Chapter 2
Platform-Based Design for Automotive and Transportation Cyber-Physical Systems

Chung-Wei Lin, Bowen Zheng, Hengyi Liang, and Qi Zhu

Acronyms

ADAS	Advanced driver assistance systems
AUTOSAR	Automotive open system architecture
CACC	Cooperative adaptive cruise control
CAN	Controller area network
ECU	Electronic control units
OEM	Original equipment manufacturer
PBD	Platform-based design
TDMA	Time division multiple access
TSN	Time-sensitive networking
V2V	Vehicle-to-vehicle
V2X	Vehicle-to-X

C.-W. Lin (✉)
National Taiwan University, Taipei, Taiwan
e-mail: cwlin@csie.ntu.edu.tw

B. Zheng
University of California, Riverside, CA, USA

H. Liang · Q. Zhu
Northwestern University, Evanston, IL, USA

M. A. Al Faruque, A. Canedo (eds.), *Design Automation of Cyber-Physical Systems*, https://doi.org/10.1007/978-3-030-13050-3_2

2.1 Platform-Based Design Methodology for Connected Vehicles

Automotive design has become more complex than ever due to the rapid development of connected and autonomous technology. This trend affects not only the design of individual vehicles but also the operation of entire vehicular transportation system, through connected vehicle applications such as intelligent traffic signals, collaborative adaptive cruise control (CACC), and vehicle platooning. The safety-critical nature of these systems makes it essential to rigorously ensure functional correctness and to quantitatively evaluate system metrics throughout the design process and across all system layers. In this chapter, we will introduce the application of the platform-based design (PBD) paradigm in connected vehicles. We will present how the principles of the PBD paradigm, in particular the definition of platforms and the mapping between functional and architectural platforms, may be carried out across the system layers, from connected vehicle applications to individual vehicle functionality, and then to in-vehicle software, hardware, and physical layers.

2.1.1 Design Challenges for Connected Vehicles

In the following, we will first introduce some of the major challenges for connected vehicles, and then outline how the PBD paradigm may be applied to their design.

- **Addressing high-volume and dynamic input data:** The size of a signal in conventional control systems is usually not very large. It can be only a binary to indicate "on" or "off" of a component, or several bytes to represent the value of a measurement. However, for advanced driver assistance systems (ADAS) and autonomous functions in modern vehicles, the inputs from lidars, radars, cameras, and other sensors could induce much larger data at a high input data rate. For example, an advanced lidar can have input data rate that is up to 100 Mbps, which far exceeds the capacity of currently prevalent in-vehicle bus protocol, the controller area network (CAN), and the processing capability of current electronic control units (ECUs). Moreover, such input data rate may significantly vary under different road conditions, moving speed, and light intensity, which presents further challenges to the system design, as detailed below.
- **Computation architecture design:** High-volume and dynamic input data has a significant impact on the design of the computation platform. Should system designers add more ECUs or upgrade existing ECUs to more powerful ones for handling the data? What types of new computation elements such as GPUs, FPGAs, or ASIC accelerators are needed? Can the computation architecture be dynamically adapted to handle the changing data rate? Answering these questions requires the development of new design methodologies.

- **Communication architecture design:** To address the high-volume and dynamic input data, original equipment manufacturers (OEMs) have been exploring new in-vehicle communication architectures such as those based on the Ethernet protocol. However, systematic methodologies are still greatly needed to meet the data processing requirements. Furthermore, the new communication protocols, including both in-vehicle protocols and inter-vehicle protocols for vehicle-to-everything (V2X) communication, should be carefully designed and integrated with the conventional protocols that are still important for conventional/legacy components. The integration of different protocols also relies on the design and analysis of gateways, which further increase the design complexity.
- **Topology design:** As there are different protocols and multiple network devices in an automotive system, it is not trivial to decide the connection of sensors, actuators, and ECUs to network devices. The decisions are constrained by design requirements and affected by the trade-offs between performance, cost, and even wiring weight. Furthermore, the topology should follow the harness and routing graph in an automotive system and is often challenging to design.
- **Safety:** Automotive systems are safety-critical systems, and there are many constraints that have to be met for ensuring system safety. For instance, the end-to-end latency from detecting sensor input to applying control often has to meet a strict deadline, which requires rigorous worst-case analysis based on formal mathematical models. However, with the increase of functional and architectural complexity, accurately building those models and conducting worst-case analysis has become increasingly challenging.
- **Reliability:** The reliability of automotive systems relies on many factors, such as the fault-tolerant and redundant architectures for single-point-of-failures. Several protocols such as the time-sensitive networking (TSN) support replications and eliminations (if redundant at destinations) of frames. As shown in Fig. 2.1, these operations can increase the reliability of communication, but they also induce higher costs and more communication traffic. Furthermore, they require multiple routing paths, which makes topology design more challenging. Similarly, redundant ECUs may increase the reliability of computation, but they also lead to higher costs and design complexity.

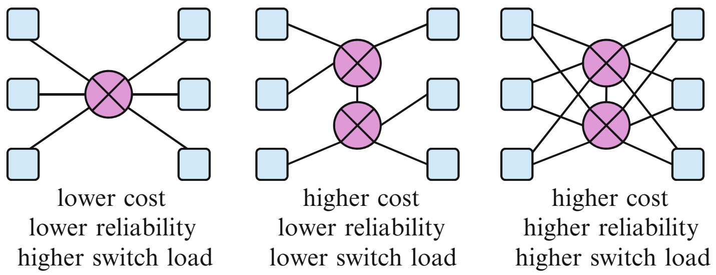

Fig. 2.1 The trade-off between cost, reliability, and switch performance in automotive design

- **Security:** System security is a rising issue for automotive systems. It requires a cross-layer solution that includes security mechanisms compatible with existing V2X communication protocols, lightweight security mechanisms within individual vehicles, and component-level security mechanisms. Due to tight resource constraints and stringent design requirements, security should be considered from the beginning of the design process; otherwise, it is often too late or impossible to add security at late stages.

