EFFICIENT ALGORITHMS FOR THE FRAME PACKING AND SLOT ALLOCATION OF FLEXRAY V3.0

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

CUMHUR ÇAKMAK

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

FEBRUARY 2017

Approval of the thesis:

EFFICIENT ALGORITHMS FOR THE FRAME PACKING AND SLOT ALLOCATION OF FLEXRAY V3.0

submitted by **CUMHUR ÇAKMAK** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Gülbin Dural Ünver _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Tolga Çiloğlu _____
Head of Department, **Electrical and Electronics Engineering**

Assoc. Prof. Dr. Ece Güran Schmidt _____
Supervisor, **Electrical and Electronics Engineering, METU**

Assoc. Prof. Dr. Klaus Schmidt _____
Co-supervisor, **Mechatronics Engineering, Cankaya Univ.**

Examining Committee Members:

Prof. Dr. Uğur Halıcı
Electrical and Electronics Engineering Department, METU _____

Assoc. Prof. Dr. Ece Güran Schmidt
Electrical and Electronics Engineering Department, METU _____

Prof. Dr. Gözde Bozdağı Akar
Electrical and Electronics Engineering Department, METU _____

Assoc. Prof. Dr. Cüneyt Bazlamaçcı
Electrical and Electronics Engineering Department, METU _____

Assist. Prof. Dr. Barbaros Preveze
Electrical and Electronics Engineering Dept., Çankaya Univ. _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: CUMHUR ÇAKMAK

Signature :

ABSTRACT

EFFICIENT ALGORITHMS FOR THE FRAME PACKING AND SLOT ALLOCATION OF FLEXRAY V3.0

Çakmak, Cumhur
M.S., Department of Electrical and Electronics Engineering
Supervisor : Assoc. Prof. Dr. Ece Güran Schmidt
Co-Supervisor : Assoc. Prof. Dr. Klaus Schmidt

February 2017, 61 pages

Contemporary vehicles employ a large number of Electronic Control Units (ECUs), sensors and actuators that exchange signals over an in-vehicle network (IVN). These signals are packed in frames which are transmitted according to a precomputed schedule to satisfy the stringent real time requirements of the vehicle operation. Both the signal packing and scheduling algorithms affect the utilization and timing properties of the IVN and hence the entire vehicle electronic system. Despite the offline computation, the run-time of these algorithms might become prohibitively long as the number of signals increases.

This thesis proposes signal packing and scheduling algorithms for the Static Segment of the FlexRay IVN standard which is dedicated to the transmission of periodic signals. The proposed algorithms achieve high bandwidth utilization, fulfill the timing requirements and run in feasible times. To this end, the first contribution of the thesis is an Integer Linear Program (ILP) formulation to pack signals into FlexRay frames based on their period properties. The second contribution is a post-processing algorithm to improve the bandwidth utilization. The third contribution is adapting an existing scheduling algorithm to the new version FlexRay v3.0 to achieve feasible schedules under the developed packing methods. The proposed algorithms are applied to signal sets with different properties to demonstrate their advantages compared to previous work in the literature.

Keywords: In-Vehicle Networks, FlexRay, Static Segment, Frame Packing, Integer Linear Programming, Slot Allocation, Scheduling Algorithm

ÖZ

FLEXRAY V3.0 İÇİN VERİMLİ ÇERÇEVE OLUŞTURMA VE DİLİM ATAMA ALGORİTMALARI

Çakmak, Cumhur
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi : Doç. Dr. Ece Güran Schmidt
Ortak Tez Yöneticisi : Doç. Dr. Klaus Schmidt

Şubat 2017 , 61 sayfa

Günümüz araçlarında bulunan çok sayıda Elektronik Kontrol Ünitesi(EKÜ), algılayıcı ve eyleyici, Araç-İçi Ağ üzerinden aralarında sinyal alışverişi yapmaktadırlar. Araç işleyişi için gerekli olan katı gerçek-zaman gereksinimlerini sağlamak amacıyla önceden hesaplanan çizelgeye göre gönderilen bu sinyaller, çerçeveler içine paketlenmiştir. Hem sinyal paketleme hem de çizelgeleme algoritmaları, Araç-İçi Ağların kullanım ve zamanlama özelliklerini ve dolayısıyla bütün araç elektronik sistemini etkilemektedir. Çevrimdışı hesaplamalar yapılmasına rağmen, sinyal sayısı arttıkça bu algoritmaların işlem süresi kabul edilemez derecede uzun olabilmektedir.

Bu tez çalışmasında, periyodik sinyallerin gönderimine ayrılmış olan FlexRay Araç-İçi Ağ Statik Dilimi için sinyal paketleme ve çizelgeleme algoritmaları sunulmaktadır. Sunulan algoritmalar bant genişliğini verimli kullanıp, zamanlama gereksinimlerini karşılarken aynı zamanda makul sürelerde çalışmaktadırlar. Bu tezin yaptığı öncelikli katkı, sinyalleri periyod özelliklerine göre FlexRay çerçeveleri içine paketleyecek bir Doğrusal Tamsayı Programlama (DTP) formülasyonu geliştirilmesidir. İkincil olarak, bant genişliği verimini artırmak için ardıl işlem uygulayanan bir algoritma geliştirilmiştir. Son olarak da geliştirilmiş olan paketleme yöntemleri için uygulanabilir mesaj çizelgesi amacıyla, halihazırda var olan bir çizelgeleme algoritması yeni FlexRay 3.0 versiyonuna uyarlanmıştır. Sunulan algoritmalar, literatürde daha önceden yapılmış olan çalışmalara göre avantajlarını göstermek amacıyla, çeşitli özelliklere sahip sin-

yal kümeleri üzerinde uygulanmıştır.

Anahtar Kelimeler: Araç-İçi Ağlar, FlexRay, Statik Dilim, Çerçeve Oluşturma, Doğrusal Tam Sayı Programlama, Bölüt Atama, Çizelgeleme Algoritması

To my family

ACKNOWLEDGMENTS

Foremost, I would like to express my deepest gratitude to my advisors, Assoc. Prof. Dr. Ece Güran Schmidt and Assoc. Prof. Dr. Klaus Schmidt, for their continuous support of my research. I would not have gotten this far without their encouragement. Besides my advisors, I would like to thank the rest of my thesis committee; Prof. Dr. Uğur Halıcı, Prof. Dr. Gözde Bozdağı Akar, Assoc. Prof. Dr. Cüneyt Bazlamaçcı and Assist. Prof. Dr. Barbaros Preveze for their insightful comments.

I cannot thank enough to Elif Sarıtaş and Murat Kumru. It would never have been possible to complete this research without their support, guidance and patience. They have made all kinds of contributions to this work, from beginning until the end.

I also appreciate the help of Utku Civelek, Mertcan Erdoğan, Ceren Ertekin, Berk Özkan, Gizem Tekin and Berker Yaşar, for giving their time and energy to make sure that I get well-prepared to my defense.

Finally, I want to thank all my friends and my colleagues for everything. I want them to know that every one of them have participated in this process with me in a unique way, which I will never forget in my life.

TABLE OF CONTENTS

ABSTRACT v

ÖZ vii

ACKNOWLEDGMENTS x

TABLE OF CONTENTS xi

LIST OF TABLES xiv

LIST OF FIGURES xv

LIST OF ABBREVIATIONS xvi

CHAPTERS

1 INTRODUCTION 1

2 PRELIMINARIES AND FORMAL PROBLEM DEFINITION 7

2.1 FlexRay Protocol 7

2.2 Differences Between v2.1 and v3.0 8

2.3 General Requirements of Static Segment Scheduling Problem 9

2.4 Performance Metrics 11

2.4.1 Bandwidth Utilization (U): 11

2.4.2 Used Static Segment (USS) 12

2.4.3 Scalability 13

2.5 Two Stages of SS Scheduling 13

2.5.1 Frame Packing Problem 13

2.5.2 Slot Allocation Problem 15

3 EXISTING FRAME PACKING AND SLOT ALLOCATION ALGORITHMS 17

3.1 Motivation 17

3.1.1 Frame Packing 18

3.1.2 Slot Allocation 20

3.2 Formulation of Frame Packing Algorithms 21

3.2.1 Variable Frame Packing ILP (VP) 21

3.2.2 Fixed Frame Packing ILP (FP) 23

3.3 Formulation of Slot Allocation Algorithms 25

3.3.1 Integration ILP (ISA) 25

4 DEVELOPED ALGORITHMS 27

4.1 Frame Packing 27

4.1.1 Motivation 27

4.1.2 Fixed Frame Packing With Multiple Periods (FMP) 30

4.1.3 FMP and Post Processing (FMP+) 32

4.2 Slot Allocation 36

4.2.1 Motivation 36

4.2.2 Schmidt Scheduling ILP for v3.0 (SSA) 36

5 PERFORMANCE EVALUATION 41
5.1 Environment 41
5.2 Frame Packing 42
5.2.1 Test Case 1: 43
5.2.2 Test Case 2: 46
5.2.3 Test Case 3: 48
5.2.4 Test Case 4: 49
5.2.5 Test Case 5: 50
5.2.6 Test Case 6: 51
5.3 Slot Allocation 52
6 CONCLUSION 57
REFERENCES 59

LIST OF TABLES

TABLES

Table 1.1 Comparison of approaches similar to the study presented in thesis . . 4

Table 4.1 Group Properties 39

Table 5.1 Benchmark Results 48

Table 5.2 Comparison of Results from Two Different Works 49

Table 5.3 T_{STS} of Optimal Solutions 50

Table 5.4 Performance of Frame Packing for Uniform Distribution of p_s and b_s 50

Table 5.5 SAE Benchmark 51

Table 5.6 Variable Packing with Jitter 51

Table 5.7 ILP Problem Size of Frame Packing Algorithms 52

Table 5.8 Number of Frames Created 53

Table 5.9 Feasibility of Solutions 54

Table 5.10 USS of Feasible Solutions 54

Table 5.11 The Run-times and Used FIDs for N_{STS} = 600 54

Table 5.12 ILP Problem Size of Slot Allocation Algorithms 55

LIST OF FIGURES

FIGURES

Figure 2.1 FlexRay Operation 8

Figure 2.2 v3.0 Differences 9

Figure 2.3 Fixed vs Varaible Packing 14

Figure 2.4 Frames assigned to slots 16

Figure 3.1 Signals packed into frames 18

Figure 3.2 Schmidt Frame Packing 19

Figure 4.1 Result of Sagstetter's Frame Packing ILP 29

Figure 4.2 Optimal Frame Packing 30

Figure 4.3 Fixed to Variable Packing 33

Figure 4.4 Creation of Empty Map $E_{(f,r)}$ for f_1 34

Figure 4.5 Choice of Period Group 37

Figure 5.1 Distribution of Signal Periods and Signal Sizes 43

Figure 5.2 N = 30, S^n = 10, 20, 30, 40, 50 44

Figure 5.3 Utilization for fixed $N = 30$ 44

Figure 5.4 run-time for fixed $N = 30$ 45

Figure 5.5 S^n = 30, N = 10, 20, 30, 40, 50 46

Figure 5.6 Utilization for fixed $S^n = 30$ 46

Figure 5.7 Run-time for fixed $S^n = 30$ 47

Figure 5.8 Distribution of Signals to Nodes for test case 3 48

LIST OF ABBREVIATIONS

ECU	Electronic Control Unit
FID	Frame Identifier
SS	Static Segment
DS	Dynamic Segment
STS	Static Slot
gdBit	Bit duration
MT	Macrotick
ILP	Integer Linear Program
VP	Variable Frame Packing
FP	Fixed Frame Packing
FMP	Fixed Frame Packing ILP With Multiple Periods
FMP+	Fixed Frame Packing ILP With Multiple Periods + Heuristic
SSA	Schmidt Slot Allocation ILP
ISA	Integration Slot Allocation ILP
U	Bandwidth Utilization
USS	Used Static Segment

CHAPTER 1

INTRODUCTION

The development of electronics in the automotive industry tries to respond to the increasing customer demands. In-vehicle network (IVN) designs are always being improved to keep up with safety and comfort requirements. In addition, innovative technologies in automotive electronics replace the mechanic and hydraulic systems by *X-by-wire* systems. Automobile manufacturers also keep introducing new systems such as active safety, traction control, lane-departure warning, etc [1]. Such systems are implemented on a distributed architecture with more than 70 electronic control units (ECUs) exchanging more than 2500 signals [2] in real-time. Transmission timings for these systems can be very critical.

Automobile manufacturers such as BMW, Audi, DaimlerChrysler and Bosch have developed the FlexRay protocol between the years 2000 and 2010 for this purpose. This development was made based on the observation that the CAN (Controller Area Network) bus protocol, which is currently predominant in IVNs. In particular, CAN cannot respond to the strict timing requirements since it is an event-triggered protocol with relatively low data rates (below 1 Mbps) [3]. FlexRay, on the other hand, separately reserves bandwidth dedicated to the transmission of time-triggered signals and event-triggered signals.

FlexRay is a high-speed (up to 10 Mbps) in-vehicle network protocol. It operates in fixed duration of communication cycles which are repeatedly executed. Each FlexRay cycle consists of 2 segments to transmit different types of signals: One is the static segment (SS) which is used to transmit time-triggered (periodic) signals and the other is the dynamic segment (DS) which is used to transmit event-triggered (sporadic)

signals.

In this study, we focus on the scheduling of the static segment of FlexRay. The SS is composed of a fixed number of static slots, in which the signals are transmitted on a time division multiple access (TDMA) basis. The transmission timings for the FlexRay signals are defined by a schedule specifying which signals are to be transmitted in each slot of each FlexRay cycle. This information is computed off-line before the network operation starts and recorded in local scheduling tables of each ECU. Each ECU transmits its signals according to this schedule, with its own synchronized clock. Once the schedule is set, there is neither a global control for transmission, nor a probabilistic procedure to start transmitting. That is, the main design problem for FlexRay is the computation of this schedule for a given IVN with its set of ECUs and set of signals for each ECU together with their scheduling requirements: First, signals have to meet their deadlines, i.e., each signal has to be transmitted within a certain period of time. Secondly, there can be no overlapping of signals in the schedule. Finally, all the signals in the system has to be scheduled.

When determining a FlexRay schedule, it is desired to utilize the network bandwidth efficiently due to the following reasons. First, it has to be considered that the number of signals to be transmitted on FlexRay is very high. Second, it is desirable to accommodate possible extensions on the same network such as new ECUs or signals for new automotive applications. If a certain signal set can be scheduled occupying a small part of the SS, there will be more room to add new signals to the schedule in the future. In principle, the computation of SS schedules is done offline such that it does not need to meet real-time requirements. Nevertheless, considering potentially large signal sets, any practical FlexRay scheduling algorithm must be scalable to large signal sets in order to avoid large waiting times in the design phase [4].

FlexRay is evolving over years and the most recent version of the standard proposes significant changes in the protocol which are expected to affect the schedule performance. With the introduction of FlexRay v3.0 in 2010 [5], the scheduling of periodic signals can be implemented more efficiently.

Taking into account the stated requirements, this thesis studies the scheduling problem of the FlexRay SS using the newest version FlexRay v3.0. In analogy to the

existing literature [4, 6, 7, 8], we assume that the set of signal periods, deadline requirements and signal sizes are fixed such that the application level requirements are separated from the schedule computation. We decompose the FlexRay scheduling problem into two modular stages. First, signals are packed into message frames and then the frames are assigned to static slots. We use both integer linear programming (ILP) and hybrid (ILP + Heuristic) methods to minimize the used bandwidth and maximize the bandwidth utilization. For the frame packing stage, we developed 2 new algorithms and implemented the existing approaches in [6] and [4] for comparison. For the slot allocation stage, we adapt the message scheduling ILP formulation in [6] to the slot allocation problem for the new FlexRay v3.0 protocol. We also implemented the formulation in [4] for comparison. Our main contributions to this problem are:

- Design of an Integer Linear Program (ILP) formulation [9] to pack signals into fixed frames based on the ideas in [6]. The condition on the periods of the signals to be packed in a frame is modified in order to achieve a higher bandwidth utilization. In this newly developed FMP (Fixed Packing of Multiple Periods) algorithm, we allow packing signals whose periods are multiples of each other, and solve the formulation accordingly.

