Architectural Synthesis Techniques for Distributed Automotive System

Wei Zheng

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2009-73
http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-73.html

May 20, 2009

Acknowledgement

This research was supported by GSRC, CHESS and GM. I would like to first thank my advisor Alberto Sangiovanni-Vincentelli from Electrical Engineering and Computer Science at UC, Berkeley. He guided me into automotive domain and provided almost unlimited resources and supports for the research. The inspiration of this research comes mainly from the discussion with Marco Di Natale from Computer Science and Computer Engineering Department at Scuola Superiore S. Anna, Pisa. His tremendous guidance, feedback, and support are gratefully appreciated by the author. Finally, I would like to thank my family and friends, especially my wife Jing Yang from BWRC here at Berkeley , for their contributions in making my life in graduate school beautiful. The life in Berkeley will always

be good memories.

Architectural Synthesis Techniques for Distributed Automotive System

by

Wei Zheng

B.E. (Tsinghua University, Beijing, China) 2000
M.S. (University of California, Berkeley) 2004

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Engineering-Electrical Engineering and Computer Sciences

in the

GRADUATE DIVISION
of the
UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:
Professor Alberto Sangiovanni-Vincentelli, Chair
Professor Sanjit Seshia
Professor Robert Leachman

Spring 2009

The dissertation of Wei Zheng is approved:

Chair Date

Date

Date

University of California, Berkeley

Spring 2009

Architectural Synthesis Techniques for Distributed Automotive System

Wei Zheng

Abstract

Architectural Synthesis Techniques for Distributed Automotive System

by

Wei Zheng

Doctor of Philosophy in Engineering-Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Alberto Sangiovanni-Vincentelli, Chair

Automotive electronic subsystems support the execution of distributed safety- and time-critical functions on a complex networked system with several buses and tens of ECUs (Electrical Control Units). Complex functions, which are designed as networks of function blocks exchanging signal information, are deployed onto the physical HW and implemented in a SW architecture consisting of a set of tasks and messages. For example, an advanced braking system, implemented on a set of four ECUs, will take responsibility of applying brakes and tightening seat belts within 80 milliseconds when it senses danger.

The objectives of this thesis are to develop analysis and synthesis techniques for vehicle electronic system designers i) to analyze worse case situations, ii) to select appropriate mapping of functionality to architectural elements and iii) to set corresponding design parameters; making sure key functionalities finish before appropriate deadlines for safety-critical applications. The design of communication subsystems is essential in guaranteeing that timing constraints are satis-

fied. They can be either time-triggered (TimeTriggeredArchitecture (TTA) and FlexRay) or event-triggered such as CAN. Being able to accommodate incremental design changes while preserving a legacy design may reduce design and verification times substantially. For CAN systems, schedulability theory allows the analysis of the worst case end-to-end latencies and the evaluation of the possible architecture configurations options with respect to timing constraints, but it can also be used in the exploration of the software architecture configurations what can best support the target application. The optimization techniques presented in this thesis are based on ILP (integer linear programming) formulation combined with search algorithms and can derive implementations of both time-triggered and event triggered system that fulfill the design constraints. The techniques proposed are evaluated using industrial examples to prove the effectiveness of the work.

Professor Alberto Sangiovanni-Vincentelli
Dissertation Committee Chair

To my dear parents

Contents

List of Figures

List of Tables

List of Algorithms

Acknowledgments

This research was supported by the Gigascale Systems Research Center (GSRC), Center for Hybrid and Embedded Software Systems (CHESS), and General Motors (GM). I would like to first thank my advisor Alberto Sangiovanni-Vincentelli from Electrical Engineering and Computer Science at UC, Berkeley. He guided me into automotive domain and provided almost unlimited resources and supports for the research. The inspiration of this research comes mainly from the discussion with Marco Di Natale from Computer Science and Computer Engineering Department at Scuola Superiore S. Anna, Pisa. His tremendous guidance, feedback, and support are gratefully appreciated by the author. The author would like to thank many researchers and engineers from GM Warren technical center, and GM Bay Area Lab in Palo Alto; more specifically, Paolo Giusto and Thomas Fuhrman, for providing intensive discussion for the industrial case study. The author would also like to appreciate Claudio Pinello from Cadence Berkeley Lab for his guidance and discussion on the project; Haibo Zeng, Jike Chong and Qi Zhu from Department of Electrical Engineering and Computer Sciences, University of California at Berkeley for their discussion and work which help me understand mapping process and design methodology. Professors Shankar Sastry, Sanjit Seshia, Candace Yano and Alberto Sangiovanni-Vincentelli, my advisor, served on the qualifying committee and/or provided invaluable feedback and guidance in the dissertation writing process. Finally, I would like to thank my family and friends, especially my wife Jing Yang from BWRC here at Berkeley , for their contributions in making my life in graduate school beautiful. Without their support, I won't be able to smoothly get to this point and I'd like to owe all of my success here in Berkeley to my wife. The life in Berkeley will always be good memories.

Chapter 1

Introduction

Function development in Electronics, Controls, and Software-based (ECS) vehicle architectures has traditionally been component or sub-system focused. In recent years, there has been a shift from the single ECU approach towards an increased networking of control modules. The implications are an increased number of distributed time-critical functions and multiple tasks in execution on each ECU.

The starting point for the definition of a car electronic software system is the specification of the set of features that the system is expected to provide. A feature is a very high level description of a system capability (e.g. Cruise Control). Functional models are created from the decomposition of the feature in a hierarchical network of component blocks. The physical architecture model captures the topology of the car network, including the communication buses (e.g. FlexRay or CAN), the CPUs and the management policies (e.g. time-driven scheduling or priority-driven scheduling) that control the shared resources.

Hence, two levels of representation of the system are defined: the functional view and

the physical architecture view. The system designer must find a mapping of the functional architecture onto the physical architecture that satisfies the timing requirements (sensor-actuator deadlines). The mapping is performed at the time the SW implementation is defined, when the functions are implemented by a set of concurrent tasks and the communication signals are transferred in the payload content of messages. To provide design-time guarantees on timing constraints, different design and scheduling methodologies can be used. Avionics controls, for example, are often built based on static, time-driven schedules, examples of standards supporting this scheduling model are the OSEK Time system standard and the FlexRay bus arbitration model. Because of resource efficiency and ultimately price, many automotive controls are designed based on run-time priority-based scheduling of tasks and messages. Examples of standards supporting this scheduling model are the OSEK operating system standard and the Controller Area Network (CAN) bus arbitration model.

This thesis presents an integrated framework for design space exploration that leverages powerful mathematical programming to solve complex scheduling problem in a platform based design context.

This introduction is structured in three parts. In the first part, Section 1.1 presents the research contribution of the thesis. The second part, Section 1.2 describes the motivation of the work. The last part, Section 1.3 contains an overview of modern system followed by a real life automotive system example which will be extensively explained throughout the thesis.

1.1 Research Contribution

The thesis is performed in the context of the design of distributed software architectures for next-generation automotive controls, where the application performance requirements impose

constraints on end-to-end latencies in the execution of the control functions.

The thesis/research goal and effort center around Platform-Based Design (an approach to the new system theory) based system analysis and synthesis for distributed systems, and are embodied in: (1) Modeling of functionality and architecture (2) Mapping from functional model to architectural model (3) Scheduling of the mapped functionality units within the architecture units according to the performance constraints specified by the system designers.

Automated architecture exploration is necessary to improve design productivity for distributed (hard) real time systems especially for Analysis and Synthesis of Design Parameters. The scheduling/schedulability analysis centered system parameter synthesis techniques within a mathematical programming optimization framework could actually solve many problems efficiently where used appropriately. However, the synthesis framework that could facilitates automated architecture exploration applied to automotive systems is the main research contribution of this thesis.

1.2 Research Motivation

1.2.1 Research motivation from System Level Design methodology

Over the past several decades, embedded systems have evolved to a level of maturity and sophistication and appeared everywhere in our everyday life. The complexity of these systems is, however, reaching levels that were not even conceivable a few years ago: There are more than 80 processors in a new generation car that control its function and implement its entertainment and communication subsystems. The problem is how to systematically conduct the design, analysis and synthesis in this new regime. Stated in another way, is it possible to quickly and effectively provide a new system design methodology with the capability of handling the distributed embedded system

design? Is there a general and flexible design flow embedded in the methodology for a large variety of application domains? Towards this objective, I believe that the recent system level design methodology advances in Platform-Based Design (PBD) offer a unique and attractive solution. The idea for PBD is to call for a successive refinement approach where functionality (what the system is supposed to do) of the design is captured at the highest level of abstraction possible and mapped to a set of predefined solutions. Specifying a design includes identifying constraints such as latency, throughput, reliability, cost, and power consumption. The architecture is an interconnection of elements that are characterized by performance indexes coming from the abstraction of their implementation. The mapping process can be automated if both function and the architecture elements that form the platform are projected to a common semantic domain. The process repeats itself by interpreting the mapped design as a function at the next layer of abstraction and new architectural elements are introduced. Here at the Berkeley Center for Hybrid and Embedded Software Systems (CHESS) and the Gigascale System Research Center (GSRC), I have proposed and developed the common semantic domain for automotive systems with an emphasis on scheduling for PBD, which provides the system designers with the flexibility of performance analysis and parameter synthesis based on the schedulability analysis theory. Furthermore, I have explored the generality and limitations of this methodology, with respect to both bottom up and top down design flows widely used in embedded system design, and provide a practical guidance on meet in the middle design scheme with an emphasis on scheduling.

1.2.2 Research motivation from automotive applications

Platform Based Design was used in a variety of industrial applications: from automotive electronic design to communication systems and semiconductor chips. The promising case stud-

ies in vehicle design demonstrate the effectiveness of PBD in all dimensions. Today's vehicles are becoming increasingly more complex as consumers demand more and better features in their automobiles. Most new features are requiring additional electronic components and control software, constantly pushing the limits of existing architectures and design methodologies. The problems with traditional design methodologies are numerous, resulting in intermittent feature failures, significant testing effort, production delays, increased warranty costs, and consumer dissatisfaction. The solution to all of these problems starts with the adoption of a system level PBD. My research focuses on one of the important system performance, timing, which is impacted by a lot of factors but most in the mapping and scheduling phase during the system design. Based on the system-level PBD, I believe that recent mathematical advances in correlated formulations and solvers offer an attractive solution. The idea is to use a mathematical programming based optimization framework to handle system communication and synchronization models, tasks to Electronic Control Unit (ECU) placement, signals to message mapping and the assignment of priorities to tasks and messages in order to meet end-to-end deadline constraints and minimize latencies. At Berkeley, I collaborated with General Motors Advanced Technology Research Group and I have proposed an ILP based optimization framework to automate the above process: set up the activation model (Synchronous VS. Asynchronous) and leverage the trade-offs between the purely periodic and the data-driven activation models to meet the latency requirements of distributed vehicle functions. Additionally, it is not difficult to find out that the same approach could be leveraged to tasks placement, signals to messages mapping and priorities assignment to expand the automation process of the whole system design. At the same time, the framework is working well for time-triggered protocols which are used by safety critical applications by most of major automotive companies especially in the

consideration of incremental design. General Motors has adopted the methodology and the optimization framework for its electronic system design. The effectiveness of all of above approaches has been verified by a real case study from GM.

1.3 Modern System and Standard Overview

1.3.1 Modern Systems

Modern system development is getting more and more complicated and usually goes through phases from system analysis, system design and system implementation to system testing. System analysis (also called system specifications) requires the designer understand application requirements in both functional and non-functional aspects, namely they need to know what the system need to provide to the end users. System design goes through the definition of the design solutions (among all possible ones) that can solve the specifications problem. This part also includes selection of the HW/SW platform, including the operating systems and the resource management policies. The system implementation (coding) phase will focus on translating the design into packages or program modules, developed in a programming language. This part can be manually performed or the result of automatic model translations (provided the design model is formal). Finally the system testing is necessary for checking that the application satisfies all functional and non-functional requirements. The definition of several systems will be introduced to better understand the complication of modern design world.

1.3.1.1 Embedded Systems

According to Wikipedia [2], an embedded system is a special-purpose computer system designed to perform one or a few dedicated functions, often with timing constraints, that means embedded system is usually a real time system. It is often embedded as part of a complete device including hardware and mechanical parts. In contrast, a general-purpose computer, such as a personal computer, can do many different tasks depending on programming. Embedded systems control many of the common devices in use today.

Physically, embedded systems range from portable devices such as digital watches and MP3 players, to large stationary installations like traffic lights, vehicle controllers, or the systems controlling nuclear power plants. Complexity varies from low, with a single micro-controller chip, to very high with multiple units, peripherals and networks mounted inside a large chassis or enclosure.

In general, ”embedded system” is not an exactly defined term, as many systems have some element of programmability. For example, hand-held computers share some elements with embedded systems - such as the operating systems and microprocessors which power them - but are not truly embedded systems, because they allow different applications to be loaded and peripherals to be connected. Almost all of the ECUs in cars are embedded systems which usually have special purpose on processing or sensing/actuating.

1.3.1.2 Real Time Systems

There are many definitions for real time systems:

1. A real-time system is an interactive system that maintains an ongoing relationship

with an asynchronous environment i.e. an environment that progresses irrespective of the real time system.

2. (IEEE) Pertaining a system or mode of operation in which computation is performed during the actual time that an external process occurs, in order that the computation results may be used to control, monitor or respond in a timely manner to the external process.

3. A real-time system responds in a (timely) predictable way to (un)predictable external stimuli arrival.

4. Open, Modular, Architecture Control user group - (OMAC) defines a hard real-time system as a system that would fail if its timing requirements were not met, and a soft real-time system can tolerate significant variations in the delivery of operating system services like interrupts, timers, and scheduling.

Overall, a real-time computing correctness depends not only on the correctness of the logical result of the computation but also on the result delivery time. Real-time applications are characterized by timing constraints which are mostly non functional requirements comes from system specification. This applies to almost all the case studies in this thesis.

1.3.1.3 Distributed Systems

Distributed system or distributed computing deals with hardware and software systems containing more than one processing element (such as ECUs in automotive electronics) or storage element, concurrent processes, or multiple programs, running under a loosely or tightly controlled regime.

In distributed computing a program is split up into parts that run simultaneously on multiple computers communicating over a network. Parallel computing is most commonly used to

describe program parts running simultaneously on multiple processors in the same computer. Both types of processing require dividing a program into parts that can run simultaneously, but distributed programs often must deal with heterogeneous environments, network links of varying latencies, and unpredictable failures in the network or the computers. Recent development of automotive electronics has the trend from single ECU application to networked ECU application, for example, automatic windows function in a car could be deployed across as many as four ECUs so that the control signal to the windows won't go through long distance transmission.

1.3.1.4 Modern Automotive System

As mentioned above, the complexity and physical distribution of modern active safety, chassis and powertrain automotive applications requires the use of distributed architectures. Complex functions designed as networks of function blocks exchanging signal information are deployed onto the physical hardware (HW) and implemented in a software (SW) architecture consisting of a set of tasks and messages. ECUs are special purpose computer systems used performing computation of tasks, recent development of safety critical applications addresses stringent timing requirements on the system.

Overall, a complex, safety critical automotive system is an embedded, real-time, distributed system. The thesis will emphasis the design space exploration of such a complicated system.

1.3.2 Automotive Standards

Most applications in automotive electronic designs are getting more and more communication intensive. Design reuse is also being highly considered by major auto-makers. A number

of standards are currently addressing the need for reuse of automotive components at different levels of abstraction. The following is a list, far from exhaustive, of the main initiatives. This thesis focuses on two communication protocols in automotive electronic system design: the time- triggered (mainly for the static segment) FlexRay communication system and the event-triggered CAN communication system.

1.3.2.1 AUTOSAR

AUTOSAR [1] has been established by original equipment manufacturers (OEM) and Tier 1 automotive suppliers to develop an open industry standard for automotive electronic architecture which will serve as a basic infrastructure for the management of functions within both future applications and standard software modules. Members include GM, BMW, Bosch, Continental, DaimlerChrysler, Volkswagen, Siemens VDO, Ford, PSA, and Toyota.

The AUTOSAR basic infrastructure objectives are:

1. Implementation and standardization of basic system functions as an industry wide ”Standard Core” solution
2. Scalability to different vehicle and platform variants
3. Transferability of functions throughout network
4. Integration of functional modules from multiple suppliers
5. Consideration of availability and safety requirements
6. Redundancy activation
7. Maintainability throughout the whole ”Product Life Cycle”

8. Increased use of ”Commercial off the shelf hardware”

9. Software updates and upgrades over vehicle lifetime

The AUTOSAR software component implementation is independent from the infrastructure. A fundamental design concept of AUTOSAR is the separation between application and infrastructure. The approach is quite similar to central ideas of the platform based design methodology used throughout the thesis, namely the orthogonalization of functionality with architecture.

1.3.2.2 FlexRay

FLEXRAY [24] is a communication standard created by an industry consortium founded in 2000 by BMW, DaimlerChrysler, General Motors, Motorola, Philips, Volkswagen, and Robert Bosch. FlexRay is a high-speed serial communication system for in-vehicle networks using bus or point-to-point (star topology) links, at 10Mbps over un-shielded Twisted Pair or Shielded Twisted Pair cable. More introduction will be discussed in 3.1.1.1 for FlexRay system.

In FlexRay, the bus time is divided in cycles. Each cycle is partitioned into a static time-triggered portion, and a dynamic event-triggered portion. The communication cycle length is defined by the application up to a maximum of 16ms. The division between the two portions is set at design time and loaded into the controllers and bus guardians. Flexray's intended use is for highly dependable and fast safety critical applications. Protection from timing faults should be provided by the bus guardian that prevents frames sent in the time-triggered segment to overlap in time with each other and with the dynamic segment. The full schedule for the time-triggered portion is not known by each controller. Instead, this segment is divided at compile time into a number of slots of fixed size, and each controller and its bus guardians are informed of which slots are allocated to their

transmissions. Nodes requiring greater bandwidth are assigned more slots than those that require less. Each controller learns the full schedule only when the bus starts up. Each node includes its identity in the messages that it sends. During startup, nodes use these identifiers to label their input buffers as the schedule reveals itself.

The Static Segment is typically used for critical messages with the following constraints:

1. All static slots are the same length in microticks
2. All static slots are repeated in order every communication cycle
3. All static slot times are assigned for use in a cycle whether they are actually used or not
4. The number of the static slots is configurable but only up to a maximum of 1023 slots

1.3.2.3 CAN

Controller Area Network (CAN) is a multicast shared serial bus standard, originally developed in the 1980s by Robert Bosch GmbH, to connect electronic control units (ECUs). CAN was specifically designed to be robust in electromagnetic noisy environments. Bit rates up to 1 Mbit/s are possible with networks long less than 40 m. Decreasing the bit rate allows longer network distances (e.g. 125 kbit/s at 500 m).

The CAN data link layer protocol is standardized in ISO 11898-1 (2003). The standard describes mainly the data link layer - composed of the Logical Link Control (LLC) and the Media Access Control (MAC) sub-layers - and some aspects of the physical layer of the ISO/OSI Reference Model. All the other protocol layers are left to the network designer's choice. CAN is an attractive solution for embedded control systems because of its low cost, light protocol management and the deterministic resolution of the contention. The protocol adopts a collision detection

and resolution scheme, where the message to be transmitted is chosen according to its identifier. The lowest identifier message is transmitted. This makes possible to encode the message priority into the identifier field and to implement priority-based real-time scheduling of periodic and a-periodic messages. However, its major limitation is in the maximum available bandwidth, which is a consequence of the adoption of a multi-master digital bus. The available bandwidth may easily become scarce given the current trend in the automotive field towards an increased number of interconnected devices, particularly intelligent sensors, and an increased amount of data to be shared. In order to overcome the speed limitation of the CAN bus, other standards are being proposed, like the Flexray. However, because of its low cost and widespread use, the CAN bus is still the dominant communication medium in the automotive industry.

1.3.2.4 OSEK

OSEK [52] stands for Offene Systeme und deren Schnittstellen fr die Elektronik in Kraftfahrzeugen (eng., "Open Systems and their interfaces for the Electronics in Motor vehicles"). The OSEK specifications have been produced for an embedded operating system, a communications stack, and a network management protocol for automotive embedded systems. The most important characteristics of an OSEK-compliant operating system are

1. The flexibility on the kernel configuration with increasing service capabilities and memory requirements

2. The possibility of configuring the kernel by means of a file in a standard configuration language (OIL)

3. The static generation of most operating system and application objects, with a very efficient

implementation and minimum kernel footprint

4. The support for real-time scheduling and resource management algorithms, allowing a predictable timing behavior

OSEK is widely used in automotive electronics today. Understanding the operation system's capability is essential for designers.

1.3.2.5 LIN

Most sensor systems are based on architectures using point to point analog communications from sensors to ECUs. In these cases, signal integrity is at risk in very noisy environments such as that of an automobile. LIN [51] was originally developed for body electronics, however, it has been increasingly used to tackle the integrity issue for sensor interfaces. Its low-cost, bidirectional, single-wire physical-layer implementation reduces wiring and wiring harness requirements. It is possible to build a sensor module that has only three wires (battery, ground and LIN), even if there are multiple sensors inside the module. This reduction in wiring and wiring harness allows for a reduction in the sensor housing size, better sensor placement and less sensitivity to wire placement. This is especially true if more than one sensor is included in the module and all the outputs are multiplexed into a single LIN bus. LIN allows for two-way communication over a single wire so that the master has the capability to request diagnostic information from the sensor, or the sensor can provide system-failure information when needed. The LIN protocol is based on a master/slave architecture - all bus communication is controlled and scheduled by the master node. This feature enables guaranteed latencies for signal transmission, as no arbitration is utilized, which is a necessary property for most sensor signals. The LIN bus architecture is scalable to 16 nodes.

1.4 A Real Life Example

A real-life case study presented in this part will be used in the following chapters for evaluating the research approaches.

Auto-makers are constantly evaluating and developing new features that bring value to the customer. One important recent example is the integration of active and passive safety features, based on a set of ranging sensors that cover a 360 degree area around the vehicle. Examples are the Virtual Bumper and Anti-theft Systems. Some of these features have been demonstrated on a prototype research vehicle, called V1 in this thesis. The electrical architecture of V1 was developed integrating required sensors, actuators, CAN networks and processing units (ECUs) supporting the execution of the features in a prototype vehicle called mule. The architecture was developed by extension from the mule vehicle in an ad-hoc fashion, without considering extensibility or compatibility with current and future production architectures.

The following version V2 program consisted of the development of a production vehicle where some of the features from V1 and the experience gained in V1 were integrated and extended with new safety-relevant features, some of which coming from Tier-1 suppliers.

In V2, the control engineers provided the deadlines for the end-to-end latencies of selected paths in the architecture. Furthermore, the utilization of the ECUs and the CAN buses of the system had to be controlled to ensure future extensibility.

For this thesis, one example modified from a specific version of V2, called Experimental Vehicle (EV), will be used for evaluation purpose. Figure 1.1 shows a part of the entire functionality graph and Figure 1.2 shows a part of the entire architecture graph of the experimented EV.

In Figure 1.1, all the left most rectangles represent sensors, all the right most rectangles

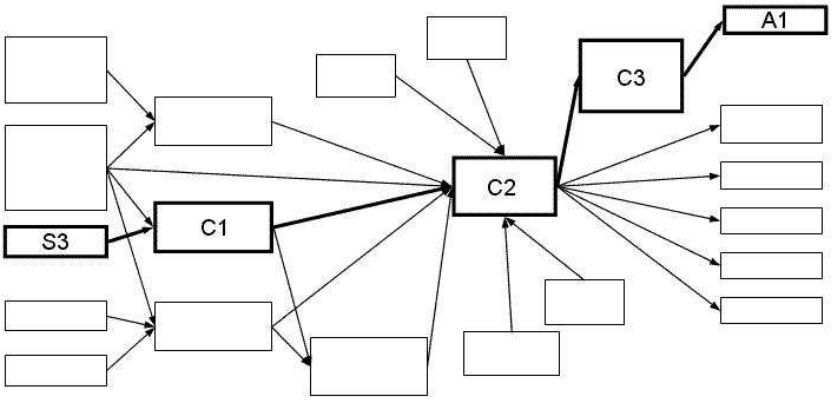


Figure 1.1: A Real-life Example: Part of Functionality for EVC.

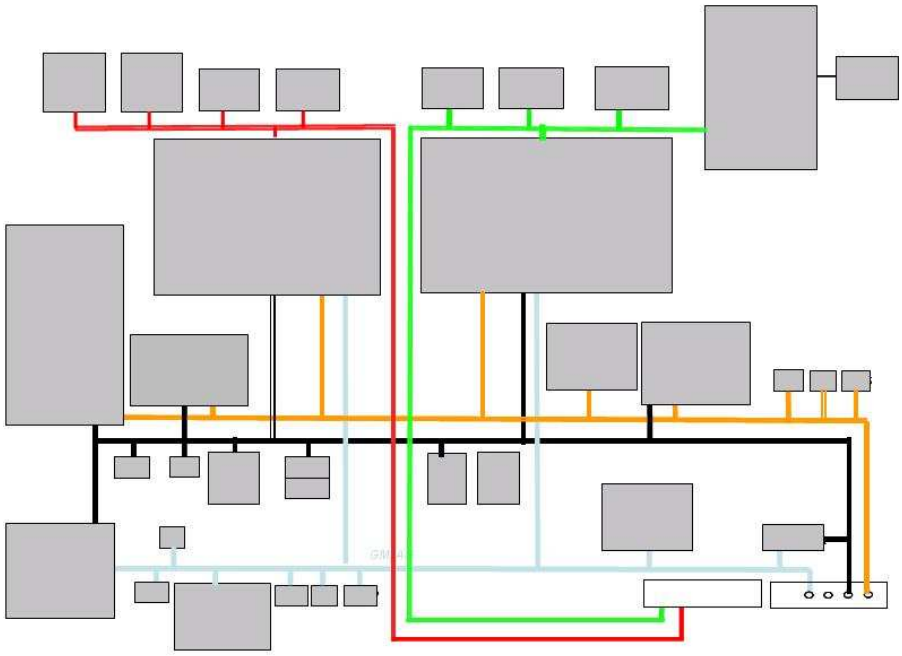


Figure 1.2: A Real-life Example: Part of Architecture for EVC.

represent actuators, and all the remaining ones denote the network of computing blocks decomposed from high level features. There is a timing constraint, required by the feature that the system is expected to provide, associated with one specific path on the functionality graph. This path may start from a sensor module S3 through computing modules C1, C2, C3 and finally reached an actuating module A1. A potential scenario is: sensor S3 runs in a non-stop mode to sense the change of the environment around the vehicle, the signal captured by sensor S3 goes through a series of intermediate computations and then the computed decision is propagated to the actuator A1 which acts on the environment. Consider a safety critical feature: lane keeping. In this example, the series of computation finally makes a decision whether the vehicle should keep the current lane or not, then the functional computation and signal propagation are expected to complete at most in a very short time, say around 100ms. Such an application is typical safety critical and has a high requirement on the path (maybe selected) latencies.

The above network of functionalities will be executed in the physical architecture which can be described in Figure 1.2. It consists of a number of ECUs connected by different types of communication buses, say either FlexRay (for safety critical messages) bus or CAN bus.

Mapping of functionality to architecture as well as scheduling of a set of functionalities to the same architectural unit is a crucial part of the design process. The process, called design space exploration, is the focus of this thesis.

Chapter 2 presents the design methodology combined with a design flow for the automotive domain where scheduling plays a fundamental role. Chapter 3 explains the scheduling synthesis problem in a FlexRay based communication system and then presents a scheduling strategy in an incremental design scenario to reduce design cost. CAN based communication system is

discussed in Chapter 4. The trade-off for the periodic activation model and the data-driven model is extensively explored using both a mathematical programming and a heuristic based approach. The effectiveness of the proposed approaches is illustrated by the EV case study. At the end of Chapter 4, the issues of integrated period synthesis is addressed. Chapter 5 investigates mapping policies together with schedulability analysis. The typical configuration features priority based scheduling of tasks and messages and imposes end-to-end deadlines. The proposed approach optimizes task placement and signal-to-message mapping, and automates the assignment of priorities to tasks and messages in order to meet end-to-end deadline constraints and minimize latencies. The effectiveness of the solution is proven again by the EVC example. Chapter 6 is the final chapter of the thesis and presents conclusion and future work.

Chapter 2

A Design Methodology and Flow with Emphasis on Scheduling for Automotive Systems

The definition of a car electronic/software system starts from the specification of the set of features, a very high level description of a system capability, that the system is expected to provide.

Functional models are created from the decomposition of the features in a hierarchical network of component blocks. The physical architecture model captures the topology of the car network, including the communication buses, the CPUs and the management policies that control the shared resources.

The design evolution led to the development of a system design methodology, Platform Based Design (PBD) that was proven successful in several automotive industrial problems. Section 2.1 presents the PBD methodology, and then Section 2.2 discusses a design flow under the PBD.

In Section 2.2, functional models, architectural models and system models are introduced first, and then a full discussion over mapping and scheduling is conducted.

2.1 Platform Based Design Methodology Introduction

According to the principles of PBD, system-level architecture design is neither a top-down nor a bottom-up design methodology. Rather, it is a meet-in-the-middle approach [35]. In a pure top-down design process, the specification of the application functionality is the starting point for the design process. The sequence of design decisions drives the designer toward a solution that minimizes the cost of the architecture. The design process selects the most attractive solution as defined by a cost function. In a bottom-up approach, a given execution architecture is designed to support a set of different applications and is, in general, developed based on designers intuition and marketing inputs.

The two main concerns of functional and architectural specification are connected by mapping a high level specification, defining an abstract model of the system, to a particular software and hardware architecture or platform. This match between function and architecture is a key aspect of the design of embedded systems and the founding principle of many design methodologies such as the platform-based design [4] and tools such as the former VCC/Sysdesign Sysdesign [54] product by Cadence, the Ptolemy and Metropolis frameworks [5].

The use of the conceptual framework of the platform-based design methodology and the meet-in-the-middle approach are advocated as key enablers for the exploration of design alternatives and architecture level solutions. Platform-based design requires the identification of clear abstraction layers and a design interface that allows for the separation of concerns between the refine-

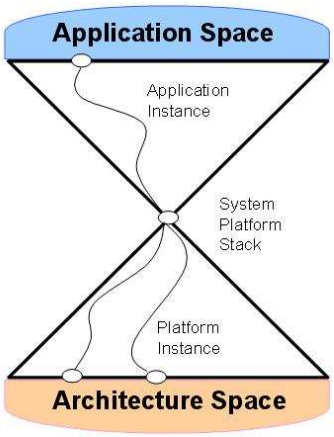


Figure 2.1: Platform Based Design.

ment of the functional architecture specification and the abstractions of possible implementations. Thus in automotive electronic design the application-layer software components are decoupled from changes in micro-controller hardware, ECU hardware, I/O devices, sensors, actuators, communication links, and from the partitioning of the software to the computing nodes. What specific layers should be used depends largely on the application space. For example, in typical automotive electronic applications, it may be sufficient to identify the architecture that one wants to target, then, the source code may be recompiled on various hardware architectures.

The basic idea of the PBD is captured in Figure 2.1 . The vertex of the two cones represents the combination of the functional model and the architecture platform. System designers map their applications into the abstract representation that includes a family of architectures that can be chosen to optimize cost, efficiency, energy consumption, reliability and flexibility (timing in this thesis). Decoupling the application-layer logic from dependencies on infrastructure-layer hardware or software enables the application-layer software components to be reused without changes across

multiple vehicle programs over a period of years.

Design space exploration consists of seeking the optimal mapping of the system plat22 form model into the candidate execution platform instances, the mapping must be driven by a set of methods and tools providing an objective and quantitative measure of the fitness of the architecture solutions with respect to a set of constraints and metric functions. The bigger is the gap between two neighboring abstraction levels, the larger is the potential for design optimization and exploration and the greater is the effort that is required to explore the mapping options. Ideally, there could be the possibility for automatic selection of the platform by appropriate software tools. In reality, the technology is not mature for a full synthesis of the mapping and the platform attributes and the approach that is currently viable is a what-if analysis where different options are selected as representatives of the principal platform options and evaluated according to measurable metric functions. Complex automotive systems are also implemented by integrating components provided (implemented) by suppliers. Today, the integration of a subsystem provided by a supplier relies on a black-box specification of the interface messages, including the priorities and possibly the execution rates.

2.2 A Design Flow with Emphasis on Scheduling for Automotive Systems

Section 2.1 introduced the concept on design space exploration which is a process of seeking an ”optimal” mapping from functionalities to architectures. The mapping is driven by a set of optimization metric functions to evaluate the fitness of the architecture solutions. Through this thesis, the optimization metrics chosen to evaluate the mapping result are timing, in another

word, different timing metrics are used as a guidance to the whole design flow when marrying functionalities to architectures.

Giving the mapping of software modules to hardware components, the design flow turns to focus on the scheduling of functionalities within the chosen architectures.Separating the ideas of mapping and scheduling is a critical assumption as for the cost function that decides the software mapping is orthogonal from the cost function that determines the task and bus message schedule by many auto-designers. [81] describes a methodology that optimizes this mapping by minimizing bus bandwidth and interconnect cost, hence validating this assumption. However, in an idealistic scenario these concerns cannot be separated in order to achieve a global optimal solution and must be accounted for through a multidimensional optimization problem. Nevertheless, in practice the industrial experience is that managing complexity is a key requirement and the approach of orthogonalizing the concern of software to hardware mapping and scheduling is a realistic assumption within the automotive domain. In this thesis, the texts focus on the scheduling part of the whole design flow.

