell}_{\text{cpu33}}})\mathcal{T}_{\text{cpu33}}(t) \\
- \; & \frac{\mathcal{T}_{\text{cpu44}}(t)}{R_{\text{cpu34}}} = \mathcal{P}^{3}_{\text{cpu}}(t),
\end{aligned} \tag{B.6}$$

and

$$
\begin{aligned}
- \quad & C_{\text{cpu14}}\frac{d}{dt}\mathcal{T}_{\text{cpu11}}(t) - C_{\text{cpu24}}\frac{d}{dt}\mathcal{T}_{\text{cpu22}}(t) - C_{\text{cpu34}}\frac{d}{dt}\mathcal{T}_{\text{cpu33}}(t) \\
& (C_{\text{cpu14}} + C_{\text{cpu23}} + C_{\text{cpu34}} + C^{\text{d}}_{\text{cpu44}} + C^{\ell}_{\text{cpu44}})\frac{d}{dt}\mathcal{T}_{\text{cpu44}}(t) \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(B.7)} \\
- \quad & \frac{\mathcal{T}_{\text{cpu11}}(t)}{R_{\text{cpu14}}} - \frac{\mathcal{T}_{\text{cpu22}}(t)}{R_{\text{cpu24}}} - \frac{\mathcal{T}_{\text{cpu33}}(t)}{R_{\text{cpu34}}} \\
+ \quad & (\frac{1}{R_{\text{cpu14}}} + \frac{1}{R_{\text{cpu24}}} + \frac{1}{R_{\text{cpu34}}} + \frac{1}{R^{\text{d}}_{\text{cpu44}}} + \frac{1}{R^{\ell}_{\text{cpu44}}})\mathcal{T}_{\text{cpu44}}(t) \\
= \quad & \mathcal{P}^{4}_{\text{cpu}}(t). \qquad\qquad \text{(B.8)}
\end{aligned}
$$

To simplify the analysis, we define the following,

$$
\begin{aligned}
\mathcal{X}_{11} &= C_{\text{cpu12}} + C_{\text{cpu13}} + C_{\text{cpu14}} + C^{\text{d}}_{\text{cpu11}} + C^{\ell}_{\text{cpu11}}, \\
\mathcal{X}_{12} &= \mathcal{X}_{21} = -C_{\text{cpu12}}, \\
\mathcal{X}_{13} &= \mathcal{X}_{31} = -C_{\text{cpu13}}, \\
\mathcal{X}_{14} &= \mathcal{X}_{41} = -C_{\text{cpu14}}, \\
\mathcal{Y}_{11} &= \frac{1}{R_{\text{cpu12}}} + \frac{1}{R_{\text{cpu13}}} + \frac{1}{R_{\text{cpu14}}} + \frac{1}{R^{\text{d}}_{\text{cpu11}}} + \frac{1}{R^{\ell}_{\text{cpu11}}} \\
\mathcal{Y}_{12} &= \mathcal{Y}_{21} = -\frac{1}{R_{\text{cpu12}}}, \\
\mathcal{Y}_{13} &= \mathcal{Y}_{31} = -\frac{1}{R_{\text{cpu13}}}, \\
\mathcal{Y}_{14} &= \mathcal{Y}_{41} = -\frac{1}{R_{\text{cpu14}}}, \\
\mathcal{X}_{22} &= (C_{\text{cpu12}} + C_{\text{cpu23}} + C_{\text{cpu24}} + C^{\text{d}}_{\text{cpu22}} + C^{\ell}_{\text{cpu22}}), \\
\mathcal{X}_{23} &= \mathcal{X}_{32} = -C_{\text{cpu23}}, \\
\mathcal{X}_{24} &= \mathcal{X}_{42} = -C_{\text{cpu24}}, \\
\mathcal{Y}_{22} &= \frac{1}{R_{\text{cpu12}}} + \frac{1}{R_{\text{cpu23}}} + \frac{1}{R_{\text{cpu34}}} + \frac{1}{R^{\text{d}}_{\text{cpu22}}} + \frac{1}{R^{\ell}_{\text{cpu22}}}, \\
\mathcal{Y}_{23} &= \mathcal{Y}_{32} = -\frac{1}{R_{\text{cpu23}}}, \\
\mathcal{Y}_{24} &= \mathcal{Y}_{42} = -\frac{1}{R_{\text{cpu24}}}, \\
\mathcal{X}_{33} &= C_{\text{cpu13}} + C_{\text{cpu23}} + C_{\text{cpu34}} + C^{\text{d}}_{\text{cpu33}} + C^{\ell}_{\text{cpu33}}, \\
\mathcal{X}_{34} &= \mathcal{X}_{43} = -C_{\text{cpu34}}, \\
\mathcal{Y}_{33} &= \frac{1}{R_{\text{cpu13}}} + \frac{1}{R_{\text{cpu23}}} + \frac{1}{R_{\text{cpu34}}} + \frac{1}{R^{\text{d}}_{\text{cpu33}}} + \frac{1}{R^{\ell}_{\text{cpu33}}}, \\
\mathcal{Y}_{34} &= -\frac{1}{R_{\text{cpu34}}}, \\
\mathcal{X}_{44} &= C_{\text{cpu14}} + C_{\text{cpu23}} + C_{\text{cpu34}} + C^{\text{d}}_{\text{cpu44}} + C^{\ell}_{\text{cpu44}}, \\
\mathcal{Y}_{44} &= \frac{1}{R_{\text{cpu14}}} + \frac{1}{R_{\text{cpu24}}} + \frac{1}{R_{\text{cpu34}}} + \frac{1}{R^{\text{d}}_{\text{cpu44}}} + \frac{1}{R^{\ell}_{\text{cpu44}}},
\end{aligned}
\tag{B.9}
$$

and

$$\begin{aligned}
\mathcal{X} &= (\mathcal{X}_{ij})_{4\times 4},\\
\mathcal{Y} &= (\mathcal{Y}_{ij})_{4\times 4},\\
\mathcal{A} &= (\mathcal{A}_{ij})_{4\times 4},\\
\mathcal{B} &= (\mathcal{B}_{ij})_{4\times 4},\\
\mathcal{A} &= -\mathcal{X}^{-1}\mathcal{Y},\\
\mathcal{B} &= \mathcal{X}^{-1}.
\end{aligned} \tag{B.10}$$

Therefore, we can simplify the system as,

$$\begin{aligned}
&\mathcal{X}\dot{\mathcal{T}}_{\text{cpu}}(t) + \mathcal{Y}\mathcal{T}_{\text{cpu}}(t) = \mathcal{P}_{\text{cpu}}(t),\\
\Rightarrow\quad &\dot{\mathcal{T}}_{\text{cpu}}(t) = \mathcal{A}\mathcal{T}_{\text{cpu}}(t) + \mathcal{B}\mathcal{P}_{\text{cpu}}(t),
\end{aligned} \tag{B.11}$$

where, $\dot{\mathcal{T}}_{\text{cpu}}(t) = \begin{bmatrix} \dot{\mathcal{T}}_{\text{cpu11}}(t) \\ \dot{\mathcal{T}}_{\text{cpu22}}(t) \\ \dot{\mathcal{T}}_{\text{cpu33}}(t) \\ \dot{\mathcal{T}}_{\text{cpu44}}(t) \end{bmatrix}$, $\mathcal{T}_{\text{cpu}}(t) = \begin{bmatrix} \mathcal{T}_{\text{cpu11}}(t) \\ \mathcal{T}_{\text{cpu22}}(t) \\ \mathcal{T}_{\text{cpu33}}(t) \\ \mathcal{T}_{\text{cpu44}}(t) \end{bmatrix}$, and $\mathcal{P}_{\text{cpu}}(t) = \begin{bmatrix} \mathcal{P}^1_{\text{cpu}}(t) \\ \mathcal{P}^2_{\text{cpu}}(t) \\ \mathcal{P}^3_{\text{cpu}}(t) \\ \mathcal{P}^4_{\text{cpu}}(t) \end{bmatrix}$.