2.1.2 Mapping Problems for Connected Vehicles

We propose the PDB methodology to address these growing design challenges of connected vehicles. The key idea of PBD is to capture the system with a number of abstraction layers called *platforms*, and divide the complex design process into a series of *mappings* from higher-layer to lower-layer platforms. The mapping between two platform layers is, in fact, a design space exploration process, where different options (abstracted as design variables) for implementing the high-layer platform model (i.e., "functionality") on the lower-layer platform components (i.e., "architecture") are explored with respect to a set of design objectives and constraints.

Figure 2.2 shows how the design of connected vehicles can be addressed with the PBD paradigm as a series of mapping problems across platform layers, including mapping connected vehicle applications to vehicle functionality, mapping vehicle functionality to software tasks, mapping software tasks to hardware components, and mapping hardware components to physical layout.

2.2 Mapping Connected Vehicle Applications to Vehicle Functionality

In the following sections, we will go through some representative problems for each of these mapping problems. At the top layer, the PBD paradigm is applied to the mapping from connected vehicle applications, such as cooperative adaptive cruise control (CACC), lane merging, and autonomous intersection, to functionality of individual vehicles. The mapping problem can be formulated as follows:

- **Platforms**: (1) The higher-layer platform is captured by the models of connected vehicle applications, such as CACC and autonomous intersections; and (2) the lower-layer platform includes the models of individual vehicles in both the cyber domain (computation and communication models) and the physical domain (vehicle dynamics).

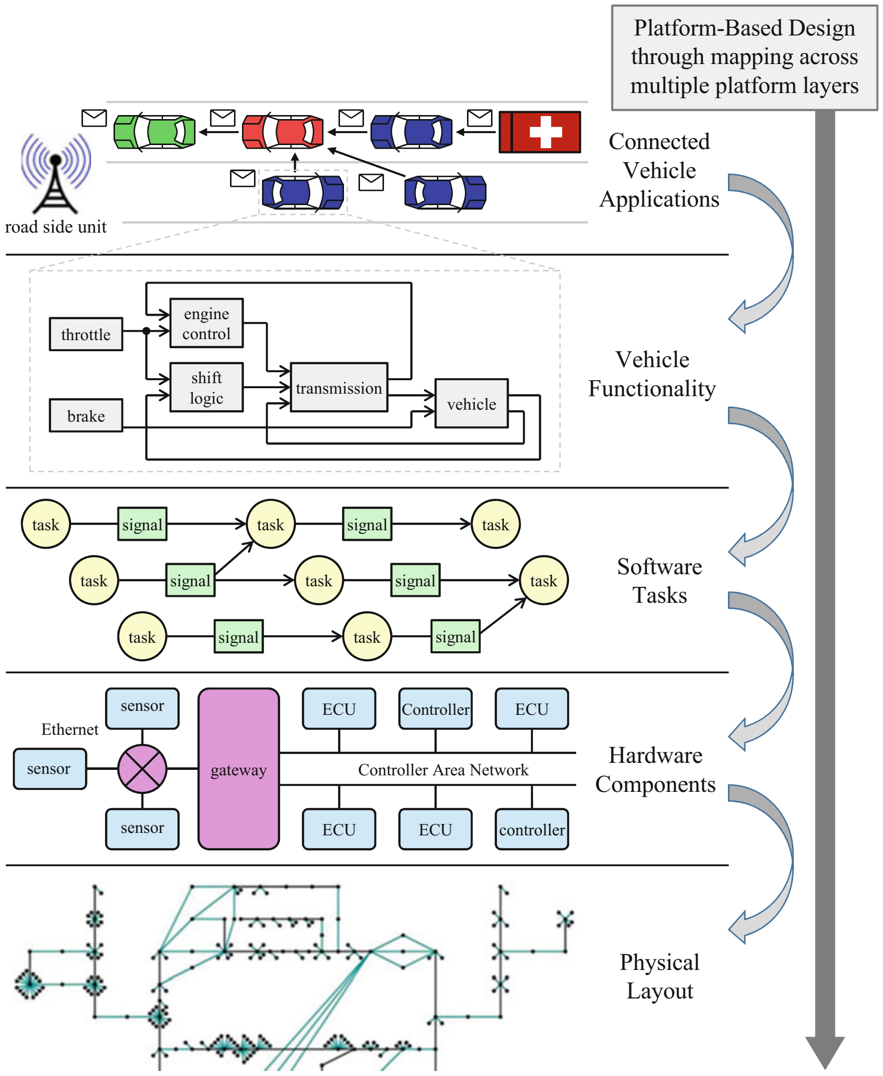

Fig. 2.2 Platform-based design for connected vehicles through mapping across multiple platform layers: (1) mapping connected vehicle applications to vehicle functionality, (2) mapping vehicle functionality to software tasks, (3) mapping software tasks to hardware components, and (4) mapping hardware components to physical layout

- **Design Space**: The design variables to be explored include the setting of contracts (constraints) on individual vehicle behavior/functionality—in the physical domain, this means constraints on vehicle's path planning and motion control; in cyber domain, this means constraints on computation latency, communication latency, reliability, etc.

- **Design Objectives and Constraints**: These could include safety, liveness, deadlock-free, fairness, traffic efficiency (e.g., for CACC and autonomous intersections), fuel consumption and emission (e.g., for eco-driving applications).

We have been developing a system-level modeling, synthesis and validation framework for connected vehicle applications [22, 27] and applying it to the above mapping (design space exploration) problem. In particular, we apply the methodology to a CACC application [9, 22] and an autonomous intersection management application [21, 23].