Aside from the orthogonalization of mapping and scheduling, orthogonalizing functions and architectures[36] is also expected by the designers. In such a system development process, a functional description is defined first, then it is mapped onto some set of virtual architectural components. Generally, the above mapping obtains two or three potential solutions. The metrics related to timing such as end-to-end latency, among other performance analysis metrics, are used to obtain an optimized schedule or post-mapping system parameters, and select a potential solution (see figure 2.2).

Functional model, architecture model, system platform model, and the mapping, schedul-

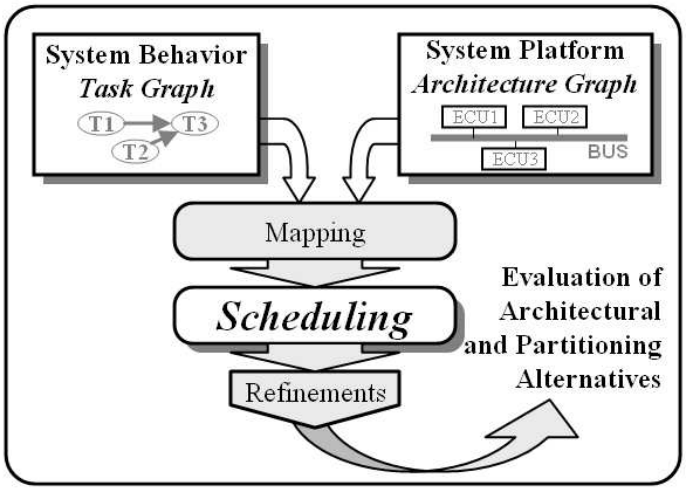


Figure 2.2: System design flow.

ing design phases are introduced one by one in the following sub-sections.

2.2.1 Functional Models

The starting point for the definition of a car electronic system is the specification of the set of features that the system is expected to provide. A feature is a very high level description of a system capability. The brake pedal force emulator in a brake-by-wire system is an example of a feature, consisting of mechanical, electronic and software parts that together emulate the feel of the conventional hydraulic brake pedal.

The software components of each feature are further developed by control engineers who devise control algorithms fulfilling the design goals. Typically, these algorithms consist of complex mathematical operations that are captured by a hierarchical set of block diagrams produced with tools such as Matlab Simulink.

The functional models are created from the decomposition of the feature in a network

of component blocks by abstracting the information that is relevant for the purpose of architecture exploration. The resulting model of the functionality is a hierarchical network of components encapsulating the system behaviors, with provided and required interfaces expressed by a set of ports. In the definition of the vehicle functionality, there is a need to work with a system view that abstracts from the details of the functional behavior and models only the interface and the communication semantics.

To give an example, some of the definitions that apply to the entities of the functional models are: 1. The activation mode of the functional blocks, synchronized on some clock or activated by the arrival event of a signal on one of the incoming ports. 2. The possible precedence constraints in the execution of the functional blocks. Whenever specified, the precedence constraints can be of type AND (all input signals must arrive before a block can execute) or of type OR (the block execution is triggered by the arrival of any of its input signals) 3. The semantics of the signal variables can be of type sticky with overwriting, meaning that the variable preserves the last value that has been written into it and the value of the signal variable is overwritten by a new output, or a signal link can represent a queue of tokens that are produced by the sender block and consumed by the receiver block. An example of the implications of a choice of an activation/communication model, the data communication between any two blocks activated periodically according to local, non synchronized clocks, is assumed to be nondeterministic in time and lossy, meaning that output values may be overwritten before having been read.

Timing constraints are expressed in the context of the functional graph by adding end-to-end deadlines to the computation paths, maximum jitter requirements to any signal and time correlation constraints between any signal pair originating from the same functional block or providing

input (possibly after further processing) to a common block, that is, having a common ancestor or successor block. A more detailed description of the fundamental concepts of end-to-end latency on a given computation path, and the corresponding deadline is however required for understanding the analysis procedures and the trade-offs in the design.

A path in the functional model is a set of functional blocks. Informally, the end-to-end latency is the largest possible time interval that is required for the change of the environment to be captured as input by the first block and propagated to the end of the chain, whatever is the state of the blocks in the path and regardless of the fact that some intermediate result may be overwritten before it is read. To understand the significance of this definition, please think of a feature like advanced collision preparation (ACP) where the environment variable models the existence of a target object on a collision route and the ending value which is the braking torque that is applied on the brake actuators.

Overall, the functional model considered in this thesis is usually represented by a directed acyclic graph (DAG) which has a network of components/tasks communicating with each other by exchanging messages.

2.2.2 Architecture Models

The model of the architecture is hierarchical and captures the logical topology of the car network, including the communication busses such as CAN and FlexRay links, the number of processors for each ECU and the resource management policies that control the allocation of each ECU and bus, but also the physical and geometric relationships, including abstractions for modeling wiring harnesses and connectors. At this stage, the hardware and software resources that are available for the execution of the application tasks and the resource allocation and scheduling

policies must also be specified. Each RTOS provides a set of services and logical resources and has a set of parameters related to the provided scheduling policy for the ECUs.

The physical model also defines entities of the physical architecture that determine its cost, including connectors and harnesses and their connectivity, ECU and ECU multiplexed buses with the associated controllers and ASICs memory, and the pin-out information for each ECU.

2.2.3 System Platform Model

If specification of functionality aims at producing a logically correct representation of system behavior, the system platform model is where physical concurrency and resource requirements are expressed. At this level, rather than applying the term process or thread - although widely used in the literature - to describe a unit of computation processed concurrently in response to environment stimuli or prompted by an internal clock, the term task will be used in adherence with OSEK. Tasks cooperate by exchanging messages and synchronization or activation signals and contend for use of the execution resources (the processors) as well as for the other resources in the system. The system platform model entities must, on one hand, be the implementation of the functional model entities and are, on the other hand, mapped onto the target hardware.

The system platform models are a synthesis of the mapping process and can be of different types for different analysis purposes, hiding unnecessary details and exporting only the necessary amount of information. Throughout the thesis, two models are investigated:

The first model is simple. In this model, all tasks and messages are activated periodically and communicate according to a semantics where the communication channel holds the last value that is written into it and it is implemented as a shared variable protected against concurrent access. This model, called periodic activation model, has many advantages, including the separation

of concerns when evaluating the schedulability of the individual resources and also allows for a very simple specification at the interface of each subsystem or component, thereby simplifying the interaction with the suppliers. The drawback is a non-deterministic time behavior and a possibly very large worst-case end-to-end delay in all computations that are performed according to this model.

The next model includes the possibility of activating messages and tasks based on some event, typically the completion of a task or the change in the value of a signal. The corresponding models, are called on-demand activation and on-event activation. In this case, the completion of a task always results in the immediate enqueuing (or activation) of all the messages that send information produced by the task and the arrival of a message results in the activation of the receiver task.

This model corresponds to an implementation in which, for example, the middleware software layers enqueue a new message whenever a task completes its execution and produced a value for one of its outgoing signals (not necessarily different from the value produced in its previous execution instance) and where the arrival of a message results in the execution of an interrupt handler activating the software task that processes the message data. In the terminology in use among operating system developers and programmers, this model is commonly defined as interrupt driven.

Finally, the most general platform models allow for an arbitrary combination of the previous message and task activation semantics, subject to a few coherency rules. Chapter 4 discusses this in a detailed manner.

2.2.4 Mapping

The mapping of the blocks defined in the functional model into system platform entities and the subsequent placement of those onto the physical architecture objects are the critical design

activities. In practice, the blocks defined in the functional part must be executed in the context of one or more system tasks with their attributes. The mapping of the threads and message model into the corresponding architecture model and the selection of resource management policies allows the subsequent validation of the mapped model against functional and non-functional constraints.

The mapping phase consists of allocating each functional block to a software task and each communication signal variable to a virtual communication object. The task activation rates must be entered as parameters of the architectural models and compliance checks are performed with the functional blocks activation rates. If more than one functional block is mapped to a task, the order of the execution must be provided during the mapping phase.

As a result of the mapping of the platform model into the execution architecture, the entities in the functional models are put in relation with timing execution information derived by worst (best) case execution time analysis (not provided directly by the environment) or back-annotations extracted from physical or virtual implementation. In the latter case, several execution timing information are maintained such as best, worst, mean, and distribution.

Given the deployment, it is possible to determine which signals are local (because the source and destination functions are deployed onto the same ECU) and which are remote, hence need to be packed into messages and go over the network. Each communication signal variable is therefore mapped to a communication resource of the implementation, that is, a message, or a tasks private variable or a protected shared variable. Each message, in turn, is mapped to a serial data link, and the mapping relation can be extended by mapping serial data links to harnesses, and harnesses to physical locations in the car.

When mapping a functional model into a platform model, the semantics of the functional

description should be preserved. Therefore, not all the mappings are allowed or should be made legal. For example, a non-deterministic communication among two functional blocks can be made deterministic, and a global execution order for all the functional blocks can be defined, after mapping them into the task set, in accordance with the partial order defined by the semantics of the functional model. However, a mapping of a communication signal with an attached precedence relation to a communication variable shared among two periodic asynchronously activated tasks should not be possible.

2.2.5 Scheduling

When the mapping is done, tasks executing on a computing resource like a ECU with a necessarily finite execution time, additional definitions must be provided, and the same to the messages transmitted on the bus. Each instance of a periodic functional tasks or a message, the release time, start time and finish time associated with each of them are defined. Also the worst case response time for each scheduling entity is needed to calculate clearly.

To obtain a better (less pessimistic) estimate of the timing and reliability properties, the modeling allows the capturing of different configurations in terms of functional, architectural, mapping and timing information of the system: these different configurations are captured as system modes. This analysis only focus on identifying the system modes that result in significant differences in the timing behavior of the system. For example, some engine control functions may be executed at a rate that depends on the engine rotation speed and at high rates, the definition of the function can change (to a simplified implementation) in order to ease the schedulability of the system.

Classification of real time system scheduling could be based on:

1. Input

2. Criticality of timing constraints

3. The nature of the real time load

Scheduling based on input could be either time-driven or event-driven. If the input is continuous (synchronous), it is time-driven while if the input is discontinuous (asynchronous), then it is event-driven.

Scheduling based on criticality of timing constrains could be hard real time scheduling if the response of the system within the timing constraints is crucial for correct behavior or could be soft real time scheduling if the response of the system within the timing constraints increases the value of the system.

Scheduling based on the nature of the real time load could be static scheduling when the load is predefined, constant and deterministic or could be dynamic when the load is variable (non-deterministic).

A real world system exhibits a combination of these characteristics, for instance, the example in Section 1.4 could be an event-driven, hard real time and static scheduling case.

The scheduling also must be driven by a set of methods and tools providing an objective and quantitative measure of the fitness of the architecture solutions with respect to a set of constraints and metric functions. End to end latencies on selected paths provide us a good metric for evaluating a scheduling result.

Modern automotive architectures support the execution of distributed safety- and time-critical, or at least time-sensitive functions on complex networked systems with several buses and tens of ECUs. Schedulability theory provides support for the analysis of the worst case latencies

in distributed computations when the architecture of the system is known and the communication and synchronization mechanisms have been defined. In the design of complex automotive systems, however, a great benefit of schedulability analysis may come from its use as an aid in the exploration of the software architecture configurations that can best support the target application.

The thesis presents several optimization frameworks to select the communication and synchronization model that leverages the trade-offs between the purely periodic and the precedence constrained data-driven activation models in the scheduling phase of the design flow. The problem is amenable to an ILP formulation and to solution based on a standard solver. Formalization of the problem is provided at different levels of approximation and shows what is the result of the optimization on a case study consisting of a complex automotive architecture. The complexity in terms of running time is estimated by evaluating derivatives of the case study with different levels of concurrency and resource utilization.

In addition to the introduction and review for distributed embedded system, this chapter presented the design methodology (Platform Based Design methodology) the thesis based on and discussed the research focus, the scheduling part, within the design flow in automotive electronic system design.

Chapter 3

Scheduling for Time-Triggered Communication System

In automotive systems, computation and communication functions can be time- or event-triggered. In the first case, task activations and message transmissions are bound to happen at predefined points in time. Considering that the development of a modern automotive system is most of time communication intensive, this chapter is dedicated to talking about scheduling synthesis for time-triggered automotive communication system. The structure of this chapter is as follows: all preliminaries and definitions related to **Time-Triggered Communication System** are presented in Section 3.1, followed by Section 3.2 where scheduling synthesis for a FlexRay based communication system in an optimization framework is described, finally scheduling synthesis in an incremental design framework for a general time-triggered system is discussed in Section 3.3.

3.1 Preliminaries and Definitions for Time-Triggered Communication System

Time-triggered communication system is supported by protocols that schedule the messages statically based on tables that define the time points when messages need to be transmitted. This section gives an overview about time-triggered communication protocols with a focus on FlexRay, Section 3.1.2 then provides previous related work especially on FlexRay. System modeling and description are introduced in Section 3.1.3.

3.1.1 Time-Triggered Protocols Overview

Supporting for time-triggered communication is provided by protocols that schedule the messages statically based on local scheduling tables that define the messages transmission time. SAFEbus [31], SPIDER [61], TTCAN [3], and the Time-Triggered Protocol (TTP) [40] are examples. TTP uses a generalized time-division multiple-access (GTDMA) scheme with variable sized slots, in which each node has only one opportunity to transmit for each cycle. In FlexRay, slots have the same size, but a node can have more than one transmission opportunity for each cycle. Scheduling techniques for the static segment have been developed by extending the work for scheduling messages in a TDMA bus [59] [29]. In [21], the authors consider the case of a hard real-time application implemented on a system with a FlexRay bus. Messages are scheduled in the static segment, and the method in practice reuses scheduling techniques developed for the TDMA bus.

3.1.1.1 FlexRay Introduction

The development of new by-wire functions with stringent requirements for determinism and short latencies, and the upcoming active safety functions, characterized by large volumes of data traffic, generated by 360^0 sensors positioned around the vehicles, are among the motivations for the definition of the FlexRay standard. FlexRay is being developed by a consortium (www.flexray.org) that includes a few core members, namely, BMW, Daimler-Benz, General Motors, Freescale, NXP, Bosch and Volkswagen/Audi, as a new communication standard for highly deterministic and high speed communication. The stated objective is to support cost-effective deployment of distributed by-wire controls.

At the core of the FlexRay system is the FlexRay communications protocol. The protocol provides flexibility and determinism by combining a scalable static and dynamic message transmission, incorporating the advantages of familiar synchronous and asynchronous protocols. The communication channel is a broadcast channel. A message sent to the channel by a ECU is received by all the other ECUs. Application messages are usually scheduled through the static segment of FlexRay bus in order to ensure the timing constraints are satisfied.

In FlexRay, the communication speed is defined at 10 Mb/s, although determination of the minimum bit-time that is necessary to ensure correct detection of the bit value in spite of delay jitter might suggest additional versions at lower speeds. The bus bandwidth is assigned according to a time-triggered pattern. The available bandwidth is divided into communication cycles and each communication cycle contains up to four segments (Static, Dynamic, Symbol and Network idle time - Nit, as in Figure 3.1). Figure 3.2 shows more details of the timing hierarchy of a FlexRay system. Clock synchronization for the purpose of communication is embedded in the standard

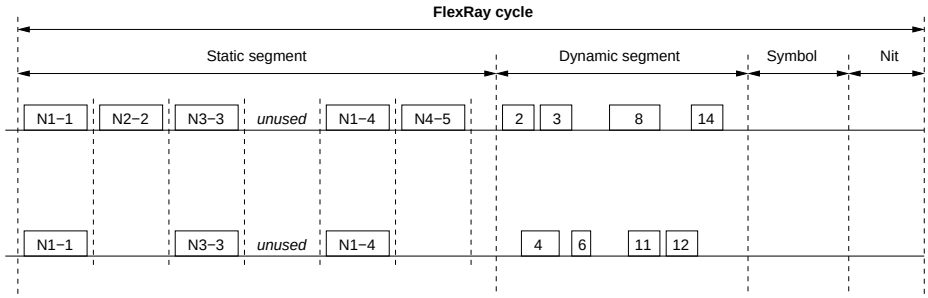


Figure 3.1: the four segments in a FlexRay cycle

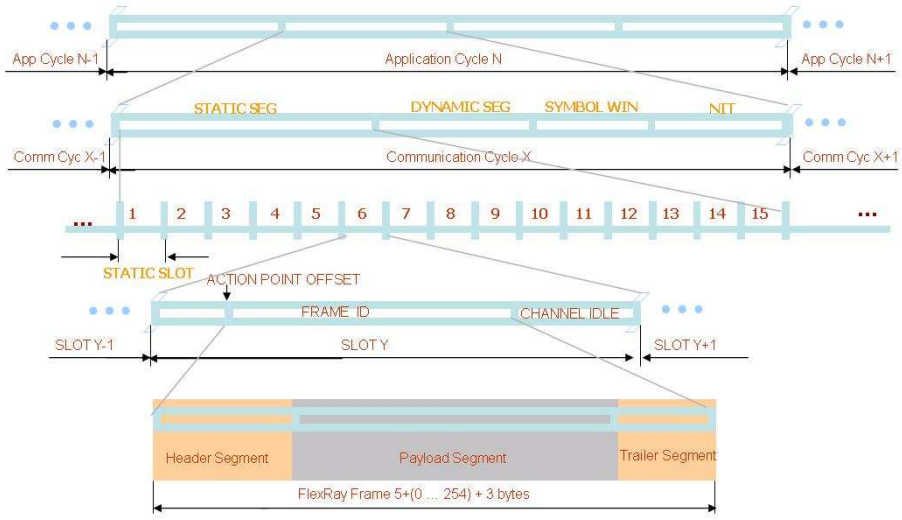


Figure 3.2: FlexRay Timing Hierarchy.

using part of the Nit segment, and therefore comes at no additional cost. This ensures deterministic communications at least from the theoretical standpoint.

The static part of the communication cycle enables the transmission of time critical messages according to a periodic cycle, in which a time slot, of fixed length and in a given position in the cycle, is always reserved to the same node. The dynamic segment allows for flexible communications. Transmission of messages in the dynamic part is arbitrated by identifier priority (the lowest identifier messages are transmitted first, somewhat like in CAN but in a slightly different

way). FlexRay includes a dual channel bus specification (for increased reliability) and will include (in its upcoming specification) bus guardians at the node and star level for increased reliability and timing protection. In a dual channel configuration, messages can be replicated on both channels for safety critical communications that leverage the physical redundancy, or the slots can be assigned independently. In the latter case, the communication bandwidth is doubled.

The time-triggered model of FlexRay, not only allows for time determinism, but is also considered as a paradigm for composability and extensibility. Each node only needs to know the time slots for its outgoing and incoming communications. The specification of these time slots is kept in local scheduling tables. No global description exists and each node executes with respect to its own (synchronized) clock. As long as the local tables are kept consistent, no timing conflicts or interferences arise.

Slots that are left free in the (virtual) global table resulting from the composition of the local tables can be used for future extensions. Time protection and isolation from timing faults are guaranteed by the reservation of time slots and guardians that avoid that node transmit outside the allocated time window as stated in the original specification.

The bus cycle time and the transmission slot time are design parameters that should be carefully selected. Fundamental issues related to the composition of subsystems, but also to future extensibility and reuse of components require careful planning and possibly standardization of the approach.

Clock synchronization and time determinism on the communication channel allow the implementation of end-to-end computations in which the data generation, data consumption and communication processes are temporally aligned, avoiding sampling delays, and therefore removing

the worst drawback of composable periodic activation semantics, that is lateness, in exchange of determinism.

Also, system-level time-triggered schedules allow the semantics-preserving implementation of distributed control models (including models with a synchronous reactive semantics, like those produced by popular commercial tools like Simulink from Mathworks [44]).

Time determinism requires that the time-triggered model of communication is propagated to the computation layers, using a time-triggered scheduler and a careful coordination of the communication and computation schedules, so that the schedule becomes global. If the schedulers are not coordinated, then not only guaranteeing time determinism is more difficult and probably altogether impossible, but the performance of the system in terms of latency is significantly worse. The synchronization of the communication and RTOS layers is only scantly addressed by current standards.

3.1.1.2 Time-Triggered Protocol (TTP) Introduction

According to [22] TTP is a real-time protocol of the Time-Triggered Architecture (TTA). The protocol uses a Time-Division Multiple Access (TDMA) scheme to enable collision-free bus allocation. TTP focuses on the interconnection of components in order to form a highly dependable realtime system that is sufficient for critical applications such as X-by-wire in the automotive and avionics domains. TTP implements a replicated bus system and a guardian that prevents babbling idiot failures.

TTP aims at an easy and economically integration of sensors and actuators into a network. TTP can be implemented on low-cost micro-controllers, which suggests each transducer having a TTP interface. The interface concept of TTP supports a modular design and an easy integration and

management of transducers.

3.1.1.3 A Comparison for Time-triggered Protocol

Three of the most viable automotive protocols: TTCAN, Flexray and TTP are analyzed and compared in this paragraph (see Table 3.1 for the summary of the analysis). FlexRay is chosen for this chapter's communication protocol as it clearly supports the requirements for a scalable and extensible scheduling scheme which is discussed in Section 3.3.1 but also provides the necessary bandwidth and fault tolerance mechanism.

Table 3.1: Protocol comparison

Requirements [1]	TTP	TTCAN	Flexray
Multiple transmissions[2]	No	Yes	Yes
Global sync. time[3]	Yes	Yes	Yes
Bandwidth (Mbbs)	825	1	10
Identity Transparency[4]	No	Yes	Yes
Variable slot length	Yes	Yes	No[5]

[1]: Scalable and extensible scheduling requirements on the TDMA protocol

[2]: Supports multiple transmissions from the same node within a single round of communication

[3]: Provides a global synchronized time

[4]: The identity of the sender node is transparent to the receiver node (for nodes that don't participate in startup)

[5]: No variable sized slot length, but can be achieved if a number of slots are concatenated logically

3.1.2 Previous Work on Time-Triggered Protocols

There are several communication protocols for time-triggered based network.Evaluation of the worst case response time is possible, and timing analysis techniques have been provided for CAN [77], TDMA [76] and TTP [59]. These analysis techniques are however subject to conditions

that are seldom verified in practice [41]. Furthermore, in event-driven and priority arbitrated buses, message response times are still subject to a possibly large jitter. Another major problem of these protocols is that they offer little or no protection against timing faults caused by nodes flooding the network with high priority messages. In conclusion, time predictability can only be achieved to a limited degree and none of them is suitable for safety-critical applications [32].

In order to accommodate a fraction of traffic that is dynamically activated, flexibility can be added with an additional transmission window reserved to this type of traffic. This is the case of hybrid protocols like Byteflight [9], introduced by BMW for automotive applications and later superseded by the FlexRay, and of the FTT-CAN protocol [23]. The dynamic segment of the FlexRay protocol is similar to Byteflight and uses a priority-based arbitration for outgoing messages based on a virtual token concept. Cena and Valenzano [17] discuss schedulability of the Byteflight protocol, which is similar to the dynamic segment of FlexRay. In order to allow time guarantees, the authors assume a quasi-periodic transmission scheme for time-critical messages, which means that the dynamic segment scheduling does not differ much from TDMA.

In [59], the authors presented an approach to timing analysis of applications communicating over a FlexRay bus, which considered the specific aspects of this protocol, including the dynamic segment. Techniques were proposed for determining the timing properties of messages transmitted in the static and the dynamic segments of a FlexRay communication cycle. The authors first presented a static cyclic scheduling technique for TT messages transmitted in the ST segment, which extended the previous work on the TTP [60]. Then, they developed a worst-case response time analysis for event-based transmissions in the dynamic segment. Message analysis techniques were integrated in a holistic schedulability analysis algorithm that computes the worst-case response

times of all the tasks and messages in the system.

In [47], an end-to-end model based development process for building a complex FlexRay based distributed control system is described if the context of safety critical x-by-wire systems for a realistic automotive application. However, the authors relied on manual scheduling for the scheduling of software and communication tasks on the bus because of lack of mature scheduling tools targeted for FlexRay based applications.

A FlexRay scheduling algorithm is given in [33]. After decoupling the ECU scheduling and FlexRay bus scheduling using time slicing technique to assign transmission time windows for messages on the bus so that the precedence relations between different processors are enforced by those time windows assignment, a uni-processor scheduling technique is applied to the messages on the bus. An extensible scheduling concept is introduced in [33] which is achieved through minimizing the peak processor utilization based on mathematical programming approach. Integer variables may limit the problem scale that the approach can handle. For FlexRay configuration, a fixed slot size is assumed while this is a very important bus parameter subjects to optimize in this thesis.

3.1.3 System Modeling

Both functionalities and architectures could be described by a model that would be used by the synthesis framework in the thesis. Application model is set of task graphs, a very simple model of computation which could capture the corresponding timing information of the design. System architecture is FlexRay based system, several ECUs are connected to the FlexRay bus through FlexRay controllers.

An automotive electronics architecture generally is a multi-cluster system. In this work,

only the FlexRay cluster scheduling design is presented. Activation of tasks on ECUs and of messages on FlexRay are purely based on progression of time.

3.1.3.1 System Functionality

In this section, a model of the system computations as a *dataflow* is considered. The vertices represent the *tasks* and the edges represent the data signals communicated among tasks.

A task τ_i is characterized by (e_i, T_i, Φ_i, C_i), where e_i is the ECU resource it needs to execute, T_i its period, Φ_i its initial phase, C_i its execution time.

The edges $L = \{l_1, l_2..., l_m\}$ represent the input/output connections between tasks. A directed edge $l_{i,j}$ between tasks τ_i and τ_j will carry a data signal with a given bit width produced by τ_i and available to τ_j.

Each periodic task reads its input at its activation time and writes its results at the end of its execution. A task pair τ_i, τ_j may communicate by exchanging a signal information σ_{ij} characterized by a bit width b_{ij}. Each signal may optionally be delivered with a unit delay, modeled with a binary variable δ_{ij} associated to it (Figure 3.3).

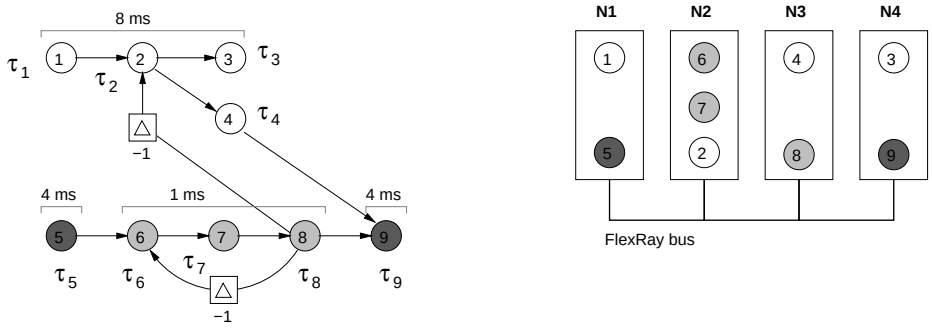


Figure 3.3: System model with tasks, links and delays.

Each task will run an infinite sequence of instances or *job*s. The *Application cycle* or

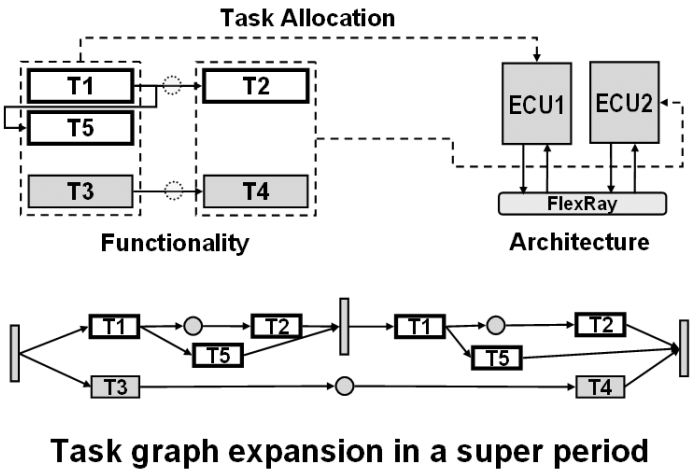


Figure 3.4: Mapping and task graph expansion.

hyperperiod or *superperiod* H is defined as the least common multiple of the periods of all tasks. Inside the hyperperiod, each job instance is considered as an individual scheduling entity and denoted as t_i. The scheduling problem consists of planning the execution of jobs and the transmission of signals into the available slots inside H. Job instances can also be denoted with reference to their task. In this case, t_{kj} denotes the j-th job of task τ_k. In another word, each task has a periodicity. The period specifies how often a task should be executed. From the above definition, a super period is the least common multiple of all task periods. All tasks in a task graph are scheduled within one super period. Time is expressed relative to the beginning of the super period.

The arrival time of a job instance t_i is denoted as a_i or, using the instance index notation as a_{kj}, with $a_{kj} = \Phi_k + (j-1) \times T_k$. The release time of a job is A_i, the time at which it is scheduled for execution is s_i and its finishing time f_i. The response time $\mathcal{R}_i$ of a job t_i is defined as the time interval from its arrival to its termination, i.e. $\mathcal{R}_i = f_i - a_i$. The worst case task response time $\mathcal{R}_{\tau_i}$ is defined as the maximum of the response times $\mathcal{R}_{ij}$ of its jobs t_{ij}.

The set of all the task instances transmitted in the application cycle defines the *Application*

instance graph, as in Figure 3.5. Signals transmitted by tasks allocated to the same node may be transmitted in the data content of a message m_i in a communication slot.

Similar to tasks, *slot instances* are also considered. The j-th slot inside cycle k is denoted as λ_{kj}. Similar to jobs, its start time is s^s_{kj}, and its finishing time f^s_{kj}.

A *functional chain* or *path* from τ_i to τ_j, or $P_{i,j}$, is an ordered sequence $P = [\tau_i, \ldots, \tau_j]$ of $n + 1$ (n=j-i) tasks such that there is a link between any two consecutive tasks. For example, in Figure 3.3 a path exists between the tasks τ_1 and τ_9. Latencies are defined for paths by extending the meaning of a task latency. The latency of a path instance $P_{i,j,k}$ is defined as $\mathcal{L}_{i,j,k} = f_{kj} - a_{ki}$

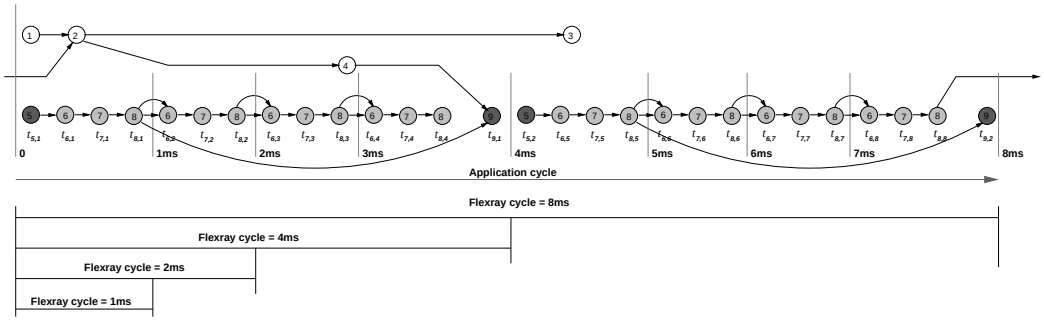


Figure 3.5: Instance graph, Application cycle and FlexRay cycle.

3.1.3.2 System Architecture

A distributed architecture comprises of a set of possibly heterogeneous nodes interconnected with one or more broadcast based bus with a time-triggered semantic. Every node would be comprised of the basic architecture as indicated in Figure 3.6.

With regards to the software architecture, the same semantics of the time-triggered operating system standard, used within the automotive industry, are implemented. This standard can be referred in [27]. The main highlight of this OS is the fact that preemption is done of the stack rather

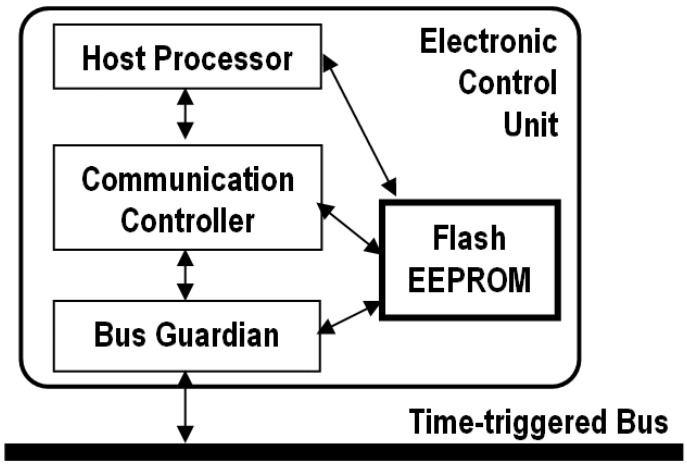


Figure 3.6: Time-triggered node architecture.

than through a dynamic scheduler. Since the designer has release time control over the execution of the tasks, any worst-case preemption scenario that could occur in a static priority system can be modeled using this semantic.

The architecture is represented by an architecture graph, which contains services that could implement processes and communications specified in the task graph. Specifically, it represents processing elements (PE) that implement processes, and bus structures that implement inter-PE communications. In the automotive domain, these PEs are electronic control units (ECUs). The architecture structures are assumed to be any number of ECUs connected to one shared bus.

