declrpg

DECLARATION

I certify that

(a) The work contained in this report has been done by me under the guidance of my supervisor.

(b) The work has not been submitted to any other Institute for any degree or diploma.

(c) I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.

(d) Whenever I have used materials (data, theoretical analysis, figures, and text) from other sources, I have given due credit to them by citing them in the text of the thesis and giving their details in the references. Further, I have taken permission from the copyright owners of the sources, whenever necessary.

Date:

Place: Roll No.

free-standing-units,overwrite-commands,space-before-unit,unit-optional-argument,use-xspace

Real-Time Scheduling of Periodic Tasks and Messages on Heterogeneous Processors and FlexRay Bus Systems

MTP-II report submitted

in partial fulfilment of requirements for the degree of

Master of Technology

in

Artificial Intelligence and Machine Learning Applications

by

B Trishaa

(19MT3AI25)

Under the supervision of

Prof. Arnab Sarkar

Centre of Excellence in Artificial Intelligence

Indian Institute of Technology Kharagpur

Spring Semester, 2023-24

Apr 30, 2024

CENTRE OF EXCELLENCE IN ARTIFICIAL INTELLIGENCE

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

KHARAGPUR - 721302, INDIA

CERTIFICATE

This is to certify that the project report entitled **Real-Time Scheduling of Periodic Tasks and Messages on Heterogeneous Processors and FlexRay Bus Systems** submitted by **B Trishaa** (Roll No. 19MT3AI25) to Indian Institute of Technology Kharagpur towards partial fulfilment of requirements for the degree of Master of Technology in Artificial Intelligence and Machine Learning Applications is a record of bonafide work carried out by her under my supervision and guidance during Spring Semester, 2023-24.

Prof. Arnab Sarkar
Centre of Excellence in Artificial Intelligence
Indian Institute of Technology Kharagpur

Date: Apr 30, 2024
Place: Kharagpur

Acknowledgements

Throughout the Masters Thesis Project I have received a great deal of support and assistance, without which this work would have been considerably more difficult.

I would like to start by expressing my gratitude to my supervisor, Professor Arnab Sarkar, whose knowledge and experience were crucial in developing the study topics and methodology. Your astute criticism encouraged me to improve my thoughts and raised the calibre of my work.

I want to express my appreciation to my mentor, Sreyashi Mukherjee ma'am, for her unwavering support and direction in making sure that my project was always on the proper track.

I also want to express my gratitude to my parents for their insightful advice and sympathetic ear. You are there for me at all times. Finally, without the help of my friends, who offered intriguing conversations and enjoyable diversion to take my mind off of my study, I would not have been able to finish this task.

Date: Apr 30, 2024 (B Trishaa)
Place: Kharagpur (19MT3AI25)

Abstract

Name of the student: **B Trishaa** Roll No: **19MT3AI25**
Degree for which submitted: **Master of Technology**
Department: **Centre of Excellence in Artificial Intelligence**
Title: **Real-Time Scheduling of Periodic Tasks and Messages on Heterogeneous Processors and FlexRay Bus Systems**
Supervisor: **Prof. Arnab Sarkar**
Month and year of thesis submission: **Apr 30, 2024**

In this thesis, we address the challenge of scheduling Periodic Tasks and Messages on heterogeneous processors and the FlexRay bus system, crucial for real-time systems. With the growing demand for higher performance and efficiency, heterogeneous platforms are increasingly prevalent. We propose a scheduling algorithm that aims to meet real-time task timing constraints while minimizing make span length. Building upon existing strategies for homogeneous multiprocessor systems, we extend the scope to heterogeneous environments. Our approach considers scheduling real-time systems modeled as Precedence-constrained Task Graphs (PTGs) on fully-connected heterogeneous systems. Additionally, we delve into the scheduling intricacies of the FlexRay protocol's static segment. Periodic signals must be packed into messages of equal size to comply with FlexRay protocol restrictions while optimizing bandwidth utilization. We formulate a nonlinear integer programming (NIP) problem to maximize bandwidth usage efficiently. By leveraging the constraints of the FlexRay protocol, we decompose the NIP and compute an optimal message set effectively. Our contributions offer insights and solutions for effectively managing real-time tasks and messages in complex, heterogeneous environments like FlexRay-based systems.

Contents

List of Figures

Chapter 1

Introduction

Cyber-Physical Systems (CPSs) are a crucial part of our daily lives. A CPS is composed of physical sub-systems together with computing and networking (cyber sub-systems), where embedded computers and networks monitor and control the physical processes.Like in a conventional aircraft, a pilot manages the aircraft by manipulating surfaces on the wings and tail, linked to the cockpit through mechanical and hydraulic systems. Conversely, in a fly-by-wire aircraft, control commands are electronically transmitted by a flight computer through a network to actuators located on the wings and tail. This design significantly reduces the weight of the aircraft compared to traditional models, leading to improved fuel efficiency.

Many CPSs in domains such as automotive, avionics, smart grids, nuclear plants, industrial process control, etc., often consist of multiple control sub-systems running on distributed processing platforms. For example, an automotive system consists of several distributed sub-systems like Adaptive Cruise Controller (ACC), Traction Controller (TC), Anti-lock Braking System (ABS), Advanced Driver Assistance System (ADAS), etc. Most of these control systems are modeled as real-time independent tasks or Precedence-constrained Task Graphs (PTGs), depending on the nature of interactions between their functional components.

In contemporary automotive systems, the proliferation of electronic devices such as microcontrollers, sensors, and actuators has supplanted traditional mechanical and hydraulic components. These electronic control units (ECUs) necessitate seamless information exchange to facilitate task execution. In modern luxury vehicles, the

intercommunication between up to 70 ECUs, exchanging approximately 2500 signals, underscores the complexity of in-vehicle networks.

To fulfill requirements concerning timing, reliability, energy efficiency, etc., and to maintain a straightforward design approach, these sub-systems often operate on dedicated processing units, resulting in a federated overall system architecture. These federated structures might cost more to design than running applications on simpler, compact integrated platforms. But, consolidated architectures can get more complicated due to increased competition for shared resources like processing elements, buses, and memories. At the same time, the ongoing need for better performance and reliability with limited resources is making a shift from homogeneous to heterogeneous processing platforms for today's Cyber-Physical Systems (CPSs).

In a distributed platform where processing elements are connected through communication channels, the effective completion of tasks and the sending of messages, meeting deadlines and other limitations, is essentially a challenge in coordinating real-time tasks and messages. Scientists have been researching the scheduling of real-time tasks on processors for many years. A scheduler assigns resources to tasks and determines their order of execution, considering constraints to to make sure everything runs smoothly.

Real-time tasks can fall into two categories: (i) Independent task set, where tasks don't rely on each other's results; their execution is not influenced by data from other tasks. (ii) Dependent task set, where a task's completion might depend on messages received from other tasks in the set. In this thesis, we consider independent task set. To schedule a set of independent tasks, a scheduler has to meet deadlines and resource limitations. Whereas, to plan a set of dependent tasks (represented as a PTG), a scheduler should also satisfy the order constraints among tasks.

A distributed platform includes several homogeneous or heterogeneous processors, connected in fully or through shared buses. Depending on the processor type, a task might need the same or different amounts of time when executed on different processors. Apart from figuring out when a task should start and finish, a scheduler also needs to decide on which processor each task should run on when creating a schedule for tasks on a heterogeneous distributed platform.

While the controller area network (CAN) serves as a prevalent communication network in automotive systems, its event-triggered nature and limited data rates pose challenges for emerging applications like x-by-wire, which demand periodic data exchange with minimal jitter. To address these demands, time-triggered technologies like time-triggered CAN (TTCAN), time-triggered protocol (TTP), and particularly FlexRay have been developed, offering predictable medium access and higher bandwidth capabilities.

FlexRay, envisioned as the new standard for in-vehicle communication, boasts a substantial bandwidth of 10 Mb/s and features a static segment (SS) with time-division multiple access (TDMA) operation, alongside a dynamic segment (DS) employing flexible TDMA (FTDMA) operation. This hybrid approach integrates the benefits of both time- and event-triggered communication paradigms.

Approaches to solving real-time scheduling problems generally fall into two categories: heuristic and optimal. Heuristic methods for creating schedules usually rely on meeting a set of minimum conditions and may not consider all the necessary requirements for scheduling. As a result, these scheduling methods tend to be less than optimal, and their outcomes frequently differ significantly from those of optimal approaches. But, optimal solution methods consider all necessary and sufficient conditions, potentially making a crucial impact on time-sensitive systems in terms of performance, reliability, and other non-functional aspects such as cost, power, and space. Optimal schedules can also serve as benchmarks, enabling precise comparisons and evaluations of heuristic solutions. Various approaches, such as modeling based on Constraint Satisfaction Problem (CSP), search-based techniques, and Integer Linear Programming (ILP), are commonly employed to create optimal schedulers for Cyber-Physical Systems (CPSs).