In the CACC application, vehicles inform each other about their speeds and accelerations via vehicle-to-vehicle (V2V) messages to maintain safe distances between them. We study the impact of communication delays and losses on the system safety and performance based on simulations, and then, in turn, derive the constraints for individual vehicle planning and control (i.e., constraints in the physical domain) [22].

In the autonomous intersection application, autonomous vehicles approaching an intersection will communicate with an intersection manager via vehicle-to-infrastructure (V2I) messages to request the right to enter and pass the intersection. The manager will then decide/schedule the entering order for the vehicles. We again study this application with consideration of communication delays and losses, and observe the significant impact of communication on system safety, liveness, and deadlock-free properties.

We then develop and analyze a delay-tolerant protocol for autonomous intersection management [21], as shown in Fig. 2.3. The protocol assures that as long

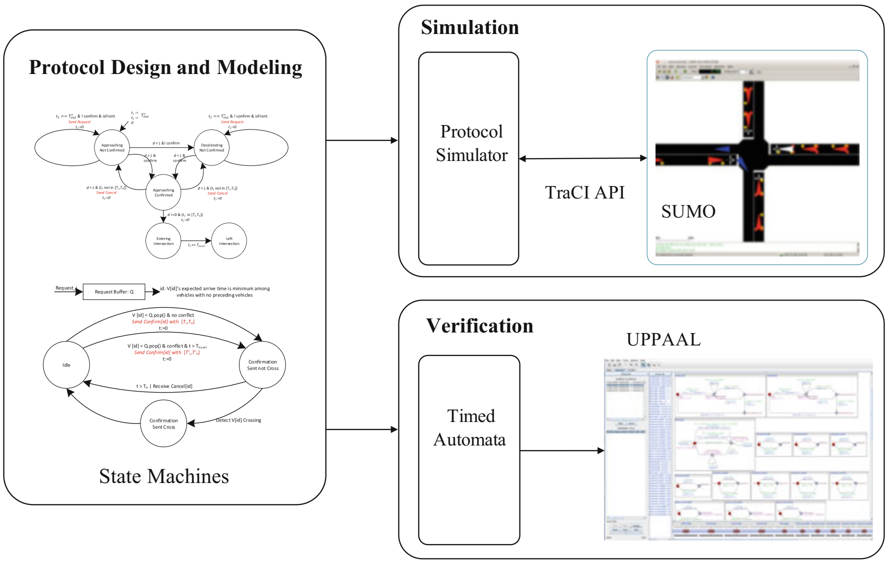
Protocol Design and Modeling
Approaching Not Confirmed
Decelerating Not Confirmed
Approaching Confirmed
Entering Intersection
Left Intersection
Send Request
Send Cancel
d < L &! confirm
Request Buffer: Q
id: V[id]'s expected arrive time is minimum among vehicles with no preceding vehicles
V [id] = Q.pop() & no conflict
Send Confirm[id] with [T_L,T_H]
V [id] = Q.pop() & conflict & t > T_travel
Idle
Confirmation Sent not Cross
Confirmation Sent Cross
t > T_H | Receive Cancel[id]
Detect V[id] Crossing
State Machines
Simulation
Protocol Simulator
TraCI API
SUMO
Verification
Timed Automata
UPPAAL

Fig. 2.3 Design and validation of delay-tolerant autonomous intersections [21]

as the communication delays are bounded, every vehicle will eventually cross the intersection (i.e., liveness property) and vehicles with conflicting routes will never enter the intersection at the same time (i.e., safety property). We verify the safety and liveness properties of our protocol by building more abstract timed automata models and leveraging the UPPAAL verification tool [19]. Finally, we implement our protocol in the SUMO traffic simulation suite [18], with the extension of modeling communication delays, to study the system performance. Such analysis allows us to derive the delay constraints on V2I communication in the cyber domain, which includes the delays of in-vehicle processing and the delays of V2I message transmissions, for ensuring system safety, liveness, deadlock-free, and performance.

2.3 Mapping Vehicle Functionality to Software Tasks

Once we have the specifications and constraints of individual vehicle functionality, the PBD paradigm can be applied to conduct the mapping from vehicle functionality to software tasks. This mapping problem can be formulated as follows:

- **Platforms**: (1) The higher-layer platform is captured by the models of vehicle functionality (e.g., Simulink models, timed automata), including in-vehicle sensing, computation and communication models, as well as V2X communication models; and (2) the lower-layer platform includes the models of software tasks and communication protocols.
- **Design Space**: The design variables to be explored include the generation of software tasks from functional models (i.e., mapping from functional blocks to tasks) and the design of communication protocols (including signals) from functional models.
- **Design Objectives and Constraints**: These may include a variety of constraints and optimization objectives on system performance, safety, security, cost, reliability, extensibility, memory size, reusability, modularity, etc.

For the mapping across these two layers, we have developed algorithms for exploring software task generation, allocation, and scheduling from functional models of finite state machines [25] and synchronous block diagrams [4, 6], two main models of computation in synchronous models that are widely used in capturing embedded sensing, control, and computation applications.

In [25], we developed a general partitioned model for multi-task implementations of synchronous finite state machines, and defined two metrics for measuring the quality of task implementations: the breakdown factor and the action extensibility. We then developed a heuristic algorithm to explore robust and extensible task generation and scheduling based on the two metrics. The experimental results demonstrated significant improvement on the two metrics from our algorithm, and showed the importance of exploring task generation options for synchronous finite state machines.

In [6], we developed an algorithm for direct generation of software tasks on single-core platforms from synchronous block diagrams, with respect to modularity, reusability, code size, and latency. This work showed the promise of exploring task generation for synchronous block diagrams.