The bus is implemented with a time-division multiple-access (TDMA) scheme. Each ECU is allocated a predetermined time segment on the bus, called a TDMA slot. A sequence of slots that can be repeated periodically is called a TDMA round. Assuming a slot can not have arbitrary sizes in the FlexRay scheduler which corresponds to the communication protocol smoothly and then relaxing this assumption for a slot can take arbitrary size, and a node is powered to control multiple slots in a round. From these assumptions, a TDMA round that corresponds to a super period can

exploit optimization exposed by task graph expansion in a multi-rate system. These abstractions are consistent with the capabilities of the FlexRay standard.

Every ECU has a bus controller (Figure 3.6) that contains the schedule of when messages (relevant to itself) should be sent or received. The schedule of a message consists of the start time and finish time of the message. An ECU can only send messages within its own TDMA slot, which starts from the message start time, and ends at the start time of the next message on the bus (or the end of the super period, if it is the last message to be transmitted in a super period). The TDMA slot allocation is statically scheduled and programmed into each ECU's bus controller.

The mapping is a refinement process where the elements in the task graph are bound to the services in the architecture graph. In the limited scope of this chapter, the mapping is the process of task allocation onto ECUs.

3.1.3.3 FlexRay Message Passing Mechanism Example

Figure 3.7 demonstrates the message passing mechanism for a functionality graph on a FlexRay bus.

Task τ_1 sends a signal σ_{12} to task τ_2 who is located on the same ECU, and the signal σ_{12} becomes a local variable and won't show up on the FlexRay bus, so σ_{12} goes through the middle-ware and is handled by the OSEK Kernel to task τ_2. Task τ_1 sends another signal σ_{13} to a remote task τ_3 who lives on another ECU, at this time σ_{13} first goes through middle-ware and is sent to the out-going buffer on the FlexRay controller who waits for slot owned by the controller/ECU and puts the σ_{13} on the specific slot number within a communication cycle. The FlexRay controller on the receiving side knows exactly when to pull messages from the FlexRay bus, and then translates the message back to signal σ_{13}, directs the signal σ_{13} to its destination task τ_3 through the OSEK

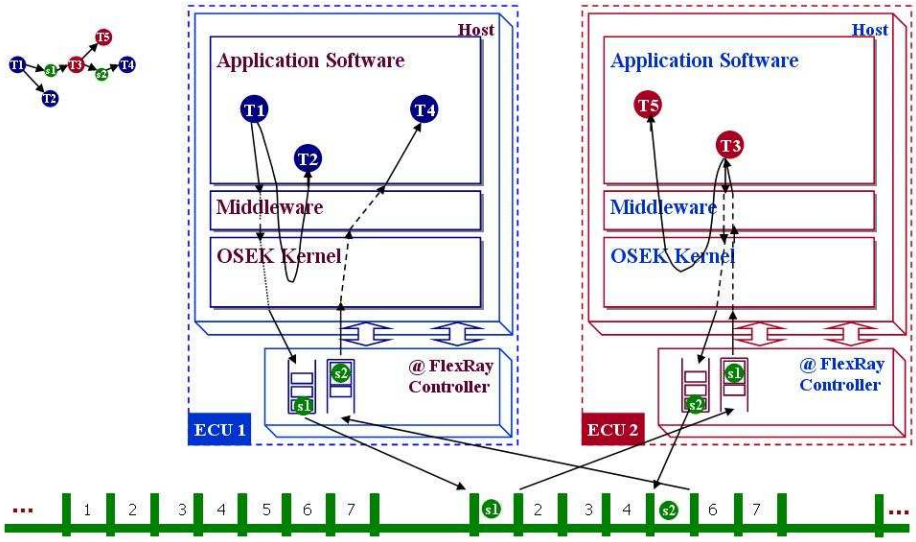


Figure 3.7: FlexRay Message Passing Mechanism.

kernel. This message passing mechanisms apply to all the tasks communicated with each other. Understanding this in detail, a FlexRay scheduler synthesis can be introduced in the following section.

3.2 Scheduling Synthesis for a FlexRay based communication network

FlexRay based communication system provides support for the transmission of time-critical periodic messages in a static segment and priority-based scheduling of event-triggered messages in a dynamic segment, for this thesis, only the static segment is considered. The design of a FlexRay schedule is not an easy task because of protocol constraints and demand for extensibility and flexibility. Studying the problem of FlexRay bus scheduling from the perspective of the application designer, a focus is given to optimizing the performance of application related (end-to-end) timing metrics. The solution is based on an MILP optimization framework and allows optimization

subject to a number of possible design metrics.

3.2.1 A Two-Steps Approach and Synchronization Mode

The method allows to optimize the scheduling configuration with respect to a number of metric functions. According to the system design approach, mapping from functionality to architecture is separated from scheduling the mapped functionality on different architectural components is generally consistent with the typical design flows in use by the auto industry.

3.2.1.1 A Two-Step Approach

The scheduling of FlexRay communication consists of the mapping of the tasks and signals defined in the application cycle into a set of communication cycle instances (Figure 3.8). This mapping can be performed in different ways, according to the selection of the communication cycle length, of the size of the static segment, of the slot size and correspondingly of the number of static slots for each communication cycle.

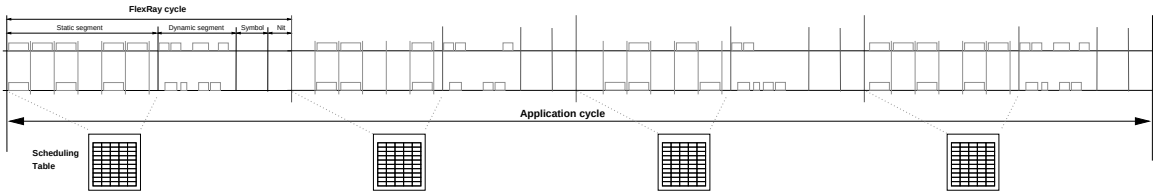


Figure 3.8: Application cycle and FlexRay cycle

It is practically impossible to encode all the above into an integrated problem formulation to be solved by an optimization framework. The resulting problem formulation would very likely suffer from issues related to the size of the search space. Hence, in this work, a two step approach (Figure 3.9) is investigated to avoid the above computational limitation. Starting from

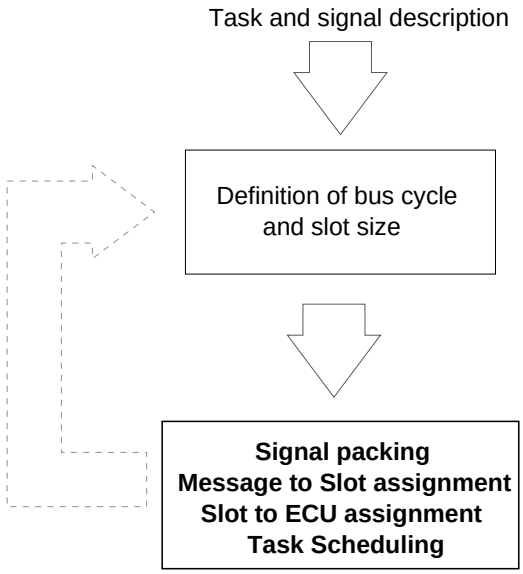


Figure 3.9: A two-step approach in scheduling FlexRay communication

the design specification, first a heuristics is designed to select a FlexRay bus configuration $< l_{app}, l_{comm}, n_{slot}, l_{slot} >$. Then, based on this configuration out of the first step, a mathematical programming framework is applied to encode the problem and synthesize other variables such as slot ownership, signal to slot mapping, message and task scheduling.

The two step approach is generally consistent with the typical design flows in use by the auto industry. The application cycle is clearly based on the application on hand, and FlexRay designers define the communication cycle and the slot size based on past experience, but especially on the need to reuse legacy components, which is likely to induce carmakers to a future standardization of these parameters, at least for their product lines.

Another option is to evaluate several possible FlexRay configurations in an initial branching of the search procedure. If the number of possible configuration is not very large, it should be possible to explore them and run the optimization framework as an inner loop, comparing the results

at the end and choosing the best one with respect to the objective function.

3.2.1.2 Synchronization modes

There are two possible synchronization patterns between tasks and messages.

- **Asynchronous scheduling** This model does not require that the job and message schedulers ares synchronized. Jobs post data values for the output signals in shared variables. The communication drivers have the responsibility, at the beginning of each cycle or before each slot, to fill the registers for the outgoing communication slot and, at the receiving side, the data are written into output registers and asynchronously read by the reader tasks.

- **Synchronized scheduling** In this case, job executions and message transmissions are synchronized in such a way that a job must complete before the beginning of the slot that transmits its output signal (with a margin determined by the necessary copy time). When schedulers are synchronized, it is possible to know what job produces the data that is transmitted by a message and the jobs that reads the data delivered by the message. Scheduling can be arranged to achieve very tight end-to-end latencies and small jitter between the best and the worst case response times.

Figure 3.10 shows two examples of scheduling without task and message start and finish time synchronization and with synchronized instances respectively. When schedulers are synchronized sampling delays can be controlled and worst case latencies can be reduced.

Also, within each synchronization pattern, there are two possible scheduling models for tasks in a FlexRay environment. In the first model tasks are scheduled according to the OSEKTime framework. In OSEKTime tasks are executed according to a time table, which defines their start

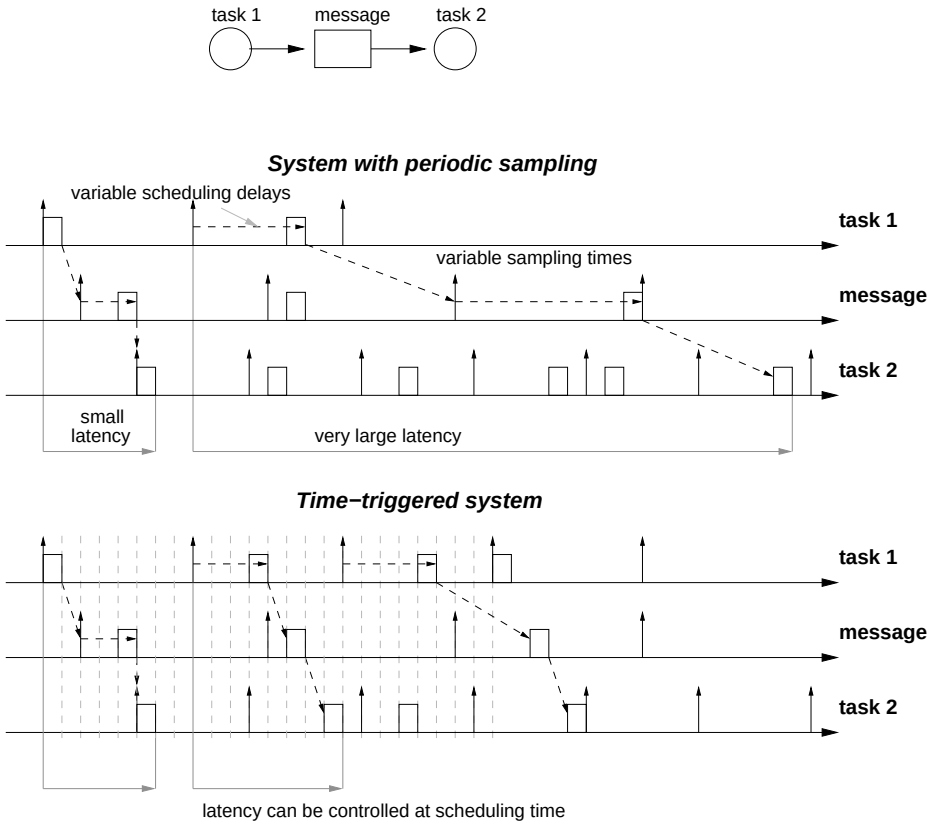


Figure 3.10: Schedulers synchronized and not synchronized

times, and can optionally preempt each other. The other option is to schedule tasks in an OSEK framework in which tasks are periodically activated, each task has a fixed priority and the scheduler is preemptive with the option of preventing preemption inside selected task groups. In this chapter, a problem formalization and an MILP solution for all the possible options are provided.

3.2.2 Synthesis FlexRay Scheduler in An Optimization Framework

For this FlexRay Scheduler, two options are explored, based on the two existing standards for RTOS and the consequent task scheduling. In one, the time-triggered OSEKTime standard is used. The other option is based on the fully-preemptive, priority-based scheduling of the OSEK standard.

The main focus of this work is the development of a methodology, based on an MILP formulation of the FlexRay scheduling problem that can

- accomodate legacy components with pre-existing task and message schedules,
- deal with end-to-end deadline constraints on the computations as well as a general type of timing constraint, including jitter constraints and output coherency constraints and
- accomodate different task scheduling options.

The method in this section allows to optimize the scheduling configuration with respect to a number of metric functions.

3.2.2.1 Mixed Integer Programming Solution

The approach is to formulate the problem in the general framework of mathematical programming (MP), where the system is represented with parameters, decision variables, and con-

straints over the parameters and decision variables. An objective function, defined over the same set of variables, characterizes the optimal solution. This problem allows a mixed integer linear programming (MILP) formulation that is amenable to automatic processing. After [15], a MILP program in standard form is:

$$\text{minimize} \quad c^T x \tag{3.1}$$

$$\text{subject to} \quad Ax = b \tag{3.2}$$

$$x \geq 0 \tag{3.3}$$

where $x = (x_1, ..., x_n)$ is a vector of positive real or integer-valued decision variables. A is an $m \times n$ full-rank constant matrix, with $m < n$, b and c are constant vectors with dimension n*1. Constraints of the type $Ax \leq b$ can be handled by adding a suitable set of variables, and then transforming such inequalities in the standard form. MILPs can be solved very efficiently by a variety of solvers. The CPLEX solver is used in this work and most of the similar approaches throughout the thesis.

3.2.2.2 Problem formulation

Mixed integer programming formulation is used to find a solution to the FlexRay scheduling problem with respect to a cost function that accounts for latencies on paths. A short summary of the notations used in the ILP formulation is provided in the following

Activation, release and deadline constraints Φ_i, a_i, s_i, f_i,d_i denote the initial phase, arrival time, start time, finish time and deadline for periodic task τ_i respectively. The periods T_i and the

deadlines d_i are the given parameters as problem inputs, while Φ_i, and consequently a_i, s_i (for OSEKTime scheduling only) and f_i are variables for the formulation framework. Given that all tasks are periodic with an initial phase, their activation time must be constrained accordingly

$$a_{i,k} - a_{i,k-1} = T_i \tag{3.4}$$

$$a_{i,o} = \Phi_i \tag{3.5}$$

$$0 \leq \Phi_i \leq T_i \tag{3.6}$$

A hard real time system requires all tasks to finish before their deadlines.

$$f_i \leq d_i \tag{3.7}$$

OSEKTime scheduling

In OSEKTime, jobs are scheduled according to the following model. Jobs are executed according to a time table. The job start times s_i must be constrained to be larger than the corresponding activation times.

$$a_i \leq s_i \tag{3.8}$$

OSEK scheduling

In OSEK, tasks are activated periodically, by an internal dispatcher or by an alarm and scheduled according to their priorities. The response time of the scheduler or the dispatcher task, introduces jitter (constant) in the activation time.

$$0 \leq a_{i,k} - A_{i,k} \leq J_i$$

Start times and preemption A binary variable is used to define the order of execution of tasks.

$$y_{i,j} = \begin{cases} 0 & \text{if start time of } \tau_i \text{ precedes start time of } \tau_j, \\ 1 & \text{otherwise.} \end{cases}$$

and the values of the y variables need to be kept consistent with the starting times according to the definition. M is a large constant used to facilitate the mutually exclusive constraints formulation.

$$s_i < s_j + y_{i,j} \times M \tag{3.9}$$

$$s_j < s_i + (1 - y_{i,j}) \times M \tag{3.10}$$

Similarly, a binary variable is used to encode preemption

$$p_{i,j} = \begin{cases} 0 & \text{if task } \tau_i \text{ is not preempted by task } \tau_j, \\ 1 & \text{otherwise.} \end{cases}$$

clearly, mutual preemption is not allowed.

$$p_{i,j} + p_{j,i} \leq 1$$

If τ_i starts later than τ_j, τ_j doesn't need to preempt τ_i. This leads to an additional constraint between y and p

$$p_{i,j} \leq 1 - y_{i,j}$$

Finally, the next inequality pair encodes additional properties of preemption with respect to starting and finishing times. If task τ_i starts before task τ_j and it is not preempted by it, then its finishing time should less than or equal to the starting time of τ_j. However, if τ_i is preempted by τ_j, then it needs to finish after the execution of τ_j, as defined by the second constraint.

$$f_i \leq s_j + y_{i,j} \times M + p_{i,j} \times M \tag{3.11}$$

$$f_j < f_i + y_{i,j} \times M + (1 - p_{i,j}) \times M \tag{3.12}$$

Feasibility Constraints The feasibility constraints are modeled according to the rules for computing the starting, finishing times and deadlines of all the jobs and signals (messages) scheduled or transmitted on the bus.

OSEKTime scheduling

In OSEKTime, tasks are scheduled according to the following model. Tasks are activated according to a time table. After activation, a task can be preempted by other tasks running on the same resource. Define θ as a set which contains all the task pairs (τ_i, τ_j) where τ_i and τ_j are tasks mapped to the same ECU but without any data dependency between them. The first equation calculates the response time of tasks,

$$f_i = s_i + C_i + \sum_{(i,j)\in\theta} p_{i,j} \times C_j \tag{3.13}$$

OSEK model

Approximation can be used again here for the tasks response time according to 4.2.1. But this requires the priority associated with the tasks. In this section, r_i is used to denote the worst case response time of task τ_i, and $hp(i)$ denotes a set which has all scheduling objects with a higher priority than τ_i.

$$r_i = C_i + \sum_{j \in hp(i)} \lceil \frac{r_i + J_j}{T_j} \rceil C_j \tag{3.14}$$

The above equation can be approximated by a linear combination with coefficient $\alpha \in [0, 1]$ of linear upper and lower bounds as follows

$$r_i = C_i + \sum_{j \in hp(i)} (\frac{r_i + J_j}{T_j} + \alpha) C_j \tag{3.15}$$

The following equation for the finishing time of tasks:

$$f_i \geq A_i + r_i \tag{3.16}$$

FlexRay protocol rules FlexRay has its own specific bus access scheme. Define l_{comm},l_{slot} as the length of communication cycle, and the slot size respectively. $s^s_{j,k}$ is used as an input parameter which denotes the starting time of the k^{th} slot from the j^{th} communication cycle. $s^s_{j,k}$ is easily calculated as $s^s_{j,k} = l_{comm} \times j + l_{slot} \times k$.

Signal to slot mapping The mapping of signals to slots is encoded in another set of binary variables

$$A_{i,j,k} = \begin{cases} 0 & \text{if signal } m_i \text{ is NOT mapped to com cycle j, slot k,} \\ 1 & \text{otherwise.} \end{cases}$$

When a signal m_i is mapped to a specific slot from a specific communication cycle, the start time and finish time for the signal will be automatically constrained to the time frame of the slot.

$$s^s_{j,k} \leq s_i + (1 - A_{i,j,k}) \times M \tag{3.17}$$

$$s_i \leq s^s_{j,k} + (1 - A_{i,j,k}) \times M \tag{3.18}$$

$$f_i \leq s^s_{j,k} + l_{slot} + (1 - A_{i,j,k}) \times M \tag{3.19}$$

$$s^s_{j,k} \leq f_i - l_{slot} + (1 - A_{i,j,k}) \times M \tag{3.20}$$

Finally, one signal could only be mapped to one slot and the sum of the transmission time over all mapped signals with one specific slot will be upper bounded by the slot size.

$$\sum_{j<n_{comm},k<n_{slot}} A_{i,j,k} = 1 \tag{3.21}$$

$$\sum_{i \in \phi} C_i \times A_{i,j,k} \leq l_{slot} \tag{3.22}$$

Slot ownership Each slot is owned by a CPU or it is free. A set of binary variable encodes the status of each slot

$$A_{e_i,j} = \begin{cases} 1 & \text{if slot j is owned by ECU } e_i, \\ 0 & \text{otherwise.} \end{cases}$$

FlexRay has its requirement for the slot ownership. If a slot is owned by one specific ECU, then the ownership applies to every communication cycle. The first constraint of following ones means if signal m_i is mapped to communication cycle j and slot number k, then m_i's source ECU must own slot k. The second one corresponds any slot couldn't be owned by more than one ECU. But if no signal is mapped to slot k in any communication cycle, then the last constraint will set the slot ownership to null.

$$A_{i,j,k} \leq A_{e_i,k} \tag{3.23}$$

$$\sum_{e_p \in \varepsilon} A_{e_p,k} \leq 1 \tag{3.24}$$

$$A_{e_p,k} \leq \sum_{i \in \phi, j < n_comm} A_{i,j,k} \tag{3.25}$$

Data dependencies If there is a data dependency between two jobs or between a job and a message, there is a need to guarantee the successors start later than the predecessors. For example, if a task τ_i sends a signal σ_j to some other tasks, then we need to make sure the sender tasks finishes its execution before the signal is scheduled for transmission on the bus. Following constraints are used to all these pairs, where γ_j represents the worst case copy time for the outputs data to be written into the input variable for the receiving task or the data transmit register for the appropriate slot in the FlexRay adapter. The following constraint is needed if τ_j depends on a signal from τ_i.

$$f_i \leq s_j - \gamma_j \tag{3.26}$$

3.2.2.3 Scheduling domain

If Φ_i is the initial phase of a generic task τ_i, the scheduling of the tasks and of the FlexRay bus must be performed until an entire application cycle of computations has been computed. This means, that the schedule must continue until time $H + \max_i(\Phi_i)$ (where H is the hyper-period). Since the initial phase values are computed as a result of the optimization, we will use an upper bound for the previous formula

$$t_{\max} = H + \max_i(T_i)$$

In the interval $[0, t_{max}]$ (see Figure 3.11) we need to schedule for each task τ_i a number of instances

$$nt_i = \left\lfloor \frac{t_{\max}}{T_i} \right\rfloor$$

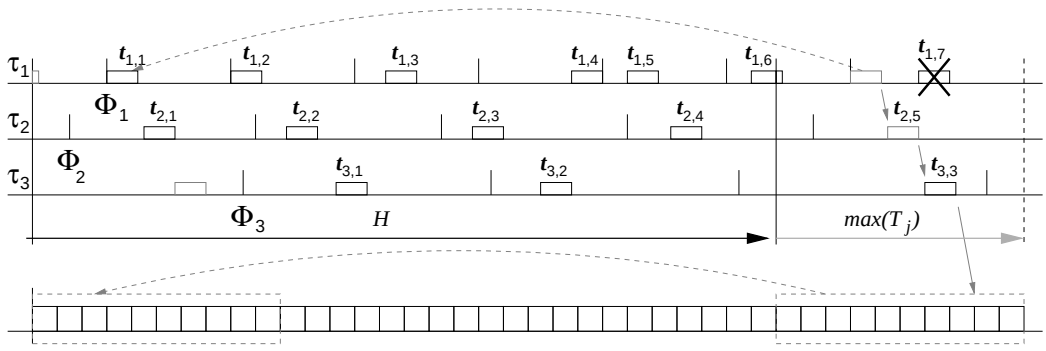


Figure 3.11: Extent and constraints in the definition of the scheduling domain

However, not all of those instances can be scheduled freely. In the example in the figure, this is true for $t_{3,3}$, but not for $t_{1,7}$, which must be scheduled in the same way as $t_{1,1}$ given that the two are actually the same instance in the scheduling tables. This translates into the constraint on the

finish times for both types of schedulers.

$$f_{i,q} = f_{i,k} + H \text{ where } q = nc_i + k$$

for a time-triggered (OsekTime) scheduler, it must also be

$$s_{i,q} = s_{i,k} + H \text{ where } q = nc_i + k.$$

where, for each task it is

$$nc_i = \frac{H}{T_i}$$

Similar constraints exist on the scheduling of the FlexRay slots. There is a need to schedule beyond the application cycle, up to an additional number of cycles.

$$n_a = \left\lceil \frac{max_i(T_i)}{l_{comm}} \right\rceil$$

in the last cycle, however, only

$$n_l = \left\lfloor \frac{max_i(T_i) - (n_a - 1)l_{comm}}{l_{slot}} \right\rfloor$$

slots need to be scheduled. Suppose that n_{af} is the number of communication cycles in an application cycle, then slot ownership must correspond. This is automatically guaranteed by constraints 3.23 to 3.25, provided that the index of the communication cycles j spans from 1 to $H/l_{comm} + n_a$ and for $j = H/l_{comm} + n_a$ the slot index goes from 1 to n_l.
Similarly, signal to slot mapping must correspond. The matching set of signals can be identified as follows. If the sender and receiver task instances of signals m_i and m_j are as follows.

$$(src(m_i) = t_{jk}) \wedge (dst(m_i) = t_{lm}) \wedge (src(m_j) = t_{jp}) \wedge (dst(m_j) = t_{lq})$$

where $p = k + nc_j$

and $q = m + nc_l$

Then the two signals are actually the same signal and must be allocated to the corresponding slots (with a distance of H). This can be denoted as $m_i = m_j$. This means that for all cycle and slot indexes k,l, for each pair $m_i = m_j$, it must be

$$A_{jml} = 1 \text{ if and only if } A_{ikl} = 1 \wedge m = k + \frac{H}{l_{comm}} \tag{3.27}$$

3.2.2.4 Objective Functions

Based on the above constraints, in addition to get a feasible solution, which satisfies the deadline constraints, we have the flexibility to get the optimal solution with respect to different cost functions. $\mathcal{EP}$ is the set of end objects on the selected paths.

$$minimize \sum_{o \in \mathcal{EP}} f_o \tag{3.28}$$

The above cost function would be meaning to minimize the end to end latencies over selected paths.

3.2.2.5 Case study: an automotive x-by-wire system

Table 3.2 and 3.3 describe a prototypical X-by-Wire application from General Motors. The application has 10 ECUs interconnected by a FlexRay bus. There are 47 tasks, with periods of

1ms, 4ms and 8ms, and 132 signals. The table shows the periods and worst case execution times in microseconds.

No end-to-end delay constraints are defined for this case, and the objective is to find the schedule with the minimum number of assigned slots. Two FlexRay configurations are tried. In both cases, $H = 8000\mu s$. For configuration 1, the FlexRay cycle $l_{\text{comm}} = 1000\mu s$ with $n_{\text{slot}} = 22$, and the slot size $l_{\text{slot}} = 200$ bits, or $35\mu s$. In configuration 2 it is $l_{\text{comm}} = H = 8000\mu s$. The slot size is unchanged and there are $n_{\text{slot}} = 222$ slots in the cycle. Also, three scheduling policies are tried: OSEK, and OSEKTime with and without preemption (all $p_{i,j} = 0$).

The problem is modeled in AMPL and solved using CPLEX with a time limit of one hour. If the problem formulation in Section 3.2.2.2 is applied "as is" to the case studies, the solver can find a feasible solution, but not the optimal one within a hour of run time (Method (1) in Table 3.4). For example, for the OSEK scheduling with bus configuration 1 (top left of Table 3.4), there are 32672 binary variables after the AMPL pre-solve phase. CPLEX finds a feasible solution with 13 assigned slots, but cannot guarantee optimality (with an optimality gap = 1.903%).

Additional knowledge on the problem can be used to further restrict the search space. First, leveraging the definition of a (tight) lower bound on the number of slots $k_i = \sum_k A_{e_i,k}$ that are needed by each ECU e_i, and then iteratively fixing the slot assignment for the ECUs that reached their lower bound on the number of used slots, and optimizing the signal to slot assignments for other ECUs (Method (2) in Table 3.4). A more aggressive reduction of the search space can be obtained at the expense of optimality by restricting the time window assigned for scheduling to tasks and messages (Method (3) in Table 3.4).

Basically, Section 3.2.1 and 3.2.2 provide solutions for different task-to-message syn-

Task name	**ECU**	**Period**	C_i
τ_8	e_9	8000	810
τ_9	e_9	8000	550
τ_{11}	e_9	8000	100
τ_{12}	e_9	8000	770
τ_{13}	e_9	8000	200
τ_{14}	e_9	8000	110
τ_{15}	e_9	8000	550
τ_{16}	e_{10}	8000	780
τ_{10}	e_{10}	8000	510
τ_{17}	e_{10}	8000	190
τ_{18}	e_{10}	8000	260
τ_{19}	e_{10}	8000	100
τ_{20}	e_{10}	8000	230
τ_{21}	e_5	1000	25
$\tau_{22}/\tau_{26}/\tau_{30}/\tau_{34}$	$e_5/e_6/e_7/e_8$	1000	60
$\tau_{25}/\tau_{29}/\tau_{33}$	$e_6/e_7/e_8$	1000	30
$\tau_{24}/\tau_{28}/\tau_{32}/\tau_{36}$	$e_5/e_6/e_7/e_8$	1000	20
$\tau_{23}/\tau_{27}/\tau_{31}/\tau_{35}$	$e_5/e_6/e_7/e_8$	1000	40
$\tau_{37}/\tau_{42}/\tau_{47}/\tau_{52}$	$e_1/e_2/e_3/e_4$	8000	1000
$\tau_{38}/\tau_{43}/\tau_{48}/\tau_{53}$	$e_1/e_2/e_3/e_4$	8000	500
$\tau_{39}/\tau_{44}/\tau_{49}/\tau_{54}$	$e_1/e_2/e_3/e_4$	8000	1500
$\tau_{40}/\tau_{45}/\tau_{50}/\tau_{55}$	$e_1/e_2/e_3/e_4$	4000	1300
$\tau_{41}/\tau_{46}/\tau_{51}/\tau_{56}$	$e_1/e_2/e_3/e_4$	8000	350

Table 3.2: Tasks for the X-by-wire example

chronization scenarios and different task scheduling policies, based on existing industry standards. The solution is based on the above MILP optimization framework and allows optimization with respect to a number of possible design metrics not limited to objective function 3.28.

3.3 Incremental Design for Time-Triggered System

Being able to accommodate incremental design changes while preserving a legacy design may reduce design and verification times substantially [55]. This aspect is of special interest in automotive electronics as system architecture is designed before the complete set of functionalities

is known. The ultimate objective is to design an architecture that is extensible and scalable, i.e., that can accommodate additional functionality with either no change to the design or with the addition of architectural models to support the incremental modification without changing the implementation of the legacy functionality. These concepts are quantified by the use of two metrics, extensibility and scalability. For this section, the thesis again focuses on the important aspect of the design on automotive systems, the scheduling problem for hard real time distributed embedded systems. Static priority preemption is considered provided release time control on the set of ECU's. The metrics are evaluated by placing them into a cost function within a mathematical programming framework. The cost of modifying a legacy system is characterized at an ECU and bus component level. Extensive case studies in the automotive domain are used to evaluate the metrics and the cost function. Results show that the optimization framework is effective in reducing development and re-verification efforts after incremental design changes.

Automotive manufactures such as General Motors (GM) have adopted a product line design approach in order to manage complexity and reduce cost. The central theme of this type of design paradigm is component reuse. The product line approach also follows an incremental design methodology, which entails incremental addition or modification to a legacy implementation versus a single new design for each generation of the product. This condition brings about very interesting implications to system design, as the change induced ripple-effect or the "coordinated change" across the entire system must be reduced. Given this, different aspects of the design can be optimized with respect to the cost of this "coordinate change". One such aspect studied in this section is with regards to the bus and task scheduling for a time-triggered distributed system.

Briefly summary from previous sections, the following input can be assumed:

1. A directed acyclic graph (DAG) denotes task graph and the temporal attributes of the functionality

2. The physical architecture of the system

3. The mapping of software to hardware of the functionality

As a note, this input can be characterized within two separate design generations:

1. Current implementation that needs to be scheduled

2. Future additions to the current implementation

Step 1 could either be a fresh new implementation or itself an addition to an existing legacy implementation. In a hard real-time embedded system, all computations must complete before their respective deadlines. Such stringent system property is essential in safety critical applications. Verification of these system properties is time and resource intensive. In 2001, Magneti-Marelli [64] reported that a power-train unit of only 50,000 lines of code took 30 months to develop, 5 of which was used to verify the system. The demand for faster time-to-market cycles requires an exploration of various approaches to relieve this verification bottleneck.

3.3.1 Extensibility and Scalability Design Metrics in a general Time-Triggered Protocol

A mathematical programming optimization framework is presented from Section 3.3.1 through Section 3.3.3 to illustrate more scenarios in incremental design consideration for a general time-triggered communication system.

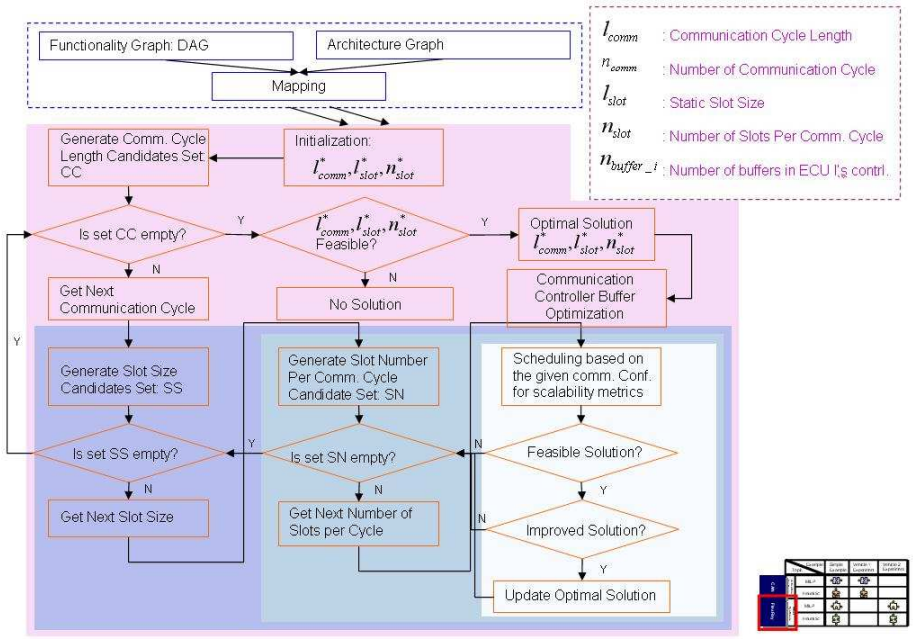


Figure 3.12: FlexRay Schedule Overall Optimization Flow.