In this thesis, we focus on creating a heuristic co-scheduling strategy for real-time CPSs having independent tasks and messages represented as PTGs. The platform consists of heterogeneous processors with shared bus based system. This paper also focuses on the computation of message schedules for the static segment (SS) of FlexRay, designed to accommodate periodic real-time messages.

Chapter 2

Literature Review

2.1 Comprehensive Literature review

The development of scheduling algorithms for real-time systems has been an area of intense research due to the critical nature of these systems in various industries. This review highlights seminal works and recent advancements that contribute to our understanding of task scheduling in heterogeneous computing environments and communication systems like FlexRay.

2.1.1 Early Foundations in Real-Time Scheduling

The journey of real-time scheduling began with foundational work by **Liu and Layland (1973)**, who introduced Rate Monotonic (RM) and Earliest Deadline First (EDF) scheduling algorithms. These algorithms have set the stage for numerous studies focusing on homogeneous environments, establishing basic principles that are still influential today.

2.1.2 Transition to Heterogeneous Systems

The evolution of computing technologies has necessitated a shift to heterogeneous systems. **Baruah et al. (1996)** were pioneers in adapting traditional real-time

scheduling algorithms to these environments. Their work highlighted the importance of optimizing task allocation to maximize the diverse capabilities of heterogeneous processors.

2.1.3 Scheduling in Automotive Systems

With the advent of complex automotive systems, scheduling for communication protocols like FlexRay became crucial. **Bauer et al. (2008)** explored the specific challenges and opportunities of scheduling within the FlexRay protocol, focusing particularly on optimizing its static segment for better communication efficiency. **Klaus Schmidt and Ece Guran Schmidt (2008)** further advanced this field with their work on message scheduling for the FlexRay protocol, emphasizing the static segment's role in ensuring reliable communication.

2.1.4 Advanced Scheduling for Cyber-Physical Systems (CPS)

The landscape of real-time systems expanded with the emergence of Cyber-Physical Systems (CPS), integrating computational and physical processes. This integration brought about sophisticated scheduling strategies tailored to the unique demands of CPS, where both computation and communication play pivotal roles.

2.1.5 Directed Task Graphs (DTGs) and Optimal Scheduling

Recent advancements have seen significant efforts in scheduling Directed Task Graphs (DTGs), which are essential for modeling task dependencies in complex systems. **Liu et al.** developed methods to minimize makespan for DTGs on multicore clusters, introducing ILP-based techniques that accommodate tasks of varying quality levels on mixed platforms. **A. Sarkar, S.K. Roy, and R. Devaraj (2019)** made notable contributions by optimizing the scheduling of PTGs with multiple service levels on heterogeneous distributed systems, addressing the challenges of service level differentiation in scheduling.

2.1.6 Heuristic Approaches for DTGs

Efficient heuristics have been developed to tackle the complexity of optimally scheduling DTGs. These heuristics generally encompass clustering, duplication, and list scheduling:

Clustering reduces communication costs by grouping tasks with their predecessors.

- **Duplication** decreases waiting times by copying tasks onto multiple processors.

- **List scheduling** involves prioritizing tasks and selecting the optimal processor for execution, focusing on minimizing execution time.

2.1.7 Addressing Communication Contention

As DTGs are commonly used in networked embedded systems, addressing communication contention is crucial. **Sinnen and Sousa** and **Tang et al.** introduced methods that consider the scheduling of DTG edges onto the links of topology graphs, with special attention to bus-based and arbitrary processor networks. **A. Sarkar, Kankana Maji, Sayani Sinha, S. Roy, and R. Devaraj (2020)** expanded on this by developing contention-aware scheduling strategies for real-time precedence-constrained task graphs on heterogeneous systems. In 2022, **A. Sarkar, S.K. Roy, and R. Devaraj** introduced contention cognizant scheduling for task graphs on shared bus-based heterogeneous platforms, highlighting the importance of accounting for shared communication resources in scheduling decisions.

2.1.8 Co-Scheduling Strategies

R. Gangopadhyay, S.K. Roy, and A. Sarkar (2021) explored processor and bus co-scheduling strategies for real-time tasks with multiple service levels, proposing innovative methods to synchronize task and communication scheduling to enhance system performance.

2.1.9 Conclusion

The literature review encapsulates the progression from foundational scheduling algorithms to sophisticated approaches that address the nuanced requirements of modern heterogeneous systems and advanced communication protocols. This body of work informs the current research, which seeks to enhance scheduling strategies by integrating advanced mathematical models and innovative scheduling techniques to better manage the complexities of real-time, heterogeneous environments. These recent studies particularly underscore the evolving nature of task and message scheduling, where differentiation in service levels and communication contention plays a critical role in the design of efficient and reliable systems.

2.2 Problem Statement and Objectives

Modern real-time systems, especially those embedded within automotive and aerospace sectors, demand increasingly sophisticated scheduling solutions due to their reliance on heterogeneous processors and complex communication protocols like FlexRay. These systems must execute a multitude of periodic tasks and messages within stringent timing constraints, necessitating efficient use of computing and communication resources. However, existing scheduling algorithms often fall short in effectively managing the unique challenges posed by heterogeneous environments and the specific requirements of communication protocols. This is particularly evident in scenarios where tasks and messages must be coordinated across diverse processing elements and communication channels, often leading to suboptimal performance and resource underutilization.

2.2.1 Research Objectives

The primary aim of this research is to develop an advanced scheduling algorithm that optimally manages both tasks and messages within real-time systems equipped with heterogeneous processors and interfaced with FlexRay bus systems. In this thesis, we introduce an algorithm that minimizes the execution time of a single job on a set of P processors. We assume P processors are available for the job, and they are not shared during job execution. Therefore, with system and job parameters known at compile time, a static scheduling approach incurs no overhead at runtime, making it more suitable for this scenario. The specific objectives are:

Develop a Comprehensive Scheduling Model:
To create a scheduling framework that incorporates both the computational and communication aspects of real-time systems. To address the unique challenges of heterogeneous processors and the FlexRay communication protocol, ensuring that both task execution and message transmission are optimized.

Formulate and Solve the Scheduling Problem:
To formulate the scheduling problem as a nonlinear integer programming (NIP)

problem that captures the complexities of task dependencies, processor capabilities, and communication constraints. To develop an effective solution approach that can find optimal or near-optimal schedules in terms of minimizing the makespan and maximizing resource utilization.

Enhance FlexRay Communication Efficiency:
To specifically address the scheduling intricacies of the FlexRay protocol's static segment. To optimize the packing of periodic signals into messages according to the protocol's stringent requirements, thereby improving bandwidth utilization and reducing communication latency.

Implementation of the Proposed Algorithm:
To implement the proposed scheduling algorithm.To compare the algorithm's effectiveness against existing scheduling methods and adherence to real-time constraints.

Provide Insights and Tools for Real-World Application:
To offer practical insights that can assist system designers and engineers in effectively managing real-time tasks and messages in heterogeneous and communication-intensive environments. To develop a toolset or software that can be readily used in industry to implement the proposed scheduling strategies.

The objective is to minimize the overall completion time or makespan. The task scheduling problem for heterogeneous systems is more complex than that for homogeneous computing systems because of the different execution rates among processors and possibly different communication rates among different processors. A popular representation of an application is the directed acyclic graph (DAG), which includes the characteristics of an application program such as the execution time of tasks, the data size to communicate between tasks and task dependences.

By accomplishing these objectives, the research aims to significantly advance the field of real-time system scheduling, providing robust solutions that enhance the performance and reliability of critical systems in dynamic and heterogeneous environments.

Chapter 3

MTP-I Scheduling Problem Formulation

The PTG scheduling techniques assumes a fully connected heterogeneous platform. Considering the assumption of a fully connected platform is advantageous in mitigating the issue of contention for communication resources, particularly when the system is presumed to have shared data transmission channels. It's worth noting that shared bus networks are commonly used in communication architectures within Cyber-Physical Systems (CPSs). Consequently, this chapter introduces the design of ILP-based optimal scheduling strategies for co-scheduling real-time PTGs. This involves their execution on a distributed platform, comprising a set of heterogeneous processing elements interconnected by heterogeneous shared buses.

3.1 System Model

The system model associated with this work is presented by describing the platform, computation model and assumptions.

