공학석사학위논문

# Real-Time Recovery from Soft-Errors on a Modern ECU Board

# 최신 ECU보드를 활용하여 소프트에러들을 실시간 복구하는 기법

**2020년 08월**


**서울대학교 대학원**

**컴퓨터공학부**

**정 재 환**

# Real-Time Recovery from Soft-Errors on a Modern ECU Board

# 최신 ECU보드를 활용하여 소프트에러들을 실시간 복구하는 기법

지도교수 이 창 건

이 논문을 공학석사 학위논문으로 제출함

2020년 06월

서울대학교 대학원

컴퓨터공학부

정 재 환

정재환의 공학석사 학위논문을 인준함

2020년 08월

위 원 장 하 순 회 (인)

부위원장 이 창 건 (인)

위 원 김 지 홍 (인)

## Abstract

# Real-Time Recovery from Soft-Errors on a Modern ECU Board

Jaehwan Jeong
Department of Computer Science and Engineering
The Graduate School
Seoul National University

This dissertation presents the fault-tolerant real-time scheduling using dynamic mode switch support of modern ECU hardware. This dissertation first describes the optimal capacity of the Periodic Resource which contains harmonic periodic task set using the exact time supply function. We show that the optimal capacity can be represented as sum of the each individual utilization of the task in the harmonic periodic task set for both normal state(i.e. no faults) and faulty state. Then, this dissertation proposes non-critical task overlapping technique by only using the idle time intervals of the Periodic Resource in order to overlap the non-critical tasks which ensures no additional capacity increase. Finally, this dissertation proposes the basic form of the Periodic Resources in order to efficiently use the dynamic mode switch support. Next, we also proposes the bin-packing heuristic algorithm that considers both making sub-taskset as a one Periodic Resource and Periodic Resource wide bin-packing which has the pseudo-polynomial time complexity. Experimental results show that the proposed algorithm performs better than the traditional partitioned fixed-priority scheduling

approach and partitioned mixed-criticality scheduling approach. Also, the achievement is made up to 18% in terms of the total needed cores compared to traditional partitioned fixed-priority approach for making the given input task set schedulable.

**keywords** : Periodic Resource Model, Fault-Tolerant Real-Time Scheduling

**Student Number** : 2018-26716

# Contents

# List of Figures

# List of Tables

# 1 Introduction

## 1.1 Motivation and Objective

Continuous technology scaling raises the susceptibility of the embedded system chip to the soft-errors [2]. The soft errors known as transient faults, are caused by a single bit-flip(i.e. Single Event Upset) or multiple bit-flips (i.e. Multiple Event Upset) which are induced by the high-energy radiation particle strikes [3]. Especially, for the real-time embedded systems, a single deadline miss caused by the soft errors can lead to the catastrophic consequences such as economic disasters or human life losses. Therefore, it is very important to ensure that the all deadlines can be met even when such kind of faults are present. There are several well-known techniques to handle such kind of faults, e.g. re-execution [4], checkpoint-based roll-back recovery [5], Dual Modular Redundancy(DMR), Dual-Core LockStep(DCLS) [6], etc. Especially, lockstep architecture is widely used in the safety-critical system domain due to its outstanding error coverage for tolerating faults [6, 7].

Recently, a new Micro-Controller Unit(MCU) [8] has been producted that can re-configure the system state either by grouping 2 cores as 1 lockstep core(i.e. lockstep mode) or separate them as parallel 2 cores(i.e. performance mode) [9] at run-time(i.e. hardware that has the dynamic mode switch support) in order to ensure the safety for the critical tasks and maximize the parallelism for the non-critical tasks.

Thus, in this paper, we propose a real-time fault-tolerant scheduling that can efficiently use the resource of the target hardware that has the dynamic mode switch support. In order to fully exploit the characteristics of the target hardware, like similar work in [1], we use Periodic Resource Model in order to ensure the temporal isola-

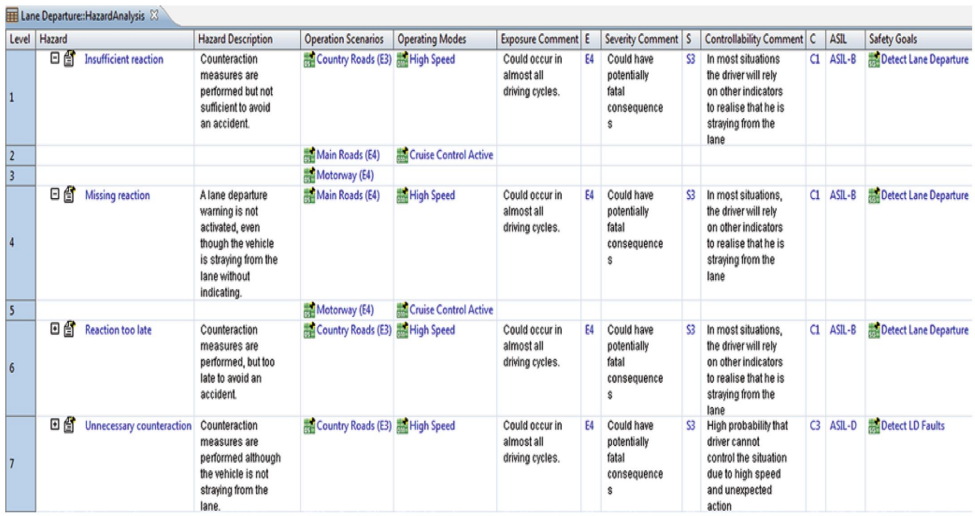
tion between 2 different modes(i.e. lockstep mode and performance mode). We aim to minimize the total number of needed cores while keeping the safety(i.e. no deadline miss) for the critical-tasks even when faults are present and also guaranteeing the non-critical tasks schedulable only when the system operation state is normal(i.e. no faults).

## 1.2 Approach

Since the brand-new MCU [8] is targetting for the automotive sector, in this paper, we propose several approaches which can be especially practical in the automotive domain. Our basic approaches are not precisely investigated in the previous real-time fault-tolerant scheduling literature [10, 11, 12, 13, 14, 15, 16, 17, 18]. These are described as follows:

- **Different Fault-Tolerance Requirements**

  According to automotive functional safety standard ISO-26262 [19], Automotive Safety-Integrity Levels(ASIL) are divided into 4 levels. From ASIL-D(highest criticality), to ASIL-A(lowest criticality). Note that the Quality-Management(QM) level is considered as non-critical tasks. For the Single Point of Failure(SPoF) [19], it requires 90% error coverage for ASIL-B, 97% for ASIL-C, 99% for ASIL-D, respectively. In practice, since the fault-detection method(e.g. DLCS, Control-Flow Check(CFC) [20], Valid Range Check [21]) of the individual task is usually determined by its ASIL-level, we assume that a certain method $M$ ensures the detection coverage $c_{guarantee}$ if it is used for the task $\tau$. Also,for the task $\tau$, it has to satisfy its own minimum detection coverage requirement $c_{req}$ depends on the criticality of the task.



Lane Departure::HazardAnalysis

| Level | Hazard | Hazard Description | Operation Scenarios | Operating Modes | Exposure Comment | E | Severity Comment | S | Controllability Comment | C | ASIL | Safety Goals |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Insufficient reaction | Counteraction measures are performed but not sufficient to avoid an accident. | Country Roads (E3) | High Speed | Could occur in almost all driving cycles. | E4 | Could have potentially fatal consequences | S3 | In most situations the driver will rely on other indicators to realise that he is straying from the lane | C1 | ASIL-B | Detect Lane Departure |
| 2 | | | Main Roads (E4) | Cruise Control Active | | | | | | | | |
| 3 | | | Motorway (E4) | | | | | | | | | |
| 4 | Missing reaction | A lane departure warning is not activated, even though the vehicle is straying from the lane without indicating. | Main Roads (E4) | High Speed | Could occur in almost all driving cycles. | E4 | Could have potentially fatal consequences | S3 | In most situations, the driver will rely on other indicators to realise that he is straying from the lane | C1 | ASIL-B | Detect Lane Departure |
| 5 | | | Motorway (E4) | Cruise Control Active | | | | | | | | |
| 6 | Reaction too late | Counteraction measures are performed, but too late to avoid an accident. | Country Roads (E3) | High Speed | Could occur in almost all driving cycles. | E4 | Could have potentially fatal consequences | S3 | In most situations, the driver will rely on other indicators to realise that he is straying from the lane | C1 | ASIL-B | Detect Lane Departure |
| 7 | Unnecessary counteraction | Counteraction measures are performed although the vehicle is not straying from the lane. | Country Roads (E3) | High Speed | Could occur in almost all driving cycles. | E4 | Could have potentially fatal consequences | S3 | High probability that driver cannot control the situation due to high speed and unexpected action | C3 | ASIL-D | Detect LD Faults |

Figure 1: ASIL-determiniation of the lane-depature task

- **Different Fault Detection Coverage & Overhead**

  Common methods for detecting the transient faults in automotive domain are DCLS, CFC [20], Valid Range Check. Especially, SW-based detection method(e.g. CFC, Valid range check) has different detection coverage depend on the target task and also each method has different overhead upon different tasks [22, 23]. Therefore, we assume every SW-based detection method can have different detection coverage and overhead[22, 23].

- **Varying Criticalities depends on the Physical Environments**

  According to the Automotive Functional Safety Standard ISO-26262 [19], The determination of the ASIL-level for each task is conducted starting with the Hazard Analysis and Risk Assessment(HARA) which is the product of exposure, controllability, and severity by using the predetermined table. Figure 1 shows the typical example of the ASIL determination of the lane depature task.

As we can find out in the Figure 1, dangerous situations only occur at some specific conditions(e.g. at country roads with high speed, unnecessary countereaction produces ASIL-D). Also, we can see that the other dangerous situations incur only ASIL-B. However, Due to the only one dangerous situation when unnecessary countereaction occurs in country roads with the high speed. The ASIL of the lane depature task in Figure 1 is determined as ASIL-D. When the ASIL of the task is ASIL-D, it is highly likely to use the detection method as DCLS since it requires 99% detection coverage. However, in other cases, detection coverage can be met even if we use the other SW-based detection method such as CFC(e.g. assume that CFC on lane departure task guarantees 91%) which is better resource usage scheme compared to DCLS. From this intuition, we have the brand-new MCU which can switch the mode either lockstep mode or performance mode at run-time. Hence, in our system model, we assume that there are many operation states and the criticality of the task can be different depends on the given state. In other words, in each state, all we need to do is that assigning appropriate detection method that can ensure the detection coverage which is higher or equal to the minimum required coverage of the task in that state. For example, the suspension control task, in state A(intersection), it has medium criticality level so we can assign the detection method as CFC or DCLS. On the contrary, in state B(highway), it has the highest criticality level so we can assign the detection method only as DCLS.

- **Possible periods of the tasks**

  Kramer et al[24] showed that the period distribution of the typical automotive tasks are not purely random.(e.g. $\{1,2,5,10,20,50,100,200,1000ms\}$) Also, the

other works related to automotive applications has similar period distributions [25]. Therefore, we assume that the given task set is harmonitizable(more details will be discussed in Section 2). Since we know the optimal capacity of the Periodic Resource that contains harmonic periodic task set, by using this property, we propose the bin-packing heuristic algorithm.

Considering those approaches, we reflect them into our system model and the problem description. And then, we show the optimal capacity of the periodic resource that contains harmonic task set for both normal operation state(i.e. no faults) or faulty operation state. Next, we propose the overlapping technique that can be used for the non-critical tasks while the system operation state is normal(i.e. no faults). Note that this is already widely used in the mixed-criticality scheduling literature. Finally, we propose the Periodic Resource wide bin-packing heuristic algorithm and show the experimental results. Our main contributions are follows:

**Our Contributions**:

- We show the optimal capacity of the Periodic Resource that contains the periodic harmonic task set for both normal operation state(i.e. no faults) and faulty operation state. The optimal capacity can be represented using the each utilization of individual periodic task inside the Periodic Resource.

- We propose the bin-packing heuristic algorithm in the hierarchical partitioned fixed-priority preemptive scheduling domain. To the best of our knowledge, This is the first attempt from making sub-taskset as a one Periodic Resource to partitioning each Periodic Resource to the given set of cores.

## 1.3 Organization

This paper is organized as follows. Section 2 formally defines our problem. Section 3 describes our schedulability analysis and the optimal capacity of the Periodic Resource. Then, Section 4 explains our proposed algorithm. Section 5 reports our experiment results. Section 6 presents related works. Finally, Section 7 concludes the paper.

## 2 System Model

We consider a system with $n$ periodic tasks and $m$ states and $l$ error detection methods. Each periodic task is denoted by $\tau_i$ $(1 \leq i \leq n)$ and the set of $n$ periodic tasks is denoted by $\Gamma$ as follows: $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$. Each task $\tau_i$ is represented as a 3-tuple, i.e., $\tau_i$=($T_i$, $D_i$, $C_i$) where $T_i$ is the period, $D_i$ is the relative deadline, and $C_i$ is the worst-case execution time(WCET). If $D_i = T_i$ for all tasks the task set has implicit deadlines. We assume that the given task set $\Gamma$ has implicit deadlines. Each state is denoted by $s_j$ $(1 \leq j \leq m)$ and the set of $m$ states is denoted by $\mathbb{S}$ as follows : $\mathbb{S} = \{s_1, s_2, ..., s_m\}$. Each fault-detection method is denoted by $d_k$ $(1 \leq k \leq l)$ and the set of $l$ detection methods is denoted by $\Lambda$ as follows: $\Lambda = \{d_1, d_2, ..., d_l\}$. Also, a system contains the overhead table denoted by $\mathbb{O}(n \times l$ array$)$ and the detection coverage table denoted by $\mathbb{C}ov(n \times l$ array$)$. Each element denoted by $o_{i,k}$ in overhead table $\mathbb{O}$ represents the overhead value when task $\tau_i$ uses a detection method as $d_k$. Each element denoted by $\theta_{i,k}$ in detection coverage table $\mathbb{C}ov$ represents the error detection coverage(e.g. detection ratio) when the task $\tau_i$ uses a detection method as $d_k$. For every state $s_j$, system contains minimum requirement table $\mathbb{R}_j = \{r_{j,1}, ..., r_{j,n}\}$. Each element denoted by $r_{j,i}$ in $\mathbb{R}_j$ represents the minimum required error coverage of the task $\tau_i$ in state $s_j$. we consider if $r_{j,i} = 0$, then the task $\tau_i$ is non-critical. Also, if the task $\tau_i$ is non-critical, then $\forall j \ \{r_{j,i} = 0 \mid 1 \leq j \leq m\}$ holds(Informally, if the task is non-critical, then it is always non-critical regardless of the state information).