In [4], we developed a complete model-based synthesis flow for automotive software systems that follow the AUTOSAR standard [1]. The synthesis flow optimizes the generation of AUTOSAR runnables from synchronous block diagrams, the mapping of runnables onto software tasks, and the allocation and scheduling of tasks onto multi-core ECU platforms. A key idea of this flow is to develop a uniformed formalism of firing and execution timing automata (FETA) to capture the periodic timing behavior of functional blocks, runnables, and tasks. Leveraging FETA, the flow can more accurately model and reason about system timing behavior across different layers during the entire mapping process. Finally, the synthesis flow addresses constraints and objectives on a variety of metrics when solving the mapping problems, including software engineering objectives such as runnable modularity, reusability, and code size as well as timing and resource objectives such as system schedulability and memory cost. In particular, the flow focuses on trading off modularity with schedulability during the mapping from functional blocks to runnables, and on minimizing memory cost under schedulability constraints during the mapping from runnables to tasks and from tasks to ECU cores. Similarly as [6, 25], this work showed the importance of exploring task generation options when mapping vehicle functionality to software tasks. Furthermore, it demonstrated the benefits of explicitly considering timing during task generation and having a uniformed formalism such as FETA to capture timing behavior across system layers.

2.4 Mapping Software Tasks to Hardware Components

Once we have a model of software tasks and their communication signals, the PBD paradigm can be further applied to explore the mapping of tasks onto hardware components. We have briefly discussed this above in [4] and will elaborate it more in this section. The mapping formulation for task to hardware platform mapping can be captured as follows:

- **Platforms**: (1) The higher-layer platform is typically modeled as task graphs with communication signals; and (2) the lower-layer platform includes architectural models of hardware components.
- **Design Space**: The design variables include task allocation, task scheduling, signal mapping to memory transactions or bus/wireless messages, message scheduling, etc.
- **Design Objectives and Constraints**: The constraints and objectives address metrics such as latency, schedulability, cost, energy consumption, extensibility, fault tolerance, and security.

In the following, we will demonstrate a few different mapping platforms across these two layers.

2.4.1 Conventional CAN-Bus Systems

The controller area network (CAN) protocol is still the most common in-vehicle network. The mapping problem from a task graph to a CAN-based system can be solved by the PBD paradigm. As shown in Fig. 2.4, the functional model is a task graph that consists of a set of tasks, denoted by $\mathtt{T} = \{\tau_1, \tau_2, \ldots, \tau_{|\mathtt{T}|}\}$, and a set of signals, denoted by $\mathtt{S} = \{\sigma_1, \sigma_2, \ldots, \sigma_{|\mathtt{S}|}\}$. Each signal σ_i is between a source task and a destination task. Each task is activated periodically and communicate with each other through signals. The architecture model is a distributed CAN-based platform that consists of a set of ECUs, denoted by $\mathtt{E} = \{\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_{n_\mathtt{E}}\}$, and a CAN bus that connects all the ECUs. Each ECU ε_k can send a set of messages, denoted by $\mathtt{M}_k = \{\mu_{k,1}, \mu_{k,2}, \ldots, \mu_{k,|\mathtt{M}_k|}\}$. ECUs are assumed to run AUTOSAR/OSEK-compliant operation systems that support preemptive priority-based task scheduling. The bus uses the standard CAN bus arbitration model that features non-preemptive priority-based message scheduling [2].

A path π is an ordered interleaving sequence of tasks and signals, defined as $\pi = (\tau_{r_1}, \sigma_{r_1}, \tau_{r_2}, \sigma_{r_2}, \ldots, \sigma_{r_{k-1}}, \tau_{r_k})$. $src(\pi) = \tau_{r_1}$ is the path's source and $snk(\pi) = \tau_{r_k}$ is its sink. Sources are activated by external events, while sinks activate actuators. Multiple paths may exist between each source–sink pair. We assume all tasks in a path perform computations that contribute to a distributed function, from the collection of sensor data to the remote actuation. The worst-case end-to-end latency incurred when traveling a path π is denoted as l_π, which

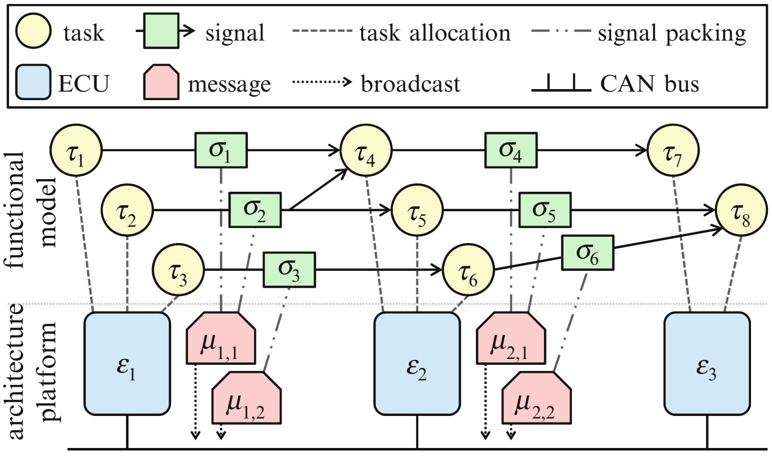

Fig. 2.4 The task mapping problem in a CAN-based system

represents the largest possible time interval that is required for the change of the input (or sensed) value at the source to be propagated and cause a value change (or an actuation response) at the sink.

During mapping, the functional model is mapped onto the architecture platform, as shown in Fig. 2.4. Specifically, the tasks are allocated to ECUs, and the signals are packed into messages and transmitted on the CAN bus in a broadcast fashion. Messages are triggered periodically and each message contains the latest values of the signals that mapped to it. Static priorities are assigned to tasks and messages for priority-based scheduling. The design space of task allocation, signal packing, and priority assignment is explored with respect to a set of design objectives and constraints.