Platform: Figure 3.1 shows the pictorial representation of the *heterogeneous multiprocessor platform* ρ as considered here. The platform is composed of a resource set $\{R_1, R_2, \ldots, R_{p+b}\}$ among which, $\{R_1, R_2, \ldots, R_p\}$ denote a set $P = \{P_1, P_2, \ldots, P_p\}$ of p heterogeneous processors; whereas, resources $\{R_{p+1}, R_{p+2}, \ldots, R_{p+b}\}$ represent

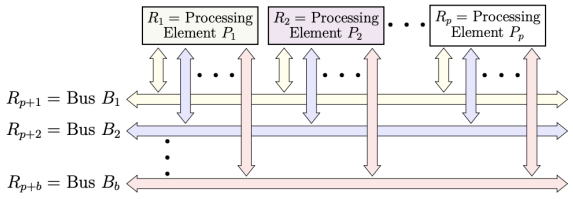


FIGURE 3.1: *Platform Model* ρ

a set $B = \{B_1, B_2, \ldots, B_b\}$ of b heterogeneous shared buses. Each bus $B_r \in B$ is connected to all processors in P.

Computation Model: A Cyber-Physical System (CPS) application as considered in this work is represented by a PTG $G = (\mathcal{V}, \mathcal{E}, ET, CT)$ where, - $\mathcal{V} = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_{n+m}\}$ represents the node. Among them, $\{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_n\}$ represent a set $T = \{T_1, T_2, \ldots, T_n\}$ of n tasks. Similarly, $\{\mathcal{V}_{n+1}, \mathcal{V}_{n+2}, \ldots, \mathcal{V}_{n+m}\}$ denotes a set $M = \{M_1, M_2, \ldots, M_m\}$ of m messages which specify communication demand between task pairs. - $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ denotes the edge set describing dependency-constraints between pairs of nodes in $\mathcal{V}$. - ET is an execution-time matrix of size $n \times p$. Each element $e_{ir} \in ET$ captures the execution demand of task T_i (vertex $\mathcal{V}_i$) on processor P_r (resource R_r). - CT is a communication-time matrix of size $m \times b$. Each element $c_{kr} \in CT$ captures the communication time associated with message M_k (vertex $\mathcal{V}_{n+k}$) on bus B_r (resource R_{p+r}).

The Assumptions:

1. PTG G has a single entry (source) node T_1 (having in-degree 0) and a single exit (sink) node T_n (having out-degree 0). If the input PTG contains multiple source/sink nodes, we add a single dummy source/sink task node (having execution time 0 on all processors) which connect to all original source/sink task nodes via dummy edges. Dummy message nodes (having transmission time 0 for all buses) are added to each of these new edges.

2. The entry (T_1) and exit (T_n) nodes are both tasks.

3. Any task T_i is preceded (except T_1)/succeeded (except T_n) by message node(s).

4. Any message M_k is preceded/succeeded by one task node only.

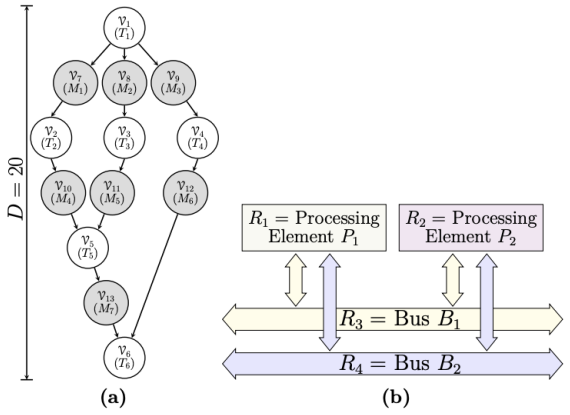


FIGURE 3.2: (a) *Example of a PTG G and* (b) *Model of Platform* ρ

5. The communication overhead associated with a message node Mk is assumed to be negligible ($\forall B_r \in B, c_{kr} = 0$) when its preceding as well as succeeding task nodes are assigned to the same processor.

Example: Figure 3.2a shows an example of a PTG G which consists of 13 nodes $\{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_{13}\}$. Among them, $\{\mathcal{V}_1, \ldots, \mathcal{V}_6\}$ are tasks and $\{\mathcal{V}_7, \ldots, \mathcal{V}_{13}\}$ are messages. Hence, n = 6 and m = 7. Figure 3.2b shows a sample *platform model* ρ consisting of four resources $R = \{R_1, R_2, R_3, R_4\}$. Among them R_1, R_2 denote two processors P_1, P_2 and R_3, R_4 denote two buses B_1, B_2. Thus, p = 2, b = 2. In Table 3.1, we show the execution time matrix ET. An element say $e_{1,1} = 4$ in this matrix specifies that task T_1 (corresponding to vertex $\mathcal{V}_1$) takes 4 units of time to finish execution on processor P_1. In the same way, Table 3.2 shows matrix CT, depicting communication time. An element say $c_{1,1} = 2$ in this matrix specifies that message M_1 (corresponding to vertex $\mathcal{V}_7$) takes 2 units of time for transmission over bus B_1 (assigned on resource R_3).

Problem Definition: Given a real-time PTG $G = (\mathcal{V}, \mathcal{E}, ET, CT)$ with end-to-end deadline D to be executed on a heterogeneous platform consisting of p processors and b buses, determine the start times for all task and message nodes, task-to-processor and message-to-bus assignments, with the objective of minimizing the *overall makespan* (i.e., schedule length) and meets the deadline D.

	T_1	T_2	T_3	T_4	T_5	T_6
P_1	4	8	3	2	4	2
P_2	3	5	4	3	2	3

FIGURE 3.3: Table 3.1: *Execution time Matrix ET of task nodes*

	M_1	M_2	M_3	M_4	M_5	M_6	M_7
B_1	2	4	5	3	3	1	3
B_2	3	3	3	4	2	3	2

FIGURE 3.4: Table 3.2: *Communication time Matrix CT of message nodes*

3.2 Earliest Start Times and Earliest Finish Time

Let, t_i^s and t_i^l be the earliest and latest time steps at which node $\mathcal{V}_i$ may start its execution. These upper and lower bounds on start times are determined separately for the task and message nodes in the given PTG. The t_i^s (ASAP time) and t_i^l (ALAP time) values for task nodes are computed as follows.

ASAP/ALAP computation procedure for task nodes: 1. We ignore message nodes in the PTG and assume directed edges between the predecessor and successor task nodes of each message node. 2. Set ASAP time of the source task node as: $t_1^s = 1$. 3. Set ALAP time for the sink task node as,

$$t_n^l = D - \min_{r \in [1,p]} e_{nr} + 1$$

4. ASAP times for the remaining task nodes (except T_1) are recursively determined (downward) as follows:

$$t_i^s = \max_{T_j \in \text{pred}(T_i)} \left(t_j^s + \min_{r \in [1,p]} e_{jr} \right)$$

	T_1	T_2	T_3	T_4	T_5	T_6	M_1	M_2	M_3	M_4	M_5	M_6	M_7
ASAP	1	4	4	4	7	9	4	4	4	9	7	6	9
ALAP	9	12	14	17	17	19	10	11	14	14	15	18	17

FIGURE 3.5: Table 3.3: *ASAP and ALAP times of nodes in G (Figure 3.2a)*

where, pred (T_i) is the set of predecessors of task node T_i. 5. ALAP times for the remaining task nodes (except T_n) are recursively defined (upward) as follows:

$$t_i^l = \min_{T_j \in \text{succ}(T_i)} \left(t_j^l - \min_{r \in [1,p]} e_{ir} \right)$$

where, succ (T_i) is the set of successors of task node T_i. Given the ASAP/ALAP times for task nodes in the PTG, we now compute these values for message nodes.

ASAP/ALAP computation procedure for message nodes: 1. The ASAP time of a message node M_k is defined as:

$$t_{n+k}^s = t_i^s + \min_{r \in [1,p]} e_{ir}$$

where, T_i (i.e., vertex $\mathcal{V}_i$) is the predecessor task node of M_k (i.e., vertex $\mathcal{V}_{n+k}$). 2. Similarly, ALAP time of a node M_k is defined as follows:

$$t_{n+k}^l = t_j^l - \min_{r \in [1,b]} c_{kr}$$

where, T_j is the successor task node of M_k.

Example (contd.): Let us assume the deadline D of PTG G (in Figure 3.2 a) to be 20 time units. Table 3.3 shows the ASAP and ALAP times corresponding to each task and message node in G, obtained through the above discussed procedure. For example, ASAP and ALAP times of task node T_1 are 1 and 9 , respectively.

3.3 ILP Formualtion

In this section, we present an ILP based solution to the PTG scheduling problem. First, let us consider a set of binary decision variables: $X = \{X_{irt} : i = 1, 2, \ldots, n+m; r = 1, 2, \ldots, p+b; t = 1, 2, \ldots, D\}$. The variable $X_{irt} = 1$, when the PTG node $\mathcal{V}_i$ (i.e.,

task node T_i/ message node M_{i-n}) starts on the resource R_r (i.e., processing element P_r/ bus B_{r-p}) at the t^{th} time step; $X_{irt} = 0$, otherwise. We now present the required constraints on the binary variables X to model the scheduling problem.