3.3.1.1 Scheduling Implications in An Extensible and Scalable Scenario

Traditional scheduling result gives the order of the tasks to be executed, tasks are expected to execute consecutively as soon as all data and control dependencies are met. In a time-triggered schedule derived this way, small incremental changes often lead to a total rescheduling and re-verification. There is significant room to exploit the scheduling objective to help reduce necessary verification procedures in an incremental design scenarios as mentioned in previous sub-section.

In this section, a set of metrics are identified to capture the extensibility and scalability of a hard real-time embedded system, a method is developed to apply the metrics to a design, and the effectiveness of the metrics is evaluated. Specifically, I study a hard real-time embedded system in the automotive domain, and focus on the scheduling aspect of system design. Extensibility and scalability are characterized in the scheduling step of a design flow, and the set of metrics are applied

in a scheduling cost function using integer linear programming (ILP) and mixed integer quadratic program (MIQP). The analysis of scheduling results show the effectiveness of the set of metrics proposed for extensibility and scalability.

3.3.1.2 Extensible and Scalable Metrics

The motivation for selecting an extensible and scalable schedule is to allow flexibility in a schedule to tolerate incremental changes. An incremental change could be a change of worst case execution time (WCET) for a task, a change of worst case transmission time (WCTT) for a message, or the addition of new elements.

A schedule is a list of starting time for tasks and starting/finishing time for messages that satisfies all constraints of the system. If an incremental design change has not changed the schedule, it shows the schedule has tolerated the incremental change, and the original set of verified system properties should still hold.

An extensible schedule must tolerate changes in task WCET and message WCTT. The first priority is to maintain the bus schedule. The bus is a shared resource, shifting a message in rescheduling a bus schedule could lead to expensive re-verification operations. An extensible schedule should also reduce interference of schedule changes across ECUs. If the schedule of one ECU must change, the change should minimally affect other ECUs in the system.

A scalable schedule must accommodate new tasks and messages by statically scheduling them on an existing system with as little effect on the legacy tasks and message schedules as possible. This involves providing blocks of idle times on the ECUs for computationally intensive tasks, and providing porosity in the schedule to accommodate tasks with tight deadlines. Stack based preemption is allowed in executing tasks on the ECUs. This provides a new task all the idle

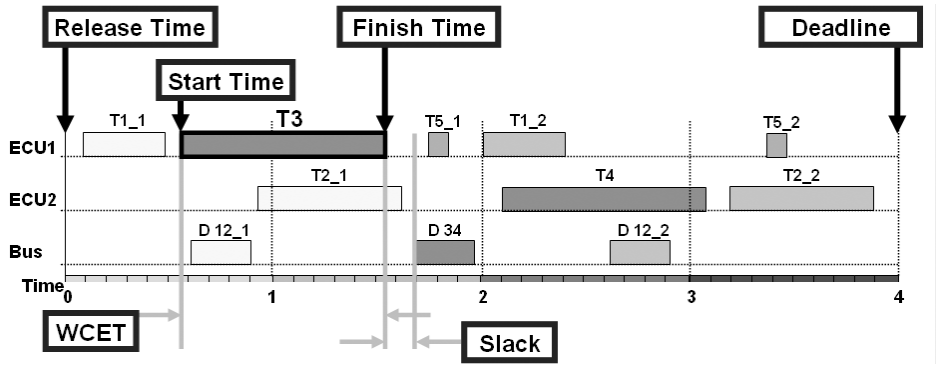


Figure 3.13: Properties for Task T3.

time on an ECU from the task release time to its deadline. The ability of accommodating a new computationally intensive task is limited only by the amount of idle time on an ECU.

It is common to assume the deadline of a task is the end of its period, subject to data dependency constraints. A task with tight deadline is a task that has a short period. This type of task is expanded into several iterations in a super period through task graph expansion, so there must be enough idle time in each of the period to accommodate it. This would require distributing idle times on an ECU, such that the new task with a tight deadline could find opportunities in each of its iterations to execute.

To accommodate new messages, similar to that of new tasks, distributing idle times on the bus is needed such that a new message would affect as few other messages as possible.

3.3.1.3 Previous Work on Extensibility and Scalability Metrics

There is a wide body of previous work in static cyclic scheduling. Classical scheduling theory typically uses metrics such as minimizing the sum of completion times, minimizing scheduling length, minimizing resources or minimizing the maximum lateness [34]. As the constraint of

deadlines is added for real time systems, the emphasis is shifted to finding a feasible solution while minimizing some metrics such as end-to-end delay, overall execution time, overall processor cost, or overall communication cost [7].

This section is concerned with the extensibility and scalability metric of schedules for hard real time embedded systems. The closest previous work is [55], where the concept of incremental design flow is first discussed. [55] uses an improved list scheduling approach to obtain a valid schedule, and then uses an algorithm to distribute task slacks on the processing elements to accommodate future tasks. However, because they only consider priority based preemption in their approach, and not our schedulability based preemption, the resulting schedule may not be suitable for accommodating future tasks with urgent deadlines. Also, mapping of software and hardware in [55] is assumed to be a degree of freedom for their extensibility and scalability considerations. Such assumptions may not be realistic, as significantly different cost functions, such as fault-tolerant cost functions, may be dominant at the mapping stage.

In the thesis, mathematical programming is chosen as a vehicle to describe and optimize for the extensibility and scalability metric. There is an extensive array of commercial solvers available, and we can obtain global optimal solution with respect to an objective function. Mathematical programming has been used in [7] for mapping and scheduling of homogeneous multi-processor real time systems. Their work is motivated by software/hardware co-design, and the objective is to obtain schedule feasibility while maximizing performance and minimizing cost.

The formulation in this part assumes statically scheduled tasks with data dependencies, in a distributed and heterogeneous multi-processor architecture. The bus is time triggered. Preemption is allowed in the ECUs, and multi-rate tasks are accommodated by using task graph expansion. All

tasks are allocated a-priori, and task migration is not permitted.

The amount of idle time on an ECU is determined by the amount of tasks allocated, the WCET of each of the tasks on the ECU, and the super period of the system. The super period of a task is least common multiple of all the periods of the tasks. The designer controls the amount of idle time on an ECU by defining a set of task allocation, and determining ECU performance (and therefore WCET of tasks on the ECU). The amount of idle time on the bus is determined by the total number of messages on the bus and the bus performance. The focuses on optimally utilizing these redundancies for maximal extensibility and scalability.

3.3.1.4 Mathematical Representation of Metrics

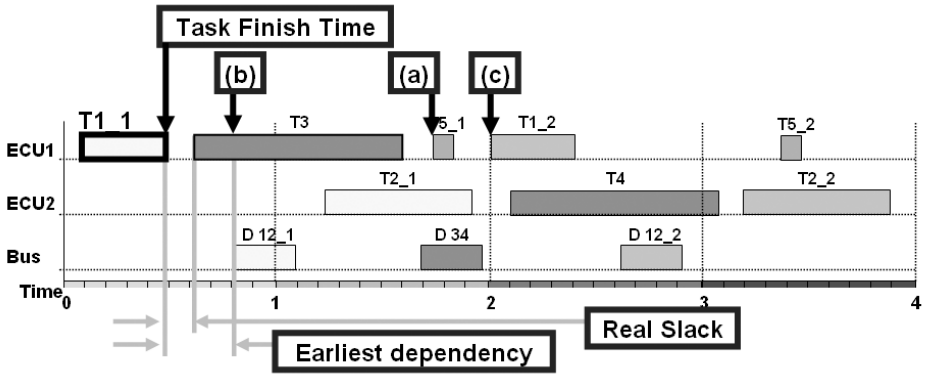
A metric is a standard of measurement used to compare results with repect to a property. The extensibility and scalability metrics provide an upper bound to the design changes a schedule can accommodate.

The extensibility metric for a task describes the maximum task WCET extension a schedule can accommodate without rescheduling.

The extensibility metric for a message describes the maximum task WCET extension a schedule can accommodate by only rescheduling the finish time of the message on the source ECU and the destination ECU.

The scalability metric is defined with respect to an architectural resource, i.e. an ECUs or a bus. It describes the maximum WCET(WCTT) a schedule can accommodate for a new independent task(message) with certain period.

Extensibility Metric for Tasks: As part of an incremental design change, a task t_i on ECU u_k may increases its WCET. As preemption is allowed, t_i may extend its finish time beyond



a) start time of next dependent task on u_k b) start time of next dependent message c) deadline of task t_i

Real slack for $T1_1$ is the idle time on ECU1 between $T1_1$ finish time and time (b)

Figure 3.14: Extensibility metric illustration.

the start time of the next task t_j on u_k without changing the schedule on t_i. Execution of t_i can resume after t_j finishes.

Numerically, the slack for task t_i is sum of all idle time on u_k between finish time of t_i and ι_i, where ι_i is the earliest of the three times below: (see Figure 3.14)

a) start time of next dependent task on u_k

b) start time of next dependent message

c) deadline of task t_i

Intuitively, t_i is allowed to extend through all idle times on u_k as long as it doesn't violate data dependency constraints and finishes before its deadline.

Extensibility Metric for Messages: Preemption is not allowed on the bus. The slack for message m_{t_i,t_j} is the bus idle time between m_{t_i,t_j} finish time and start time of the next massage.

Scalability Metric for Tasks: An incremental design change may introduce new tasks in the system. When a task t_k is added to the schedule, we would like to accommodate it without

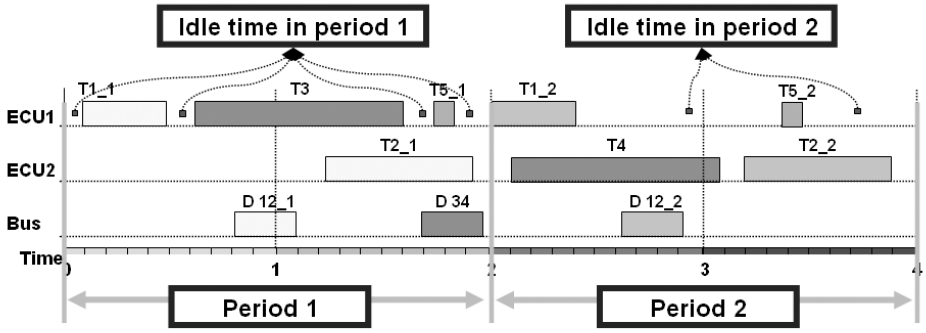


Figure 3.15: Scalability metric illustration.

changing the existing schedule, i.e. maintaining all existing task start times. The new task must be executed before its deadline, which is equal to its period. As preemption is allowed, all idle time in a period can be used by a new task. If the task period is less than the super period, the task is repeated through task graph expansion until the length of periods equals the super period. Each repetition must contain enough idle time to accommodate the new task. (see Figure 3.15)

Numerically, the maximum WCET of a new task t_k that could be accommodated in the existing schedule is defined as the minimum all idle time across all repetitions of the period.

Intuitively, all repetitions of the new task must fit into their respective period. The period with the least amount of idle time limits the size of the new task that can be accommodated.

Scalability Metric for Messages: Preemption is not allowed on the bus, so new messages must fit into existing contiguous bus idle time blocks. For a new message with a specific period, the maximum WCTT that could be accommodated is the largest contiguous block of idle time within that period. If the period is smaller than the super period, the period is repeated through task graph expansion.

Numerically, the metric is defined as the minimum of such largest contiguous idle time

blocks across all repetitions of the period.

Intuitively, all repetitions of the new message must fit into their respective period. The largest contiguous idle time in each period limits the size of messages they can accommodate.

3.3.2 Time-Triggered Incremental Design in an Optimization Framework

3.3.2.1 Mathematical Programming Formulation

Mathematical programming has a concise semantics for describing optimization problems. The thesis first develops a set of notations, parameters, and variables to describe the problem (in this sub-section, the notations are slightly different from other part of the thesis for encoding purpose), then applies feasibility constraints afterwards. The extensibility, scalability and multi-objective cost functions are finally presented.

Notations A short summary of the notations used in the formulation is given first.

Sets

τ a set of tasks $\tau = \{t_i | i = 1, ..., n\}$

ϵ a set of ECUs $\epsilon = \{u_i | i = 1, ..., m\}$

λ a set of task allocation for ECUs

$\{\kappa_{u_k} | u_k \in \epsilon\}$ $(\kappa_{u_k} = \{t_i | a_{t_i,u_j} = 1\})$ [*1]

π a set of task pairs running on the same ECU

$\{(t_i, t_j) | t_i, t_j \in \kappa_{u_k}, t_i \neq t_j, u_k \in \epsilon\}$

σ a set of task pairs with DD* on the same ECU

$\{(t_i, t_j) | t_i, t_j \in \kappa_{u_k}, t_i \prec t_j, u_k \in \epsilon\}$

θ a set of task pairs with no DD* on the same ECU

$\{(t_i, t_j) | t_i, t_j \in \kappa_{u_k}, t_i \prec\succ t_k, u_k \in \epsilon\}$

ϖ a set of task pairs with DD* on different ECU

$\{(t_i, t_j) | t_i \in \tau, t_j \in \tau, t_i \prec t_j, a_{t_i,u_k} + a_{t_j,u_k} \leq 1\}$

$*^1$ $\mathbf{a}_{t_i,u_j}$: denotes task t_i is allocated in ECU u_j,

one task could only be allocated on one ECU

DD*: data-dependency

$t_i \prec t_j$ denotes task t_i precedes task t_j, and $t_i \prec\succ t_j$ denotes task t_i do not have data dependency between them, while $t_i \neq t_j$ means that task t_i and task t_j do not reference the same task.

Parameters and Variables: A task t_i with period p_i, deadline d_i released at r_i, with WCET C_i, starts to execute at s_i and finishes at time f_i. During execution, it may be preempted by another task t_j. A message sent from task t_i to task t_j starts to transmit on the bus at time $s^m_{t_i,t_j}$, with WCTT $t^m_{t_i,t_j}$, and finishes transmission at time $f^m_{t_i,t_j}$.

Following is the binary variables used in the formulation:

$$y_{t_i,t_j} = \begin{cases} 0 & \text{if start time of } t_i \text{ precedes start time of } t_j, \\ 1 & \text{otherwise.} \end{cases}$$

$$p_{t_i,t_j} = \begin{cases} 0 & \text{if task } t_i \text{ is not preempted by task } t_j \\ 1 & \text{otherwise.} \end{cases},$$

$$z_{t_i,t_j,t_k,t_l} = \begin{cases} 0 & \text{if message transmitted between } t_i,t_j \quad \text{precedes message between } t_k,t_l, \\ 1 & \text{otherwise.} \end{cases}$$

3.3.2.2 Feasibility Constraints

A schedule is feasible if it satisfies the constraints of the architecture. Each ECU cannot process more than one task concurrently, although a task could be suspended to allow another task to execute, i.e. preemption. The bus can only transmit one message at a time. The is no preemption for the messages. Collision of messages implies infeasible schedule.

Release and Deadline Constraints: A hard real-time system required all tasks to start after their release time, and finish before their deadlines.

$$r_{t_i} \leq s_{t_i}, t_i \in \tau \tag{3.29}$$

$$f_{t_i} \leq d_{t_i}, t_i \in \tau \tag{3.30}$$

Task and Message Constraints: A task t_i in execution may be preempted by another task t_j, which may again be preempted by some other tasks. Task t_i resumes its execution after task t_j finishes. A message is not allowed to be preempted by another message during its transmission. Its finish time is trivially calculated.

$$f_{t_i} = s_{t_i} + C_{t_i} + \Sigma_{(t_i,t_j)\in\theta}(p_{t_i,t_j} \times C_{t_j}) \tag{3.31}$$

$$s^m_{t_i,t_j} + t^m_{t_i,t_j} = f^m_{t_i,t_j} \qquad (t_i,t_j) \in \varpi \tag{3.32}$$

Data Dependency Constraints: Consider a task pair with data dependency between them, if they are running on the same ECU, then there is no communication between them through

bus, this corresponds to Equation 3.33. There will be a message transmitted on the bus if the task pair running on different ECU, then this will lead to constraints 3.34 and 3.35.

$$f_{t_i} \leq s_{t_j} \quad (t_i, t_j) \in \sigma \tag{3.33}$$

$$f_{t_i} \leq s^m_{t_i,t_j} \quad (t_i, t_j) \in \varpi \tag{3.34}$$

$$s^m_{t_i,t_j} + t^m_{t_i,t_j} \leq s_{t_j} \quad (t_i, t_j) \in \varpi \tag{3.35}$$

Task Mutually Exclusive Constraints: Scheduling tasks on an ECU implies selecting an order to execute the tasks. y_{t_i,t_j} describes the order for a pair of tasks on the same ECU. Order is implied for task pairs in σ, the set of task pairs, with data dependency, allocated to the same ECU. This constraint only applies to task pair set θ, pairs with no data dependencies, and are allocated to the same ECU. M is a large constant used to facilitate the mutually exclusive constraints formulation. Task mutual exclusivity is defined only on the task start time. Preemption constraints must be used to define task finish time.

$$s_{t_i} < s_{t_j} + y_{t_i,t_j} \times M \quad (t_i, t_j) \in \theta \tag{3.36}$$

$$s_{t_j} < s_{t_i} + (1 - y_{t_i,t_j}) \times M \quad (t_i, t_j) \in \theta \tag{3.37}$$

$$y_{t_i,t_j} + y_{t_j,t_i} = 1 \quad (t_i, t_j) \in \theta, (t_j, t_i) \in \theta \tag{3.38}$$

Preemption Constraints: Based on the mutually exclusive constraints, there is a need to have more constraints as follows to finish modeling the preemption constraints:

$$f_{t_i} \leq s_{t_j} + y_{t_i,t_j} \times M + p_{t_i,t_j} \times M \quad (t_i, t_j) \in \theta \tag{3.39}$$

$$f_{t_j} < f_{t_i} + y_{t_i,t_j} \times M + (1 - p_{t_i,t_j}) \times M \quad (t_i, t_j) \in \theta \tag{3.40}$$

$$p_{t_i,t_j} + p_{t_j,t_i} \leq 1 \quad (t_i,t_j) \in \theta, (t_j,t_i) \in \theta \tag{3.41}$$

$$p_{t_i,t_j} \leq 1 - y_{t_i,t_j} \quad (t_i,t_j) \in \theta \tag{3.42}$$

Equation 3.39 just make sure that if task t_i precedes task t_j and is also preempted by it, the finish time of task t_j should be earlier than the finish time of task t_i. Both task t_i and t_j can not be preempted by each other leads to equation 3.41. It is very clear that if task t_i does not precede task t_j, then t_i can not be preempted by t_j, this corresponds to Equation 3.42.

Message Mutually Exclusive Constraints: Similarly, for the message transmitted on the bus, the corresponding mutually exclusive constraints apply as follows:

$$s^m_{t_i,t_j} + t^m_{t_i,t_j} \leq s^m_{t_k,t_l} + z_{t_i,t_j,t_k,t_l} \times M \tag{3.43}$$

$$s^m_{t_k,t_l} + t^m_{t_k,t_l} \leq s^m_{t_i,t_j} + (1 - z_{t_i,t_j,t_k,t_l}) \times M \tag{3.44}$$

Where $(t_i,t_j) \in \varpi, (t_k,t_l) \in \varpi, i \neq k, or, j \neq l.$

3.3.2.3 Extensibility, Scalability, and Multi-Objective Cost Function

In order to define the cost function exactly in terms of the extensibility and scalability metric the resulting formulation becomes a Mixed Integer Non-linear Programming problem, there is not an efficient way to solve even a small scale problem due to the computing ability of non-linear integer problem solver. A good approximation to the cost function is needed so that the formulation becomes a practically solved problem in a reasonable time. The approximated cost functions are:

$$Max \quad E = \sum_{(t_i,t_j)\in\varpi} w_{t_i,t_j} \times ((s^m_{t_i,t_j} - f_{t_i}) + (s_{t_j} - f^m_{t_i,t_j})) \tag{3.45}$$

$$Min \quad S = \sum_{u_j\in\epsilon} \sum_{t_i\in\tau} a_{t_i,u_j} \times (\iota_{t_i} - \alpha_{u_j})^2 + \sum_{(t_i,t_j)\in\varpi} (\iota^{bf}_{t_i,t_j} - \alpha_{bus})^2 \tag{3.46}$$

$$Max \quad ES = k_1 \times E - k_2 \times S \tag{3.47}$$

α_{u_j} is the average idle time on ECU u_j, while α_{bus} is the average time on the bus. In equation 3.45, w_{t_i,t_j} is the a weight parameter to the task pair in ϖ, which might denote the correspondence critical level for the task pair. The function try to maximize the sum over all the slacks associated with each task and each message. Generally, it is a good idea to put a big weight to make more room in time for the task pair which has higher probability to increase their WECT. By this summation, approximation of the extensibility metric is achieved. Equation 3.46 evenly distributes all the idle time as much as possible by minimizing the variance of idle time over all the ECUs. Based on these extensible and scalable approximation cost function, it is simple to consider them jointly to make a more suitable schedule for some specific incremental design requirement. Coefficient k_1 and k_2 are used to tune the cost function to see the fitness of resulting schedule for different application instances.

To better understand the above objective function and finish the formulation, constraints for the idle time on tasks ι_{t_i} and $\iota^{bf}_{u_k}$, idle time on bus ι_{t_i,t_j} and ι^{bf}_{bus} go as follows,

$$\iota_{t_i} \leq (s_{t_j} - f_{t_i}) + y_{t_i,t_j} \times M + p_{t_i,t_j} \times M, \quad (t_i, t_j) \in \pi \tag{3.48}$$

$$\iota_{t_i} \leq P - f_{t_i}, \quad t_i \in \tau \tag{3.49}$$

$$\iota_{t_j} \leq 0 + y_{t_i,t_j} \times M + (1 - p_{t_i,t_j}) \times M, \quad (t_i, t_j) \in \pi \tag{3.50}$$

$$\iota^{bf}_{u_k} \leq s_{t_i}, \quad u_k \in \epsilon, t_i \in \kappa_{u_k} \tag{3.51}$$

$$\sum_{t_j \in \kappa_{u_k}} \iota_{t_j} + \iota^{bf}_{u_k} = P - \sum_{t_j \in \kappa_{u_k}} C_{t_j}, \quad u_k \in \epsilon \tag{3.52}$$

Equation 3.48 basically sets a constraint for the idle time after each task on a specific ECU. For example, consider task t_i and its consecutive task t_j on the same ECU, the idle time after task t_i on

this ECU should be less than or equal to the difference of start time of task t_j and the finish time of task t_i if t_j does not preempt t_i. $\iota_{u_k}^{bf}$ is the first idle time on each ECU, and similar to equation 3.48, it is easy to get the meaning of constraints 3.49- 3.52

$$\iota_{t_i,t_j} \leq (s_{t_k,l}^m - f_{t_i,t_j}^m) + z_{t_i,t_j,k,l} \times M \tag{3.53}$$

$$\iota_{t_i,t_j} \leq P - f_{t_i,j}^m \tag{3.54}$$

Where $(t_i, t_j) \in \varpi, (t_k, t_l) \in \varpi, i \neq k, or, j \neq l$

$$\iota_{t_i,t_j}^{bf} \leq s_{t_i,t_j}^m, \qquad (t_i, t_j) \in \varpi \tag{3.55}$$

$$\sum_{(t_i,t_j)\in\varpi} \iota_{t_i,t_j} + \iota_{bus}^{bf} = P - \sum_{(t_i,t_j)\in\varpi} C_{t_i,t_j} \tag{3.56}$$

Similar to equation 3.48- 3.52, equations 3.53- 3.56 are the idle time constraints set on the message transmitted on the bus, the implication of these equations are straight forward.

3.3.2.4 Cost Function Evaluation based on Metrics

The extensibility and scalability metrics described in Section 3.3.1 is abstracted into a multi-objective cost function in Section 3.3.2.3. To evaluate the effectiveness of this abstraction, I need to describe a real industrial problem to be solved in the mathematical programming framework, apply the multi-objective cost function, and extract extensibility and scalability metrics from the resulting schedule. The extraction is applied to the scheduling result for the solver, outside the mathematical programming framework. It follows exactly the metrics described in Section 3.3.1. Therefore it is an objective monitor of the effectiveness of objective functions in abstracting the metrics.

3.3.3 Experimental Results

The formulation is described using AMPL, the mathematical programming modeling language. ILOG CPLEX 9.0.0 mathematical programming solver is used on i686 or equivalent server running Linux. Data from results generated are plotted and extracted with our own interpreters.

3.3.3.1 System Description

The thesis presents a set of advanced automotive control application case used by Nagarajan et al [48]. The system contains three threads of control: adaptive cruise control (ACC), electric power steering (EPS), and traction control (TC). Data is collected from sensors and processed in one or more ECUs. The ECUs then control actuators to respond to the sensed environment. ACC maintains safe distance between two cars. EPS provides assistance to help steer. TC actively stabilizes the vehicle under slippery road conditions. These functions are mapped to an architecture with 10 sensors, 3 processor ECUs, and 4 actuators. The scheduling problem contains 24 tasks, of which 6 are expanded using task expansion. During the task graph expansion, a task Tn is renamed to Tn_i, where n is the task ID shown in the Function Graph, and i is natural number representing the instance it appears in. As these are safety critical functions in a vehicle, all deadlines must be met.

3.3.3.2 Results

A typical scheduling heuristic for distributed embedded system is to minimize schedule end-to-end delay. I compare such a heuristic with the results optimized with respect of to extensibility and scalability cost functions.

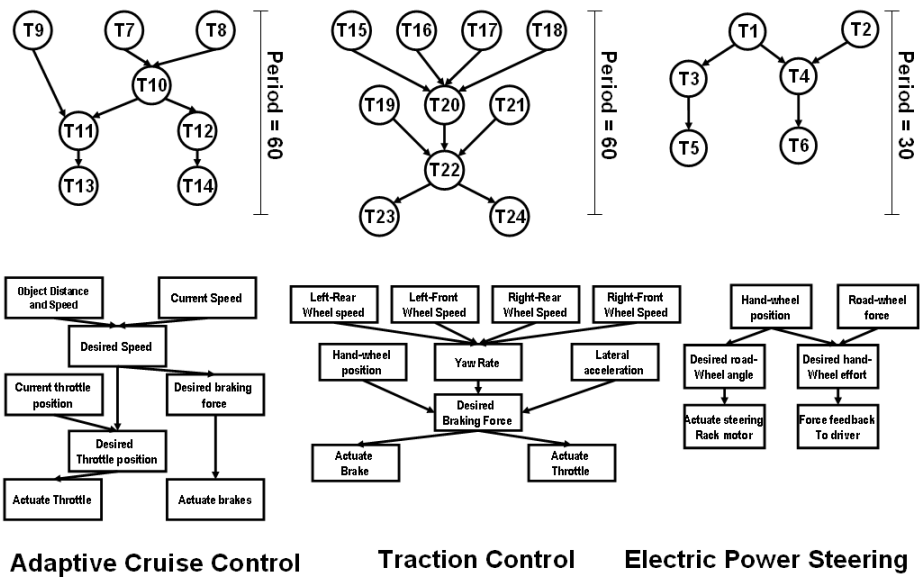


Figure 3.16: Case study functionality graph.

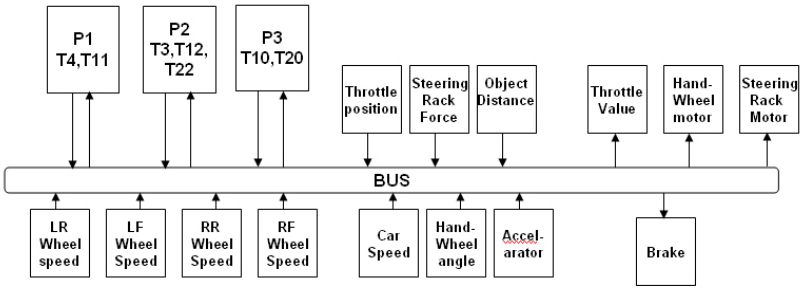


Figure 3.17: Case study architecture graph.

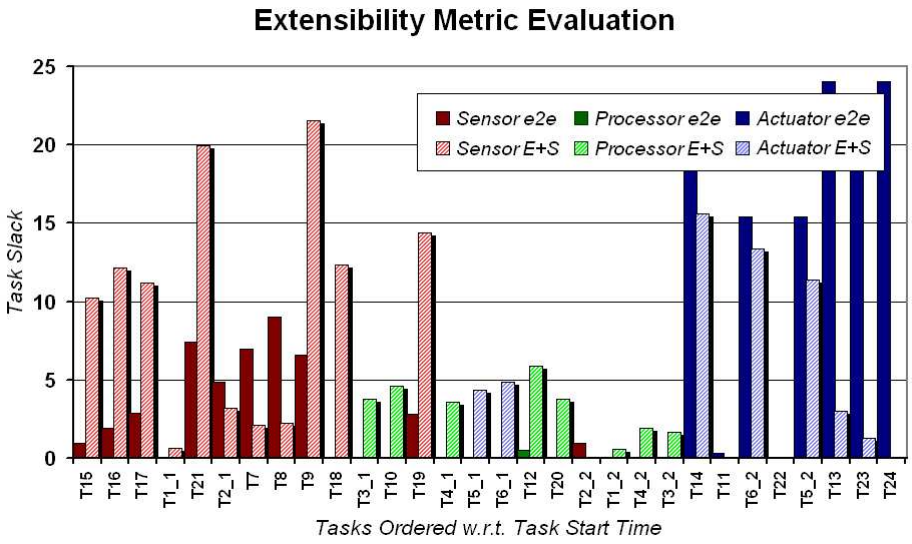
- **Traditional result:** minimize end-to-end delay
- **Optimized result:** maximize extensibility and scalability

When scheduled for minimum end-to-end delay, the function consumes approximately 3/4 of the available super period. This implies a 33% margin for future incremental developments. The cost function abstracted from extensibility and scalability metrics utilizes this margin to enhance the scheduling result.

Extensibility is captured by quantifying the maximum slack of a task in a schedule. It is the upper bound of task WCET extension the schedule could accommodate. The task slacks, as defined in Section 3.3.1, are extracted from the two scheduling results.

The y-axis shows task slack for each task, and the x-axis is ordered with respect to task start times. The lighter bars are the optimized results, and the darker bars are the traditional result. Compared to the traditional result, the total slack in the optimized result increased significantly from 170.3 to 189.6 units of time. Slacks accumulated at the actuator nodes in the traditional result are more distributed among the sensors and tasks on processors in the optimized results. This shows that the multi-objective function is effective in optimizing for extensibility as we defined it.

Scalability is captured by quantifying the maximum new task size the schedule can accommodate with certain periods. The ECU scalability is defined in Section 3.3.1. The vertical axis represents the period of the new tasks to be added, and the horizontal axis represent the cumulative idle time in one period from on all processor ECUs. Darker bars show the traditional result, and lighter bars show the optimized result. At each period, the optimized result can accommodate larger new tasks than the traditional result. This shows the effectiveness in the multi-objective function for optimizing scalability.

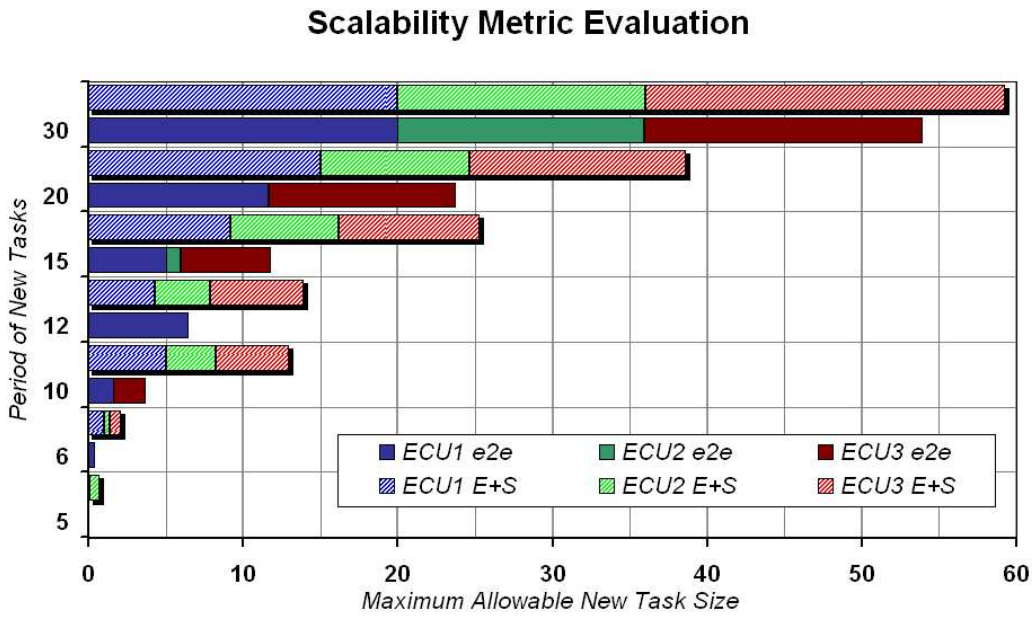


Dark bars: Real task slack from traditional results

Light bars: Real task slack from optimized results

Graph shows redistribution of slacks

Figure 3.18: Extensibility metric evaluation.



