On the Design of Time-Predictable Low-Leakage Cache Memory for Real-Time Embedded Systems

MUTAZ AL-TARAWNEH

Computer Engineering Department

Mu'tah University

P.O. Box 7, Mu'tah 61710

JORDAN

Email: mutaz.altarawneh@mutah.edu.jo

Abstract: - This paper presents a multidisciplinary study that aims at designing a time-predictable low-leakage cache memory for real-time embedded systems. Both state-preserving and state-destroying leakage-saving mechanisms have been tested on a superscalar processor with two-level cache hierarchy. Full system simulation has been used to examine leakage-saving capability of each mechanism. In addition, a statistical approach has been proposed to study processor's time-predictability under potential leakage-saving techniques. Furthermore, the performance of real-time embedded systems in presence of leakage-saving techniques has been thoroughly analyzed using Matlab/Simulink-based models. Each possible design alternative has been evaluated in terms of four parameters that include: average power saving, degree of predictability (DoP), loss of schedulability (LoS) and performance of the underlying embedded system. Our results have shown that applying a state-preserving leakage-saving mechanism on either first-level data cache or last-level unified cache provides the most viable design option. The first alternative has achieved an average power saving of 32.61 %, a DoP of 93.05% and a LoS of 0% while the second alternative has achieved an average power saving of 50.21%, a DoP of 80.30% and a LoS of 13.68%. Moreover, neither of them has caused any disruption in the performance of the experimental embedded system models. Consequently, using a first-level data cache with a state-preserving leakage-saving mechanism represents the best feasible option for systems with very critical timing requirements while employing a state-preserving low-leakage last-level cache can be the suitable option for systems with soft timing requirements and stringent power constrains.

Key-Words: - cache memory, real-time, embedded, leakage power, time-predictability, performance.

1 Introduction

Real-time systems (RTS) have already become a ubiquitous computing platform in which programs (tasks) should maintain temporal correctness besides logical correctness [1]. In reality, most real-time systems are embedded systems where an embedded processor is employed to control a physical process [2]. Examples of some fields where embedded systems can be used include mobile devices, power plants control, automobile systems and cyber-physical systems. Nowadays, real-time applications have become more complex in terms of their memory footprint and processing requirements. Hence, their usage in real-time systems creates overly demanding tasks that require a very complex processor in order to cope with their functional and temporal requirements. To meet these requirements, embedded processors have employed many performance enhancement features such as cache memory. Cache memory leverages the principle of locality of reference and keeps the most heavily accessed instructions and data physically close to the processor [3]. Cache memory is much faster than the main memory and can run at the processor speed. Therefore, storing the required instructions and data items in the cache will reduce the number of clock cycles needed to fetch them which ,in turn, improves the overall performance of the processor. However, the potential performance improvement achieved by using cache memories comes at the expense of

reducing the time predictability required by real-time systems [4]. Time predictability is a major concern in the design space of real-time embedded systems such as automotive systems and nuclear power plants [5, 6]. Such systems are known by their strict timing requirements and high cost of failure and are usually referred to as safety-critical systems. In this kind of systems, missing the deadline of a particular task may lead to catastrophic outcomes and endanger human lives. Thus, software applications, in these systems, are implemented as hard real-time tasks with hard deadlines. A hard deadline is a deadline that should never be missed [1]. Moreover, real-time embedded systems often have constrained resources that should be carefully allocated in order to provide reasonable resource utilization and guarantee timing requirements of the running tasks [2]. In order to guarantee such stringent requirements, Worst-Case Execution Time (WCET) analysis is usually employed. Its goal is to obtain an upper-bound of the execution time of all tasks such that resource utilization and scheduling analysis can be studied under worst-case scenarios [7-9]. However, WCET depends not only on the running application but also on the micro-architecture of the host processor. Unfortunately, most of the performance enhancement hardware such as cache memories, instruction pipelines and branch predictors are designed to improve the average-case performance, not the worst-case performance. Hence, accurate WCET estimation on modern processors is very complicated task due to non-deterministic timing behavior imposed by the use of performance enhancement hardware. In particular, modern processors have heavily relied on cache memory to bridge the widening performance gap between the processor speed and the memory speed. Caches, unfortunately, can lead to unpredictable timing since whether or not a memory reference will hit in the cache is dependent on the program's dynamic behavior i.e. the interaction between program's working set and cache organization [10]. The difference between a hit and a miss can result in an order of magnitude difference in the execution time of a program. On the other hand, power consumption has recently become a major design concern in high-performance embedded processors [11, 12]. Although dynamic power -caused by switching activity- constitutes the majority of power dissipation in old processors implemented with large feature sizes, leakage power—caused by leakage current even when circuits are not switching— contributes to the majority of power consumption in modern processors implemented with nano-scale feature sizes [13,14]. Therefore, leakage power control has become an essential design constraint to maintain control of power dissipation in both general purpose and embedded processors. Moreover, as cache memories constitute a significant portion of processor's transistor budget, minimizing its leakage power consumption is of utmost importance to processor designers [15]. Hence, several leakage-control mechanisms have been proposed to reduce leakage power consumption especially in cache memories. These techniques can be classified as either state-preserving or state-destroying [16]. In state-destroying techniques, the cache line is completely turned which leads to a complete loss of the stored data. On the other hand, state-preserving techniques put the cache line into low-power mode such that its leakage power will be reduced to some extent while the stored data is still valid for future usage. This paper studies GatedVss [17] as a candidate for state-destroying techniques and Drowsy cache [18] as a state-preserving alternative. In the domain of real-time embedded systems, in which embedded processor are employed, the use of these leakage-power saving mechanisms introduces some degree of unpredictability in tasks' execution time. For example, in the GatedVss technique; a cache line will be switched off if it spends a sufficiently long time interval without being used. Hence, the data stored in this cache line will be completely lost and a future access to this line will cause a cache miss which need more clock cycles to be handled by accessing lower levels of the memory hierarchy; this means that the time required to handle a particular cache reference depends on the current power mode of that line; this fact leads to timing unpredictability especially in the contest of real-time systems. The fact that cache leakage-control mechanisms cause non-deterministic timing behavior of embedded processors motivates the study of WCET in presence of such mechanisms and the design of low-leakage cache hierarchy that attains the best tradeoff between leakage power consumption and time predictability in real-time embedded processors. This paper addresses leakage control in real-time systems and makes the following innovative contributions:

a. Analyzing the potential power savings of each candidate leakage-saving mechanism based on

full-system simulation and realistic processor configurations.

b. Proposing a statistical approach to estimate task's WCET on processors with low-leakage cache hierarchy. This makes this paper, to the best of our knowledge, the first research effort to tackle this issue.
c. Modeling and analyzing the impact of leakage-saving mechanisms on time predictability and overall performance of standalone and networked real-time embedded systems.
d. Identifying the suitable leakage-saving mechanism for each type of real-time embedded systems based on the stringency of its power budget and the criticality of its timing requirements.

The rest of this paper is organized as follows. Section 2 summarizes the related work, section 3 describes our methodology, results and analysis are presented in section 4, and section 5 summarizes and concludes.

2 Related Work

This section summarizes previous research work that is closely related to this paper in three main aspects: WCET estimation, leakage-control in real-time systems and real-time embedded control of physical systems.

2.1 WCET Estimation

WCET estimation is an important part in the analysis of real-time systems. Estimation techniques can be classified into five main categories [19, 20]: Static WCET Analysis [21-23], Measurement-based WCET Analysis [24-26], Hybrid-Measurement–based WCET Analysis [27, 28], Parametric-WCET Analysis [29] and Statistical WCET Analysis [30,31]. In static techniques, a static WCET is made based on the knowledge of the control-flow graph (CFG) of the program being analyzed and a model of the processor on which the program is going to execute. It is usually performed by the complier. The WCET is the duration of the longest path through the CFG. Static WCET estimation usually involves pessimistic assumptions regarding some variables such as the number of loop iteration or unrealistic assumptions about some hardware components such as assuming a perfect cache level i.e. a cache level with no misses or a pipelined processor with no data or control hazards. In other words, performance enhancement techniques employed in modern processors such as caching and pipelining introduces a high degree of difficulty in applying static estimation methods [32]. In measurement-based techniques, the program is executed, for sufficiently large number of times, on the target processor and the longest observed execution time will be used as a WCET. This procedure involves testing the program with different input combinations that are assumed to be representative of the whole input space of the program. However, none of the tested input patterns may excite the worst-case path of the program. To remedy this situation, the observed WCET is usually scaled with an appropriate safety factor; however, there is no systematic way for choosing such a factor. In other words, measurement-based techniques could underestimate the actual WCET of the program. On the other hand, a hybrid measurement-based technique combines static and measurement-based techniques in order to eliminate the potential overestimation and underestimation caused by them respectively. In this technique, the execution times of program segments, generated via instrumentation points, are collected and used in subsequent stages of WCET estimation. The rationale behind this technique is that the WCET of each program segment has been obtained by applying representative input patterns. However, failing to satisfy such assumption may compromise the final WCET estimate i.e. it may lead to overestimation or underestimation of the actual WCET. In parametric WCET estimation, the WCET estimate is expressed as a closed-form function in terms of parameters of the program, rather than just a single numerical value. A parametric WCET formula contains more information about the program being analyzed and can be applied in situations where some parameters are not known until runtime or to determine which program segments have more influence on the overall WCET. However, this technique can be applied for very small programs with small working sets but not for large programs with large instruction footprints and data-set size. Finally, statistical WCET analysis, to which our proposed technique belongs, applies statistical methods to obtain a WCET estimate with an extremely low and quantifiable probability of being exceeded (e.g. 10^{-10}). The majority of these techniques are based on Extreme Value Theory (EVT) [33]. In this type of analysis, execution time values obtained from end-to-end measurements are subjected to statistical analysis based on the techniques of EVT. The outcome of this analysis is a probability function of execution time from which WCET estimates, with pre-defined exceedance probability, can be obtained. Only this category will

be elaborated and compared to our proposed methodology. In [30], execution time measurements were fit to Gumbel Max distribution and WCET estimates were obtained using an excess distribution function. However, their proposed methodology incorrectly fits raw execution time measurements to the Gumbel Max distribution; Gumbel Max and other members of EVT family of distributions are intended to model random variables that are minimum or maximum of a large number of other random variables [20]. Moreover, they did not apply any Goodness-of-Fit (GOF) test to identify whether the estimated parameters of the Gumbel Max distribution can truly fit the measured execution times. On the other hand, the work presented in [31] has extended the study of [30] by predicting the likelihood that the WCET estimate made by EVT distributions will be exceeded. They have employed the method of block maxima [33] to fit execution time measurements to Gumbel Max distribution. In this method, the measured values will be divided into blocks, the maximum value in each block will be observed and then the observed maximum values will be fitted to the Gumbel Max distribution. In addition, they have performed a Goodness-of-Fit analysis to check whether Gumbel Max parameters actually fit the measured execution time values. Our work differs from the previous studies in two main aspects. First, the proposed WCET estimation technique depends on Generalized Extreme Value (GEV) distribution which provided more accuracy as compared to other distributions [34]. Second, the statistical analysis performed in this work was based on execution time measurements that take into account all possible interactions between the application's working set and the micro-architecture of the processor, unlike the previous studies which have overlooked this issue.

2.2 Leakage Reduction in Cache Memory

Several techniques have been proposed to reduce leakage power consumption of cache memories. These techniques can be either state-destroying or state-preserving [16]. In [17], a state-destroying, leakage-saving technique was proposed based on a Gated-V_{dd} circuit. In this technique, a high-threshold sleep transistor is used to disconnect a particular storage cell from V_{dd}. This technique can achieve drastic leakage savings since it breaks the connection with the power supply. However, it can lead to significant performance losses and dynamic power consumption due to sate losses [35]. In [18], a state-preserving leakage-saving mechanism known as Drowsy cache has been presented and evaluated. It achieves significant leakage savings by putting a cache line into low-power "Drowsy" mode. In this technique, the information stored in the cache line will be preserved by switching its V_{dd} to another power supply that is only 1.5 times the threshold voltage [16, 35]. In other words, Gated-V_{dd} techniques completely turns that cache line off while Drowsy cache puts a cache line into a low retention voltage level such that its contents are retained. Hence, state-preserving techniques produce much less penalty as compared to state-destroying ones while the latter provide much more leakage power savings. However, the primary focus of the aforementioned studies was leakage-power savings and the tradeoff achieved with the average-case performance of the processor. In other words, they did not study the impact of such techniques on the worst-case performance of the processor and time-predictability of real-time systems. On the other hand, [36] has presented a timing-aware leakage control mechanism suited for hard real-time systems. Their proposed methodology relies on using system slack. They have proposed a joint use of Gated-V_{dd} and Drowsy cache based on the overall utilization of the processor. Their proposed technique put cache lines into low-leakage mode such that leakage-power is saved while timing requirements are met. However, our work differs from theirs in a primary facet: they have assumed that the WCET of the program is already known and did not propose any mechanism to estimate the WCET of a program in presence of low-leakage caches.

2.3 Design of Real-Time Embedded Control Systems

Efficient implementation of a real-time control system needs a codesign of both computer and control systems [36]. In other words, the computer system must be designed such that all functional requirements are met and the controllers must be carefully designed taking into account the resource-constrains of embedded systems [37]. In embedded control systems, the control algorithm is implemented as a real-time task that executes concurrently with other tasks, including other control tasks. Moreover, embedded systems are usually resource-constrained in terms of their processing capability and memory capacity [2]. Therefore, the codesign of computer (scheduling) and control systems can be stated as follows [36]:

Given a group of physical processes to be controlled by a computer with limited resources, implement a

set of controllers and schedule them as real-time tasks such that the control performance and stability of each controlled process is maintained and optimized.

Previous research efforts [38-43] have studied the extent to which real-time scheduling can affect the performance and stability of controlled processes in both standalone and networked control systems. However, they have performed their analysis based on the fact that timing requirements of real-time tasks can be satisfied by using modern high-performance processors. Unfortunately, such high-performance processors are typically optimized for average-case performance rather than predictable worst-case performance required by real-time systems. In other words, they have overlooked the processor-induced unpredictability in their analysis which, consequently, can lead to unreliable results. This paper sheds light on the impact of processor-induced delays on control performance and focuses on schedulability analysis of real-time control tasks in presence of low-leakage cache hierarchy. In other words, it investigates the impact of processing delays induced by leakage-saving mechanisms on the schedulability of control tasks and, therefore, on the performance and stability of the controlled processes.

3. Methodology

The experimental procedure that has been followed to perform this research consists of three main steps: **a)** estimation of leakage-power savings using different leakage-saving techniques at different cache levels, **b)** time-predictability analysis and WCET estimation under different leakage-saving techniques, **c)** modelling of standalone and networked real-time embedded control systems and analysis of their timing behaviour and control performance in presence of leakage-saving techniques. The next sub-sections thoroughly explain each step.

3.1 Power Tradeoffs Analysis

This step aims at comparing Gated-Vss technique and Drowsy cache in terms of their potential power savings in a superscalar processor whose configuration is shown in table 1. All results were obtained using Hotleakage [44]. Hotleakage is a full-system simulator with an architectural model for sub-threshold and gate-leakage in caches and cache-like structures. This paper assumes a processor model that closely resembles an alpha 21264 [45]. In addition, leakage-saving mechanism is applied on exactly one cache level at a time. Power estimation was based on six randomly chosen benchmarks from SPEC2000 benchmark suite [46].

Table 1: Baseline Processor Configuration

Parameter	Value
Processor	
Functional Units	4 Integer ALU 1 Integer multiplier/divider 4 FP ALU 1 FP multiplier/divider
LSQ Size	8
RRU Size	8
Fetch Width	4 instruction / cycle
Decode Width	4 instruction / cycle
Issue Width	4 instruction / cycle
Commit Width	4 instruction / cycle
Fetch Queue Size	4 instruction
Clock Frequency	2800 MHz
Cache and Memory Hierarchy	
L1 Instruction Cache	Size: 32KB, 64 byte blocks Associativity: 2-way 1 cycle latency
L1 Data Cache	Size: 64KB , 64 byte blocks Associativity: 2-way 1 cycle latency Write policy: write back
Last Level(L2)	Size: 256 KB unified , 128 byte blocks Associaitivity: 8 6 cycle latency Write policy: write back
Main Memory	100 cycle latency
Brach Predictor	
Predictor	Combined , bimodal 2KB table Two-level 1KB table 8-bit history
BTB	512 entry, 4-way
Misprediction Penalty	3 Cycles

Table 2 shows the used benchmarks along with a short description for each one. The two leakage-saving mechanisms have been implemented such that a particular cache line is put into low-leakage mode if it spends a sufficiently long time interval without being accessed. We refer to this interval to as *Low-Power Interval (LPI)*.