3.3.1 Unique Start Time Constraints

The start time of each task node should be unique. That is, each task node T_i must start its execution at a unique time step t on a distinct processing element P_r.

$$\forall i \in [1, n] \quad \sum_{r=1}^{p} \sum_{t=t_i^s}^{t_i^l} X_{irt} = 1 \tag{3.1}$$

Similarly, each message node M_k must have a unique start time if M_k is actually transmitted over a bus (refer Assumption 5). So, the following constraint must hold:

$$\begin{aligned} \forall M_k \mid T_i = \text{pred}\,(M_k) \text{ and } T_j = \text{succ}\,(M_k)\,, \\ \sum_{r=p+1}^{p+b} \sum_{t=t_{k'}}^{t_{k'}^l} X_{k'rt} = 1 - Y_k \end{aligned} \tag{3.2}$$

where,

$$k' = n + k \text{ and } Y_k = \sum_{r=1}^{p} \sum_{t_1=t_i^s}^{t_i^l} \sum_{t_2=t_j^s}^{t_j^l} X_{irt_1} * X_{jrt_2}$$

It may noted that in the above equation, $Y_k = 1$ when both the predecessor (T_i) and successor (T_j) task nodes of message node M_k are assigned to the same processing element P_r, forcing the LHS of Equation 3.2 to become 0. Otherwise, $Y_k = 0$. As X_{irt_1} and X_{jrt_2} are binary decision variables, we linearize their multiplication by introducing another binary decision variable $U_{krt_1t_2}$ ($= X_{irt_1} * X_{jrt_2}$) as shown below:

$$Y_k = \sum_{r=1}^{p} \sum_{t_1=t_i^s}^{t_i^l} \sum_{t_2=t_j^s}^{t_j^l} U_{krt_1t_2} \tag{3.3}$$

3.3.2 Linearization of Non-linear Term

Now, the non-linear variables $U_{krt_1t_2}$ can be linearized using the following four inequalities.

$$X_{irt_1} \geqslant U_{krt_1t_2} \tag{3.4}$$

$$X_{jrt_2} \geqslant U_{krt_1t_2} \tag{3.5}$$

$$U_{krt_1t_2} \geqslant X_{irt_1} + X_{jrt_2} - 1 \tag{3.6}$$

$$U_{krt_1t_2} \in \{0, 1\} \tag{3.7}$$

3.3.3 Resource Constraints

Resource bounds must be satisfied at each time step for both processing elements and buses. Any resource R_r can execute/transmit at most one task/message node at a given time. In this regard, it may be noted that a task node T_i can only be executing on processing element P_r at time t, if it has started at most $t - e_{ir} + 1$ time steps earlier.

$$\forall t \in [1, D] \text{ and } \forall r \in [1, p] \quad \sum_{i=1}^{n} \sum_{t'=\psi}^{t} X_{irt'} \leqslant 1 \tag{3.8}$$

where, $\psi = t - e_{ir} + 1$. Similarly, a message node M_k can only be transmitting through the bus B_r at time t, if it has started at most $t - c_{kr} + 1$ time steps earlier. The range of t is

$$\forall t \in [1, D] \text{ and } \forall r \in [1, b] \quad \sum_{k=1}^{m} \sum_{t'=\psi}^{t} X_{k'r't'} \leqslant 1 \tag{3.9}$$

where, $k' = k + n, r' = r + p$ and $\psi = t - c_{kr} + 1$.

3.3.4 Dependency Constraints

The dependencies between nodes must be satisfied. The following three constraints enforce satisfaction of the precedence relationships among task and message nodes of a PTG. Constraints 3.10 and 3.11 assert that the preceding task node (say, T_i) of any message node (say, M_k) completes its execution (i) before the start of the succeeding task node (say, T_j) of M_k (in case, both T_i and T_j are assigned to the same processing element) and, (ii) before the start of M_k (in case, both T_i and T_j are assigned to different processing elements).

$$\forall M_k \mid T_i = \text{pred}\,(M_k) \text{ and } T_j = \text{succ}\,(M_k)\,,$$

$$\sum_{r=1}^{p} \sum_{t=t_i^s}^{t_i^l} (t + e_{ir}) * X_{irt} \leqslant \sum_{r=1}^{p} \sum_{t=t_j^s}^{t_j^l} t * X_{jrt} \tag{3.10}$$

$$\sum_{r=1}^{p} \sum_{t=t_i^s}^{t_i^l} (t + e_{ir}) * X_{irt} \leqslant \sum_{r=p+1}^{p+b} \sum_{t=t_{k'}^s}^{t_{k'}^l} t * X_{k'rt} + C * Y_k \tag{3.11}$$

where, $k' = n + k$ and C is a constant. It may be observed that by setting C to a sufficiently large value, the constraint in Equation 3.11 is trivially satisfied when both T_i and T_j are assigned to the same processing element ($Y_k = 1$). Suppose, the M_k^{th} message node is scheduled on a bus (i.e., $Y_k = 0$). Then, task node T_j (= succ (M_k)) should commence its execution only after the completion of M_k. This constraint is represented as follows:

$$\forall M_k \mid T_j = \text{succ}\,(M_k)$$

$$\sum_{r=p+1}^{p+b} \sum_{t=t_{k'}^s}^{t_{k'}^l} (t + c_{kr}) * X_{k'rt} \leqslant \sum_{r=1}^{p} \sum_{t=t_j^s}^{t_j^l} t * X_{jrt} \tag{3.12}$$

where, $k' = n + k$. It is noteworthy that when $Y_k = 1$, the constraint imposed by Equation 6.2 enforces $\sum_{r=p+1}^{p+b} \sum_{t=k_{k'}^s}^{t_{k'}^l} X_{k'rt}$ to be 0. Hence, $\sum_{r=p+1}^{p+b} \sum_{t=t_{k'}^s}^{t_{k'}^l} (t + c_{kr}) * X_{k'rt}$ in the LHS of Equation 6.12 also reduces to 0. So, Constraint 6.12 is implicitly satisfied, when Y_k is 1 .

3.3.5 Deadline Constraint

All tasks in the PTG have to complete their execution within the deadline D. This can be satisfied by restricting the finish time of the sink node to be at most the deadline D.

This constraint can be written as,

$$\sum_{r=1}^{p} \sum_{t=t_n^s}^{t_n^l} X_{nrt} * (t + e_{nr}) - 1 \leq D \tag{3.13}$$

3.3.6 Objective Function

Our objective is to minimize the schedule length of PTG G. It can be achieved by minimizing the finish time of the sink node T_n. Hence, the objective function can be written as:

$$\text{Minimize} \sum_{r=1}^{p} \sum_{t=t_n^s}^{t_n^l} X_{nrt} * (t + e_{nr}) \tag{3.14}$$

subject to constraints presented in Equations 3.1 - 3.13.

3.4 Experimental Results and Discussions

To evaluate the relative performance of the heuristics, we first considered randomly generated application graphs. For this purpose, we used a synthetic DAG generation program available at [11]. In this thesis, we used this synthetic DAG generator to create the DAG structure, which includes the specific number of nodes and their dependencies.

We have generated a PTG with 55 nodes having 27 tasks and 28 messages with an average computational cost = 20 and height = 8.

```
Graph # 1
n=70,Average Computational Cost=20, height=8


width of level 1=11
width of level 2=2
width of level 3=13
width of level 4=10
width of level 5=2
width of level 6=3
width of level 7=8
width of level 8=6

Final number of vertices=55
```

FIGURE 3.6: *Random task graph generated*

Computation matrix is

---------	*Pr1*	*Pr2*	*Pr3*	*Pr4*	*Pr5*
Task # 1	20	11	25	30	9
Task # 2	15	30	6	15	37
Task # 3	12	31	4	4	35
Task # 4	22	21	32	31	10
Task # 5	21	4	36	17	17
Task # 6	32	22	17	40	20
Task # 7	3	29	21	14	25
Task # 8	36	11	30	30	7
Task # 9	16	18	25	11	25
Task # 10	14	4	19	22	30
Task # 11	6	8	19	20	41
Task # 12	36	12	11	8	26
Task # 13	9	6	10	5	23
Task # 14	3	14	35	9	15
Task # 15	17	28	21	24	39
Task # 16	32	30	22	38	33
Task # 17	8	30	35	7	31
Task # 18	28	7	31	22	9
Task # 19	22	37	5	30	15
Task # 20	9	26	5	21	11
Task # 21	11	39	40	10	17
Task # 22	21	38	18	34	32
Task # 23	10	31	34	3	26
Task # 24	26	36	19	14	10
Task # 25	26	23	41	6	38
Task # 26	23	11	13	35	16
Task # 27	26	24	12	38	4