Light bars: Upper bound of new task size in optimized results

Dark bars: Upper bound of new task size in traditional results

Graph shows potential for new task to be inserted at different periods

Figure 3.19: Scalability metric evaluation.

I use three incremental design scenario to illustrate the effectiveness of our metrics in handling design changes. The EPS unit has a period of 30 units of time. It is duplicated through task graph expansion to be scheduled in a 60 unit super period. The EPS tasks represent a critical section in the schedule. The first 2 scenarios contain changes to the EPS unit. The third one concerns changes to the ACC unit.

Scenario one: A Hand Wheel Effort upgrade allows different drivers to select profiles of desired force feedback on the steering wheel based on road conditions. The upgrade causes the WCET of Hand Wheel Effort to be extended by one unit of time.

Scenario two: Enhanced steering uses Road-Wheel Force as input. The enhancement requires a new task T_{new} to be added between $T2$ and $T5$ on $P2$. There will be two new messages, $T2$ to T_{new} and T_{new} to $T5$.

Scenario three: Basic ACC is upgraded to a Stop-N-Go ACC. The feature would predict a desired speed based on digital map information and forward looking sensor. To implement these design changes, we add a new task to the ECU, $P1$, extend the existing task $T10$, and add in two messages, one between $T7$ and $T11$, and another between $T19$ and $T10$.

The baseline results are defined as the traditional and optimized result. Changes to a task would infer an ECU reprogramming. Changes to message start or finish time implies reprogramming both the message source and destination ECU. Minimum number of ECUs that must be re-programmed to achieve a feasible schedule is calculated. New tasks and messages are scheduled under the same cost functions as the legacy tasks. The ECU count is confirmed with a rescheduling of only the reprogrammed ECU and keeping all other ECU schedules fixed. This ECU reschedule is an indicator of the amount of re-verification that is required. The result is shown in Table 3.5.

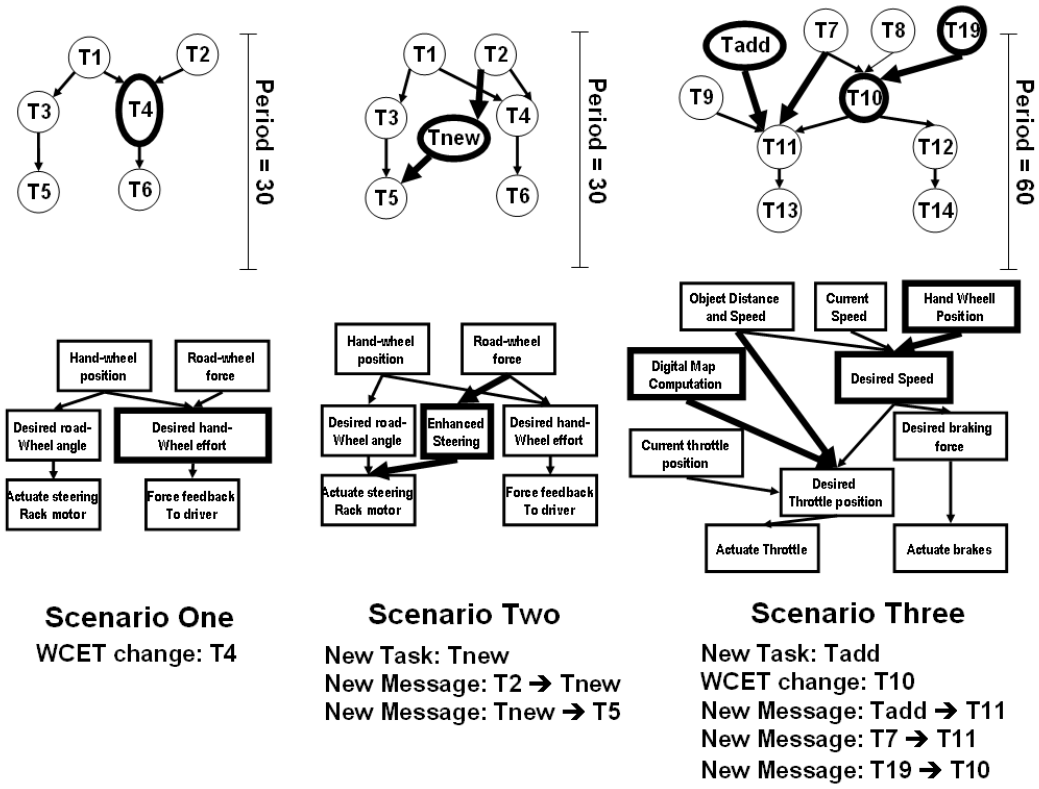


Figure 3.20: Incremental design scenarios.

Scenario one is a simple illustration of the effectiveness of our approach. A tightly coupled schedule as in the traditional result responds poorly to small incremental changes. A small increase in one task execution time requires 15 out of 17 ECUs in the system. As messages in the bus are forced to shift according to task start and finish time changes, the change is propagated to almost all other ECUs. In the optimized result, $T4$ has enough slack to accommodate this scenario, and the change is accommodated by only reprogramming $T4$ on the ECU being upgraded.

Scenario two describes a larger incremental change on EPS. This change involves tasks on 3 ECUs, and does not affect the most critical section of the schedules. The traditional result requires changes to 8 ECUs. The ECU accommodating the new task needs to shift legacy tasks to reach a feasible solution, which affects other messages. The bus also lack idle time slots to accommodate the new messages. The new messages and new task induced message shifts, which would aggregate the effects of the changes. One important observation is that since EPS duplicated through task graph expansion, there are two instances of $T4$ to change. $T4_1$ in the first half of the schedule required the most amount of reprogramming. $T4_2$ in the second half of the cycle utilizes the slacks at the end of the cycle and required the minimal amount of reprogramming.

In the optimized result, scalability cost function is effective in spreading out tasks on the ECUs and the messages on the bus such that idle time slots are available to accommodate the new task and messages. Only the minimal of 3 involved ECUs are reprogrammed.

Scenario three is the largest incremental change we analyzed. It involves one new task, one task extension and two new messages. Five ECUs are used to implement the changes. In the traditional result, similar change propagation mechanisms affected a total of 8 ECUs. In the optimized result, only the necessary five ECUs required reprogramming.

These three scenarios represent a variety in the sizes of incremental changes. The optimized result is shown to reduce the necessary reprogramming of the system. Evaluation and re-verification of the system could be simplified.

The multiple ECU (processor) scheduling problem is an NP-hard problem [73]. The mathematical programming approach is computationally intensive, and suitable only for moderately sized problems. The case study shown here required one hour of run-time for the full extensibility and scalability cost functions. The complexity mainly resides in solving for the precedence binary variable y_{t_i,t_j} and z_{t_i,t_j,t_k,t_l}. To deal with larger problem sizes, a heuristic could be used to obtain a feasible solution first, lock down all precedence binary variables, then use the cost function to optimize. However, there is a risk of being locked into a bad task ordering. Algorithms to perturb a feasible ordering need to be developed in order to move toward a more optimal solution.

This chapter presented an optimization framework based on MILP for the scheduling of real-time applications on FlexRay-based systems. The work provided solutions for task scheduling policies based on existing industry standards and optimality with respect to an extensibility metric. Additionally, I captured extensibility and scalability metrics in scheduling a hard real time embedded system, and recognized its implications in accelerating time-to-market of a system development process by reducing development and re-verification burden in an incremental design flow. The approaches are shown to be effective for industrial problems.

Signal	Send	Size	Recv	Signal	Send	Size	Recv
σ_1	τ_{15}	32	$\tau_{22}/\tau_{26}/\tau_{30}/\tau_{34}$	σ_{67}	τ_{44}	16	τ_{12}
σ_2	τ_{15}	32	$\tau_{22}/\tau_{26}/\tau_{30}/\tau_{34}$	σ_{68}	τ_{44}	16	τ_{12}
σ_3	τ_{15}	32	$\tau_{22}/\tau_{26}/\tau_{30}/\tau_{34}$	σ_{69}	τ_{44}	16	τ_{12}
σ_4	τ_{15}	32	$\tau_{22}/\tau_{26}/\tau_{30}/\tau_{34}$	σ_{70}	τ_{44}	16	τ_{12}
σ_5	τ_{20}	32	$\tau_{22}/\tau_{26}/\tau_{30}/\tau_{34}$	σ_{71}	τ_{44}	16	τ_{12}
σ_6	τ_{20}	32	$\tau_{22}/\tau_{26}/\tau_{30}/\tau_{34}$	σ_{72}	τ_{44}	16	τ_{12}
σ_7	τ_{20}	32	$\tau_{22}/\tau_{26}/\tau_{30}/\tau_{34}$	σ_{73}	τ_{44}	16	τ_{12}
σ_8	τ_{20}	32	$\tau_{22}/\tau_{26}/\tau_{30}/\tau_{34}$	σ_{74}	τ_{44}	16	τ_{12}
σ_{37}	τ_{29}	32	$\tau_{22}/\tau_{26}/\tau_{30}/\tau_{34}$	σ_{103}	τ_{12}	16	τ_{17}
σ_{38}	τ_{30}	16	τ_8/τ_{16}	σ_{104}	τ_{12}	16	τ_{17}
σ_{39}	τ_{30}	16	τ_8/τ_{16}	σ_{105}	τ_{12}	16	τ_{17}
σ_{40}	τ_{30}	1	τ_8/τ_{16}	σ_{106}	τ_{12}	16	τ_{17}
σ_{41}	τ_{35}	32	$\tau_{23}/\tau_{27}/\tau_{31}$	σ_{107}	τ_{12}	16	τ_{17}
σ_{42}	τ_{35}	32	$\tau_{23}/\tau_{27}/\tau_{31}$	σ_{108}	τ_{12}	16	τ_{17}
σ_{43}	τ_{35}	32	$\tau_{23}/\tau_{27}/\tau_{31}$	σ_{109}	τ_{12}	16	τ_{17}
σ_{44}	τ_{33}	32	$\tau_{22}/\tau_{26}/\tau_{30}/\tau_{34}$	σ_{110}	τ_{12}	16	τ_{17}
σ_{45}	τ_{33}	32	$\tau_{22}/\tau_{26}/\tau_{30}/\tau_{34}$	σ_{111}	τ_{12}	16	τ_{17}
σ_{46}	τ_{33}	32	$\tau_{22}/\tau_{26}/\tau_{30}/\tau_{34}$	σ_{112}	τ_{12}	16	τ_{17}
σ_{47}	τ_{34}	16	τ_8/τ_{16}	σ_{113}	τ_{12}	16	τ_{17}
σ_{48}	τ_{35}	8	τ_8/τ_{16}	σ_{114}	τ_{12}	16	τ_{17}
σ_{49}	τ_{35}	8	τ_8/τ_{16}	σ_{115}	τ_{12}	16	τ_{17}/τ_{18}
σ_{50}	τ_{34}	16	τ_8/τ_{16}	σ_{116}	τ_{12}	16	τ_{18}/τ_{17}
σ_{51}	τ_{34}	16	τ_8/τ_{16}	σ_{117}	τ_{12}	16	τ_{18}/τ_{17}
σ_{52}	τ_{37}	16	τ_{12}	σ_{118}	τ_{12}	16	τ_{18}/τ_{17}
σ_{53}	τ_{37}	16	τ_{12}	σ_{119}	τ_{12}	16	τ_{18}/τ_{17}
σ_{54}	τ_{39}	16	τ_{12}	σ_{120}	τ_{12}	16	τ_{18}/τ_{17}
σ_{55}	τ_{39}	16	τ_{12}	σ_{121}	τ_{12}	16	τ_{18}/τ_{17}
σ_{56}	τ_{39}	16	τ_{12}	σ_{122}	τ_{12}	16	τ_{18}/τ_{17}
σ_{57}	τ_{39}	16	τ_{12}	σ_{123}	τ_{12}	16	τ_{17}
σ_{58}	τ_{39}	16	τ_{12}	σ_{124}	τ_{12}	16	τ_{17}
σ_{59}	τ_{39}	16	τ_{12}	σ_{125}	τ_{12}	16	$\tau_{39}/\tau_{44}/\tau_{49}/\tau_{54}$
σ_{60}	τ_{39}	16	τ_{12}	σ_{126}	τ_{12}	16	$\tau_{39}/\tau_{44}/\tau_{49}/\tau_{54}$
σ_{61}	τ_{39}	16	τ_{12}	σ_{127}	τ_{12}	16	$\tau_{39}/\tau_{44}/\tau_{49}/\tau_{54}$
σ_{62}	τ_{39}	16	τ_{12}	σ_{128}	τ_{12}	16	$\tau_{39}/\tau_{44}/\tau_{49}/\tau_{54}$
σ_{63}	τ_{39}	16	τ_{12}	σ_{129}	τ_{12}	16	$\tau_{39}/\tau_{44}/\tau_{49}/\tau_{54}$
σ_{64}	τ_{42}	16	τ_{12}	σ_{130}	τ_{12}	16	$\tau_{39}/\tau_{44}/\tau_{49}/\tau_{54}$
σ_{65}	τ_{42}	16	τ_{12}	σ_{131}	τ_{12}	8	$\tau_{39}/\tau_{44}/\tau_{49}/\tau_{54}$
σ_{66}	τ_{44}	16	τ_{12}	σ_{132}	τ_{12}	16	$\tau_{39}/\tau_{44}/\tau_{49}/\tau_{54}$

Table 3.3: Signals for X-by-wire example

		OSEK+confg1				OSEK+confg2			
Method	Iter	# Binary	Time(s)	result	Opt	# Binary	Time(s)	result	Opt
(1)		32672	3600	13	No	42916	3600	44	No
(2)	1	32672	719.9	13	Yes	42916	3600	53	No
	2	-	-	-	-	5848	18.8	44	Yes
(3)	1	25886	631.0	13	Yes	34412	887.4	44	Yes
		OSEKTime+confg1				OSEKTime+confg2			
Method	Iter	# Binary	Time(s)	result	Opt	# Binary	Time(s)	result	Opt
(1)		48447	3600	13	No	63195	3600	45	No
(2)	1	47943	627.0	13	Yes	62537	3600	45	No
	2	-	-	-	-	6953	129.0	44	Yes
(3)	1	33391	606.3	13	Yes	43657	3298.2	44	Yes
		OSEKTime+Preem+confg1				OSEKTime+Preem+confg2			
Method	Iter	# Binary	Time(s)	result	Opt	# Binary	Time(s)	result	Opt
(1)		51079	3600	N/A	No	65863	3600	N/A	No
(2)	1	50460	3600	16	No	65054	3600	47	No
	2	7787	439.5	14	Yes	8559	2404.7	47	Yes
(3)	1	35729	3600	14	No	45995	3600	46	No
	2	4266	1.3	13	Yes	7329	357.2	44	Yes

Table 3.4: Results on the X-by-wire Example

Scenario	Traditional Result	Optimized Result
Scenario one	15	1
Scenario two	8	3
Scenario three	8	5

Table 3.5: Incremental design change evaluation

*Value in table shows number of ECU requiring reprogramming

Chapter 4

Scheduling based Synthesis for Event Triggered Automotive System

Modern automotive architectures support the execution of distributed safety- and time-critical functions on a complex networked system with several buses and tens of ECUs. Schedulability theory provides support for analyzing the worst case end-to-end latencies when the architecture of the system is deployed and the communication and synchronization mechanisms have been defined among all tasks in the system. Schedulability theory not only allows the analysis of the worst case end-to-end latencies and the evaluation of the possible architecture configurations options with respect to timing constraints, but it can also be used in an optimization framework to synthesize design parameters such as selecting the communication and synchronization model that exploits the trade-offs between the purely periodic and the precedence constrained data-driven activation models to meet the latency and jitter requirements of the application, at the same time, period synthesis could be included to the procedure.

For FlexRay based communication system discussed in previous chapter, the starting time of messages transmission on the bus is usually pre-determined during design time. For periodic driven activation of messages transmission on CAN based communication system, the starting time is not pre-determined at design time. The thesis attempts at locking down the worst case response time of tasks/messages and finally compute the end to end latency even for a mixed activation model on the selected path.

In this chapter, preliminaries and definitions are first introduced on event-triggered communication system in Section 4.1, then in Section 4.2 the worst case response time for tasks and messages in CAN is discussed, after the explanation of system activation model in Section 4.3, an optimization framework, based on an ILP formulation of the problem, is presented in Section 4.4 and its effectiveness is demonstrated by applying it to the optimization of a complex real-life GM architecture.

4.1 Preliminaries and Definitions for Event Triggered System

4.1.1 Controller Area Network Background

CAN as discussed in Section 1.3.2.3, is a multicast shared serial bus standard, originally developed in the 1980s by Robert Bosch GmbH, to connect electronic control units (ECUs). CAN was specifically designed to be robust in electromagnetic noisy environments.

This chapter presents the transmission of messages on CAN buses, and shows that the arbitration of messages is based on their ID or simply called priority. The system architecture is usually composed of a number of ECUs inter-connected by a CAN bus.

4.1.2 CAN based System Modeling

The model of distributed end-to-end real-time computations is a *dataflow* of tasks, represented with a *Directed Acyclic Graph*. The *model* is a tuple $\{\mathcal{V}, \mathcal{E}, \mathcal{R}\}$, where $\mathcal{V}$ is the set of vertices, $\mathcal{E}$ is the set of edges, and $\mathcal{R} = \{R_1, \ldots, R_z\}$ is the set of shared resources supporting the execution of the tasks (on CPUs) and the transmission of the messages (on buses).

$\mathcal{V} = \{o_1, \ldots, o_n\}$ is the set of objects implementing the computation and communication functions of the system. o_i can be a task or a message and is characterized by a maximum time requirement C_i and a resource R_{o_i} that it needs to execute or for its transmission. The minimum time requirement is always assumed to be 0. While inaccurate (especially for message objects), this simplification in most cases does not affect in a significant way the quality of the end-to-end latency analysis. All objects are scheduled according to their priority; π_i is the priority of o_i and indexes are assigned by decreasing priority levels; r_i is the worst case response time of o_i, from the activation of the object to its completion in case it is a task, or its arrival at the destination node in case it is a message. w_i is defined as the worst case time spent from the instant the job is released with maximum jitter J_i to its completion or arrival. An object o_i has conceptually one or more *input ports* and one or more *output ports* that are used to exchange data and optionally *activation signals* or events. Each object runs at a base period T_i (and optionally, jitter J_i). It reads its inputs at the time it starts executing, if it is a task, or it samples the incoming signal values and it is enqueued at the activation time in case it is a message. An object may receive its activation signal from one or more of its input ports or from a timer sending periodic activation signals. If an object is activated by a signal on an incoming link, then it must have the same period to the predecessor object sending the activation signal. At the end of its execution or transmission, it delivers its results (task) or its

data content (message) and, where required, activation signals on its output ports.

$\mathcal{E} = \{l_1, \ldots, l_m\}$ is the set of *links*. A link $l_i = (o_h, o_k)$ connects the output port of object o_h (the source) to the input port of object o_k (the sink). Alternatively, a link may be labeled with the indexes of the source and destination task as in $l_{h,k} = (o_h, o_k)$. A link l_i may carry the activation signal produced when the source object completes its execution or transmission and instantaneously received on the input port of the sink. However, a different communication and synchronization model is possible, where the sink is activated by a periodic timer and, when it executes, reads the latest value that was transmitted over the link (and stored into a buffer). The source and the sink of link l_i are also denoted by $src(l_i)$ and $snk(l_i)$, respectively.

When an object is activated by the completion of a predecessor an *event-driven* activation model can be defined. In this case, I need to find a model for the activation semantics that can be expressed in terms of a periodic event stream with jitter. If an object is activated by a single completion event, then the only condition is that its period must be an integer multiple of the predecessor object period. In this case, the activation semantics is of one every k signals. I define a less restrictive activation semantics by allowing an object to be activated by multiple completion events. In this case, the activation is of type AND, meaning that all the predecessor objects on the selected links must be completed in order for the object to be activated. The only allowed case for multiple activation events from multiple incoming links is when the links are connected to predecessor objects having periods that are integer dividers of the target object period, have a unique common predecessor, and are scheduled on the same resource. In this case, I define a set of link groups $\mathcal{G} = \{lg_1, \ldots, lg_k\}$ where each link group $lg_i = \{l_{i_0}, \ldots l_{i_{k_i}}\}$ has the following properties, $snk(l_{i_j}) = snk(l_{i_l})$ and $R_{src(l_{i_j})} = R_{src(l_{i_l})}$ for any link pair $l_{i_j}, l_{i_l} \in lg_i$. If $\tau_{j1} = src(l_{i_j})$ and

$\tau_{j2} = snk(l_{i_j})$ then $kT_{j1} = T_{j2}$ for some integer k. Finally, $\forall lg_i, \exists! o_p$ such that $\forall l_{j,k} \in lg_i$ there exists a link $l_{p,j} \in \mathcal{E}$. and there is no other incoming link to o_j. If all the links in a group carry an activation signal, then the source objects must be activated at the same time or they must all be activated by a completion event. These last conditions do not apply to singleton groups. $G(o_k)$ is the set of link groups that are incoming to o_k. For example, in Figure 4.1, l_1, l_2, l_3 belong to group lg_1, l_4, l_5 to lg_2 and l_6 to lg_3 consisting of only one link. Hence, an object can be activated by a periodic trigger, by a signal coming from a single predecessor object or by the AND composition of signals coming from a single link group. In this last case, the object is actually activated by the completion of the lowest priority object o_r in the group lg_i, which is called *group representative* $o_r = rep(lg_i)$.

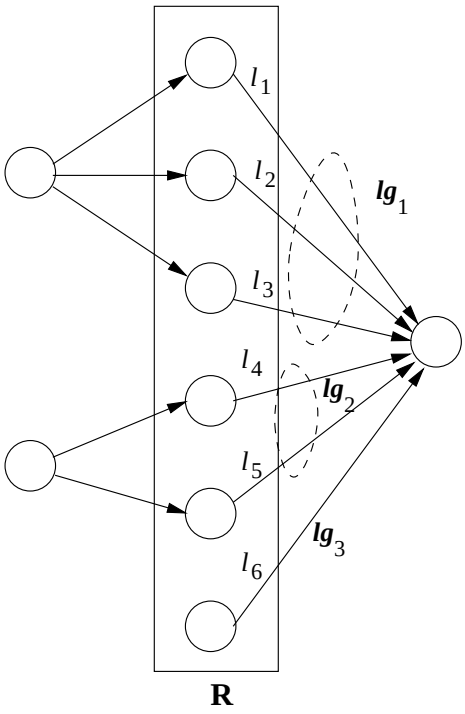


Figure 4.1: Example of link groups.

An *external event* results from the execution of a virtual object o_i with no input links, representing the environment. External events can be*periodic* with period T_i and jitter J_i, or *sporadic* with a minimum interarrival time, equally denoted by T_i.

An *output* object o_j represents data consumption by the environment, e.g. when the system updates an actuator.

A *functional chain* or *Path* from o_i to o_j, or $P_{i,j}$, is an ordered sequence $P = [l_1, \ldots, l_n]$ of links that, starting from $o_i = src(l_1)$, reach $o_j = snk(l_n)$ crossing a unique sequence of $n + 1$ objects such that $snk(l_k) = src(l_{k+1})$. o_i is the chain's source and o_j its sink.

When tasks and messages are activated periodically and communicate on a freshest value semantics, several definitions of end-to-end latency (and the associated deadline) are possible. In the work, the end-to-end latency $L_{i,j}$ associated to a path $P_{i,j}$ is defined as the largest possible time interval that is required for the change of the input at one end of the chain to be propagated to the last task at the other end of the chain, whatever is the state of the tasks in the path and regardless of the fact that some intermediate result may be overwritten before it is read. $d_{i,j}$ is the corresponding path deadline.

I assume in this chapter that the application can tolerate the semantic variation when changing from one synchronization model to the other. In many control applications, the non-determinism in time introduced by the periodic activation model and the jitter introduced by the event-driven activation can both be tolerated within acceptable ranges.

4.2 Design Analysis of CAN based Communication System

In order to characterize the end to end latency for a path, there is a need to calculate the response time for tasks and messages on the selected path.

4.2.1 Response Time for Tasks: Processor Scheduling

The worst case response time for a periodic task τ_i, activated with maximum jitter J_i in a generic preemptive and priority based scheduled system, when the response time can be larger than T_i, is given by the following formula.

$$\begin{aligned} & w_i(q) = (q+1)C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i(q) + J_j}{T_j} \right\rceil C_j \\ & w_i = \max_q \{w_i(q) - qT_i\} \\ & r_i = J_i + w_i \\ & \text{for all } q = 0 \ldots q^* \text{ until } r_i(q^*) \leq T_i \end{aligned} \tag{4.1}$$

Where $j \in hp(i)$ means all the object indexes such that $\pi_j \geq \pi_i$ and $R_{o_i} = R_{o_j}$. The need of evaluating the first q instances inside the busy period is caused by the uncertainty about the instance which causes the worst case response time. However, a lower bound on the worst case response time can be obtained by restricting the computation to the first instance, see 4.2. This bound is tight in case the response time is lower than the period.

$$\begin{aligned} & w_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i + J_j}{T_j} \right\rceil C_j \\ & r_i = J_i + w_i \end{aligned} \tag{4.2}$$

Linear upper and lower bounds for the solution to the previous fixed point problem can be obtained from

$$w_i^{\uparrow} = C_i + \sum_{o_j \in hp(o_i)} (\frac{w_i^{\uparrow} + J_j}{T_j} + 1) C_j \tag{4.3}$$

$$w_i^{\downarrow} = C_i + \sum_{o_j \in hp(o_i)} (\frac{w_i^{\downarrow} + J_j}{T_j}) C_j \tag{4.4}$$

$$w_i^{\downarrow} = \frac{C_i + \sum \frac{C_j}{T_j} J_j}{1 - \sum \frac{C_j}{T_j}} \quad r_i^{\downarrow} = w_i^{\downarrow} + J_i \tag{4.5}$$

$$w_i^{\uparrow} = \frac{C_i + \sum \frac{C_j}{T_j} (J_j + 1)}{1 - \sum \frac{C_j}{T_j}} \quad r_i^{\uparrow} = w_i^{\uparrow} + J_i \tag{4.6}$$

if $u_i = C_i/T_i$ then the previous formula can be solved, and a linear combination of the upper and lower bounds, with coefficient α, yields the following

$$\tilde{r_i}(\alpha) = \alpha r_i^{\uparrow} + (1 - \alpha) r_i^{\downarrow} \tag{4.7}$$

$$\tilde{r_i}(\alpha) = J_i + \frac{C_i + \alpha \sum_{j \in hp(i)} C_j + \sum_{j \in hp(i)} J_j u_j}{1 - \sum_{j \in hp(i)} u_j} \tag{4.8}$$

4.2.2 Response Time for Messages: Bus Scheduling

In this chapter I assume that message objects are transmitted over CAN buses. The CAN bus provides a channel arbitration policy that gives at each round the transmission rights to the message with the lowest identifier, therefore allowing (fixed) priority based scheduling. The evaluation of the worst case latencies for the messages sent over the CAN bus follows the same principles that are used for evaluating the worst case response time of the tasks, with the exception that an additional blocking term B_i must be included in the formula in order to account for the non preemptability of CAN frames and the transmission time of the message must also be considered as

non preemptable. The blocking term B_i for a generic message o_i can be computed as the worst case transmission time of any frame having a priority lower than π_i and sharing the same bus resource.

$$\begin{aligned} & wq_i(q) = B_i + qC_i + \sum_{j \in hp(i)} \left\lceil \frac{wq_i(q) + J_j}{T_j} \right\rceil C_j \ (wq_i > 0) \\ & w_i = \max_q \{C_i + wq_i(q) - qT_i\} \\ & r_i = w_i + J_i \\ & \text{for all } q = 0 \ldots q^* \text{ until } r_i(q^*) \leq T_i \end{aligned} \tag{4.9}$$

again, a lower bound on w_i and r_i can be computed by only considering the first instance ($q = 0$).

$$\begin{aligned} & wq_i = B_i + \sum_{j \in hp(i)} \left\lceil \frac{wq_i + J_j}{T_j} \right\rceil C_j \ (wq_i > 0) \\ & w_i = wq_i + C_i \\ & r_i = w_i + J_i \end{aligned} \tag{4.10}$$

Similar to processor scheduling, the response times of messages can be approximated by linear functions of the jitter variables, and the linear combination with coefficient α is

$$\tilde{r_i}(\alpha) = J_i + C_i + \frac{B_i + \alpha \sum_{j \in hp(i)} C_j + \sum_{j \in hp(i)} J_j u_j}{1 - \sum_{j \in hp(i)} u_j} \tag{4.11}$$

4.2.3 End to End Latencies

The end to end latencies should be clear after identifying all the worst case response time of tasks on ECUs and messages on buses. However, there is another aspect related to the computation of the end to end latencies on paths, which is activation model of tasks/messages, it

could be either periodic activation model or data-driven. I will illustrate the end to end latency starting from next section given the activation of computing objects.

4.3 System Activation Model

4.3.1 Periodic Activation Model

In the periodic activation model (an example in Figure 4.2), the release jitter is zero and the worst case end-to-end latency (denoted as $L_{i,j}$ for a path $P_{i,j}$) can be computed for each path by adding the worst case response times and the periods of all the objects in the path ($r_k = w_k$).

$$L_{(i,j)} = \sum_{k:o_k \in P(i,j)} (T_k + r_k) \tag{4.12}$$

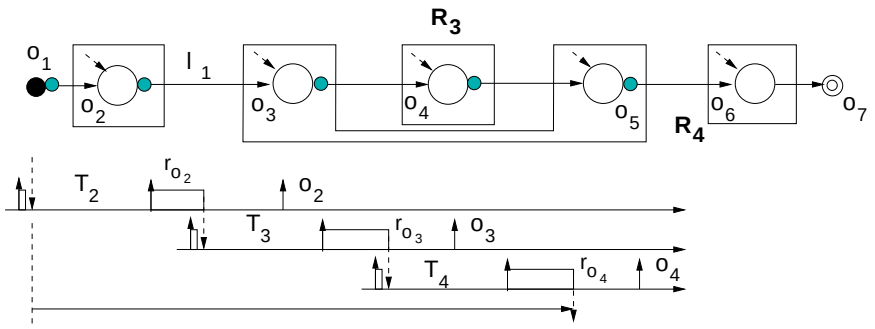


Figure 4.2: Periodic activation model.

In the worst case, as shown in Figure 4.2, the external event arrives right after the completion of the first instance of task o_2 with minimum (negligible) response time. The event data will be read by the task on its next instance and the result will be produced after its worst case response time, that is, $T_2 + r_2$ time units after the arrival of the external event. The same reasoning applies to the execution of the following objects.

4.3.2 Data-driven Activation Model

In the data driven activation model (an example in Figure 4.3), if I assume the same activation period for all the nodes that are activated in a computation chain, then for all the intermediate neighboring nodes $o_i \rightarrow o_j$ it is clearly $r_i = J_j$. The worst case end-to-end latency can be computed for each path by adding the worst case queuing and execution/transmission times of all the objects in the path.

$$L_{(i,j)} = \sum_{k:o_k \in P(i,j)} w_k \tag{4.13}$$

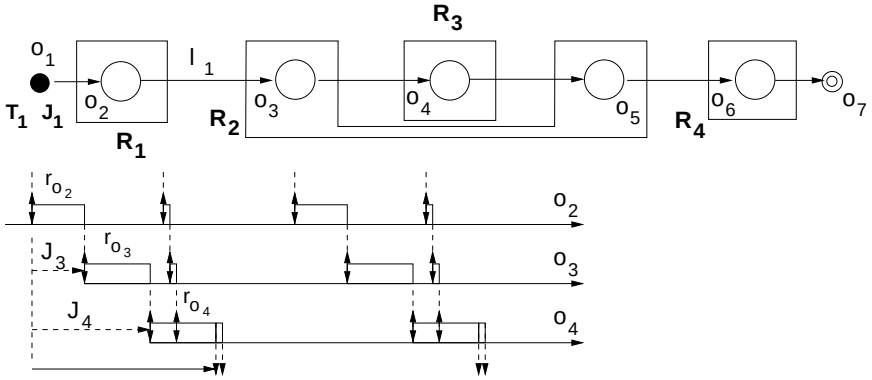


Figure 4.3: Data driven activation model.

However, in this case, as shown in the figure, the worst case jitter of the activation events grows larger as the computations propagate along the chain. The latency along the path is typically lower if compared with the previous case, but the large jitter in the activation of the intermediate tasks and messages means that tasks and messages may be activated according to bursty pattern of events. These bursts of high priority tasks and messages may increase the response time of the lower priority objects that share resources with them.

4.4 Parameter Synthesis of CAN based Communication System

4.4.1 Parameter Synthesis Overview

Both static and dynamic priority, distributed as well as centralized scheduling methods have been proposed in the past for distributed systems. Static and centralized scheduling is typical of time triggered design methodologies, like the Time-Triggered Architecture (TTA) [39] and its network protocol TTP and of implementations of synchronous reactive models, including Esterel and Lustre [8]. The recent Flexray standard [25] for high speed communication in automotive systems, provides two transmission windows, one dedicated to time-driven periodic streams with static, design-time assignment of transmission slots, and the other for asynchronous, event-driven communication.