Admittedly, both leakage-control mechanisms require extra hardware that adds more dynamic and leakage power to that consumed by the original processor hardware. Their power cost include: dynamic power due to extra hardware, leakage power resulting from extra hardware, dynamic power due to mode transitions and dynamic power Due to loss of state [44, 45]. Hence, this paper has measured the effectiveness of each leakage-saving technique taking into account both benefits and costs of each technique. For each technique, all benchmarks have been run, till completion, on the baseline configuration assuming different LPI values. The efficacy of each technique at each possible LPI has then been quantified in terms of its average power savings among all benchmarks.

Table 2: Benchmarks Description.

Benchmark	Category
ammp	Computational Chemistry
apsi	Meteorology: Pollutant Distribution
equake	Seismic Wave Propagation Simulation
bzip2	Compression
galgel	Computational Fluid Dynamics
mesa	3-D Graphics Library

3.2 Statistical WCET Estimation Using EVT

This part compares Gated-Vss and Drowsy cache in terms of time predictability; it estimates the WCET of real-time tasks in presence of these techniques and figures out which technique would provide a close predictability as compared to a baseline system with no leakage-control. WCET estimation using EVT consists of two main steps as shown in the following sub-sections.

3.2.1 Generation of Execution Time Population (ETP).

An important step in any statistical approach is to generate data samples from which statistical models can be obtained and evaluated. In this work, data samples represent execution time values obtained from end-to-end measurements on the target processor. In general, the total population of execution time values in a real-time system is extremely large and difficult to determine. Therefore, it is important to limit the population of interest into a representative dimension that can be treated in a tractable manner. This fact requires understanding the factors with strong influence on the execution time of a particular task and quantifying the extent to which a particular factor can affect the execution time of any run of the task under study. These factors are usually known as Source of Execution Time Variability (SETV) [47].Typically, the execution time of a program depend on its working set and the extent to which this working set can interact with the underlying processor micro-architecture. In this paper, a Placket and Burman (PB) design [48] has been used to obtain execution time samples that take into account the impact of each micro-architectural parameter on the execution time of a task. PB design has been employed since it requires only about N simulations to produce the desired level of information that takes into account the impact of N different parameters. However, the major critique against PB design is that it is unable to quantify the effects of all possible interactions between different parameters. Hence, it is possible that a significant but unobserved interaction may change the apparent impact of a particular parameter. However, this situation probably does not occur for processor designers; it has been shown in [49] that the interaction between parameters is significant only when the impact of each parameter is by itself significant. End-to-end measurements of execution times were obtained using Hotleakage [44]. For each simulation run, a particular configuration is obtained from the PB design matrix. The rows of the design matrix represent different processor configurations while the columns represent the values of different parameters in each configuration. For instance, table 3 illustrates a PB design matrix that is useful to study the impact of 8 or less parameters on the execution time of a program.

Table 3: PB Design Matrix for 8 Parameters.

A	B	C	D	E	F	G	H	Time
-1	-1	-1	-1	-1	-1	-1	-1	7
1	-1	+1	+1	-1	+1	-1	-1	11
-1	-1	-1	+1	+1	+1	-1	+1	18
-1	-1	+1	+1	+1	-1	+1	+1	22
+1	+1	+1	-1	+1	+1	-1	+1	9
+1	-1	-1	-1	+1	+1	+1	-1	10
-1	+1	-1	-1	-1	+1	+1	+1	33

In the PB design matrix, a “+1” or high value indicates a parameter value that is higher than the normal range of values whereas a “-1” or low value indicates a parameter value that is lower than the normal range of that parameter values. The values of a particular parameter are not restricted to numerical values only. For instance, in case of cache replacement policy, a random replacement policy may indicate a low value while a Least Recently Used policy may indicate a high value. In this work,

execution samples were generated based on PB design matrix that captures the interactions between program's working set and 39 different micro-architectural parameters. The selected parameters and their PB values are shown in Table 4. As mentioned earlier, this work assumes a superscalar processor with a 2-level cache hierarchy. The first level (L-1) consists of separate instruction cache (I-Cache) and a separate data cache (D-Cache) while the second level contains and a unified cache (U-Cache). The PB design matrix takes into account the size, block size, associativity, replacement policy and the latency of each cache level. The cache hierarchy also employs an instruction translation lookaside buffer (I-TLB) and a data lookaside buffer (D-TLB). The size, page size, latency and associativity of each TLB have also been considered in the PB matrix. Other parameters of the memory hierarchy include main memory latency (Latency-First, Latency-Second) and its bandwidth. The PB matrix includes the parameters of the processor and its functional units as well. Processor parameters include Instruction Fetch Queue (IFQ), Return Address Stack (RAS), Branch Prediction Type, Branch Misprediction Penalty, Branch Target Buffer (BTB), Reorder Buffer (ROB), Load/Store Queue (LSQ) and Number of Memory Ports. On the other hand, functional unit's parameters include: number of integer arithmetic and logic units (ALU), number of integer Multiplier/Divider units, number of floating point (FP) ALUs and the number of FP Multiplier/Divide units. Based on the parameters shown in table 4, a PB matrix with 48 simulation runs has been created. The PB matrix has been created using minitab statistical software [50]. This number of simulation runs provides a reasonable threshold to identify the interactions among the parameters of interest. In order to create the ETP, based on which the statistical analysis will be carried out, an application should be tested with different input patterns or data sets that are guaranteed to excite all possible control-flow paths within the CFG of the application. However, it is intractable to test the application with all possible input patterns. Consequently, it is important to test the application with a reduced yet a representative set of input patterns. In this work, simple random sampling (SRS) has been used to select a set of input patters from the population of possible inputs. In SRS, each possible input has the same chance of being selected [20].

Table 4: PB Matrix Parameters and their Values.

Parameter	Low Value	High Value
Processor		
IFQ Size	4	32
Branch Predictor	Not Taken	Perfect
Branch Misprediction Penalty	10	2
Number of RAS Entries	4	64
Number of BTB Entries	16	512
Associatively of BTB	Fully	2
Speculative Branch Update	Decode	Write Back
LSQ Size	2	64
Number of Memory Ports	1	4
Functional Units		
Number of Integer ALUs	1	4
Number of (FP) ALUs	1	4
Number of Integer Multiplier/Divisor Units	1	4
Number of FP Multiplier/Divisor Units	1	4
Memory Hierarchy		
L-1 I-Cache Size	4 KB	128 KB
L-1 I-Cache Assoc.	1	8
L-1 I-Cache Block Size	16	64
L-1 I-Cache Replacement Policy	Random	LRU
L-1 I-Cache Latency	4	1
L-1 D-Cache Size	4 KB	128 KB
L-1 D-Cache Assoc.	1	8
L-1 D-Cache Block Size	16	64
L-1 D-Cache Replacement Policy	Random	LRU
L-1 D-Cache Latency	4	1
L-2 U-Cache Size	256 KB	8192 KB
L-2 U-Cache Assoc.	1	8
L-2 U-Cache Block Size	64	128
L-2 U-Cache Replacement Policy	Random	LRU
L-2 U-Cache Latency	20	5
Memory Latency, First	200	50
Memory Latency, Second	4	1
Memory Bandwidth	8	32
I-TLB Size	32	256
I-TLB Page Size	4 K	4096 K
I-TLB Associativity	2	fully
D-TLB Size	32	256
D-TLB Page Size	4 K	4096 K
D-TLB Associativity	2	fully
I-TLB, D-TLB Latency	80	30

Its usage assures that regardless of the size of input population, statistical analysis based on the sampling distribution generated by SRS can reasonably estimate the parameters of the underlying population. Having selected the input patterns, each selected input has been simulated under all possible configurations provided by the PB matrix and its execution time has been observed. The set of all execution times obtained by testing each selected input on all possible PB configurations yields the population of execution times based on which statistical inference can be made. For each cache level, three execution time populations have been created; one for the base case where no leakage-control is applied and the other two represent the cases where either Gated-Vss or Drowsy cache is applied.

3.2.2 Statistical Estimation of WCET

In this part, EVT techniques are applied on execution time populations in order to obtain a statistical WCET estimate. The block maxima (BM) technique [33] is used to fit execution time measurements to the GEV distribution. The probability density function (PDF) of the GEV distribution is given by [33]:

$$f(x \mid \mu, \sigma, \xi) = \frac{1}{\sigma} y(x)^{\xi+1} e^{-y(x)} \qquad (1)$$

Such that: $$y(x) = \begin{cases} \left(1+\xi\left(\frac{x-\mu}{\sigma}\right)\right) & , \xi \neq 0 \\ e^{-(x-\mu)/\sigma} & , \xi = 0 \end{cases}$$

Where: μ, σ and ξ are location, scale and shape parameters of the distribution, respectively. The basic operation of the BM method is illustrated in the flow chart shown in fig. 1. The ultimate goal of this process is to estimate distribution parameters that yields distribution's PDF that represents the sampled data and captures the tail behaviour of the underlying ETP. The suitability of the selected PDF to the sampled ETP can be tested via an appropriate GOF test. This work uses Chi-square test [51] to check whether the estimated parameters of the GEV distribution can truly fit the measured execution time values. Once the estimated PDF passes the GOF test, it will represent the probabilistic model upon which WCET estimation can be made. Our proposed WCET estimation algorithm, based on BM method, is depicted in fig. 2. This approach has been implemented using MATLAB [52]. The proposed algorithm takes as input an ETP which consists of a number of execution time samples and produces a WCET estimate with a predefined confidence level.

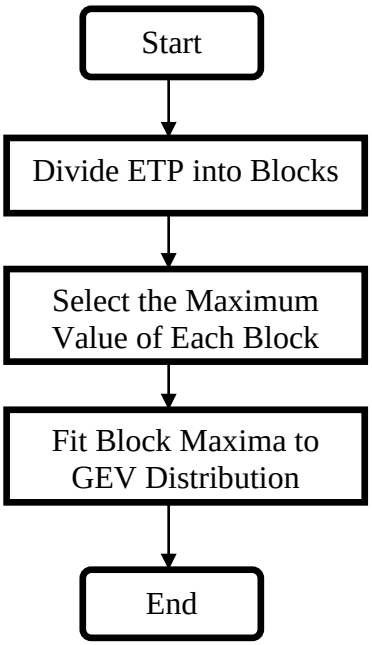


Fig. 1: Block maxima (BM) flow chart.

Algorithm 1 WCET Estimation

```
1:  procedure GEVFIT(ETP)
2:      bSize ← 40                          ▷ Initial Block Size
3:      nB ← ⌈(n/bSize)⌉                    ▷ Number of Blocks
4:      Flag ← False
5:      while Flag ≠ True do
6:          Divide ETP into nB Blocks
7:          if nB < 20 then return          ▷ Terminate
8:          end if
9:          i ← 0
10:         while i ≠ nB do
11:             Maxima[i] ← Maximum [ Block i]
12:             i ← i + 1
13:         end while
14:         Flag ← GOF(Maxima, bSize, nB)
15:         nB ← nB + 1
16:     end while
17:     [σ, μ, ξ] ← MLE(Maxima)
18:     MakeWCETEstimate(σ, μ, ξ)
19: end procedure
```

Fig. 2: WCET Estimation Algorithm.

The ETP is first divided into an equal-size number of blocks (*nB*) and the maximum value in each block will be observed. However, the number of blocks is not arbitrarily chosen but rather should be selected based on two important conditions. First, the number of blocks should be greater than 20 in

order to provide a sufficient number of samples for statistical analysis [51]. Second, the value of *nB* should be chosen such that the GEV distribution parameters estimated based on that value pass the GOF test performed in line 14. Hence, our algorithm checks multiple values of *nB* until the aforementioned conditions are satisfied. This can be seen in lines 5 through 16 in fig. 2. Fig. 3 illustrates the pseudocode of the GOF test. This test takes as input block maxima that have been observed based on a particular value of *nB* and returns a Boolean value that indicates whether the observed block maxima complies with the GEV distribution. As shown in Fig. 2, the GOF procedure is called iteratively until a distribution fit is achieved. The GOF test procedure receives as input a vector of block maxima and uses the Maximum Likelihood Estimate (MLE) method to estimate relevant GEV distribution parameters.

Algorithm 1 Goodness of Fit Analysis

```
1: procedure GOF(Maxima)
2:     Flag ← False
3:     [σ, μ, ξ] ← MLE(Maxima)
4:     Flag ← ChiSquareTest(Maxima, σ, μ, ξ)
5:     Return Flag
6: end procedure
```

Fig. 3: Pseudocode of GOF Test.

The estimated parameters and the vector of maxima are then passed to the Chi-Square test procedure in order to determine the suitability of the estimated GEV parameters to the sampled block maxima. The MLE method is an extremely important approach to estimation in statistical inference. A formal definition of MLE can be stated as follows [53]:

Definition 1: Given independent observations x_1,x_2,x_3 .... x_n from a probability density function (PDF) $f(x; \theta)$ where x and θ represent variables and distribution parameters respectively, the maximum likelihood estimator is that which maximizes the likelihood function

$$L(x_1, x_2, ..., x_n; \theta) = \prod_{i=1}^{n} f(x_i, \theta) \qquad (2)$$

Such that θ = (σ, μ, ξ). In general, likelihoods are conditional probability densities that can be used in either of two cases: First, when θ is fixed, the PDF $f(x; \theta)$ is used to compute the density at x, $f(x| \theta)$. Second, when x is fixed, the PDF is used to find the likelihood of the parameters θ, $f(\theta|x)$. Quiet often, it is more convenient to work with the natural logarithm of the likelihood function in finding the maximum of that function. Hence, If the set of execution times $\{x_i\}$ contained in the vector of block maxima are independent and identically distributed from a GEV distribution, then the log-likelihood function for a sample of *n* block maxima $\{x_1,x_2,...,x_n\}$ is

$$\ln[L(\theta \mid x)] = -n\ln(\sigma) + \sum_{i=1}^{n}\left[\left(\frac{1}{\xi - 1}\right)\ln(y_i) - (y_i)^{\xi}\right] \qquad (3)$$

Such that: $y_i = \left[1 - (\frac{\xi}{\sigma})(x - \mu)\right]$.

Once the GEV parameters are obtained from the MLE method, Chi-square test is performed to assess whether the observed block maxima have truly come from a GEV distribution whose parameters have been returned by the MLE method. Chi-square test is a mathematical formalization of the intuitive idea of comparing the histogram of block maxima to the shape of the candidate GEV distribution. In this work, the Chi-square test has been implemented according to the pseudo code shown in fig. 4. This implementation has been derived from [52]. As shown in fig. 4, the test divides the vector of block maxima into K class intervals based on the rules given in table 5 [52]. It then computes the test statistic as:

$$\chi_0^2 = \sum_{i=1}^{k} \frac{(O_i - E_i)^2}{E_i} \qquad (4)$$

Such that: O_i and E_i are the observed and expected frequencies in the i^{th} class interval, respectively. The expected frequency E_i in a particular class interval is computed as np_i where n is the number of elements in the maxima vector and p_i is the theoretical probability associated with the i^{th} class interval. The value of p_i is computed based on the parameters estimated by the MLE method. According to [52], the test statistic given by eq. (4) follows the chi-square distribution with *(K-s-1)* degrees of freedom (*df*), where s is the number of parameters of the candidate distribution estimated by the MLE method. The GEV distribution has an *s* value of 3. Having computed the test statistic, it should then be compared against the ***critical value*** of that statistic at *df* degrees of freedom and a particular level of significance [52].

Algorithm 1 Chi-Square Implementation

```
1: procedure ChiSquareTest(Maxima, σ, μ, ξ)
2:     Flag ← False
3:     Divide Maxima Vector into K class Intervals
4:     s ← 3
5:     df ← K − s − 1
6:     i ← 0
7:     while i ≤ K do
8:         Compute O[i]
9:         Compute E[i]
10:        i ← i + 1
11:    end while
12:    χ²0 ← Σ_{i=1}^{n} (O[i]−E[i])² / E[i]
13:    α ← 0.05                          ▷ Significance Level
14:    if χ²0 ≤ χ²α,df then
15:        Flag ← True
16:    else
17:        α ← 0.01
18:        if χ²0 ≤ χ²α,df then
19:            Flag ← True
20:        end if
21:    end if
22:    Return Flag
23: end procedure
```

Fig. 4: Chi-Square Test Pseudocode.