FIGURE 3.7: *Computation Matrix*

```
        PROCESS 11

EST     0       0       0       0       0
EFT     6       8       19      20      41
Actual Finish Time:     6.0
Processor Selected:     1
Processor State: 6      0
```

FIGURE 3.8: *Example for T2*

Communictaion matrix is

--From / To---	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15	#16	#17	#18	#19	#20	#21	#22	#23	#24	#25	#26	#27
From Task # 1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	6	6	-1	-1	2	20	-1	-1	18	-1	-1	1	8	7	11	-1
From Task # 2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	18	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	16	-1	8	-1
From Task # 3	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	20	-1	-1	-1	-1	-1	-1	-1	5	12	1	-1	-1	9
From Task # 4	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	3	7	-1	-1	1	-1	-1	-1	10	19	2	20	-1	2	-1
From Task # 5	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	16	-1	-1	-1	-1	-1	-1	10	-1	-1	-1	-1	-1	-1	18
From Task # 6	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	8	-1	17	-1	14	4	5	-1	-1	-1	-1
From Task # 7	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	4	-1	-1	-1	12	16	6	-1	16	-1	12	20
From Task # 8	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	11	-1	17	9	9	-1	-1	12	1	5	8	15	-1	-1	12
From Task # 9	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	2	-1	-1	19	-1	-1	5	-1	9	4	-1	18	-1	12
From Task # 10	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	9	18	20	-1	6	17	4	-1
From Task # 11	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	20	9	4	3	2	9	-1	-1	-1	7	5	12	11	5	11
From Task # 12	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	6	-1	7	-1	-1	-1	-1	-1	17	-1	6	-1	-1	-1
From Task # 13	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	18	-1	17	12	20	17	-1	-1	5	-1	4	2	17	-1
From Task # 14	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	9
From Task # 15	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	7
From Task # 16	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	5
From Task # 17	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	8
From Task # 18	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	19
From Task # 19	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	13
From Task # 20	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	11
From Task # 21	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	2
From Task # 22	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	16
From Task # 23	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	5
From Task # 24	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	8
From Task # 25	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	11
From Task # 26	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	5
From Task # 27	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

FIGURE 3.9: *Communication Matrix*

```
T2 T5 T11 T8 T4 T10 T12 T13 T9 T3 T7 T6 T20 T25 T24 T17 T16 T23 T26
```

FIGURE 3.10: *Task schedule created*

Chapter 4

Scheduling Problem Formulation

This chapter delves into the formulation of the scheduling problem for real-time periodic task graphs (PTGs) in a heterogeneous computing environment, specifically focusing on systems interconnected by heterogeneous shared buses. Given the computational and communication requirements of Cyber-Physical Systems (CPSs), our goal is to optimally co-schedule tasks and messages to minimize overall makespan while adhering to real-time constraints.

4.1 System Model

The system model under consideration comprises a heterogeneous multiprocessor platform, depicted as ρ. The platform includes both processors and buses, modeled as:

- **Processors (P):** Represented as a set $P = P_1, P_2, ..., P_p$ of p heterogeneous processors, each differing in capabilities and performance characteristics.
- **Buses (B):** Defined as a set $B = B_1, B_2, ..., B_b$ of b heterogeneous shared buses, facilitating communication between processors.

The computational tasks and communications within the system are structured into a PTG $G = (\mathcal{V}, \mathcal{E}, ET, CT)$, where:

- **Nodes ($\mathcal{V}$):** Comprise n tasks and m messages, structured as $\mathcal{V} = \mathcal{V}_1, \mathcal{V}2, ..., \mathcal{V}n + m$.
- **Edges ($\mathcal{E}$):** Represent dependencies between tasks and messages.
- **Execution Time Matrix (ET):** An $n \times p$ matrix defining the execution times of tasks on each processor.
- **Communication Time Matrix (CT):** An $m \times b$ matrix detailing the transmission times of messages over each bus.

Each task Ti has both an input message Mi received before the execution of Ti and an output message $Mi^{'}$ sent after the completion of Ti. Let, T^kij denote the kth instance of task Ti , with mx^kij being its input message and my^kij the output message. The designed algorithm must guarantee that mx^kij (from a sensor) is transmitted and received during the previous period so that the message is available as input to the task at the beginning of the current period. Similarly, the output message my^kij produced by T^kij in the current period should be transmitted over the bus to designated actuators during the next period. This pipelined design criterion ensures the completion of one task execution every period, as illustrated below.

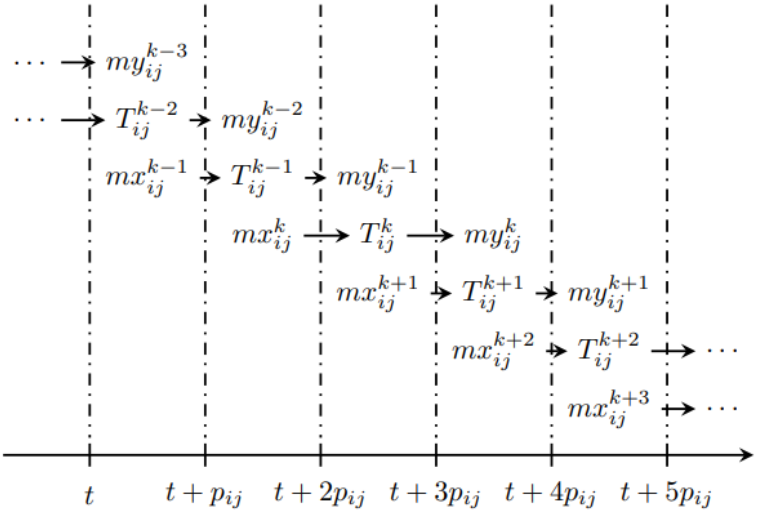


FIGURE 4.1: *3-Stage Pipeline structure*

4.2 Assumptions

The PTG includes designated source (entry) and sink (exit) nodes, ensuring a directed flow from a single entry to a single exit, facilitating clear scheduling start and end points. Message nodes (M_k) assume negligible communication overhead when their linked tasks are assigned to the same processor, simplifying the scheduling process under certain conditions.

4.3 Problem Definition

The scheduling problem aims to determine the optimal assignment of tasks to processors and messages to buses, alongside their respective start times, such that the overall makespan is minimized and all real-time constraints are satisfied.

4.4 ILP Formulation for Scheduling

4.4.1 Variables and Constraints

- **Binary Decision Variables (X_{irt}):** Indicates whether a task/message node starts at time t on resource r.
- **Unique Start Time Constraints:** Ensures each task and message begins at a distinct time on a specific processor or bus.
- **Resource Constraints:** Guarantees that each processor or bus does not handle more than one task or message at any given time.
- **Dependency Constraints:** Maintains the precedence relationships among tasks and messages, ensuring correct execution order.

4.4.2 Objective Function

The primary objective is to minimize the finish time of the sink task, effectively reducing the schedule length. This involves minimizing the latest finish time among all tasks, represented mathematically as:

$$\text{Minimize} \sum_{r=1}^{p} \sum_{t=t_n^s}^{t_n^l} X_{nrt} \cdot (t + e_{nr}) \tag{4.1}$$

where e_{nr} is the execution time of the sink task on processor r.

4.5 Implementation in C++

The C++ implementation provided facilitates the simulation of this scheduling approach on a hypothetical platform configuration. The code defines structures for tasks, messages, processors, and buses, and includes functions for calculating task ranks, earliest finish times, and the actual scheduling of tasks and messages based on rate-monotonic principles.

4.5.1 Key Functions

- **calculateRank and calculateEFT:** Compute priority metrics for tasks based on their periods and determine the earliest finish times based on available processor times.
- **schedulePeriodicTasks and scheduleMessages:** Implement the scheduling logic for tasks and messages, respectively, ensuring adherence to the formulated constraints and objectives.

This chapter lays the foundational theory and practical implementation details necessary for optimizing the scheduling of real-time tasks and messages in a heterogeneous environment, aligning closely with the needs of advanced Cyber-Physical Systems.

4.6 Algorithm for Scheduling Periodic Tasks and Messages

The algorithm schedules periodic tasks based on their periods using a rate-monotonic approach and messages according to transmission time requirements on heterogeneous buses.