**Definition 1.** *Feasible Task-to-Detection Method Mapping* $M_{i,j,k} = \{(i, j, k) \mid \tau_i \in \Gamma, s_j \in \mathbb{S}, d_k \in \Lambda\}$

*If there exists a feasible mapping* $(i, j, k) \in M_{i,j,k}$, *then* $\theta_{i,k} \geq r_{j,i}$ *s.t.* $\theta_{i,k} \in \mathbb{O}, r_{j,i} \in$

$\mathbb{C}ov, s_j \in \mathbb{S}$.

Informally, this feasible mapping $M$ means that there exists a detection method $d_{i,k}$ for task $\tau_i$ that can bring detection coverage value $\theta_{i,k}$ over than the minimum required detection coverage $r_{j,i}$ in state $s_j$. We assume that if the task is non-critical, then the required detection coverage is 0 in every state. (i.e. given non-critical task $\tau_i$, it holds $\forall s_j \in \mathbb{S}, r_{i,j} = 0$ )

**Definition 2.** *Feasible Task-to-Core Mapping* $M_{i,j,k,l} = \{(i,j,k,l) \mid \tau_i \in \Gamma, s_j \in \mathbb{S}, d_k \in \Lambda, c_l \in \mathbb{C}^j\}$

*If there exists a feasible task-to-core mapping* $(i,j,k,l) \in M_{i,j,k,l}$*, then there exists a feasible task-to-detection mapping* $M_{i,j,k}$*. Also, for the task* $\tau_i$ *with the modified WCET as* $\bar{C}_i = C_i(1+o_{i,k})$*, the task* $\tau_i$ *is schedulable on core* $c_l$ *and* $c_{l+1}$ *if* $d_{i,k} = \{DCLS\}$*. the task* $\tau_i$ *is schedulable on core* $c_l$ *if* $d_{i,k} = \Lambda - \{DCLS\}$

Note that we constrained the lockstep detection method only for using dual-cores(i.e. DCLS). Informally, this feasible task-to-core mapping means that there exists a feasible task-to-detection mapping $M_{i,j,k}$ as described in Definition 1. Moreover, the task $\tau_i$ with the modified WCET as $\bar{C}_i = C_i(1+o_{i,k})$ is schedulable on core $c_l$ and $c_{l+1}$ if the detection method $d_{i,k}$ is lockstep. Otherwise, $\tau_i$ is schedulable on core $c_l$. Also, note that if the task $\tau_i$ is non-critical, then $\bar{C}_i = C_i$. We assume that maximum 1 fault occurs during the HyperPeriod(i.e. $lcm(T_1, T_2, ..., T_n)$) of the task set. Also, for the schedulability condition of the task $\tau_i$ in the core $c_l$, we divide them into 2 categories. If the task $\tau_i$ is critical task, then the schedulability condition would require real-time guarantee in the faulty system operation. Contrarily, if the task $\tau_i$ is non-critical task, then the schedulability condition would require the real-

time guarantee only in the normal system operation(i.e. no faults). More details about the schedulability conditions are described in Section 3 and Section 4.

**Definition 3.** *Feasible Mapping* $M = \{M_{i,j,k,l} \mid \forall \tau_i \in \Gamma, \forall s_j \in \mathbb{S}, \exists M_{i,j,k,l}$ *s.t.* $d_{i,k} \in \Lambda, c_l \in \mathbb{C}\}$

*If there exists a feasible mapping M, then for every task* $\tau_i$*, for every state* $s_j$*, there exists at least one feasible task-to-core mapping* $M_{i,j,k,l}$

Informally, Feasible mapping $M$ represents that in every state $s_j$, for every task $\tau_i$, there exists at least one detection method $d_{i,k}$ that can ensure detection coverage over the minimum required coverage $r_{i,j}$. Also, the task must be schedulable on the core $c_l$ and $c_{l+1}$ if the detection method is DCLS. Otherwise, the task must be schedulable on the core $c_l$. Recall that if the the task is critical, it has to be schedulable even when the faults are present. Otherwise(i.e. the task $\tau_i$ is non-critical), it has to be schedulable only when the system operation is normal(i.e. no faults).

**Problem Definition:** Given input set as $\Gamma$(periodic task set), $\mathbb{S}$(state set), $\Lambda$(detection method set), $\mathbb{O}$(overhead table), $\mathbb{C}ov$(detection coverage table), and $\mathbb{R}_j$(minimum required coverage table) in each state $s_j \in \mathbb{S}$, for every state $s_j \in \mathbb{S}$, find a feasible mapping $M$ that minimizes the size of $\mathbb{C}^j$(core set in state $s_j$). The final number of needed cores is given as $\max_{j=1,2,\ldots,m} |\mathbb{C}^j|$ (i.e. maximum number of the cores for all state $s_j \in \mathbb{S}$).

# 3 Schedulability Analysis

We use Periodic Resource Model [26] in order to properly use the lockstep mode and performance mode. Cirinei et al [1] proposed a flexible scheme that finds the period $P$(i.e. period of the Periodic Resource $\Pi$) and the respective minimum budget $Q$(i.e. the amount of time supply for the Periodic Resource $\Pi$ per period $P$) to the period $P$ of the Periodic Resource $\Pi$ that makes the periodic task set $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$ in Periodic Resource $\Pi$ schedulable using lockstep/performance mode architecture. They proposed 2 approaches one for minimizing the needed bandwidth for the given input set, the other for maximizing the slack bandwidth also for the given input set. They showed that their work is the first attempt to dynamically reconfigure the hardware architecture in the different operating modes. In this paper, we use [1] as a baseline method and aim to minimize the needed bandwidth(i.e. maximize the slack bandwidth) for given Periodic Resource $\Pi$ that contains harmonic periodic task set $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$. For the sake of the consistency to our goal, we assume that the mode change overhead between lockstep mode and performance mode is negligible.

## 3.1 Background

The Periodic Resource denoted by $\Pi$ that contains the periodic task set $\Gamma = \{\tau_1, ..., \tau_n\}$ has a period denoted by $P$ and the budget denoted by $Q$(i.e. the amount of time supply per every time $P$). For the sake of simplicity, we denote the Periodic Resource as $\Pi(P, Q)$ and the task set as $\Gamma = \{\tau_1, ..., \tau_n\}$.

**Theorem 1.** ***(Theorem 1 in [1])***

*A task set* $\Gamma = \{\tau_1, ..., \tau_n\}$*(*$\tau_1$ *for the highest priority,* $\tau_n$ *for the lowest priority) with*

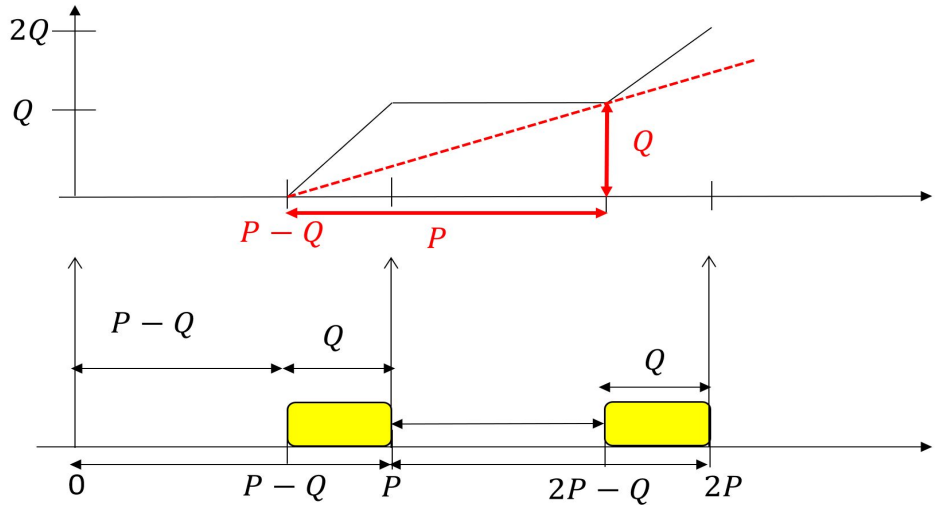



Figure 2: Lower bound of the supply function for the Periodic Resource $\Pi(P,Q)$

*the Periodic Resource* $\Pi(P,Q)$ *is schedulable by Fixed Priorities*

*if* $\exists Q \leq P$ *and* $Q$ *satisfies following the equation :*

$$Q \geq \max_{\tau_i \in \Gamma} \min_{t \in SchedP_i} \frac{\sqrt{(t-P)^2 + 4PW_i(t)} - (t-P)}{2} \tag{1}$$

*where*

$$\begin{cases} SchedP_i(t) = \{t\} \\ SchedP_i(t) = SchedP_{i-1}(\lfloor \frac{t}{T_i} \rfloor T_i) \cup P_{i-1}(t) \end{cases} \tag{2}$$

*and*

$$W_i(t) = C_i + \sum_{j=1}^{i-1} \lceil \frac{t}{T_j} \rceil C_j \tag{3}$$

*Proof.* Proof of Lemma 1 and Theorem 1 in [1] □

Since the whole proof of the Theorem 1 is slightly complex, we briefly describe the concept of the Theorem 1. First, it uses the Time-Demand Analysis [27]. Eq. (3) de-

notes the worst-case processor demand of task $\tau_i$ in given task set $\Gamma$. In [27], they also describe the scheduling point of each $\tau_i$. After then, Bini et al [28] reduce the scheduling point region of the task $\tau_i$ such as Eq. (2). Next, we introduce the lower bound supply function of the Periodic Resource by Figure 2. In [1], they first proposed the methodology in order to efficiently use the lockstep/performance mode. They fixed the period of Periodic Resources as one period $P$ because lockstep mode requires synchronization processes. In that case, suppose that Periodic Resource $\Pi(P,Q)$ contains the periodic task set $\Gamma = \{\tau_1, ..., \tau_n\}$, then for the each task $\tau_i$, worst-case supply occurs when the time supply is guaranteed at the end of the period $P$, Formally, it can be represented as follows(**Lemma 1 in [1]**) :

$$S(t) = \begin{cases} jQ & if \quad t \in [jP, (j+1)P - Q] \\ t - (j+1)(P-Q) & otherwise \end{cases} \quad where \quad j = \lceil \frac{t}{P} \rceil \qquad (4)$$

As we can see in the Figure 2, every time interval from $[jP, (j+1)P - Q]$ gets exactly $jQ$ amount of time supply and the other time interval gets the proportional time amount to $t$ as decribed in Eq. (4). From the Figure 2, we can lower bound this time supply function with the 2 points as $(P-Q, 0)$ and $(2P-Q, Q)$. Formally lower bound supply function $\bar{S}(t)$ can be represented as follows :

$$\bar{S}(t) = \frac{Q}{P}(t - (P-Q)) \qquad (5)$$

Finally we know the period value $P$, the lower bound supply function as Eq (5), and the scheduling point set of task $\tau_i$ as Eq (2), worst-case processor demand of time $t$ as Eq (3). What Eq (1) implies is that for every task $\tau_i \in \Gamma$, we have to check ev-

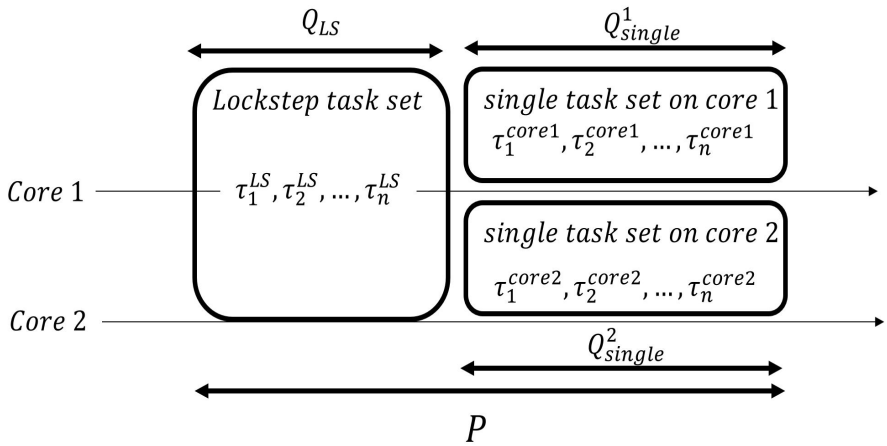



Figure 3: Schedulability test of the Periodic Resources in [1]

ery $SchedP_i$ and take the minimum $Q_{SchedP_i}$ among them. After calculating $Q_{SchedP_i}$ for every task, then we choose maximum $Q_{SchedP_i}$ among them. Therefore, maximum $Q_{SchedP_i}$ ensure that every task is schedulable, and if $Q \leq P$, that means it can supply time supply amount $Q$ every time period $P$. Note that Theorem 1 holds only there exists one Periodic Resource $\Pi(P,Q)$.

Cirinei et al [1] proposed Periodic Resource wide schedulability test as follows. Since lockstep mode requires synchronization process, they fixed the period of every Periodic Resource $\Pi_i$ as certain value $P$. We describe it with the example in Figure 3. In Figure 3, there exists 3 Periodic Resources as $\Pi_{LS}$, $\Pi^1_{single}$ and $\Pi^2_{single}$. Each Periodic Resource contains their task set and by Theorem 1, they obtain the minimum $Q_i$ for each Periodic Resource $\Pi_i$. Then, the schedulability test can be represented as $Q_{LS} + \max(Q^1_{single}, Q^2_{single}) \leq P$. Since Periodic Resource $\Pi^1_{single}$ is mapped to core 1 and $\Pi^2_{single}$ is mapped to core 2, we take the maximum $Q$ value of them. We use the similar approach as [1], However, we constrained the periodic task set with the only harmonic periodic task set in one Periodic Resource $\Pi_i$ and this restriction leads us

not to necessarily use the lower bound supply function. Details are described in the next subsection.