- Design of a heuristic post processing algorithm to improve the result of an FMP solution. The overall approach is denoted as FMP+ (Fixed Packing of Multiple Periods + Post Processing). FMP+ improves the bandwidth utilization compared to FMP by filling unoccupied space of frames in FMP. This approach promises to find solutions with higher bandwidth utilization, in smaller runtimes, than the VP (Variable Packing) algorithm in [4].

- Adaptation of the message scheduling algorithm in [6] to the problem of frame-to-slot assignment. As a result, it is possible to benefit from the changes in FlexRay v3.0. The modified ILP formulation finds a minimum bandwidth solution in constant time and is hence considerably faster than the approach in [4].

There has been many studies to respond the demands of SS scheduling problem. Most of them focus on the minimization of the used static segment. ILP formulations

are usually preferred to find the optimal solution under certain assumptions, whereas heuristic algorithms are preferred to decrease run-time.

The studies in [6], [4], [7] and [8] focus on the minimization of the used bandwidth in the SS, by scheduling signals independently of the application layer requirements. Table 1.1 shows the methods of minimization, used version of FlexRay protocol and the addressed problem in these studies. [6] and [7] pack signals into fixed frames, i.e., without multiplexing of signals in the same time portion of frame. Both works optimize over the frame size, whereas the frame size is kept as a predefined constant in [4] and [8].

Table 1.1: Comparison of approaches similar to the study presented in thesis

	Heuristic / ILP	FlexRay	Frame Packing/Slot Allocation
[6]	ILP	v 2.1	Both
[4]	Both	v 3.0	Both
[7]	Both	v 2.1	Frame Packing
[8]	Heuristic	v 2.1	Both

[10] develops a heuristic algorithm to perform SS scheduling with minimum jitter. [11] also defines a similar problem, where the only consideration is to perform schedule synthesis so that all the signals meet their deadlines. They transform the frame packing problem into bin packing as an intermediate step and assign packed bins to slots. They implement their method both with heuristic algorithms and ILP formulations. In [12], although the freshness constraint is taken into account, the tasks in the application level are asynchronous with FlexRay cycles. In other words, the transmission of signals are independent of task execution orders, destination nodes etc. The packing of signals into frames, however, is not implemented in this work. Thus, one signal is sent in each frame, which significantly decreases the bandwidth utilization. The authors of [6] quantify jitter, which is the delay variance of the transmission of signals in [13]. Their work aims at minimizing jitter and used SS bandwidth at the same time.

The works in [14], [15], [16] and [17] on the other hand, approach the problem at system level. They consider end-to-end deadlines for signals, synchronization of signal generations to the FlexRay communication cycle and the task precedence constraints between task executions and signal transmissions. [17] presents an early work, in

which a genetic algorithm is employed. The authors of [14] and [15] implement both task and signal scheduling, using ILP formulations. [16] on the other hand, uses heuristic methods to decrease the run-time. It is also one of the few studies that make use of the features introduced by FlexRay v3.0. However, they implement their algorithm with a very limited test case, which is not sufficient to demonstrate the efficiency for real-life examples.

There are other works, which have the same goal of efficient use of bandwidth, but approach the problem from different perspectives. Unlike the majority of the studies on SS scheduling problem, [18] and [19] remove the constraint on signal periods to be integer multiples of the FlexRay communication cycle duration, using response time analysis. [20] points out the industry demands of having SS scheduling to be flexible to changes, updates and extensions in the future. They make the definition of extensibility, create an uncertainty model and develop their algorithms based on these. The study in [21] presents a conceptual hardware design of a switched FlexRay network with active star topology. The function of the switch is similar to Ethernet switches. It can forward multiple signals that have different source and different destination nodes. A special scheduling is applied where signals can be scheduled at overlapping cycle of slots with the use of switches.

The outline of the thesis is as follows. In Chapter 2 the FlexRay protocol, the general formulation of the SS scheduling problem and the performance metrics are introduced. The main updates in the new version v3.0 are demonstrated and the idea of the two stage scheduling approach is described. Existing approaches and possible implementations for both stages are explained in Chapter 3. Chapter 4 presents the developed ILP formulations for each stage and the heuristic post processing algorithm for frame packing stage. In Chapter 5 , the test results for the implemented algorithms of both stages are presented. The results are compared with the ones in [4] to demonstrate the obtained improvements. Chapter 6 draws conclusions on the overall performance of the implemented algorithms. In addition, the obtained results are discussed and possible future work is presented.

CHAPTER 2

PRELIMINARIES AND FORMAL PROBLEM DEFINITION

2.1 FlexRay Protocol

FlexRay is a communication bus protocol in which the time is decomposed into fixed size communication cycles. Each cycle is divided into 4 segments, namely Static Segment (SS), Dynamic Segment (DS), Symbol Window (SW) and Network Idle Time (NIT). SS and DS are used to transmit signals from the ECUs of the distributed system, whereas SW and NIT are used for internal protocol related communication to exchange control information. The periodic signals are transmitted in SS of each cycle, whereas the sporadic ones are transmitted in DS. SS is the focus of this study, so no further details will be given on DS.

The SS in each FlexRay cycle is divided into a certain number of equal size static slots (STS). The operation is on TDMA basis, which means that a single frame (group of signals from a certain ECU) is transmitted in a single slot of a single FlexRay cycle and another can be transmitted in the same slot of a different FlexRay cycle, depending on the frame period. Each frame is assigned to a static slot named with a frame identifier (FID), and repeated once every r_f cycles, where r_f is the repetition period of a frame. r_f is imposed by the periods of signals to be transmitted in that frame. The signals in the system have to be scheduled for transmission, i.e. a certain time duration has to be explicitly reserved for each signal. Since the signals are periodic, the scheduling has to be done within a scheduling window, set by a a number k.

Fig. 2.1 illustrates the FlexRay operation. As the example suggests, frame B is to be

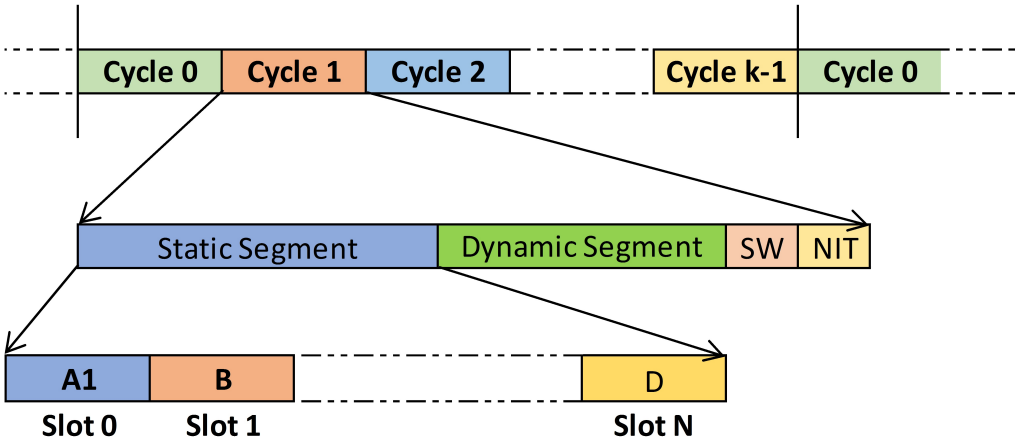


Figure 2.1: FlexRay Operation

transmitted in Slot 1 of Cycle 1. Assume that frame B has a repetition period $r_f = 4$. Then Slot 1 of Cycles 5, 9, 13, 17... also have to be allocated for frame B. All k cycles are programmed in this manner, i.e. the frames in each slot in each cycle is determined in the design process. Then this schedule of k cycles is repeated one after the other, in continuous time. This k cycle schedule, is kept in all local ECUs. They transmit their signals according to this schedule, without any other network control.

2.2 Differences Between v2.1 and v3.0

The recently introduced version of FlexRay is v3.0 [5]. The changes introduced in this version allows a more flexible scheduling of signals into slots. In this work, we point out to two major differences from previous version v2.1, which are (1) multiplexing of slots among multiple ECUs and (2) possibility of having a different cycle number other than 64.

(1) In the previous versions of FlexRay, each node (ECU) is assigned a number of unique *frame identifiers* (FIDs) corresponding to slots in the static segment, so that in each FlexRay cycle, only the signals from that specific node can be transmitted in that frame. In other words, each slot is owned by only one node. With such an exclusive allocation of static slots to nodes, the bandwidth would not be used very efficiently since it is not allowed to fill the empty cycles of one slot with messages of other nodes. The FlexRay bus does not transmit anything during that period, resulting to a waste of bandwidth. In v3.0, this limitation is removed. With the new version,

any two message frame can be assigned to the same slot, regardless of their ECUs. This allows a more efficient use of the FlexRay bus. Fig. 2.2 illustrates the difference explained above.

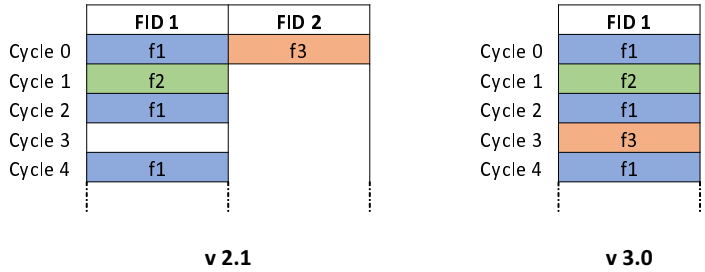


Figure 2.2: In FlexRay 2.1, message frames of ECU_1 $\{f_1, f_2, f_3\}$and ECU_2 $\{f_4\}$ had to be in separate slots, causing empty spaces in each slot. In the new version, all the message frames can be scheduled in the same slot.

(2) In FlexRay version 2.1, the entire static segment schedule repeats every 64 cycles where the repetition periods r_s^n of the signals are restricted to {1, 2, 4, 8, 16, 32, 64}. Signals having a different repetition would have to be scheduled as a signal with a rate from this given set. This restriction introduces more jitter to the signal transmission times and/or more bandwidth demand than actually required. In FlexRay v3.0 on the other hand, the static segment can be scheduled for any scheduling period k between 1 and 64 and the repetition periods r_s^n of the signals can be selected from the larger set of {1, 2, 4, 5, 8, 10, 16, 20, 32, 40, 50, 64}. This gives more flexibility to the designer to choose a different scheduling period depending on the signal set. For a set of signals having 1,2 and 40 as repetition periods as in SAE benchmark [22] set for instance, Choosing cycle number as 40 for this case could possibly give a more efficient and jitter-free result than a cycle number of 64.

The parameters of the FlexRay protocol such as maximum bandwidth C = 10 Mbit/s, bit time gdBit = 0.1 μs, minimum and maximum values of macrotick duration T_{MT}= (1 μs, 6 μs) remain unchanged in this new version.

2.3 General Requirements of Static Segment Scheduling Problem

In the problem of static segment scheduling, we are given a set of nodes $\mathcal{N}$ with a set $\mathcal{S}^n$ of periodic signals for each sending node n. Each signal s_s^n has a period p_s^n, a

number of bits b_s^n and a deadline d_s^n in which it must be transmitted. We assume the macrotick duration T_{MT}, bit time gdBit and bandwidth C parameters of the FlexRay bus are given. The requirement is to find a k-cycle scheduling of this signal set for a given FlexRay bus which would fit into the static segment (2) and which would satisfy the deadline requirements of all signals (1).

Other requirements such as the order of task executions or destination nodes of signals are out of our scope in static segment scheduling. It is assumed a successful transmission if all the signals meet their deadlines.

We set FlexRay cycle T_C as a constant parameter. It is chosen as the greatest common divisor of all signal periods. Upon this choice, all signal periods can be represented as an integer multiple of T_C i.e. the repetition period r_s^n. For the sake of simplicity, the repetition periods of signals will be referred to as signal periods throughout this thesis. In other words, $p_s^n = r_s^n$ is used rather than the actual value $p_s^n = r_s^n \cdot T_C$. Then the total duration of the static segment $T_{c,SS}$ is set. T_C is limited to 16 ms. A single static slot is limited to 664 macroticks. Max number of static slots is 1023. Depending on the application (DS demand, T_{MT} selection etc.), a $T_{c,SS}$ is chosen within those boundaries. T_C and $T_{c,SS}$ will be assumed to be predefined constants throughout this work.

(1) Each signal s_s^n has a deadline d_s^n, stating the maximum delay from the time the signal is ready until it is transmitted to the FlexRay bus. The propagation and delay and other delays in the software stack are not taken into consideration within the scope of this work. For periodic signals, the deadline requirements are satisfied if they are transmitted within their periods, so it can be written as $d_s^n = p_s^n$.

(2) As stated in Section 2.1, the static segment is composed of a number of static slots each having an equal duration of T_{STS} seconds. There are $N_{STS} = T_{SS}/T_{STS}$ static slots in total. The data is transmitted as frames in the static segment, which are simply groups of signals plus overhead. One frame is transmitted in each static slot of a FlexRay cycle. T_{STS} is determined with respect to frame size b, represented as T_{STS}^b. The calculation is as follows:

$$T_{STS}^b = \left\lceil \frac{b \cdot 20\text{bit} + O_F}{T_{MT} \cdot C} \right\rceil \cdot T_{MT} \tag{2.1}$$

There are b 2-byte words in a frame, thus there can be a maximum of $b \cdot 16$ data bits in a frame [1]. The framing overhead according to [5] is $b \cdot 4 + O_F$ bits where O_F is 90 bits. The value of b and an according T_{STS} are to be determined, which have to be the same for all slots. The scheduling of periodic signals is done by assigning each message frame to a static slot [2]. Each used static slot has a frame identifier FID_i. As the requirement states, the obtained schedule has to fit in the static segment. So the number of possible FIDs is limited by the number of available static slots.

$$FID_{tot} \leq N_{STS} \tag{2.2}$$

2.4 Performance Metrics

Other than the general SS scheduling requirements, we have additional goals in static segment scheduling problem, as described in Section 1. A decent SS scheduling algorithm has to find an applicable SS schedule which:

1. Use the bandwidth with maximum efficiency
2. Use the minimum possible amount of bandwidth
3. Is scalable to large signal sets and large number of nodes

In order to measure the performance of the implemented algorithms, the following metrics are defined :

2.4.1 Bandwidth Utilization (U):

The utilization metric demonstrates how efficiently FlexRay bus is used. U is formulated as the ratio of the bandwidth demand of all periodic signals (D) to the bandwidth allocated to these signals in static segment (A):

$$U := \frac{D}{A} \tag{2.3}$$

[1] The lower and upper limits on the number of 2-byte words in a frame is specified by FlexRay protocol to be 2 and 127, respectively.

[2] This, however, is not a 1:1 assignment. Although a message frame can be assigned to one and only one slot, a slot can be allocated to multiple message frames. Thus, the maximum number of FIDs is limited by the number of message frames.

We define D and A in Section 2.5.1. The aim of this work is to develop a scheduling algorithm that would minimize U for any given signal set. Demand is defined by the problem itself, since the deadline requirements and the number of data bits of the signals are already given in the signal set and they cannot be modified. Thus, we seek ways to decrease the bandwidth allocated to signals, which depends on the the frame size and the packing of signals in frames. Calculation of D and A will be explained in details in Section 2.5.1, where frame packing problem is described.