In this work, significance levels of 0.05 and 0.01 have been considered. As can be seen in fig. 4, the chi-square test will return ***true*** if the computed statistic is less than the critical value of that statistic at the given degrees of freedom and either of the considered levels of significance - with the 0.05 level being checked first. Otherwise, it will return ***false***. A ***true*** value indicates that the block maxima conform to the GEV distribution with the parameters estimated by the MLE method. Consequently, the fitted GEV distribution can then be used to make WCET estimations.

Table 5: Class Interval Rules

Sample Size (n)	Number of Class Intervals
< 20	Do not use chi-square test
50	5 to 10
100	10 to 20
>100	$\sqrt{n}$ or $n/5$

In this work, WCET estimation is based on the type of systems in which real-time task is implemented. Real-time systems can be classified as either soft or hard systems [1]. Soft real-time systems are those systems in which task deadlines can occasionally be missed while hard real-time systems are those systems in which deadlines are never allowed to be missed. For soft RTS, the WCET is quantified in terms of the ***return level*** (R_m) which is defined as the block maxima value that is expected to be exceeded only once in exactly *m* blocks. On the other hand, the WCET of hard RTS is quantified in terms of the ***WCET at risk*** (WCETaR) which is defined as the block maxima value with an extremely low probability of being exceeded. Fig. 5 depicts our proposed WCET estimation procedure. It gives the implementation of the procedure invoked in line 18 of the main algorithm given in fig. 2. As shown in fig. 5, WCET estimates are based on the GEV cumulative distribution function (CDF) which can be easily obtained by integrating the GEV PDF. Once the CDF is obtained, its inverse can be used to make WCET estimates for either soft or hard real-time systems.

Algorithm 1 WCET Calculation Algorithm

```
1: procedure MakeWCETEstimate (σ, μ, ξ)
2:     F(x) = ∫_{−∞}^{x} f(t)dt          ▷ Get GEV CDF
3:     p ← 0
4:     if RTS is soft then
5:         p ← 1 − 1/R_m
6:     else
7:         p ← (1 − p_e)
8:     end if
9:     WCET = F^{−1}(p)
10:    Return WCET
11: end procedure
```

Fig. 5: WCET Calculation Procedure.

3.3 Modelling of Real-Time Embedded Control Systems (RTECS).

This part studies the impact of processing delays caused by leakage-saving mechanisms on the schedulability of control tasks and, therefore, on the performance and stability of the controlled processes. RTECS models have been created using TrueTime [54]. TrueTime is a Matlab/Simulink-based package that can be used to simulate the temporal behavior of RTECS systems in which control algorithms are implemented as real-time tasks managed by a multitasking real-time kernel. The real-time kernel emulates the fundamental operation of a real-time operating system (RTOS). In this work, three types of RTECS models have been studied: **A)** a standalone RTECS in which a single physical plant is controlled by a CPU with RTOS, **B)** a standalone RTECS where multiple physical plants are controlled by a single CPU with multitasking RTOS. **C)** A networked RTECS in

which control system's components i.e. sensors, actuators and the controller process are communicating via a computer network. As mentioned before, control algorithms, in all scenarios, are implemented as real-time tasks. These tasks can be used to simulated both periodic and aperiodic activities. Associated with each task are a set of parameters that include task name, a release time which defines the time instant at which the task becomes ready to execute, a worst-case execution time (WCET) which is the maximum time budget allocated for a task, relative and absolute deadlines that define the time instant by which the task should finish its execution and a period which defines the exact difference between the releases of two consecutive instants of a particular task. Whereas some attributes like period and priority are kept constant, other attributes such as release time and absolute deadline are constantly updated by the real-time kernel. Most importantly, the WCET of each task can be set as a constant, a data-dependent value or a random variable. In this work, the WCET of each control task is modeled as a random variable generated from a particular GEV distribution. The selection of a particular GEV distribution depends on the used low-leakage technique and the cache level on which that technique is applied.

4. Results and Analysis

This section presents and analyzes the results that have been obtained by applying the experimental procedure outlined in the previous section.

4.1 Power Estimation Results

This section gives a comparative analysis between Gated-Vss and Drowsy cache in terms of their net power savings (NPS). Both techniques have been applied on L-1 I-Cache, L-1 D-Cache and L-2 unified cache. The aforementioned cache levels have been tested in a mutually exclusive manner i.e. only one cache level is put in low-leakage mode at a time while other levels are assumed to be in normal mode where no leakage-saving mechanism is applied. Fig. 6 compares the net power savings achieved by Gated-Vss and Drowsy cache techniques when applied on the L-1 D-Cache. Each bar represents the average net savings among the used benchmarks. The observations that can be made from this figure are twofold; first: the average net power savings, in both cases, is directly proportional to the LPI value; small values of LPI will prematurely put the cache line into low-leakage mode leading to a situation in which leakage-power savings are offset by the dynamic power caused by extra penalty cycles required to retain the cache line in the normal access mode. Second, Gated-Vss achieves higher average net power savings, as compared to Drowsy cache, at all values of LPI since it completely turns the cache line off which in turn leads to a zero leakage power consumption.

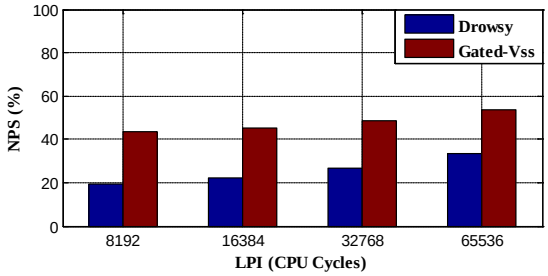


Fig. 6: L-1 D-Cache Power Savings.

On the other hand, fig. 7 depicts the average net power savings achieved by the two techniques when applied on L-1 I-Cache. Fig. 7 confirms the first observation that has been noticed in fig. 6. However, it shows a totally different behavior where Drowsy cache technique outperforms Gated-Vss at all possible LPI values; L-1 I-Cache has a much higher access rate as compared to the L-1 D-Cache. The access rate is defined as the number of cache accesses per cycle. Whereas the I-Cache is accessed at least once every clock cycle, the D-Cache will be accessed in only a particular portion of cycles which depends on the frequency of memory access instructions.

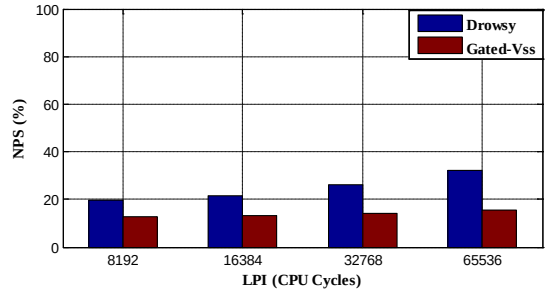


Fig. 7: L-1 I-Cache Power Savings.

Consequently, applying the Gated-Vss technique on the L-1 I-Cache has led to frequent accesses to the lower levels of the memory hierarchy. Such accesses require high dynamic power consumption which can ultimately outweigh the leakage savings achieved by the Gated-Vss technique. Moreover, the Drowsy cache technique does not incur any extra accesses to the lower levels of the memory hierarchy and, therefore, has achieved much better power savings when applied on the L-1 I-Cache.

Nevertheless, the average net power savings obtained by applying either of the techniques on L-1 I-Cache is smaller than that of the L-1 D-Cache. Finally, fig. 8 summarizes the average net power savings obtained by applying the two techniques on the unified L-2 cache. Fig. 8 also reveals the fact that power savings increase as the ILP value increases. In addition, the two techniques achieve almost equal net power savings. This observation can be explained as follows: Gated-Vss technique saves more leakage power but leads to high dynamic power consumption since a cache miss induced by putting a L-2 cache line into power-off mode requires an a access to the off-chip resources which consume extremely more power as compared to on-chip resources. On the other hand, Drowsy cache technique saves less leakage power but does not cause any extra accesses to the off-chip resources and, consequently, consume less dynamic power as compared to the Gated-Vss technique. Therefore, the two techniques appear to be identical in their potential power savings.

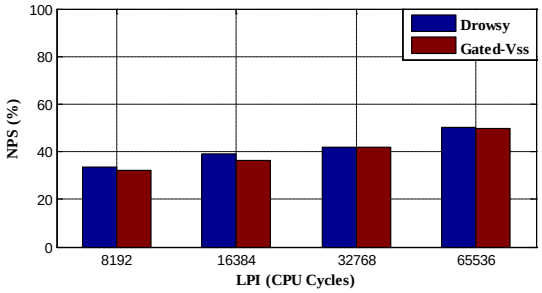


Fig. 8: L-2 Cache Power Savings.

4.2 Statistical WCET Estimation

This section shows and compares WCET estimates under Gated-Vss and Drowsy cache techniques at different cache levels. This analysis was performed based on the *insertsort* benchmark [55]. This benchmark has been tested with an integer array of one million elements. Several input patterns have created and one hundred of them have been chosen using the SRS method for ETP generation. Each input will be run on the 48 configurations provided by the PB matrix. Hence, each ETP consists of 4800 execution time value. There are three ETPs to test each particular cache level i.e. L-1 I-Cache, L-1 D-Cache and L-2 U-Cache. Table 6 gives the GEV maximum likelihood parameter estimates obtained by applying the algorithm given in fig. 2 on the ETPs generated for L-1 D-Cache analysis.

Table 6: GEV Parameters for L-1 D-Cache.

Mode	Parameter		
	ξ	σ	μ
Base	0.0533	0.6363	1.5149
Gated-Vss	0.1442	2.5114	4.880
Drowsy	0.0654	0.6489	1.6722

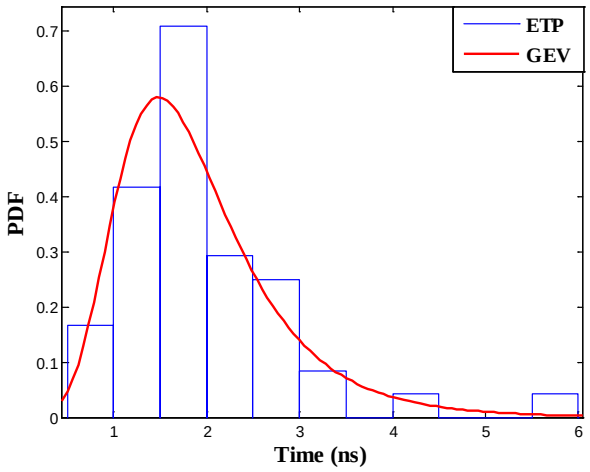
The base mode indicates the case when no leakage-saving mechanism is applied on any of the cache levels. These parameters can then be used to construct the GEV PDF and CDF upon which WCET estimates can be made. The quality of the ETP to GEV fit can be assessed visually by investigating the fitted PDF and CDF and comparing them against the histogram and the empirical CDF of the respective ETP, respectively. This step is summarized in fig. 9 and fig. 10. It can be observed that the empirical and estimated counter parts go in harmony with each other. The results shown in fig. 9 and fig. 10 in conjunction with the parameters given in table 6 indicate that the temporal behavior of the processor when the L-1 D-Cache is operated in the Drowsy mode is almost identical to that of the base mode where no leakage-saving mechanism is applied. Furthermore, the Gated-Vss mode has a higher parameter values and different tail behavior as compared to the other two modes. On the other hand, table 7 lists the GEV's maximum likelihood parameters estimated by applying the GEV fitting algorithm on the ETPs generated for the L-1 I-Cache. Based on table 7, it can be observed that the Drowsy mode has a close parameter estimates to that of the base mode while the Gated-Vss mode has higher parameter values as compared to the two other modes.

Table 7: GEV Parameters for L-1 I-Cache.

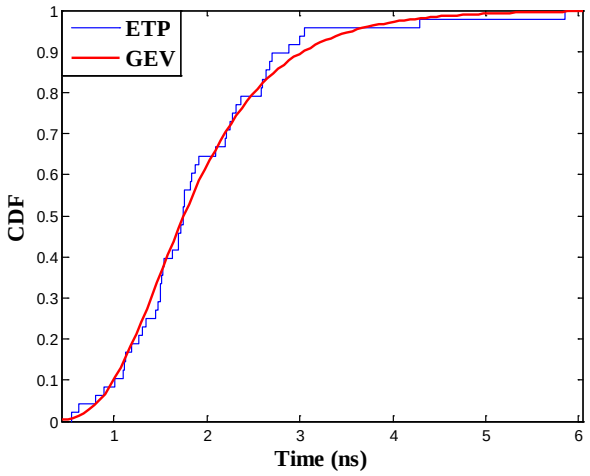
Mode	Parameter		
	ξ	σ	μ
Base	0.0533	0.6363	1.5149
Gated-Vss	0.8275	5.7159	6.6971
Drowsy	0.1182	0.7859	1.8875

Moreover, the GEV distribution under both leakage-saving techniques has higher parameter estimates when compared to the case of L-1 D-Cache. Consequently, applying the same leakage-saving technique on different cache levels could potentially lead to different tail behavior of the GEV distribution which in turn can lead to different WCET estimates. This fact can be proved

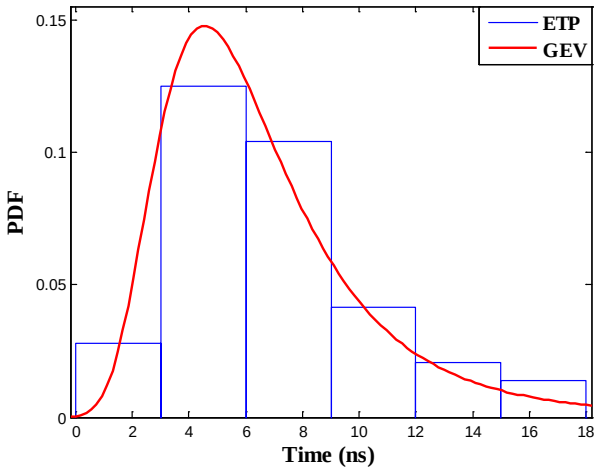
graphically by comparing the empirical and fitted PDF and CDF functions.



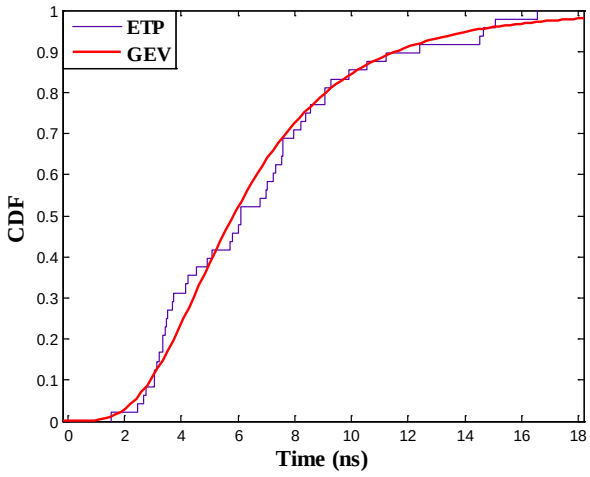
(a) L-1 D-Cache Base PDF



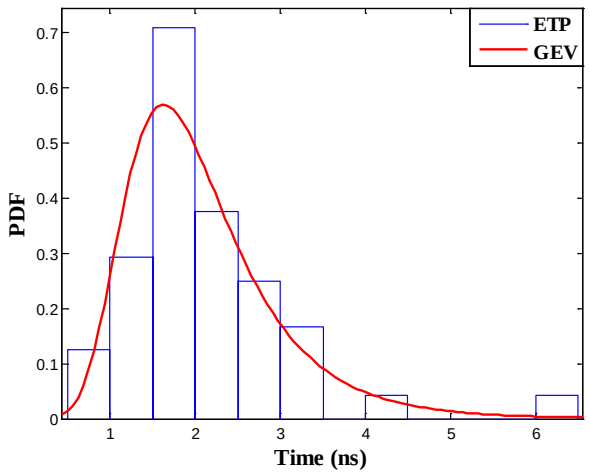
(a) L-1 D-Cache Base CDF



(b) L-1 D-Cache Gated-Vss PDF

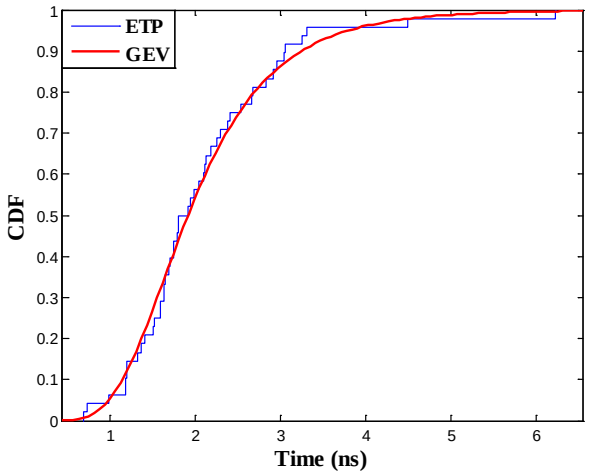


(b) L-1 D-Cache Gated-Vss CDF



(c) L-1 D-Cache Drowsy Cache PDF

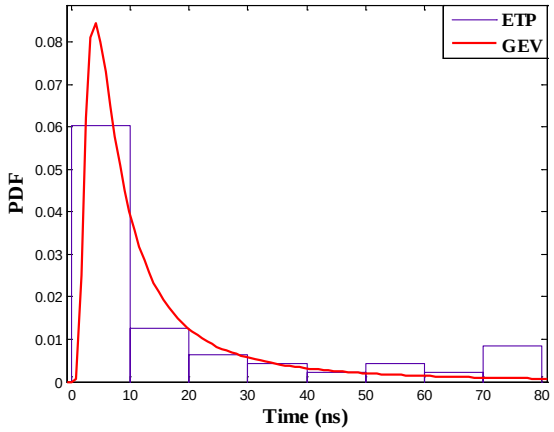
Fig. 9: L-1 D-Cache PDF.