Algorithm 1 Rate Monotonic Scheduling for Periodic Tasks and Messages

procedure SCHEDULEPERIODICTASKS($tasks, num\_processors$)
  Sort $tasks$ by increasing period
  Initialize $processor\_busy\_until$ array of size $num\_processors$
  for each $task$ in $tasks$ **do**
    $task.rank \leftarrow$ calculateRank($task$)
    for $processor = 1$ to $num\_processors$ **do**
      $task.eft[processor] \leftarrow$ calculateEFT($task, processor$)
    end for
    $selected\_processor \leftarrow \arg\min_{processor} task.eft[processor]$
    Schedule $task$ at $task.eft[selected\_processor]$ on $selected\_processor$
    Update $processor\_busy\_until[selected\_processor]$
  end for
end procedure
procedure SCHEDULEMESSAGES($messages, buses$)
  for each $message$ in $messages$ **do**
    $selected\_bus \leftarrow \arg\min_{bus}$ communication_time($message, bus$)
    Schedule $message$ on $selected\_bus$ at earliest possible time
  end for
end procedure
function CALCULATERANK($task$) **return** $\frac{1}{task.period}$
end function
function CALCULATEEFT($task, processor$)
  $next\_release \leftarrow \max(task.next\_release, processor\_busy\_until[processor])$
  return $next\_release + task.processing\_time[processor]$
end function

4.7 Explanation of the Algorithm

- **Initialization:**

  - The algorithm starts by sorting the tasks based on their periods, with shorter periods indicating higher priority (rate-monotonic scheduling).

- **Task Scheduling:**
  - Each task's rank and earliest finish time (EFT) on each processor are calculated.
  - The task is scheduled on the processor where it can finish the earliest, updating the processor's busy time to reflect the task's completion.

- **Message Scheduling:**
  - Each message is scheduled based on the minimum transmission time across the available buses, ensuring that messages are sent as efficiently as possible.

- **Functions:**
  - **calculateRank:** Returns the inverse of the task's period, with a higher rank for tasks with shorter periods.
  - **calculateEFT:** Computes the earliest time a task can finish on a given processor, considering the next available start time on the processor and the task's processing time.

This algorithm provides a systematic approach to scheduling in heterogeneous computing environments with multiple processors and communication buses, ensuring that tasks and messages are handled efficiently according to real-time constraints.

4.8 FlexRay Protocol and Message Scheduling

4.8.1 Overview of the FlexRay Protocol

The FlexRay protocol has emerged as a popular choice for real-time communication in distributed systems, particularly in safety-critical applications. It offers high data

rates, fault-tolerance, and deterministic behavior, making it suitable for automotive systems and other time-sensitive domains.

The FlexRay protocol is a time-triggered protocol. Its operation is based on a repeatedly executed FlexRay cycle (FC) with a fixed duration. Messages are transmitted in FlexRay frames that consist of message data as multiples of two-byte words, a framing overhead and a security overhead. If the message data comprise b words, then the frame size f in bit evaluates to

$$f = b \cdot 16 \text{ bits} + (b \cdot 4 \text{ bits} + OF) + OS = b \cdot 20 \text{ bits} + OF + OS \quad (1)$$

where the framing overhead according to is $b \cdot 4$ bits + OF and security overhead is calculated according to bit size and security level.

To gain a deeper understanding of the FlexRay protocol and its scheduling aspects, we refer to the work of Klaus Schmidt and Ece Guran Schmidt on message scheduling for the FlexRay protocol, specifically targeting the static segment. Their research provides valuable insights into the structure and operation of the FlexRay protocol, which serves as a foundation for our work. The SW and the NIT provide time for the transmission of internal control information and protocol-related computations.

4.8.2 Description of the FC

The FC comprises an SS, a DS, a symbol window (SW), and the network idle time (NIT). A generic FC is depicted in the upper part of Fig. 4.1.

The organization of the SS consists of a fixed number of equalsize static slots (STSs) that are incrementally counted by a slot counter in each FC, starting from 1. The bus arbitration is performed by uniquely assigning FIDs to nodes such that in each STS, the node with the FID that is equal to the current value of the slot counter can send a message. Fig. 4.1 shows an SS with six STSs. The FIDs have been assigned such that, for example, the messages A and B are transmitted by the node with FID 1. The SW and the NIT provide time for the transmission of internal control information and protocol-related computations.

In this paper, the case where several network nodes are connected by a single FlexRay communication channel is addressed. According to the FlexRay specification, each

node consists of a host and a communication controller (CC) that are connected by a controller–host interface (CHI), as depicted in Fig. 4.2(a). Here, the CHI serves as a buffer between the host and the CC. The host processes incoming messages and generates outgoing messages, while the CC independently implements the FlexRay protocol services

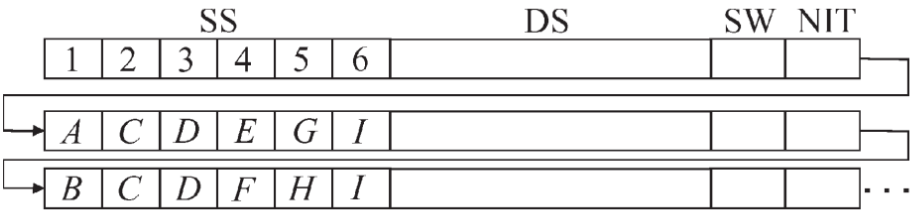
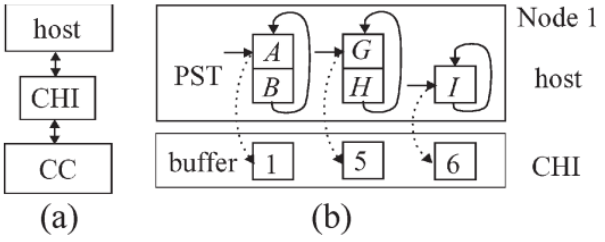


FIGURE 4.2: *FlexRay cycle description*

To support the periodic (jitter free) transmission of periodic messages in the SS, we propose the following software architecture. In compliance with the protocol specification, each slot in the FC with its corresponding FID is uniquely assigned to a host, where multiple FIDs can be allocated to each host. In addition, we adopt the assignment of messages to FIDs in such that each individual message cannot have more than one FID. With this prerequisite, we propose that each host holds a periodic scheduling table (PST) per allocated FID. In each FC, the PST determines a unique message to be transferred to the corresponding transmit buffer of the CHI among the periodic messages with the same FID.



(a) FlexRay node. (b) Software architecture.

FIGURE 4.3: *FlexRay Architecture*

Fig. 4.2(b) shows the software architecture for a host that generates the periodic messages A, B (period 2), G, H (period 2), and I (period 1) with the respective FIDs 1, 5, and 6 (see Fig. 4.1). For each FID, there is a PST that holds the related

messages. An arrow indicates the current message to be transferred to the respective transmit buffer in the CHI, i.e., in the first FC, the periodic messages A, G, and I are transmitted. The arrow moves one step ahead in each FC.

4.9 Message Framing Techniques for the Static Segment

The static segment of the FlexRay protocol plays a crucial role in the deterministic communication of real-time tasks. It is responsible for transmitting static frames that have fixed transmission times and predictable behavior. In this section, we discuss the message scheduling techniques employed in the static segment, drawing from the research by Klaus Schmidt and Ece Guran Schmidt.

One commonly used technique for message scheduling in the static segment is the slot-based scheduling approach. In this approach, the static segment is divided into a series of fixed-length time slots, and messages are assigned to these slots based on their transmission requirements. Each time slot is associated with a specific node or a group of nodes in the FlexRay network, ensuring the orderly transmission of messages.

Another technique utilized in the static segment is the static segment cycle length calculation. The cycle length defines the period of time over which the static frames repeat their transmission pattern. By carefully selecting the cycle length, it is possible to optimize the bandwidth utilization and meet the timing requirements of the real-time tasks.

Furthermore, the research by Klaus Schmidt and Ece Guran Schmidt explores the concept of multiplexing in the static segment. Multiplexing allows the transmission of multiple messages within a single time slot, improving the utilization of the available bandwidth. Various multiplexing strategies, such as static slot sharing and dynamic slot sharing, have been proposed to achieve efficient message transmission while satisfying the timing constraints.

By leveraging these message scheduling techniques for the static segment of the FlexRay protocol, we aim to enhance the scheduling algorithm's effectiveness and efficiency in our research on security-aware scheduling of persistent periodic real-time tasks.

4.10 Explanation of the Algorithm

- **Calculate GCD (Greatest Common Divisor):**
  - This function calculates the GCD of all signal periods. The GCD represents the cycle length Tc which is crucial in determining the synchronization point for all signals. This step ensures that all periodic signals align over the least common multiple of their periods.
- **Find Maximum Slot Size:**
  - For each possible time unit up to Tc , this function calculates the total size of signals that need to be transmitted at that time. It keeps track of the maximum size encountered, representing the worst-case scenario for slot size.
- **Find Optimal Slot Size:**
  - This procedure aims to find the optimal slot size that maximizes resource utilization. It iteratively calculates the total utilization for different potential slot sizes, selecting the size that results in the highest utilization without exceeding the cycle length Tc.
- **Display Message Sets:**
  - After determining the optimal slot size, this function groups signals into messages based on their periodicity and the optimal slot size. It displays the resulting groups, providing a clear view of how signals are batched together for efficient transmission.