For detailed problem formulations and their corresponding algorithms, please refer to our previous publications on task mapping for the CAN-based platform, with the consideration of end-to-end latency [3, 5, 24, 30], extensibility [10, 26, 28, 29], fault tolerance [20], and security [13, 15].

In the following, we will introduce task mapping onto two different architectural platforms—one replaces the CAN bus by a time division multiple access (TDMA) switch, and the other one utilizes an OS hypervisor to support multiple operating systems running on a hardware component.

2.4.2 Advanced Architecture: TDMA-Based Systems

The TDMA-based protocol is a very representative synchronous protocol and an abstraction of many existing protocols, such as the FlexRay [7], the Time-Triggered Protocol [17], the Time-Triggered Ethernet [16], and the Time-Sensitive Networking [8]. These protocols are likely to be adopted in future intelligent vehicles to support high and dynamic data rate. Compared with Ethernet, they also have more deterministic and predictable timing behavior. Compared with priority-based networks such as the CAN protocol, TDMA-based systems have fundamental differences in system modeling (in particular for latency modeling), on security mechanism selection (a global time is available for security reasons), on design space (network scheduling is the focus of this work but not a factor for CAN-based systems), and on algorithm design. Therefore, the approaches for CAN-based systems in the previous section do not apply to TDMA-based systems.

As shown in Fig. 2.5, similar to the system model in the previous section, the functional model is a task graph that consists of a set of tasks, denoted by $\mathtt{T} = \{\tau_1, \tau_2, \ldots, \tau_{|\mathtt{T}|}\}$, and a set of signals, denoted by $\mathtt{S} = \{\sigma_1, \sigma_2, \ldots, \sigma_{|\mathtt{S}|}\}$. Each signal σ_i is between a source task and a destination task, and each task is activated periodically and communicates with each other through signals. The architecture model is a distributed platform that consists of a set of ECUs, denoted

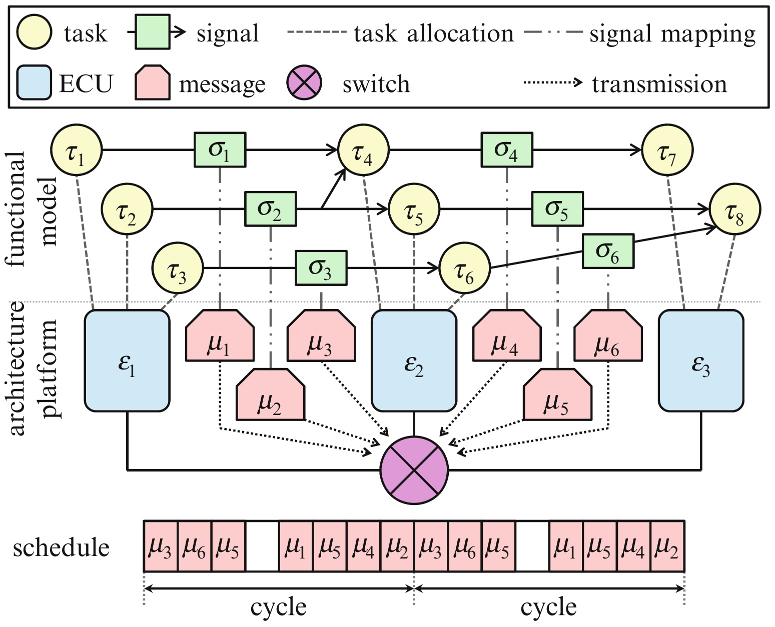
task
signal
task allocation
signal mapping
ECU
message
switch
transmission
functional model
architecture platform
τ_1 τ_2 τ_3 τ_4 τ_5 τ_6 τ_7 τ_8
σ_1 σ_2 σ_3 σ_4 σ_5 σ_6
ε_1 ε_2 ε_3
μ_1 μ_2 μ_3 μ_4 μ_5 μ_6
schedule
μ_3 μ_6 μ_5 μ_1 μ_5 μ_4 μ_2 μ_3 μ_6 μ_5 μ_1 μ_5 μ_4 μ_2
cycle
cycle

Fig. 2.5 The mapping problem of a TDMA-based system

by $\mathtt{E} = \{\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_{n_\mathtt{E}}\}$, and ECUs are assumed to support preemptive priority-based task scheduling. The nodes are connected through a TDMA-based switch (we focus on the single-switch case in this chapter, and our formulation can be extended to multi-switches cases). A set of messages is communicated among nodes through the switch, denoted by $\mathtt{M} = \{\mu_1, \mu_2, \ldots, \mu_{|\mathtt{M}|}\}$. The switch uses a TDMA-based model for scheduling, in which each *time slot* in the schedule can be assigned to one message. Several time slots form a *cycle*, and the network switch repeats the same scheduling sequence after each cycle. It is possible that a time slot is empty (not assigned to any message) in a schedule, and it is also possible that there are more than one time slots assigned to the same message in a cycle.

During mapping, the functional model is mapped onto the architecture platform, as shown in Fig. 2.5. Specifically, the tasks are allocated to ECUs, and the signals are one-to-one mapped onto messages and transmitted on the network. Messages are triggered periodically and each message contains the latest values of the signals that are mapped to the message. Static priorities are assigned to tasks for priority-based scheduling, and the time slots in the schedule are assigned to messages. The design space of task allocation, priority assignment, and switch scheduling is explored with respect to a set of design objectives and constraints.

For detailed problem formulation and its corresponding algorithm, please refer to our previous publications [12, 14] that address security in the mapping process.