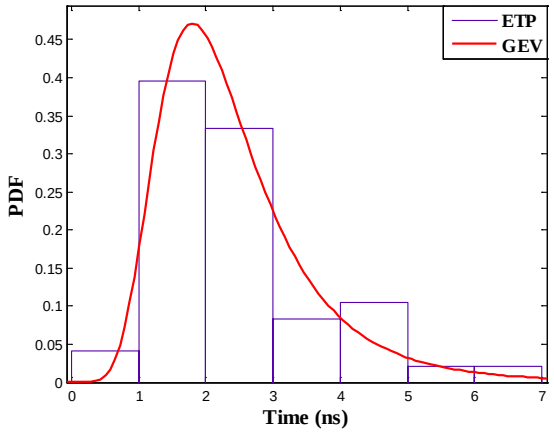
(c) L-1 D-Cache Drowsy Cache CDF

Fig. 10: L-1 D-Cache CDF.

Fig. 11 shows the histogram of the generated ETP, for L-1 I-Cache, overlaid with the PDF of the fitted GEV distribution. Similarly, Fig. 12 shows the empirical and fitted CDF functions. Based on fig. 9-12, it is apparent that the fitted and empirical distributions are consistent with each other. Moreover, both fitted and empirical distributions under Drowsy mode are almost identical to those estimated under the base mode.

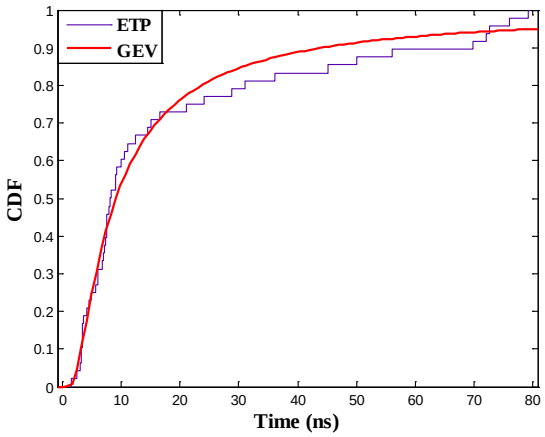


(a) L-1 I-Cache Gated-Vss PDF

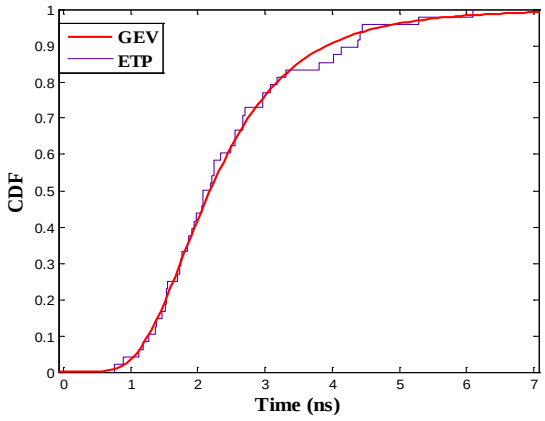


(b) L-1 I-Cache Drowsy PDF.

Fig. 11: L-1 I-Cache PDF.



(a) L-1 I-Cache Gated-Vss CDF



(b) L-1 I-Cache Drowsy CDF.

Fig. 12: L-1 I-Cache CDF.

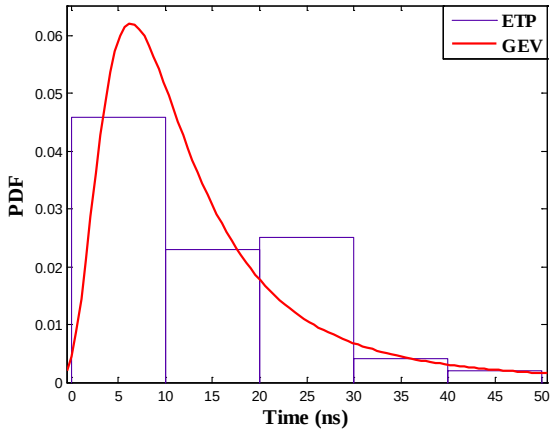
Hence, the temporal behavior of the processor under Drowsy mode is very close to that under base mode. Consequently, the processor can still provide the same level of time-predictability while at the same time consuming less power as compared to the base mode. On the other hand, for both L-1 I-Cache and L-1 D-Cache, the fitted GEV distributions under Gated-Vss mode are different from that of the base mode which, consequently, suggests a different temporal behavior from that observed under base and Drowsy modes. Hence, applying the Gated-Vss technique could potentially lead to different degree of time-predictability as compared to the base mode

Albeit providing significant leakage-savings especially when applied on L-1 D-Cache. Furthermore, table 8 shows the maximum likelihood parameter estimates for the GEV distributions obtained by applying the GEV fitting algorithm on the ETPs of the L-2 U-Cache.

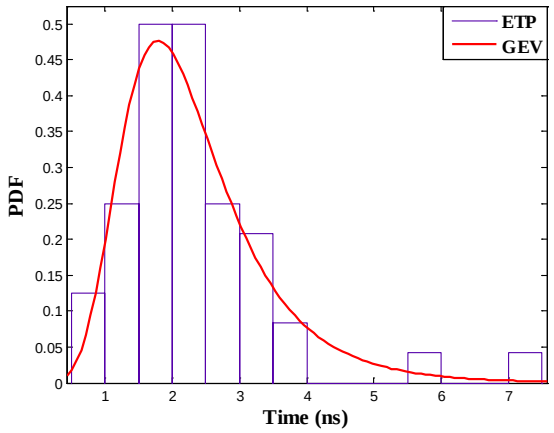
Table 8: GEV Parameters for L-2 U-Cache.

Mode	**Parameter**		
	ξ	**σ**	**μ**
Base	0.0533	0.6363	1.5149
Gated-Vss	0.3767	6.3096	8.2498
Drowsy	0.0618	0.7732	1.8407

Based on table 8, it is apparent that Gated-Vss mode has higher parameter estimates as compared to the other two modes.In addition, the GEV parameters for the Drowsy mode are very close to those of the base mode. Comparing these parameters with previous estimates, it can be noted that applying the Drowsy mode on L-1 D-Cache or L-2 U-Cache Would result in a temporal behavior that is to somehow identical to that of the base mode where no leakage-saving mechanism is applied. Moreover, the Gated-Vss technique yields a higher parameter estimates and, in turn, different temporal behavior as compared to the other modes.

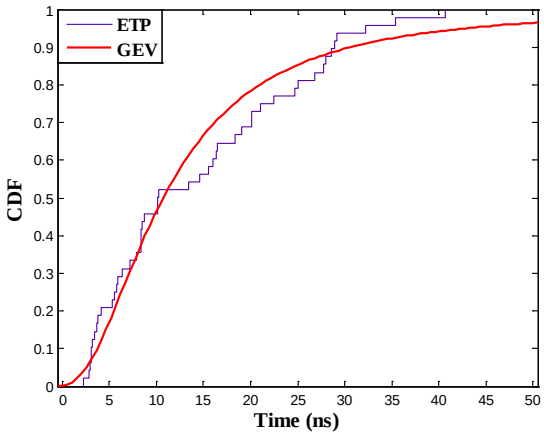


(a) L-2 U-Cache Gated-Vss PDF

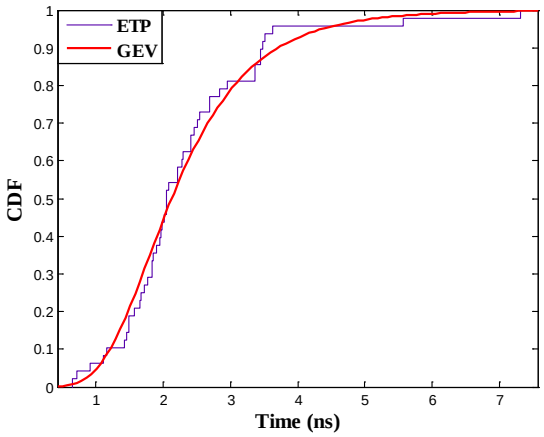


(b) L-2 U-Cache Drowsy Cache PDF

Fig. 13: L-2 U-Cache PDF.



(a) L-2 U-Cache Gated-Vss CDF



(b) L-2 U-Cache Drowsy Cache CDF.

Fig. 14: L-2 U-Cache CDF.

Similar to the previous cases, the quality of L-2 U-Cache ETP to GEV fit can be assessed graphically as shown in fig. 13 and fig. 14. As shown in the aforementioned figures, the histogram and the empirical CDF of the generated ETP are consistent with the fitted PDF and CDF of the GEV distribution. Furthermore, the PDF and CDF of the GEV generated under Drowsy cache mode are always very close to those of the base mode. On the other hand, the Gated-Vss mode has always different shape and longer tail behaviour in its PDF and CDF as compared to other modes of operation. The impact of such shape and tail behaviour differences will be further analyzed in the next paragraphs. Once the fitted GEV distributions for different low-leakage mechanisms at various cache levels have been obtained, they can be used to make WCET estimates for both soft and hard RTSs.Fig. 15 shows the WCETaR estimates for a processor in which the L-1 D-Cache is put into low-leakage mode. The WCETaR estimates are made at different exceedance probabilities (P_e). It can be observed that the Gated-Vss technique leads to higher WCETaR estimates as compared to the other modes while the Drowsy mode yields estimates that are very close to the base mode. In order to quantify the

impact of such discrepancy, in WCET estimates, on the suitability of leakage-saving mechanisms for RTSs, we propose the *Degree of Predictability* (DoP) measure. The DoP is defined as follows:

$$DoP = \frac{WCERaR_B}{WCETaR_{LK}} * 100\% \quad (5)$$

Where $WCETaR_B$ and $WCETaR_{LK}$ are the WCETaR estimates in the base and low-leakage modes, respectively.

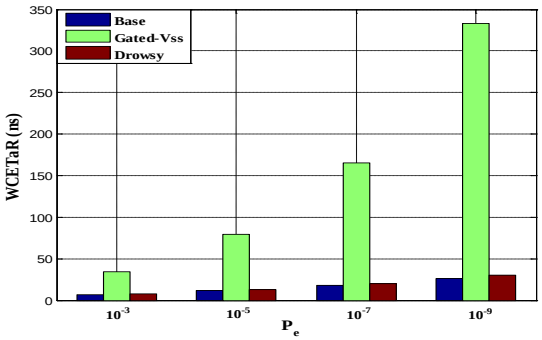


Fig. 15: L-1 D-Cache WCETaR.

Fig. 16 depicts and compares the DoP values under Gated-Vss and Drowsy cache modes. The DoP values under the Drowsy mode are much higher than those of the Gated-Vss mode; the WCETaR values under the Drowsy mode are very close to the estimates made under the base mode.

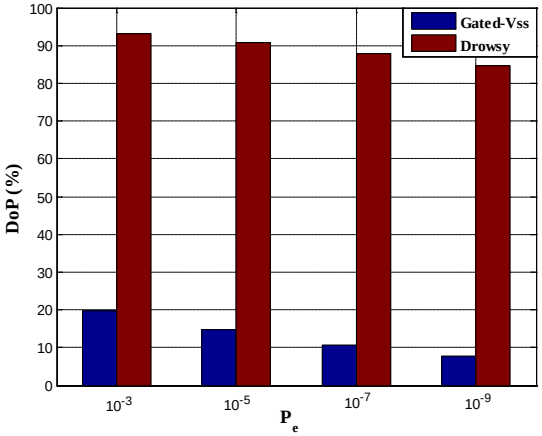


Fig. 16: DoP Comparison for L-1 D-Cache.

On the other hand, fig. 17 shows the WCETaR estimates assuming that the L-1 I-Cache is put into low-leakage mode while other levels are left intact. Fig. 17 confirms the fact that Gated-Vss always leads to higher WCET estimates as compared to other modes. Moreover, it can be noticed that WCET estimates under low-leakage L-1 I-Cache are higher than those made under low-leakage L-1 D-Cache. In addition, Fig. 18 compares the DoP values assuming a low-leakage L-1 I-Cache. The observations that can be made based on fig. 18 are twofold; first, using Gated-Vss technique introduces higher degree of unpredictability in the temporal behaviour of the processor as compared to the Drowsy cache mode. Second, the DoP under low-leakage L-1 I-Cache is always lower than the DoP under low-leakage L-1 D-Cache. Hence, from time-predictability perspective, it is preferred to apply leakage-saving mechanisms on L-1 D-Cache rather than applying it on L-1 I-Cache.

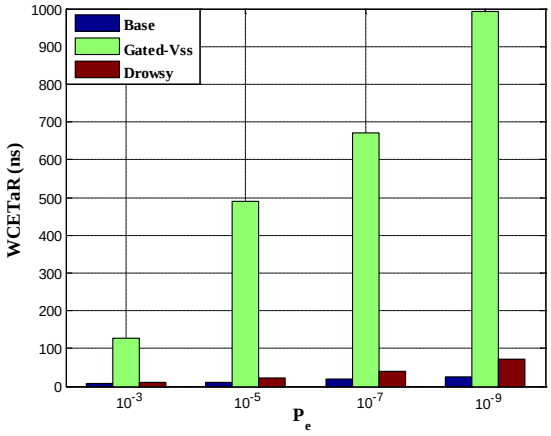


Fig. 17: WCETaR Estimates for L-1 I-Cache.

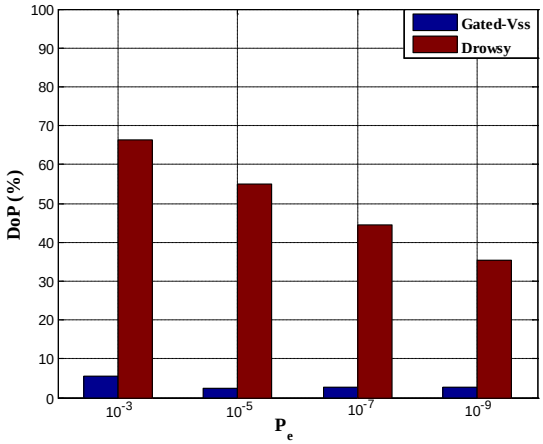


Fig. 18: DoP Comparison for L-1 I-Cache.