Event-based priority scheduling is also very popular in control applications. Priority based scheduling of single processor systems has been thoroughly analyzed with respect to worst case response time and feasibility conditions [30, 42]. The assignment of the transmission channel on the basis of priorities is also supported by the native CAN network arbitration protocol [12], and the worst case transmission latency of real-time CAN messages (with timing constraints) has been analyzed and discussed in past research works [74]. Also, the OSEK operating system standard for automotive applications [52], supports not only priority scheduling, but also an implementation of the immediate priority ceiling protocol [72] for sharing resource with predictable worst case blocking time, and of preemption threshold and non-preemptive groups [79].

End-to-end deadlines have been discussed in research works in the context of single-processor as well as in distributed architectures. The synthesis of the task parameters (activation rates and offsets) and (partly) of the task configuration itself in order to guarantee end-to-end dead-

lines in single processor applications is discussed in [26]. Later, the work has been tentatively extended to distributed systems [66].

Similarly, other research works have been dedicated to finding a slotted allocation model that can provide a non-overlapping execution window to all the tasks that are involved in a distributed end-to-end computation with precedence constraints [20]. All these models require clock synchronization among all nodes.

End-to-end latencies in a distributed system can be analyzed quite easily in the worst case in the case of a periodic activation model with asynchronous communication, because the end-to-end schedulability problem can easily be decomposed in local instances of the problem, one for each resource (CPU or network). This is not true in the case of event-driven activation models, where local schedulers have cross dependencies because of the propagation of the activation signals. In this case, the problem of distributed hard real-time analysis has been first addressed by the holistic model [75] based on the propagation of the release jitter along the computation path. The holistic analysis has been later adapted to different types of networks, such as TDMA [57]. The response time analysis of distributed end-to-end transactions with offsets among the activation instants of the tasks in the transaction chains is discussed in [53]. The authors demonstrated that exact analysis is of exponential complexity and provided a solution for finding an upper bound to the worst case response time.

The two competing models of periodic activation with asynchronous communication and data-driven activation are reconciled by a new conceptual framework for the analysis of distributed chains of computations, based on network calculus [13] and its application for evaluating the propagation of event models [18]. In [28] this model is used for distributed schedulability analysis,

where the system can be described as an arbitrary mix of data-driven and periodic asynchronous interaction models.

Other works, such as [68], focused on providing lock-free and wait-free communication mechanisms that ensure deterministic delays in the implementation of models integrating both event and time triggered subsystems. Later, the mechanism has not only been extended to EDF scheduled systems, but the authors also provided optimal (tight) bounds for buffer allocation in the implementation of Rate Transition blocks for many-to-one communication channels [78]. Optimization of buffer implementation is also the objective of [6].

Finally, the trade offs between a purely periodic activation model and an event-driven activation semantics are explored in [45] with respect to the composability of subsystems scheduled according to the two models.

While these works provide analysis procedures with increasing speed and precision, the synthesis problem is today largely open, except for [62], where the authors discuss the use of genetic algorithms for optimizing priority and period assignments with respect to a number of constraints, including end-to-end deadlines and jitter.

4.4.2 A Simple Example

Figure 4.4 represents a sample system consisting of 3 ECUs, one CAN bus, 8 tasks and 5 messages. Three computation paths are defined, ending respectively in tasks τ_5, τ_8 and τ_{13}. In the example, the task and messages have priorities, worst case execution times and periods as in the following table.

The last four columns of Table 4.1 explain the tradeoffs in the analysis between the two discussed activation models. In the case all the tasks and messages are activated periodically and

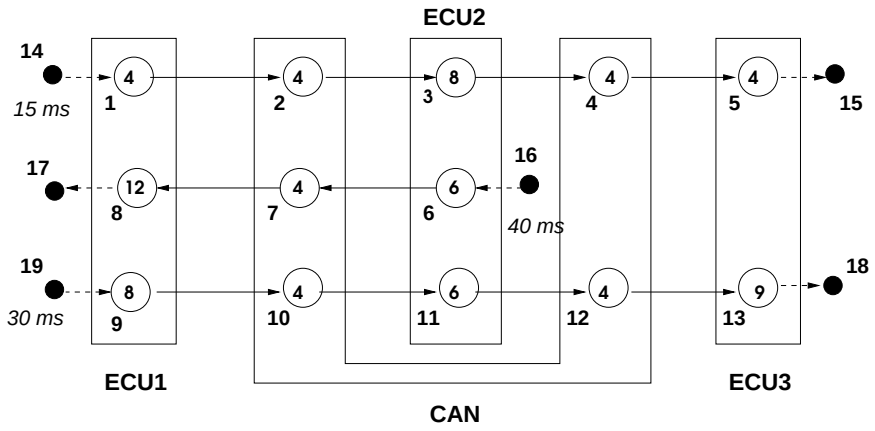


Figure 4.4: Example graph.

communicate by means of asynchronous buffers, the latencies for the three paths, assuming no sampling delay on the first task are, respectively, 100, 130, 260, as shown in the fourth to last column. If, however, the activation of the objects is always driven by the completion of their predecessor, with the exception of the first tasks in the paths (in our case τ_1, τ_6 and τ_9), then the latencies are much better for the highest priority paths, that is, 40 and 88, respectively, but are significantly larger for the lowest priority path ending in o_{18}, 312 time units from the triggering event. Although the jitter analysis is characterized by pessimism (relative offset information and best case response times are not considered in the analysis), the results clearly show the tradeoff between the two models and the opportunity for design optimization.

If the deadlines are defined as $d_{14,15} = 80$, $d_{16,17} = 120$ and $d_{18,19} = 280$, then in neither of the two cases, the deadlines can be guaranteed. However, if the activation model is defined in such a way that messages m_2, m_4 and m_7 are activated periodically and all the other objects on the paths are activated by the completion of their predecessors, then the worst case latencies are $l_{14,15} = 70$, $l_{16,17} = 100$ and $l_{18,19} = 208$, with all the deadline constraints satisfied.

Object	π_i	T_i	C_i	r_i	l_i	J_i	w_i	r_i
τ_1	1	15	4	4	4	0	4	4
m_2	2	15	4	8	27	4	8	12
τ_3	3	15	8	8	50	12	8	20
m_4	4	15	4	12	77	20	12	32
τ_5	5	15	4	8	**100**	32	8	**40**
τ_6	6	40	6	14	14	0	30	30
m_7	7	40	4	16	66	30	28	58
τ_8	8	40	12	20	**130**	58	30	**88**
τ_9	9	30	8	28	28	0	60	60
m_{10}	10	30	4	28	82	60	44	104
τ_{11}	11	30	6	28	140	104	60	164
m_{12}	12	30	4	28	198	164	88	252
τ_{13}	13	30	9	28	**260**	252	60	**312**

Table 4.1: Simple Example Data

Calculating the worst case response time of tasks and messages from equations (4.1), (4.2), (4.9), and (4.10) means solving a least fixed point equation. In some cases, the use of the exact formula is not practical and the problem may be tentatively approached by using linear upper and lower bounds for the response time of the first object instance in the critical instant hypothesis, (which is itself a lower bound of the real value) as in (4.6, 4.5.)

The question is of course to determine the amount of pessimism (and optimism) introduced by the previous linear approximations. The data of the example show that the linear approximations become progressively less accurate when the priority of the objects in the chain is

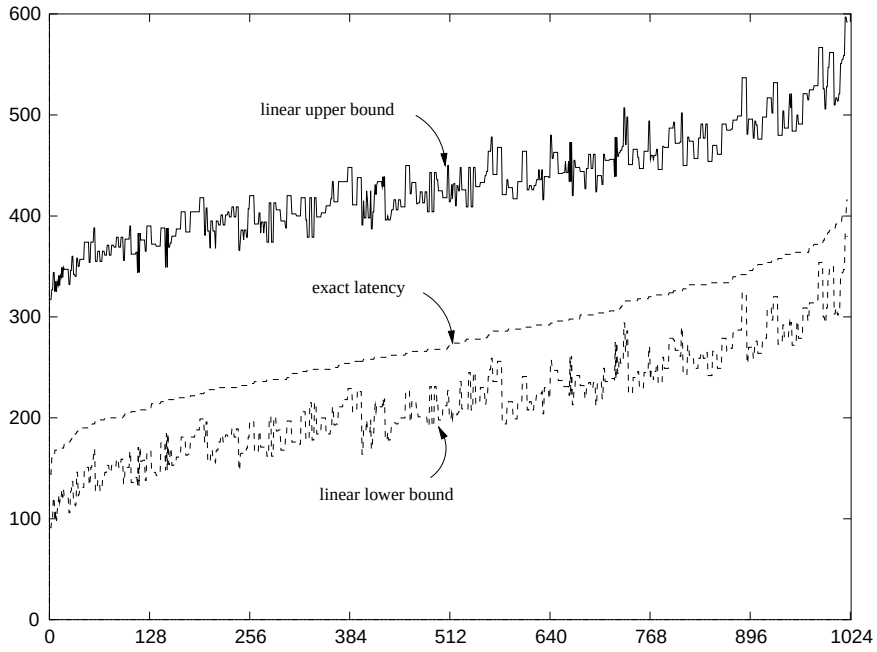


Figure 4.5: End-to-end latencies for the lowest priority path of the example.

lowered. For example, for the event-driven activation mode, the upper and lower bound latencies for the three paths are, respectively, $\{44.36, 130.86, 507.03\}$ and $\{38.91, 79.43, 294.96\}$. However, a linear combination of the linear upper and lower bounds can be sufficiently accurate to be used as an estimator of the actual end-to-end latencies on the paths.

The following Figure 4.5 shows the latencies on the lowest priority path in non-decreasing order, computed for all the possible configurations of the activation modes on the 10 links of the example. In the example, the worst case latency based on the exact analysis could be approximated fairly well by a linear combination of the pessimistic and optimistic linear bounds. The effectiveness of the linear approximation is illustrated in the following real design case of an automotive system architecture.

4.4.3 Optimization Framework for Synthesizing Activation Model

4.4.3.1 Mixed Integer Linear Programming

A mixed integer programming formulation can be used to find a solution to the synthesis problem with respect to a suitable cost function that accounts for deadline constraints on the paths. A short summary of the notations used in the ILP formulation is provided in the following.

Sets : ($\mathcal{V}$, $\mathcal{E}$, $\mathcal{P}$)

Variables: in addition to r_i, J_i, w_i, $L_{s,t}$ we define $y_{h,k}$ as

$$y_{h,k} = \begin{cases} 0 & \text{if the activation of } o_k \text{ is periodic,} \\ 1 & \text{if the activation of } o_k \text{ is event-driven by } o_h. \end{cases}$$

4.4.3.2 Feasibility Constraints

The feasibility constraints are modeled according to the rules for computing the jitter, the response times and the latencies at all nodes in the graph.

The jitter inheritance rule is encoded as follows. Consider a scheduled object o_k with multiple incoming link groups. I am only interested in those groups (links) that can possibly carry an activation signal (for all the other links $l_{j,k}$ it is clearly $y_{j,k} = 0$).

I enforce the condition that all the links in one group assume the same activation model. This means that

$$y_{r,k} = y_{s,k} \tag{4.14}$$

for all the pairs $l_{r,k}, l_{s,k}$ belonging to the same group lg_h.

If o_k has more than one incoming link group, only one of the group representatives can provide its activation signal. For each object o_k it must be ($rep(lg_h)$ denotes the group representative defined in Section 4.1.2)

$$\sum_{lg_h \in G(o_k)} y_{r,k} \leq 1 \text{ where } o_r = rep(lg_h).$$

If all group links have a periodic activation mode (all $y_{r,k} = 0$) then o_k is activated periodically and $J_k = 0$. Otherwise, J_k will be equal to the response time of the representative object in the group from which it gets the activation signal. The two alternative ways of computing J_k can be encoded in a pair of constraint sets leveraging a typical formulation in use in integer linear programming.

A very large constant value M is used to nullify one or more constraints by making them always true depending on the value of a set of binary variables ($y_{r,k}$ in our case).

$$J_k \leq \sum_{lg_h \in G(o_k)} y_{r,k} \times M \text{ where } o_r = rep(lg_h) \tag{4.15}$$

$$0 \leq J_k \tag{4.16}$$

If all $y_{r,k} = 0$, then the first and the last inequality constrain the value of J_k to 0. If $y_{r,k} = 1$ for one of the incoming link groups, then the first inequality is redundant and the following two set of constraints (a pair for each group $lg_h \in G(o_k)$) make J_k equal to the worst case response time r_r of the predecessor object o_r that is the representative of the activating group.

$$J_k \leq r_r + (1 - y_{r,k}) \times M \text{ where } o_r = rep(lg_h) \tag{4.17}$$

$$r_r - (1 - y_{r,k}) \times M \leq J_k \text{ where } o_r = rep(lg_h) \tag{4.18}$$

If o_k has only one incoming link from object o_h that can possibly provide an activation signal, then a simpler set of constraints apply

$$r_h + (y_{h,k} - 1) \times M \leq J_k \tag{4.19}$$

$$J_k \leq r_h \tag{4.20}$$

$$J_k \leq y_{h,k} \times M \tag{4.21}$$

$$0 \leq J_k \tag{4.22}$$

The worst case response time r_h for object o_h can be computed as

$$r_h = w_h + J_h$$

Because of the non-linearity and even non convexity of the fixed point formula that provides the exact value of w_h, a linear combination of the linear upper (4.6) and lower bounds (4.5) is used.

$$w_h = C_h + \alpha * \sum_{o_k \in hp(o_h)} (\frac{w_h + J_k}{T_k} + 1)C_k + (1 - \alpha) * \sum_{o_k \in hp(o_h)} (\frac{w_h + J_k}{T_k})C_k \tag{4.23}$$

where α is chosen as to minimize the following mean square fit function, computed for all $y = 0$ and assuming α does not depend significantly on the value of the y variables, as suggested by Figure 4.5.

$$\sum_{P_r \in \mathcal{P}} (\alpha * L^{\uparrow}_{P_r} + (1-\alpha) * L^{\downarrow}_{P_r} - L_{P_r})^2 \tag{4.24}$$

where $L^{\uparrow}_{P_r}$ and $L^{\downarrow}_{P_r}$ are the latencies computed on the path P_r using the upper and the lower linear bound respectively.

Finally, for computing the end-to-end latencies, a variable $z_{i,j}$ is defined for each link $l_{i,j}$ to express the link contribution to the end-to-end latencies of all the paths containing it. The variable $z_{i,j}$ is equal to w_j if the link $l_{i,j}$ carries an activation event (first two of the above constraints.) Otherwise, $z_{i,j}$ will be equal to $w_j + J_j + T_j$, considering the fact that o_j may be activated by some other signal with release jitter J_j. Hence, the contribution to the latency depends on the value of $y_{i,j}$ and the usual formulation is used to express the alternative.

$$w_j \leq z_{i,j} \tag{4.25}$$

$$z_{i,j} \leq w_j + (1 - y_{i,j}) \times M \tag{4.26}$$

$$z_{i,j} \leq w_j + J_j + T_j \tag{4.27}$$

$$w_j + J_j + T_j - y_{i,j} \times M \leq z_{i,j} \tag{4.28}$$

The end-to-end latency $L_{s,t}$ associated with path $P_{s,t}$ is computed as

$$L_{s,t} = \sum_{l_{u,v} \in \mathcal{E}_{p_{s,t}}} z_{u,v}$$

and should not exceeds its deadline.

$$L_{s,t} \leq d_{s,t}.$$

4.4.3.3 Objective Functions

Based on the above constraints, in addition to get a feasible solution, which satisfies the deadline constraints, there is a flexibility to get the optimal solution with respect to different cost functions. The minimization of the number of buffers could be used for reducing cost through the following objective function.

$$maximize \sum_{(o_h,o_k)\in\mathcal{E}} y_{o_h,o_k}$$

Other interesting cost functions are the sum of the end-to-end latencies, or the sum of difference between the end-to-end latency of each path and the corresponding deadline over all the paths in the system. The second objective function may be defined to assign a penalty for the violation of a specific path deadline through a weight w_{p_r}.

$$\sum_{p_r\in\mathcal{P}} L_{p_r} \qquad \sum_{p_r\in\mathcal{P}} w_{p_r} * Max(L_{p_r} - d_{p_r}, 0)$$

4.4.3.4 Optimization of the example graph

For the example in Figure 4.4, I used the objective functions defined in the previous section. The results are shown in the following table where $P_1 = o_{14} \rightarrow o_{15}$, $P_2 = o_{16} \rightarrow o_{17}$, $P_3 = o_{18} \rightarrow o_{19}$ and the objective functions are $F_1 =$ minimization of the number of event buffers, $F_2 =$ minimization of the sum of the path latencies, $F_3 =$ minimization of the sum of weighted lateness for all the paths exceeding the deadline and $F_4 =$ minimization of the lowest priority path latency.

Objective	P_1	P_2	P_3	periodic objects	event objects
F_1	55	84	304	m_4	remainings
F_2	70	58	266	τ_3, m_4, τ_{10}	remainings
F_3	55	112	236	m_2, m_7	remainings
F_4	70	98	168	τ_3, m_4, τ_8	$m_2, \tau_5, m_7, \tau_{10}, \tau_{11}, m_{12}, \tau_{13}$

4.4.4 Heuristic Framework for Synthesizing Activation Model

4.4.4.1 Heuristics

The example in 4.4.2 shows how the application of the two activation models, that is, all objects are activated periodically, or all objects are activated based on an event-driven activation model, results in a system that may be infeasible for a given problem. To find a schedulable solution, a search procedure can be defined that starts from the endpoint in which all objects are executed periodically, and tries to construct a feasible model by selectively changing at each step the activation model of one of the objects to be event-driven (see Figure 4.6).

At first, any incoming link to the lowest priority object on each resource can be set to carry an activation event, in case it is the only incoming link ($l_{10,11}$, $l_{11,12}$ and $l_{12,13}$ in the example). Given that the activated objects have the lowest priority, this change can only improve the latencies on the system paths.

Then, the search procedure is defined as in Algorithm 1.

The search algorithm mainly operates on two lists containing, respectively, the objects in the system and the paths for which a deadline is defined. At each step, the algorithm identifies the

Algorithm 1 Search Algorithm

```
1:  procedure MainSearch
2:      OL ← Object List, PL ← Path List
3:      InitDataStructures(OL, PL, A, BM, Bm)
4:      α ← ComputeAlpha(OL, A, BM, Bm)
5:      optval ← EvalSolution(OL, PL, A, BM, Bm, α)
6:      RecursivePart(OL, PL, A, BM, Bm, α, optval)
7:  end procedure
8:
9:  procedure RecursivePart(OL, PL, A, BM, Bm, α, optval)          ▷ recursive part
10:     curval ← EvalSolution(OL, PL, A, BM, Bm, alpha)
11:     if curval ¡ optval then
12:         optval ← curval
13:     end if
14:     if FeasibleSolution(OL, PL) then
15:         SolutionFound(OL, PL)
16:     end if
17:     Children ← ListChildren(PL)
18:     for each link in Children do
19:         link.SetActivation(EVENT, A, BM, Bm)
20:         val ← EvalSolution(OL, PL, A, BM, Bm, α)
21:         SortedChildren.Insert(link, val)
22:         link.SetActivation(PERIODIC, A, BM, Bm)
23:     end for
24:     for each link in SortedChildren do
25:         link.SetActivation(EVENT, A, BM, Bm)
26:         RecursivePart(OL, PL, A, BM, Bm, α, optval)
27:         link.SetActivation(PERIODIC, A, BM, Bm)
28:     end for
29: end procedure
```

list of the candidate children of the current node (line 17 in the recursive part). This list consists of a set of links that do not carry an activation event but may be changed to define an event-driven activation of their sink node. These links are selected in turn. For each of them, the result of the change to event-driven activation is estimated (lines 18 to 23) and based on the estimated value of a cost function, they are sorted in a list. Finally, the children are visited depth-first, in order of the metric function, until a solution is found (lines 24 to 28 and line 15). When a solution is found, the algorithm can simply store the result and exit, or it can try to continue the search looking for further improvements. In the latter case, if the function call at line 15 does not terminate the program, the algorithm results in an exhaustive search of the tree. The exact definition of the search algorithm depends on the method for generating and evaluating new children solutions and, especially, on the metric function that is used to evaluate the children and define their search order.

Selection of the children nodes The first policy that may be used for the selection of the children nodes is a very simple greedy policy: at each stage the algorithm computes the *critical path*, that is the path with the largest lateness (difference between the end-to-end latency and the deadline). The children nodes are all the links in the critical path for which a change to event-driven activation is possible. The second option consists of considering as children all the links in the system graph and not only those belonging to the critical path.

Evaluation of the children nodes Two main cost functions have been considered for the evaluation of the children nodes. The first cost function is the maximum lateness among all the system paths. The child with minimum cost is evaluated first.

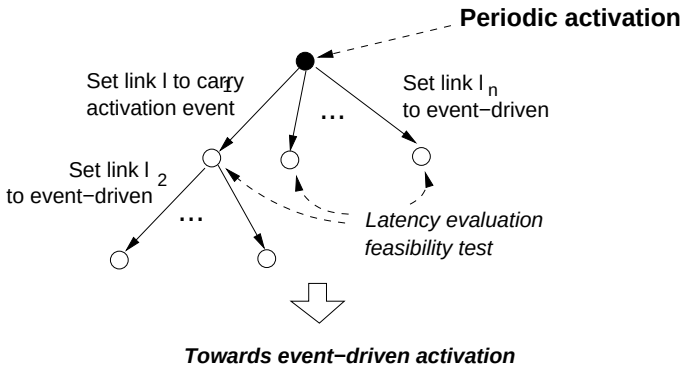


Figure 4.6: Search tree.

$$\max_{P_{i,j} \in \mathcal{P}} (L_{i,j} - d_{i,j})$$

The second function consists of computing the sum of all the path latencies or, as a variant, the weighted sum of all the positive differences between latencies and deadlines.

Node solutions can be evaluated by computing the solution to the fixed point formula for the worst case response times and by propagating the response time as the jitter of the following object, in case this is driven by the completion of the predecessor (as in [75]). Similarly, the children nodes are evaluated by setting the corresponding link to carry an activation event and then evaluating the corresponding solution.

As shown in the case study, the system complexity can easily reach the point in which there are hundreds of objects and the links subject to optimization are approximately two hundred. In this case, the breadth and width of the search tree may be too large to compute the end-to-end latencies by calculating the worst case response time of tasks and messages from equations (4.1) and (4.9), that require solving a least fixed point equation and then by repeating these computations in the jitter propagation cycle, waiting for the jitter values to converge to a fixed point solution. In this case,

a simpler approximate method is required to evaluate the effect of changing the activation model of one link on the end-to-end latencies, and possibly also an approximate formula for speeding up the computation of the end-to-end latencies at intermediate nodes.

Previous equations (4.8, 4.11) express r_i as a linear combination of the activation jitter of o_i and of the higher priority objects o_j. If o_i provides the activation signal for o_j, then $J_j = r_i$ and, therefore, the following linear function expresses the dependencies among the activation jitters with a data driven activation model.

$$\begin{bmatrix} J_1 \\ J_2 \\ \dots \\ J_n \end{bmatrix} = \begin{bmatrix} a_{1,1} & \dots & a_{1,n} \\ a_{2,1} & \dots & a_{2,n} \\ \dots & \dots & \dots \\ a_{n,1} & \dots & a_{n,n} \end{bmatrix} \begin{bmatrix} J_1 \\ J_2 \\ \dots \\ J_n \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_n \end{bmatrix}$$

where

$$\begin{cases} a_{i,j} = 1 & o_j \rightarrow o_i \\ a_{i,j} = \dfrac{u_j}{1 - \displaystyle\sum_{l \in hp(k)} u_l} & o_k \rightarrow o_i \wedge \pi_j \geq \pi_k \wedge R_{o_j} = R_{o_k} \\ 0 & \text{otherwise} \end{cases}$$

and

$$
\begin{cases}
b_i = \dfrac{C_j + \alpha \displaystyle\sum_{k \in hp(j)} C_k}{1 - \displaystyle\sum_{k \in hp(j)} u_k} & o_j \rightarrow o_i \wedge \pi_k \geq \pi_j \\
& \wedge R_{o_j} = R_{o_k} \\
& \wedge o_j \text{ is a task} \\
b_i = C_j + \dfrac{B_j + \alpha \displaystyle\sum_{k \in hp(j)} C_k}{1 - \displaystyle\sum_{k \in hp(j)} u_k} & o_j \rightarrow o_i \wedge \pi_k \geq \pi_j \\
& \wedge R_{o_j} = R_{o_k} \\
& \wedge o_j \text{ is a message} \\
0 & \text{otherwise}
\end{cases}
$$

With link groups, only the representative object must be considered in the definition of the values $a_{i,j}$, b_i of the matrices.

Hence, the problem can be formalized as a fixed point problem

$$\mathbf{J} = \mathbf{A}\mathbf{J} + \mathbf{B}$$

and has solution

$$\mathbf{J} = (\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} \tag{4.29}$$

The value of α is computed as the coefficient which makes the linear combination of the upper and lower bound linear approximations equal to the exact latency value for the path with the maximum lateness.

The condition $\sum_i u_i < 1$ for all the resources in the system is provably sufficient for the existence of a valid jitter solution to the previous set of linear dependencies. When this is not true, either $(\mathbf{I} - \mathbf{A})^{-1}$ does not exist or negative jitter values will result from (4.29).

The complexity of the exact response time analysis, and the following jitter propagation is pseudo-polynomial (exponential in the number of tasks and messages in the worst case), while the approximate computation allows computing the activation jitter of all the tasks with matrix operations (difference, inversion and multiplication) on $\mathbf{A}$ and $\mathbf{B}$ of complexity $O(n^3)$. In reality, considering that $\mathbf{A}$ is a sparse matrix, there are numeric packages capable of performing these operations in a much shorter time. Following the evaluation of $\mathbf{J}$, the computation of the end-to-end latencies on the paths can be obtained by applying equation (4.3) where the w_k terms can be computed for each object by using a linear approximation of the interference from high priority objects (from (4.8) and (4.11)).

In the algorithm, three matrices are used: $\mathbf{A}$ (which is independent of α), $\mathbf{BM}$, that is $\mathbf{B}$ computed for $\alpha = 1$, and $\mathbf{Bm}$, that is $\mathbf{B}$ for $\alpha = 0$. α can be computed only once, at the beginning of the search, or it can be updated before expanding a child node. Since the branching factor is expected to be large, the exact evaluation of the child to be expanded is not affecting significantly the speed of the algorithm.

At the beginning of our search algorithm, all objects are activated periodically, meaning that no link carries an activation signal. Therefore, the initial definition of $\mathbf{A}$, $\mathbf{BM}$ and $\mathbf{Bm}$ consists of all zeros. As the search progresses, and new links are defined as carrying activation signals, the definition of the matrices is updated accordingly.

Visiting a new child A visit to a new child consists in setting an event-driven activation signal associated to the selected link. This means that the activation of the sink object changes to event-driven and new values for the path latencies can be computed or estimated. The child solution becomes the new expansion point for generating new children and continuing the search, until a

feasible solution is possibly found or there is no further possible improvement.

In the data structures, the definition of a link as the carrier of an activation event means that the method for propagating the release times to jitters changes in the exact analysis and the definition of the matrices $\mathbf{A}$, $\mathbf{BM}$ and $\mathbf{Bm}$ needs to be updated by setting all the values that are affected by the new definition of the activation event on the selected link.

4.4.4.2 Optimization of the example graph

The texts show the result of the application of the search algorithm on the example presented in the previous Section 4.4.2 when children nodes are extracted from the critical path and the lateness of the critical path is used as the cost function. At initialization, all objects are defined as periodically activated, except for those singleton links that are incoming to lowest priority objects, which carry an activation event. After the evaluation of the root node, the path ending in o_{17} is found to be the critical path, with a maximum lateness of 32 units (Figure 4.7 shows the steps in the optimization of the sample graph.)

There are two possible children for the root node, corresponding to the two links in the critical path. After the evaluation of the children cost, the lowest lateness for the critical path is 22 units, when the activation of message m_7 is set to event driven (if τ_8 were to be activated by $l_{7,8}$, then the path ending in o_{19} would become the critical path, with a maximum lateness of 42 units). A child node is generated by setting m_7 to be activated by the completion of τ_6, with latencies for the three paths computed at, respectively, 92, 92 and 134.

After the second step, the search algorithm selects message m_2 to be activated by an event on the link $l_{1,2}$. The new latencies become 77, 92 and 138, with the third one exceeding the deadline by 18 units. The next two possible activation events are selected on node m_{10}, and m_4 respectively,

with new latencies values {77,92,110} and, finally, {62,92,114}, which is a feasible solution.

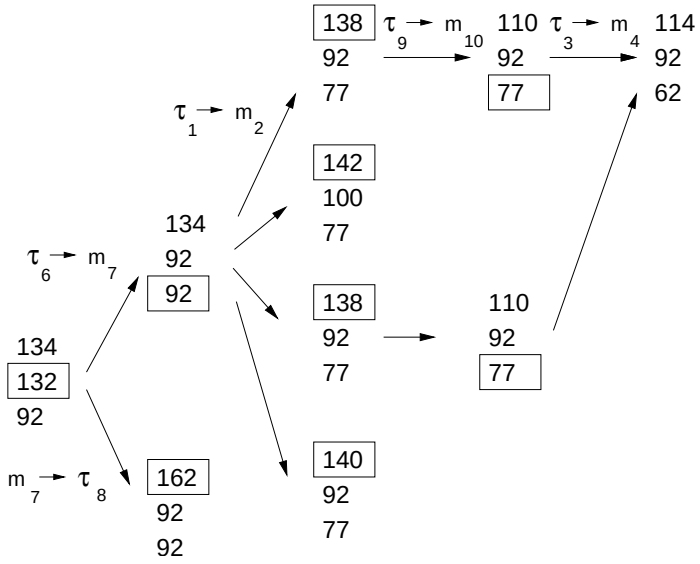


Figure 4.7: Steps of the search algorithm for the example.

4.4.5 An Industrial Example

The proposed approaches in Section 4.4.3 and 4.4.4 to the synthesis of the task activation model have been applied to the architecture configuration of an experimental vehicle under development at General Motors.

For the approach in Section 4.4.3:

The architecture consists of 38 nodes connected by 6 CAN buses. A total number of 100 tasks are executed on the ECU nodes and 322 messages are exchanged over the six buses. Ten pairs of endpoints have been identified in the graph as sources and destinations of computation paths with deadlines. An analysis of the graph found 184 paths between these 10 pairs of nodes and deadlines ranging from 100 ms to 300 ms have been defined for them.

The analysis of the graph with a periodic activation of all the tasks and messages found

end-to-end latencies largely exceeding (in the worst case) the desired deadlines. For example, a worst case latency of 627ms was found for paths with deadline 300 and 302.83 for paths with deadline 100.

After the application of the optimization the results were much closer to the desired deadlines, but still not feasible for 12 of the 148 paths. It was necessary to change the period of one more task (from 12.5 to 10) and one more message (from 100 to 80), making it shorter so that an event driven activation mode could be defined on the corresponding incoming and outgoing links.

After another optimization round, all the latencies became lower than the deadlines, with the largest value of 265 for paths with deadline 300, 190 for paths with deadline 200 and 97 for paths with deadline 100.

The value of alpha changed from 0.239 to 0.224 and when repeating the optimization with the new value, the same result was obtained.

For the approach in Section 4.4.4:

The architecture consists of 38 nodes connected by 6 CAN buses, with speeds from 25kb/s to 500kb/s. The vehicle supports advanced distributed functions with end-to-end computations collecting data from 360° sensors to the actuators, consisting of the throttle, brake and steering subsystems and of advanced HMI (Human-Machine Interface) devices. A total number of 100 tasks are executed on the ECU nodes, supporting from 1 to 22 tasks each, and 322 messages are exchanged over the six buses, with a minimum and maximum number of messages of, respectively, 32 and 145 for each bus. Worst case execution time estimates have been obtained for all the tasks. The number of links in the dataflow graph is 507. Bus utilizations are between 30% and 50% and CPU utilizations are estimated between 5% and 60%. Unfortunately, because of IP protection, the

exact layout of the architecture cannot be shown.

Ten pairs of endpoints have been identified in the graph as sources and destinations of computation paths with deadlines. An analysis of the graph found 184 paths between these 10 pairs of nodes. Deadlines ranging from 100 ms to 300 ms have been defined for them.

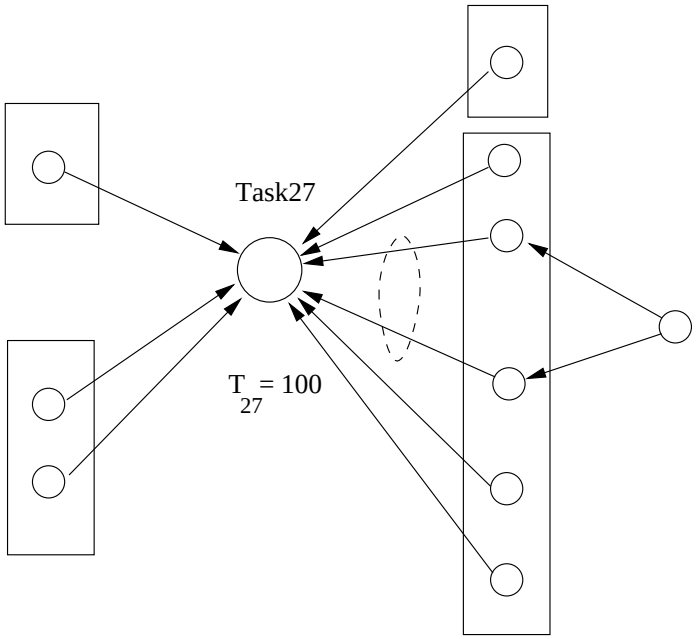


Figure 4.8: Activation options for one of the tasks in the case study.