2.4.2 Used Static Segment (USS)

The used region in static segment in units of time (seconds). An STS is marked as used if there is at least one message frame assigned to it, i.e. data is transmitted in at least one cycle of that static slot. The total number of used slots is FID_{tot} and the duration of a single slot is T_{STS}, therefore the used amount of static segment is:

$$USS \coloneqq FID_{tot} \cdot T_{STS} \tag{2.4}$$

The aim is to minimize USS in SS schedule, in order to have an extensible static segment, i.e. to make it possible to add extra signals to the FlexRay configuration after the first design phase in the factory. As the signals are not transmitted on the bus individually, they are not directly assigned to slots. The signals are rather assigned to frames and the frames are assigned to slots. Thus, the minimization of USS is taken care of in slot allocation stage, after the frames are packed. The details are given in Section 2.5.2

It must be noted that USS depends highly on U. Having a higher bandwidth allocation for a certain signal set inherently means that there is a need for a higher number of slots. That is, a slot allocation algorithm can find the optimal solution only if the frame packing algorithm comes up with the optimal packing of frames (with minimum U). Thus, it can be stated that the first stage of the problem also aims at decreasing USS.

2.4.3 Scalability

This metric is trivial: The algorithm must have feasible run-times for large signal sets, which is the case in real-life problems.

2.5 Two Stages of SS Scheduling

The given signal set, FlexRay configuration and requirements of the SS scheduling problem are described in Section 2.3 and the performance metrics are defined in Section 2.4. The task is composed of two subproblems: (1) signal-to-frame assignment and (2) frame-to-slot assignment. In the first stage, signals are packed into frames with maximum efficiency and in the second stage frames are assigned to static slots so that minimum number of slots is used. The two stages are independent from each other, which means that different algorithms can be implemented for the two stages of a single SS scheduling problem. The performance of the second stage; though, may depend on the performance of the first stage.

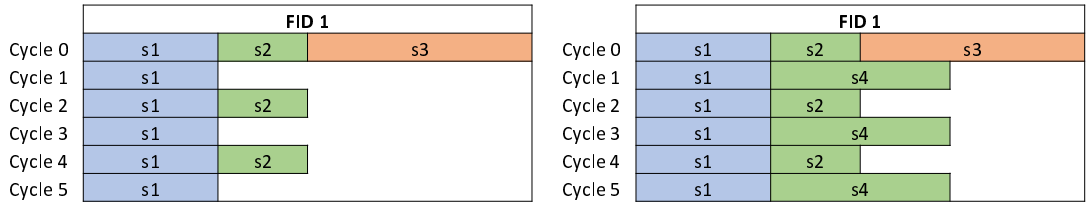
2.5.1 Frame Packing Problem

The first subproblem will be referred to as frame packing, where all the signals in the given signal set $\mathcal{S}^n$ are assigned into a set of frames $\mathcal{F}^n$. A frame f_f^n is the set of signals $s_s^n \in \mathcal{S}_f^n$ transmitted in one slot duration of a FlexRay cycle, meaning a signal s of a node n is assigned to a frame f. The size $b_{f_f}^n$ of frames is equal to $b \cdot 16$ bits + overhead, where b is the maximum number of data bits that can be sent in T_{STS}.

The period of a frame is $p_{f_f}^n$. In the approach we use, frame period is set to be the gcd of all the signals in $\mathcal{S}_f^n$, so that no jitter is introduced at this stage. As described in FlexRay protocol in section 2.1, in frame f_f^n, only the signals of node n can be transmitted.

The assignment of signals to message frames can be either fixed or variable: At each repetition of a frame, the same set $\forall s_s^n \in \mathcal{S}_f^n$ (fixed) or a different set of signals $\exists s_s^n \in \mathcal{S}_f^n$ (variable) can be transmitted. Fig. 2.3 illustrates fixed and variable frame

packing (Please note that the overhead bits are omitted). In the illustrated schedule,



(a) Fixed Packing (b) Variable Packing

Figure 2.3: Fixed vs Varaible Packing

it is assumed that there is only one message frame[3], which is f_1^n in Slot 1 and it has a period $p_{f_1}^n = 1$. In (a), the set of signals in frame f_1^n is $\{s_1, s_2, s_3\}$, with periods 1, 2 and 6. In this scheme, s_1, s_2 and s_3 can be transmitted at every FlexRay cycle (if they are ready to be sent). s_2 and s_3 can be ready once every 2 and 6 cycles, respectively, and no other signal is transmitted in the empty spaces in those cycles. No y-offset assignment of signals is required here since every signal has its own private region at every cycle, i.e. no interference by other signals. x-offset assignment is also trivial: Placing the signals next to each other is enough. In variable packing on the other hand, the set of signals in f_1^n is $\{s_1, s_2, s_3, s_4\}$. s_4 shares the region of s_2 and s_3, as can be seen in (b). x and y-offset assignment is required this time in order to prevent collisions.

The size limitations on fixed and variable packed frames also differ slightly from each other. For a fixed packed frame, one can easily write:

$$\sum\nolimits_{s_s^n \in S_f^n} b_s^n \leq b \cdot 16 \qquad \text{for } \forall f, n \tag{2.5}$$

whereas for the variable packed frames one must consider all cycles and write the above equation for each cycle separately.

Together with the definition of a frame, we can formulate the bandwidth demand of a signal and the bandwidth allocation for a frame, as described in Section 2.4. The fraction of bandwidth that is demanded by a signal is:

$$D_s^n := \frac{b_s^n}{p_s^n \cdot T_C \cdot C} \tag{2.6}$$

[3] Indeed, no other frames than f_1^n can be assigned to this slot, since the period is 1, i.e. the frame is repeated every cycle

where C is in bps, b_s^n is in bits, T_C is in seconds and p_s^n is unitless.

The fraction of bandwidth that is allocated to a frame is:

$$A_f^n \coloneqq \frac{b_{f_f}^n}{p_{f_f}^n \cdot T_C \cdot C} \tag{2.7}$$

$b_{f_f}^n$ is the number of bits that can be transmitted in one slot, whose duration is T_{STS} so we can replace the above equation to:

$$A_f^n \coloneqq \frac{b_{f_f}^n}{p_{f_f}^n \cdot T_C \cdot C} = \frac{T_{STS} \cdot C}{p_{f_f}^n \cdot T_C \cdot C} = \frac{T_{STS}}{p_{f_f}^n \cdot T_C} \tag{2.8}$$

The total bandwidth demand of all signals:

$$D \coloneqq \sum_{n=1}^{N} \sum_{s=1}^{S^n} D_s^n \tag{2.9}$$

The total bandwidth allocation for all frames:

$$A \coloneqq \sum_{n=1}^{N} \sum_{f=1}^{F^n} A_f^n \tag{2.10}$$

2.5.2 Slot Allocation Problem

In slot allocation stage, signals ar no more taken into consideration. We are given the set of frames from all nodes as input:

$$\mathcal{F} = \mathcal{F}^0 \cup \mathcal{F}^1 \cup \cdots \mathcal{F}^N \tag{2.11}$$

The nodes of the frames in the given set is not important anymore, thanks to the change (1) in FlexRay v3.0 (Section 2.2). Thus we can eliminate the node index from the notation of message frame variable f_f.

Our task is to assign each $f_f \in \mathcal{F}$ to a static slot with FID_t and set the y-offset to form an SS schedule. Each frame can be assigned to one and only one slot, but a slot can be allocated for multiple frames. y-offset defines the *base cycle* of the frame, i.e. the first cycle it is transmitted. This is required to avoid any conflict. A frame f_f is repeated once every p_{f_f} cycles in its assigned slot with the slot identifier FID_t. Since all the frame sizes are equal to the number of bits that can be transmitted in one T_{STS},

only one frame can be assigned to a single cycle of a slot, i.e. no slot multiplexing within a cycle.

Fig. 2.4 shows an example result of a frame to slot assignment for a given frame set of $\{f_1, f_2, f_3, f_4\}$ with periods being {2, 6, 1, 6}. The resulting schedule for each frame is represented with a tuple (FID, offset): {(1,0),(1,1),(2,0),(1,3)}. This gives the actual schedule to be implemented on the system.

	FID 1	FID 2
Cycle 0	f1	f3
Cycle 1	f2	f3
Cycle 2	f1	f3
Cycle 3	f4	f3
Cycle 4	f1	f3
Cycle 5		f3

Figure 2.4: Frames assigned to slots

The performance metric in slot allocation problem is the number of FIDs used. The frames are formed in the first stage and the frame size ($b \cdot 20 + O_F$ bits) is set. Accordingly, T_{STS} is also set by equation 2.1. Thus, finding the minimum number of FIDs is sufficient to find the minimum bandwidth used in static segment (USS) from equation 2.4. For the example in Fig. 2.4, $USS = 2 \cdot T_{STS}$.

CHAPTER 3

EXISTING FRAME PACKING AND SLOT ALLOCATION ALGORITHMS

3.1 Motivation

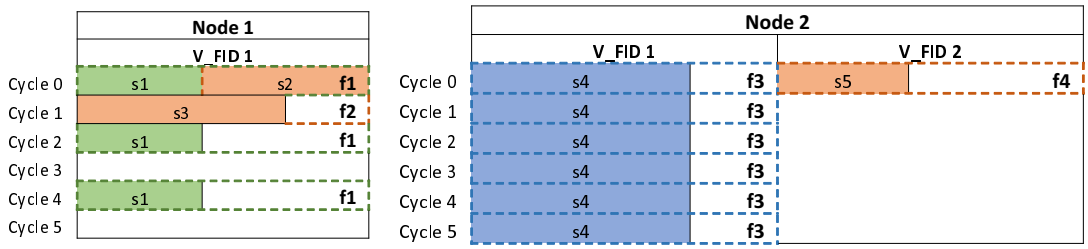
The communication on FlexRay bus is done by the transmission of groups of signals in frames of the SS. The signals have to be packed in frames and the frames have to be assigned to static slots. Theoretically the signals can be assigned directly to slots and pack the signals in each slot as frames by adding the transmission overheads. However, it is not a feasible approach to this problem, as shown in [4]. In their work, they construct a general ILP formulation which solves the two stages of the problem together. Signals are directly assigned to certain cycles of certain slots, rather than being packed into frames in a separate stage. The objective function of the formulation is to minimize the number of used slots (FIDs). Although this approach does not guarantee maximum efficiency, it finds solutions with significantly high utilization and with optimal USS. However, the downside of this solution is that the algorithm is not applicable to realistic problems, since it tries to solve a formulation in which all cycles of all slots are processed together with all the signals. The ILP is solved basically for all possible variables without any pre-processing, which results to impractical run-times.

This leads us to give up on single-shot approaches and go over other algorithms which deal with the two subproblems separately.

3.1.1 Frame Packing

The studies presented here have similar approaches to the frame packing problem:

(a) The authors of [4] divided the problem into two stages as described in Section 2.5. In this approach, the first stage is also divided into N separate problems, one ILP for each node. Such a division was possible due to the FlexRay restriction which prevents signals from different ECUs to be transmitted within the same cycle of the same slot. For each $n \in \mathcal{N}$, a separate ILP is solved. These ILPs are very similar to the general one-step formulation above. At each ILP(n), all $s_s^n \in \mathcal{S}^n$ are assigned to certain cycles of certain slots from the set of all slots (N_{STS} slots in total), as if there are no other signals from other nodes. The objective of each formulation is to minimize number of FIDs for each node. However, it is not the actual signal-to-slot assignment, it is rather a temporary assignment of signals into virtual slots to later pack frames. This algorithm uses variable-type packing, as illustrated in Fig. 3.1.



(a) Frames of Node 0

(b) Frames of Node 1

Figure 3.1: Signals packed into frames

In this example signal set, Node 1 has signals $\mathcal{S}^1 = \{s_1, s_2, s_3\}$, which are packed into frames f_1 and f_2. Please note that the content of f_1 is different in different cycles. Node 2 has signals $\mathcal{S}^2 = \{s_4, s_5\}$, which are packed into frames f_3 and f_4.

The idea of dividing the general formulation into smaller formulations brings the run-time down to feasible values, but it is still very high. (Up to several days for real-life examples).

(b) The algorithms and ILP solutions presented in [6] are implemented for FlexRay version 2.1, but the approach is similar. For each node $n \in \mathcal{N}$, all the signals in $\mathcal{S}^n$ are first packed into signals with an ILP formulation. The objective of the formulation is to minimize bandwidth utilization U. Since the frames are formed independently for each node, the very same frame packing approach would as well work for FlexRay v3.0 protocol. The formulation is simpler than the one in [4]: Only signals with equal periods are allowed to be packed together in the same frame, thus:

$$p^n_{f_f} = p^n_i = p^n_j = ... \qquad \text{for } \forall s^n_i, s^n_j \in \mathcal{S}^n_f \tag{3.1}$$

Due to this packing of signals with the same period, the frames are identical at each cycle. We can state that this method is a fixed-type packing by its nature. As shown in Fig. 3.2, frame f^n_1 consisting of s_1 and s_2 has period 2. s_1 and s_2 are transmitted at every repetition of f^n_1.

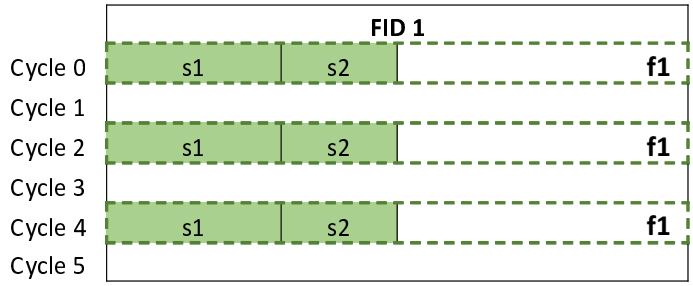


Figure 3.2: Signals packed with Schmidt Packing, fixed set of signals are repeated at each cycle

(c) The authors in [7] focuses on the frame packing stage of the problem only. They implement fixed packing of frames, based on the ideas in [6]. Rather than packing signals with same periods only, they propose a method to pack signals with different periods into fixed frames, using both ILP and heuristic methods. However, their ILP formulation is not consistent with the algorithm described in the paper. It is not clear which signals are allowed together into the same frame and how this is guaranteed. Thus, we skip this approach.

The approaches (a) and (b) (VP and FP, namely) are implemented in our study, as described in sections 3.2.1 and 3.2.2, respectively. The evaluation results are presented in Chapter 5 .

3.1.2 Slot Allocation

The studies presented here have similar approaches to the slot allocation problem:

(a) The study in [4], which is also referenced for their frame-packing algorithm in Section 3.1.1, employs an ILP formulation to find the optimal solution which gives the schedule with minimum number of FIDs. The formulation in this stage is very similar to their approach in the first stage. Instead of signals, frames $f_f \in \mathcal{F}$ are assigned to slots this time. Message frames from all nodes are processed together with all cycles of all available slots.

This approach, however, has no pre- or post-processing, it rather uses brute force to do the scheduling. The ILP formulation tries to place any frame into any cycle of any available slot at the same time. All the frames are processed together to make use of FlexRay version 3.0 which allows slot multiplexing among different ECUs. So the formulation has basically all possible variables in the formulation, which would lead to a high run-time. The implementation of this approach is explained in details in Section 3.3.1.

(b) Study in [6] also uses ILP to solve the slot allocation. In contrast to the free, unconstrained formulation of [4], this algorithm generates an ILP with a more refined search space, without any drop on the performance. The authors make use of frame periods for this. The idea is to differentiate slots with respect to frame periods and book each different slot exclusively for those groups of frames. The ILP formulation does not assign frames into slots, it rather calculates how many slots are needed for each differentiated group and how many frames will be in group. This algorithm has been developed and implemented for FlexRay v2.1. An ILP is solved and the minimum number of FIDs is found for each node. Summing them up gives the total number of FIDs in static segment. The same approach is adapted for v3.0 and the modified ILP formulation is implemented as described in Section 4.2.2.