Furthermore, Fig. 19 gives the WCETaR values assuming a low-leakage L-2 U-Cache. Fig. 19 also confirms that Drowsy cache mode provides a close estimates to those made under base mode whereas Gated-Vss technique results in a wide predictability gap as compared to the base mode. Moreover, WCET estimates obtained under low-leakage L-2 U-Cache are higher than that of the L-1 D-Cache but lower than that of the L-1 I-Cache. The suitability of using low-leakage L-2 U-Cache in hard real-time systems can be evaluated by means of the DoP metric. Fig. 20 summarizes the DoP values

under possible L-2 U-Cache leakage-saving modes. By analyzing the results provided in fig. 20, it is clear that Drowsy L-2 U-Cache provides much better DoP as compared to the Gated-Vss one. However, the achieved level of predictability is less than that obtained by applying the same low-leakage mechanism on the L-1 D-Cache.

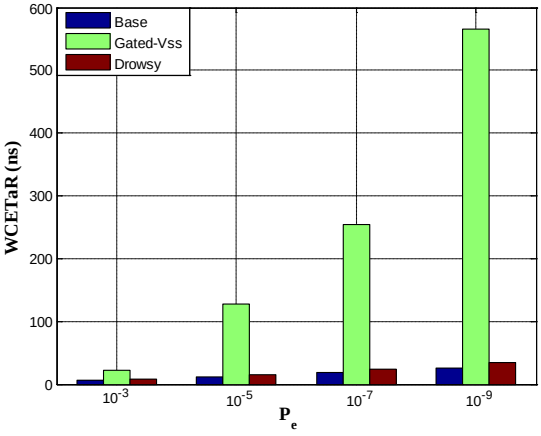


Fig. 19: WCETaR Estimates for L-2 U-Cache.

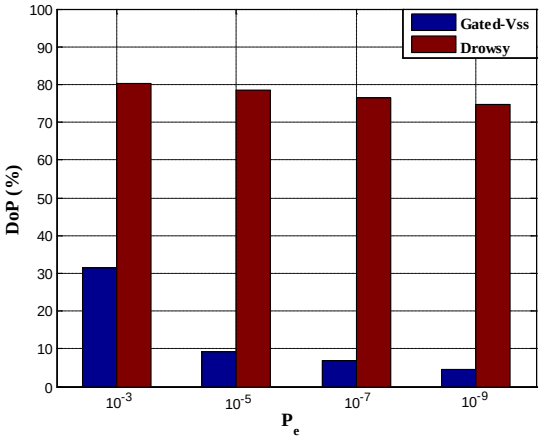


Fig. 20: DoP Comparison for L-2 U-Cache.

Thus, it can be concluded that applying the Drowsy cache technique on L-1 D-Cache provides high level of predictability and, consequently, represents a viable design consideration in low-leakage hard real-time systems. While the previous analysis serves as a solid guideline for hard RTS designs, it may not be as helpful for soft RTS analysis. Hence, the impact of each leakage-saving mechanism on the time predictability of such systems has been quantified in terms of another metric which is the return level (R_m). Table 9 shows the return level values for different values of m. It gives all possible combinations of leakage-saving mechanism and cache levels. Several observations can be made based on table 9: First, applying the Gated-Vss technique on any cache level results in higher R_m values as compared to the other modes. Second, applying the Drowsy mode one different cache levels causes varying degrees of proximity to the base mode; a Drowsy L-1 D-Cache or L-2 U-Cache produce R_m values that are closer to the base mode as compared to that of the L-1 I-Cache.

Table 9: Return Level Values.

Mode	**L-1 D-Cache**			
	R_{100}	**R_{400}**	**R_{800}**	**R_{1000}**
Base	4.83	6.00	6.62	6.83
Gated-Vss	21.28	28.79	33.14	34.63
Drowsy	5.15	6.43	7.11	7.34
Mode	**L-1 I-Cache**			
	R_{100}	**R_{400}**	**R_{800}**	**R_{1000}**
Base	4.83	6.00	6.62	6.83
Gated-Vss	310.66	981.92	1743.60	2097.5
Drowsy	6.69	8.73	9.89	10.28
Mode	**L-2 U-Cache**			
	R_{100}	**R_{400}**	**R_{800}**	**R_{1000}**
Base	4.83	6.00	6.62	6.83
Gated-Vss	86.26	151.48	199.26	217.50
Drowsy	5.95	7.45	8.24	8.50

Third, applying the Drowsy mode on any cache level tends to be more beneficial , from temporal perspective, than applying the Gated-Vss technique. In order to quantify the suitability of every combination of cache level and leakage-saving mechanism for use in soft RTS, we propose the soft unpredictability factor (SUF) as follows:

$$SUF = \frac{R_{m-LK}}{R_{m-b}} \qquad (6)$$

Where: $R_{m\text{-}LK}$ and $R_{m\text{-}b}$ are the return level values under the low-leakage and base modes respectively. Eq. (6) computes the factor by which $R_{m\text{-}LK}$ is higher than $R_{m\text{-}b.}$ Table 10 shows the values of SUF under all possible combinations of cache level, leakage-saving mechanism and return level. The level of predictability provided at each possible value of R_m is inversely proportional to the SUF at this value. However, depending on the value of SUF only may not clearly state the impact of each leakage-saving technique on the predictability of soft RTS. Hence, another comprehensive metric has been proposed and computed through an iterative process. This metric is referred to as Loss of Schedulability (LoS). Every iteration of this process consists of the following steps:

Table 10: SUF Values.

Mode	L-1 D-Cache			
	R_{100}	R_{400}	R_{800}	R_{1000}
Gated-Vss	4.41	4.80	5.01	5.07
Drowsy	1.07	1.07	1.07	1.07
Mode	**L-1 I-Cache**			
	R_{100}	R_{400}	R_{800}	R_{1000}
Gated-Vss	64.32	163.65	263.38	307.10
Drowsy	1.39	1.46	1.49	1.51
Mode	**L-2 U-Cache**			
	R_{100}	R_{400}	R_{800}	R_{1000}
Gated-Vss	17.86	25.25	30.10	31.84
Drowsy	1.23	1.24	1.24	1.24

A. Generate a total of 1000 random task sets. Each set contains four periodic tasks.
B. Loop through all task sets generated in step A and identify the task sets that are schedulable according to a fixed priority scheduling algorithm and add them to the set $\mathbf{T}_{sched}$. The number of tasks in this set is denoted as NUM_{sched} . Note: a task set is schedulable according to a particular scheduling algorithm if every task in this set finishes before its deadline.
C. For each task set in $\mathbf{T}_{sched}$, multiply the execution time of every task by the corresponding SUF value.
D. After updating the execution time of all tasks, iterate through all tasks sets and identify the tasks sets that became non-schedulable according to the scheduling algorithm used in step B. the number of non-schedulable tasks sets is denoted as $NUM_{non\text{-}sched}$.
E. Calculate LoS as ($NUM_{non\text{-}sched}$ / NUM_{sched}) * 100%.

In this work, the iterative process has been performed assuming 1000 iterations. The rate-monotonic (RM) scheduling algorithm has been used and its schedulability test has been performed using the Time-Demand Analysis (TDA) method [1]. Details on the RM algorithm and the TDA method can be found in [1]. Table 11 shows the LoS value under all possible leakage-saving techniques and cache levels. The reported LoS value is the average among all iterations of the iterative process. According to table 11, almost all tasks will miss their deadlines in two cases: 1) when the Gated-Vss technique is employed on any cache level, 2) when the Drowsy cache technique is applied on the L-1 I-Cache. Moreover, applying the Drowsy cache technique on L-1 D-Cache leads to a LoS of 0% which means that no task will miss its deadline due to the application of this technique, on L-1 D-Cache, as compared to the base mode.

Table 11: LoS Values.

Mode	L-1 D-Cache			
	R_{100}	R_{400}	R_{800}	R_{1000}
Gated-Vss	100%	100%	100%	100%
Drowsy	0%	0%	0%	0%
Mode	**L-1 I-Cache**			
	R_{100}	R_{400}	R_{800}	R_{1000}
Gated-Vss	100%	100%	100%	100%
Drowsy	98%	100%	100%	100%
Mode	**L-2 U-Cache**			
	R_{100}	R_{400}	R_{800}	R_{1000}
Gated-Vss	100%	100%	100%	100%
Drowsy	13.68%	22.9 %	22.95%	31.24 %

Furthermore, applying the Drowsy cache technique on L-2 U-Cache provides an acceptable range of LoS values. In summary, it can be concluded that Drowsy L-1 D-Cache is a suitable low-leakage design alternative for systems where tasks' usefulness decreases sharply after missing its deadline while Drowsy L-2 U-Cache can be an acceptable option for non-critical soft RTSs i.e. systems where task's usefulness decreases gradually after missing its deadline.

4.3 Analysis of RTECS Temporal Behaviour

In this section, the fitted GEV distributions and associated WCET estimates will be employed to study the impact of leakage-reduction techniques on the temporal behaviour and stability of RTECS in which a physical plant in controller by a resource-constrained computer with an RTOS. Fig. 21 shows a model of an RTECS in which a DC servo motor in controller by a digital computer. A Proportional Integral Derivative (PID) controller [56] is used to control the motor. A digitized form the PID controller is employed and implemented as a periodic task inside the RTOS. Periodically, the controlled process is sampled and its state variable (y) is compared with a reference input (r). The result of this comparison will be used as an input to the PID control algorithm which will compute an appropriate control signal (u) based on which the motor behaviour will be adjusted. The execution time of the PID control task is crucial to the operation of this RTECS; an excessive delay in the

computation of the control signal (u) could potentially lead to instability in the operation of the DC servo motor. In order to study the temporal behaviour of the system shown in fig. 21 under a particular low-leakage mechanism, the execution time of the control task is set as a random variable generated from the corresponding GEV distribution. Fig. 22 illustrates the behaviour of the RTECS shown in fig. 21 under the base mode where no leakage mechanism is applied. The upper part of fig. 22 shows the reference input r(t) overlaid with the actual output (y(t)) of the controlled DC motor. On the other hand, the lower part shows the control signal u(t) generated by the PID control algorithm. Overall, it can be observed that the DC motor has a satisfactory control performance and stability level under the base mode where no leakage-saving mechanism is applied. On the other hand, fig. 23 shows the temporal behaviour of the RTECS where the underlying computer has a Drowsy L-1 D-Cache. It indicates that the performance and stability of the RTECS is almost identical to that of the base mode where no leakage-saving technique is applied. Moreover, fig. 24 depicts the performance of the RTECS where the control task is executed by a computer with Gated-Vss technique is applied on the L-1 D-Cache. Fig. 24 illustrates the fact that the Gated-Vss technique has caused a poor control performance as compared to the other two modes; Gated-Vss technique has caused an elevation in the control task's execution time that the controller does not respond in a timely manner to the changes in the controlled process output.

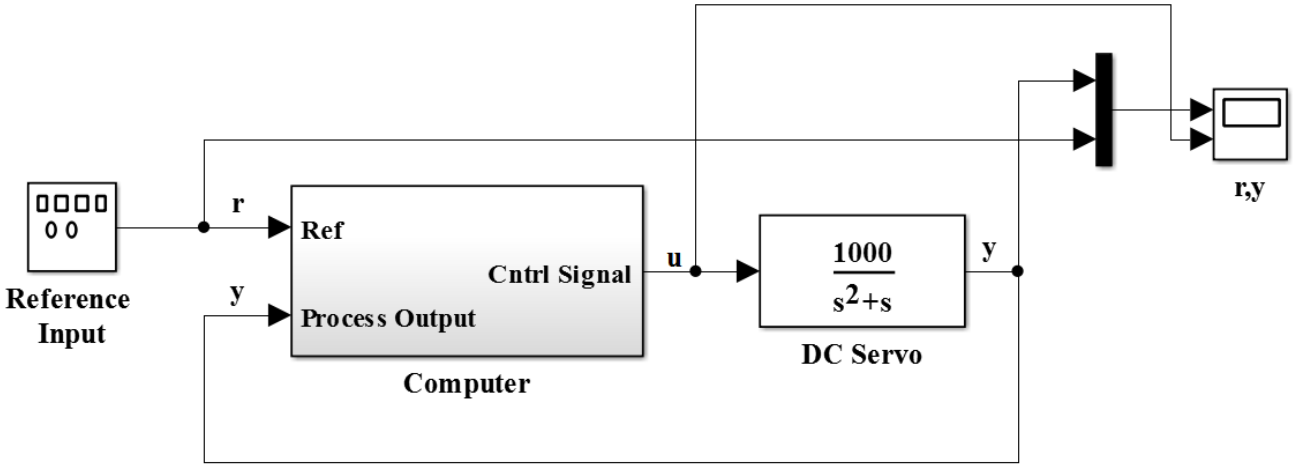


Fig. 21: A single-task RTECS Model.

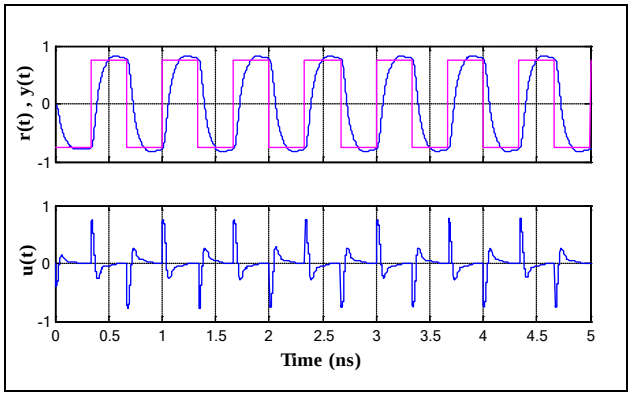


Fig. 22: Single-task RTECS Performance Under Base Mode.

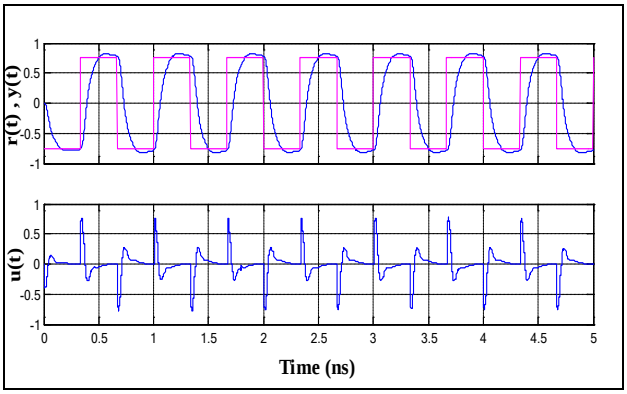


Fig. 23: Single-task RTECS Performance Under Drowsy L-1 D-Cache.

Furthermore, fig. 25 depicts the performance of the single-task RTECS assuming that the computer has a Drowsy L-1 I-Cache. Apparently, the performance of the RTECS is identical to that under the base mode. The performance of the single-task RTECS has also been studied in a system where the controlling computer applies Gated-Vss on the L-1 I-Cache as shown in fig. 26. Fig. 26 shows that the application of Gated-Vss technique on L-1 I-Cache has resulted in a system; applying the Gated-Vss

technique on L-1 I-Cache has yielded an execution time that is much higher than the sampling period of the control algorithm and , in turn, resulted in a situation where the controller cannot keep pace with the changes in the motor's state variable.

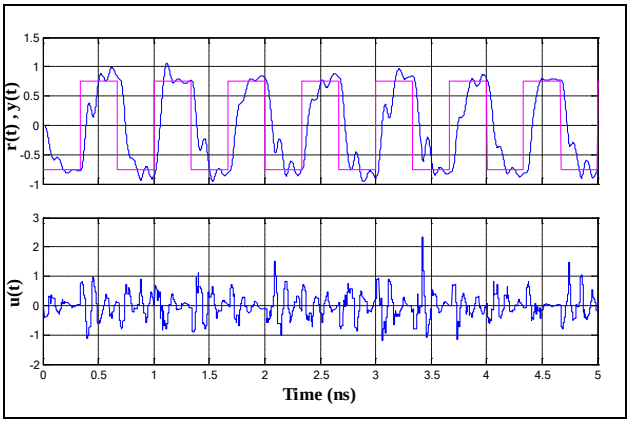


Fig. 24: Single-task RTECS Performance Under Gated-Vss L-1 D-Cache.

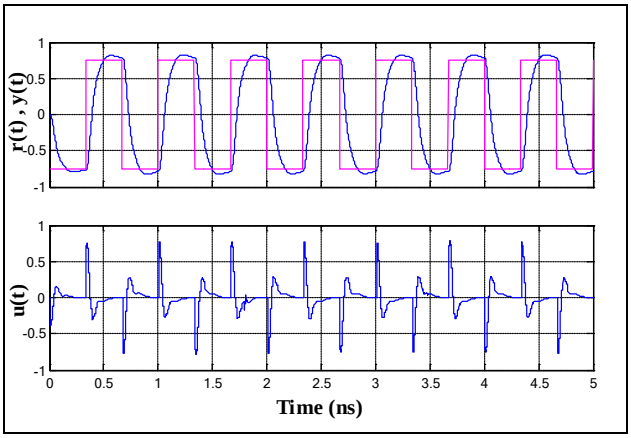


Fig. 25: Single-task RTECS Performance Under Drowsy L-1 I-Cache.