B.2 The Testbed Parameters

We run 20000 testbed intervals to generate IO data required for the SI process. During SI process, the a random $\Theta^{\mathcal{C}}$ value is generated, and each CPU was allowed to execute a workload for a duration specified by $\Theta^{\mathcal{C}}$ $\mathcal{C} \in \{1 \ldots 8\}$ We found the following parameter values:

$$
\begin{aligned}
G &= \\
&\begin{bmatrix}
0.984 & 0.035 & 0.019 & -0.019 & 0.018 & 0.235 & 0.204 & -0.545 \\
0.115 & 1.070 & -0.068 & -0.102 & 0.056 & 0.269 & -0.902 & 0.803 \\
0.065 & -0.166 & 0.842 & 0.179 & 0.050 & 0.313 & 0.276 & 0.960 \\
0.109 & 0.156 & -0.037 & 0.814 & 0.061 & 0.564 & -0.796 & 0.041 \\
-0.029 & 0.012 & 0.051 & -0.029 & 0.951 & -0.370 & -0.127 & 0.005 \\
-0.112 & 0.029 & 0.120 & -0.027 & -0.066 & 0.507 & 0.330 & -0.804 \\
0.063 & 0.129 & 0.014 & -0.160 & 0.006 & -0.120 & 0.026 & 0.426 \\
-0.034 & 0.089 & 0.068 & -0.086 & -0.012 & -0.016 & -0.030 & 0.321
\end{bmatrix} \\
H &= 10^{-3} \times \\
&\begin{bmatrix}
0.035 & 0.012 & 0.024 & 0.022 & 0.017 & 0.032 & 0.019 & 0.025 \\
-0.132 & 0.244 & -0.175 & 0.102 & -0.026 & 0.117 & -0.094 & 0.046 \\
0.097 & 0.269 & 0.019 & -0.267 & -0.023 & 0.194 & -0.062 & -0.251 \\
-0.124 & 0.245 & -0.181 & 0.180 & -0.002 & 0.133 & -0.083 & 0.120 \\
-0.045 & -0.150 & 0.003 & 0.059 & -0.002 & -0.119 & 0.022 & 0.059 \\
0.017 & -0.322 & 0.096 & 0.077 & 0.024 & -0.202 & 0.092 & 0.104 \\
-0.177 & 0.029 & -0.156 & 0.209 & -0.017 & -0.041 & -0.048 & 0.153 \\
-0.026 & -0.106 & 0.002 & 0.126 & 0.018 & -0.066 & 0.036 & 0.118
\end{bmatrix} \\
K_0 &= 10^{-3} \times \\
&\begin{bmatrix}
0.357 & 0.096 & 0.153 & 0.047 & 0.006 & 0.430 & 0.150 & -0.543 \\
0.328 & 0.317 & 0.160 & 0.172 & -0.074 & 0.218 & -0.386 & 0.652 \\
0.319 & -0.129 & 0.140 & -0.201 & 0.139 & -0.044 & 0.407 & -0.289 \\
0.316 & 0.294 & -0.172 & 0.005 & 0.079 & 0.134 & -0.237 & -0.205 \\
0.315 & 0.011 & 0.080 & 0.103 & 0.070 & 0.276 & 0.086 & -0.398 \\
0.381 & 0.466 & 0.136 & 0.085 & -0.039 & 0.379 & -0.353 & 0.279 \\
0.202 & 0.043 & 0.061 & -0.151 & 0.057 & 0.042 & 0.184 & -0.180 \\
0.244 & 0.462 & -0.202 & 0.052 & 0.047 & 0.361 & -0.354 & -0.463
\end{bmatrix}
\end{aligned}
\tag{B.12}
$$

Also, the Q and R are 8×8 identity matrices.

B.3 The $\mathcal{T}_{\text{cpu}}$ Temperature Calculation

In this section, we solve the Equation B.11 to calculate the temperature of the individual CPU cores. We assume that the system is stable and should have 4 real solutions for the Equation B.11. Taking the eigenvalues of the system as $\lambda_i, \quad i \in \{1 \ldots 4\}$, the general solution for the system of equations (considering the homogeneous system) can be found as follows,

$$\mathcal{T}_{\text{cpu}}(t)_c = c_1\mathcal{V}_1 e^{\lambda_1 t} + c_2\mathcal{V}_2 e^{\lambda_2 t} + c_3\mathcal{V}_3 e^{\lambda_3 t} + c_4\mathcal{V}_4 e^{\lambda_4 t}$$

$$, \tag{B.13}$$

where, $\mathcal{V}_i = \begin{bmatrix} v_{1,i} \\ v_{2,i} \\ v_{3,i} \\ v_{4,i} \end{bmatrix}$ $i \in \{1 \ldots 4\}$ are eigenvectors of the system.

Also the fundamental matrix is given by,

$$\phi(t) = \begin{bmatrix} v_{1,1}e^{\lambda_1 t} & v_{1,2}e^{\lambda_2 t} & v_{1,3}e^{\lambda_3 t} & v_{1,4}e^{\lambda_4 t} \\ v_{2,1}e^{\lambda_1 t} & v_{2,2}e^{\lambda_2 t} & v_{2,3}e^{\lambda_3 t} & v_{2,4}e^{\lambda_4 t} \\ v_{3,1}e^{\lambda_1 t} & v_{3,2}e^{\lambda_2 t} & v_{3,3}e^{\lambda_3 t} & v_{3,4}e^{\lambda_4 t} \\ v_{4,1}e^{\lambda_1 t} & v_{4,2}e^{\lambda_2 t} & v_{4,3}e^{\lambda_3 t} & v_{4,4}e^{\lambda_4 t} \end{bmatrix} \tag{B.14}$$

.

We can further simplify the solution as,

$$\begin{aligned} \mathcal{T}_{\text{cpu}}(t)_c &= \phi(t)\mathcal{C}, \\ \Rightarrow \mathcal{T}_{\text{cpu}}(t)_c &= \phi(t)\phi(t)^{-1}\mathcal{T}_{\text{cpu}}(0)_c. \end{aligned} \tag{B.15}$$

Also, the particular solution is given by,

$$\mathcal{T}_{\text{cpu}}(t)_p = \phi(t)\int \phi(t)^{-1}\mathcal{B}\mathcal{P}_{\text{cpu}}(t)dt. \tag{B.16}$$

Therefore, the general solution is,

$$\mathcal{T}_{\text{cpu}}(t) = \mathcal{T}_{\text{cpu}}(t)_p + \mathcal{T}_{\text{cpu}}(t)_c. \tag{B.17}$$