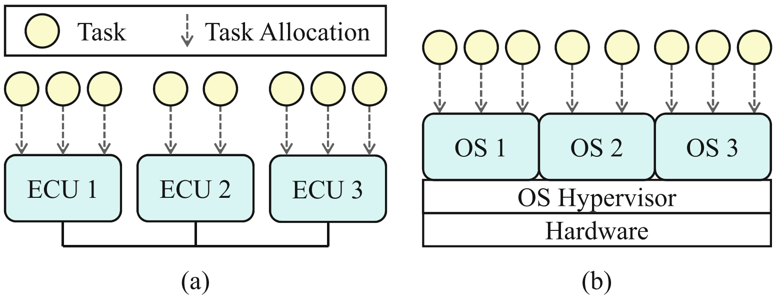

Fig. 2.6 (**a**) A traditional architecture, and (**b**) an architecture supported by an OS hypervisor

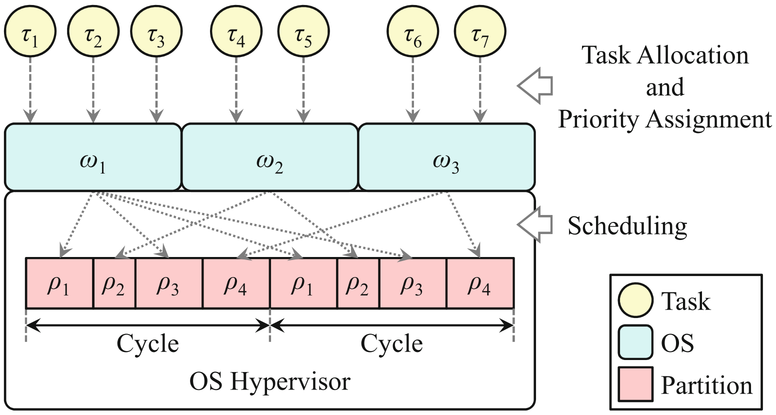

Fig. 2.7 The tasks are allocated to the operating systems, and the operating systems are scheduled on the OS hypervisor

2.4.3 *Advanced Architecture: OS-Hypervisor-Based Systems*

In this section, we consider mapping onto platforms with OS hypervisor. In Fig. 2.6a, there is a traditional architecture where the tasks are allocated directly on the ECUs. In Fig. 2.6b, an OS hypervisor runs between hardware and operating systems and virtualizes hardware. As a result, tasks and operating systems can be executed in a hardware-independent way. The OS hypervisor in Fig. 2.6b is categorized as a type-1 OS hypervisor which runs directly on hardware, while a type-2 OS hypervisor runs on a host operating system and supports other guest operating systems. In the market, there have been several OS hypervisors available. Although they have different features and specific applications (not only for automotive systems), the fundamental goal is still to virtualize hardware and provide high flexibility and isolation.

As shown in Fig. 2.7, the system model consists of a set of tasks, a set of operating systems, and an OS hypervisor. Each task τ_i is triggered periodically. We assume that all operating systems are identical, and each operating system ω_i supports the preemptive fixed-priority scheduling. The OS hypervisor supports the TDMA scheduling and maintains a *schedule* in which each *partition* ρ_i is assigned to one operating system,[1] and the OS hypervisor repeats the schedule after each *cycle*. It is possible that there is more than one partition assigned to the same operating system in a cycle.

The research on developing the mapping algorithm for this model is still ongoing.

2.4.4 Heterogeneous Communication Architectures

There are still some limitations with those approaches above. First, there is usually only one protocol to be considered, so the design methods cannot be applied to heterogeneous communication architectures. Next, the designs are for conventional functions which do not have very high data rates, and thus they cannot support ADAS and autonomous functions. Lastly, the architectures are usually fixed so that system designers have no flexibility to select appropriate hardware devices and design a topology for them. To address these problems and the challenges in Sect. 2.1, in this section, we propose a design methodology based on the PBD paradigm for heterogeneous communication architectures in automotive systems.

The design methodology is based on the mapping from functional models to architectural models. The notations which will be used in the methodology are listed in Table 2.1. We first define a device and an architectural model as follows:

Definition 2.1 A device δ is either a sensor, an actuator, an ECU, or a network device.

The location of a device is usually fixed according to the floor planning of an automotive system. A network device can be a CAN bus, a TSN switch, or a gateway.

Definition 2.2 An architectural model Δ is a set of devices.

An architectural model can be given by system designers directly or extracted from standardized languages. Each device in an architectural model is only a *candidate*, which means that it is possibly not selected during the mapping.

Definition 2.3 For each device δ, it is associated with a parameter C_δ as the device cost of δ. For each pair of devices δ and δ', it is with a parameter $D_{\delta,\delta'}$ as the connection cost of δ and δ' and another parameter $E_{\delta,\delta'}$ as the compatibility of δ and δ'.

[1]Some existing OS hypervisors allow one partition to be assigned to more than one operating system, and those operating systems are scheduled by their priorities. This can be generalized to the system model by defining task priority as a 2-tuple.

Table 2.1 Notations in the design methodology for heterogeneous communication architectures

δ	A device
C_δ	The device cost of δ
$D_{\delta,\delta'}$	The connection cost of δ and δ'
$E_{\delta,\delta'}$	The compatibility of δ and δ'
ι	An implementation
S_ι	The set of devices of ι
T_ι	The set of logical connections between devices of ι
U_ι	The set of reliability and safety constraints on logical paths between devices of U_ι
Δ	An architectural model or a set of devices
I	A functional model or a set of implementations
n	The number of functional models
σ	A sensor
π	An actuator
θ	An ECU
ϕ	A network device
Σ	The set of sensors
Π	The set of actuators
Θ	The set of ECUs
Φ	The set of network devices

The parameter $D_{\delta,\delta'}$ can be pre-computed based on the harness and routing graph in an automotive system, and it can be set as the distance or the wiring weight between δ and δ' which are physically connected. If $D_{\delta,\delta'} = \infty$, it means that there is no physical connection between δ and δ'. On the other hand, $E_{\delta,\delta'} = 1$ if and only if δ and δ' can be selected at the same time. The existence of the parameter $E_{\delta,\delta'}$ is to address the challenge of device selection mentioned in Sect. 2.1, e.g., if both of a regular ECU and an upgraded ECU are the candidates at the same location, only one of them can be selected.