## 3.2 Optimal Capacity Analysis During Normal State

We first propose the optimal capacity of the Periodic Resource $\Pi(P,Q)$ that contains harmonic periodic task set $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$ using the summation of the utilization $u_i$ of the each periodic task $\tau_i$ in $\Gamma$. In [1], they proposed a method regarding how to find a appropriate $P,Q$ of the Periodic Resource $\Pi$. However, if the periodic task set $\Gamma$ is harmonic, then we do not have to struggle with that issue. We present the method for how to find a optimal $P$ and $Q$ that ensures minimum bandwidth for scheduling the given harmonic task set $\Gamma$.

**Theorem 2.** *Given Periodic Resource* $\Pi(P,Q)$ *that contains harmonic periodic task set* $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$*, If the period P of* $\Pi$ *is the common divisor of the periods of the all periodic tasks(i.e.* $P = CD(T_1, ..., T_n)$*), then the exact time supply function* $S(t)$ *of every task* $\tau_i$ *in* $\Pi$ *at every job release point* $t \in \{kT_i \mid k = 1, 2, ...\}$ *is* $\frac{Q}{P}t$

*Proof.* No matter how many other Periodic Resources exist $\Pi_{other_1}$, $\Pi_{other_2}$,....., If the Periodic Resource $\Pi$ is guaranteed to the supply time amount $Q$ until each period $P$, for every task $\tau_i$, let the job release time points of the task $\tau_i$ as $t_i \in \{kT_i \mid k = 1, 2, ...\}$. Since $t_i$ is the multiple of the period $P$, the time supply function of $S(t_i) = \frac{Q}{P}t_i$

□

**Theorem 3.** *Given Periodic Resource* $\Pi$ *that contains harmonic periodic task set* $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$*, the optimal capacity* $A(= \frac{Q}{P})$ *of Periodic Resource* $\Pi(P,Q)$ *is* $\sum_{i=1}^{n} u_i$

*Proof.* From Theorem 1, We find out that selecting the period $P$ as a common divisor of $\{T_1, T_2, ..., T_n\}$ can give the exact time supply function as $S(t) = \frac{Q}{P}t$ for every relase points $t$ of the task $\tau_i$ in $\Gamma$(i.e. no need for finding the lower bound supply function), we fix P as common divisor of $\Gamma$ and prove it using the Time Demand Analysis(TDA) [27].

**Definition 4.** *(Time Demand Analysis [27]). Given periodic task set* $\Gamma = \{\tau_1, ..., \tau_n\}$ *denoting* $\tau_1$ *as the highest priority, and* $\tau_n$ *is the lowest priority.* $\tau_i$ *is schedulable if the following equation holds:*

$$\exists\, t \; s.t. \; W_i(t) = \; C_i + \sum_{j=1}^{i-1} \lceil \frac{t}{T_j} \rceil C_j \leq t, \{t = kT_i \mid k = 1, 2, ..., \lceil \frac{T_i}{T_j} \rceil\} \quad (6)$$

For Periodic Resource, until time $t$ the exact time supply is not $t$, but $S(t)$. So the Eq (6) can be transformed as follows:

$$\exists\, t \; s.t. \; W_i(t) = \; C_i + \sum_{j=1}^{i-1} \lceil \frac{t}{T_j} \rceil C_j \leq S(t), \{t = kT_i \mid k = 1, 2, ..., \lceil \frac{T_i}{T_j} \rceil\} \quad (7)$$

For every task $\tau_i$, we find a Q that there exists a $t \in SchedP_i$ that satisfies Eq (7). For the proof, we use mathematical induction as follows.

- Basic Step : For the task $\tau_1$, there is no higher priority tasks. And the only time point we have to check is $T_1$ by Eq. (2). Therefore $S_1(T_1) = \frac{Q_1}{P} T_1$ and $W_1(T_1) = C_1$. we have to satisfy $S_1(T_1) \geq W_1(T_1)$. (for minimum $Q_1$, $S_1(T_1) = W_1(T_1)$)
  $\frac{Q_1}{P} T_1 = C_1 \iff Q_1 = \frac{C_1}{T_1} P = U_1 P \quad \therefore$ Capacity $A_1 = \frac{Q_1}{P} = \frac{U_1 P}{P} = U_1$.

- Induction Step : Suppose that for the task $\tau_k$, capacity $A_k = \sum_{i=1}^{k} U_i$. This means that $Q_k = P\sum_{i=1}^{k} U_i$ is the minimum budget for scheduling $\tau_1, ..., \tau_k$. We need to show that $A_{k+1} = \sum_{i=1}^{k+1} U_i$. For $\tau_k$, the scheduling point is only at $T_k$. Also, for $\tau_{k+1}$, the scheduling point is only at $T_{k+1} = kT_k$, where $k = \frac{T_{k+1}}{T_k}$ by Eq. (2). Note that this condition holds only when the task set $\Gamma$ is harmonic. The worst-case processor demand of the task $\tau_{k+1}$, $W_{k+1}(kT_k) = kW_k(T_k) + C_{k+1}$ due to the harmonic property of the task set $\Gamma$. The exact time supply function $S_{k+1}(t)$ at $kT_k$ is $S_{k+1}(kT_k) = \frac{(Q_k+\alpha)}{P}(kT_k)$. where $\alpha$ denotes the additional budget for scheduling the task $\tau_{k+1}$. In order to minimize the budget $Q_{k+1}$, we have to set the time supply amount equal to the worst-case processor demand(i.e. $S_{k+1}(kT_k) = W_{k+1}(kT_k)$).

  $\rightarrow S_{k+1}(kT_k) = kW_k(T_k) + C_{k+1} \quad \because W_{k+1}(kT_k) = kW_k(T_k) + C_{k+1}$

  $\rightarrow \frac{(Q_k^N+\alpha)}{P}(kT_k) = kW_k(T_k) + C_{k+1} \quad \because S_{k+1}(kT_k) = \frac{(Q_k+\alpha)}{P}(kT_k)$

  $\rightarrow C_{k+1} = kT_k\frac{\alpha}{P} \qquad \because \frac{Q_k}{P}(kT_k) = kW_k(T_k)$

  $\rightarrow \alpha = P\frac{C_{k+1}}{kT_k} = P\frac{C_{k+1}}{T_{k+1}} = PU_{i+1} \qquad \therefore \; Q_{k+1} = Q_k + PU_{i+1} = P\sum_{i=1}^{k+1} U_i$

  Thus, minimum capacity for the task set $\{\tau_1, ..., \tau_{k+1}\}$, $A_{k+1} = \frac{Q_{k+1}}{P} = \sum_{i=1}^{k+1} U_i$

  □

### 3.3 Optimal Capacity Analysis During Fault State

Next, we propose the optimal capacity of the Periodic Resource $\Pi(P,Q)$ which ensures 1-fault tolerant property(i.e. re-execution can be met before the deadline) when

maximum 1 fault occurs during the HyperPeriod(i.e. $LCM(T_1, T_2, ..., T_n)$) of the given task set $\Gamma$. Note that Theorem 4 can be easily extended to the $k$ faults situation.

**Theorem 4.** *Given Periodic Resource* $\Pi(P,Q)$ *that contains harmonic periodic task set* $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$*, the optimal capacity* $A^F$ *of Periodic Resource* $\Pi$ *for tolerating maximum 1 fault is* $\max_{k=1,2,...,n} ((\sum_{i=1}^{k} u_k) + u_k)$

*Proof.* We prove it by the following equation holds for every $\tau_i \in \Gamma$

$$Q_i^F = \begin{cases} 2Q_i^N & i = 1 \\ \max(Q_{i-1}^F, 2Q_i^N - Q_{i-1}^N) & i \geq 2 \end{cases} \tag{8}$$

$Q_i^N$ denotes the **normal** capacity for scheduling the task set $\{\tau_1, ..., \tau_i\}$ which can be calculated using Theorem 2. $Q_i^F$ denotes **the 1-fault tolerant capacity** for scheduling the task set $\{\tau_1, ..., \tau_i\}$. For the worst-case processor demand, we use the following equation(Eq. (2) in [29]) :

$$W_i^F(t) = C_i + \sum_{j=1}^{i-1} \lceil \frac{t}{T_j} \rceil C_j + \max_{j=1,2,...,i} \lceil \frac{t}{T_F} \rceil C_j \tag{9}$$

Since we assume that maximum 1 fault occurs during the HyperPeriod of task set $\Gamma$, the most right hand side of the Eq. (9) can be represented as $\max_{j=1,2,...,i} C_j$.

- Basic Step : For the task $\tau_1$, there is no higher priority tasks. And the only time point we have to check is $T_1$. Therefore $S_1(T_1) = \frac{Q_1^F}{P} T_1$ and $W_1^F(T_1) = 2C_1$. We have to satisfy $S(T_1) \geq W_1^F(T_1)$. (for minimum $Q_1^F$, $S(T_1) = W_1^F(T_1)$

  $\frac{Q_1^F}{P} T_1 = 2C_1 \iff Q_1^F = \frac{2C_1}{T_1} P = 2U_1 P \ \ \therefore$ Capacity $A_1^F = \frac{Q_1^F}{P} = \frac{2U_1 P}{P} = 2U_1$.

- Induction Step : Suppose that for the task $\tau_k$ (i.e. $n = k$) , $Q_k^F$ is true. This means that $Q_k^F$ is the minimum budget for tolerating maximum 1 fault in task set $\{\tau_1, ..., \tau_k\}$. Thus, we cannot reduce the amount of the budget. In order to get the minimum $Q_{k+1}^F$, we need to care about 2 cases.

  1) Faults on the task $\tau_{k+1}$

  2) Faults on the higher priority tasks.

  For the case of 1), All we need to care about is ensuring additional time supply as $2C_{k+1}$ until $T_{k+1} = kT_k$. In this case, the worst-case processor demand can be represented as $W_{k+1}^F(kT_k) = kW_k^N(T_k) + 2C_{k+1}$. where $W_k^N(T_k)$ is the normal demand(i.e. no faults) until $T_k$ for the all higher priority tasks. We know that $S_{k+1}(kT_k) = \frac{Q_{k+1}^F}{P} kT_k$ and in order to get the minimum $Q_{k+1}^F$,

  $\rightarrow S_{k+1}(kT_k = T_{k+1}) = W_{k+1}^F(kT_k = T_{k+1})$

  $\rightarrow \frac{Q_{k+1}^F}{P} kT_k = W_{k+1}^F(kT_k) \qquad \because S_{k+1}(kT_k) = \frac{Q_{k+1}^F}{P} kT_k$

  $\rightarrow \frac{Q_{k+1}^F}{P} kT_k = kW_k^N(T_k) + 2C_{k+1} \quad \because W_{k+1}^F(kT_k) = kW_k^N(T_k) + 2C_{k+1}$

  $\rightarrow \frac{Q_k^N + \alpha}{P} kT_k = kW_k^N(T_k) + 2C_{k+1}$ $\because \alpha$ is additional time supply for task $\tau_{k+1}$

  $\rightarrow \frac{\alpha}{P} kT_k = 2C_{k+1} \qquad \because \frac{Q_k^N}{P}(kT_k) = kW_k^N(T_k)$

  $\rightarrow \alpha = P\frac{2C_{k+1}}{T_{k+1}} = 2PU_{k+1}$

  $\therefore Q_{k+1}^F = Q_k^N + \alpha = P\sum_{j=1}^{k} U_j + PU_{k+1} = 2P(\sum_{j=1}^{k+1} U_j) - P(\sum_{j=1}^{k} U_j) = 2Q_{k+1}^N - Q_k^N$

  Therefore for tolerating $\tau_{k+1}$, $Q_{k+1}^F = 2Q_{k+1}^N - Q_k^N$ and for tolerating $\{\tau_1, ..., \tau_k\}$ $Q_{k+1}^F = Q_k^F$. Simply considering this could lead $Q_{k+1}^F = \max(Q_k^F, 2Q_{k+1}^N - Q_k^N)$. However, we need to consider each cases with the each opposite side. If $Q_{k+1}^F = Q_k^F$, we have to consider the normal execution of $\tau_k$ addition to 1 faults on

higher priority tasks. If $Q^F_{k+1} = 2Q^N_{k+1} - Q^N_k$, we have to consider all normal executions of all higher priority tasks and 1 fault on task $\tau_{k+1}$. Thus, if $Q^F_{k+1} = Q^F_k$, we have to show that $W^F_{k+1}(kT_k) = kW^N_k(T_k) + C_{max} + C_{k+1} \leq \frac{Q^F_k}{P} kT_k$ is true where $C_{max}$ denotes the maximum WCET of the higher priority tasks for 1 fault and $C_{k+1}$ denotes the normal execution of the task $\tau_{k+1}$.

If $Q^F_{k+1} = Q^F_k$(i.e. $Q^F_k \geq 2Q^N_{k+1} - Q^N_k$ ), then $\frac{kT_k}{P}(Q^F_k - Q^N_k) > 2C_{k+1}$ is true since $kW^N_k(T_k) = \frac{Q^N_k}{P} kT_k$ holds.

By using this inequality, we have to show that $\frac{(Q^F_k - Q^N_k)}{P} kT_k \geq C_{max} + C_{k+1}$ is true.

$\rightarrow \frac{Q^F_k - Q^N_k}{P} T_{max} \geq C_{max}$ $\quad \because$ By the assumption of Theorem 4

$\iff \frac{Q^F_k - Q^N_k}{P} T_{max} = (1+\alpha)C_{max}$ where $\alpha \geq 0$ $\quad \because$ Above equation holds

We already know that

$\frac{kT_k}{P}(Q^F_k - Q^N_k) > 2C_{k+1} \iff \frac{knT_{max}}{P}(Q^F_k - Q^N_k) > 2C_{k+1} \because T_k = nT_{max}(n \geq 1)$

$\rightarrow kn(1+\alpha)C_{max} > 2C_{k+1}$ $\quad \because$ Substitute $\frac{T_{max}}{P}(Q^F_k - Q^N_k) = (1+\alpha)C_{max}$

$\rightarrow XC_{max} > 2C_{k+1}$ $\quad \because$ Put $X = kn(1+\alpha)$, $X \geq 2$ $(k \geq 2, n \geq 1)$

We have to check

$\frac{(Q^F_k - Q^N_k)}{P} kT_k \geq C_{max} + C_{k+1} \iff \frac{(Q^F_k - Q^N_k)}{P} knT_{max} - C_{max} \geq C_{k+1}$ holds.