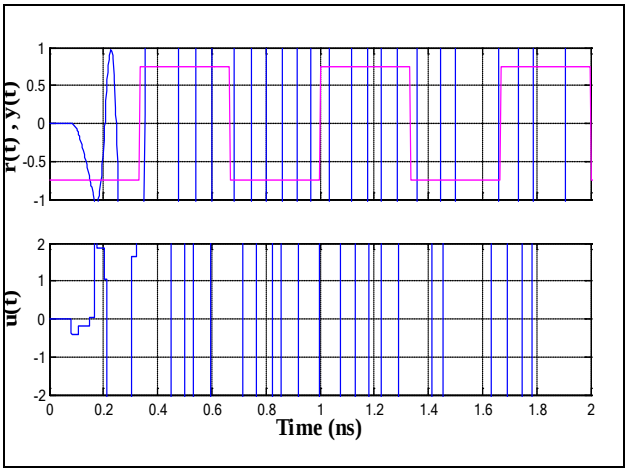


Fig. 26: Single-task RTECS Performance Under Gated-Vss L-1 I-Cache.

Finally, Fig. 27 and 28 shows the performance of the single-task RTECS under Drowsy and Gated-Vss L-2 U-Cache, respectively. Once again, the Drowsy cache technique outperforms the Gated-Vss technique and leads to better control performance and system stability when applied on the L-2 U-Cache. As shown in fig. 28, the Gated-Vss technique has also resulted in extremely poor control performance and unstable system when applied on the L-2 U-Cache.

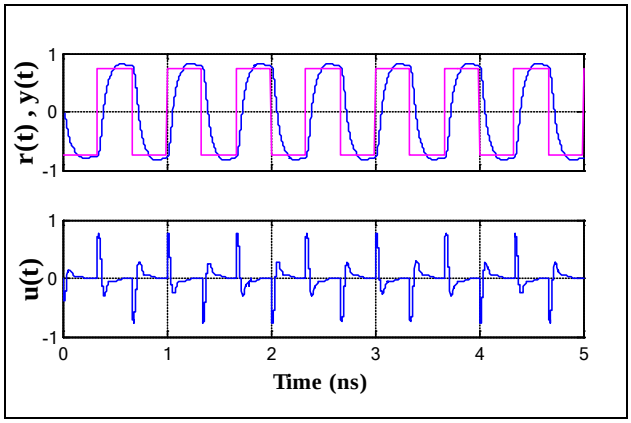


Fig. 27: Single-task RTECS Performance Under Drowsy L-2 U-Cache.

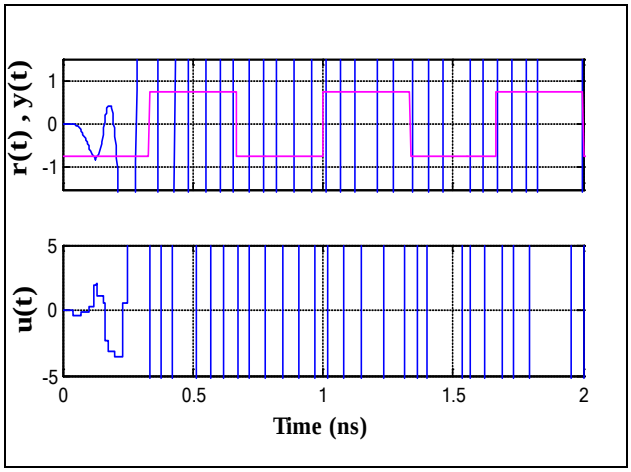


Fig. 28: Single-task RTECS Performance Under Gated-Vss L-2 U-Cache.

On the other hand, fig. 29 shows a TrueTime model of a multi-task RTECS with three DC servo motors controlled by a single computer equipped an RTOS. The three motors are controlled by PID control algorithms with different sampling period for each motor. The sampling periods of DC servos 1, 2 and 3 are set as 6, 5 and 4 ns respectively. Each algorithm is implemented as a real-time task within the RTOS kernel. The RTOS uses the preemptive earliest-deadline first (EDF) algorithm [1] to schedule the executing control tasks. According to the EDF algorithm, the task with earliest deadline

has the highest priority and can preempt any other executing task. The EDF algorithm has been selected due to its optimality; it can find a feasible task schedule provided that one exists. The control algorithms have been executed under different sampling periods in order to create a task set with

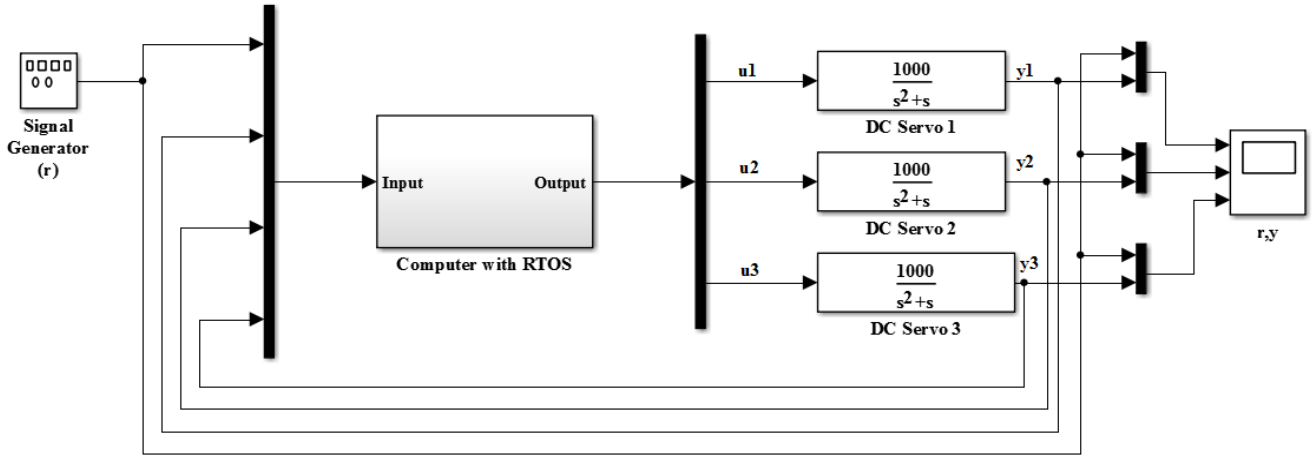
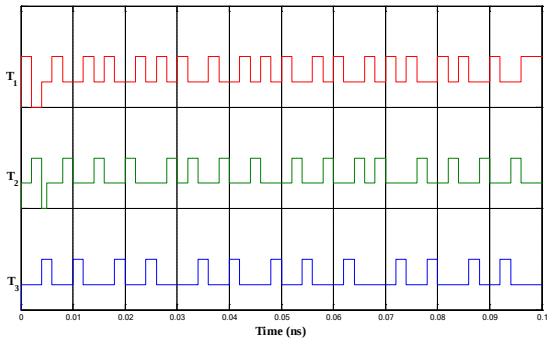
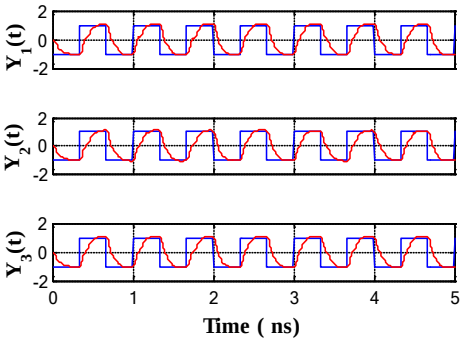


Fig. 29: A multi-task RTECS Model.

Different priorities. Having tasks with different priorities can create preemption points under which the impact of execution time variations, due to leakage-saving mechanism, can be studied not only for a single-task but also for other competing tasks in the system especially those tasks with lower priority. Like the previous RTECS, the impact of each leakage-saving mechanism on the control performance and stability of each individual DC servo motor and on the system as a whole is modeled by setting the execution time of the control tasks as a random variable generated from a particular GEV distribution. In this model, the execution time of the three control tasks, under any mode, is generated from the same GEV distribution since they are executing on the same computer. Fig. 30 shows the results of simulating the multi-task RTECS in base mode. The upper part of this figure shows control task's schedule while the lower part shows the performance of each DC servo motor as compared to the reference input. The control tasks of DC motor i is denoted as T_i for i = 1, 2, 3. At any time instant, only one control task can be executing while other tasks can be either idle or preempted. Hence, any task can be in one of three possible states: executing, idle, preempted. These states can be seen as high, medium or low levels on the schedule chart, respectively. As shown in fig. 30-a, control tasks has gone through an alternating priority behavior, for example, task T_1 may have higher priority than T_2 at some point of time while T_2 may have higher priority in another time instant; the EDF algorithm is a dynamic priority scheduling algorithm in which instances of the same task may execute at different priority levels throughout system operation.



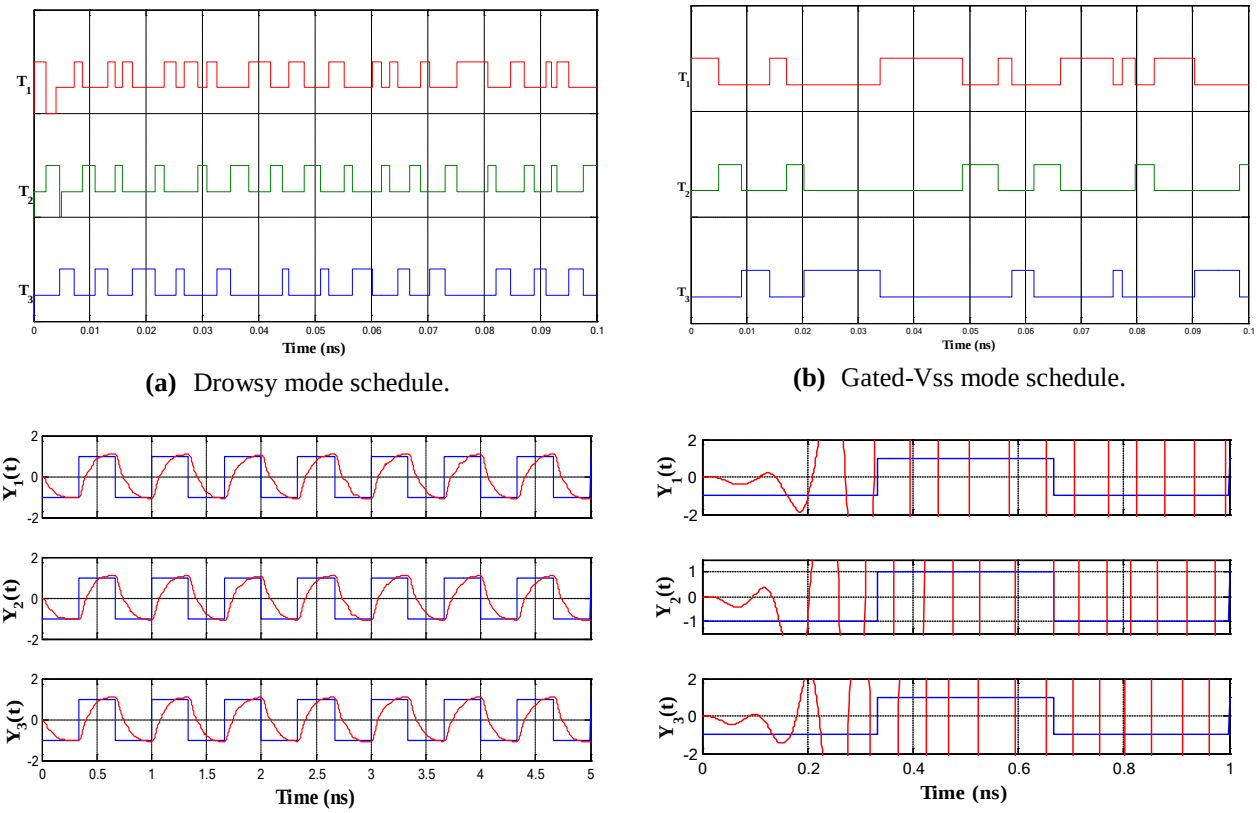
(a) Control tasks schedule.



(b) Performance of multi-task RTECS in base mode.

Fig. 30: Simulation of Multi-task RTECS in Base Mode.

This fact is important in two main facets: it ensures system fairness where each task gains access to the underlying computer and ensures that control tasks can mutually affect each other. The later point is very important since it allows RTS designers to study low-leakage mechanisms under high contention levels and, therefore, under situations that aggressively exercise the computer's memory system. Furthermore, fig. 30-b shows the performance of each controlled motor, under the base mode. Despite the large number of preemptions, the system can, in general, achieve a satisfactory control performance and stability margins. On the other hand, The performance of the multi-task RTECS has also been observed under leakage-saving techniques at L-1 I-Cache, L-1 D-Cache and L-2 U-Cache. Fig. 31, 32 and 33 show the schedule of control tasks besides the performance of the controlled processes under low-leakage L-1 D-Cache, L-1 I-Cache and L2- U-Cache, respectively.



(a) Drowsy mode schedule.

(b) Gated-Vss mode schedule.

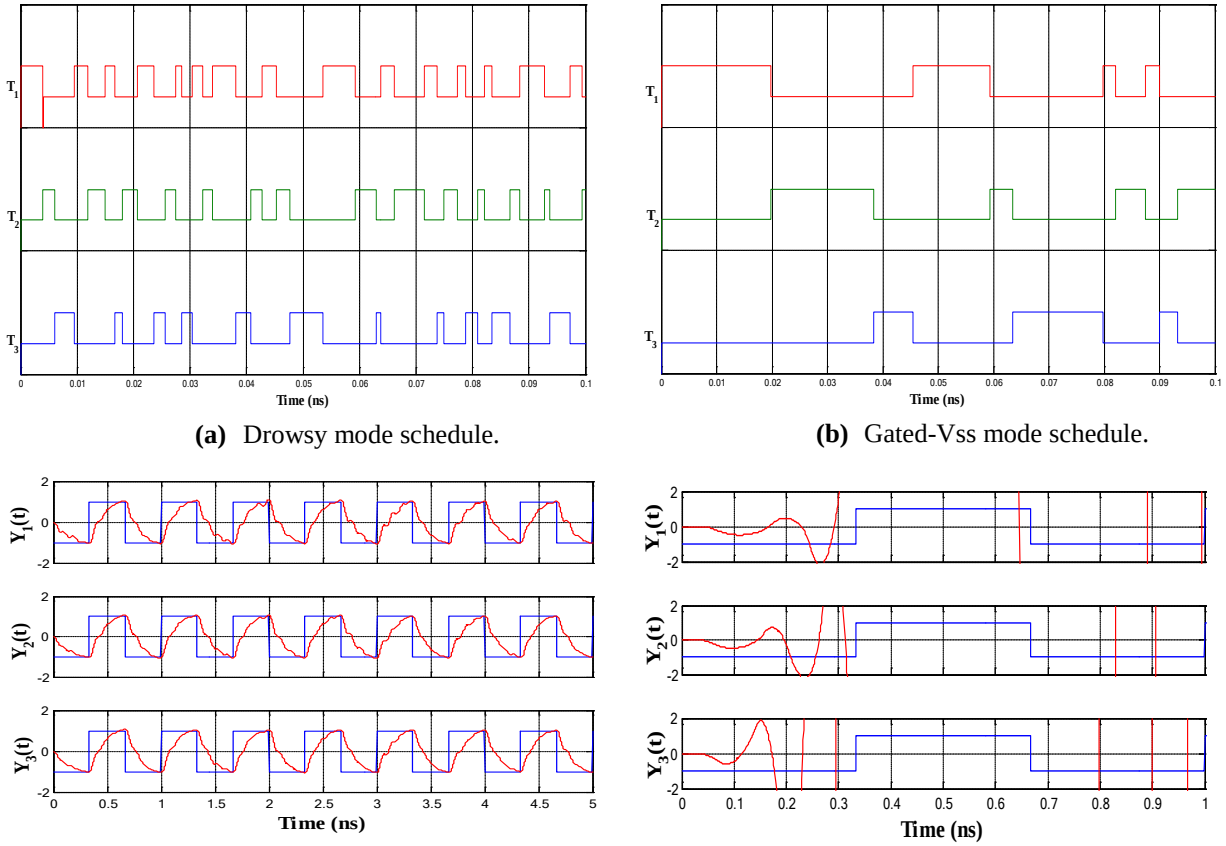
(c) Drowsy mode performance.