As shown in Fig. 2.8a, the architectural model Δ has five devices including one sensor, one actuator, two ECUs, and one CAN bus. If δ_2 is a regular ECU, δ_3 is an upgraded ECU, and both of them are the candidates at the same location, then only one of them can be selected. Therefore, $E_{\delta_2,\delta_3} = 0$, while $E_{\delta_i,\delta_j} = 1$ for any other pair of devices. On the other hand, all devices except the CAN bus are only connected to the CAN bus, so $D_{\delta_i,\delta_j} = \infty$ for any pair of devices where $i, j \in \{1, 2, 3, 4\}$.

Definition 2.4 Given Δ, Σ is the set of sensors, Π is the set of actuators, Θ is the set of ECUs, Φ is the set of network devices, and thus $\Delta = \Sigma \cup \Pi \cup \Theta \cup \Phi$. Throughout the section, σ is a sensor, π is an actuator, θ is an ECU, ϕ is a network device, Then, we define an implementation and a functional model as follows:

Definition 2.5 An implementation ι is associated with S_ι as the set of devices, T_ι as the set of logical connections between devices, and U_ι as the set of reliability and safety constraints on logical paths between devices.

Note that T_ι can be represented by a set of subsets in S_ι, and U_ι can be represented by a set of reliability and safety constraints on tuples of elements in S_ι.

Definition 2.6 A functional model I is a set of implementations, and it can be implemented by any $\iota \in I$.

We define a functional model by its possible implementations on devices because we can translate system designers' experience into candidate implementations and significantly reduce the complexity and search space during design space exploration (e.g., we can keep the scenario that both ECUs need to be upgraded at the same time).

It should be mentioned that, in most cases, a functional model has the same sensors and actuators in all of its implementations, e.g., the sensors and actuators that the functions of a blind spot monitor use are fixed. On the other hand, there is usually some flexibility selecting ECUs to execute corresponding functions, no matter they are at the same location or at different locations, so a functional model usually has different sets of ECUs in its implementations. Lastly, there is usually no network device in an implementation, although it may be implied by the harness and routing graph or objective optimization and constraint satisfaction during mapping.

As shown in Fig. 2.8b, the functional model is $\{\iota_1, \iota_2\}$. For ι_1, $S_{\iota_1} = \{\delta_1, \delta_2, \delta_4\}$, $T_{\iota_1} = \{\{\delta_1, \delta_2\}, \{\delta_2, \delta_4\}\}$, and U_{ι_1} consists of the constraints on path $(\delta_1, \delta_2, \delta_4)$, e.g., its end-to-end latency of the functional path of ι_1 must be smaller than its deadline. Similarly, for ι_2, $S_{\iota_2} = \{\delta_1, \delta_3, \delta_4\}$, $T_{\iota_2} = \{\{\delta_1, \delta_3\}, \{\delta_3, \delta_4\}\}$, and U_{ι_2} consists of the constraints on path $(\delta_1, \delta_3, \delta_4)$.

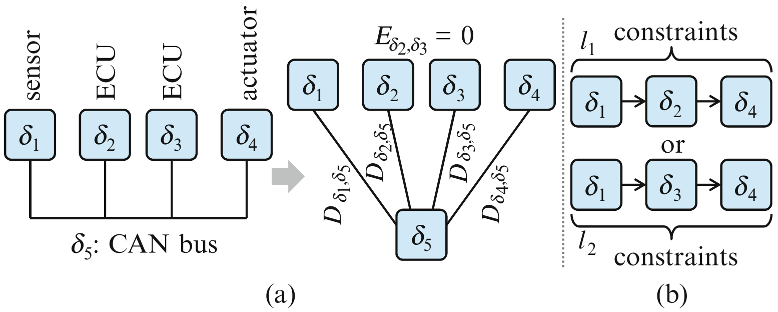

Fig. 2.8 (**a**) An architectural model with one sensor, one actuator, two ECUs, and one CAN bus, where the two ECUs are incompatible, and only one of them can be selected. (**b**) Two implementations of a functional model. One of them should be selected, depending on objective optimization and constraint satisfaction during mapping

With the definitions of an architectural model and a functional model, the design problem can be defined as follows:

- Given an architectural model Δ and n function model $\{I_1, I_2, \ldots, I_n\}$, select an implementation for each functional model such that all devices of selected implementations are compatible, all reliability and safety constraints of selected implementations are satisfied, and the objective is optimized.

As mentioned in Sect. 2.1, a reliability constraint can be the requirement of multiple routing paths. A safety constraint can be the utilization bound of each device or the requirement that the end-to-end latency of a functional path must be smaller than its deadline. The most typical objective is to minimize total cost which includes all device costs and all connection costs. Some other possible objectives are weight minimization, latency minimization, and performance maximization. As shown in Fig. 2.8, one implementation in Fig. 2.8b should be selected to implement the functional model, depending on objective optimization and constraint satisfaction during mapping.

Here is the summary of how the methodology addresses the design challenges mentioned in Sect. 2.1.