Assume that the controller calculates four capacities $\Theta_1, \Theta_2, \Theta_3$, such that $\Theta_4,\ \Theta_1 > \Theta_2 > \Theta_3 > \Theta_4$. Then, we can calculate the temperature as follows,

$$\begin{aligned}
\mathcal{T}_{\text{cpu}}(\Theta_4) &= \phi(t)\phi(t)^{-1}\mathcal{T}_{\text{cpu}}(0)_c + \phi(t)\int_0^{\Theta_4}\phi(t)^{-1}\mathcal{B}\mathcal{P}_{\text{cpu}}(t)_1 dt,\\
\mathcal{T}_{\text{cpu}}(\Theta_3) &= \phi(t)\phi(t)^{-1}\mathcal{T}_{\text{cpu}}(\Theta_4)_c + \phi(t)\int_0^{\Theta_3-\Theta_4}\phi(t)^{-1}\mathcal{B}\mathcal{P}_{\text{cpu}}(t)_2 dt,\\
\mathcal{T}_{\text{cpu}}(\Theta_2) &= \phi(t)\phi(t)^{-1}\mathcal{T}_{\text{cpu}}(\Theta_3)_c + \phi(t)\int_0^{\Theta_2-\Theta_3}\phi(t)^{-1}\mathcal{B}\mathcal{P}_{\text{cpu}}(t)_3 dt,\\
\mathcal{T}_{\text{cpu}}(\Theta_1) &= \phi(t)\phi(t)^{-1}\mathcal{T}_{\text{cpu}}(\Theta_2)_c + \phi(t)\int_0^{\Theta_1-\Theta_2}\phi(t)^{-1}\mathcal{B}\mathcal{P}_{\text{cpu}}(t)_4 dt,\\
\mathcal{T}_{\text{cpu}}(\Pi) &= \phi(t)\phi(t)^{-1}\mathcal{T}_{\text{cpu}}(\Theta_1)_c + \phi(t)\int_0^{\Pi-\Theta_1}\phi(t)^{-1}\mathcal{B}\mathcal{P}_{\text{cpu}}(t)_5 dt,
\end{aligned}$$

$$\begin{aligned}
\Rightarrow \mathcal{T}_{\text{cpu}}(\Pi)) &= \big(\phi(t)\phi(t)^{-1}\big)^5 \mathcal{T}_{\text{cpu}}(0)_c \\
&+ \big(\phi(t)\phi(t)^{-1}\big)^4 \phi(t) \int_0^{\Theta_4} \phi(t)^{-1} \mathcal{B}\mathcal{P}_{\text{cpu}}(t)_1 dt + \\
&+ \big(\phi(t)\phi(t)^{-1}\big)^3 \phi(t) \int_0^{\Theta_3-\Theta_4} \phi(t)^{-1} \mathcal{B}\mathcal{P}_{\text{cpu}}(t)_2 dt, \\
&+ \big(\phi(t)\phi(t)^{-1}\big)^2 \phi(t) \int_0^{\Theta_2-\Theta_3} \phi(t)^{-1} \mathcal{B}\mathcal{P}_{\text{cpu}}(t)_3 dt, \\
&+ \big(\phi(t)\phi(t)^{-1}\big)^1 \phi(t) \int_0^{\Theta_1-\Theta_2} \phi(t)^{-1} \mathcal{B}\mathcal{P}_{\text{cpu}}(t)_4 dt, \\
&+ \phi(t) \int_0^{\Pi-\Theta_1} \phi(t)^{-1} \mathcal{B}\mathcal{P}_{\text{cpu}}(t)_5 dt.
\end{aligned}$$

Therefore,

$$\begin{aligned}
\Rightarrow \mathcal{T}_{\text{cpu}}(n\Pi)) &= \big(\phi(t)\phi(t)^{-1}\big)^5 \mathcal{T}_{\text{cpu}}((n-1)\Pi)_c \\
&+ \kappa\Big(\big(\phi(t)\phi(t)^{-1}\big)^4 \phi(t) \int_0^{\Theta_4} \phi(t)^{-1} \mathcal{B}\mathcal{P}_{\text{cpu}}(t)_1 dt + \\
&+ \big(\phi(t)\phi(t)^{-1}\big)^3 \phi(t) \int_0^{\Theta_3-\Theta_4} \phi(t)^{-1} \mathcal{B}\mathcal{P}_{\text{cpu}}(t)_2 dt, \\
&+ \big(\phi(t)\phi(t)^{-1}\big)^2 \phi(t) \int_0^{\Theta_2-\Theta_3} \phi(t)^{-1} \mathcal{B}\mathcal{P}_{\text{cpu}}(t)_3 dt, \\
&+ \big(\phi(t)\phi(t)^{-1}\big)^1 \phi(t) \int_0^{\Theta_1-\Theta_2} \phi(t)^{-1} \mathcal{B}\mathcal{P}_{\text{cpu}}(t)_4 dt, \\
&+ \phi(t) \int_0^{\Pi-\Theta_1} \phi(t)^{-1} \mathcal{B}\mathcal{P}_{\text{cpu}}(t)_5 dt\Big),
\end{aligned}$$

where,

$$\begin{aligned}
\mathcal{P}_{\text{cpu}}(t)_1 dt &= \begin{bmatrix} \mathcal{P}_{\text{act}} \\ \mathcal{P}_{\text{act}} \\ \mathcal{P}_{\text{act}} \\ \mathcal{P}_{\text{act}} \end{bmatrix}, \\
\mathcal{P}_{\text{cpu}}(t)_2 dt &= \begin{bmatrix} \mathcal{P}_{\text{act}} \\ \mathcal{P}_{\text{act}} \\ \mathcal{P}_{\text{act}} \\ \mathcal{P}_{\text{inc}} \end{bmatrix}, \\
\mathcal{P}_{\text{cpu}}(t)_3 dt &= \begin{bmatrix} \mathcal{P}_{\text{act}} \\ \mathcal{P}_{\text{act}} \\ \mathcal{P}_{\text{inc}} \\ \mathcal{P}_{\text{inc}} \end{bmatrix}, \\
\mathcal{P}_{\text{cpu}}(t)_4 dt &= \begin{bmatrix} \mathcal{P}_{\text{act}} \\ \mathcal{P}_{\text{inc}} \\ \mathcal{P}_{\text{inc}} \\ \mathcal{P}_{\text{inc}} \end{bmatrix}, \\
\mathcal{P}_{\text{cpu}}(t)_5 dt &= \begin{bmatrix} \mathcal{P}_{\text{inc}} \\ \mathcal{P}_{\text{inc}} \\ \mathcal{P}_{\text{inc}} \\ \mathcal{P}_{\text{inc}} \end{bmatrix}.
\end{aligned} \tag{B.18}$$

B.4 GSR Details:Run Time of the System

For our case study that involved a battery-operated system, using the conservation of energy rules, we can calculate the time period t_{bat} that system may be stable as follows,

$$t_{bat} = \frac{\mathcal{E}_{bat}}{\frac{r_{cpu}\cdot\mathcal{W}_{cpu}}{r_{ext}} + \mathcal{W}_{cpu}}, \tag{B.19}$$

or with the charging source with $\mathcal{I}_{crg}$ rate,

$$t_{bat} = \frac{\mathcal{E}_{bat}}{\frac{r_{cpu}\cdot\mathcal{W}_{cpu}}{r_{ext}} + \mathcal{W}_{cpu} - \mathcal{V}_{bat}\cdot\mathcal{I}_{crg}}, \tag{B.20}$$

where, $\mathcal{E}_{bat}$ is the initial energy of the battery.