(d) Gated-Vss mode performance.

Fig. 31: Simulation of Multi-task RTECS Under Low-leakage L-1 D-Cache.

They reveal two main observations: first, the control task's schedule under Drowsy mode is almost identical to that of the base mode; Although applying Drowsy mode leads to an increase in control algorithm's execution time, this increase is very small that competing tasks do not undergo long preemption times which, therefore, has given each control algorithm a sufficient amount of processor's time to respond to changes in the state variable of the controlled process. Hence, the RTECS has achieved an acceptable control performance similar to that of the base mode as shown in part (c) of fig. 31, 32 and 33. Second, the control task's schedule and the corresponding control performance shows a totally different behavior under Gated-Vss mode; applying the Gated-Vss technique on any cache level has caused a significant increase in the control algorithm's execution time which , in turn, has caused long preemption delays. A long preemption delay would normally prevent some control tasks from gaining access to the computer. Hence, they will not respond to changes in the state variables of their respective controlled processor in a timely manner. Therefore, the RTECS has encountered a very poor and unstable control performance as shown in part (d) of the aforementioned figures. In summary, applying the Drowsy cache technique on any cache level provides a satisfactory control

performance and, consequently, provides a suitable design alternative for low-leakage RTECS.



(a) Drowsy mode schedule.

(b) Gated-Vss mode schedule.

(c) Drowsy mode performance.

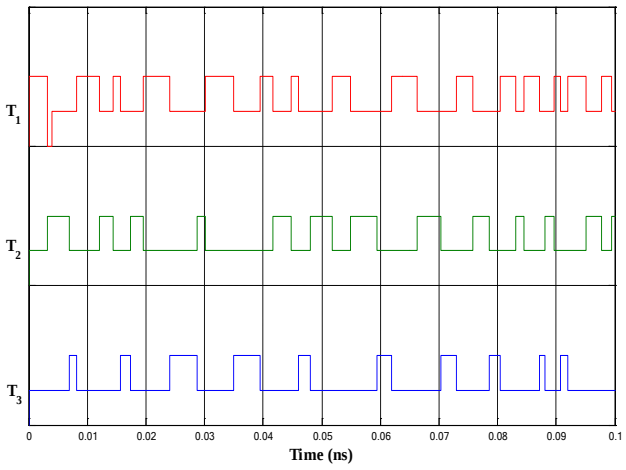
(d) Gated-Vss mode performance.

Fig. 32: Simulation of Multi-task RTECS Under Low-leakage L-1 I-Cache.

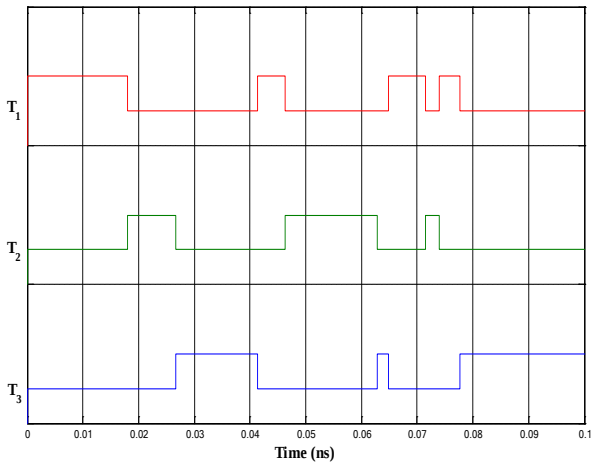
On the other hand, fig. 34 shows a model for a networked RTECS. In this model, the components of the control loop i.e. sensor, actuator and the controller (computer) are assigned to different nodes in the network. The network block is event-driven and executes when a message enters or leaves the network [55]. In this block, users can change the transmission rate of the network, the medium access control (MAC) protocol such as CSMA/CD, CSMA/CA, round robin, FDMA and TDMA [55]. In this work, the network has been configured with CSMA/CD (Ethernet) MAC protocol, a data rate of 80Kbps and a minimum frame size of 80 bits. Whereas the sensor node works in a time-driven manner, the actuator and the controller are event-driven components. Periodically, the sensor measures the current state of the DC motor and sends the sampled value to the controller over the network. The controller computes the appropriate control signal, based on a PID control algorithm, and sends this signal to the actuator node via the network. Finally, the state of the controller DC motor will be actuated accordingly. Hence, the total roundtrip time (T) taken between sensing and actuation can be computed as follows:

$$T = t_{sc} + t_p + t_{ca} \qquad (7)$$

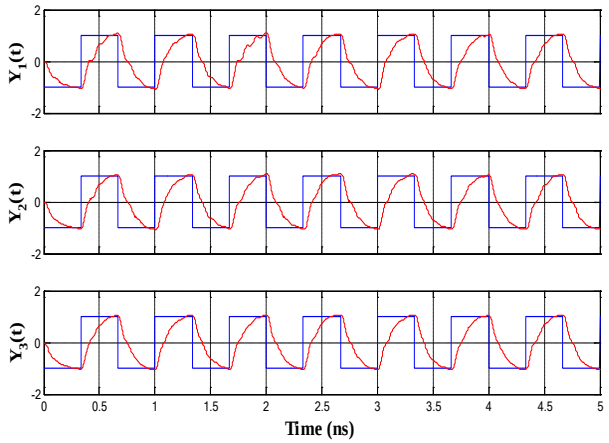
Such that: t_{sc} denotes the sensor-to-controller delay, t_{ca} is the controller-to-actuator delay and t_p is the processing delay taken by the controller (computer) to compute the appropriate control signal based on the value received from the sensor node. While the previous research efforts [38-43] have focused on only network-induced delays, our work has focused on computer-induced delays and shows its importance in the temporal behavior and performance of the networked control loop. Similar to the previous models, the impact of leakage-saving mechanisms on the performance of the networked RTECS model has been captured by setting the execution time of The control algorithm as a random variable generated from a specific GEV distribution.



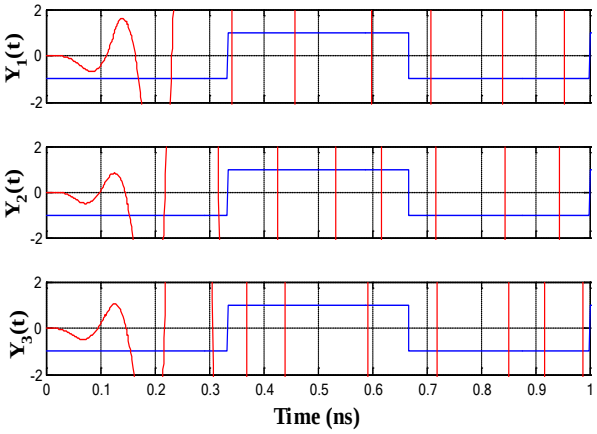
(a) Drowsy mode schedule.



(b) Gated-Vss mode schedule.



(c) Drowsy mode performance.



(d) Gated-Vss mode performance.

Fig. 33: Simulation of Multi-task RTECS Under Low-leakage L-2 U-Cache.

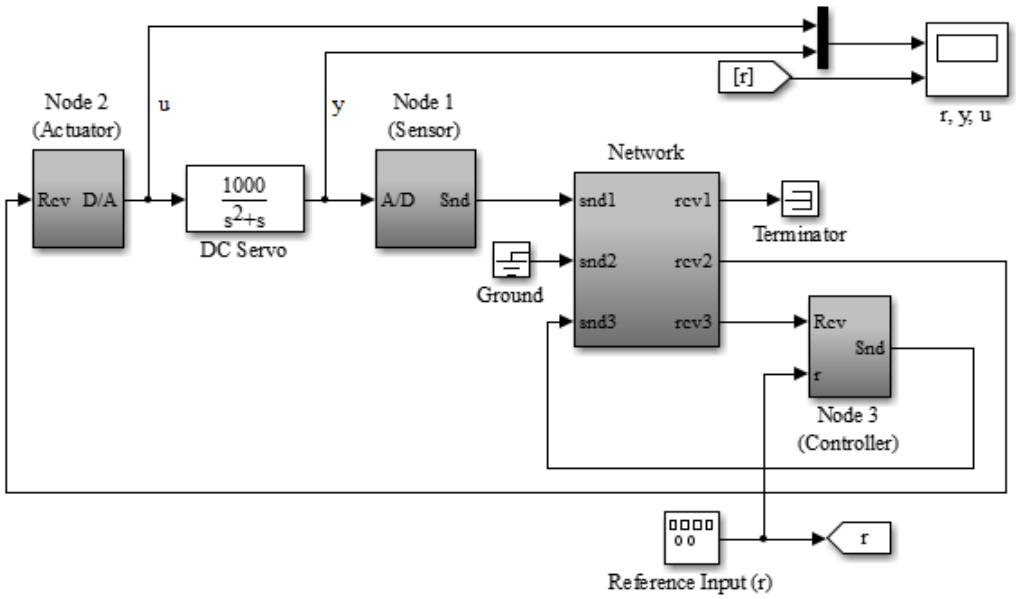


Fig. 34: Networked RTECS Model.

The GEV distribution is selected based on the leakage-saving mode and the cache level on which this mode is applied. Fig. 35 shows the performance of the DC motor assuming that the controller in operating in the base mode where no leakage-saving mechanism is applied. It shows that the output of the DC motor settles to the level of the reference input in a very small amount of time after undergoing a small overshoot level. Overall, the RTECS has achieved a satisfactory temporal behavior and control performance.

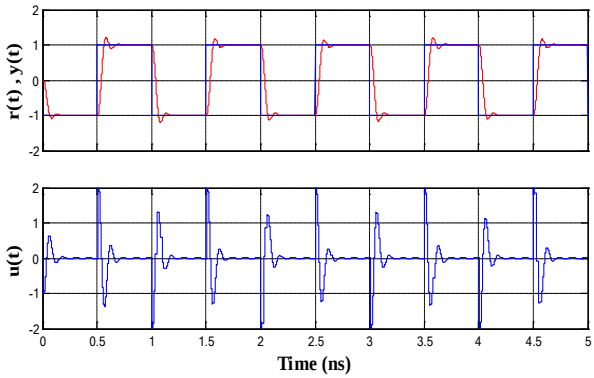
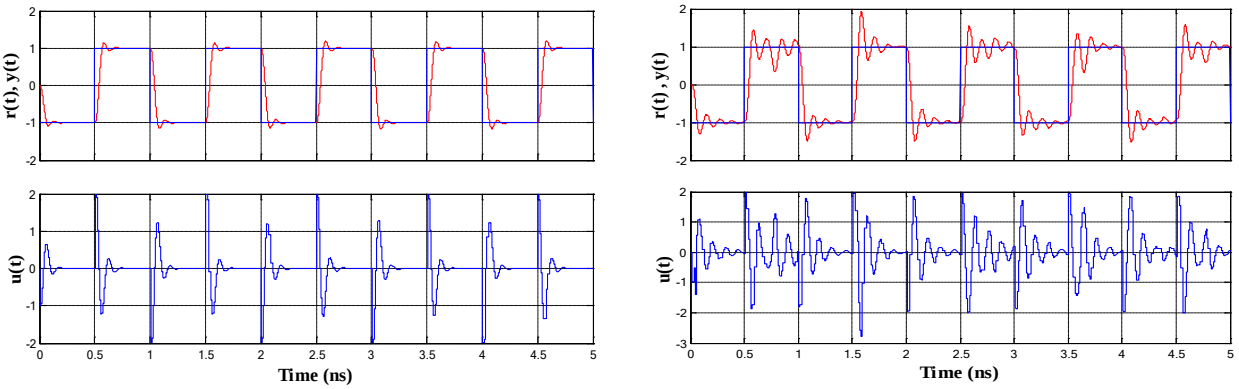


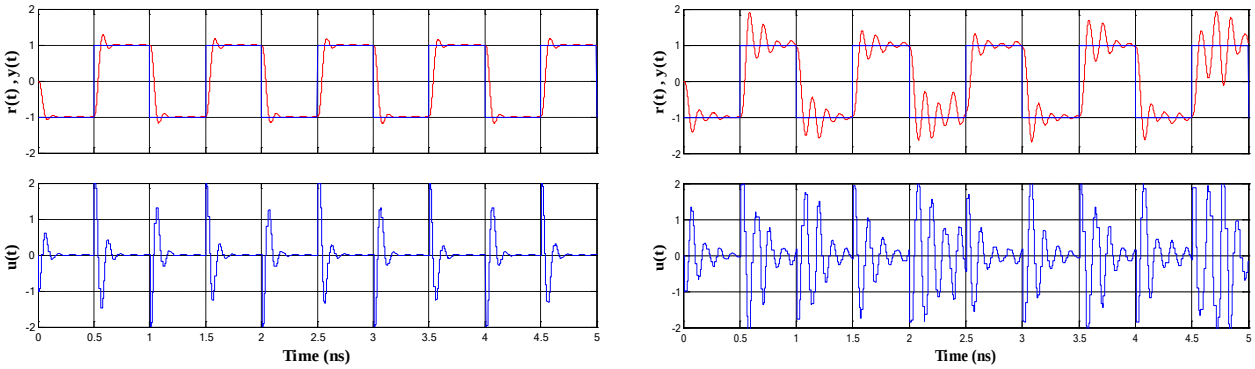
Fig. 35: Performance of Networked RTECS in Base Mode.

In addition, fig. 36 illustrates the control performance of the networked RTECS assuming a low-leakage L-1 D-Cache. Fig. 36-a shows the impact of applying Drowsy cache technique on the performance of the networked RTECS. It points out the the performance of the networked RTECS in presence of Drowsy L-1 D-Cache is identical to that achieved under the base mode. On the other hand, Fig. 36-b gives the performance of the networked RTECS in presence of a controller (computer) with an L-1 D-Cache operated in Gated-Vss mode. Apparently, the application of the Gated-Vss techniques on the L-1 D-Cache has resulted in a system with poor control performance. The waveform of the DC motor's output shows a high overshoot levels and very long settling time. In other words, the motor keeps fluctuating without settling to the steady-state case marked by the reference input signal. Therefore, it can be concluded that applying the Gated-Vss technique on L-1 D-Cache has caused a situation in which the time required to process a single sample, sent by the sensor, is very long that other samples cannot be processed in a timely manner and, consequently, has prevented the actuator from updating the motor's state appropriately. Hence, unlike the assumption made by previous work [38-43], the processing delay induced by the controller especially under low-leakage modes cannot be hidden or overlapped with network-induced delays and should be handled appropriately in order to maintain a satisfactory performance of networked control loops. On the other hand, fig. 37 and 38 depicts the performance of the networked RTECS under low-leakage L-1 I-Cache and L-2 U-Cache, respectively. They confirm the observations that have been made based on fig. 36.



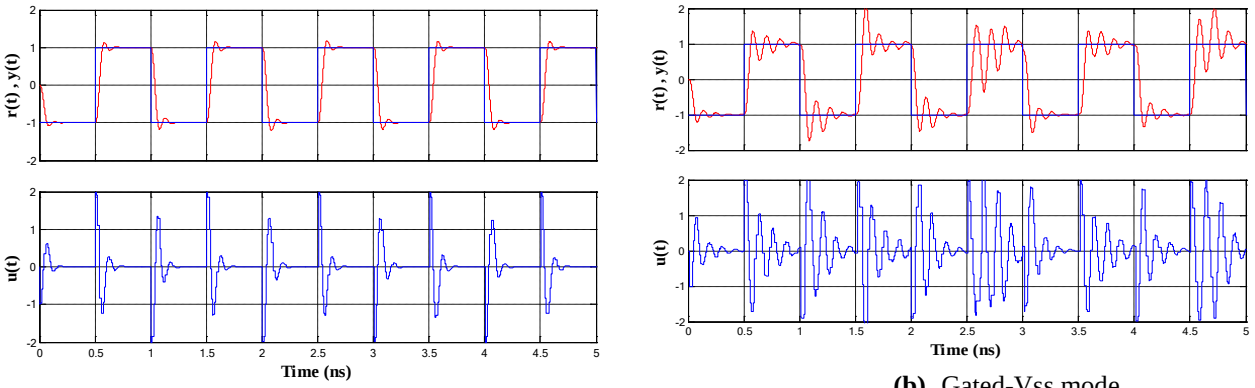
(a) Drowsy mode. **(b)** Gated-Vss mode.