3.2 Formulation of Frame Packing Algorithms

3.2.1 Variable Frame Packing ILP (VP)

In the approach by the authors of [4], variable frame packing is implemented via ILP formulation (This algorithm will be referred to as VP, throughout thesis). For each subproblem, a set of signals of a node is given as $s_s^n \in \mathcal{S}^n$. There are a total of N subproblems, each to be solved independently. Thus we can omit index n in constants and variables of individual ILPs. Each ILP is used to assign signals into certain cycles of certain slots (virtual slots indeed). The following set of constants is given:

- $v \in \mathcal{V} = \{1, 2 \cdots, N_{STS}\}$: Virtual slot index v
- $s \in \mathcal{S}^n$: Index for signals s_s of node n
- $c \in \mathcal{C} = \{0, 1, \cdots, k-1\}$: Cycle numbers c within scheduling window, defined by scheduling period k.
- $c_b \in \mathcal{C}$: Base cycle numbers (y-offsets). Defines the first cycle a signal is transmitted.
- b_s: Length of signal s_s, i.e. the number of bits in a signal.
- b: Slot length (Frame size). The number of 2-byte words transmitted within a frame [1].
- p_s: Period of signal s_s.

The variables in ILP formulation are defined as follows:

- $z_{(s,v,c_b)}$: Binary variable which indicates whether signal s is assigned to base cycle c_b of virtual slot v or not.
- y_v: Binary variable to mark if virtual slot v is used or not.

[1] Although the actual frame size is bf, here we use the term "frame size" here to refer to the sum of signal bits in the frame, which is $b \cdot 16$ bits maximum

Using the above constants and variables, ILP constraints to satisfy the frame packing problem can be written as:

$$\forall s \in \mathcal{S}^n : \sum_{v \in \mathcal{V}} \sum_{c_b=0}^{p_s-1} z_{(s,v,c_b)} = 1 \tag{3.2a}$$

$$\forall v \in \mathcal{V}, c \in \mathcal{C} : \sum_{c_b = c \bmod(p_s)} \sum_{s \in \mathcal{S}^n} z_{(s,v,c_b)} \cdot b_s \leq b \cdot 16 \tag{3.2b}$$

$$\forall v \in \mathcal{V}, s \in \mathcal{S}^n : \sum_{c_b=0}^{p_s-1} z_{(s,v,c_b)} \leq y_v \tag{3.2c}$$

and the objective function is to minimize the number of used virtual slots for node n:

$$\min_x \sum_{v \in \mathcal{V}} y_v \tag{3.2d}$$

Constraint 3.2b guarantees that each signal is assigned to one and only one frame (i.e. to a base cycle of a virtual slot). Constraint 3.2c states that the sum of all signal bits in a frame is not greater than the frame size. Finally, constraint 3.2c ensures that a virtual slot v is marked as used if there is any signal is assigned to it.

The solution of the above formulation states a tuple (v, c_b) for each signal, stating the virtual slot and base cycle it is assigned to. The assignment to base cycles can be expanded to assignment into cycles by making use of the period of the signals. This gives the set of signals to be transmitted at each FlexRay cycle for each virtual slot. Two signals are in the same frame's signals set if they are transmitted in at least one cycle. With this information, frames f_f are formed with the set of signals shown by $\mathcal{S}_f$, shown by the following expression:

$$\exists c \in \mathcal{C} : (z_{(i,v,c_b)} = 1) \cap (z_{(j,v,c_b)} = 1) \Longrightarrow s_i, s_j \in \mathcal{S}_f \tag{3.3}$$

where $c_b = c \bmod(p_s)$.

After the set of signals in each frame (S_f) is determined, an x-offset computation is required to set the transmission times of individual signals within frames, which is only needed to set the system properties. In other words, it does not affect the performance metrics. Thus it is not implemented in the scope of this work.

Since the frames are packed, the percentage of allocated bandwidth for all frames can

be calculated from Equation 2.10. Frame period is set to the gcd of all signal periods in that frame. This is necessary not to introduce any jitter in this stage.

The frame size b needs to be constant inside ILP formulations presented here. The designers of the approach [4] have kept frame size as constant ($b = 21$) outside ILPs, too. However, utilization metric depends on the choice of frame size since it is highly effective on how the signals can be packed into frames. To find the optimal solution, the whole packing algorithm has to be executed for each possible value of b. Algorithm 1 describes the method used in this work for this purpose.

Algorithm 1 Iterating Frame Packing ILP Over Frame Size

```
optimal_A = High
optimal_b = 0
for all 2 ≤ b ≤ 127 do
    Solve ILP
    if A < optimal_A then
        optimal_A = A
        optimal_b = b
    end if
end for
```

The number of variables for each ILP(n) for each frame size b is formulated as:

$$(k \cdot S^n + 1) \cdot N_{STS} \tag{3.4}$$

The number of constraints is:

$$(k + S^n) \cdot N_{STS} + S^n \tag{3.5}$$

3.2.2 Fixed Frame Packing ILP (FP)

The idea introduced in [6] is to pack signals with equal periods into fixed frames, with the objective of maximum bandwidth utilization (The approach will be called FP in short). Since the signals from different ECUs cannot be packed into same frames, the problem can be divided into N subproblems each to be solved with independent ILP formulations, similar to the first stage approach in [4].

The frame size b is again kept constant in ILP formulations. The frame periods are also fixed due to the packing of equal period signals. Consequently, the only way to decrease bandwidth allocation (Equation 2.10) is to have as few frames as possible.

A maximum of of $F^n = S^n$ frames are defined initially, i.e. one frame for each signal. A signal can be placed in any of the frames whose period p_{f_f} is equal to its period p_s. Let node n has the set of signals $\{s_1, s_2, s_3\}$ with periods $\{1, 2, 2\}$. The set of possible frames is then $\{f_1, f_2, f_3\}$ with periods $\{1, 2, 2\}$. s_1 can be placed in f_1 only. On the other hand, s_2 and s_3 can be packed in either f_2 or f_3. With the observation here, we can define set of signals allowed and set of messages for each period. The following sets are defined as follows:

- $\mathcal{P}^n$: Set of periods p of the signals in $\mathcal{S}^n$
- $\mathcal{S}_p^n$: Set of all signals $s \in \mathcal{S}^n$ with period p
- $\mathcal{F}_p^n$: Set of all frames $f \in \mathcal{F}^n$ with period p

Since each subproblem is solved independently, node index n can be omitted in the ILP formulations. The following constants are given:

- $s = 1, \cdots, S$: Signal index
- $f = 1, \cdots, F$: Frame index
- b_s: Signal bits of s
- b: Frame size, i.e. the number of 2-byte words

Following variables are defined for the formulation:

- $x_{(s,f)}$: Binary variable indicating whether signal s is packed in frame f or not
- y_f: Binary variable indicating whether frame f is used or not

The constraints of the ILP are:

$$\forall p \in \mathcal{P}, s \in \mathcal{S}_p : \sum_{f \in \mathcal{F}_p} x_{(s,f)} = 1 \tag{3.6a}$$

$$\forall p \in \mathcal{P}, f \in \mathcal{F}_p : \sum_{s \in \mathcal{S}_p} x_{(s,f)} \cdot b_s \leq y_f \cdot b \cdot 16 \tag{3.6b}$$

and the objective is to minimize the bandwidth allocation:

$$\min_{x} \sum_{p \in \mathcal{P}} \sum_{f \in \mathcal{F}_p} \frac{y_f \cdot T^b_{STS}}{p \cdot T_c} \tag{3.6c}$$

Constraint 3.6b ensures that each signal is assigned to exactly one frame. The total number of bits in a frame is limited by frame size with constraint 3.6b.

The frames are packed once the ILPs of all nodes are solved. y-offset assignment is not needed since the frames are fixed to the same group of signals at every cycle. x-offset assignment is done simply by placing each signal next to each other within frame.

Each ILP solution gives the minimum possible bandwidth allocation for the packing of equal period signals. The total bandwidth allocation (obtained by adding up all N results) is still optimal.

The number of variables in FP is in the range of [S^n,$(S^n)^2$)], depending on the signal set. The minimum case is when all signals have different periods and the maximum case is when all signals have the same period. The number of constraints is always equal to $2 \cdot S^n$.

The same procedure described in Algorithm 1 is used to find the optimal result among possible frame sizes.

3.3 Formulation of Slot Allocation Algorithms

3.3.1 Integration ILP (ISA)

The slot allocation approach in [4] uses an ILP formulation which is very similar to their frame packing approach presented in Section 3.2.1. This algorithm will be referred to as "Integration Slot Allocation" (ISA). The differences are as follows:

- Variable size **signals** are assigned to **virtual slots** in frame packing, whereas fixed size **frames** are assigned to **real slots** in slot allocation. This is imple-

mented by changing the signal variables and indices to frame variables:

$$s \rightarrow f$$

$$p_s \rightarrow p_{f_f}$$

$$b_s \rightarrow \text{removed}$$

$$z_{(s,v,c_b)} \rightarrow z_{(f,v,c_b)}$$

$$\text{virtual slot} \rightarrow \text{real slot}$$

- In frame packing, **N** separate ILPs are solved for of each node; whereas in slot allocation a **single** ILP is solved for all frames.

The formulation is therefore:

$$\forall f \in \mathcal{F} : \sum_{v \in \mathcal{V}} \sum_{c_b=0}^{p_{f_f}-1} z_{(f,v,c_b)} = 1 \tag{3.7a}$$

$$\forall v \in \mathcal{V}, c \in \mathcal{C} : \sum_{c_b = c \bmod (p_{f_f})} \sum_{f \in \mathcal{F}} z_{(f,v,c_b)} \leq 1 \tag{3.7b}$$

$$\forall v \in \mathcal{V}, f \in \mathcal{F} : \sum_{c_b=0}^{p_{f_f}-1} z_{(f,v,c_b)} \leq y_v \tag{3.7c}$$

and the objective function is to minimize the number of used slots in SS:

$$\min_x \sum_{v \in \mathcal{V}} y_v \tag{3.7d}$$

Constraint 3.2b guarantees that each frame is assigned to one and only one base cycle of a slot. Constraint 3.2c asserts that a maximum of one frame is assigned to a base cycle of a slot. Finally, constraint 3.2c ensures that a slot v is marked as used if there is any frame assigned to it.

This formulation is theoretically able to find a solution (if there is any) with minimum slot allocation for any given signal set. The calculation of the problem size is also similar to VP algorithm. Replacing S^n with F in equations 3.4 3.5 gives the number of constraints and variables in Integration ILP.

CHAPTER 4

DEVELOPED ALGORITHMS

In this chapter, we describe the algorithms we have developed for both frame packing and slot allocation problems. FMP algorithm is an improved version of FP in [6], to increase the bandwidth utilization. FMP+ algorithm uses the output of FMP solution and applies a heuristic post-processing on it to transform the fixed frames into variable frames. SSA is an adaptation of message scheduling ILP in [6] to use in minimum slot allocation problem for FlexRay v3.0.

4.1 Frame Packing

4.1.1 Motivation

The ILP solutions presented in Section 3.2 can be used to solve frame packing problem. The optimal solution would pack the frames such that the bandwidth utilization is at its maximum possible value. Nevertheless, they have some shortcomings which can be improved further to achieve a higher bandwidth utilization.

Each of the N ILP solutions in the algorithm implemented in Section 3.2.1 manages to pack signals into frames such that a minimum number of FIDs is required for each single node. However, FlexRay v3.0 allows frames from different ECUs to be placed in the same static slot. The frames formed in virtual nodes in separate ILP solutions can be combined in the same slot in the resulting SS schedule. Therefore, less number of FIDs will be needed:

$$FID_{tot} \leq \sum_{n=1}^{N} FID_n$$

Although the utilization VP algorithm is very high (See Section 5), the optimal solution is not guaranteed. That is because the objective of subproblems do not build up to the objective of the overall problem. While packing the frames of one node, the fact that the frames from other nodes can also be scheduled in the same slot at the end is neglected. This may increase the allocated bandwidth for a frame where the packed signals' periods are not integer multiples of each other. This is illustrated by an example in Fig. 4.1

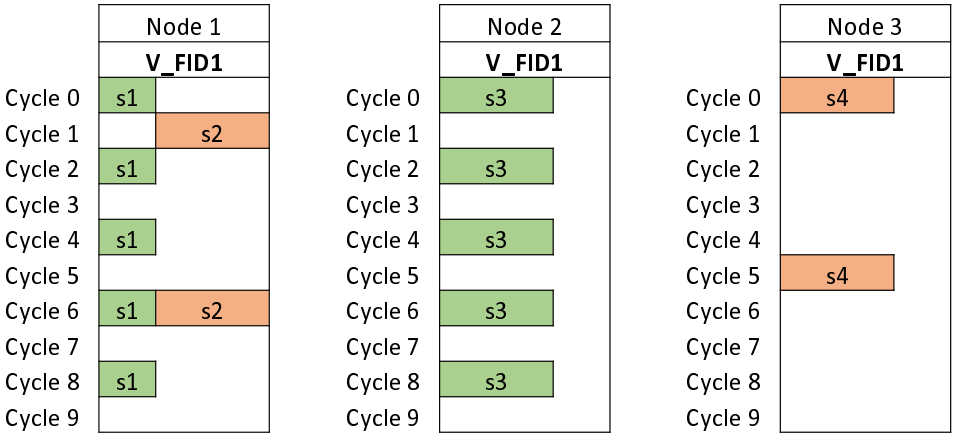
Consider the signals $s_1, s_2 \in \mathcal{S}^1$, $s_3 \in \mathcal{S}^2$ and $s_4 \in \mathcal{S}^3$ of 3 nodes on a FlexRay network. The periods of the signals are p_1 = 2, p_2 = 5, p_3 = 2 and p_4 = 5. A possible solution of frame packing is shown in (a). Please note that the objective is to minimize the number of virtual FIDs for each node. s_1 and s_2 are placed in the same virtual slot and they are packed in the same frame since they are transmitted in same frame in Cycle 6. The frame period of f_1 is therefore gcd(2,5) = 1. From Equation 2.10, the percentage of bandwidth allocation of the solution is $\left(\frac{1}{1} + \frac{1}{2} + \frac{1}{5}\right) \cdot \frac{T_{STS}}{T_c} = 1.7 \cdot \frac{T_{STS}}{T_c}$.

If the slot allocation stage is applied to this set of frames, the result would look like the one in (b), in which a total of 3 FIDs are used.

If the frame packing problem for the same signal set had been solved with the objective of minimum bandwidth allocation for each node, then s_1 and s_2 would have been placed in separate frames. The result would have looked like the one in Fig. 4.2.

The bandwidth allocation A becomes $\left(\frac{1}{2} + \frac{1}{5} + \frac{1}{2} + \frac{1}{5}\right) \cdot \frac{T_{STS}}{T_c} = 1.4 \cdot \frac{T_{STS}}{T_c}$ for the optimal solution. If a slot allocation stage is applied to this set of frames, a total of 2 FIDs are used, as shown in (b). Thus, the solution found from the variable frame packing ILP proposed by Sagstetter et. al. is not optimal in terms of neither U nor USS. A better approach would be to use the maximum utilization as an objective function in the problem formulation for single nodes.