Figure 4.8 shows a snapshot of one of the system tasks with some of its connections and gives an idea of the connectivity of the graph for the case study and also of the possible options for the activation of one of the graph tasks. In this case, the task receives nine messages that satisfy the conditions for an event-driven activation. These messages come from four of the system buses and eight possible link groups originate from them. The search algorithm must select which of the eight groups (if any) should carry the activation signal for the task in the optimal configuration.

If all tasks and messages are activated periodically, the end-to-end latencies in the system

largely exceed (in the worst case) the desired deadlines. A worst-case latency of 577ms is found for paths with deadline 300, 255.5 for paths with deadline 200, and 145.38 for paths with deadline 100. Of the 507 links, 313 are subject to optimization, including the link groups.

In its original formulation, the problem does not have a solution, because of a path in which most of the links are constrained to be periodic. After reducing the period of one of the messages in the path from 100 to 50 ms, the problem admits a solution.

The first algorithm option, which generates children nodes from the path with the largest lateness, was able to find the solution after changing 24 link groups to event-driven activation, with the largest latency being 294.802 ms for paths with a 300 ms deadline, 158.091 ms for paths with a 200 ms deadline and 95.46 for the only path with a 100ms deadline. The sum of the differences between latencies and deadlines is -11329.6 at this stage (-61.57 on average on all paths). After finding a feasible solution, the sum of the latencies was used as a metric to further increase the quality of the solution and the algorithm found, after 5 extra branches, a new solution with cost -14129.6 (-76.79 for the average difference between latencies and deadlines).

The time required to solve the problem using the exact computation of the latencies was approximately 2.6 seconds (1.4 GHz PC). In all cases, the program was stopped whenever a feasible solution was found or no solution was found after 2 hours of computation.

When trying different options for the selection of the children nodes and for the cost function it is not difficult to find out that the quality of the results obtained for the case study depends heavily on the selection of the cost metric function, but not on the possible use of the linear approximation in the evaluation of the latencies. The use of the cost function consisting of the sum of the latencies (or even the weighted sum of the positive differences between latencies

and deadlines) did not allow finding a feasible solution, but the program ended in a local optimum after 29 iteration steps. Backtracking could not solve the problem in a reasonable time, given the extremely high branching factor.

When the approximate linear evaluation was used in place of the exact computation of the response times, with a value of $\alpha = 0.465$ computed for the initial configuration, a feasible solution was obtained in the case the largest lateness was used as a metric (regardless of the children selection method). When repeating the optimization procedure by recomputing the value of α at each step, the same result was obtained. Of course, a solution was not found when using the weighted sum of the path latencies as the cost metric function. However, while the linear approximation proved to be a quite accurate estimator for the fitness of the children solutions, it did not result in any savings in the running time of the algorithm, but actually in a larger computation time (approx. 7 seconds). The cause for this (partly) unexpected behavior for our case study are the relatively short number of tasks and messages in each path (always less than 10) and especially the limited degree of concurrency at the nodes, which make the solution of the fixed point equations and the evaluation of the jitter propagation quite fast in practice. However, a higher degree of concurrency and even higher complexity is expected for future architectures, for which the linear approximation may be significantly faster than the exact fixed point analysis.

Schedulability theory provides support for the analysis of the worst case latencies in distributed computations when the architecture of the system is known and the communication and synchronization mechanisms have been defined. In the design of complex automotive system, however, a great benefit of schedulability analysis may come from its use as an aid in the exploration of the software architecture configurations that can best support the target application. This chapter

presented both an ILP formulation and a heuristic based optimization framework that leverages the trade-offs between the purely periodic and the data-driven activation models to meet the latency requirements of distributed vehicle functions on a CAN architecture. The effectiveness is demonstrated on a complex automotive example.

Chapter 5

Mapping with Scheduling for Automotive System

To provide design-time guarantees on timing constraints, different design and scheduling methodologies can be used. Because of resource efficiency, most automotive controls are designed based on run-time priority-based scheduling of tasks and messages. Examples of standards supporting this scheduling model are the OSEK operating system standard and the CAN bus arbitration model as mentioned in previous section.

In the typical model that is used for the implementation of distributed computations, periodic tasks and messages communicate according to a semantics in which the communication channel holds the last value that is written into it and is implemented as a shared variable protected against concurrent access. This model, called *periodic activation model*, has some advantages, including the separation of concerns when evaluating the schedulability of the individual resources. It also allows for a very simple specification at the interface of each subsystem or component, thereby

simplifying the interaction with the suppliers. The drawback is a non-deterministic time behavior and a possibly large worst-case end-to-end delay in the computations.

The execution model considered in this chapter is the following. Input data (generated by a sensor, for instance) are available at one of the system's ECUs. A periodic activation event from a local clock triggers an application task on this ECU. The task reads the input data *signal*, computes intermediate results as output *signals*, and writes them to the output buffer from where they can be read by another task or used for assembling the data content of a message. Messages - also periodically activated - transfer the data from the output buffer on the current ECU over the bus to an input buffer on another ECU. Eventually, task outputs are sent to a system output (an actuator, for instance). The application typically imposes end-to-end latency requirements between a subset of the source-sink task pairs in the system.

The complexity and physical distribution of modern active safety, chassis and powertrain automotive applications requires the use of distributed architectures. Complex functions designed as networks of function blocks exchanging signal information are deployed onto the physical HW and implemented in a SW architecture consisting of a set of tasks and messages. The typical configuration features priority-based scheduling of tasks and messages and imposes end-to-end deadlines. This chapter optimizes the task placement (task mapping) and the signal to message mapping and automates the assignment of priorities to tasks and messages in order to meet end-to-end deadline constraints and minimize latencies. This is again accomplished by leveraging worst case response time analysis within a mixed integer linear optimization framework. The approach is applied to an automotive case study to prove its feasibility.

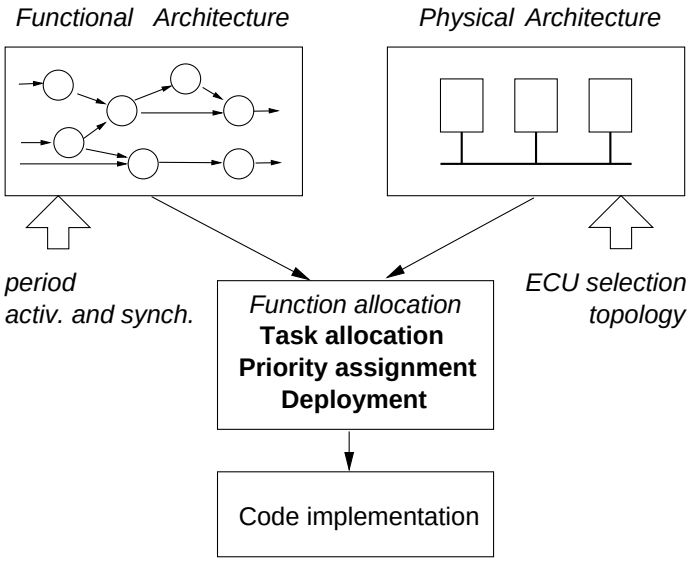


Figure 5.1: Design flow stages and optimization objectives (in bold)

5.1 Design Flow Revisited

The optimization of the placement of tasks, the definition of the mapping of signals into messages, and the priority assignment to tasks and messages are the design stages addressed in this work, as part of the larger design flow shown in Figure 5.1. The design flow is based on the Y-chart approach [37], where the application description and the architectural description are initially separated, and joined together later in an explicit mapping step. Previously, the step after the above is that giving the task/signal mapping and conduct the scheduling process, while in this chapter, I involve the mapping process to have a bigger design space to explore. In the application, nodes represent function blocks and edges represent data dependencies, which consist of signal information. The application description is further characterized by end-to-end latency constraints along selected paths from sources to sinks. The architectural description is a topology consisting of ECUs connected with buses. In this work, I assume *heterogeneous ECUs*, which run a priority-

based, preemptive OSEK-compliant operating system. Furthermore, I target the case of a *single bus*, which uses the *standard CAN bus arbitration*, featuring non-preemptive priority-based message scheduling. In reality, automotive architectures feature heterogeneous computation nodes and the CAN communication standard represents the large majority of the communication links, with LIN connections being used for a relatively small number of local low speed data communications [19].

Mapping deploys functional blocks to tasks (this problem is not handled in this work) and tasks to ECUs. Correspondingly, signals can be mapped into local communication or messages that are exchanged over the buses. The mapping must be performed such that the end-to-end latency constraints from the application are satisfied. Within the mapping step are the operations of *task allocation, signal to message assignment, and priority assignment*.

5.2 System Modeling and Notations

Similar to previous chapters, the system (an example in Figure 5.2) consists of a *physical architecture*, in which m heterogeneous ECUs $E = \{e_1, e_2, ..., e_m\}$ are connected through a single Controller Area Network (CAN) bus, and a *logical architecture* in which n tasks belonging to the set $T = \{\tau_1, \tau_2, ..., \tau_n\}$ perform the distributed computations required by the functions. Signals $S = \{s_{i,j} | \tau_i, \tau_j \in T\}$ are exchanged among pairs of tasks. Each signal carries a variable amount of information (expressed as number of bits). $\beta_{s_{i,j}}$ is the length of the signal $s_{i,j}$. The signal exchanged between two tasks τ_i and τ_j is also represented as a directed link $\tau_i \rightarrow \tau_j$, so that the computation flow may be expressed as a directed graph. A *path* $P(\tau_i, \tau_j)$ or $P(i, j)$ is an ordered sequence $P = [\tau_i, \ldots, \tau_j]$ of tasks that, starting from from τ_i, reaches τ_j, going through $n + 1$ tasks such that each one of them receives a signal from its predecessor and sends information to its successor.

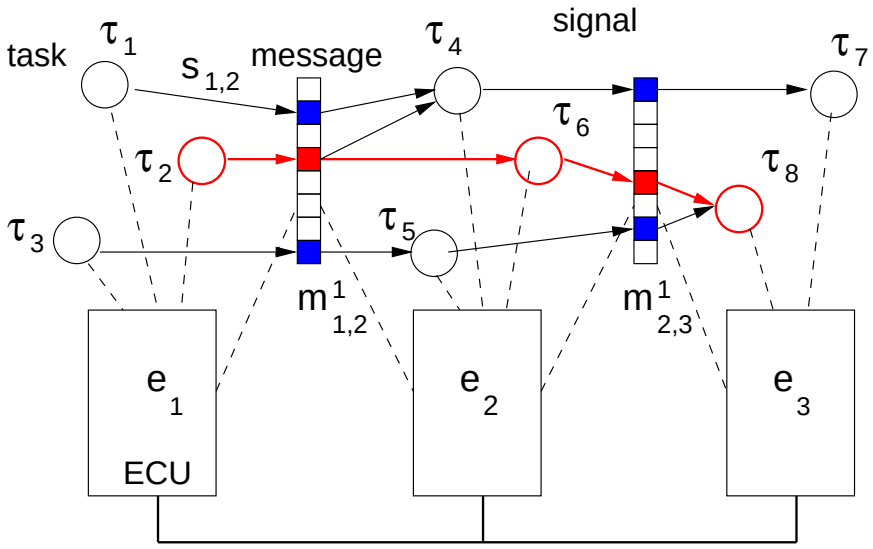


Figure 5.2: Mapping of tasks to ECUs and signals to messages.

A path represents one end-to-end execution of the system, from the production of a signal corresponding to an external event, to the generation of the output. More than one path can be originated by one initial task. The *path deadline* for $P_{i,j}$, denoted by $d_{i,j}$, is the end-to-end constraint for the computation performed in the path. Similarly, the worst case end-to-end latency for a computation spanning a path $P_{i,j}$ is denoted as $l_{i,j}$.

Tasks are executed on the ECUs and activated periodically. The placement of a task is indicated as a relation $A_{i,c}$ meaning that task τ_i is executed on e_c. The period of τ_i is indicated as T_i. At the end of their execution, tasks produce their output signal, which inherits the period of the sender task. I allow the system ECUs to be of heterogeneous nature, but I assume that the worst case computation time of each task τ_i on each ECU e_c is known or can be estimated as $c_{i,c}$.

After the mapping of the tasks to the ECUs, the signals are mapped into messages exchanged between ECU pairs. $M = \{m^r_{p,q} | e_p, e_q \in E, r = 1...u^{max}_{p,q}\}$ is the message set. All

messages are periodic, with period $T_{m^r_{p,q}}$ and are scheduled according to their priority $p^r_{p,q}$ on the CAN bus (as defined by the standard) The mapping rules require that each signal mapped to a message must have the same source and destination ECUs (its transmitter and receiver tasks must be allocated on the source and destination ECU of the message) and the same period of the message.

In CAN, the message size is limited to a maximum of 64 bits. Hence there is the possibility that a signal larger than 64 bits is fragmented and transmitted in multiple messages. In this approach, I don't consider signal fragmentation, but I assume that the length of each signal always allows it to be transmitted in a single message. The designer may perform an a-priori fragmentation of larger signals to fit this model.

5.3 Optimization on Task Placement and Signal Mapping

5.3.1 Previous Work

[65] Both static and dynamic priority, distributed as well as centralized scheduling methods have been proposed in the past for distributed systems. Static and centralized scheduling is typical of time triggered design methodologies, like the Time-Triggered Architecture (TTA) [39] and its network protocol TTP and of implementations of synchronous reactive models, including Esterel and Lustre [8]. Also, the recent Flexray standard [25] for high speed communication in automotive systems, provides two transmission windows, one dedicated to time-driven periodic streams with static, design-time assignment of transmission slots, and the other for asynchronous, event-driven communication.

Priority based scheduling is also very popular in control applications. It is supported by the native CAN network arbitration protocol [12]. The response times of real-time CAN messages

(with timing constraints) have been analyzed and discussed in past research work [74] where a formula for computing the worst case response time was presented. The formula has recently been found to be flawed and corrected [19]. Also, the OSEK operating system standard for automotive applications [52], supports not only priority scheduling, but also an implementation of the immediate priority ceiling protocol [72] for sharing resources with predictable worst case blocking time. Priority-based scheduling of single processor systems has been thoroughly analyzed with respect to worst case response time and feasibility conditions [30, 42].

End-to-end deadlines have been discussed in research work in the context of single-processor as well as in distributed architectures. The synthesis of the task parameters (activation rates and offsets) and (partly) of the task configuration itself in order to guarantee end-to-end deadlines in single processor applications is discussed in [26]. Later, the work has been tentatively extended to distributed systems [66].

The periodic activation model with asynchronous communication can be analyzed quite easily in the worst case, because it allows the decomposition of the end-to-end schedulability problem in local instances of the problem, one for each resource (CPU or network). This is not true in the case of data-driven activation models, where local schedulers have cross dependencies because of the propagation of the activation signals. In this case, the problem of distributed hard real-time analysis has been first addressed by the holistic model [75, 57] based on the propagation of the release jitter along the computation path. Other methods have been defined for scheduling periodic tasks and messages in a distributed time-triggered architectures. One early proposal can be found in [63]

While these works provide analysis procedures with reduced pessimism, increasing speed

and precision, the synthesis problem is largely open, except for [62], where the authors discuss the use of genetic algorithms for optimizing priority and period assignments with respect to a number of constraints, including end-to-end deadlines and jitter. In [10], the authors describe a procedure for period assignment on priority-scheduled single-processor systems. In [58] a design optimization heuristics-based algorithm for mixed time-triggered and event-triggered systems is proposed. The algorithm, however, assumes that nodes are synchronized and the bus transmission time is allocated according to the Universal Communication Model.

In [46], a SAT-based approach for task and message placement was proposed. Like my approach, the method provides optimal solutions to the placement and priority assignment. However, it did not consider signal packing.

The problem of optimal packing of periodic signals into CAN frames when the transmission of signals is subject to deadline constraints and the optimization metric is the minimization of the bus utilization has been proven to be NP-hard in [67]. Commercial (the middleware tool by Volcano [16]) and research solutions [65, 67] exist to this problem. However, they are all based on the assumption that the designer already allocated the tasks to the ECUs and partitioned the end-to-end deadlines into task and message deadlines.

5.3.2 Objective and Formulation

5.3.2.1 Preprocessing

Mapping problem involves allocating a finite number of elements from one space to another space which also contains a finite number of containers. The tasks to ECUs mapping problem, designers know concretely about the ECU entities. However, the signals to messages mapping does

not make it clear what the message entities are. I want to set an upper bound number of messages between every ECU pairs so that I know the concrete containers when conduct the signals packing.

However, how to set priorities to the packed messages could be a hard problem without knowing the detail information about the signals packed into the message. Two approaches are presented for conquering this difficulties.

Firstly, a procedure which interleaves priority synthesis and allocation synthesis could be designed to iteratively solve the problem. For example, the allocation optimization engine takes the priority value assigned from a previous priority optimization step as input, and generates outputs for the next round priority optimization. The re-packing of signals for the allocation optimization step could probably introduce re-work for the following priority optimization engine.

5.3.2.2 Resource scheduling

The worst case response time for a periodic task τ_i, as discussed in previous chapter, in a generic preemptive and priority based scheduled system is given by:

$$\begin{aligned} & r_i(q) = (q+1)C_i + \sum_{j \in hp(i)} \left\lceil \frac{r_i(q)}{T_j} \right\rceil C_j \\ & r_i = \max_q \{r_i(q) - qT_i\} \\ & \text{for all } q = 0 \ldots q^* \text{ until } r_i(q^*) \leq T_i \end{aligned} \tag{5.1}$$

where $j \in hp(i)$ spans over all the tasks with higher priority executed on the same CPU as τ_i. The need of evaluating the first q instances inside the busy period is caused by the uncertainty about the instance which causes the worst case response time. A lower bound on the worst case response time can be obtained by restricting the computation to the first instance. This bound is tight in case

$r_i \leq T_i$.

$$r_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{r_i}{T_j} \right\rceil C_j \tag{5.2}$$

Message objects are transmitted over a CAN bus. The evaluation of the worst-case latency for the messages follows the same rules for the worst-case response time of the tasks, with the exception that an additional blocking term B_i must be included in the formula in order to account for the non preemptability of CAN frames and that the transmission time of the message cannot be preempted. An upper bound of the response time is obtained [19] when the blocking term B_i for a generic message m_i is approximated with the largest possible frame transmission time ($w_i > 0$ is the queuing delay part of r_i, without the transmission time).

$$\begin{aligned} & w_i(q) = B_i + qC_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i(q)}{T_j} \right\rceil C_j \\ & r_i = \max_q \{C_i + w_i(q) - qT_i\} \\ & \text{for all } q = 0 \ldots q^* \text{ until } r_i(q^*) \leq T_i. \end{aligned} \tag{5.3}$$

A lower bound on w_i and r_i can be computed by only considering the first instance ($q = 0$). Once again, the bound is tight if $r_i \leq T_i$.

5.3.2.3 Periodic activation model

In the periodic activation model, the worst case end-to-end latency is computed for each path by adding the worst case response times and the periods of all the objects in the path ($r_k = w_k$).

$$L_{(i,j)} = \sum_{k:o_k \in P(i,j)} (T_k + r_k)$$

Due to unsynchronized timers, in the worst case, at each step, the input signal datum arrives right after the completion of an instance of the receiving task τ_i, executed with minimum

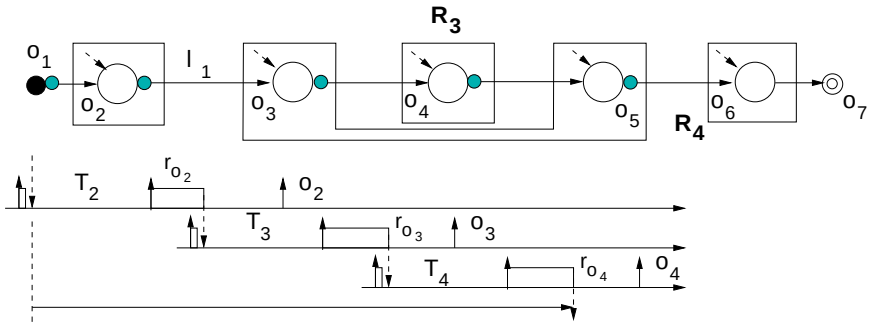


Figure 5.3: Periodic activation model.

(negligible) response time. The signal data will be read by the next instance of the task and the result will be produced after its worst case response time, that is, $T_i + r_i$ time units after the arrival of the input signal. The same reasoning applies to the execution of all the tasks and messages in the path.

5.3.2.4 Objective and Formulation

The objective of this design problem is to find the best possible

- Placement of tasks onto the CPUs
- Packing of signals to messages
- Assignment of priorities to tasks and messages

Given

- Constraints on (some) end-to-end latencies
- Constraints on the message size

with respect to the

- minimization of a set of end-to-end latencies

The problem is formualated in the general framework of mathematical programming (MP), where the system is represented with parameters, decision variables, and constraints over the parameters and decision variables. An objective function, defined over the same set of variables, characterizes the optimal solution. The problem allows a mixed integer linear programming (MILP) formulation that is amenable to automatic processing. After [15], a MILP program in standard form is:

$$\text{minimize} \quad c^T x \tag{5.4}$$

$$\text{subject to} \quad Ax = b \tag{5.5}$$

$$x \geq 0 \tag{5.6}$$

where $x = (x_1, ..., x_n)$ is a vector of positive real or integer-valued decision variables. A is an $m \times n$ full-rank constant matrix, with $m < n$, b and c are constant vectors with dimention n*1. Constraints of the type $Ax \leq b$ can be handled by adding a suitable set of variables, and then transforming such inequalities in the standard form. MILPs can be solved very efficiently by a variety of solvers.

The main difficulty of a MILP approach lies in the possible large number of variables and constraints and the resulting large solution time. The form of the constraints and objective function must be chosen carefully such that the formulation captures the behavior of the system, and yet remains amenable to efficient solving.

Preface to the optimization problem definition Task allocation, signal packing and priority optimization are managed at the same time by the MILP framework. However, while the tasks are

mapped into the ECUs and the number and type of ECUs is known at problem definition time, the number, period and priority of the messages exchanged by the ECUs is unknown in advance and results from the number and type of the signals that need to be exchanged among ECUs, which depend, in turn, from the task allocation. The problem formulation requires representing messages by a suitable set of variables in order to define the signal to message mapping constraints and the latency associated to each message (and signal).

Therefore, I bound the number of messages that can be possibly exchanged between any ECU pair and we define a corresponding number of message placeholders acting as possible signal containers. $u_{p,q}^{max}$ is the upper bound for the number of messages $m_{p,q}^{r}$ between ECU pair e_p and e_q ($r \in 1..u_{p,q}^{max}$). Of course, this means that I need a preprocessing procedure to determine such upper bound for the number of messages between every ECU pair. Furthermore, one set of such messages is needed for each possible period.

Because of the large number of sets, variables and constraints and, ultimately, for sake of clarity, in the following sections I explain the optimization constraints, each section referring to a specific aspect of the problem.

5.3.3 Task to ECU mapping

Sets and Variables

Based on the problem characterization, the following binary variables are defined

$$A_{i,j} = \begin{cases} 1 & \text{if } \tau_i \text{ is mapped to ECU } e_j, \\ 0 & \text{otherwise.} \end{cases}$$

$$a_{i,j} = \begin{cases} 1 & \text{if } \tau_i \text{ and } \tau_j \text{ are mapped to the same ECU,} \\ 0 & \text{otherwise.} \end{cases}$$

Feasibility Constraints Each task can be mapped to at most one ECU (N constraints)

$$\sum_{j \in E} A_{i,j} = 1 \tag{5.7}$$

Furthermore, there are dependencies among the $A_{i,j}$ and the $a_{i,j}$ variables. If tasks τ_i and τ_j are mapped to the same ECU e_k, then (5.8) constrains the variable $a_{i,j} = 1$. However, if task τ_i and τ_j are mapped to different ECUs, then (5.9) will set $a_{\tau_i,\tau_j} = 0$.

$$A_{i,k} + A_{j,k} - 1 \leq a_{i,j} \tag{5.8}$$

$$2 - A_{i,p} - A_{j,q} \geq a_{i,j} \tag{5.9}$$

5.3.4 Signal to message mapping

Sets and Variables

$$\alpha_{s_{i,j},m^r_{p,q}} = \begin{cases} 1 & \text{if } s_{i,j} \text{ is mapped to } m^r_{p,q}, \\ 0 & \text{otherwise.} \end{cases}$$

$$\gamma_{s_{i,j},m^r_{p,q}} = \begin{cases} 1 & \text{if } s_{i,j} \text{ adds to the length of } m^r_{p,q}, \\ 0 & \text{otherwise.} \end{cases}$$

$\gamma_{s_{i,j},m^r_{p,q}}$ provides additional information with respect to $\alpha_{s_{i,j},m^r_{p,q}}$ in the case of multicast signals, that is, signals that have multiple receivers (Figure 5.4). In this case, even though the model defines one signal for each pair sender-receiver, there is no need to copy the signal multiple times into a message, since CAN messages are broadcast and all remote tasks can read the signal value from the

message. $\gamma_{s_{i,j},m^r_{p,q}}$ is used to nullify multiple copies of such multicast signals. Finally, since the

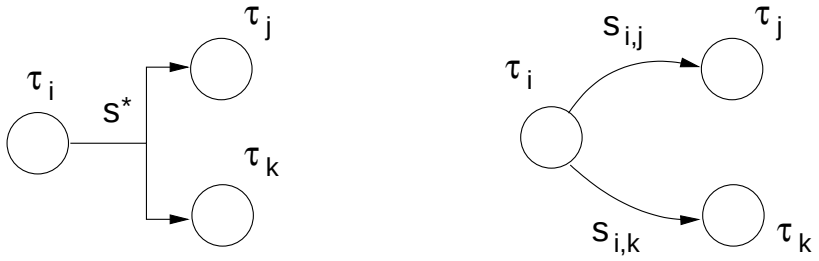


Figure 5.4: Multicast signals and their representation.

messages are actually placeholders, and some of them may be empty after the signal mapping stage, we need an additional set of variables

$$Y_{m^r_{p,q}} = \begin{cases} 1 & \text{if message } m^r_{p,q} \text{ is non-empty,} \\ 0 & \text{otherwise.} \end{cases}$$

to avoid considering those messages in the scheduling and response time computation stage.

Mapping Constraints Each signal must be mapped to at most one message (or not mapped to any, when communication is local, (5.10))

$$\sum_{r \le u^{max}_{p,q}, p,q \in E} \alpha_{s_{i,j},m^r_{p,q}} \le 1 \tag{5.10}$$

Signal $s_{i,j}$ is exchanged between task τ_i and τ_j. If tasks τ_i and τ_j are mapped to ECUs e_p and e_q, respectively, ($p \neq q$), then the signal must be mapped to one of the messages between e_p and e_q (5.11).

$$A_{i,p} + A_{j,q} - 1 \le \sum_{r \le u^{max}_{p,q}} \alpha_{s_{i,j},m^r_{p,q}} \tag{5.11}$$

If tasks τ_i and τ_j are on the same ECU, then all the mapping variables must be 0 for this signal (5.12).

$$\alpha_{s_{i,j},m^r_{p,q}} \leq 1 - a_{i,j} \tag{5.12}$$

If the successors of τ_i are mapped to the same ECU, and this is different from the one hosting τ_i, then (5.13) guarantees that only one of the multicast output signals contributes to message $m^r_{p,q}$. The set $succ_i$ includes the successors of τ_i, which receive the same multicast signal. Information about these successors sets is available at design time. The message length in bit $\beta_{m^r_{p,q}}$ is computed by adding up the bits of all the signals mapped into it. The linear bound assumes that this length is always a multiple of 8, as required by the standard.

$$\sum_{j \in succ_i} \gamma_{s_{i,j},m^r_{p,q}} = \alpha_{s_{i,j},m^r_{p,q}} \tag{5.13}$$

$$\sum_{s_{i,j} \in S} \gamma_{s_{i,j},m^r_{p,q}} \beta_{s_{i,j}} = \beta_{m^r_{p,q}} \leq 64 \tag{5.14}$$

Finally, I need to constrain the $Y_{m^r_{p,q}}$ variables that define if a message has at least one signal or is empty.

$$Y_{m^r_{p,q}} \geq \alpha_{s_{i,j},m^r_{p,q}} \qquad \forall s_{i,j} \in S \tag{5.15}$$

$$Y_{m^r_{p,q}} \leq 1 \tag{5.16}$$

Sets, Variables and Constraints For each pair of tasks (τ_i, τ_j), define

$$p_{i,j} = \begin{cases} 1 & \text{if task } \tau_i \text{ has higher priority than } \tau_j, \\ 0 & \text{otherwise.} \end{cases}$$

For the antisymmetric and transitive properties of the priority order relation, it must be (I assume no two tasks have the same priority level.)

$$p_{i,j} + p_{j,i} = 1 \tag{5.17}$$

$$p_{i,j} + p_{j,k} - 1 \leq p_{i,k} \tag{5.18}$$

The formula that allows to compute the worst case response time of a task τ_i is

$$r_i = C_i + \sum_{j \in hp(i)} I_{j,i} C_j$$

where $hp(i)$ spans over the set of all the higher priority tasks that are allocated on the same CPU as τ_i, and $I_{j,i}$ is the number of interferences of τ_j on τ_i during its response time.

$$I_{j,i} = \left\lceil \frac{r_i}{t_j} \right\rceil$$

To compute r_i in our MILP framework, I start by adding the following variable

$$y_{i,j} = \begin{cases} n \in \mathbb{N} & \text{number of possible interference of } \tau_j \text{ on } \tau_i, \\ 0 & \text{otherwise.} \end{cases}$$

The definition of the possible number of interferences as function of the response times and periods is captured by

$$0 \leq y_{i,k} - r_{\tau_i}/t_{\tau_k} \leq 1 \tag{5.19}$$

in addition, I define

$$x_{i,j} = \begin{cases} n \in \mathbb{N} & \text{number of possible interference of } \tau_j \text{ on } \tau_i \text{ if } p_{j,i} = 1, \\ 0 & \text{otherwise.} \end{cases}$$

$x_{i,j}$ can be defined in terms of $y_{i,j}$ and $p_{i,j}$ as follows, using the "'big M"' formulation (M is a large constant) in use in linear programming to express conditional constraints.

$$y_{i,k} - M \times (1 - p_{k,i}) \leq x_{i,k} \leq y_{i,k} \tag{5.20}$$

$$0 \leq x_{i,k} \leq M p_{k,i} \tag{5.21}$$

Furthermore, to take into account the placement condition, I need to define also

$$w_{i,j} = \begin{cases} n \in \mathbb{N} & \text{number of possible interferences of } \tau_k \text{ on } \tau_i \text{ if } p_{k,i} = 1 \text{ when on the same ECU,} \\ 0 & \text{otherwise.} \end{cases}$$

and

$$z_{i,j,k} = \begin{cases} n \in \mathbb{N} & \text{number of possible interferences of } \tau_k \text{ on } \tau_i \text{ if } p_{k,i} = 1 \text{ when they are on CPU } e_j, \\ 0 & \text{otherwise.} \end{cases}$$

Please note that $w_{i,k} \neq 0$ is the only case in which τ_k can actually preempt (i.e. interfere with) τ_i. An additional variable $z_{i,j,k}$ is used to put this information in the context of a given CPU (e_j) These variables can be computed from the previous ones as

$$x_{i,k} - M \times (1 - a_{i,k}) \leq w_{i,k} \leq x_{i,k} \tag{5.22}$$

$$0 \leq w_{i,k} \leq M a_{i,k} \tag{5.23}$$

for $w_{i,k}$, and

$$w_{i,k} - M \times (1 - A_{k,j}) \leq z_{i,j,k} \leq w_{i,k} \tag{5.24}$$

$$0 \leq z_{i,j,k} \leq M A_{k,j} \tag{5.25}$$

for $z_{i,j,k}$.

Finally, the response time of task τ_i (an additional variable $r_i \in \mathbb{R}^+$) can be computed as

$$r_i = \sum_j A_{i,j} c_{i,j} + \sum_k \sum_j z_{i,j,k} c_{k,j}. \tag{5.26}$$

5.3.4.1 Worst case response time of messages

Sets, Variables and Constraints For each pair of messages ($m^r_{p,q}$, $m^f_{s,t}$), we define

$$p_{m^r_{p,q}, m^f_{s,t}} = \begin{cases} 1 & \text{if } m^r_{p,q} \text{ has higher priority than } m^f_{s,t}, \\ 0 & \text{otherwise.} \end{cases}$$

For the antisymmetric and transitive properties of the priority order relation, it must be (we assume no two messages have the same priority level.)

$$p_{m^r_{p,q}, m^f_{s,t}} + p_{m^f_{s,t}, m^r_{p,q}} = 1 \tag{5.27}$$

$$p_{m^r_{p,q}, m^f_{s,t}} + p_{m^f_{s,t}, m^z_{x,y}} - 1 \leq p_{m^r_{p,q}, m^z_{x,y}} \tag{5.28}$$

As a result of the optimization, a priority level is assigned to all messages, including empty placeholders, and it is possible that a high priority level is assigned to one of the empty messages. An additional constraint ensures that this never happens. In (5.29) message $m^r_{p,q}$, when nonempty ($Y_{m^r_{p,q}} = 1$), has higher priority than the empty message $m^f_{s,t}$ ($Y_{m^f_{s,t}} = 0$).