B.5 Temperature Calculation of the System

In this section, we show how to calculate the temperature of the system when the term, $\mathcal{S}_{error}$ of $\int \mathcal{K}_{ifb}(-\mathcal{C}\mathcal{X}_I + I_{ref}) + \min(0, \mathcal{S}_{error})dt$ is zero at the stability. This corresponds to a system with reasonably high thermal constraint, and the thermal constraint is not violated by current-tracking action. In this case, the Θ is not excess enough to reduce the $\mathcal{I}_{ref}$, thus $\Theta(t)$ becomes, $\Theta(t) = -\mathcal{K}_{sfb}\mathcal{X}_I(t) + \int \mathcal{K}_{ifb}(-\mathcal{C}\mathcal{X}_I + I_{ref})dt$.

Therefore, we can solve the current state-variable $\mathcal{X}_I(t)$ as follows,

$$\begin{aligned}
\dot{\mathcal{X}}_I(t) &= \mathcal{A}_I\mathcal{X}_I(t) + \mathcal{B}_I(-\mathcal{K}_{sfb}\mathcal{X}_I(t) \\
&+ \int \mathcal{K}_{ifb}(-\mathcal{C}\mathcal{X}_I + I_{ref})dt), \\
\Rightarrow \ddot{\mathcal{X}}_I(t) &- (\mathcal{A}_I - \mathcal{B}_I\mathcal{K}_{sfb})\dot{\mathcal{X}}_I(t) \\
&+ \mathcal{K}_{ifb}\mathcal{B}_I\mathcal{C}_I\mathcal{X}_I(t) = \mathcal{B}_I\mathcal{K}_{ifb}I_{ref}, \\
\Rightarrow \ddot{\mathcal{X}}_I(t) &+ \mathcal{V}_1\dot{\mathcal{X}}_I(t) + \mathcal{V}_2\mathcal{X}_I(t) = \mathcal{V}_3,
\end{aligned} \tag{B.21}$$

where, $\mathcal{V}_1 \stackrel{\text{def}}{=} -(\mathcal{A}_I - \mathcal{B}_I\mathcal{K}_{sfb})$, $\mathcal{V}_2 \stackrel{\text{def}}{=} \mathcal{K}_{ifb}\mathcal{B}_I\mathcal{C}_I$, and $\mathcal{V}_3 \stackrel{\text{def}}{=} \mathcal{B}_I\mathcal{K}_{ifb}I_{ref}$

This is a second order inhomogeneous differential equation. Therefore the solution is,

$$\mathcal{X}_I(t) = \mathcal{C}_1 e^{r_1 t} + \mathcal{C}_2 e^{r_2 t} + \mathcal{C}_3, \tag{B.22}$$

where, $r_{1/2} \overset{\text{def}}{=} \frac{-\mathcal{V}_1 \pm \sqrt{(\mathcal{V}_1^2 - 4\mathcal{V}_2)}}{2}$ and $\mathcal{C}_3 \overset{\text{def}}{=} \frac{\mathcal{V}_3}{\mathcal{V}_2}$. Furthermore, $\mathcal{C}_1$ and $\mathcal{C}_2$, and are constants. Therefore,

$$\begin{aligned}
\Theta(t) &= -\mathcal{K}_{sfb}\mathcal{X}_I(t) + \int \mathcal{K}_{ifb}(-\mathcal{C}_I\mathcal{X}_I(t) + I_{ref})dt, \\
&= -\mathcal{K}_{sfb}(\mathcal{C}_1 e^{r_1 t} + \mathcal{C}_2 e^{r_2 t} + \mathcal{C}_3) \\
&+ \int \mathcal{K}_{ifb}(-\mathcal{C}_I(\mathcal{C}_1 e^{r_1 t} + \mathcal{C}_2 e^{r_2 t} + \mathcal{C}_3) + I_{ref})dt, \\
&= -\mathcal{K}_{sfb}(\mathcal{C}_1 e^{r_1 t} + \mathcal{C}_2 e^{r_2 t} + \mathcal{C}_3) \\
&+ t\mathcal{I}_{ref}\mathcal{K}_{ifb} - \mathcal{C}_I\mathcal{K}_{ifb}(\frac{1}{r_1}\mathcal{C}_1 e^{r_1 t} + \frac{1}{r_2}\mathcal{C}_2 e^{r_2 t} + \mathcal{C}_3).
\end{aligned} \tag{B.23}$$

Now we solve the thermal point of the system for calculated control input $\Theta(t)$ as follows from Equation 6.10,

$$\begin{aligned}
\dot{\mathcal{X}}_T(t) &= \mathcal{A}_T\mathcal{X}_T(t) + \mathcal{B}_T(-\mathcal{K}_{sfb}(\mathcal{C}_1 e^{r_1 t} + \mathcal{C}_2 e^{r_2 t} + \mathcal{C}_3) \\
&+ t\mathcal{I}_{ref}\mathcal{K}_{ifb} - \mathcal{C}_I\mathcal{K}_{ifb}(\frac{1}{r_1}\mathcal{C}_1 e^{r_1 t} + \frac{1}{r_2}\mathcal{C}_2 e^{r_2 t} + \mathcal{C}_3)).
\end{aligned} \tag{B.24}$$

This can be solved for $\mathcal{X}_T$,

$$\begin{aligned}
\mathcal{X}_T(t) &= \frac{1}{e^{\int \mathcal{A}_T dt}}\Big(\int e^{\int \mathcal{A}_T dt}\mathcal{B}_T(-\mathcal{K}_{sfb}(\mathcal{C}_1 e^{r_1 t} + \mathcal{C}_2 e^{r_2 t} + \mathcal{C}_3) \\
&+ t\mathcal{I}_{ref}\mathcal{K}_{ifb} - \mathcal{C}_I\mathcal{K}_{ifb}(\frac{1}{r_1}\mathcal{C}_1 e^{r_1 t} + \frac{1}{r_2}\mathcal{C}_2 e^{r_2 t} + \mathcal{C}_3))\Big)dt.
\end{aligned} \tag{B.25}$$

B.6 Testbed Details

We develop a Linux native posix thread libraries (NTPL) based multi-threaded application. Our application consists of a scheduler simulator (SS) and a thread activator. To simplify the testbed implementation, our SS runs a Θ resource period and makes it idle for $\Pi - \Theta$ period in each processor. The real-time loop runs as a high-priority thread (the priority is higher than the threaded IRQ handlers). First, the scheduler simulator invokes the optimal controller. Then, the optimal controller reads the CPU energy consumption and calculate the instantaneous CPU load-current value, and calculates the optimal Θ for the next period. The calculated Θ, value is applied to SS to select and activate the corresponding Θ. During this the entire Θ period, the CPU core is set to the highest power level. During the $\Pi - \Theta$ period, the idle job (thread) is executed, and the each CPU core is set to the low power level. The controller measures the CPU temperature and the external temperature in each control cycle. In case the Θ value causes to violate the temperature constraint, $\mathcal{T}_{\text{ref}}$, the Θ value is dynamically adjusted in terms of CPU target load-current value.

The scheduler simulator emulates the schedule tick functionality of the Linux kernel in a higher level granularity. Similar to the Linux kernel scheduler tick, the scheduler simulator, with the help of thread activator, sleeps until it wakes up in the Θ boundaries. Our thread activator wakes up to schedule resource capacity, Θ in different cores simultaneously, raises the appropriate thread of the core which should have the priority, and goes back to the sleep. This process repeats for the Θ given by the optimal controller. Also, the scheduler simulator (with the help of a thread activator) in each core completes the above process in parallel. When control period is passed, the scheduler simulator invokes the optimal controller to calculate the Θ value again.