In addition to that, even if there was only a single node, this approach still would not pack the frames with the optimal utilization, since there is no such optimization for that in VP [4]. Assume that there are two possible solutions, both using the same



(a)

	FID 1	FID 2	FID 3
Cycle 0	f1	f2	f3
Cycle 1	f1		
Cycle 2	f1	f2	
Cycle 3	f1		
Cycle 4	f1	f2	
Cycle 5	f1		f3
Cycle 6	f1	f2	
Cycle 7	f1		
Cycle 8	f1	f2	
Cycle 9	f1		

(b)

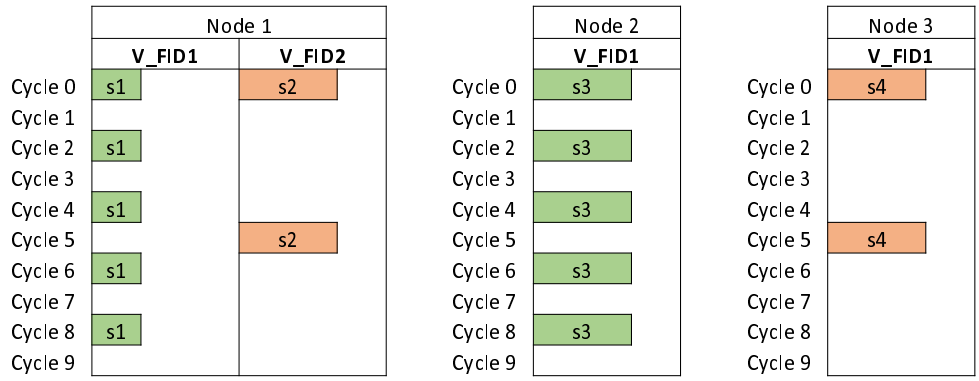
Figure 4.1: The frames are packed as in (a), and the resulting optimal slot allocation is shown in (b)

number of virtual nodes, but one of them leading to a higher utilization. This approach would pick any of them, regardless of the utilization.

The FP implementation by Schmidt [6], on the other hand, aims at minimizing the bandwidth allocation for the given signal set for each ECU. The result of each ILP(n) has direct impact on the overall outcome. That is, the solution of one ILP is not affected by solutions of other ILPs:

$$A_{tot} = \sum_{n=1}^{N} A_n$$

Although the optimal solution can be found, the performance would not always be as high as other approaches since this method of only packing signals with equal periods is not a very flexible method. The number of possible combination of signals



(a)

	FID 1	FID 2
Cycle 0	f1	f2
Cycle 1	f3	f4
Cycle 2	f1	
Cycle 3	f3	
Cycle 4	f1	
Cycle 5	f3	f2
Cycle 6	f1	f4
Cycle 7	f3	
Cycle 8	f1	
Cycle 9	f3	

(b)

Figure 4.2: The frames are packed as in (a), and the resulting optimal slot allocation is shown in (b)

in frames is very limited.

⇒ Our motivation is to come up with an algorithm that targets high utilization for each node, and is also more flexible than Schmidt's FP [6]. In addition, it is intended to keep the problem set small. The approach is described in Section 4.1.2, which is developed by modifying FP.

4.1.2 Fixed Frame Packing With Multiple Periods (FMP)

FP in [6] packs signals into fixed frames, where all the signals have equal periods. Our algorithm, referred to as Fixed, Multiple Period Packing (FMP) is very similar to FP. The idea of FMP is to pack signals into fixed frames as in FP, but also to allow signals with different periods to be in the same frame. The condition for a signal to

be packed in a frame is that the signal period p_s has to be an integer multiple of the frame period p_{f_f}.

Similar to [6], a separate ILP for each node is solved independently. For each ILP, a maximum of of $F^n = S^n$ frames are defined initially, i.e. one frame for each signal. The following sets are defined as follows:

- $\mathcal{P}^n$: Set of periods p of the signals in $\mathcal{S}^n$
- $\mathcal{S}_p^n$: Set of all signals $s \in \mathcal{S}^n$ with period $p_s = p$
- $\mathcal{F}_p^n$: Set of all frames $f \in \mathcal{F}^n$ whose period is an integer divider of p:

$$\mathcal{F}_p^n = \{f \in \mathcal{F}^n \mid p_f \bmod p = 0\}$$

$\mathcal{F}_p^n$ represents the set of frames in which a signal s with period p can be packed. Since an ILP can run for each node independently, the index n can be omitted in the ILP formulation. The rest of the formulation is similar to FP in Section 3.2.2. The list of constants are:

- $s = 1, \cdots, S$: Signal index
- $f = 1, \cdots, F$: Frame index
- b_s: Signal bits of s
- b: Frame size, i.e. the number of 2-byte words

The following variables are defined for the formulation:

- $x_{(s,f)}$: Binary variable indicating whether signal s is packed in frame f or not
- y_f: Binary variable indicating whether frame f is used or not

The constraints of the ILP are:

$$\forall p \in \mathcal{P}, s \in \mathcal{S}_p : \sum_{f \in \mathcal{F}_p} x_{(s,f)} = 1 \tag{4.1a}$$

$$\forall p \in \mathcal{P}, f \in \mathcal{F}_p : \sum_{s \in \mathcal{S}_p} x_{(s,f)} \cdot b_s \leq y_f \cdot b \cdot 16 \tag{4.1b}$$

and the objective is to minimize the bandwidth allocation:

$$\min_{x} \sum_{p \in \mathcal{P}} \sum_{f \in \mathcal{F}_p} \frac{y_f \cdot T^b_{STS}}{p \cdot T_c} \tag{4.1c}$$

Although the formulation is similar to FP, the impact on the solution is significant. Since there are more possible frames a signal can be packed into, the frame packing is more flexible to achieve higher bandwidth utilization.

The range on the number of variables and constraints is the same as in FP. This time the average number of variables is higher; though, leading to a bigger problem set and longer run-times than FP.

4.1.3 FMP and Post Processing (FMP+)

The ILP approaches to the frame packing problem can find optimal solutions, however they have higher problem sets, thus longer run-times. Heuristic approaches on the other hand are very fast, but they do not guarantee optimal solution. The Fixed Frame Packing ILP With Multiple Periods + Post Processing (FMP+) algorithm presented here employs both methods together.

Variable frame packing as demonstrated in 2.5.1 provides the most free assignment possible. Making use of this method can be advantageous to increase bandwidth utilization. As it is illustrated in Fig. 2.3 in Section 2.5.1, variable packing allows more signals to be packed in frames, which decreases the number of frames and allocation. Yet, variable frame packing implementation as ILP creates a huge problem set, making the solution impractical.

To implement variable frame packing in a practical method, we have developed FMP+ algorithm. This algorithm performs a heuristic post-processing on the solution of FMP method in Section 4.1.2 to convert the fixed frames into variable frames by adding more signals to them. After this conversion is applied, some frames become redundant. Hence they are removed from the set of frames, decreasing the total bandwidth allocation of the solution. Fig. 4.3 illustrates this conversion. Consider an example set of signals $\{s_1, s_2, s_3\} \in \mathcal{S}^n$ with $p_1 = 1$ and $p_2, p_3 = 2$. The ILP solution of FMP method packs signals into f_1 with $p_{f_1} = 1$ and $p_{f_2} = 2$ as shown in (a). At

every second instance of frame f_1, no signal is transmitted in the time allocated exclusively for s_2, i.e. the bandwidth is wasted. We apply a post-processing that checks all the signals in other frames to see if they can fit into this empty space. The signals satisfying the condition are placed into the frame. After all the frames are processed, some of the frames can be left without any signals in them. Then they are discarded from the set of frames $\mathcal{F}^n$.

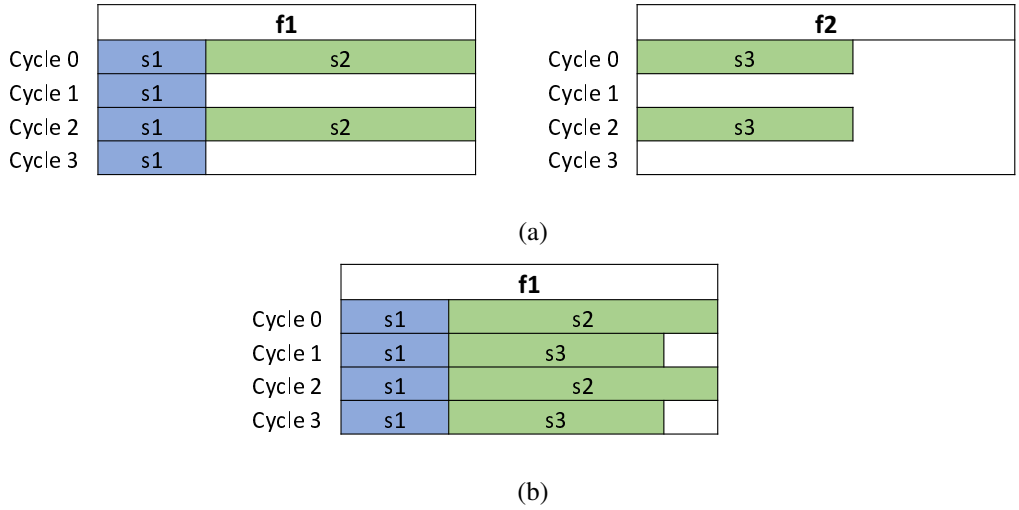


Figure 4.3: The resulting frames f_1, f_2 of (a) FMP and (b) FMP+. s_3 is placed in f_1 with post processing

As a result of this post-processing step, the frames are not fixed anymore since the allocated times for signals in a SS are overlapping now. The post-processing is applied after each ILP(n), independent of the other ILPs, as always. The pseudo code of the overall algorithm is given in Algorithm 2. First the frames are sorted w.r.t their periods (2). Then iteratively for each nonempty frame, the frame period is calculated again and an "empty map" $E_{(f,r)}$ is created (4-7). $E_{(f,r)}$ shows the amount of empty bits in each repetition r of frame f, as illustrated in Fig. 4.4. (4,5) are necessary because some signals of frame f may have been taken and placed in other frames in previous iterations. If all the signals are taken, f is deleted (22). Then for each signal that is not in f and that can be placed in frame f [1] (8-9), we look for suitable spaces in the empty map $E_{(f,r)}$ (10-11). If such space is found, the signal is placed in f (12), the empty map is updated (13-14) and the program moves on to the next signal (17). Consider the example in 4.4 where signals s_1, s_2, s_3 with periods $s_1 = 2$,

[1] This algorithm is implemented only for scheduling period of k = 64, 32, 16 etc where the only prime multiplier of the period is 2. Therefore, if $p_f \leq p_s$, then s can be packed in f, no other condition is needed

$s_2 = 4$, $s_3 = 16$ are packed in frame f_1 with $p_{f_1} =$. Assume that the scheduling period is $k = 16$. If assigned to a slot, f_1 is to be transmitted once every 2 cycles. The transmitted data at each instance of f_1 is shown on the left. At each repetition r of f_1, there will be a certain amount of empty space, which is represented with $E_{(f_1,r)}$. The resulting empty map is shown on the right.

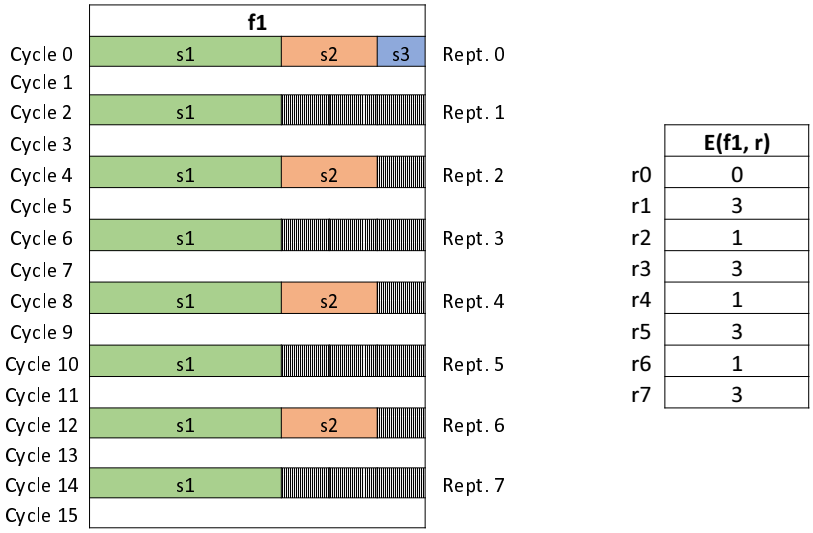


Figure 4.4: Creation of Empty Map $E_{(f,r)}$ for f_1

FMP+ algorithm is only implemented for scheduling period of $k = 2^x$. When all the signal periods is a power of 2, there is very little restriction on signals to be packed in other frames, since the prime period is the same. The possibility of finding other signals to fill the empty bits is easier. In other words, the effect of post processing is expected to be more significant in this case, compared to other scheduling periods. The results presented in Section 5 supports this expectation. Nevertheless, a generic design of FMP+ will be implemented as a future work.

Algorithm 2 FMP+: Post Processing on FMP

for all $n \in N$ **do**
  Solve ILP(n) as in FMP
  Generate a list L^n s.t. $f \in \mathcal{F}^n$ are sorted w.r.t p_f in increasing order
  for all $f \in L^n$ **do**
    if $\mathcal{S}_f^n \neq \varnothing$ **then**
      $p_f \leftarrow$ gcd $(\forall s \in \mathcal{S}_f^n)$
      Create $E_{(f,r)}$
      for $1 \leq s \leq S^n$ **do**
        if $(s \notin S_f^n)$ and $(p_f \leq p_s)$ **then**
          for $0 \leq r \leq \frac{p_s}{p_f} - 1$ **do**
            if $b_s \leq E_{(f,r)}$ **then**
              $\mathcal{S}_f^n \leftarrow \mathcal{S}_f^n \cup s$
              for $0 \leq i \leq p_s - 1$ **do**
                $E_{(f,\, r+i\cdot\frac{p_s}{p_f})} \leftarrow E_{(f,\, r+i\cdot\frac{p_s}{p_f})} - b_s$
              end for
            end if
            break
          end for
        end if
      end for
    else
      $\mathcal{F}^n \leftarrow \mathcal{F}^n - f$
    end if
  end for
end for

4.2 Slot Allocation

4.2.1 Motivation

The Integration ILP of Sagstetter et. al. described in Section 3.3.1 is capable of finding the optimal solution, however the problem set of the formulation is huge, which leads to impractical run-time. Shortcomings of the formulation in [4] are:

- Frames with the same properties (repetition) are enumerated explicitly. However, from the scheduling perspective, only the number of frames with a certain repetition is relevant (such frames can be arbitrarily exchanged).
- FIDs are explicitly enumerated. However, from the scheduling perspective, the exact placement of FIDs is not needed, only the number of used FIDs is essential.
- Slot allocation is explicitly computed, which is unnecessary. It is only neccessary to determine the type of frames to be assigned in an FID together.

The message scheduling algorithm in [6] formulates an ILP which finds the minimum frame allocation for a single node. We found that the same formulation can be used for interleaving frames of different nodes in v3.0. The algorithm guarantees the minimum slot allocation. Besides, it has very few variables and constraints in the formulation which doesn't even depend on the signal set.

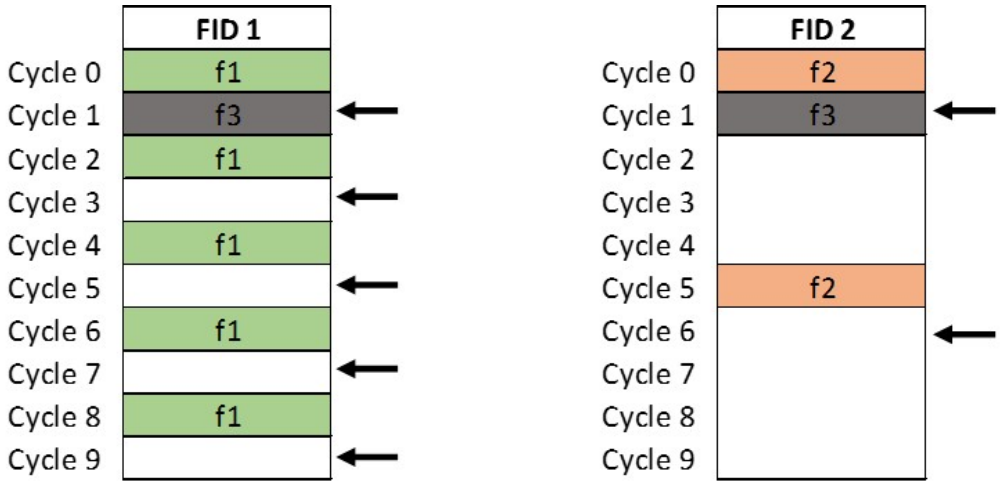
$\Rightarrow$ Our motivation is to adapt the ILP formulation in [6] to v3.0. The method is expected to find the optimal solution in small time.