Algorithm 2 Optimal Signal Scheduling

```
procedure CalculateGCD(periods)
    Tc ← accumulate(periods.begin(), periods.end(), periods[0], gcd)
    return Tc
end procedure
procedure FindMaxSlotSize(signals, Tc)
    max_sl_s ← 0
    for i = 1 to Tc do
        aggregate_size ← 0
        for each signal in signals do
            if i%signal.period == 0 then
                aggregate_size ← aggregate_size + signal.size
            end if
        end for
        max_sl_s ← max(max_sl_s, aggregate_size)
    end for
    return max_sl_s
end procedure
procedure FindOptimalSlotSize(signals, Tc, max_sl_s)
    optimal_slot_size ← FindMinSlots(Tc, max_sl_s, periods)
    min_slots ← countCoprimePeriodicities(periods)
    min_sl_s1 ← 0
    for each signal in signals do
        min_sl_s1 ← max(min_sl_s1, signal.size)
    end for
    max_slots ← Tc/min_sl_s1
    max_utilization ← 0
    for k = min_slots to max_slots do
        Calculate total utilization for slot size k
        if total_utilization > max_utilization then
            optimal_slot_size ← Tc/k
            max_utilization ← total_utilization
        end if
    end for
    return optimal_slot_size
end procedure
procedure DisplayMessageSets(signals, optimal_slot_size)
    Message bins creation and display based on periodicity and slot size.
end procedure
periods ← extract from signals
Tc ← CalculateGCD(periods)
max_sl_s ← FindMaxSlotSize(signals, Tc)
optimal_slot_size ← FindOptimalSlotSize(signals, Tc, max_sl_s)
DisplayMessageSets(signals, optimal_slot_size)
```

Chapter 5

Experimental Results and Discussions

5.1 Message Framing in FlexRay

5.1.1 Results Overview

The scheduling algorithm successfully computed the total cycle length (T_c) and the optimal slot size for the system based on the provided set of signals. The key results obtained from the execution of the algorithm are summarized as follows:

- **Total Cycle Length (T_c)**: The algorithm determined that the total cycle length for the system is 5 milliseconds. This represents the least common multiple of the periodicities of all input signals, allowing for synchronization of all signal transmissions within this interval.
- **Final Optimal Slot Size**: The optimal slot size for the communication was calculated to be 5 bytes. This size is sufficient to accommodate the largest signal within the cycle, ensuring efficient use of the communication channel.
- **Message Configuration**: The signals were grouped into messages based on their periodicities, leading to a compact and organized communication structure as detailed below:

  - Signals with a periodicity of 20 ms were grouped into Message 1, comprising Signal 3 and Signal 6.
  - Signals with a periodicity of 5 ms were organized into Message 2 (Signal 2) and Message 3 (Signal 5), indicating higher communication frequency.
  - Signals with a periodicity of 10 ms were combined into Message 4, containing Signal 1 and Signal 4.

- **Total Number of Messages**: A total of 4 distinct messages were created during this scheduling process.

5.1.2 Discussion

The results highlight several important aspects of the scheduling strategy and its effectiveness:

1. **Efficiency in Communication**: The optimal slot size of 5 bytes precisely matches the total cycle length, which suggests that the communication channel's capacity is fully utilized during each cycle. This efficiency is crucial in systems where the bandwidth is limited and must be optimally used.

2. **Signal Grouping Logic**: By grouping signals based on their periodicity, the algorithm ensures that signals transmitted more frequently are given more opportunities for transmission within each cycle. This approach not only simplifies the transmission schedule but also aligns well with the requirements of real-time systems where periodic monitoring is crucial.

3. **Impact on System Performance**: The structured grouping of signals can significantly enhance system performance by reducing the time and complexity involved in managing signal transmissions. It also helps in reducing the overhead that comes with handling numerous individual transmissions.

4. **Scalability**: The method demonstrated scalability in handling different signal sizes and periodicities. The approach can be adapted to systems with varying requirements and constraints, showcasing its flexibility and robustness.

5.1.3 Implications

These results have several implications for the design and operation of heterogeneous communication systems, particularly in environments where efficient use of resources is critical. The methodology used can serve as a basis for further research into more complex systems involving a larger number of signals or more diverse communication requirements. Future work could explore adaptive scheduling strategies that dynamically adjust to changing system conditions or signal priorities.

TABLE 5.1: Signal Parameters

Signal ID	Size (bytes)	Periodicity (ms)
1	2	10
2	3	5
3	1	20
4	2	10
5	3	5
6	1	20

```
Total cycle length (Tc): 5 milliseconds
Final optimal slot size: 5 bytes
Total cycle length (Tc): 5 milliseconds
Final optimal slot size: 5 bytes
Final message sets framed by combining signals:
Periodicity: 20ms, Messages:
Message 1: Signal 3 Signal 6
Periodicity: 5ms, Messages:
Message 2: Signal 2 Message 3: Signal 5
Periodicity: 10ms, Messages:
Message 4: Signal 1 Signal 4 /nNumber of messages created: 4
```

FIGURE 5.1: *Messages Framed*

5.2 Task Scheduling in Heterogeneous Systems

5.2.1 Results Overview

The task scheduling algorithm was executed on a heterogeneous computing system with three nodes and two processors. Key results from the scheduling process are summarized as follows:

- **Nodes and Processors**: The system included three nodes and two processors, indicating a small-scale yet complex scheduling environment.
- **Processing Cost Matrix**: The costs associated with processing each task on each processor were provided, illustrating the heterogeneous nature of the system where different processors have varying capabilities and efficiencies.
- **Adjacency and Processor Matrices**: These matrices provide insights into task dependencies and processor assignments, crucial for understanding task interactions and processor load distribution.
- **Task Scheduling Metrics**: Each task's Earliest Start Time (EST), Earliest Finish Time (EFT), and the actual finish time were calculated, alongside the selection of processors based on the scheduling algorithm.
- **Task Scheduling Order**: Tasks were scheduled primarily on Processor 1, reflecting its preference based on the calculated EFT and other scheduling metrics.

5.2.2 Discussion

The scheduling output provides several insights into the efficiency and challenges of task scheduling in heterogeneous environments:

1. **Processor Utilization**: The results show a heavy reliance on Processor 1, indicating either its superior processing capabilities or an imbalance in task distribution that could lead to potential bottlenecks. Processor 2 was underutilized, suggesting possible improvements in load balancing across processors.

2. **Task Dependency and Processing Cost**: The adjacency matrix and the processing cost matrix together influenced the scheduling decisions. Tasks with higher interdependencies and costs were scheduled in a manner that minimizes waiting time and optimizes processor usage.

3. **Efficiency of Scheduling Strategy**: The algorithm effectively scheduled tasks based on their ranks and EFTs, ensuring that tasks with tighter deadlines or higher priorities were allotted appropriate processing times. However, the concentrated use of one processor points to the need for an improved strategy that could better distribute tasks based on their computational demands and the available processor capabilities.

4. **Implications for System Performance**: The scheduling strategy, while effective in meeting task deadlines, suggests that the overall system performance could be enhanced by algorithms that dynamically adapt to the state of processor loads, potentially incorporating more advanced techniques like machine learning for predictive scheduling adjustments.

5.2.3 Future Work

Future research could focus on several areas to enhance the scheduling algorithm's performance and applicability:

- **Load Balancing Algorithms**: Developing more sophisticated algorithms that can dynamically balance the load across processors could prevent over-reliance on a single processor and improve overall system efficiency.

- **Real-Time Adaptability**: Incorporating real-time monitoring and adaptability into the scheduling algorithm could help in adjusting task priorities and processor assignments based on current system states.