$\rightarrow (kn(1+\alpha) - 1)C_{max} \geq C_{k+1}$ $\quad \because$ Substitute $\frac{T_{max}}{P}(Q^F_k - Q^N_k) = (1+\alpha)C_{max}$

$\rightarrow (X-1)C_{max} \geq C_{k+1}$ $\quad \because$ Put $X = kn(1+\alpha)$, $X \geq 2$ $(k \geq 2, n \geq 1)$

Since $\frac{2}{X} > \frac{1}{X-1} (X \geq 2)$ , $(X-1)C_{max} \geq C_{k+1}$ is also true.

If $Q^F_{k+1} = 2Q^N_{k+1} - Q^N_k$(i.e. $2Q^N_{k+1} - Q^N_k \geq Q^F_k$), then $kT_k \frac{Q^F_{k+1} - Q^N_k}{P} = 2C_{k+1}$ holds. By using this equation, we have to show that $kT_k \frac{Q^F_{k+1} - Q^N_k}{P} - C_{max} \geq C_{k+1}$

is true.

$\rightarrow T_{max}\frac{(Q^F_{k+1}-Q^N_k)}{P} \geq T_{max}\frac{(Q^F_k-Q^N_k)}{P} \geq C_{max} \because Q^F_{k+1} = 2Q^N_{k+1} - Q^N_k \geq Q^F_k$

$\iff T_{max}\frac{(Q^F_{k+1}-Q^N_k)}{P} = (1+\alpha)C_{max}$ where $\alpha \geq 0 \because$ Above equation holds

We already know that

$\rightarrow kT_k\frac{Q^F_{k+1}-Q^N_k}{P} = knT_{max}\frac{Q^F_{k+1}-Q^N_k}{P} = 2C_{k+1} \qquad \because T_k = nT_{max}(n \geq 1)$

$\rightarrow kn(1+\alpha)C_{max} = 2C_{k+1} \quad \because$ Substitute $\frac{T_{max}}{P}(Q^F_{k+1}-Q^N_k) = (1+\alpha)C_{max}$

$\rightarrow XC_{max} = 2C_{k+1} \qquad \because$ Put $X = kn(1+\alpha), X \geq 2\ (k \geq 2, n \geq 1)$

We have to check

$kT_k\frac{Q^F_{k+1}-Q^N_k}{P} - C_{max} \geq C_{k+1} \iff knT_{max}\frac{Q^F_{k+1}-Q^N_k}{P} - C_{max} \geq C_{k+1}$ holds.

$\rightarrow (kn(1+\alpha)-1)C_{max} \geq C_{k+1} \because$ Substitute $\frac{T_{max}}{P}(Q^F_{k+1}-Q^N_k) = (1+\alpha)C_{max}$

$\rightarrow (X-1)C_{max} \geq C_{k+1} \qquad \because$ Put $X = kn(1+\alpha), X \geq 2\ (k \geq 2, n \geq 1)$

Since $\frac{2}{X} > \frac{1}{X-1}(X \geq 2)$ , $(X-1)C_{max} \geq C_{k+1}$ is also true. Thus, Theorem 4 holds.

□

### 3.4 Periodic Resource Wide Schedulability Test

In this section, we present our Periodic Resource wide schedulability test in order to correctly use the brand-new ECU which has the lockstep/performance mode switch support. Suppose that we have critical task set $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$ which has to be ensured minimum coverage as 99.9%. In that case, all of them would require the detection methods only as DCLS(i.e. Dual-Core LockStep). Let say for every task $\tau_i$, the period $T_i$ of $\tau_i$ is either 2 or 5. In this case, we can group the task set $\Gamma$ as a one Periodic Resource $\Pi(P,Q)$ like in [1] using Theorem 1. However, our goal is to

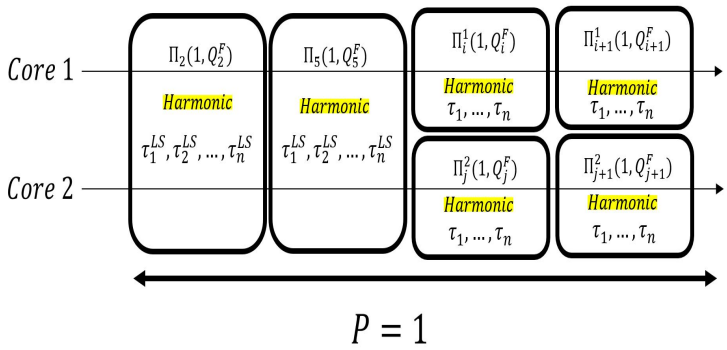



Figure 4: Schedulability test of the Periodic Resources

minimize the bandwidth for scheduling the task set $\Gamma$, and we know that the minimum bandwidth(i.e. minimum capacity) can be achieved for the all tasks when the task set is harmonic. Thus, we divide the given $\Gamma$ as 2 disjoint set as $\Gamma_2$ and $\Gamma_5$.(i.e. $\Gamma_2 \cap \Gamma_5 = \Phi$ and $\Gamma_2 \cup \Gamma_5 = \Gamma$). $\Gamma_2$ denotes the set of tasks which has the period as 2. Also, $\Gamma_5$ denotes the set of tasks which has the period as 5. Now, we have 2 harmonic task sets as $\Gamma_2$ and $\Gamma_5$. By Theorem 3 and 4, we can obtain the optimal capacity of $\Pi_2$ and $\Pi_5$ for both normal and faulty case(i.e. $\Pi_2$, $\Pi_5$ contains task set $\Gamma_2$, $\Gamma_5$, respectively). However, if we consider the Periodic Resource-wide schedulabiity, it is more highly likely schedulable if the period of $\Pi_2$ and $\Pi_5$ is also harmonic. Therefore, we fix the period of $\Pi_2$ and $\Pi_5$ as time unit 1. Note that by Theorem 2, we can obtain the optimal capacity with the period of $\Pi_2$ and $\Pi_5$ as time unit 1 since the period $P$(i.e. $P = 1$) of $\Gamma_2$ and $\Gamma_5$ is also the common divisor of the task set $\Gamma_2$ and $\Gamma_5$, respectively. Finally, we can obtain $Q_2^F$ and $Q_5^F$ and this can represented as $\Pi_2(1, Q_2^F)$, $\Pi_5(1, Q_5^F)$. All we need to check is whether $Q_2^F + Q_5^F \leq 1$. In summary, we propose a similar approach like [1], However, we fix the period $P$ of the Periodic Resource $\Pi(P, Q)$ as time unit 1 in order to increase Periodic Resource wide schedulability. This example

is described in Figure 4. Conclusionally, schedulability test can be check as follows :

$$\sum_{\Pi_i \in \Pi_{LS}} Q_i^F + \max(\sum_{\Pi_j \in \Pi^1} Q_j, \sum_{\Pi_k \in \Pi^2} Q_k) \leq 1 \tag{10}$$

$\Pi_{LS}$ denotes the lockstep Periodic Resource set which each element contains harmonic task set. $\Pi^1$ denotes the Periodic Resource set which each element contains harmonic task set where it is assigned to core 1. $\Pi^2$ denotes the Periodic Resource set which each element contains harmonic task set where it is assigned to core 2. Since $\Pi^1$ contains harmonic set either critical tasks with SW-based detection method or non-critical tasks, we do not represent the capacity $Q_j$ of individual $\Pi_j$. If it has the set of critical tasks, then it will have the capacity $Q_j^F$. Otherwise, it will have the capacity $Q_j^N$. Note that if the period $P$ is 1, then the capacity is equal to the budget $Q$. Xi et al [30] implemented the hierarchical real-time scheduling framework in RT-Xen. They reported that the context switch overhead ratio compare to the total used time is about 0.21% using Periodic Server [31] with the time quanta 1ms in RT-Xen which is the real-time extension of the virtual machine Xen [32]. Since our proposed approach does not require any scheduling decision in run-time. Also, if the number of harmonic task set is reasonably small(i.e. $\Gamma = \Gamma_1 \cup ... \cup \Gamma_n$, $n$ is small and $\forall\ \Gamma_i$ is harmonic), then we claim that our proposed approach can be used with the advantage of increasing the resource usage with the slightly increased mode switch overhead. Further discussions about this topic are out of the scope of this paper, we leave it as our future work. Finally, we show a interesting finding about fixed-priority schedulability test combined with Periodic Resource Model.

**Theorem 5.** *On single processor, given periodic task set* $\Gamma = \{\tau_1, ..., \tau_n\}$ *and the*

*period of all* $\tau_i$ *is the multiple of time unit 1(i.e.* $T_i = k \in Z^+$*), if* $\sum_{\tau_i \in \Gamma} u_i \leq 1$ *then, it is schedulable.*

*Proof.* The main concept for this proof is that we make one Periodic Resource $\Pi_i(P,Q)$ per every task $\tau_i$ and set the period of the each Periodic Resource $\Pi_i(P,Q)$ as 1 (i.e. $\Pi_i(1,Q)$). From Theorem 3, if we set the $P = 1$, and the period $T_i$ of the task $\tau_i$ is multiple of $P$, we can obtain the optimal capacity of the Periodic Resource $\Pi(1,Q)$ (i.e. $Q = u_i \ \because Q = A$ if $P = 1$). After then, we have to check Periodic Resource wide schedulability test for all $\Pi_i$. Since the period of every Periodic Resource $\Pi_i(1,u_i)$ is 1, and if $\sum_{\tau_i \in \Gamma} u_i \leq 1$ is true, then all Periodic Resource $\Pi_i$ is schedulable. Thus, Theorem 5 holds. □

This scheduling policy is neither EDF [33] nor Least-Laxity-First(LLF) [34]. It does not require any scheduling decision at run-time. However, the most critical weak point is that it requires additional context switch overhead exactly the same number of the Periodic Resources(i.e. the number of tasks in the given task set) per every 1 time unit. Thus, we expect that it cannot be efficiently used in practice because of the large context switch overhead. Note that this is different compare to our proposed approach(i.e. Periodic Resource wide schedulability test). Because ours use the same period of the every Periodic Resource $\Pi_i$ as 1. However, for our proposed approach, one Periodic Resource $\Pi_i$ contains harmonic task set not just only one task, so we expect that the context switch overhead is reasonably small if the number of the harmonic set is small.

### 3.5 Non-Critical Task Overlapping

From Theorem 3 and Theorem 4, it is obivious that if a fault does not occur, there are some idle time intervals (i.e. slacks) during the HyperPeriod of Periodic Resource $\Pi(P,Q)$ that contains harmonic task set $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$. Let $A_n^F$ denotes the minimum capacity in case of fault for the Periodic Resource $\Pi(P,Q)$ and $A_n^N$ as the minimum capacity in case of normal state for the periodic resource $\Pi(P,Q)$. If the periodic resource $\Pi(P,Q)$ contains harmonic task set $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$, then $A_n^N = \sum_{i=1}^{n} u_i$ and $A_n^F = \max_{k=1,2,...,n}((\sum_{i=1}^{k} u_k) + u_k)$. Since Periodic Resource gives the temporal isolation between itself and the other Periodic Resources, also we do not want to increase the capacity of $\Pi(P,Q^F)$ for inserting non-critical tasks into the Periodic Resources that only contains critical tasks. We permit to insert the non-critical tasks $\tau_i$ only when it uses idle time intervals in normal operation. First, we put every non-critical tasks having lower priority than every critical tasks in one Periodic Resource $\Pi(P,Q^F)$. If we let priority interleaving between critical tasks and non-critical tasks, there might be a consequences that we have to increase the capacity of the Periodic Resource $\Pi(P,Q^F)$.

**Theorem 6.** *Given Periodic Resource $\Pi(P,Q^F)$ that contains critical harmonic periodic task set $\Gamma_c = \{\tau_1, ..., \tau_n\}$ and non-critical periodic task set $\Gamma_{nc} = \{\tau_1, ..., \tau_{m-1}\}$. $\tau_m$ can be inserted into periodic resource $\Pi(P,Q^F)$ if the following equation holds.*

$$\frac{Q^F}{P} t_m \geq \sum_{j=1}^{n} \lceil \frac{t_m}{T_j} \rceil C_j + C_m + \sum_{j=1}^{m-1} \lceil \frac{t_m}{T_j} \rceil C_j \tag{11}$$

*Proof.* Since $\Gamma_{nc}$ is not harmonic, We use TDA with the scheduling point region as Eq. (6) (i.e. if there exists at least one $t_m \in SchedP_m$ that satisfies Eq. (11), then $\tau_m$

is schedulable). We assume that each priority ordering of critical task set and non-critical task set is based on the Rate-Monotonic(RM) assignment. Since we know the capacity of the periodic resource $\Pi$ as $A_n^N = \sum_{i=1}^{n} u_i$ and $A_n^F = \max_{k=1,2,\ldots,n}((\sum_{i=1}^{k} u_k) + u_k)$. The worst-case idle time amount is equal to $\frac{Q^F}{P}t_m$ - $\sum_{j=1}^{n} \lceil \frac{t_m}{T_j} \rceil C_j$ until $t_m \in SchedP_m$ in normal operation since we choose the period of $\Pi(P,Q)$ as 1(i.e. $P = 1$). This means that the worst-case time supply for the non-critical task $\tau_m$ occurs at the release time point 0 which is the critical instant for task set $\Gamma_c$. Consequently, if the worst-case processor demand of $\Gamma_{nc}$ until time $t_m$ and additional demand which is equal to $C_m$ is smaller or equal to the idle time amount until $t_m$, then it is schedulable without increasing the capacity of Periodic Resource $\Pi(P,Q)$. Thus, if Eq (11) holds for at least one $t_m \in SchedP_m$, then $\tau_m$ is schedulable.