4.2.2 Schmidt Scheduling ILP for v3.0 (SSA)

As proved in [6], frames that are not multiples of each other cannot be scheduled in the same slot. The idea they proposed is to rewrite frame periods as multiplication of prime numbers and group them in slots accordingly.The following example illustrates their approach.

Example: Let us have frames f_1 and f_2 with periods $p_{f_1} = 2$ and $p_{f_2} = 5$. They cannot be put together, thus have to be scheduled in separate slots. f_1 fills up half $\left(\frac{1}{2}\right)$ of slot FID_1 and f_2 fills up one fifth $\left(\frac{1}{5}\right)$ of slot FID_2. This can be expressed as: "A frame with period p_f can fill $\frac{1}{p_f}$ of a (1) group or all of a (p_f) group, where p_f is a prime number. A (p_f) group represents a partition of a slot where a frame is transmitted every p_f cycles. If we had a frame f_3 with $p = 8$, it could fill $\frac{1}{8}$ of a (1) group, $\frac{1}{4}$ of a (2) group, $\frac{1}{2}$ of a (2,2) group or all of a (2,2,2) group.

The reason of naming groups as (2,2) instead of (4) can be understood better if we add frame f_3 to the above example, with p_{f_3} = 10. f_3 can be in group (2,5) or (5,2), which are composed of different groups of frames. If we choose it to be in (2,5) group, it would end up filling $\frac{1}{5}$ of a (2) group in FID_1, as shown in Fig. 4.5 (a). If we choose it to be in (5,2) group, on the other hand, it would fill half of a (5) group in FID_2, as shown in Fig. 4.5 (b).



(a) (b)

Figure 4.5: In (a), f_3 is labeled as (2,5) group, so it fills one fifth of a (2) group. In (b), it is labeled as (5,2) group, it covers half of a (5) group.

Let us add f_4 with $p_{f_4} = 20$ to the example set. We now have to decide whether it will be in group (2,2,5), (2,5,2) or (5,2,2). If we choose the first group, it can be placed together with frame f_1 only and would fill $\frac{1}{5}$ of a (2,2) group. If we choose the second, it can be placed with f_1 and possibly f_3 (depending on the choice of it). If we choose the third option, it can be scheduled in the same slot with f_2 and again, possibly with f_3. We can arrive at the following conclusions from this observation:

- The impact of the choice for one frame on the overall solution highly depends on the choice for other frames, too. Therefore, a sequential process would not guarantee an optimal solution. An ILP is required.
- The possible selection of slots for a frame is limited by the choice of groups for frames. This significantly reduces the problem set, which is neglected in [4].
- An ILP can be written to determine the groups of frames first. The objective in this decision would be to use minimum number of slots.
- The number of used slots is also set once these decisions are made, because there would be no more variables left in the problem.

This approach that is originally developed for the frame allocation on FlexRay v2.1 is adapted to the slot allocation on FlexRay v3.0 in this thesis. It is done rather than running a separate ILP for each node, we now have to run a single ILP for all frames to find the optimal solution. This algorithm will be referred to as "Schmidt Slot Allocation" (SSA).

Writing a generic ILP formulation would be a complex, time consuming job due to the nature of the approach. On the other hand, writing an application specific ILP would be much easier since there are very few variables and constraints. In addition, this formulation only depends on the scheduling period, which can take a very limited number of different values[2]. Thus, we have preferred developing the ILP directly for the test cases in this work, where $k = 40$ and $k = 64$ are used. A scheduling period of 64 would not require any ILP solution, indeed. Since all the frame periods are powers of 2, all signals can be scheduled together in the same slot, if possible. It means that the groups of frames are already set, hence no variables to decide on. For $k = 40$, the allowed frame periods is {1,2,4,5,8,10,20,40}

The objective function is to minimize the number of FIDs:

$$\min_{x}(n_{FID}(1) + n_{FID}(2) + n_{FID}(5)) \tag{4.2}$$

where $n_{FID}(p)$ is the number of FIDs allocated for (p) group. $n_{FID}(1)$ is equal to the number of frames with $p_{f_f} = 1$, so it is already fixed. The terms are written as:

[2] For the latest FlexRay protocol, there are only 12 different scheduling periods, 7 of which do not require any ILP formulation with this approach. Thus, writing 5 different formulations is enough

$$n_{FID}(2) = \left\lceil \frac{n_{(2)}}{2} + \frac{1}{2}\left(\left\lceil \frac{n_{(2,2)}}{2} + \frac{1}{2}\left(\left\lceil \frac{n_{(2,2,2)}}{2} + \frac{1}{2}\left\lceil \frac{n_{(2,2,2,5)}}{5}\right\rceil\right\rceil + \left\lceil \frac{n_{(2,2,5)}}{5} + \frac{1}{5}\left\lceil \frac{n_{(2,2,5,2)}}{2}\right\rceil\right\rceil\right)\right\rceil + \left\lceil \frac{n_{(2,5)}}{5} + \frac{1}{5}\left(\left\lceil \frac{n_{(2,5,2)}}{2} + \frac{1}{2}\left\lceil \frac{n_{(2,5,2,2)}}{2}\right\rceil\right\rceil\right)\right\rceil\right)\right\rceil$$

$$n_{FID}(5) = \left\lceil \frac{n_{(5)}}{5} + \frac{1}{5}\left\lceil \frac{n_{(5,2)}}{2} + \frac{1}{2}\left\lceil \frac{n_{(5,2,2)}}{2} + \frac{1}{2}\left\lceil \frac{n_{(5,2,2,2)}}{2}\right\rceil\right\rceil\right\rceil\right\rceil$$

The variables $n_{(2,2)}$, $n_{(2,5,2)}$, $n_{(5,2,2,2)}$ etc. represent the number of frames in each related group. Frames with $p_f = 40$, for instance, can be in either of four different groups. If we have 10 such frames, $n(5, 2, 2, 2)$ of them will be in group (5,2,2,2), and the remaining 10 - $n(5, 2, 2, 2)$ will be in other available groups. The number of frames in all groups will be set once the ILP formulation is solved. Table 4.1 lists all variables and explains the corresponding group properties. $n_{(2,2,5)}$, for instance is the number of frames with p_f = 20 in group (2,2,5); each such frame fills one-fifth of a (2,2) group.

Table 4.1: Group Properties

Variable	p_f	Comment
$n_{(2)}$	2	Fills $\frac{1}{2}$ of a full slot
$n_{(5)}$	5	Fills $\frac{1}{5}$ of a full slot
$n_{(2,5)}$	10	Fills $\frac{1}{5}$ of a (2) group
$n_{(5,2)}$	10	Fills $\frac{1}{2}$ of a (5) group
$n_{(2,2,5)}$	20	Fills $\frac{1}{5}$ of a (2,2) group
$n_{(2,5,2)}$	20	Fills $\frac{1}{2}$ of a (2,5) group
$n_{(5,2,2)}$	20	Fills $\frac{1}{2}$ of a (5,2) group
$n_{(2,2,2,5)}$	40	Fills $\frac{1}{5}$ of a (2,2,2) group
$n_{(2,2,5,2)}$	40	Fills $\frac{1}{2}$ of a (2,2,5) group
$n_{(2,5,2,2)}$	40	Fills $\frac{1}{2}$ of a (2,5,2) group
$n_{(5,2,2,2)}$	40	Fills $\frac{1}{2}$ of a (5,2,2) group

The constraints are as follows:

- $n_{(40)} = n_{(2,2,2,5)} + n_{(2,2,5,2)} + n_{(2,5,2,2)} + n_{(5,2,2,2)}$

- $n_{(20)} = n_{(2,2,5)} + n_{(2,5,2)} + n_{(5,2,2)}$
- $n_{(10)} = n_{(2,5)} + n_{(5,2)}$

meaning that the total number of frames with period p_f must be equal to the number of frames in each possible group for this p_f .

The constants are: $n_{(2)}$, $n_{(5)}$, $n_{(2,2)}$, $n_{(2,2,2)}$, $n_{(2,2,2,2)}$, $n_{(10)}$, $n_{(20)}$ and $n_{(40)}$. Please note that the number of frames with p_f = 2,4,5 and 8 are constants, since there is only a single possible group for each.

The ceiling operators in the objective function makes the formulation nonlinear. A linearization can be done on this nonlinear program by replacing all the ceiling operators with linear expressions. Two new variables are introduced for each substitution:

$$\left\lceil \frac{n_{(2,5,2,2)}}{2} \right\rceil \quad \longrightarrow \quad K_{(2,5,2,2)} \cdot 2 = k_{(2,5,2,2)} + n_{(2,5,2,2)}$$

The whole formulation is transformed as such and all the constraints and variables are obtained. There are 16 constraints and 35 variables in total.

Once the grouping of all frames is determined, these frames are assigned to allocated slots. The slots are filled with the frames from each group, starting from the smallest group [3], as in [23]. This stage is only to display the schedule of the system. It does not effect the performance, thus it is skipped in our work.

[3] Groups which cannot be filled with other groups are called smaller. (2,2,5,2) is one of the smallest groups, for instance, since there are no frames with period 200 or 80 to fill a portion of that group.

CHAPTER 5

PERFORMANCE EVALUATION

We present the evaluation methods we used and the results of the algorithms implemented in this study. The algorithms are compared with each other in terms of the defined performance metrics. Based on the observations, the discussions and the planned future works are mentioned, as well.

5.1 Environment

We have conducted all the experiments on an Intel i5 5200U CPU with 8 GB RAM and with Windows 10 OS. All the algorithms are programmed on MATLAB environment version R2015b [24]. TOMLAB / CPLEX optimization tool v12.1 [25] is used as ILP solver. The size of the matrices created for ILP formulations are very large so they can consume up to hundreds of gigabytes of memory. Since these matrices are very sparse (i.e. most of the values are zero), we were able to define them as sparse matrices, which decreased the memory requirements down to kb~mb range for each.

CPLEX has numerous options to improve the performance of the ILP solver. While testing the algorithms, we have observed that the run-time of the ILP solutions increase exponentially with the number of signals per node. Even for the smallest set where there are 10 signals per node, the tests can last more than a day for a single node. We set EPGAP to 0.01 (default = 10^{-4}) meaning that the CPLEX solver can stop if it finds a solution proved to be within 1% of the optimal solution. This causes the solutions to be suboptimal, but it also helps to quickly get that result. In addition, we set TILIM to 10, which forces the solver to stop running after 10 ms. This limits

the time for each solution. For all frame packing algorithms, an ILP is employed for each node and the whole process is repeated for each T_{STS}. There are 84 possible values of T_{STS}, thus the run-time of the algorithm is limited to $N \cdot 84 \cdot 10ms$ (excluding other steps of the program), which is 10 hours at max. The utilization, though, is not effected very much by this time limit. For a set of 900 signals, the utilization differed only by 1% for TILIM = 10 and 100 cases. Putting a time limitation is a must, indeed, since the solution may take significantly long times for certain inputs, which are not predictable.

We assume that the following configuration of FlexRay bus is given: $T_{MT} = 3\mu s$, gdBit = 1 and $O_F = 90$ bits. We assumed that all the signal periods are multiples of 5 ms, thus T_c = 5 ms, and $T_{c,SS}$ is taken as 3162 μs, to be consistent with the study in [4].

The algorithms related to the two stages of SS scheduling problem are evaluated separately in sections 5.2 and 5.3.

5.2 Frame Packing

We have implemented four different algorithms for frame packing stage (VP, FP, FMP and FMP+) and evaluated the results according to U, USS (equations 2.3 and 2.4) and run-time. run-time is defined as the total time spent in frame packing stage of an algorithm for a single test case For computing USS, we use SSA, which gives optimal result as discussed in Section 4. Note that we are only interested in the result of USS here. We perform a comparison of different algorithms for computing the optimal USS in Section 5.3.

In fact USS cannot be calculated in this stage, it is rather determined in slot allocation stage. However, we will discuss this metric in frame packing part since it depends highly on this stage. In addition to that, the algorithms used in second stage are capable of finding the optimal slot allocation for all cases (Both can do it theoretically, one of them can always find it in feasible run-time, as shown in Section 5.3). Thus we can interpret this metric as: "Using **x** algorithm for **y** case, the minimum possible amount of used static segment is **z**. You can implement either of the two algorithms

(ISA or SSA) to find this value". For practical reasons, we picked SSA algorithm to find the minimum possible USS value for each different frame packing algorithm.

All the algorithms here are tested under scheduling period k = 40 and k = 64 as scheduling period, except for FMP+ algorithm, which is only implemented for $k = 64$ case. Among possible scheduling periods, we picked 40 since it is the period that allows the largest number of different signal periods to be scheduled together (1, 2, 4, 5, 8, 10, 20, 40). This makes the frame packing more diverse and flexible. We have conducted simulations with five different types of test cases:

5.2.1 Test Case 1:

The number of nodes is fixed to $N = 30$ while the number of signals per node (S^n) is increased for each synthetic test case. The algorithms are tested for 300, 600, 900, 1200 and 1500 signals in total. For each number, 5 test cases are randomly generated with the probabilistic distributions of the signal periods and the number of signal bits given in Fig. 5.1 and the average of the results are recorded. The used distributions are from an automotive case study, retrieved from the [4].

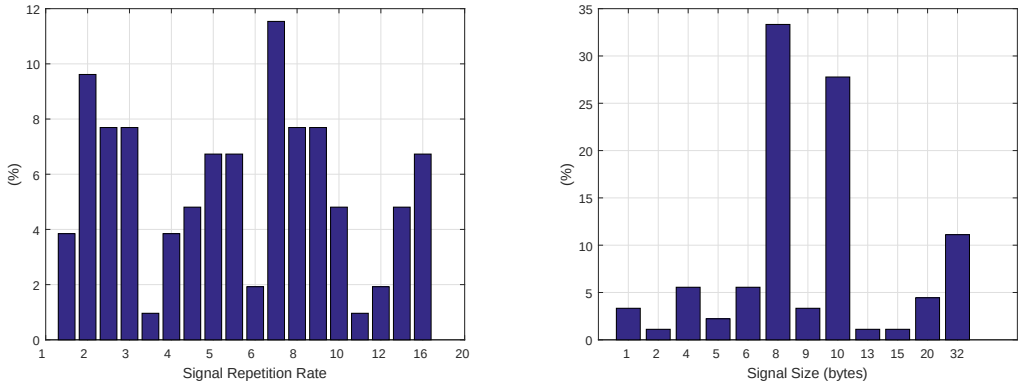


Figure 5.1: Distribution of signal periods and signal sizes [4]. These distributions are used in test case 2 and test case 3, too.

The total bandwidth demand D (Equation 2.9) for the signals in these test cases are shown in Fig. 5.2. D is equal for different scheduling periods since the demand is set by the input, not on the algorithm.