Fig. 36: Networked RTECS Performance Under Low-leakage L-1 D-Cache.



(a) Drowsy mode. **(b)** Gated-Vss mode.

Fig. 37: Networked RTECS Performance Under Low-leakage L-1 I-Cache.



(a) Drowsy mode. **(b)** Gated-Vss mode.

Fig. 38: Networked RTECS Performance Under Low-leakage L-2 U-Cache.

Finally, Table 12 summarizes and compares Drowsy cache and Gated-Vss techniques based on power savings, time-predictability and control performance. In other words, it compares the two techniques based on the best results each technique has achieved under each of the aforementioned parameters. In this table, the power saving column gives the largest net power saving achieved by the corresponding technique among all possible values of LPI, the DOP and LoS columns quantify the time-predictability of the processor in presence of leakage-saving techniques and the control performance describes the performance of single-task, multi-task and networked RTECSs under every combination of leakage-saving technique and the cache level on which this technique is applied.

Table 12: Trade-offs Comparison between Drowsy Cache and Gated-Vss Techniques.

Technique	Cache level	Parameter			
		Power Saving (%)	DoP (%)	LoS (%)	Control Performance
Drowsy Cache	L-1 D-Cache	33.61	93.05	0	Good
	L-1 I-Cache	32.23	66.39	98	Good
	L-2 U-Cache	50.21	80.30	13.68	Good
Gated-Vss	L-1 D-Cache	53.64	19.72	100	Poor
	L-1 I-Cache	15.68	5.42	100	Poor
	L-2 U-Cache	49.93	31.39	100	poor

Several observations can be made based on table 12. First, applying the Gated-Vss technique on L-1 D-Cache has achieved the highest possible net power savings as compared to all other alternatives. However, this scenario falls short under other parameters; it achieves low time-predictability for

both hard and soft real-time systems and leads to a poor control performance when applied in an RTECS. Second, while applying the Drowsy cache technique on the L-1 D-Cache leads to relatively lower power savings, it has outperformed the Gated-Vss technique in terms of time-predictability and control performance. In fact, this scenario has achieved the best time-predictability i.e. DoP and LoS values among other design alternatives. Moreover, it has resulted in good control performance in the underlying RTECS. Third, applying the Gated-Vss technique on the L-1 I-Cache achieves the lowest value of power savings, the lowest DOP value, a high LoS value and a poor control performance. Hence, this scenario does not provide a viable option for low-leakage RTS design. On the other hand, employing a Drowsy L-1 I-Cache can be a suitable design option for systems with non-stringent power constraints and non-critical timing constraints; this scenario has achieved a relatively low power savings and a moderate time-predictability with a good control performance. Fourth, while applying the Gated-Vss technique has achieved relatively high power savings, it has low time-predictability (low DoP value and high LoS value) besides leading to a poor control performance. Hence, this technique does not provide a suitable design alternative for low-leakage real-time systems. On the other hand, using a Drowsy L-2 U-Cache can achieve high power savings, relatively high time predictability and good control performance. Therefore, this scenario can be considered as an effective design choice for low-leakage real-time embedded systems. In summary, our results indicate the using a Drowsy L-1 D-Cache or a Drowsy L-2 U-Cache represent the most suitable design alternatives for low-leakage real-time embedded systems. Comparing these two alternatives, it can be observed that a Drowsy L-2 U-Cache would normally achieve higher power savings but a Drowsy L-1 D-Cache can achieve higher time-predictability. Therefore, a Drowsy L-2 U-Cache can be a practical option for systems with stringent power constraints and softer timing requirements. However, a Drowsy L-1 D-Cache is the most feasible leakage-saving technique for systems with more critical timing requirements.

5 Conclusion and Future Work

This paper has extensively tackled the design of low-leakage cache hierarchy for embedded real-time systems with a single core processor. A multidisciplinary research methodology has be applied to assess and compare state-preserving and state-destroying leakage-saving mechanisms based on their power-saving capability and their impact on the time-predictability of the underlying processor. Our results have shown that applying a state-preserving technique on particular cache level represent the most feasible design alternative in which reasonable tradeoff between power-savings and time-predictability can be attained. However, the decision, of which cache level should be put in low-leakage mode, depends the criticality of timing requirements and power constraints.

As a future research, we will study the impact of leakage-saving mechanisms on time-predictability and performance of multi-core real-time systems especially those with heterogeneous architectures where each core may apply a different leakage-saving mechanism.

References:

[1] J. Liu, Real-Time Systems, Prentice Hall PTR, Upper Saddle River, NJ, USA, 2000.

[2] T. Noergaard, Embedded Systems Architecture: A Comprehensive Guide for Engineers and Programmers, second edition, Elsevier Inc., 2012.

[3] J. L. Hennessy and D. A. Patterson, Computer Architecture: A Quantitative Approach, Fifth Edition, the Morgan Kaufmann Series in Computer Architecture and Design, 2011.

[4] H. Abouja, Real-Time Systems Performance Improvement with Multi-Level Cache Memory, In Proc. Of the Canadian Conference on Electrical and Computer Engineering, 2006, pp. 78-81.

[5] L. Thiele and R. Wilhelm, Design for Timing Predictability, Real-Time Systems, Vol. 28, 2004, pp. 157-177.

[6] M. Schoeberl, Is time predictability quantifiable?, In Proc. Of the International Conference on Embedded Computer Systems, 2012, pp. 333-338.

[7] M. Lv, N. Guan, Y. Zhang and Q. Deng, A Survey of WCET Analysis of Real-Time Operating Systems, In Proc. Of the International Conference on Embedded Software and Systems, 2009, pp. 65-72.

[8] Y. Dinag and W. Zhang, Bounding the Worst-Case Execution Time of Static NUCA Caches, In Proc. of the 33rd IEEE International Performance Computing and Communications Conference (IPCCC), Dec, 2014, pp. 1181-1184.

[9] S. Chattopadhyay, C.L. Kee, A. Roychoudhury and T. Kelter, A Unified WCET Analysis Framework for Multi-core Platforms, In Proc. Of the IEEE 18th Real-Time and Embedded Technology and Applications Symposium (RTAS), 2012, pp. 99-108.

[10] R. Wilhelm, D. Grund, J. Reineke, M. Schlickling, M. Pister and C Ferdinand, ,Memory Hierarchies, Pipelines, and Buses for Future Architectures in Time-Critical Embedded Systems, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 28, Issue: 7, 2009, pp. 966 – 978.

[11] T. Hattori, Challenges for Low-power Embedded SOC's, In Proc. of the International Symposium on VLSI Design, Automation and Test, 2007, pp. 1-4.

[12] D. Vilcu, Real time scheduling and CPU power consumption in embedded systems, In Proc. of the IEEE International Conference on Automation, Quality and Testing, Robotics, 2008, pp. 261-266.

[13] M. Kondo, H. Kobyashi, R. Sakamoto and M. Wada, Design and evaluation of fine-grained power-gating for embedded microprocessors, In. Proc. of Design, Automation and Test in Europe Conference and Exhibition,2014, pp. 1-6.

[14] M. Z. Hasan and M. Bird, Energy reductions for embedded processors in reconfigurable hardware, In Proc. of the IEEE International Conference on Electro/Information Technology (EIT), 2011, pp. 1-8.

[15] M. Alipour, H. Taghdisi and S. H. Sadeghzadeh, Multi objective design space exploration of cache for embedded applications, In. Proc. of the 25th IEEE Canadian Conference on Electrical and Computer Engineering, 2012, pp.1-4.

[16] Y. Li, D. Parikh, Y. Zhang and K. Sankaranarayanan, State-preserving vs. non-state-preserving leakage control in caches, In Proc. of the Design, Automation and Test in Europe Conference and Exhibition, 2004, pp. 22-27.

[17] M. D. Powell, S.-H Yang, B. Falsafi, K. Roy, and T. N. Vijaykumar, Gated–Vdd: a circuit technique to reduce leakage in deep-submicron cache memories. In Proc. of the International Symposium on Low Power Electronics and Design, 2000, pp. 90-95.

[18] K. Flautner, N. S. Kim, S. Martin and D. Blaauw, Drowsy caches: simple techniques for reducing leakage power, In Proc. of the 29th Annual International Symposium on Computer Architecture, 2002, pp. 148-157.

[19] Y. Lu, Approximation Techniques for Timing Analysis of Complex Real-Time Embedded Systems, Licentiate Thesis, School of Innovation, Design and Engineering, Mälardalen University, Sweden, 2010.

[20] Y. Lu, T. Nolte, I. Bate, L. Cucu-Grosjean, A New Way about using Statistical Analysis of Worst-Case Execution Times, ACM SIGBED Review, Vo. 8, Issue 3,2011, pp. 11-14.

[21] K. Patil, K. Seth and F. Muller, Compositional static instruction cache simulation, In. Proc. of the 2004 ACM SIGPLAN/SIBBED Conference on Languages, Compliers and Tools for Embedded Systems, 2004, pp. 136-145.

[22] S. Byhlin, A. Ermedahl, ; J. Gustafsson, B. Lisper, Applying static WCET analysis to automotive communication software, In Proc. Of the 17th Euromicro Conference on Real-Time Systems, 2005, pp. 249-258.

[23] J. Yan and W. Zhang, Analyzing the Worst-Case Execution Time for Instruction Caches With Prefetching, ACM Transactions on Embedded Computer Systems (TECS), Vol. 8, No. 1, Article 7, December 2008.

[24] I. Wenzel, R. Kirner, B. Rieder and P. P. Puschner, Measurement-based timing analysis, In Proc. of the 3rd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation, 2008, pp. 430–444.

[25] L. Kong and J. Jiang, A Safe Measurement-base Worst-case Execution Time Estimation Using Automatic Test-data Generation, In Proc. of the IEEE 16th Pacific Rim International Symposium on Dependable Computing, 2010, pp. 245-246.

[26] A. Marref and A. Betts, Accurate Measurement-Based WCET Analysis in the Absence of Source and Binary Code, In Proc. of the 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2011, pp. 127-135.

[27] S. Bunte, M. Zolda, M. Tautschnig and R. Kirner, Improving the Confidence in Measurement-Based Timing Analysis, In Proc. of the 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2011, pp. 144-151.

[28] A. Betts, A. Donaldson, Estimating the WCET of GPU-Accelerated Applications Using Hybrid Analysis, In Proc. Of the 25th Euromicro Conference on Real-Time Systems (ECRTS), 2013, pp. 193-202. .

[29] S. Bygde, Static WCET Analysis Based On Abstract Interpretation and Counting of Elements, Licentiate Thesis, School of Innovation, Design and Engineering, Mälardalen University, Sweden, 2010.

[30] S. Edgar and A. Burns, Statistical Analysis of WCET for Scheduling, In Proc. of the 22nd IEEE Real-Time Systems Symposim,2001,pp. 215-224.

[31] J. Hansen, S. Hissam and G. A. Moreno, Statistical-based WCET Estimation and Validation, In. Proc. of the 9th International Workshop on Worst-case Execution Time Analysis, WCET, 2009.

[32] N. Hillary and K. Madsen, You Can't Control what you Can't Measure, OR Why it's Close to Impossible to Guarantee Real-time Software Performance on a CPU with on-chip cache, In Proc. of the 2nd International Workshop on WCET Analysis, 2002.

[33] J. Beirlan, Y. Geogebeur, J. Segers and J. Teugels, Statistics of Extremes: Theory and Applications, Wiley Press, 2004.

[34] H. Aghababa, A. Khosropour, A. Afzali-kusha and B. Forouzandeh, Statistical estimation of leakage power dissipation in nano-scale complementary metal oxide semiconductor digital circuits using generalised extreme value distribution, IET Circuits, Devices and Systems, Vol. 6, Issue 5, 2012,pp. 273-278.

[35] N.S. Kim, T. Austin, D. Blaauw, T. Mudge, K. Flautner, J.S. Hu,M.J. Irwin, M. Kandemir, and V. Narayanan, Leakage Current: Moore's Law Meets Static Power, Computer, vol. 36, no. 12, 2003, pp. 68-75.

[36] A. Cervin, Towards the Integration of Control and Real-Time Scheduling Design, Licentiate Thesis, Department of Automatic Control, Lund Institute of Technology,2000.

[37] F. Xia and Y. Sun, Control-Scheduling Codesign: A Perspective on Integrating Control and Computing, Dynamic of Continous, Discrete and Impulsive Systems- Series B, Vol. 13, no. S1, 2006, pp. 1352-1358.

[38] A. Cervin and J. Eker, Control-scheduling codesign of real-time systems: The control server approach, Journal of Embedded Computing, Vol. 1, Issue 2, 2005, pp. 209-224.

[39] S. Dai, H. Lin, S. Sam and S. S. Ge, Scheduling-and-Control Codesign for a Collection of Networked Control Systems With Uncertain Delays, IEEE Transactions on Control Systems Technolgy, Vol. 18, No. 1, 2010,pp. 66-78.

[40] R. A. Gupta and M. Chow, Netwroked Control Systems: Overview and Research Trends, IEEE Transactions on Industrial Electronis, Vol. 57, No. 7, 2010, pp. 2527-2535.

[41] Y. Wang and L. He, Analysis and Simulation of Networked Control Systems Delay Characteristics Based on TrueTime, Computer Modeling and New Technologies, Vol. 17, No. 4, 2013, pp. 210-216.

[42] K. Nishanth and P. S. S Sashank, Real-Time Computer Control of Disctribued Control Systems Using TrueTime Toolbox in Matlab, International Journal of Advanced Science, Engineering and Technology, Vol. 3, Issue 1,2014, pp. 8-13.

[43] O. M. M. Vall, Compensation of Time Delay Acting in Networked Control Systems Using Smith Predictor and Pade Approximation, International Journal of Information Technology and Computer Science, Vol. 15, Issue 1, 2014, pp. 9-15.

[44] Y. Zhang, D. Parikh, K. Sankaranarayanan, K. Sakdron and M. Stan, HotLeakage: A Temperature-Aware Model of Subthreshold and Gate Leakage for

Architects, Tech. Report CS-2003-05, University of Virginia, Dept. of Computer Science,2003.

[45] R. E. Kessler, E. J. MclLellan and D. A. Webb, The Alpha 21264 Microprocessor Architecture, In Proc. of the International Symposium on Low-Power Electronics and Design, 1998, pp. 293-298.

[46] www.spec.org.

[47] F. J. Cazolrla, T Vardanega, E. Quinones and J. Abella, Upper-bounding Program Execution Time with Extreme Value Theory, 13th International Workshop on Worst-Case Execution Time Analysis (WCET 2013), pp. 61-70, 2013.

[48] R. Placket and J. Burman, The Design of Optimum Multifactorial Experiments, Biometrica, Vol. 33, Issue 4, 1956, pp. 305-325.

[49] J. Yi and D. Lilja, Effects of Processor Parameter Selection on Simulation Results, MSI Report 2002/16, 2002.

[50] www.minitab.com.

[51] J. Banks, J. S. Carson II, B. Nelson, and D. M. Nicol, Discrete-Event System Simulation, Pearson Education,2009.

[52] www.mathworks.com

[53] R. E. Walpole, R. H. Mayers, S. L. Mayers and K. Ye, Probability and Statisctics for Engineers and Scientists, Prentice Hall, 2011.

[54] A. Cervin, D. Henriksson, B. Lincoln, J. Eker and K. Årzén, How Does Control Timing Affect Performance? Analysis and Simulation of Timing Using Jitterbug and TrueTime." *IEEE Control Systems Magazine*, Vol 23, No. 3, pp. 16–30, June 2003.

[55] Jan Gustafsson, Adam Betts, Andreas Ermedahl, and Björn Lisper. The Mälardalen WCET benchmarks – past, present and future. In Björn Lisper, editor, Proc. 10th International Workshop on Worst-Case Execution Time Analysis (WCET'2010), pages 137–147, Brussels, Belgium, July 2010. OCG.

[56] K. J. Astrom and B. Wittenmark, Computer-Controlled Systems: Theory and Design, 3rd edition, Prentice Hall, 1996.