□

# 4 Proposed Approach

In this section, we first describe about the minimum harmonic partitions of the given task set with the given periods. Then, using the definition of minimum harmonic partition, we further suggest the concept of the free tasks following with the minimum harmonic partitions concept. Next, we present our algorithm that proposes grouping a sub-taskset as one Periodic resource $\Pi$ and Periodic Resource wide bin-packing heuristic algorithm. Finally, we analyze the time complexity of the proposed algorithm.

## 4.1 Minimum Harmonic Partitions of the Task Set

Recall that the period distirbution of the task set in practice is not purely random. Burchard et al[35] proposed the partitioned fixed-priority preemptive scheduling approach using the concept that if the given period distribution of the given tasks that are mapped to the same single processor is more close to harmonic, then usually the overall schedulability of the task set tends to achieve higher than the well-known bin-packing heuristics (e.g. Best-Fit(BF), Worst-Fit(WF)). Also, we find out that the optimal capacity of the Periodic Resource $\Pi$ which contains harmonic periodic task set $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$. Thus, simliar to [35], we aim to use the concept that 'more close to harmonic, then more schedulable'. We introduce the definition of the minimum harmonic partitions of the task set, and free tasks as follows:

**Definition 5.** ***Minimum harmonic partitions of the task set***

*Given periodic task set* $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$ *and their period set as* $\mathbb{T} = \{T_1, T_2, ..., T_n\}$, ***the minimum harmonic partitions of the task set*** $H(\Gamma)$ *is the* ***minimum number***

***of partitions*** *that each partition is harmonic and their union is* $\Gamma$. *Formally,* $\Gamma = \{\Gamma_1 \cup \Gamma_2 \cup ... \cup \Gamma_h = \Gamma \mid \forall i, 1 \leq i \leq h, \Gamma_i$ *is harmonic and* $\forall 1 \leq i, j \leq h, \Gamma_i \cap \Gamma_j = \phi\}$.

**Definition 6.** ***Free tasks***

*Given periodic task set* $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$ *and their period set as* $\mathbb{T} = \{T_1, T_2, ..., T_n\}$, *if the minimum harmonic partition of the task set* $H(\Gamma) = h$ *and* $\Gamma = \{\Gamma_1 \cup ...\Gamma_h\}$, *the task* $\tau_i \in \Gamma_x$ *is the free task if and only if all partitions are harmonic even when it moves to any other partitions* $\Gamma_y \in \{\Gamma_1 \cup ...\Gamma_h\} - \Gamma_x$.

We describe 2 above definitions using a simple example. Consider a task set with periods as $\mathbb{T} = \{1, 2, 5, 10, 20, 50, 100, 200, 1000\}$ then, the minimum harmonic partitions of the task set $H(\Gamma)$ is 2. One example of the partition is $\mathbb{T}_1 = \{1, 2, 10, 50, 100\}$ and $\mathbb{T}_2 = \{5, 20, 200, 1000\}$. Thus, the set of free tasks is $\mathbb{T}_{free} = \{1, 10, 100, 200, 1000\}$ and the sets of minimum harmonic partitions are $\mathbb{T}_1 = \{2, 50\}$ and $\mathbb{T}_2 = \{5, 20\}$. First, we introduce how to find the minimum partitions of the task set. From the beginning, let us assume that the given period set of the task set is as follows : $\mathbb{T} = \{T_1, T_2, ..., T_n\}$. Then, we sort the set by non-increasing order (i.e. $T_n \geq T_{n-1} \geq ... \geq T_1$). After then, we construct a Directed Acyclic Graph(DAG) which each vertex represents the value $T_i \in \mathbb{T}$ and each edge $T_i \rightarrow T_j (i \neq j)$ represents that $T_i \geq T_j$ and $T_j$ can be divided by $T_i$. Note that the time complexity of constructing DAG takes $O(n^2)$. After constructing this DAG, we make a bipartite graph G with the vertex set $\mathbb{L}$(left), $\mathbb{R}$(right) and edge set $\mathbb{E}$. From pre-constructed DAG, if there exists a edge $T_i \rightarrow T_j (i \neq j)$, we push the vertex $T_i$ to $\mathbb{L}$, push the vertex $T_j$, and push the edge $T_i \rightarrow T_j (i \neq j)$ to $\mathbb{E}$. Then, we run the maximum bipartite matching algorithm for the given bipartite graph $G$. After then, we erase all the edges in the pre-constructed DAG, and only add the edges that is found by the maximum bipartite matching algorithm. Then, we know that from the

start node(i.e. that has no predecessor) to finish node(i.e. that has no successor), there exists only one path. From this observation, we can find out that the elements of the each partition is composed of the set of vertices of the each path. Now we can easily find out the set of free tasks by checking each partition with the Definition 6.

## 4.2 Proposed Heuristic Algorithm

Recall that our objective is to find a feasible mapping $M$ in Definition 2. In order to do that, we first find a Feasible Task-to-Core Mapping $M_{i,j,k,l}$ in Definition 2 for every task in every state from starting core number as 1.

### 4.2.1 Choosing Detection method

So first, in each state, we select the detection method of the critical task $\tau_i$, if more than one detection method is possible (i.e. guaranteed error coverage is higher or equal to the minimum required coverage). Given candidate detection method set is $\Lambda_i = \{d_1, d_2, ..., d_l\}$, selection is determined as follows:

$$d_j : \min_{j=1,2,...,l} \bar{C}_j, \quad \bar{C}_j = \begin{cases} C_i(1+o_{i,j}) & \text{if } d_j \text{ is not lockstep} \\ 2C_i(1+o_{i,j}) & \text{otherwise} \end{cases} \tag{12}$$

Recall that $o_{i,j}$ represents the overhead of the detection method $d_j$ if it is used on task $\tau_i$. Rationale behind this selection is simple. We expect that lower WCET $\bar{C}_i$ will bring higher schedulability on the overall bin-packing process. We assume that maximum 1 fault occurs during the HyperPeriod of the given task set and the recovery mechanism is re-execution. After the selection of the detection method per every task, we divide them into 3 categories. One for the task set that uses DCLS detection method, another

for the task set that uses either Control-Flow Check(CFC) or Valid-Range Check method, the other for the task set that is non-critical. After then, in each task set, we find out the minimum harmonic partitions of the task set and the set of free tasks. After figuring out them, we first try to pack the minimum harmonic partitions first. Next, we pack the set of free tasks.

### 4.2.2 Packing Minimum Harmonic Partitions

In order to save the wasted time supply in each Periodic Resource $\Pi_i(1, Q_i)$, we try to separate different partitions into the different cores as much as possible. Since one partition(i.e. one Periodic Resource $\Pi(1, Q)$) wastes the idle time for the 1 fault and we cannot increase the capacity of the one partition even though we know maximum 1 fault occurs and more than 2 partitions are exist in same core. There are 2 reasons for our heuristic. First, suppose that we have more than 2 critical Periodic Resources in the same core, then it would require additional time supply for the recovery for each Periodic Resource. However, we cannot decrease the amount of the time supply because of the Theorem 3 and Theorem 4. Suppose that we have 2 critical Periodic Resources as $\Pi_1(1, Q_1^F)$ and $\Pi_2(1, Q_2^F)$. Simply thought that, it seems like we can reduce the amount of the capacity as $Q_1^N$ and $Q_2^N$. Thus, making spare capacity as $\max(Q_1^F - Q_1^N, Q_2^F - Q_2^N)$. However, by Theorem 3, if $\Pi_1$ only gets its budget as $Q_1^N$, for the lowest priority jobs, they will finish their execution exactly at the deadline if all jobs finish with their WCET. Hence, we cannot reduce the amount of time supply and we cannot determine where to give the spare capacity. (e.g. in this case, for $\Pi_1$ and $\Pi_2$). Second, even though we use the overlapping the non-critical tasks with using the idle time intervals of critical Periodic Resources, it cannot give

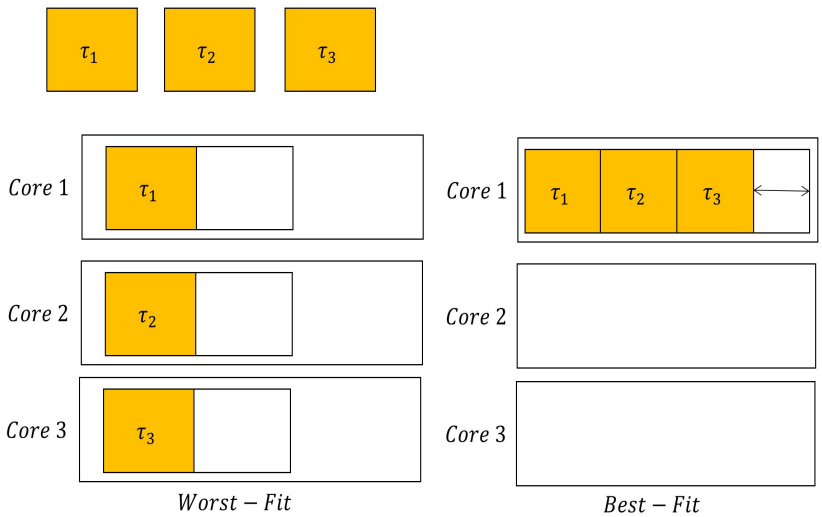



Figure 5: Heuristic for the free tasks

many chance to higher priority non-critical tasks. We assume that priority assignment is Rate-Monotonic(RM). More details are described in Section 4.2.4. Finally, after spreading the minimum harmonic partitions as much as possible, we try to pack the free tasks which leads the net capacity increase smallest among feasible cores. For this rationale is simple because the bigger the one Periodic Resource $\Pi(1,Q)$ is(i.e. contains many tasks), the smaller the net capacity increase is. Details are described in Section 4.2.3.

### 4.2.3 Packing Free Tasks

We know all free tasks can be packed into any Periodic Resource of any cores since it does not violate the harmonic property of the assigned Periodic Resource. Note that if task $\tau_i$ is critical, then it can only be packed to the Periodic Resource $\Pi(1,Q)$ that contains critical task set. We find out that the minimum capacity of the Periodic Resource $\Pi$ that contains the harmonic periodic task set $\Gamma = \{\tau_1, ..., \tau_n\}$ as

$\max_{k=1,2,\ldots,n}((\sum_{i=1}^{k} u_k) + u_k)$. This equation tells us that if we want to add a new task $\tau_i$ into Periodic Resource $\Pi(1,Q)$ which is also harmonic, then the net capacity increase depends not only the utilization of new task $\tau_i$, but also the already packed tasks. Figure 5 describes this behavior. Suppose that $\tau_1, \tau_2, \tau_3$ have a same period and WCET and have a utilization value $u$. If we make a Periodic Resources to the core like Worst-Fit manner, every Periodic Resource on every core wastes the capacity of $u$ in normal operation. If we make Periodic Resources to the core like Best-Fit manner, 1 Periodic Resource on core 1 wastes only $\frac{1}{4}$ of total capacity. From this observation, we select the Worst-Fit Decreasing in terms of net capacity increase for the free tasks. Note that Figure 5 can be interpreted as total capacity increase wide Best-Fit manner or net capacity increase wide Worst-Fit manner. Also, this is why we select to keep one Periodic Resource $\Pi(1,Q)$ bigger as much as possible for the minimum harmonic partitions and the free tasks in the same reason.

#### 4.2.4 Packing Non-Critical Tasks

The main difference between critical-task and non-critical task is that non-critical task has a chance to be overlapped in the idle time intervals of the critical Periodic Resource $\Pi$. From Theorem 6, we know that the overlapping the non-critical tasks in the critical Periodic Resources does not incur further capacity increase. So we first try to overlap the non-critical tasks into the Periodic Resource $\Pi$ either DCLS Periodic Resource $\Pi_{DCLS}$ or 1-core protection Periodic Resource $\Pi_{1core}$(i.e. SW-based detection method). However, this time, we first sort the non-critical task set as RM priority order. Given Eq (11), if we pack non-critical task $\tau_i$ randomly, then we always have to double-check all tasks which has lower priority than $\tau_i$. This is why we sort the

non-critical tasks as RM-order. After overlapping some non-critical tasks, With the rest non-critical task set(i.e. cannot be overlapped in any critical Periodic Resources), we try to use the same approach as critical tasks for making Periodic Resources for minimum harmonic partitions and free tasks. Note that for non-critical tasks, we can use capacity of the Periodic Resource $\Pi$ as $\sum_{i=1}^{n} u_i$ by Theorem 3.