The bandwidth utilization graphs are shown in Fig. 5.3. It must be noted that these

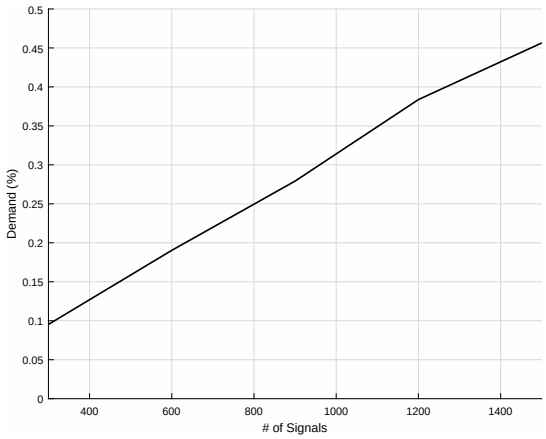
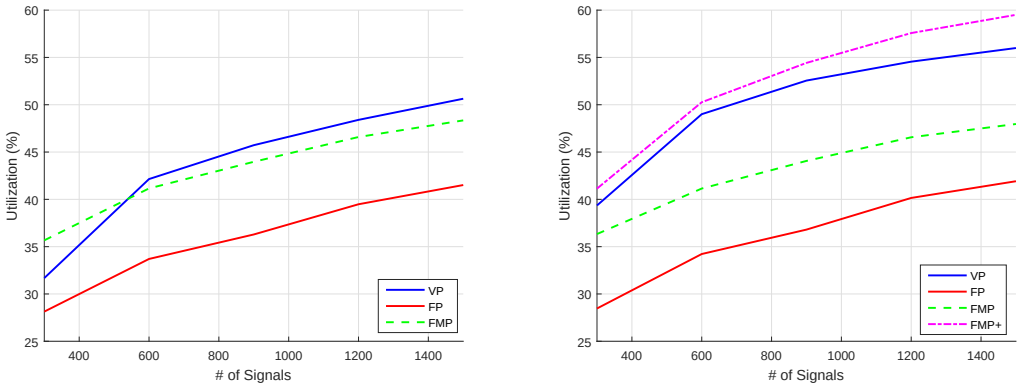


Figure 5.2: N = 30, S^n = 10, 20, 30, 40, 50



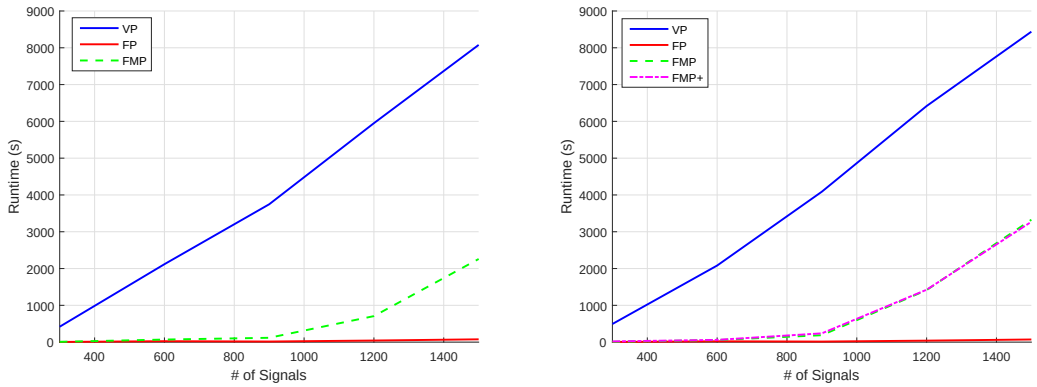
(a) k = 40

(b) k = 64

Figure 5.3: Utilization for fixed $N = 30$

are not the optimal results to be found by these algorithms, due to the two control options we used in the ILP solver, as described in Section 5.1.

For all tests FP gives the lowest utilization as expected. Since very few signals can be packed, frames have large empty spaces in them. FMP, as the improved version of FP, finds better solutions with the increased possibilities for packing signals together. It can even compete with variable packing of frames for $k = 40$. cases. For $k = 64$ case on the other hand, almost all the signals can be combined, thus VP outperforms the algorithms in which the frames are fixed. Although FMP+ is not supposed to find the optimal solution (has a heuristic stage), it still has higher utilization than VP. That is because VP is also not optimal, since the objective function used for each separate sub ILP solution does not contribute to the overall solution independent of the other solutions. For all the algorithms, the utilization increases with increasing S^n. As there



(a) k = 40 (b) k = 64

Figure 5.4: run-time for fixed $N = 30$

are more signals in each ILP, the diversity of the signals increase, thus the packing possibilities also increase. It is less likely now to waste space in frames.

The run-times of the simulations are presented in Fig. 5.4. More than 99% of the run-time is spent during ILP solution, as observed by MATLAB Profiler, which depends highly on the size of the problem set.

The problem size of each ILP(n) formulation in VP approach can be calculated from equations 3.4 and 3.5. In our test cases, the scheduling period k is (40 or 64), number of signals for each node $\mathcal{S}^n$ is between [1,50] and the number of static slots N_{STS} ranges from 15 to 264 for different values of b. It means that there can be up to 844,800 variables 30,000 in the problem set [1]. Thus, the constraint matrix can be as large as 25 billion units. This makes the algorithm unfavorable in terms of scalability.

The calculation of the problem size of ILP formulations in FP,FMP and FMP+ (see Section 3.2.2) are exactly the same. The number of constraints can be as large as 100 and the number of variables is between [50, 2500]. For FMP and FMP+ algorithms, the number of variables has always been between 1450-1550. Even the maximum size (250,000) of the problem is much smaller than the one in VP. The solution does not scale to very large theoretical values, but it can be used for practical examples of current IVNs.

Although the run-time is expected to increase exponentially with S^n, we observe a

[1] The number of static slots is limited to 1023, according to the FlexRay protocol standards [5]

linear graph after a certain limit in Fig. 5.4. This is due to the TILIM option of 10 ms.

5.2.2 Test Case 2:

Unlike test case 1, the number of signals per node is fixed $S^n = 30$ and the number of nodes is increased this time. The algorithms are tested for 300, 600, 900, 1200 and 1500 signals in total. The distribution of signal periods, sizes and the number of test cases are the same as in test case 1.

The total demanded bandwidth D for the signals in these test cases are shown in Fig. 5.2.

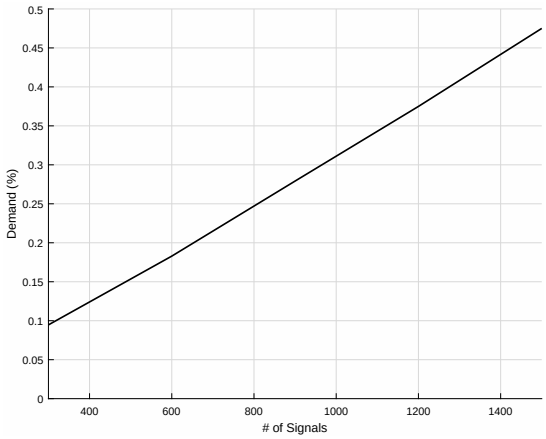
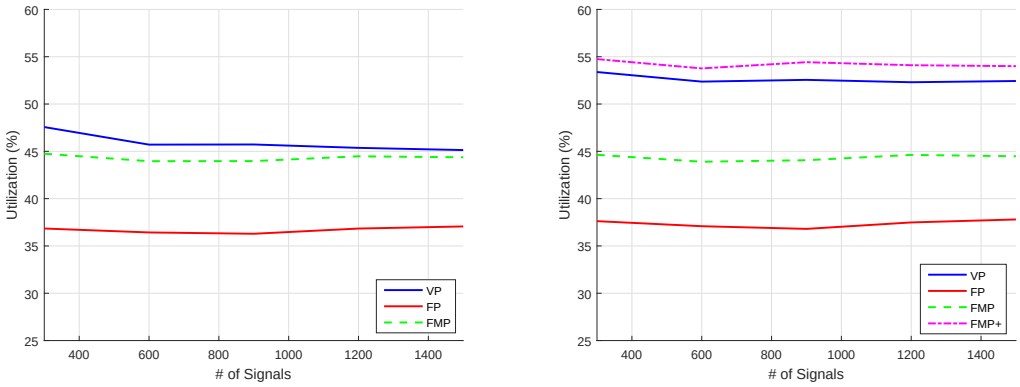


Figure 5.5: S^n = 30, N = 10, 20, 30, 40, 50

The bandwidth utilization graphs are shown Fig. 5.6.

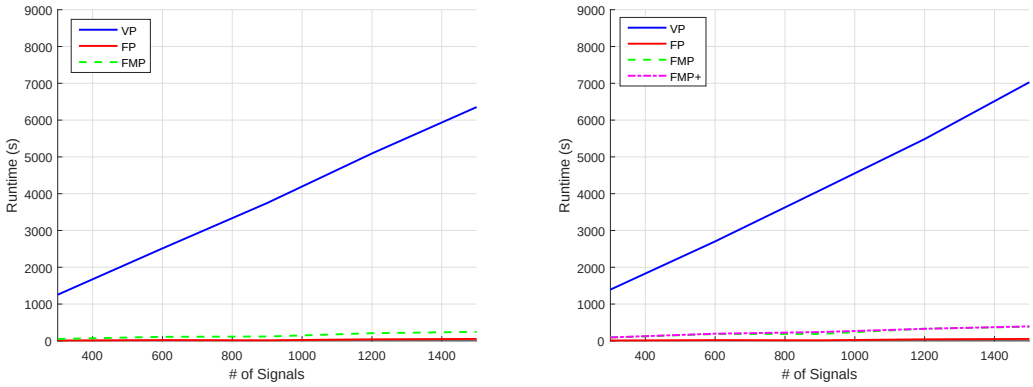


(a) k = 40 (b) k = 64

Figure 5.6: Utilization for fixed $S^n = 30$

The comparison between algorithms w.r.t. U is similar to the results of test case 1. Variable packing approaches outperform fixed approaches.

For FP, FMP and FMP+, the utilization remains pretty much unaffected from the number of nodes. Except for VP, in which the objective function is to minimize the number for used slots for a single node. The utilization in VP solution has a slight tendency to decrease with increasing N, in Fig. 5.6. If there was a single node, then the result would be the optimal in terms of USS, and also better in terms of U, intuitively. When the number of nodes starts increasing, then the disturbance on the overall result also goes up, even though the individual ILP solutions still perform optimally, w.r.t their objective functions.



(a) k = 40 (b) k = 64

Figure 5.7: Run-time for fixed $S^n = 30$

The run-times of the simulations are presented in Fig. 5.7. Since each ILP(n) is solved separately, the complexity of individual ILP formulations do not depend on N. The complexity of each ILP(n) is the same, thus the run-times are also similar. Number of nodes only effects the total run-time:

$$\text{Total run-time} \cong N \cdot t_{ILP(n)}$$

This is the case when the algorithm is solved sequentially for each node. If all ILP(n) are processed in parallel is used, the run-time of the algorithm would be equal to the run-time of a single ILP. It can be stated that all the algorithms are scalable to number of signals S, if S^n is not increased.

5.2.3 Test Case 3:

The distribution of signal periods and signal sizes are the same as in test cases 1 and 2. This time, S^n is not the same for all n. S^n is randomly picked for each node, using the distribution given in Fig. 5.8. This will be used as a benchmark to show the performance of the algorithms from different perspectives.

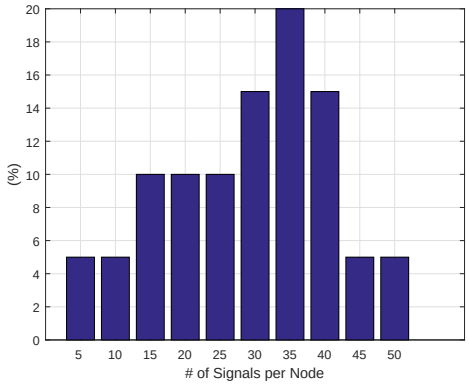


Figure 5.8: Distribution of Signals to Nodes for test case 3

The selection of possible S^n values are taken from real life cases, by taking SAE benchmark set [22] as a sample and by extending it so as to have a total of 900 ~1000 signals. 10 test cases are generated of this type. The average number of signals is calculated as 932 [2]. The average results obtained from these 10 test cases are presented in Table 5.1.

Table 5.1: Benchmark Results

	k = 40			k = 64			
	VP	FP	FMP	VP	FP	FMP	FMP+
Utilization (%)	45.1	37.1	44.2	51.3	37.8	44.4	**52.9**
USS (us)	3241	3928	3310	2847	3858	3294	**2776**
Run-time (s)	4003	34	362	4328	**33**	650	659

Bandwidth utilization of the solutions are consistent with the ones in test case 1 and 2. The amount of SS used is in accordance with U, confirming our statement that the optimal value of USS is also set by optimal U. The maximum U gives the minimum USS (FMP+ algorithm, k = 64). The run-times again, follow the same behavior as

[2] There are 932 signals in the benchmark set of [4]. Although we wanted to test under similar sizes, it is coincidental that the average size of our set is also 932.

explained in test case 1 and test case 2.

We can compare the simulation results obtained from a similar test case by Sagstetter et. al. in [4] with the one presented here, for k = 64 case (Since they did not experiment with k = 40) in Table 5.2.

Table 5.2: Comparison of Results from Two Different Works

	VP	VP in [4]
USS (us)	**2847**	3519
Run-time (s)	4328	**907**

The run-time of VP found here is much higher than the study in [4]. This is due to two reasons:

- The computation capacity of their hardware is higher than the one used in our work
- The measurement of run-time is different. The frame size is fixed in [4] to 42 and accordingly T_{STS} to 51 μs, whereas we have found the best solution for all possible T_{STS} values. The run-time defined in our work is the sum of all 84 different runs. In addition to that, we used a time limit option on ILP solvers, which decreases the actual run-time of the algorithm.

One can discuss the need for iteration over frame size. Table 5.3 shows the duration of static slots which give the solution with minimum utilization among all 84 possible values. As the table shows, the optimal T_{STS} value is not always 51 μs, which proves that it necessary to iterate over frame size. The USS value in Table 5.2 also shows that the solution obtained for this single T_{STS} value may not be feasible ($T_{c,SS} = 3162\,\mu s$) while there are feasible solutions for other values of it. Though, the range of possible values can be limited, based on the observations.

5.2.4 Test Case 4:

To observe the effect of the distribution of p_s and b_s over the performance for large sets, we generated 10 test cases of test case 4. This time, the probability of signal

Table 5.3: T_{STS} of Optimal Solutions

	k = 40			k = 64			
TSTS (us)	VP	FP	FMP	VP	FP	FMP	FMP+
avg	57	43	51	55	43	51	67
min	51	42	51	51	42	45	51
max	66	51	51	63	51	57	81

periods and signal sizes are distributed randomly over the set of values in Fig. 5.1. The distribution of S^n is the same as in Fig. 5.8. The results are very similar to the ones in test case 3, as shown in Table 5.4.

Table 5.4: Performance of Frame Packing for Uniform Distribution of p_s and b_s

	k = 40			k = 64			
	VP	FP	FMP	VP	FP	FMP	FMP+
Utilization (%)	45.6	36.5	44.69	51.2	37.3	44.6	**52.8**
USS (us)	3671	4583	3754	3278	4486	3770	**3188**
Run-time (s)	4739	53	590	5015	**49**	742	948

5.2.5 Test Case 5:

To observe the effect of p_s and b_s in a more extreme example, we used SAE benchmark set [22] and extended it to have a set as large as 932 signals by generating a large set with the same distributions as in [6] and [7]. The signal periods are taken only as 1, 2 and 40 with probabilities 0.5, 0.125 and 0.375, respectively. All the signals in this set has 8 bits of data. Since all the signals are schedulable within $k = 40$, we also have the chance to observe the effect of k being other that 64, for such a signal set. The results presented in Table 5.5, however, states that even for a test case so suitable to schedule within 40 cycles, the performance is barely higher than for k = 64 case (For the other types of test cases, k = 64 always gives better performance). Thus, this feature of v3.0 is questionable in terms of efficiency[3].

[3] It can be effective in terms of jitter, though, which is not in the scope of this study.