In this experiment, we use curve fitting techniques to derive power function and determine the following,

$$\begin{aligned} \mathcal{W}_{cpu}(\Theta) &= 30.81\sin(0.16055\Theta - 0.2251) \\ &+ 45.05\sin(0.26075\Theta + 2.404) \\ &+ 26.77\sin(0.295\Theta + 5.323). \end{aligned} \tag{B.26}$$

Table B.1: Testbed parameters for generalized system resiliency simulations

Parameter	Value
$\mathcal{G}_I$	$\begin{bmatrix} 0.034 & 0.063 & 0.022 & 0.040 \\ 0.210 & 0.799 & -0.352 & -0.135 \\ -0.081 & -0.228 & 0.541 & -0.296 \\ -0.041 & 0.101 & 0.057 & -0.284 \end{bmatrix}$
$\mathcal{G}_T$	$\begin{bmatrix} 0.979 & 0.051 & 0.008 & 0.010 \\ 0.199 & 0.232 & -0.767 & 0.027 \\ 0.016 & -0.092 & 0.433 & -0.042 \\ -0.009 & -0.007 & 0.180 & -0.987 \end{bmatrix}$
$\mathcal{H}_I^T$	$\begin{bmatrix} 0.007 & -0.002 & -0.000 & -0.001 \end{bmatrix}$
$\mathcal{H}_T^T$	$\begin{bmatrix} 0.000 & -0.0043 & -0.0006 & 0.0011 \end{bmatrix}$
$\mathcal{K}_{sfb}$	$\begin{bmatrix} 5.078 & 13.423 & -1.336 & 2.128 \end{bmatrix}$
$\mathcal{K}_{ifb}$	0.6021
γ_T	0.878

Furthermore, we derive CPU resistance as,

$$\begin{aligned} r_{cpu}(25\Theta) &= p_1\Theta^9 + p_2\Theta^8 + p_3\Theta^7 + p_4\Theta^6 + p_5\Theta^5 \\ &+ p_6\Theta^4 + p_7\Theta^3 + p_8\Theta^2 + p_9\Theta + p_{10}, \end{aligned} \tag{B.27}$$

where, $p_1 = 1.315x10^{-021}$, $p_2 = -2.07x10^{-018}$, $p_3 = 1.168x10^{-015}$, $p_4 = -2.811x10^{-013}$, $p_5 = 3.777x10^{-011}$, $p_6 = -7.016x10^{-009}$, $p_7 = -3.247x10^{-006}$, $p_8 = 0.003028$, $p_9 = -0.7726$, and $p_{10} = 81.55$.

Figure B.1 confirms that the behavior of our controller in this environment is stable, and with a minimum effort the controller achieves its goal. Also, Figure B.2 shows the CPU impedance variation over the resource capacity. It shows Θ is inversely proportional to the CPU impedance.

B.7 System-Resiliency

§**Multiple System Constraints.** We can seamlessly add more constraints to the system design, and no changes in the framework are needed to handle the additional constraints. As a summary, to calculate the system resiliency for multi-constrained system, the following steps needs to be followed:

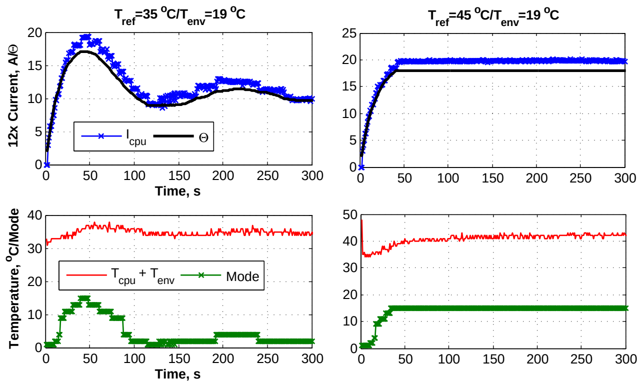


Figure B.1: The effect of the reference temperature, $\mathcal{T}_{\text{ref}}$ on CPU current $\mathcal{I}_{cpu}$ from testbed.

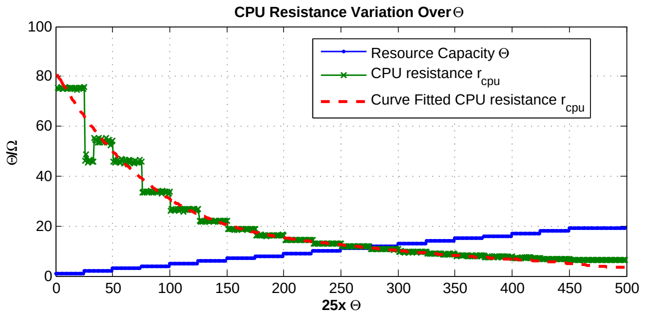


Figure B.2: The CPU impedance variation of the testbed.

The CPU impedance variation of the testbed over resource capacity, Θ for 3.33Ω fixed external load, r_{ext}.

1. Define the real-time performance modes.
2. State the modes as a function of the controllable variable(s), e.g., modes as a form of the CPU frequency/resource capacity.
3. Combine all the physical constraints in the CPU system to a system-of-equations.
4. Solve this system in terms of controllable variable(s)–in this case study, the CPU resource capacity .
5. Substitute different controllable-variable-values corresponding to different modes to above step to obtain the system resiliency.

B.8 Load Requirements

The external load and the CPU impedance are parallel loads. Also, the CPU impedance depends upon the resource capacity, Θ. Therefore, when system operates at various points (Θ values), the load impedance also should be modified by introducing additional resistances. For example, in a system design domain, suppose the designer wants to run the load with $\mathcal{I}_e$ load-current-requirement (specification) at Θ_1 resource capacity. Under this situation, a designer should introduce a parallel impedance compensation, r_a as follows.

$$\frac{1}{r_e} = \frac{1}{r_{ext}} + \frac{1}{r_a}, \tag{B.28}$$

and from Equation 6.17, we get,

$$r_e = \frac{1}{\mathcal{V}_{bat} \cdot \mathcal{I}_e} \cdot r_{cpu}(\Theta_1) \cdot \mathcal{W}_{cpu}(\Theta_1), \tag{B.29}$$

where, r_e is the modified load impedance of the load seen by the battery. If the designer can modify the load impedance to r_e, then load will operate at $\mathcal{I}_e$ current point while resource capacity is at Θ_1. Furthermore, from Equation B.28 and from Equation B.29, we can calculate the parallel impedance compensator, r_a,

$$r_a = r_{ext} \cdot \frac{r_{cpu}(\Theta_1) \cdot \mathcal{W}_{cpu}(\Theta_1)}{\mathcal{V}_{bat} \cdot \mathcal{I}_e \cdot r_{ext} - r_{cpu}(\Theta_1) \cdot \mathcal{W}_{cpu}(\Theta_1)}. \tag{B.30}$$

Therefore, the load impedance, r_{ext} should be changed to r_e, or in other words, additional impedance specified by r_a should be introduce to the load *internally* to run the $\mathcal{I}_e$ load at correct specification.

REFERENCES

[1] *Intel 64 and IA-32 Architectures Software Developer Manuals.* Intel Corp. http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html.

[2] Matlab system identification tools. Available at `http://www.mathworks.com/products/sysid/`.

[3] Measuring processor power (TDP vs. ACP). `http://www.intel.com/content/dam/doc/white-paper/measuring-processor-power-paper.pdf`.

[4] *Intel Pentium 4 processor in the 423-pin package thermal design guidelines,.* Intel Corp., 2000.