### 4.3 Algorithm Description

In summary, our proposed heuristic algorithm can be represented as Algorithm 1 described in the next page. The algorithm first finds the detection method of the critical tasks with Eq (12). This takes $O(ln)$ where $l$ is the size of the detection method set $\Lambda$ and $n$ is the size of task set $\Gamma$. Then, The **for**-loop finds the total needed core in each state from Line 7 to Line 33. The **if**-conditional branch from Line 30 to Line 31 saves the maximum number of cores for the entire states. The **while**-loop from Line 10 to Line 29 finds a feasible mapping $M$. If it cannot find a feasible mapping $M$, then it increases the number of cores one by one until it gets a feasible mapping $M$. The statements from Line 11 to Line 13, finds the minimum harmonic partitions and the set of free tasks. It takes $O(n^2)$. Each **for**-loop from Line 14 to Line 16, from Line 17 to Line 19 and from Line 24 to Line 26 try to pack every tasks in each task set(i.e. critical-by-lockstep or critical-by-SW-detection or non-critical). For critical task, on each core, capacity calculation of the task for one Periodic Resource takes $O(n)$ by Theorem 3 and Theorem 4. For non-critical task, on each core, capacity calculation of the task for one Periodic Resource also takes $O(n)$. However, it would leads the time complexity of $O(TDA(n))$ since non-critical task first tries to find a critical Periodic Resource that can be overlapped by Theorem 6. In the worst case, there might be $n$ Pe-

**Algorithm 1** Proposed Algorithm

**Input:** $\Gamma$(task set), $\mathbb{S}$(state set), $\mathbb{O}$(overhead table),$\Lambda$(detection method set), $\mathbb{C}ov$(coverage table), $\mathbb{R}$(minimum requirement table)

**Output:** Number of cores, Feasible Mapping $M$

$max_{core} \leftarrow 1$
**for** $\tau_i \in \Gamma$ **do**
  **if** $\tau_i$ is critical **then**
    find a detection method using Eq (12)
  **end if**
**end for**
**for** $s_i \in \mathbb{S}$ **do**
  $num_{core} \leftarrow 1$
  $schedulable \leftarrow false$
  **while** $schedulable = true$ **do**
    Find Definition 5-6 in $\Gamma_{LS}$ // lockstep
    Find Definition 5-6 in $\Gamma_{1core}$// CFC, valid-range
    Find Definition 5-6 in $\Gamma_{nc}$ // non-critical
    **for** $\tau_j$ in $\Gamma_{LS}$ **do**
      try pack $\tau_j$ **if** false **then** $num_{core} \leftarrow num_{core} + 1$, continue
    **end for**
    **for** $\tau_j$ in $\Gamma_{1core}$ **do**
      try pack $\tau_j$ **if** false **then** $num_{core} \leftarrow num_{core} + 1$, continue
    **end for**
    **for** $\tau_j$ in $\Gamma_{nc}$ **do**
      sort $\Gamma_{nc}$ in RM ascending order
      try overlapping on the Periodic Resources using Eq (11)
    **end for**
    **for** $\tau_j$ in $\Gamma_{nc_{rest}}$ **do**
      try pack $\tau_j$ **if** false **then** $num_{core} \leftarrow num_{core} + 1$, continue
    **end for**
    $schedulable \leftarrow true$
    save feasible mapping M on state $s_i$
  **end while**
  **if** $num_{core} > max_{core}$ **then**
    $max_{core} \leftarrow num_{core}$
  **end if**
**end for**
**return** $max_{core}$ and $Mapping\ M$

riodic Resources a task can be assigned to. So checking one core takes $O(nTDA(n))$ and Periodic Resource wide schedulability analysis takes $O(n)$ Thus, for one core, it takes $O(nTDA(n))$. In the worst case, there might be a feasible mapping $M$ at maximum the number of core is $n$, so scanning all cores for one task takes $O(n^2TDA(n))$. Thus, checking every task takes $O(n^2TDA(n))$. Thus, overall time complexity of the algorithm is $O(n^2TDA(n)+nl)$ which is pseudo-polynomial [36].

# 5 Evaluation

We use synthetic task sets that has similar period distribution of [24]. In Section 7.1, we describe our experimental setup. In Section 7.2, we analyze our proposed algorithm compare to partitioned fixed-priority preemptive scheduling approach, and partitioned fixed-priority preemptive mixed-criticality scheduling approach. In Section 7.3, we analyze the effect of our proposed heuristics as the effect of minimum harmonic partitions, free tasks, overlapping the non-critical tasks and state-wise computation separately.

## 5.1 Experimental Setup

| Period | Rate |
|---|---|
| 1ms | 3.4% |
| 2ms | 2.5% |
| 5ms | 2.5% |
| 10ms | 29.4% |
| 20ms | 29.4% |
| 50ms | 3.4% |
| 100ms | 23.5% |
| 200ms | 1.2% |
| 1000ms | 4.7% |

Table 1: Period distribution of the experiment

For our simulation, a synthetic task $\tau_i = (C_i, T_i)$ is generated as follows : (1) Like [24], we fixed task period set as $\{1,2,5,10,20,50,100,200,1000\}$. However, we do not use the sporadic tasks in the system model, so we scaled up the utilization sum of the periodic tasks are equal to 100%. The distribution of the tasks among those periods is given in Table 1. For the detection method set, we choose the Valid-Range Check,

CFC and DCLS. For every task, we randomly assign the overhead of each method as follows : Valid-Range Check = 0%, overhead of CFC =[10%-60%] [20], overhead of DCLS=3% [37]. We randomly choose the task either critical-task or non-critical-task with the uniform probability(i.e. $\frac{1}{2}$). If the task is critical, then we randomly assign the criticality level among 3 different critiality-level with the uniform distribution. For criticality A(lowest), we set the task uses detection methods as valid-range check or CFC or DCLS. For criticality B, we set the task uses detection methods as CFC or DCLS. For criticality C, we set the task uses the detection method only DCLS. We generate the 4 state for every task set. In each state, we assign the criticality of the task among lower criticality or original criticality with the uniform probability (e.g. if task $\tau_i$ has criticality B in original one, it can only have criticality of A or B in every state). The utilization of each task is generated using UUnifast algorithm [38]. When the utilization of the task exceeds 0.5, then the task is not schedulable because of the re-execution cost. We exclude such cases. We generate 100 task sets for every time points and each time points results the same utilization sum from 0.1 to 4.0 with the increase amount 0.1. One task set contains 40 periodic tasks and the WCET of the task is determined by the period and the utilization of the task. We set the final WCET $\bar{C}_i$ as the summation of original WCET + overhead of the selected detection method. Then we test the schedulability of every task with the final WCET $\bar{C}_i$

## 5.2 Simulation Results

At first, we compare our algorithm to partitioned fixed-priority preemptive scheduling approach as FP and partitioned fixed-priority preemptive mixed-critical scheduling approach as FP-MC. FP is implemented as follows : first we select the detection

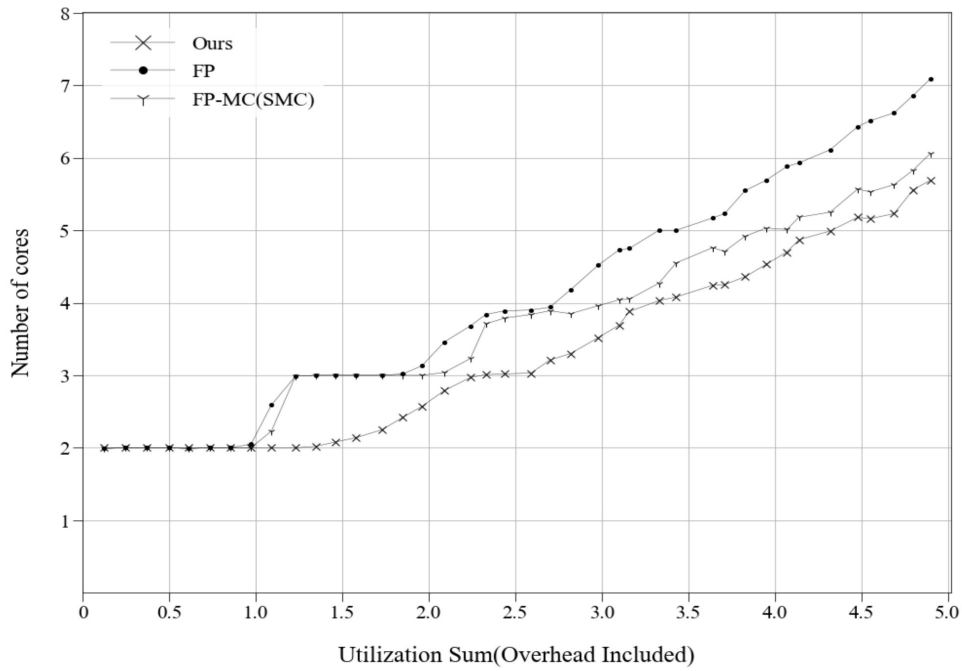



Figure 6: Simulation results compared to FP and FP-MC

method same as ours. Then, we divide task set into 3 categories same as ours. Then, we sort each task set according to RM-order. and try to packing the tasks using Time Demand Analysis(TDA) from highest priority to lowest priority. After packing of the lockstep task is finished, we try to pack the critical-task or non-critical task in the lockstep core as a single task if the period of the task is larger or equal to the lowest period of the lockstep task. Then, we try to pack the critical-task or non-critical task as a lockstep task since a single task cannot be schedulable if it has a higher priority than any lockstep task. (e.g. lockstep task with period 8 and WCET 4 on core 1 and core 2, a single task with period 2 and WCET 1 on core 1 and the other single task with period 4 and WCET 2 on core 2). After that, if there exists unpacked tasks, then we add 1 core and try to pack it with the traditional fault-tolerant fixed-priority schedulability

test [29]. Note that if the task is non-critical, then we guarantee no job drop of the every non-critical tasks even if 1 fault occurs. However, we do not guarantee re-execution of the every non-critical task when the faults occur. FP-MC is implemented as follows : same approach is used as FP, but the only difference is that we conduct the schedulablility test based on SMC [39]. For critical-tasks, we guarantee the normal execution with the all tasks and also we guarantee the re-execution with the other critical tasks. For non-critical tasks, we only guarantee the normal execution with all tasks when faults does not occur. Main difference between FP and FP-MC is that FP-MC drops non-critical tasks when the faults occur. Figure 6 shows the number of needed cores for ours, FP and FP-MC. Note that x-axis denotes the summation of utilization with the modified WCET $\bar{C}_i$. We can observe that ours performs better than FP and FP-MC for the all points of the x-axis. Ours can save up to 1.3 cores when the utilization sum is 5.0. This is because our proposed algorithm efficiently uses the lockstep tasks and single tasks based on optimal capacity, proposed heuristics that uses harmonic property, non-critical task overlapping and state-wise computation with the Periodic Resource Model. Note that ours only drops the non-critical tasks that are packed in the critical Periodic Resources and it can resume normal execution when the faults are absent.

### 5.2.1 Free Task Bin-Packing

In this section, we compare our proposed heuristic approach from the following 4 different approaches in order to investigate the effect of the free task. Note that non-critical task overlapping and state-wise computation is not conducted. Also, minimum harmonic partition tasks are packed with the our proposed apporach.

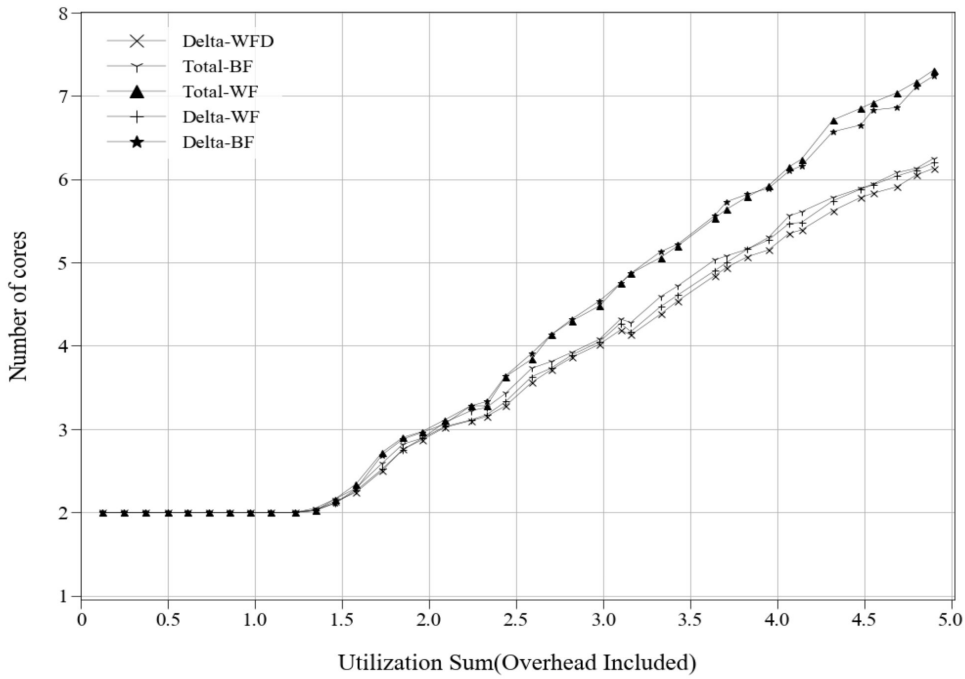



Figure 7: Free task packing

- Delta-WFD(Ours) : If there exists feasible cores and feasible Periodic Resources more than one, the task is packed to the core which incurs minimum net capacity increase of the Periodic Resource. The only difference between Delta-WF and Delta-WFD is that for Delta-WFD, task is sorted with non-increasing order by its utlization.

- Total-BF : If there exists feasible cores and feasible Periodic Resources more than one, the task is packed to the core which incurs maximum total capacity of the core.

- Total-WF : If there exists feasible cores and feasible Periodic Resources more than one, the task is packed to the core which incurs minimum total capacity

of the core.

- Delta-WF : If there exists feasible cores and feasible Periodic Resources more than one, the task is packed to the core which incurs minimum net capacity increase of the Periodic Resource.

- Delta-BF : If there exists feasible cores and feasible Periodic Resources more than one, the task is packed to the core which incurs maximum net capacity increase of the Periodic Resource.