This ensures that empty messages don't contribute to the interference of non-empty messages.

$$Y_{m^r_{p,q}} - Y_{m^f_{s,t}} \times M \leq p_{m^f_{s,t}, m^r_{p,q}} \tag{5.29}$$

The formula that allows to upper bound the worst case response time of a message $m^r_{p,q}$ is

$$r_{m^r_{p,q}} = C_{m^r_{p,q}} + q_{m^r_{p,q}}$$

$$q_{m^r_{p,q}} = B_i + \sum_{j \in hp(i)} I_{m^f_{s,t}, m^r_{p,q}} C_j$$

where $hp(i)$ spans over the indexes of all the messages with priority higher than $r_{m^r_{p,q}}$ and $I_{m^f_{s,t}, m^r_{p,q}}$ is the number of interferences of $m^f_{s,t}$ on $m^r_{p,q}$ during its queuing time. It is

$$I_{m^f_{s,t}, m^r_{p,q}} = \left\lceil \frac{q_{m^r_{p,q}}}{t^r_{p,q}} \right\rceil$$

Furthermore, the transmission times of messages depend upon their length according to

$$C_{m^r_{p,q}} = \sum_{s_{i,j} \in S} \frac{\beta_{m^r_{p,q}} + 46}{B_{speed}} = \sum_{s_{i,j} \in S} \frac{\beta_{m^r_{p,q}}}{B_{speed}} + \sum_{s_{i,j} \in S} \frac{46}{B_{speed}}$$

where $\beta_{m^r_{p,q}}$ is the size of the message (total number of bits mapped into it) and 46 is the number of protocol bits in a CAN frame.

Similar to the computation of the response times of the tasks, the additional variables $z_{m^r_{p,q}, m^f_{s,t}, s_{i,j}}$, $w_{m^r_{p,q}, m^f_{s,t}}$, $x_{m^r_{p,q}, m^f_{s,t}}$ and $y_{m^r_{p,q}, m^f_{s,t}}$ are defined.

The possible number of interferences $y_{m^r_{p,q}, m^f_{s,t}}$ is obtained from

$$0 \leq y_{m^r_{p,q}, m^f_{s,t}} - (r_{m^r_{p,q}} - \sum_{s_{i,j} \in S} \frac{\beta_{m^r_{p,q}} + 46}{B_{speed}}) / T_{m^f_{s,t}} \leq 1 \tag{5.30}$$

$x_{m^r_{p,q}, m^f_{s,t}}$, the possible number of interferences of a higher priority message $m^r_{p,q}$ on $m^f_{s,t}$ can be computed from $y_{m^r_{p,q}, m^f_{s,t}}$

$$y_{m^r_{p,q}, m^f_{s,t}} - (1 - p_{m^r_{p,q}, m^f_{s,t}}) \times M \leq x_{m^r_{p,q}, m^f_{s,t}}$$

$$x_{m^r_{p,q}, m^f_{s,t}} \leq y_{m^r_{p,q}, m^f_{s,t}}$$

$$0 \leq x_{m^r_{p,q}, m^f_{s,t}} \leq p_{m^r_{p,q}, m^f_{s,t}} \times M$$

$w_{m^r_{p,q},m^f_{s,t}}$ is computed from $x_{m^r_{p,q},m^f_{s,t}}$ and represents the number of interferences from a higher priority non-empty message $m^r_{p,q}$ (the only ones of practical relevance).

$$x_{m^r_{p,q},m^f_{s,t}} - (1 - Y_{m^f_{s,t}}) \times M \leq w_{m^r_{p,q},m^f_{s,t}}$$

$$w_{m^r_{p,q},m^f_{s,t}} \leq x_{m^r_{p,q},m^f_{s,t}}$$

$$0 \leq w_{m^r_{p,q},m^f_{s,t}} \leq Y_{m^f_{s,t}} \times M$$

and $z_{m^r_{p,q},m^f_{s,t},s_{i,j}}$ from $w_{m^r_{p,q},m^f_{s,t}}$ and $\gamma_{s_{i,j},m^r_{p,q}}$

$$w_{m^r_{p,q},m^f_{s,t}} - (1 - \gamma_{s_{i,j},m^r_{p,q}}) \times M \leq z_{m^r_{p,q},m^f_{s,t},s_{i,j}}$$

$$z_{m^r_{p,q},m^f_{s,t},s_{i,j}} \leq w_{m^r_{p,q},m^f_{s,t}}$$

$$0 \leq z_{m^r_{p,q},m^f_{s,t},s_{i,j}} \leq \gamma_{s_{i,j},m^r_{p,q}} \times M$$

Finally, (5.31) computes the bound for the worst case response time $r_{m^r_{p,q}} \in \mathbb{R}^+$ of message $m^r_{p,q}$ based on the number of interference from other messages

$$\begin{aligned} r_{m^r_{p,q}} \leq & \sum_{s_{i,j} \in S} \frac{\beta_{m^r_{p,q}} + 46}{B_{speed}} + \\ & b_{m^r_{p,q}} + \frac{1}{B_{speed}} \sum_{m^f_{s,t}} \sum_{s_{i,j}} z_{m^r_{p,q},m^f_{s,t},s_{i,j}} \beta_{s_{i,j}} \\ & + \frac{46}{B_{speed}} \sum_{m^f_{s,t}} w_{m^r_{p,q},m^f_{s,t}} \end{aligned} \tag{5.31}$$

and we ensure that $r_{m^r_{p,q}} > 0$ only when $Y_{m^r_{p,q}} \neq 0$.

$$\begin{aligned} & \sum_{s_{i,j} \in S} \frac{\beta_{m^r_{p,q}} + 46}{B_{speed}} + b_{m^r_{p,q}} + \\ & \frac{1}{B_{speed}} \sum_{m^f_{s,t}} \sum_{s_{i,j}} z_{m^r_{p,q},m^f_{s,t},s_{i,j}} \beta_{s_{i,j}} + \\ & \frac{46}{B_{speed}} \sum_{m^f_{s,t}} w_{m^r_{p,q},m^f_{s,t}} + (Y_{m^r_{p,q}} - 1) \times M \leq r_{m^r_{p,q}} \end{aligned}$$

$$r_{m^r_{p,q}} \leq Y_{m^r_{p,q}} \times M$$

Paths are defined on the tasks and the communication signals between them. the following constraints bond the latency of a message to the latency of all the signals that are mapped into it. The response time of a signal $s_{i,j}$ is denoted as $r_{s_{i,j}}$.

$$r_{m^r_{p,q}} - (1 - \alpha_{s_{i,j},m^r_{p,q}}) \times M \leq r_{s_{i,j}} \tag{5.32}$$

$$r_{s_{i,j}} \leq r_{m^r_{p,q}} + (1 - \alpha_{s_{i,j},m^r_{p,q}}) \times M \tag{5.33}$$

$$r_{s_{i,j}} \leq (1 - a_{i,j}) \times M \tag{5.34}$$

If τ_i and τ_j are mapped to different ECUs, (5.32) and (5.33) bonds the signal latency $r_{s_{i,j}}$ to the latency of message $r_{m^r_{p,q}}$. However, inequality (5.34) forces $r_{s_{i,j}}$ to 0 when the signal is local, that is, when τ_i and τ_j are mapped to the same ECU.

5.3.4.2 Worst case end-to-end latency

The end to end latency is computed as:

$$\sum_{\tau_i \in P_{l,m}} (r_r + T_i) + \sum_{l_{j,k} \in Link(P_{l,m})} (r_{s_{j,k}} + T_{s_{j,k}}) \leq d_{l,m} \tag{5.35}$$

where τ_i is the generic task in the path $P_{l,m}$. Latencies on the paths should be no greater than the deadline (5.35).

Objective Function Given that the performance of the functions is better with small response time of the actuators, the objective function minimizes the sum of the latencies over all paths

$$Min \sum_{P} (\sum_{\tau_i on P} r_{\tau_i} + \sum_{l_{j,k} \in Link(P)} r_{s_{\tau_j,\tau_k}}) \tag{5.36}$$

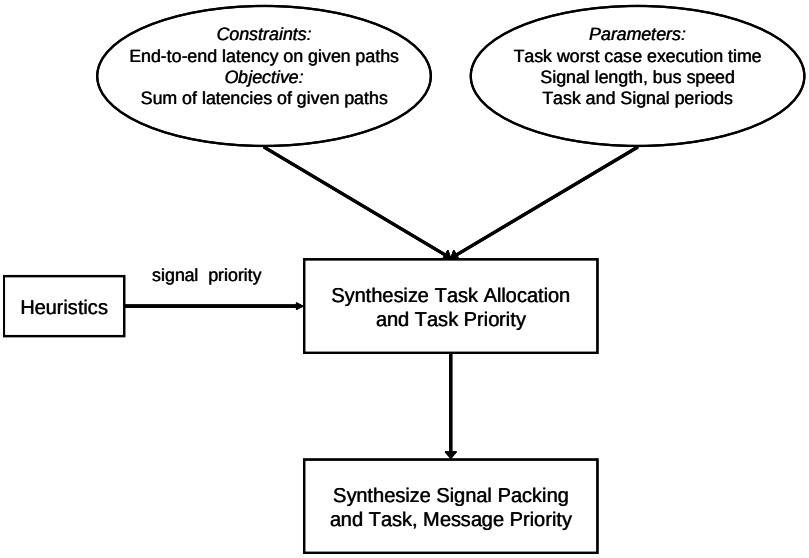


Figure 5.5: Two Step Synthesis Approach

5.3.5 Two Steps Synthesis Procedure

Section 5.3.2.4 describes an integrated formulation for task allocation, signal packing and task/message priority optimization. This problem formulation provides an optimal solution when solvable. However, the complexity is typically too high for the sizes of industrial applications. Therefore, I propose, as an approximation, a two-step synthesis method, as shown in Figure 5.5. The whole synthesis problem is divided into two sub-problems. At each step, the sub-problem is formulated as an MILP based on the variables and constraints defined in Section 5.3.2, then solved by mathematical programming tools. In Step 1, I assume one message is reserved for each signal, and that the priorities of one-signal messages are given by a preprocessing heuristic that assigns priorities to signals, based on their period, and according to the Rate Monotonic policy. In the first sub-problem, I synthesize the task allocation and task priority to optimize the sum of the latencies of given paths, while also satisfying the deadline constraints on those paths.

In Step 2, I use the task allocation result from Step 1, and synthesize signal packing, message priority and task priority. The objective is still to optimize the sum of the latencies of given paths, while satisfying the deadline constraints on paths and the constraints on message size.

5.4 Experimental Result

The work demonstrated the applicability and the possible benefits of the approach with a case study derived from the analysis of a bus subsystem of an experimental vehicle that incorporates advanced active safety functions.

The architecture platform consists of 9 ECUs connected with a single CAN-bus at speed 500kb/s. The vehicle supports advanced distributed functions with end-to-end computations collecting data from 360° sensors to the actuators, consisting of the throttle, brake and steering subsystems and of advanced HMI(Human-Machine Interface) devices.

The analysis focuses on the subset of tasks and signals that are part of paths with timing constraints. I assume the remaining tasks and signals are assigned lower priorities and allocated to ECUs and messages based on other considerations (possibly load balancing) in such a way that they do not interfere with the latencies of the critical paths.

For the purpose of our experiments, I assumed all ECUs to have the same computation power (which is not actually true in reality).

The subsystem that is the subject of our study consists of a total of 41 tasks executed on the ECU nodes, and 83 CAN signals exchanged among the tasks. Worst-case execution time estimates have been obtained for all tasks, and the bit length of the signals is between 1 (for binary information) and 64 (full CAN message).

End-to-end deadlines are placed over 10 pairs of source-sink tasks in the system. This corresponds to 171 paths. The deadline is set at 300ms for 8 source-sink pairs and 100ms for the other two.

The experiments were run on a 1.4-GHz processor with 2GB RAM. CPLEX 10.1 as the MILP solver is used in this experiment.

For step 1, the total number of variables was 21249, 3430 of them binary variables. The number of constraints (automatically generated by a purposely written C++ program based on the system configuration) was 801083. For step 2, the number of variables was 17797, 2582 of them binary variables. The number of constraints was 136221.

In Step 1, a feasible solution satisfying all path deadline constraints was found in 8.9 seconds. The objective value - sum of the latencies of given paths - was 36486ms for this feasible solution. Within 20000 seconds, the best solution found by the solver was 13060.3ms. Although the optimum had not been reached yet, the optimization was stopped with the obtained solution within 0.07% of the optimum for the formulation of Step 1. The largest latency among all the paths with deadline at 300ms was 135.82ms and the largest latency for 100ms deadline paths was found at 63.19ms.

In Step 2, I set the solution of Step 1 as the initial point and further optimized the objective function by packing the signals and synthesizing the priorities of tasks and messages. Within 20000 seconds, a solution with 12899.9 total latency was found. This was within 1.13% of the optimal for Step 2 formulation. The result improves the output of Step 1 by 1.22%. After this second step, the largest latency among all the paths with deadline at 300ms was 133.398ms and the largest latency for 100ms deadline paths was found at 62.09ms.

The improvement is small because message transmission times and response times are much smaller than task response times and both are small if compared with the task and message periods that contribute to the end-to-end latency. The majority of the path latencies were not significantly affected by the steps of signal packing and message priority optimization.

This chapter presented an integrated formulation (ILP based) for optimizing task placement, signal mapping as well as task and message priorities in distributed automotive systems to meet end-to-end deadline constraints and minimize latencies. I applied this method to an automotive case study, and showed it can affectively reduce the end-to-end latencies in a CAN based communication system.

Chapter 6

Conclusions and Future Work

In this thesis I proposed analysis and synthesis techniquesfor vehicle electronic system designers to make sure key functionalities finish before appropriate deadlines for safety-critical applications. . An integrated framework was presented for design space exploration using powerful mathematical programming to solve scheduling problems in a platform based design process.

System-level architecture design is neither a top-down nor a bottom-up design methodology. Rather, it is a ”meet-in-the-middle” approach. Chapter 2 presented a platform-based system-level design flow with an emphasis on scheduling and then explained the flow in automotive electronic design scenarios.

In chapter 3, extensibility and scalability metrics were captured in scheduling a hard real time embedded system, and implications were analyzed in accelerating time-to-market of a system development process by reducing development and re-verification burden in an incremental design flow. This was done by describing scheduling as a mathematical programming optimization problem, and solving it with respect to appropriate multi-objective cost functions derived from the

metrics. The metrics and the method were shown to be effective for industrial problems in a case study.

Chapter 4 presented two novel synthesis framework procedures (one is mathematical programming approach and the other one is a search algorithm) based on approximate timing analysis to optimize the definition of the activation model in the functional network with respect to latency constraints. The effectiveness of the approaches were demonstrated in the design of a complex real-world automotive architecture of a GM product. Besides the assignment of priorities to tasks and messages, or the definition of task and message periods, another possible objective for the synthesis of the software architecture is to find the optimal placement of the tasks on the ECUs. Some of these optimization problems were discussed in chapter 5.

Chapter 5 investigated an integrated formulation for optimizing task placement, signal mapping as well as task and message priorities in distributed automotive systems to meet end-to-end deadline constraints and minimize latencies. To make it practical for industrial applications, a two-step synthesis method approximating the integrated formulation was proposed to reduce complexity. Applying this method to an automotive case study showed that it can effectively reduce the end-to-end latencies.

In this thesis, large efforts were dedicated to an optimization framework for synthesizing scheduling table for time-triggered system, selecting activation models, allocating tasks to ECUs and mapping signals to messages, and assigning messages priorities for event-triggered system. There are, however, several other design optimization problems that could be addressed:

For time triggered system, the multiple ECU (processor) scheduling problem is an NP-hard problem. The mathematical programming approach is computationally intensive, and suitable

only for moderately sized problems. The case study required one hour of run time for the full extensibility and scalability cost functions. To deal with larger problem sizes, a heuristic is needed to obtain a feasible solution first, locks down all precedence binary variables, then use the same cost function to optimize. However, risks of being locked into a bad task ordering exist and algorithms to perturb a feasible ordering can be developed in order to move toward a better solution. In refining the bus model, one may describe and optimize the bus slot size and slot order with respect to utilization and buffer usage constraints, and optimize the bus cycle length to allow multiplex tasks in order to reduce buffer overhead.

For the event triggered system, although the techniques proposed in Chapter 5 targets only a part of the architecture definition stages, they are flexible enough for the designers to accommodate many constraints of interest, and effectively find solution of problems with practical size. Their applicability is however limited to a single bus subsystem at a time. The proposed technique should be merged with optimization results already devised for the assignment of the periods and the task and messages activation modes in the future, as detailed in previous chapters. Also, the possible extension to multi-bus subsystems (including the entire car architecture) and routing/gatewaying should be explored.

Bibliography

[1] Autosar. available at http://www.autosar.org, visited Sep 2007.

[2] Wikipedia. available at http://en.wikipedia.org, visited Sep 2007.

[3] International organization for standardization. Road vehicles-Controller Area Network (CAN) - Part 4: Time-triggered communication. ISO/DIS 11898-4, 2002.

[4] Sangiovanni Vincentelli A. Defining platform-based design. *EEDesign of EETimes*, Feb 2002.

[5] Lavagno L. Passerone C. Sangiovanni-Vincentelli A. Balarin F., Hsieh H. and Watanabe Y. Metropolis: An integrated environment for electronic system design. *IEEE Computer*, April 2003.

[6] M. Baleani, A. Ferrari, L. Mangeruca, and A. Sangiovanni Vincentelli. Efficient embedded software design with synchronous models. In *Proceedings of the 5th ACM EMSOFT conference*, 2005.

[7] A. Bender. Design of an optimal loosely coupled heterogeneous multiprocessor system. *Proc. Electronic Design and Test Conference*.

[8] A. Benveniste, P. Caspi, S.A. Edwards, N. Halbwachs, P. Le Guernic, and R. de Simone. The synchronous languages 12 years later. *Proceedings of the IEEE*, 91, January 2003.

[9] Griessbach R Berwanger J, Peller M. A new high performance data bus system for safety-related applications. available at http://www.byteflight.de, 2000, visited Sep 2007.

[10] Enrico Bini, Marco Di Natale, and Giorgio Buttazzo. Sensitivity analysis for fixed-priority real-time systems. In *Euromicro Conference on Real-Time Systems*, Dresden, Germany, June 2006.

[11] R. Bosch. Can specification, version 2.0, visited sep 2007. available at http://www.can-cia.org.

[12] R. Bosch. Can specification, version 2.0. Stuttgart, 1991.

[13] J.-Y. Le Boudec and P. Thiran. Network calculus - a theory of deterministic queuing systems for the internet. In *LNCS 2050, Springer*, 2001.

[14] S. Boyd and L. Vandenberghe. Convex optimization. Available at http://www.stanford.edu/ boyd/cvxbook.html.

[15] S. Boyd and L. Vandenberghe. Convex optimizations. Cambridge University Press, 2004.

[16] L. Casparsson, A. Rajnak, K. Tindell, and P. Malmberg. Volcano, a revolution in on-board communications. *Technical report, Volvo Technology report*, 1999.

[17] G. Cena and A. Valenzano. Performance analysis of byteflight networks. In *Proceedings of the IEEE International Workshop on Factory Communication Systems*, pages 157–166, 2004.

[18] Samarjit Chakraborty and Lothar Thiele. A new task model for streaming applications and

its schedulability analysis. In *IEEE Design Automation and Test in Europe (DATE)*, Munich, Germany, March 2005.

[19] Robert I. Davis, Alan Burns, Reinder J. Bril, and Johan J. Lukkien. Controller area network (can) schedulability analysis: Refuted, revisited and revised. *Real-Time Syst.*, 35(3):239–272, 2007.

[20] M. DiNatale and J. Stankovic. Dynamic end-toend guarantees in distributed real-time systems. In *Real-Time Systems Symposium*, Puerto Rico, 1994.

[21] Tomiyama H Takada H Ding S, Murakami N. A ga-based scheduling method for flexray systems. In *Proceedings of EMSOFT*, 2005.

[22] Wilfried Elmenreich. Introduction to ttp/c and ttp/a.

[23] Joaquim Ferreira, Paulo Pedreiras, Luís Almeida, and José Alberto Fonseca. The ftt-can protocol for flexibility in safety-critical systems. *IEEE Micro*, 22(4):46–55, 2002.

[24] Flexray. www.flexray.com.

[25] Flexray. Protocol specification v2.1 rev. a. available at http://www.flexray.com, 2006, visited Sep 2007.

[26] Richard Gerber, Seongsoo Hong, and Manas Saksena. Guaranteeing real-time requirements with resource-based calibration of periodic processes. *IEEE Transaction on Software Engineering*, 21(7):579–592, July 1995.

[27] OSEK group. Osekvdx: Time-triggered operating system specification 1.0. *www.osek-vdx.orgmirrorttos10.pdf*, July 2001.

[28] Arne Hamann, Rafik Henia, Marek Jerzak, Razvan Racu, Kai Richter, and Rolf Ernst. SymTA/S symbolic timing analysis for systems. available at http://www.symta.org, 2004.

[29] Ernst R Hamann A. Tdma time slot and turn optimization with evolutionary search techniques. In *Proceedings of the Design, Automation and Test in Europe Conference, vol 1, pp 312-317*, 2005.

[30] M. Gonzalez Harbour, M. Klein, and J. Lehoczky. Timing analysis for fixed-priority scheduling of hard real-time systems. *IEEE Transactions on Software Engineering*, 20(1), January 1994.

[31] Driscoll K Hoyme K. Safebus. IEEE Aerosp Electron Syst Mag 8(3):34-39.

[32] Rushby J. Bus architectures for safety-critical embedded systems. *Lecture notes in computer science*, 2211:306–323, 1.

[33] Shengbin Jiang. Flexray scheduler using window slicing techniques. GM Internal Report, 2005.

[34] K. Ramamritham John A. Stankovic. Advances in real-time systems. *IEEE Computer Society Press*, 1993.

[35] R. Newton A. Sangiovanni-Vincentelli K. Keutzer, Sharad Malik. System-level design: Orthogonalization of concerns and platform-based design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(12), Dec 2000.

[36] R. Newton J. Rabaey K. Keutzer, S. Malik and A. Sangiovanni-Vincentelli. System level

design: Orthogonalization of concerns and platform-based design. *IEEE Transactions on Computer-Aided Design of Circuits and Systems*, Vol. 19, No. 12, December 2000.

[37] Bart Kienhuis, Ed F. Deprettere, Pieter van der Wolf, and Kees A. Vissers. A methodology to design programmable embedded systems - the Y-chart approach. In Ed F. Deprettere, Jürgen Teich, and Stamatis Vassiliadis, editors, *Embedded Processor Design Challenges: Systems, Architectures, Modeling, and Simulation - SAMOS*, volume 2268 of *Lecture Notes in Computer Science*, pages 18–37. Springer, 2002.

[38] H. Kopetz. Real-time systems-design principles for distributed embedded applications. *Kluwer Academic Publishers*, 1997.

[39] H. Kopetz, A. Damm, C. Koza, M. Mulazzani, W. Schwabla, C. Senft, and R. Zainlinger. Distributed fault-tolerant real-time systems: The mars approach. *IEEE Micro*, 9(1), February 1989.

[40] Bauer G Kopetz H. The time-triggered architecture. Proc IEEE 91(1):112-126, 2001.

[41] H. Hansson K.W. Tindell and A.J. Wellings. Analysing real-time communications: Controller area network (can). In *Proc. 15 th RealTime Systems Symposium*, pages 259–263, 1994.

[42] J. W. S. Liu. *Real-Time Systems*. Prentice Hall, 2000.

[43] Layland Liu. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 1973.

[44] Mathworks. *The Mathworks Simulink and StateFlow User's Manuals*. web page: http://www.mathworks.com.

[45] Slobodan Matic and Tom Henzinger. Trading end-to-end latency for composability. In *Proceedings of the 26th IEEE Real-Time Systems Symposium*, 2005.

[46] Alexander Metzner and Christian Herde. Rtsat– an optimal and efficient approach to the task allocation problem in distributed architectures. In *RTSS '06: Proceedings of the 27th IEEE International Real-Time Systems Symposium*, pages 147–158, Washington, DC, USA, 2006. IEEE Computer Society.

[47] Pradyumna Mishra and Sanjeev Naik. Distributed control system development for flexray based systems. In *Proceedings of SAE Congress, Paper No. 2005-05AE-329*, Detroit, MI, 2004.

[48] B. T. Murray N. Kandasamy, J. P. Hayes. Dependable communication synthesis for distributed embedded systems. *Proc. 22nd Int'l Conf. on Computer Safety, Reliability, Security (SAFECOMP)*, 2003.

[49] Simont-Lion F Wilwert C Navet N, Song Y. Trends in automotive communication systems. *Proc IEEE*, 93(6):1204–1223, July 2005.

[50] OSEK. Trends in automotive communication systems, 2005.

[51] OSEK. Local interconnect network protocol specification. available at http://www.lin-subbus.org, 2005, visited Sep 2007.

[52] OSEK. Osek os version 2.2.3 specification. available at http://www.osek-vdx.org, 2006, visited Sep 2007.

[53] J.C. Palencia and M. Gonzáles Harbour. Schedulability analysis for tasks with static and

dynamic offsets. In *19th IEEE Real-Time Systems Symposium*, Madrid, Spain, December 1998.

[54] Thilo Demmeler Paolo Giusto. Rapid design exploration of safety-critical distributed automotive applications via virtual integration platforms. *The Journal of Systems and Software*, 70(3), March 2004.

[55] Pop Paul. Analysis and synthesis of communication-intensive heterogeneous real-time systems. *Ph. D. Thesis No. 833, Dept. of Computer and Information Science, Linkping University*, June 2003.

[56] M Plankensteiner. Development of modular electrical systems. *www.tttech.comtechnologydocsprotocol_comparisonsTTTech-Comparison_TTP-TTCAN-FlexRay.pdf*, 2003.

[57] Traian Pop, Petru Eles, and Zebo Peng. Holistic scheduling and analysis of mixed time/event-triggered distributed embedded systems. In *10th International Symposium on Hardware/Software Codesign (CODES 2002)*, pages 187–192, Estes Park, Colorado, USA, May 6-8 2002.

[58] Traian Pop, Petru Eles, and Zebo Peng. Design optimization of mixed time/event-triggered distributed embedded systems. In *CODES+ISSS '03: Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 83–89, New York, NY, USA, 2003. ACM Press.

[59] Peng Z Pop P, Eles P. Schedulability-driven communication synthesis for time-triggered embedded systems. *Real-Time Systems Journal*, 24:297–325, 2004.

[60] Peng Z Doboli A Pop P, Eles P. Scheduling with bus access optimization for distributed embedded systems. *EEE Trans VLSI Syst*, 8(5):472–491, 2000.

[61] Miner PS. Safebus. Analysis of the SPIDER fault-tolerance protocols, Proceedings of the 5th NASA Langley Formal Methods Workshop, 2000.

[62] Razvan Racu, Marek Jersak, and Rolf Ernst. Applying sensitivity analysis in real-time distributed systems. In *Proceedings of the 11th Real Time and Embedded Technology and Applications Symposium*, pages 160–169, San Francisco (CA), U.S.A., March 2005.

[63] Krithi Ramamritham, Gerhard Fohler, and Juan Manuel Adan. Issues in the static allocation and scheduling of complex periodic tasks. In *RTOSS '93: Proceedings of the tenth IEEE workshop on Real-time operating systems and software*, pages 11–16, Washington, DC, USA, 1993. IEEE Computer Society.

[64] Fabio Romeo. Magneti- marelli. DAC, Las Vegas, June 20th, 2001.

[65] R. Saket and N. Navet. Frame packing algorithms for automotive applications. *Journal of Embedded Computing, vol. 2, n 1*, pages 93–102, 2006.

[66] M. Saksena and S. Hong. Resource conscious design of distributed real-time systems – an end-to-end approach. In *Proc. IEEE Int'l Conf on Engineering of Complex Computer Systems*, 1996.

[67] K. Sandstrom, C. Norstom, and M. Ahlmark. Frame packing in real-time communication. *Seventh International Conference on Real-Time Computing Systems and Applications*, pages 399–403, 2000.

[68] N. Scaife and P. Caspi. Integrating model-based design and preemptive scheduling in mixed time- and event-triggered systems. In *6th Euromicro Conference on Real-Time Systems (ECRTS'04)*, July 2004.

[69] Kiran Seth, Aravindh Anantaraman, Frank Mueller, and Eric Rotenberg. FAST: Frequency-aware static timing analysis. In *Proceedings of the* 24^{th} *IEEE Real-Time Systems Symposium*, pages 40–51, Cancun, Mexico, December 2003.

[70] Danbing Seto, John P. Lehoczky, and Lui Sha. Task period selection and schedulability in real-time systems. In *Proceedings of the* 19^{th} *IEEE Real-Time Systems Symposium*, pages 188–198, Madrid, Spain, December 1998.

[71] Danbing Seto, John P. Lehoczky, Lui Sha, and Kang G. Shin. On task schedulability in real-time control systems. In *Proceedings of the* 17^{th} *IEEE Real-Time Systems Symposium*, pages 13–21, Washington, DC, December 1996.

[72] Lui Sha, Ragunathan Rajkumar, and John P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE transaction on computers*, 39(9), September 1990.

[73] D. B. Shmoys and Tardos. Scheduling unrelated machines with costs. In *Proc. 4th Ann. ACM-SIAM Symp. on Discrete Algorithms, ACM-SIAM*, pages 448–454, 1993.

[74] Ken Tindell, Alan Burns, and A. J. Wellings. Calculating controller area network (can) message response times. *Control Eng. Practice*, 3(8):1163–1169, 1995.

[75] Ken W. Tindell. Holistic schedulability analysis for distributed hard real-time systems. Technical Report YCS 197, Department of Computer Science, University of York, 1993.

[76] Clark J Tindell K. Holistic schedulability analysis for distributed hard real-time systems. *Microprocess. Microprogram*, 50:2–3, 1994.

[77] Wellings A Tindell K, Burns A. Calculating can message response times. *Control Engineering Practice*, 3(8):1163–1169, 1995.

[78] Stavros Tripakis, Christos Sofronis, Norman Scaife, and Paul Caspi. Semantics-preserving and memory-efficient implementation of inter-task communication on static-priority or edf schedulers. *Proceedings of the 5th ACM EMSOFT conference*, 2005.

[79] Y. Wang and M. Saksena. Scheduling fixed priority tasks with preemption threshold. In *Proceedings, IEEE International Conference on Real-Time Computing Systems and Applications*, December 1999.

[80] Niraj Shah William Plishker, Kaushik Ravindran and Kurt Keutzer. Automated task allocation on single chip, hardware multithreaded, multiprocessor systems. *Workshop on Embedded Parallel Architectures (WEPA-1)*, February.

[81] G.J. Zakarian, A. Rushton. Development of modular electrical systems. *IEEE/ASME Transactions on Mechatronics*, December 2001.

Appendix A

Alphabetic Notations

Throughout the dissertation, we use unified characters for notations goes as follows:

- τ: tasks running on ECU
- σ_{ij}: Signals communicated between tasks
- m_i: Messages transmitted on the bus
- b_{ij}: bit width for signal σ_{ij}
- δ_{ij}: unit delay for signal σ_{ij}
- T_i: The period of task τ_i
- Φ_i: The initial phase of task τ_i
- C_i: The worst case execution time of task τ_i
- l_i: The input/output connections between tasks
- t_i: An individual scheduling entity, job instance
- $t_k j$: The j-th job of task τ_k
- a_i: The arrival time of a task τ_i
- A_i: The release time of a task τ_i
- s_i: The start execution time of a task τ_i
- f_i: The finish time of a task τ_i
- d_i: The deadline of a task τ_i
- r_i: The worst case response time of a task τ_i
- $\mathcal{R}_i$: The response time of a job instance t_i

- $\mathcal{R}_{\tau_i}$: The worst case response time of a task τ_i
- H: Super period of all tasks in the functionality graph
- λ_{kj}: The j-th slot inside cycle k in a FlexRay system
- $P_{i,j}$: A *functional chain* or *Path* from τ_i to τ_j
- $\mathcal{EP}$: The set of end objects on the selected paths
- γ_j: The worst case copy time for the outputs data to be written into the input variable for the receiving task or the data trasmit register for the appropriate slot in the FlexRay adapter.
- Γ: The set of task graphs
- Γ': The set of task graphs after task graph expansion
- Γ': The set of task graphs after task graph expansion
- $\mathcal{C}$: The communication cycle set
- n_{slots}: number of slots per communication cycle
- l_{slot_min}: minimum slot size, determined by the length of the biggest signal sent by ECUs
- $rl_{message}$: The message ready list
- rl_{task}: The task ready list
- $\mathcal{E}$: The ECU list which is $\{e_i | i = 1...m\}$
- l_{comm}: The length of communication cycle of a FlexRay bus configuration
- l_{slot}: The slot size of a FlexRay bus configuration
- $s^s_{j,k}$: The starting time of the k^{th} slot from the j^{th} communication cycle of a FlexRay bus configuration
- $\mathcal{V}$: The set of vertices
- $\mathcal{E}$: The set of edges
- $\mathcal{R} = \{R_1, \ldots, R_z\}$: The set of shared resources supporting the execution of the tasks (CPUs) and the transmission of the messages (buses).
- R_{o_i}: The resource for a task/message needs to execute or for its transmission
- π_i: The priority of o_i and indexes are assigned by decreasing priority levels
- w_i: The worst case time spent from the instant the job is released with maximum jitter J_i to its completion or arrival.
- B_i: The blocking term for a message transmission on the bus

- $S = \{s_{i,j} | \tau_i, \tau_j \in T\}$: The signals are exchanged among pairs of tasks.
- $\beta_{s_{i,j}}$: The length of the signal $s_{i,j}$.