[5] Masud Ahmed, Nathan Fisher, Shengquan Wang, and Pradeep Hettiarachchi. Minimizing peak temperature in embedded real-time systems via thermal-aware periodic resources. *Sustainable Computing: Informatics and Systems*, 1(3):226 – 240, 2011.

[6] Nikhil Bansal and Kirk Pruhs. Speed scaling to manage temperature. In *Symposium on Theoretical Aspects of Computer Science*, 2005.

[7] Sanjoy Baruah and Joel Goossens. Scheduling real-time tasks: Algorithms and complexity. In Joseph Y.-T Leung, editor, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRC Press LLC, 2003.

[8] S. P. Bhattacharyya, H. Chapellat, and L. H. Keel. *Robust Control: The Parametric Approach*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1995.

[9] E. Bini and G. Buttazzo. Biasing effects in schedulability measures. In *Proceedings of the 16th Euromicro Conference on Real-Time Systems*, pages 196–203. IEEE Computer Society, 2004.

[10] Enrico Bini, Giorgio Buttazzo, and Giusepe Lipari. Speed modulation in energy-aware real-time systems. In *17th Euromicro Conference on. Real-Time Systems (ECRTS 05)*, 2005.

[11] Robert L Borrelli. *Differential Equations: A Modeling Perspective (2nd ed.).* John Wiley & Sons, Inc., Crosspoint Blvd., IN, USA, 2004.

[12] David Brooks and Margaret Martonosi. Dynamic thermal management for high-performance microprocessors. In *International Symposium on High-Performance Computer Architecture*, 2001.

[13] Thidapat Chantem, Robert P. Dick, and X. Sharon Hu. Temperature-aware scheduling and assignment for hard real-time applications on MPSoCs. In *Design, Automation and Test in Europe*, 2008.

[14] Thidapat Chantemand, Hu X. Sharon, and Robert P. Dick. Online work maximization under a peak temperature constraint. In *ISLPED '09: Proceedings of the 14th ACM/IEEE international symposium on Low power electronics and design*, pages 105–110, New York, NY, USA, 2009. ACM.

[15] Jian-Jia Chen, Chia-Mei Hung, and Tei-Wei Kuo. On the minimization of the instantaneous temperature for periodic real-time tasks. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, 2007.

[16] Jian-Jia Chen, Shengquan Wang, and Lothar Thiele. Proactive speed scheduling for frame-based real-time tasks under thermal constraints. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2009.

[17] Aviad Cohen, Finkelstein Finkelstein, Avi Mendelson, Ronny Ronen, and Dmitry Rudoy. On estimating optimal performance of cpu dynamic thermal management. *IEEE Comput. Archit. Lett.*, 2(1), 2003.

[18] Jim Cooling. *Software Engineering for Real-Time Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 2000.

[19] A.K. Coskun, T.S. Rosing, and K. Whisnant. Temperature aware task scheduling in mpsocs. pages 1 –6, apr. 2007.

[20] P. Dorato. A historical review of robust control. *Control Systems Magazine, IEEE*, 7(2):44–47, April 1987.

[21] Richard C. Dorf and Robert H. Bishop. *Modern Control Systems*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2000.

[22] John Comstock Doyle, Bruce A. Francis, and Allen R. Tannenbaum. *Feedback Control Theory*. Prentice Hall Professional Technical Reference, 1991.

[23] Arvind Easwaran. *Compositional Schedulability Analysis Supporting Associativity, Optimality, Dependency and Concurrency*. PhD thesis, Computer and Information Science, University of Pennsylvania, 2007.

[24] Arvind Easwaran, Madhukar Anand, and Insup Lee. Compositinonal analysis framework using EDP resource models. In *Proceedings of the IEEE Real-time Systems Symposium*, Tuscon, Arizona, December 2007. IEEE Computer Society.

[25] Hadi Esmaeilzadeh, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, pages 365–376, New York, NY, USA, 2011. ACM.

[26] A.P. Ferreira, D. Mosse, and J.C. Oh. Thermal faults modeling using a rc model with an application to web farms. In *Proceedings of the Euromicro Conference on Real-Time Systems*. IEEE Computer Society, July 2007.

[27] Re P. Ferreira and Daniel Moss. Thermal faults modeling using a rc model with an application to web farms. In *In Proceedings of RTS*, 2007.

[28] N. Fisher and F. Dewan. Approximate bandwidth allocation for compositional real-time systems. In *Real-Time Systems, 2009. ECRTS '09. 21st Euromicro Conference on*, pages 87–96, July.

[29] Nathan Fisher. An FPTAS for interface selection in the periodic resource model. In *Proceedings of 17th International Conference on Real-Time and Network Systems*, Paris, France, October 2009.

[30] Nathan Fisher and Masud Ahmed. Tractable real-time schedulability analysis for mode changes under temporal isolation. In *Proceedings of the 9th IEEE Symposium on Embedded Systems for Real-Time Multimedia (ESTImedia)*. IEEE Computer Society, October 2011.

[31] Nathan Fisher, Jian-Jia Chen, Shengquan Wang, and Lothar Thiele. Thermal-aware global real-time scheduling on multicore systems. In *Proceedings of the 15th IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE Computer Society Press, April 2009.

[32] Gene F. Franklin, Michael L. Workman, and Dave Powell. *Digital Control of Dynamic Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[33] Xing Fu and Xiaorui Wang. Utilization-controlled task consolidation for power optimization in multi-core real-time systems. In *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2011 IEEE 17th International Conference on*, volume 1, pages 73 –82, aug. 2011.

[34] Xing Fu, Xiaorui Wang, and E. Puster. Dynamic thermal and timeliness guarantees for distributed real-time embedded systems. In *Embedded and Real-Time Computing Systems and Applications, 2009. RTCSA '09. 15th IEEE International Conference on*, pages 403 –412, aug. 2009.

[35] Xing Fu, Xiaorui Wang, and Eric Puster. Dynamic thermal and timeliness guarantees for distributed real-time embedded systems. *Real-Time Computing Systems and Applications, International Workshop on*, 0:403–412, 2009.

[36] Xing Fu, Xiaorui Wang, and Eric Puster. Simultaneous thermal and timeliness guarantees in distributed real-time embedded systems. *Journal of Systems Architecture*, 2010. To Appear.

[37] Yong Fu, Nicholas Kottenstette, Yingming Chen, Chenyang Lu, Xenofon D. Koutsoukos, and Hongan Wang. Feedback thermal control for real-time system. In *Proceedings of the Real-Time and Embedded Technology and Applications Systems Symposium*, Stockholm, Sweden, April 2010. IEEE Computer Society Press.

[38] David Geer. Chip makers turn to multicore processors. *Computer*, 38(5):11–13, May 2005.

[39] Arkadii? Khai?movich Gelig and 1953 Churilov, Alexander N. *Stability and oscillations of nonlinear pulse-modulated systems / Arkadii Kh. Gelig, Alexander N. Churilov*. Boston : Birkhauser, 1998. Includes bibliographical references (p. [343]-359) and index.

[40] P. Gepner and M.F. Kowalik. Multi-core processors: New way to achieve high system performance. In *Parallel Computing in Electrical Engineering, 2006. PAR ELEC 2006. International Symposium on*, pages 9–13, Sept 2006.

[41] Pawel Gepner, DavidL. Fraser, and MichalF. Kowalik. Evaluating performance of new quad-core intelxeon5500 family processors for hpc. In Roman Wyrzykowski, Jack Dongarra, Konrad Karczewski, and Jerzy Wasniewski, editors, *Parallel Processing and Applied Mathematics*, volume 6067 of *Lecture Notes in Computer Science*, pages 1–10. Springer Berlin Heidelberg, 2010.