Figure 7 shows the experimental result on the effect of the free tasks. As we can find out from the Figure 7, Total-BF, Delta-WFD, Delta-WF performs similar results. Among them, Delta-WFD(Ours) is the best. On the other hand, Total-WF and Delta-BF performs poor on every time point. This is because Total-WF and Delta-BF tends to separate the tasks among cores that it wastes the large amount of idle time intervals in each core(i.e. wasted capacity(fault capaity-normal capacity) ratio is large compare to normal capacity)

### 5.2.2 Minimum Harmonic Partitions Bin-Packing

In order to investigate the effect of the minimum harmonic partitions, we compare our proposed heuristic approach from the following 2 different approaches. Note that non-critical task overlapping and state-wise computation is not conducted. Also, free tasks are packed same as the our proposed approach. All approaches have the same as described in Section 7.2.1.The only difference is that target task is the element of minimum harmonic partitions. Figure 8 shows the experimental result on the effect of the minimum harmonic partition tasks. As we can find out from the Figure 8,

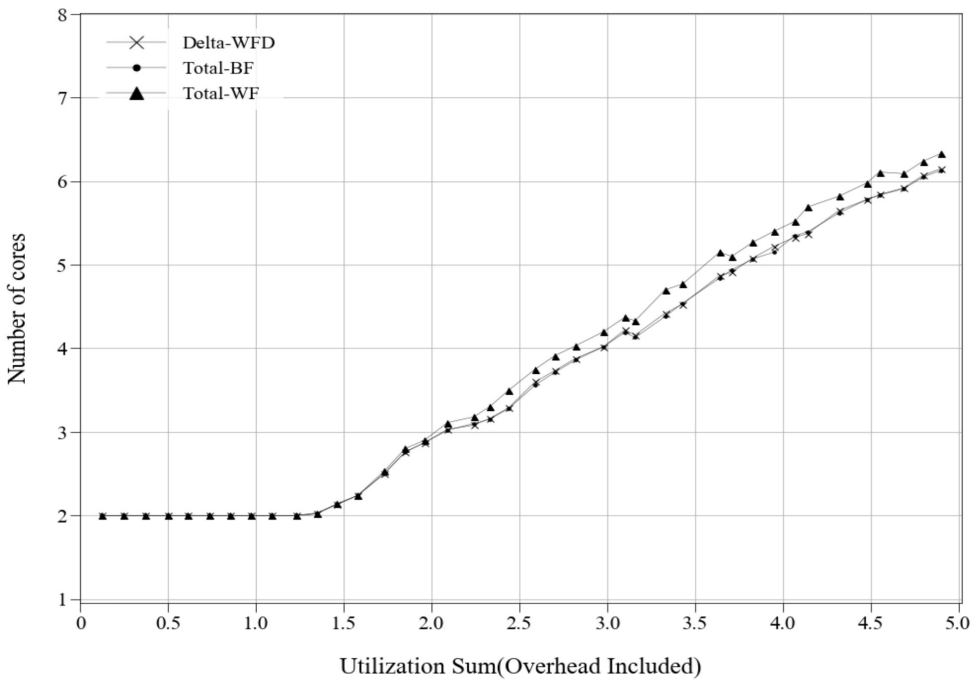
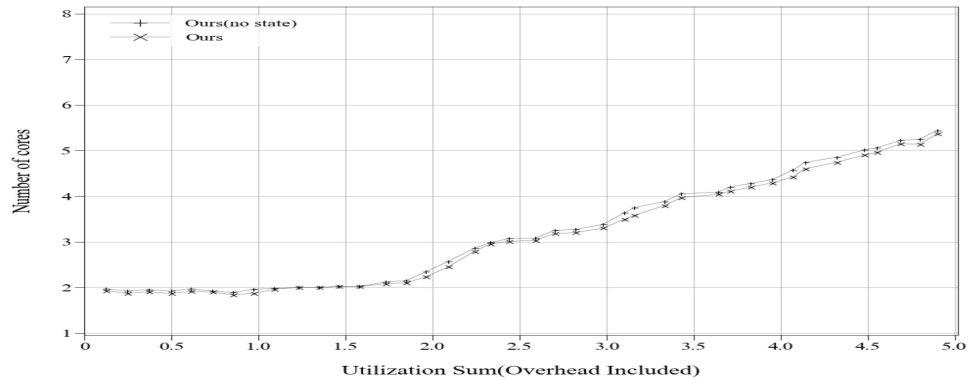
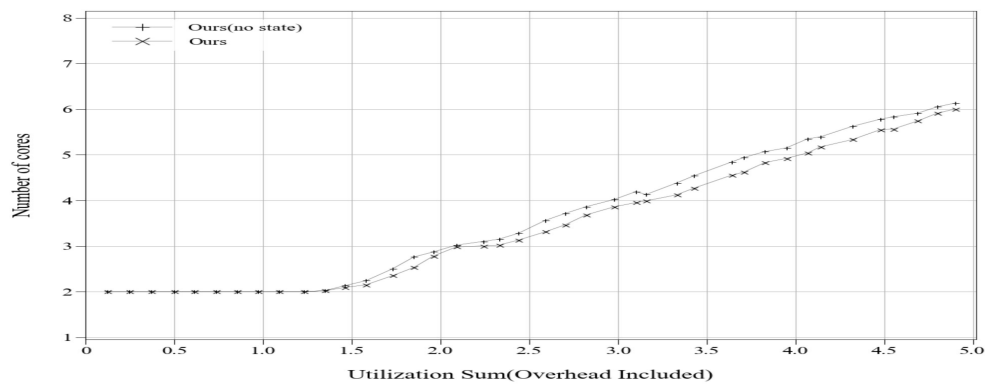



Figure 8: Minimum harmonic partitions packing

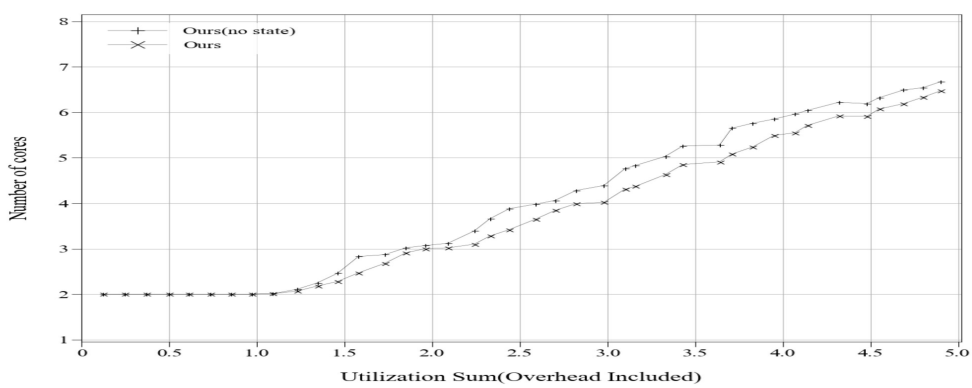
Delta-WFD and Total-BF performs better than Total-BF. Same reasoning in Section 7.2.1 can be used in this result. However, the difference is not big enough compare to Figure 7. This is because from the Table 1, the total utilization ratio of the minimum harmonic partitions are smaller than the total ratio of the free tasks. Note that we conduct the another approach that is contrary to our proposed algorithm such that it does not consider the harmonic property of the Periodic Resources, and turns out to perform very poorly compare to ours(e.g. difference is more than 2 cores in utilization point 5.0).




(a) 2:8 (Critical : Non-Critical)




(b) 5:5 (Critical : Non-Critical)




(c) 8:2 (Critical : Non-Critical)

Figure 9: Non-critical task overlapping effect

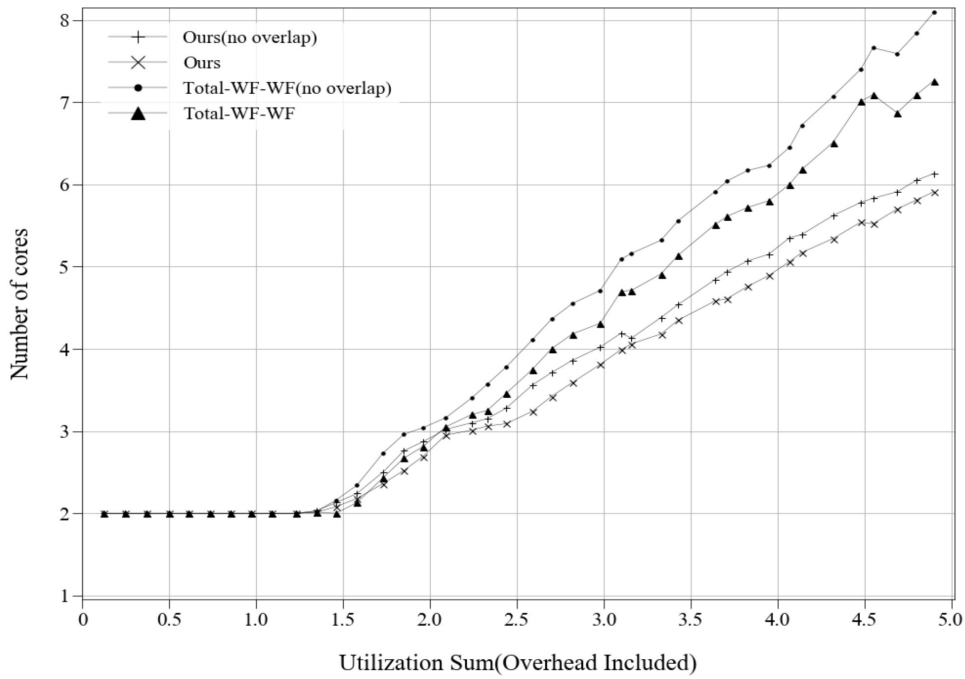
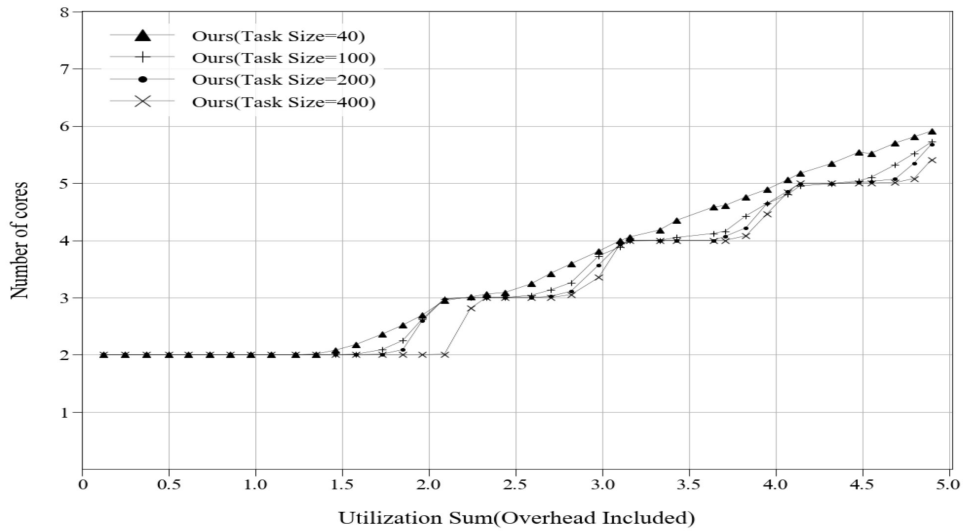


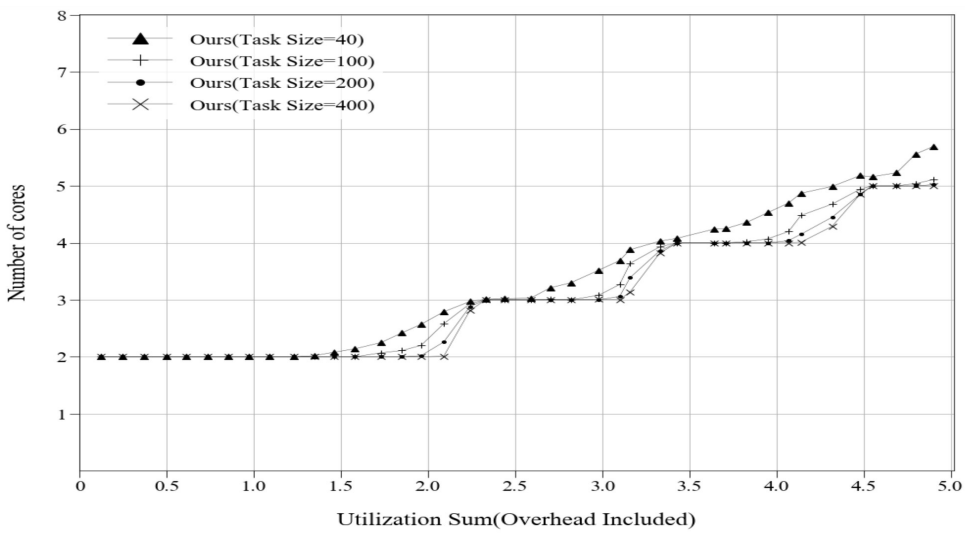

Figure 10: Effect of non-critical task overlapping between different packing heuristics

### 5.2.3 Effect of Non-Critical Task Overlapping

In order to investigate the effect of the non-critical task overlapping, we vary the ratio of critical task and non-critical tasks as follows: Figure 9(a) has 2:8 ratio. Figure 9(b) has 5:5 ratio. Figure 9(c) has 8:2 ratio. From the Figure 9, we can observe that as the ratio of the critical task gets high, difference between non-overlapped result and overlapped result gets larger. This indicates that our non-critical task overlapping efficiently works for the given task set. Also, Figure 10 shows that if we pack tasks in more separated manner(e.g. Total-WF for minimum harmonic partitions and free tasks), non-critical task overlapping performs much better compare to our proposed algorithm. This is because Total-WF tends to waste much bigger portion compare to its normal capacity. The proposed overlapping performs well since there exists more




(a) No State-wise computation




(b) State-wise computation

Figure 11: Effect of state-wise computation

idle time intervals.

### 5.2.4 Effect of State-Wise Computation

In order to investigate the effect of the state-wise computation, we vary the number of tasks as 40,100,200,400. If the number of tasks in one task set increases, then the chance of state-wise criticality reduction increases. In Figure 11, we can observe that state-wide reduction performs better results especially in the duration of [1.8, 2.3],[2.7,3.2] and [4.5, 5.0]. This is because the more the number of tasks, the more chance to be reduced in criticality level. We can find out when the number of the tasks in one task set gets bigger, the state-wise computation performs better. Note that if we increase the number of tasks in one task set fixing the total utilization sum as same value, then there might be improvement for the number of needed cores since tolerating fault cost(i.e. re-execution cost) can be reduced with the large amount.

Overall, Ours performs better than traditional approach(i.e. FP and FP-MC) when we use the lockstep core and single core simultaneously(i.e. reconfigurable system). Proposed algorithm based on the optimal capacity of the Periodic Resource, preserving Periodic Resource wide the harmonic property, bin-packing strategy based on the net increase by the worst-fit decreasing manner, overlapping non-critical tasks, state-wise computation shows that we can save up to 18% on the total number of needed cores.