Table 5.5: SAE Benchmark

	k = 40			k = 64			
	VP	FP	FMP	VP	FP	FMP	FMP+
Utilization (%)	**37.4**	28.8	35	37.3	28.6	34.8	37.3
USS (us)	**1147**	1482	1215	**1147**	1488	1230	1149
Run-time (s)	1663	**39**	57	1816	**39**	57	70

5.2.6 Test Case 6:

A further performance metric that can be considered is jitter, i.e. the delay variance in transmission times. Jitter is not the focus of this thesis, however application layer protocols may have jitter requirements. We excluded the solutions where some jitter is introduced to the transmission of signals, since we intended to develop our algorithms so as to make it suitable for most applications. Nevertheless, we perform a small experiment here to compare the performance of two versions of VP implementation: with and without jitter.

Although the authors of [4] do not define jitter as a performance metric, they actually allow it in their implementation of VP. The ILP formulation does not change w.r.t jitter allowance, only the resulting frame period is changed, which effects the overall result. Consider two frames s_1 and s_2 with periods $p_1 = 2$ and $p_2 = 5$. If these signals are packed in the same variable frame, the frame period will have to be gcd(p_1, p_2) = 1 to have zero jitter. However, it can also be set to min(p_1, p_2) = 2. In this case, the time between two instances of s_2 transmissions will be different every time. As an extended study, we also implemented VP as described in [4].

Table 5.6: Variable Packing with Jitter

VP with jitter	k =40	k=64
Utilization (%)	55.8	51.5
USS (us)	2676	2897
Run-time (s)	4066	4402

The utilization of the solution increases significantly in this case, as shown in Table 5.6. The solutions presented here are obtained for the test cases of test case 3. A comparison of these results with the ones in Table 5.1 shows that FMP+ algorithm

outperforms "VP with jitter" in all metrics, even though FMP+ packing has limited possibilities for packing since it does not allow jitter. For k = 40 case we did not implement any other variable packing algorithm than VP. FMP+ will also be implemented for this case as future work to analyze the results in terms of both utilization and jitter.

The evaluation of these 4 (VP, FP, FMP and FMP+) frame packing algorithms have shown that the variable frame packing provides a high utilization of bandwidth, if formulated with a well defined ILP. The ILP in VP approach of [4] does not guarantee the optimal result since the objective function does not target the optimal result. In addition, it has larger run-times since the problem size is bigger. FP, FMP and FMP+ has similar ILP formulations, which all have much smaller problem sizes (Table 5.7). The run-time of an algorithm with ILP formulation greatly depends on the size of the matrices for the ILP. Thus, we measure the complexity in matrix size. Let S be the maximum number of signals of any node n. Whereas VP has a complexity of $O(S^2 \cdot {N_{STS}}^2)$, the other algorithms have a complexity of $O(S^3)$, in terms of the number of signals and the number of static slots. S is in the order of N_{STS} (usually smaller), thus the complexity of fixed packing algorithms are smaller than VP.

FP and FMP do not perform as good as VP. However with FMP+, one can not only benefit the small problem size of fixed packing approach's ILP but also the high bandwidth utilization of variably packed frames.

Table 5.7: ILP Problem Size of Frame Packing Algorithms

	# of constraints	# of variables	Max Size
VP	$(k+S) \cdot N_{STS} + S$	$(k \cdot S + 1) \cdot N_{STS}$	$25 \cdot 10^9$
FP, FMP, FMP+	$2 \cdot S$	[S , S^2]	$25 \cdot 10^4$

5.3 Slot Allocation

The evaluation of the methods used for the second stage is executed as a seperate test procedure. ISA and SSA algorithms are implemented after the frames are packed in FMP, FMP+ and VP algorithms. The algorithms are tested under 10 cases of test case 3.

Both algorithms are supposed to find the optimal solution, which is the minimum USS. However, the ILP formulation of ISA is very large, hence it can have very high run-times. The run-time depends highly on the number of frames. The number of frames (F), though, is not known before the frame packing stage. For each algorithm, F can get very different values, which affects the run-time. Table 5.8 shows the number of generated frames after a frame packing stage.

Table 5.8: Number of Frames Created

	k = 40			k = 64			
# of frames	VP	FP	FMP	VP	FP	FMP	FMP+
avg	133	427	264	191	412	266	114
min	88	338	247	152	329	224	**79**
max	178	**469**	283	224	457	326	172

The minimum (79) and maximum (469) number of frames define the boundaries of the size of the constraints matrix of ISA. We can calculate the range of N_{STS} from Table 5.3 as 50 ~60, usually. Using these values, we can calculate the range of the problem size using equations 3.4 and 3.5 which is from **1.3 to 55 billion** units. This is even larger than the ILP sizes in frame packing stage. Such large problems could lead to very high run-times and very large memory requirements as well. Thus we have to put time limits on ILP solver again, possibly leading to suboptimal results.

The SSA algorithm, on the other hand, has very small problem size, which is presented in Section 4.2.2. For the case of $k = 40$ the ILP constraints matrix has only 16 constraints and 35 variables, resulting in a constant ILP problem size of 560 units. For $k = 64$ case, there are no variables in the algorithm, thus no ILP is needed.

It is practical to find the optimal results of SSA and try to achieve the same results with ISA, using different time limits. Table 5.9 shows how many of the found solutions can actually fit into SS ($USS \leq T_{c,SS}$). As it can be seen, ISA can also find all possible feasible solutions in 100 seconds. However the USS values are still not always optimal, as Table 5.10 suggests.

If we increase the number of frames, we would need more static slots. If we have more static slots, then the run-time of ISA would increase more. We experimented both algorithms under different numbers of frame with $N_{STS} = 600$. Table 5.11

Table 5.9: Feasibility of Solutions

# of feasible solutions (out of 10)	k = 40			k = 64			
	VP	FP	FMP	VP	FP	FMP	FMP+
ISA (TILIM = 10s)	2	-	-	6	-	-	10
ISA (TILIM = 100s)	3	-	3	10	-	3	10
SSA	3	-	3	10	-	3	10

Table 5.10: USS of Feasible Solutions

Average USS (s) of feasible solutions	k = 40			k = 64			
	VP	FP	FMP	VP	FP	FMP	FMP+
ISA (TILIM = 100s)	**3049**	-	3043	2859	-	**3013**	2781
SSA	**3049**	-	**3026**	**2847**	-	**3013**	**2776**

shows the average results of these 5 tests for each case. After the number of frames goes past 150, the problem size gets huge and the ILP solver does not manage to find even a single solution in 100 seconds.

Table 5.11: The Run-times and Used FIDs for N_{STS} = 600

# of frames	20	50	100	150	200	250	500	1000
# of FIDs (ISA)	**10**	25.4	49.6	508.8	-	-	-	-
# of FIDs(SSA)	**10**	**24.6**	**48.8**	**68.4**	**93**	**116.8**	**230.8**	**460.2**
Run-time(s) (ISA)	45	101	102	105	102	102	102	105
Run-time(s) (SSA)	**0.01**	**0.01**	**0.01**	**0.01**	**0.02**	**0.02**	**0.02**	**0.02**

The run-time of SSA is always in the order of miliseconds, while in ISA it can theoretically go up to several days or weeks of simulation. Both algorithms are able to find the optimal results. So there is no advantage of using ISA in any way. We can even further suggest that there can be no other improvements for slot allocation problem. ISA does the job optimally, in practically no time.

Similar to frame packing stage, the complexity of the algorithms can be defined with the matrix size of the ILP formulation. The size of the problem formulation can be calculated with the same formula in VP with equations 3.4 and 3.5 just by replacing S term with F. From Table 5.12, we can write the complexity of ISA as $O(F^2 \cdot {N_{STS}}^2)$. For practical examples, this number can go op to the order of trillions of data units,

whereas the implementation of SSA for any scheduling period would have constant complexity of $O(1)$[4].

Table 5.12: ILP Problem Size of Slot Allocation Algorithms

	# of constraints	# of variables	Max Size
ISA	$(k+F) \cdot N_{STS} + F$	$(k \cdot F + 1) \cdot N_{STS}$	10^{12}
SSA	16	35	560

[4] If we had formulated a general ILP for SSA, the complexity would not have been constant. However, since the number of possible scheduling periods is very limited by FlexRay protocol standards, there is no need for a general formulation

CHAPTER 6

CONCLUSION

The topic of this thesis is the development of scheduling algorithms for the FlexRay static segment (SS) with the aim of a high bandwidth utilization and a low usage of the available bandwidth in the SS. In addition, the particular properties of the new version FlexRay v3.0 are taken into account. The developed algorithms are based on a two stage approach which makes it possible to compute FlexRay schedules with high bandwidth utilization in practicable run-times.

For the frame packing stage, different algorithms can be used for different problem sets and different requirements. When the signal sets get very large the variable packing (VP) algorithm from the recent literature becomes impractical to solve. Although the proposed FMP+ algorithm has a high complexity of $O(S^3)$, this theoretical computation time is not reflected in our experiments with real life problem sizes. It is found that FMP+ gives the best performance both in bandwidth utilization and run-time.

In our experiments, it is observed that the distribution of signals to nodes has a large effect on the performance of frame packing. For a node with a large set of signals, the utilization and run-time of all algorithms is high, whereby FMP+ gives the best results. When there are few signals, on the other hand, the utilization is low and all algorithms run fast. For signal sets where the distribution of signals to nodes has a high variance, a hybrid algorithm can be implemented, where different algorithms are used for different nodes. This is possible since the frame packing of each node is independent from the others. In fact, the overall run-time for a frame packing algorithm would be equal to the run-time for a single node, by using parallel computation.

Among the two algorithms implemented in the slot allocation stage, SSA (Schmidt Slot Allocation) outperforms ISA (Integration Slot Allocation) by far, in both performance metrics. On the one hand, the SSA solution is indeed the global optimum, where the used bandwidth gets the minimum possible value under the condition that the frames are packed with highest possible utilization. Furthermore, it runs in constant time, which is the minimum possible time complexity.

In the literature, jitter is also introduced as a performance metric for FlexRay scheduling. In a future work, it is intended to quantify jitter so that the scheduling algorithms can be compared with jitter as a performance metric. In our simulations, we also observed the effect of using scheduling periods other than 64 cycles. In our experiments, it turned out that there is no quantifiable effect on the bandwidth utilization. It is however expected that the choice of the scheduling period affects jitter. With an extensive study where jitter is measured as a performance metric, this feature of FlexRay v3.0 can be better understood. We finally note that the current version of FMP+ is implemented for a FlexRay scheduling period of 64 cycles, since this is the most frequently encountered case. In future work, FMP+ will also be implemented for other possible scheduling periods such as 40 cycles. Since FMP, which is the basic version of FMP+, performs almost as good as VP for that case, FMP+ is expected to outperform VP for other scheduling periods as well.

REFERENCES

[1] Sinem Coleri Ergen, Alberto Sangiovanni-Vincentelli, Xuening Sun, Riccardo Tebano, Sayf Alalusi, Giorgio Audisio, and Marco Sabatini. The tire as an intelligent sensor. *IEEE Transactions on computer-aided design of integrated circuits and systems*, 28(7):941–955, 2009.

[2] Nicolas Navet, Yeqiong Song, Francoise Simonot-Lion, and Cédric Wilwert. Trends in automotive communication systems. *Proceedings of the IEEE*, 93(6):1204–1223, 2005.

[3] Amos Albert. Comparison of event-triggered and time-triggered concepts with regard to distributed control systems. *Embedded world*, 2004:235–252, 2004.

[4] F. Sagstetter, M. Lukasiewycz, and S. Chakraborty. Generalized asynchronous time-triggered scheduling for flexray. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(2):214–226, Feb 2017.

[5] FlexRay Consortium. Flexray communications system protocol specification version 3.0.1. `https://tr.scribd.com/document/77021636/FlexRay-Protocol-Specification-V3-0-1`, October 2010. Accessed on January 2017.

[6] K. Schmidt and E. G. Schmidt. Message scheduling for the flexray protocol: The static segment. *IEEE Transactions on Vehicular Technology*, 58(5):2170–2179, Jun 2009.

[7] M. Kang, K. Park, and M. K. Jeong. Frame packing for minimizing the bandwidth consumption of the flexray static segment. *IEEE Transactions on Industrial Electronics*, 60(9):4001–4008, Sept 2013.

[8] Zdenek Hanzálek, David Beneš, and Denis Waraus. Time constrained flexray static segment scheduling. *RTN'2011*, page 23, 2011.

[9] S.P. Bradley, A.C. Hax, and T.L. Magnanti. *Applied Mathematical Programming*. Addison-Wesley Publishing Company, 1977.

[10] Özgür Kızılay. *Message scheduling for the static and dynamic segment of FlexRay: algorithms and applications*. MS Thesis, METU, 2015.

[11] Martin Lukasiewycz, Michael Glaß, Jürgen Teich, and Paul Milbredt. Flexray schedule optimization of the static segment. In *Proceedings of the 7th*

IEEE/ACM international conference on Hardware/software codesign and system synthesis, pages 363–372. ACM, 2009.

[12] Mathieu Grenier, Lionel Havet, and Nicolas Navet. Configuring the communication on flexray-the case of the static segment. In *4th European Congress on Embedded Real Time Software (ERTS 2008)*, 2008.

[13] K. Schmidt and E. G. Schmidt. Optimal message scheduling for the static segment of flexray. In *2010 IEEE 72nd Vehicular Technology Conference - Fall*, pages 1–5, Sept 2010.

[14] H. Zeng, M. Di Natale, A. Ghosal, and A. Sangiovanni-Vincentelli. Schedule optimization of time-triggered systems communicating over the flexray static segment. *IEEE Transactions on Industrial Informatics*, 7(1):1–17, Feb 2011.

[15] Haibo Zeng, Wei Zheng, M. Di Natale, A. Ghosal, P. Giusto, and A. Sangiovanni-Vincentelli. Scheduling the flexray bus using optimization techniques. In *2009 46th ACM/IEEE Design Automation Conference*, pages 874–877, July 2009.

[16] A. Darbandi, D. D. Mai, and M. K. Kim. Message scheduling on static segment of flexray networks. In *Third International Conference on Innovative Computing Technology (INTECH 2013)*, pages 321–326, Aug 2013.

[17] Shan Ding, Naohiko Murakami, Hiroyuki Tomiyama, and Hiroaki Takada. A ga-based scheduling method for flexray systems. In *Proceedings of the 5th ACM international conference on Embedded software*, pages 110–113. ACM, 2005.

[18] R. Lange, F. Vasques, P. Portugal, and R. S. de Oliveira. Guaranteeing real-time message deadlines in the flexray static segment using a on-line scheduling approach. In *2012 9th IEEE International Workshop on Factory Communication Systems*, pages 301–310, May 2012.

[19] Rodrigo Lange, Francisco Vasques, Rômulo S de Oliveira, and Paulo Portugal. A scheme for slot allocation of the flexray static segment based on response time analysis. *Computer Communications*, 63:65–76, 2015.

[20] Y. Xie, G. Zeng, H. Takada, and R. Li. Extensibility-aware message scheduling algorithm for the static segment of the flexray. In *2012 IEEE 15th International Conference on Computational Science and Engineering*, pages 508–515, Dec 2012.

[21] Thijs Schenkelaars, Bart Vermeulen, and Kees Goossens. Optimal scheduling of switched flexray networks. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*, pages 1–6. IEEE, 2011.

[22] C Class. Application requirement considerations. *SAE Recommended Practice J*, 2056, 1993.

[23] Can Öztürk. *Software tool development for the automated configuration of FlexRay networks for In-Vehicle Communication*. MS Thesis, METU, 2013.

[24] The Mathworks, Inc., Natick, Massachusetts. *MATLAB version 8.5.0.197613 (R2015b)*, 2015.

[25] Anders O. Goran Kenneth Holmstrom and Marcus M. Edvall. *User's Guide for TOMLAB/CPLEX v12.1*. TOMLAB Optimization.