[42] Sourav Ghosh, Ragunathan Rajkumar, Jeffery Hansen, and John Lehoczky. Integrated qos-aware resource management and scheduling with multi-resource constraints. *Real-Time Syst.*, 33(1-3):7–46, July 2006.

[43] Sathish Gopalakrishnan, Marco Caccamo, Chi-Sheng Shih, Chang-Gun Lee, and Lui Sha. Finite-horizon scheduling of radar dwells with online template construction. In *RTSS*, 2004.

[44] Newport Innovative Product Hardware. Hot spot: How modern processors cope with heat emergencies. http://www.newport.com/store/genContent.aspx/Control-Theory-Terminology/178319/1033, September 2010.

[45] Pradeep M. Hettiarachchi, Nathan Fisher, Masud Ahmed, Le Yi Wang, Shinan Wang, and Weisong Shi. The design and analysis of thermally-resilient hard-real-time systems (extended version). Technical report, Wayne State University, 2011. Available at `http://www.cs.wayne.edu/~fishern/papers/thermal-control-rtas2012.pdf`.

[46] Pradeep M. Hettiarachchi, Nathan Fisher, Masud Ahmed, Le Yi Wang, Shinan Wang, and Weisong Shi. The design and analysis of thermal-resilient hard-real-time systems. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 67–76, 2012.

[47] Pradeep M. Hettiarachchi, Nathan Fisher, Masud Ahmed, Le Yi Wang, Shinan Wang, and Weisong Shi. A design and analysis framework for thermal-resilient hard real-time systems. *ACM Trans. Embed. Comput. Syst.*, 13(5s):146:1–146:25, July 2014.

[48] Pradeep M. Hettiarachchi, Nathan Fisher, and Le Yi Wang. Achieving thermal resiliency for multicore hard-real-time systems (extended version). Technical report, Wayne State University, 2013. Available at `http://www.cs.wayne.edu/~fishern/papers/ECRTS-2013-Thermal-TR.pdf`.

[49] W. L Hung, Y. Xie, N. ViJ'aykrishnan, M. Kandemir, and M.J. Irwin. Thermal-aware task allocation and scheduling for embedded systems. In *Design, Automation and Test in Europe, 2005. Proceedings*, pages 898–899 Vol. 2, March 2005.

[50] G. Kelly. Body temperature variability (part 1): a review of the history of body temperature and its variability due to site selection, biological rhythms, fitness, and aging. *Alternative Medicine Review*, 11(4):278–293, 2006.

[51] Hideaki Kikuchi, Rajiv K. Kalia, Aiichiro Nakano, Priya Vashishta, Fuyuki Shimojo, and Subhash Saini. Scalability of a low-cost multi-teraflop linux cluster for high-end classical atomistic and quantum mechanical simulations. In *IPDPS '03: Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, page 66.2, Washington, DC, USA, 2003. IEEE Computer Society.

[52] N.S. Kim, T. Austin, D. Baauw, T. Mudge, K. Flautner, J.S. Hu, M.J. Irwin, M. Kandemir, and V. Narayanan. Leakage current: Moore's law meets static power. *Computer*, 36(12):68–75, Dec 2003.

[53] Sohee Kim, P. Tathireddy, R.A. Normann, and F. Solzbacher. Thermal impact of an active 3-d microelectrode array implanted in the brain. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 15(4):493–501, December 2007.

[54] Joseph C. LaManna, Kimberly A. McCracken, Madhavi Patil, and Otto J. Prohaska. Stimulus-activated changes in brain tissue temperature in the anesthetized rat. *Metabolic Brain Disease*, 4(4):225–237, 1989.

[55] G. Lazzi. Thermal effects of bioimplants. *IEEE Engineering in Medicine and Biology Magazine*, 24(5):75–81, September - October 2005.

[56] Symos L. Vassilis Lewis, Frank L. *Optimal Control*. John Wiley and Sons, Inc., New York, MA, USA, 1995.

[57] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[58] Jane W. S. Liu. *Real-Time Systems*. Prentice-Hall, Inc., Upper Saddle River, New Jersey 07458, 2000.

[59] Yongpan Liu, Robert P. Dick, Li Shang, and Huazhong Yang. Accurate temperature-dependent integrated circuit leakage power estimation is easy. In *Proceedings of the conference on Design, automation and test in Europe*, pages 1526–1531, Nice, France, 2007.

[60] Advantech Co. Ltd. Advancech product hardware. http://www.advantech.com/products/, September 2013.

[61] Analog Devices Co. Ltd. Linux industrial input-output subsystems. http://wiki.analog.com/software/linux/docs/iio/iio, July 2012.

[62] Chenyang Lu, Xiaorui Wang, and X. Koutsoukos. Feedback utilization control in distributed real-time systems with end-to-end tasks. *Parallel and Distributed Systems, IEEE Transactions on*, 16(6):550 – 561, june 2005.

[63] M. Ma, S.H. Gunther, B. Greiner, N. Wolff, C. Deutschle, and T. Arabi. Enhanced thermal management for future processors. pages 201 – 204, jun. 2003.

[64] A. K. Mok. *Fundamental Design Problems of Distributed Systems for The Hard-Real-Time Environment*. PhD thesis, Laboratory for Computer Science, Massachusetts Institute of Technology, 1983. Available as Technical Report No. MIT/LCS/TR-297.

[65] Norman S. Nise. *Control Systems Engineering*. John Wiley & Sons, Inc., New York, NY, USA, 2000.

[66] Katsuhiko Ogata. *Discrete-time control systems (2nd ed.)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1995.

[67] Katsuhiko Ogata. *Modern Control Engineering*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.

[68] G. Quan and Y. Zhang. Leakage Aware Feasibility Analysis for Temperature-Constrained Hard Real-Time Periodic Tasks. In *Proceedings of the 2009 21st Euromicro Conference on Real-Time Systems-Volume 00*, pages 207–216. IEEE Computer Society, 2009.

[69] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek. A resource allocation model for qos management. In *Proceedings of the 18th IEEE Real-Time Systems Symposium*, RTSS '97, pages 298–, Washington, DC, USA, 1997. IEEE Computer Society.

[70] P.S. Ruggera, D.M. Witters, G. von Maltzahn, and H.I. Bassen. *In vitro* assessment of tissue heating near metallic medical implants by exposure to pulsed radio frequency diathermy. *Physics in Medicine and Biology*, 48(17):2919–2928, 2003.

[71] Euiseong Seo, Jinkyu Jeong, Seonyeong Park, and Joonwon Lee. Energy efficient scheduling of real-time tasks on multicore processors. *Parallel and Distributed Systems, IEEE Transactions on*, 19(11):1540 –1552, nov. 2008.

[72] Jerry Sergent and Al Krum. *Thermal Management Handbook for Electronic Assemblies*. McGraw-Hill Professional, 1998.

[73] Insik Shin and Insup Lee. Periodic resource model for compositional real-time guarantees. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 2–13. IEEE Computer Society, 2003.

[74] Insik Shin and Insup Lee. Compositional real-time scheduling framework with periodic model. *ACM Transactions on Embedded Computing Systems*, 7(3), April 2008.

[75] K.G. Shin and Xianzhong Cui. Computing time delay and its effects on real-time control systems. *Control Systems Technology, IEEE Transactions on*, 3(2):218 –224, jun. 1995.