# 6 Related Works

## 6.1 Hierarchical Fault-Tolerant Real-Time Scheduling

Shin and Lee [26] first presents the Periodic Resource Model and they present the methodology for finding the minimum budget $Q$ with the given period $P$ and the periodic task set $\Gamma$ for both RM and Earliest-Deadline-First(EDF) scheduling domain. Crinei et al [1] first present the methodology for finding the period $P$ and the budget $Q$ with the given periodic task set to efficiently use the reconfigurable system(lockstep or parallel mode). However, they did not consider the partitioning about each task. Also, they did not present the capacity value using the utilization of each individual task in one task set. Hyun et al [11] proposed the hierarchical real-time scheduling framework. Similar to [1], they did not present the capacity value as the summation of the utilization of the each individual task in one task set. Tchamgoue et al [14] proposed the Periodic Resource Model corporating the fault-tolerant property as the number of additional times and additional supplies. However, They did not consider the fault-tolerance property when the fault is occurred at the end of the deadline in the proposed fault-tolerant Periodic Resource Model.

## 6.2 Error Detection Method

Kottoke et al [9] first presents the reconfigurable lockstep/parallel architecture. Salloum et al [40] showed that lockstep architecture can bring higher error detection ratio upon transient faults(i.e. bit-flip). Kottke et al [7] showed that lockstep architecture can bring higher error detection ratio upon stuck-at-faults. Touloupis et al [21] showed that lockstep detection method performs better than the valid-range check

in terms of error detection. Rhisheekesan et al [22] showed that control-flow-check methods cannot protect important registers in CPU. For further interested readers on lockstep architecture, please refer [41, 42, 43, 44]. For further interested readers on SW-Based error detection technique, please refer [37].

# 7 Conclusion

This paper presents hierarchical fault-tolerant real-time scheduling framework in order to efficiently use the given reconfigurable system. Proposed approach first describes the optimal capacity of Periodic Resource that contains harmonic periodic task set using the utilization of the each individual task when the system is normal (i.e. no faults) and faulty(i.e. faults are present), respectively. After then, we present non-critical task overlapping technique which does not increase the capacity of the Periodic Resource using only idle time intervals of the Periodic Resources. And we propose bin-packing algorithm using net capacity increase-based Worst-Fit Decreasing algorithm for free tasks and minimum harmonic partition tasks. At the end, we propose state-wise criticality consideration scheme. The proposed algorithm finds the number of needed cores based on worst-case state. Through out the simulation results, we confirm that our proposed algorithm performs better than the traditional partitioned fixed-priority scheduling approach and partitioned mixed-critical fixed-priority scheduling approach. Moreover, the proposed heuristic algorithm gives better performance compare to the other well-known heuristic algorithms. To the best of our knowledge, This is the first work to finding out the optimal capacity of the Periodic Resource that contains harmonic periodic task set with the utilization of each individual task of one Periodic Resource for both normal state and faulty state. Also, This paper first presents the method for grouping sub-taskset as a one Periodic Resource with the given input set and partitioning the Periodic Resources upon with the given cores. The following are our future works to cover the limitations of the proposed approach in this paper.

- **Measuring the actual mode swtich overhead:** In this paper, we assume that mode switch between lockstep and parallel mode is negligible. However, more precise evaluation based on mode switch overhead is needed. In the future, we expect to improve the practicality by reflecting the mode switch overhead into our system model.

- **Heterogenous computing** : In this paper, we assume that the periodic task is executed only on the CPU(i.e. core). However, in practice, the tasks require addtional processors (e.g. GPU, NPU, etc). In the future, we plan to reflect those heterogeneous relation into our problem. By doing this, we expect that our proposed approach can be widely used in practice.

# References


[1] Michele Cirinei, Enrico Bini, Giuseppe Lipari, and Alberto Ferrari. A flexible scheme for scheduling fault-tolerant real-time tasks on multiprocessors. In *2007 IEEE International Parallel and Distributed Processing Symposium*, pages 1–8. IEEE, 2007.

[2] Premkishore Shivakumar, Michael Kistler, Stephen W Keckler, Doug Burger, and Lorenzo Alvisi. Modeling the effect of technology trends on the soft error rate of combinational logic. In *Proceedings International Conference on Dependable Systems and Networks*, pages 389–398. IEEE, 2002.

[3] Shubhendu S Mukherjee, Joel Emer, and Steven K Reinhardt. The soft error problem: An architectural perspective. In *11th International Symposium on High-Performance Computer Architecture*, pages 243–247. IEEE, 2005.

[4] Florian Many and David Doose. Scheduling analysis under fault bursts. In *2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 113–122. IEEE, 2011.

[5] Ifeanyi P Egwutuoha, David Levy, Bran Selic, and Shiping Chen. A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems. *The Journal of Supercomputing*, 65(3):1302–1326, 2013.

[6] Thomas Kottke and Andreas Steininger. A generic dual-core architecture. In *7th IEEE International Workshop on Design and Diagnostics of Electronic Circuits and Systems (DDECS 2004)*, pages 159–166, 2004.

[7] Thomas Kottke and Andreas Steininger. A generic dual core architecture with error containment. *Computing and Informatics*, 23(5-6):517–535, 2012.

[8] ARM. Cortex-a76ae. `http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.101392_0000_01_en/index.html`, 2018. Accessed 10 June. 2020.

[9] Thomas Kottke and Andreas Steininger. A reconfigurable generic dual-core architecture. In *International Conference on Dependable Systems and Networks (DSN'06)*, pages 45–54. IEEE, 2006.

[10] Albert MK Cheng, Guangli Dai, Pavan Kumar Paluri, Mansoor Ansari, Darrel Knape, and Yu Li. Fault-tolerant regularity-based real-time virtual resources. In *2019 IEEE 25th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 1–12. IEEE, 2019.

[11] Jongsoo Hyun and Kyong Hoon Kim. Fault-tolerant scheduling in hierarchical real-time scheduling framework. In *2012 IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 431–436. IEEE, 2012.

[12] Hyun-Wook Jin. Fault-tolerant hierarchical real-time scheduling with backup partitions on single processor. *ACM SIGBED Review*, 10(4):25–28, 2013.

[13] Jongsoo Hyun, Sungshin Lim, Yeodeuk Park, Kun Su Yoon, Jae Hyun Park, Byung Moon Hwang, and Kyong Hoon Kim. A fault-tolerant temporal partitioning scheme for safety-critical mission computers. In *2012 IEEE/AIAA 31st Digital Avionics Systems Conference (DASC)*, pages 6C3–1. IEEE, 2012.

[14] Guy Martin Tchamgoue, Junho Seo, Jongsoo Hyun, Kyong Hoon Kim, and Yong-Kee Jun. Supporting fault-tolerance in a compositional real-time scheduling framework. *ACM SIGBED Review*, 12(2):7–15, 2015.

[15] Pengcheng Huang, Hoeseok Yang, and Lothar Thiele. On the scheduling of fault-tolerant mixed-criticality systems. In *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2014.

[16] Bowen Zheng, Yue Gao, Qi Zhu, and Sandeep Gupta. Analysis and optimization of soft error tolerance strategies for real-time systems. In *2015 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*, pages 55–64. IEEE, 2015.

[17] Yue Gao, Sandeep K Gupta, and Melvin A Breuer. Using explicit output comparisons for fault tolerant scheduling (fts) on modern high-performance processors. In *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 927–932. IEEE, 2013.

[18] Georg von der Brüggen, Kuan-Hsun Chen, Wen-Hung Huang, and Jian-Jia Chen. Systems with dynamic real-time guarantees in uncertain and faulty execution environments. In *2016 IEEE Real-Time Systems Symposium (RTSS)*, pages 303–314. IEEE, 2016.

[19] ISO26262 ISO. 26262: Road vehicles-functional safety. *International Standard ISO/FDIS*, 26262, 2011.

[20] Nahmsuk Oh, Philip P Shirvani, and Edward J McCluskey. Control-flow checking by software signatures. *IEEE transactions on Reliability*, 51(1):111–122, 2002.

[21] Emmanuel Touloupis, James A Flint, Vassilios A Chouliaras, and David D Ward. A fault-tolerant processor core architecture for safety-critical automotive applications. *SAE transactions*, pages 1–6, 2005.

[22] Abhishek Rhisheekesan, Reiley Jeyapaul, and Aviral Shrivastava. Control flow checking or not?(for soft errors). *ACM Transactions on Embedded Computing Systems (TECS)*, 18(1):1–25, 2019.

[23] Ramtilak Vemu and Jacob Abraham. Ceda: Control-flow error detection using assertions. *IEEE Transactions on Computers*, 60(9):1233–1245, 2011.

[24] Simon Kramer, Dirk Ziegenbein, and Arne Hamann. Real world automotive benchmarks for free. In *6th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, 2015.

[25] Arne Hamann, Dakshina Dasari, Simon Kramer, Michael Pressler, and Falk Wurst. Communication centric design in complex automotive embedded systems. In *29th Euromicro Conference on Real-Time Systems (ECRTS 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[26] Insik Shin and Insup Lee. Periodic resource model for compositional real-time guarantees. In *RTSS 2003. 24th IEEE Real-Time Systems Symposium, 2003*, pages 2–13. IEEE, 2003.

[27] John Lehoczky, Lui Sha, and Yuqin Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *RTSS*, volume 89, pages 166–171, 1989.

[28] Enrico Bini and Giorgio C Buttazzo. The space of rate monotonic schedulability. In *23rd IEEE Real-Time Systems Symposium, 2002. RTSS 2002.*, pages 169–178. IEEE, 2002.

[29] Alan Burns, Robert Davis, and Sasikumar Punnekkat. Feasibility analysis of fault-tolerant real-time task sets. In *Proceedings of the Eighth Euromicro Workshop on Real-Time Systems*, pages 29–33. IEEE, 1996.

[30] Sisu Xi, Justin Wilson, Chenyang Lu, and Christopher Gill. Rt-xen: Towards real-time hypervisor scheduling in xen. In *2011 Proceedings of the Ninth ACM International Conference on Embedded Software (EMSOFT)*, pages 39–48. IEEE, 2011.

[31] Lui Raymond Sha, John P Lehoczky, and Ragunathan Rajkumar. Solutions for some practical problems in prioritized preemptive scheduling. In *Unknown Host Publication Title*, pages 181–191. IEEE, 1986.

[32] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. *ACM SIGOPS operating systems review*, 37(5):164–177, 2003.

[33] Chung Laung Liu and James W Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61, 1973.

[34] David B Stewart and Pradeep K Khosla. Real-time scheduling of sensor-based control systems. *IFAC Proceedings Volumes*, 24(2):139–144, 1991.

[35] Almut Burchard, Jörg Liebeherr, Yingfeng Oh, and Sang Hyuk Son. New strategies for assigning real-time tasks to multiprocessor systems. *IEEE transactions on computers*, 44(12):1429–1442, 1995.

[36] Enrico Bini and Giorgio C Buttazzo. Schedulability analysis of periodic fixed priority systems. *IEEE Transactions on Computers*, 53(11):1462–1473, 2004.

[37] Dimitris Gizopoulos, Mihalis Psarakis, Sarita V Adve, Pradeep Ramachandran, Siva Kumar Sastry Hari, Daniel Sorin, Albert Meixner, Arijit Biswas, and Xavier Vera. Architectures for online error detection and recovery in multicore processors. In *2011 Design, Automation & Test in Europe*, pages 1–6. IEEE, 2011.

[38] Enrico Bini and Giorgio C Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2):129–154, 2005.

[39] Sanjoy K Baruah, Alan Burns, and Robert I Davis. Response-time analysis for mixed criticality systems. In *2011 IEEE 32nd Real-Time Systems Symposium*, pages 34–43. IEEE, 2011.

[40] Christian El Salloum, Andreas Steininger, Peter Tummeltshammer, and Werner Harter. Recovery mechanisms for dual core architectures. In *2006 21st IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pages 380–388. IEEE, 2006.

[41] Carles Hernandez and Jaume Abella. Timely error detection for effective recovery in light-lockstep automotive systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(11):1718–1729, 2015.

[42] Xabier Iturbe, Balaji Venu, Emre Ozer, Jean-Luc Poupat, Gregoire Gimenez, and Hans-Ulrich Zurek. The arm triple core lock-step (tcls) processor. *ACM Transactions on Computer Systems (TOCS)*, 36(3):1–30, 2019.

[43] Xabier Iturbe, Balaji Venu, and Emre Ozer. Soft error vulnerability assessment of the real-time safety-related arm cortex-r5 cpu. In *2016 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 91–96. IEEE, 2016.

[44] Ádria Barros de Oliveira, Gennaro Severino Rodrigues, Fernanda Lima Kastensmidt, Nemitala Added, Eduardo LA Macchione, Vitor AP Aguiar, Nilberto H Medina, and Marcilei AG Silveira. Lockstep dual-core arm a9: Implementation and resilience analysis under heavy ion-induced soft errors. *IEEE Transactions on Nuclear Science*, 65(8):1783–1790, 2018.

# 요약(국문초록)

본 논문에서는 효율적인 재구성가능 시스템 사용을 위한 계층기반 실시간 결함 감내 스케줄링 기법을 제안한다. 본 연구는 주기 자원 모델을 기반으로, 최적 주기 자원 서버의 용량을 주기 자원 모델이 가지는 실시간 주기 태스크 셋의 유틸라이제이션의 합으로 제시한다. 본 논문은 해당 최적 서버 용량을 시스템이 정상 동작할때와 오동작 할때 모두에 대해서 제시한다. 다음으로, 비중요 태스크 셋들을 중요 주기 자원 서버의 여분 공백 시간을 활용해 서버 용량의 증가 없이 비중요 태스크를 중요 주기 자원 서버에 할당하는 방법론을 제시한다. 마지막으로 본 논문은 주기 자원 서버 단위의 파티션 기법과 주기 태스크를 하나의 주기 자원 서버로 만드는 빈패킹 휴리스틱 알고리즘을 제시한다. 실험 결과, 본 논문에서 제시한 알고리즘은 기존에 사용되었던 파티션 기반 우선순위 스케줄링 알고리즘과 파티션 기반 우선순위 혼잡 중요도 알고리즘보다 더 작은 수의 코어의 개수를 도출 할 수 있음을 보인다. 실험결과를 기반으로, 본 연구에서 제안한 알고리즘을 재구성가능 시스템에 활용한다면 기존 방법 대비 최대 18%의 코어절감효과를 기대할수 있다.

**주요어** : 주기 자원 모델, 실시간 결함 감내 스케줄링
**학 번** : 2018-26716