- **Scalability Testing**: Extending the algorithm to handle larger systems with more tasks and processors would provide valuable data on its scalability and effectiveness in more complex scenarios.

```
Task Scheduling For Heterogeneous Computing Systems
------------------------------------------------------------------



Nodes: 3        Processor: 2

Processing Cost Matrix
2       4
3       5
4       6

Adj Matrix
1 -1 1
-1 1 -1
-1 -1 1

Processor Matrix
1       0

0       1

Node[3] 2.33333
Node[2] 2.2
Node[1] 4.33333
        PROCESS 2

EST     1       0
EFT     4       5
Actual Finish Time:     4
Processor Selected:     1
Processor State:        4       0
```

```
_____________________________________________________________

        PROCESS 3

EST     4       0
EFT     8       6
Actual Finish Time:     8
Processor Selected:     1
Processor State:        8       0


_____________________________________________________________

        PROCESS 1

EST     8       0
EFT     10      4
Actual Finish Time:     10
Processor Selected:     1
Processor State:        10      0


_____________________________________________________________


Task scheduling order (based on Rank and EFT with heterogeneous processors):
Task 1 with EFT 10 on Processor 1
Task 2 with EFT 4 on Processor 1
Task 3 with EFT 8 on Processor 1
```

FIGURE 5.2: *Task Scheduling in Heterogeneous Processors*

```
Communication Cost Matrix
Bus 1: 1        2
Bus 2: 2        3

EST and EFT for each message
Message 1 - EST: 0, EFT: 1
Message 2 - EST: 3, EFT: 4
Message 3 - EST: 0, EFT: 3
Message 4 - EST: 6, EFT: 8

Message scheduling order
Message 1 scheduled from Stage 1 to Stage 2 on Bus 1 from time 0 to 1
Message 2 scheduled from Stage 2 to Stage 3 on Bus 1 from time 3 to 4
Message 3 scheduled from Stage 1 to Stage 2 on Bus 2 from time 0 to 3
Message 4 scheduled from Stage 2 to Stage 3 on Bus 1 from time 6 to 8
Nodes: 2        Processor: 2

Processing Cost Matrix
Processor 1: 2  3
Processor 2: 3  4

Adj Matrix
Task 1: 1       -1
Task 2: -1      1

Processor Matrix
1       0
0       1

Ranks calculated
Task 1: 5
Task 2: 10
```

```
EST and EFT for each task
Task 1:
Processor 0 - EST: 1, EFT: 3
Processor 1 - EST: 1, EFT: 4
Task 2:
Processor 0 - EST: 3, EFT: 6
Processor 1 - EST: 3, EFT: 7

Task scheduling order
Task 1 scheduled on Processor 1 from time 1 to 3
Task 2 scheduled on Processor 1 from time 3 to 6
```

FIGURE 5.3: *Task and Message Schedule*

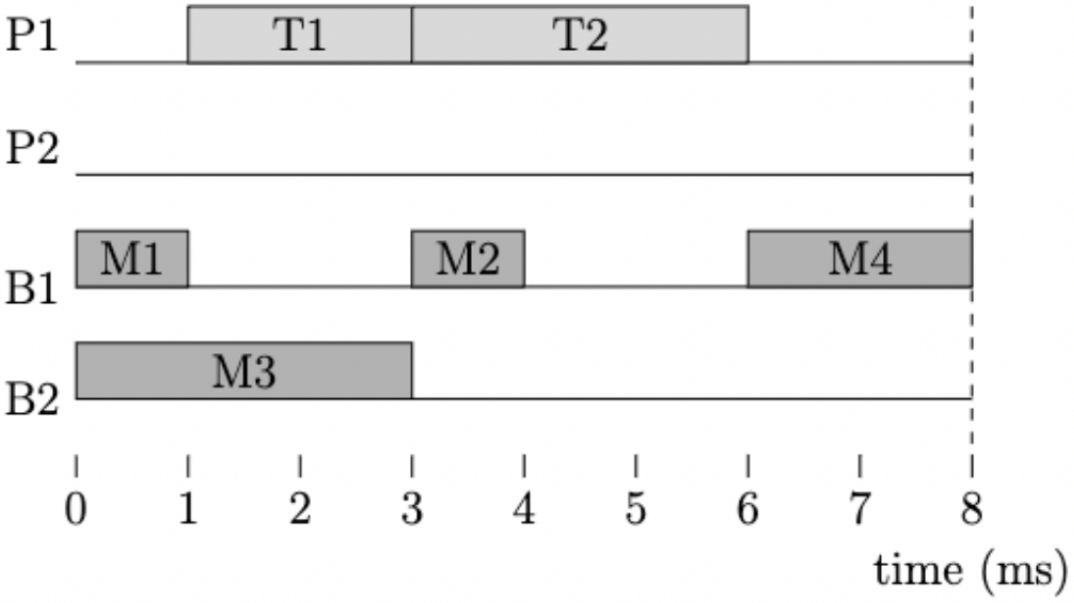


FIGURE 5.4: *Final Schedule for the PTG*

Chapter 6

Conclusion and Future Work

6.1 Conclusions

This research has successfully addressed several key challenges associated with scheduling periodic tasks and messages in real-time systems that utilize heterogeneous processors and the FlexRay bus system. By developing a sophisticated scheduling algorithm and formulating the problem as a nonlinear integer programming (NIP) challenge, we have created a comprehensive solution that optimizes both task execution and message transmission. The main achievements of this work include:

1. **Development of a Unified Scheduling Framework:**
   We introduced a scheduling model that effectively integrates the computational and communication demands of real-time systems, particularly those with heterogeneous architectures and strict communication protocols like FlexRay.

2. **Optimization of Task and Message Scheduling:**
   The proposed algorithm optimally manages the allocation and execution of tasks and the transmission of messages, significantly improving the overall system efficiency. It ensures that real-time constraints are met while maximizing resource utilization and minimizing makespan.

3. **Enhanced Efficiency of FlexRay Communication:**
   The research specifically targeted the static segment of the FlexRay protocol,

achieving optimized message packing and bandwidth utilization, which reduces communication latency and ensures reliable data transmission.

6.2 Future Work

While this research has made significant contributions to the field of real-time system scheduling, several avenues remain open for further exploration:

1. **Dynamic and Shared Environments:**
   Explore scheduling solutions that adapt to dynamic changes and can efficiently manage shared communication architectures like bus networks, which are prevalent in CPSs.

2. **Group Scheduling on Shared Buses:**
   Develop strategies for scheduling groups of independent periodic applications, each modeled as a PTG, on processors sharing buses. This will address the challenges of resource contention more comprehensively.

3. **Integration with Other Communication Protocols:**
   Extend the scheduling algorithm to interact seamlessly with other key communication protocols such as Ethernet and Time-Triggered Ethernet (TTE), enhancing its applicability and effectiveness in diverse real-time systems

4. **Machine Learning Approaches:**
   Incorporating machine learning techniques to predict system behavior and dynamically adjust scheduling parameters could enhance the adaptability and efficiency of the scheduling process.

5. **Security-Aware Scheduling:**
   Given the critical role of ECUs in safety-sensitive applications, future iterations of the algorithm should incorporate security considerations, particularly for communication networks like FlexRay.

6. **Hardware Implementation and Testing:**
   Implementing the algorithm in actual hardware setups and conducting real-world testing would provide deeper insights into its performance and practical challenges.

7. **Energy Efficiency Considerations:**
   Future developments could also explore the impact of scheduling on energy consumption, aiming to optimize not just time and resource usage but also the energy efficiency of heterogeneous systems.

By pursuing these directions, subsequent research can continue to push the boundaries of what is possible in real-time systems scheduling, driving further improvements in system performance and reliability across various high-stakes industries.

Bibliography

[1] Chandan Karfa Debabrata Senapati, Arnab Sarkar. Energy-aware real-time scheduling of multiple periodic dags on heterogeneous systems, 2022.

[2] K. Li G. Xie, R. Li. Heterogeneity-driven end-to-end synchronized scheduling for precedence constrained tasks and messages on networked embedded systems, 2015.

[3] Jorge G. Barbosa Hamid Arabnejad. List scheduling algorithm for heterogeneous systems by an optimistic cost table, 2014.

[4] S. Gu J. Hu G. Zhu E. H.-M. Sha J. Liu, Q. Zhuge. Minimizing system cost with efficient task assignment on heterogeneous multicore processors considering time constraint, 2014.

[5] Ece Guran Schmidt Klaus Schmidt. Message scheduling for the flexray protocol: The static segment, 2008.

[6] L. A. Sousa O. Sinnen. Communication contention in task scheduling, 2005.

[7] A. Sarkar S. K. Roy, R. Devaraj. Optimal scheduling of ptgs with multiple service levels on heterogeneous distributed systems, 2019.

[8] A. Sarkar Kankana Maji Sayani Sinha S. Roy, R. Devaraj. Contention-aware optimal scheduling of real-time precedence constrained task graphs on heterogeneous distributed systems, 2020.

[9] A. Sarkar S.K. Roy, R. Devaraj. Contention cognizant scheduling of task graphs on shared bus-based heterogeneous platforms, 2022.

[10] R. Gangopadhyay S.K. Roy, A. Sarkar. Processor and bus co-scheduling strategies for real-time tasks with multiple service-levels, 2021.

[11] F. Suter. Dag generation program, 2010.

[12] D. Padua X. Tang, K. Li. Communication contention in apn list scheduling algorithm, 2009.

[13] Tao Xie. Security-aware scheduling for real-time systems, 2005.