[76] Joseph Sifakis. Modeling real-time systems - challenges and work directions. In *In Proceedings of the 1st International Workshop on Embedded Software (EMSOFT), Lecture Notes in Computer Science*, pages 373–389. Springer Verlag, 2001.

[77] K. Skadron, T. Abdelzaher, and M.R. Stan. Control-theoretic techniques and thermal-rc modeling for accurate and localized dynamic thermal management. pages 17 – 28, feb. 2002.

[78] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-aware microarchitecture. In *International Symposium on Computer Architecture*, 2003.

[79] Kevin Skadron. Hybrid architectural dynamic thermal management. In *DATE '04: Proceedings of the conference on Design, automation and test in Europe*, page 10010, Washington, DC, USA, 2004. IEEE Computer Society.

[80] Eduardo D. Sontag. Nonlinear regulation: The piecewise linear approach. *Proceedings of the IEEE Transactions on Automatic Control*, 26(2):346–358, Apr 1981.

[81] Selmo Tauber. Existence and uniqueness theorems for solutions of difference equations. *The American Mathematical Monthly*, 71(8):859–862, October 1964.

[82] Michael B. Taylor. Is dark silicon useful?: Harnessing the four horsemen of the coming dark silicon apocalypse. In *Proceedings of the 49th Annual Design Automation Conference*, DAC '12, pages 1131–1136, New York, NY, USA, 2012. ACM.

[83] N.F. Timmons and W.G. Scanlon. An adaptive energy efficient mac protocol for the medical body area network. In *1st International Conference on Wireless Communication, Vehicular Technology, Information Theory and Aerospace Electronic Systems Technology, 2009*, pages 587 –593, May 2009.

[84] V. Tiwari, D. Singh, S. Rajgopal, G. Mehta, R. Patel, and F. Baez. Reducing power in high-performance microprocessors. In *Design Automation Conference, 1998. Proceedings*, pages 732–737, June 1998.

[85] D. Vilcu. Real time scheduling and cpu power consumption in embedded systems. In *Automation, Quality and Testing, Robotics, 2008. AQTR 2008. IEEE International Conference on*, volume 1, pages 261–266, May 2008.

[86] Le Yi Wang, Pramod P. Khargonekar, and Ali Beydoun. Robust control of hybrid systems: Performance guided strategies. In *Hybrid Systems V*, pages 356–389, London, UK, 1999. Springer-Verlag.

[87] S. Wang and R. Bettati. Delay analysis in temperature-constrained hard real-time systems with general task arrivals. In *IEEE Real-Time Systems Symposium*, 2006.

[88] S. Wang and R. Bettati. Reactive speed control in temperature-constrained real-time systems. In *Euromicro Conference on Real-Time Systems*, 2006.

[89] S. Wang and R. Bettati. Reactive speed control in temperature-constrained real-time systems. *Real-Time Systems Journal*, 39(1-3):658–671, 2008.

[90] Yefu Wang, Kai Ma, and Xiaorui Wang. Temperature-constrained power control for chip multiprocessors with online model estimation. *SIGARCH Comput. Archit. News*, 37(3):314–324, 2009.

[91] Dan-Li Wen and Guang-Hong Yang. Quantized h∞ control for networked control systems with random delays. In *CCDC'09: Proceedings of the 21st annual international conference on Chinese Control and Decision Conference*, pages 643–647, Piscataway, NJ, USA, 2009. IEEE Press.

[92] F. Yao, A Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*, pages 374–382, Oct 1995.

[93] Jianguo Yao, Xue Liu, Zonghua Gu, Xiaorui Wang, and Jian Li. Online adaptive utilization control for real-time embedded multiprocessor systems. *Journal of Systems Architecture*, 56(9):463 – 473, 2010.

[94] Francesco Zanini, David Atienza, Luca Benini, and Giovanni De Micheli. Multicore Thermal Management with Model Predictive Control. In *Proceedings of the 19th European Conference on Circuit Theory and Design (ECCTD 2009)*, volume 1, pages 90–95, New York, 2009. IEEE Press.

ABSTRACT

A CONTROL-THEORETIC DESIGN AND ANALYSIS FRAMEWORK FOR RESILIENT HARD REAL-TIME SYSTEMS

by

PRADEEP M. HETTIARACHCHI

August 2015

Advisor: Dr. Nathan Fisher

Major: Computer Science

Degree: Doctor of Philosophy

We introduce a new design metric called *system-resiliency* which characterizes the maximum unpredictable external stresses that any hard-real-time performance mode can withstand. Our proposed system-resiliency framework addresses resiliency determination for real-time systems with physical and hardware limitations. Furthermore, our framework advises the system designer about the feasible trade-offs between external system resources for the system operating modes on a real-time system that operates in a multi-parametric resiliency environment.

Modern multi-modal real-time systems degrade the system's operational modes as a response to unpredictable external stimuli. During these mode transitions, real-time systems should demonstrate a reliable and graceful degradation of service. Many control-theoretic-based system design approaches exist. Although they permit real-time systems to operate under various physical constraints, none of them allows the system designer to predict the system-resiliency over multi-constrained operating environment. Our framework fills this gap; the proposed framework consists of two components: the design-phase and runtime control. With the design-phase analysis, the designer predicts the behavior of the real-time system for variable external conditions. Also, the runtime controller navigates the system to the best desired target using advanced control-theoretic techniques. Further, our framework addresses the system resiliency of both uniprocessor and multicore processor systems.

As a proof of concept, we first introduce a design metric called *thermal-resiliency*, which characterizes the maximum external thermal stress that any hard-real-time performance mode can withstand. We verify the thermal-resiliency for the external thermal stresses on a uniprocessor system through a physical testbed. We show how to solve some of the issues and challenges of designing predictable real-time systems that guarantee hard deadlines even under transitions between modes in an unpredictable thermal environment where environmental temperature may dynamically change using our new metric.

We extend the derivation of thermal-resiliency to multicore systems and determine the limitations of external thermal stress that any hard-real-time performance mode can withstand. Our control-theoretic framework allows the system designer to allocate asymmetric processing resources upon a multicore processor and still maintain thermal constraints.

In addition, we develop real-time-scheduling sub-components that are necessary to fully implement our framework; toward this goal, we investigate the potential utility of *parallelization* for meeting real-time constraints and minimizing energy. Under malleable gang scheduling of implicit-deadline sporadic tasks upon multiprocessors, we show the non-necessity of *dynamic* voltage/frequency regarding optimality of our scheduling problem. We adapt the canonical schedule for DVFS multiprocessor platforms and propose a polynomial-time optimal processor/frequency-selection algorithm.

Finally, we verify the correctness of our framework through multiple measurable physical and hardware constraints and complete our work on developing a generalized framework.

AUTOBIOGRAPHICAL STATEMENT

Pradeep Hettiararachchi received a B.S. degree in Electronics and Telecommunication Engineering from University of Moratuwa, Sri Lanka, and the M.S. degree in Computer Science from St. Cloud State University, MN, in 2000 and 2008 respectively. He worked with Sumathi Global Consolidated Group, Sri Lanka, from 2001 to 2005 as a systems Engineer, and as an IT Manger. Furthermore, he worked for Milltronics CNC Machines, MN, as a software developer from 2006 to 2008.

Pradeep started his PhD in Computer Science in 2009 at Wayne State University, MI, under the supervision of Dr. Nathan Fisher and did research on control-theoretic technique based constrained-aware real-time systems design, and he completed his PhD in 2015.