- **Addressing high-volume and dynamic input data**. By objective optimization and constraint satisfaction during mapping, a functional model with high data rate will be served by faster network devices (protocols) after mapping. If nearby ECUs are not powerful enough for dynamic data rate, a functional model will connect its sensors or actuators to further ECUs, and related objectives (e.g., connection cost) and constraints (e.g., end-to-end latency) will also be considered during mapping.
- **Computation architecture design**. In the methodology, different types of ECUs at the same location are all included in an architectural model and marked by the compatibility ($E_{\delta,\delta'}$). As mentioned above, this allows us to translate system designers' experience into candidate implementations and keep the scenario that both ECUs need to be upgraded at the same time. Then, the challenges in Sect. 2.1 can be addressed by objective optimization and constraint satisfaction during mapping. If a device has no load after mapping, it means that it is not selected.
- **Communication architecture design**. Similar to device selection, a network device may have no load on it, which means that it is not selected. On the other hand, gateways are considered in the methodology to composite different protocols.
- **Topology design**. In an architectural model, all possible connections and their costs ($D_{\delta,\delta'}$) are pre-computed. During mapping, those connections are candidates, and their costs can be considered.
- **Safety**. The end-to-end latency of a path can be defined with U_ι in a functional model and its implementations. Note that a frame is a special case of a path between two devices. The methodology leaves flexibility for system designers to apply different timing models. If those models are not available or their results are over-pessimistic so that simple bounds on the utilization of network devices

are adopted as safety constraints, the utilization of a device, which is a special case of a path with only one device, can also be defined with U_ι.
- **Reliability**. Similarly, all possible connections are candidates so that a function model can construct multiple routing paths from them.
- **Security**. Although security is not the focus of the methodology, other protocols and gateways in heterogeneous communication architecture can provide opportunities for adding security protections. The methodology is a platform for further security considerations during design stages.
- **Optimization objective**. The methodology leaves flexibility for system designers to set total cost, wiring weight, reliability, or performance as their objectives. To deal with different objectives, generalized optimization approaches should be applied.

Heterogeneous communication architectures are expected to be deployed to support ADAS and autonomous functions. In this section, we propose a methodology to address those challenges on heterogeneous communication architectures. Based on the methodology, we can formulate a problem and its corresponding algorithm to solve mapping problems at this level. The corresponding research is still ongoing.

2.5 Mapping Hardware Components to Physical Layouts

Finally, we can apply the PBD paradigm to map hardware components to physical layouts. Hardware components typically have pre-defined places for them. For example, radars should be placed at the front or rear side of a vehicle, not inside the vehicle. These components are connected by wires, which need to go through harnesses as shown in Fig. 2.9. The mapping problem from hardware to physical layouts can be captured as follows and illustrated in Fig. 2.10.

- **Platforms**: (1) The higher-layer platform includes a set of logical connections between hardware components; and (2) the lower-layer platform includes a physical routing graph that consists of wiring harnesses, connections between harnesses, locations (where a wire gets in or out of a harness) of wire harnesses, and hardware components.
- **Design Space**: The design variables to be explored include placement of splices, physical routing paths, and wire sizes.
- **Design Objectives and Constraints**: The metrics to be considered include total wiring length, total wiring weight, fuel efficiency, resistance, signal quality, space, and capacities of locations.

One problem formulation and its corresponding algorithm have been proposed in [11]. The features of the problem are:

- A logical connection can be defined as a hypergraph, i.e., a connection (hyperedge) can connect more than two components (vertices). To physically connect those components, we need to add splices physically (Steiner vertices logically),

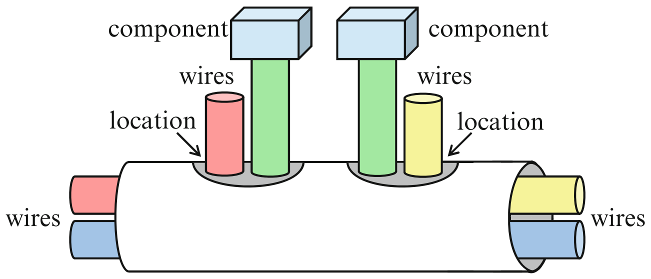

Fig. 2.9 A harness model with its locations [11]. Two components are connected by a wire, and the wire goes through the harness

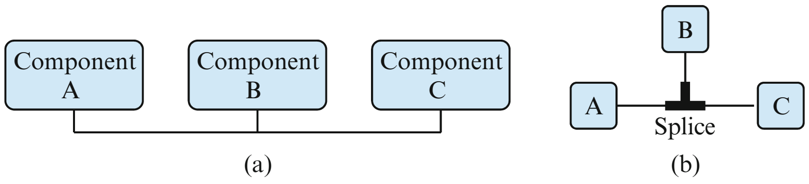

Fig. 2.10 (**a**) The logical connection between three hardware components is mapped to (**b**) the physical routing graph including a splice

which are similar to switches in network routing. A Steiner-tree problem for wire routing is also common in electronic design automation.

- The placement of harnesses is fixed. As a result, the problem is to select routing paths upon the given harnesses, and thus the number of potential routing paths is limited. From this perspective, the problem is closer to network routing rather than wire routing in electronic design automation.
- Similarly, a splice can only be placed at a location of a harness, so the number of potential locations is also limited.
- Considering resistance, the total wiring length and the total wiring weight have a quadratic relation because to maintain the same resistance for a wire, its length and the area of its cross section need to increase or decrease linearly. The total wiring weight is relevant to fuel efficiency as it is up to 30 kg in modern vehicles.

Please refer to [11] for detailed problem formulation and its corresponding algorithm.

2.6 Summary

In this chapter, we introduced the platform-based design (PBD) paradigm for automotive and transportation systems, and the application of PBD to map the high-level specification of connected vehicle applications to individual vehicle functionality, and then to software and hardware implementations, and finally to physical layouts. We believe that the PBD paradigm is a promising methodology to address the rapidly growing complexity of automotive design and improve design quality and productivity.

Acknowledgements We gratefully acknowledge the support from the National Science Foundation of the United States under Awards 1834324, 1834701, and 1839511, the Ministry of Education in Taiwan under Grant Number NTU-107V0901, and the Ministry of Science and Technology in Taiwan under Grant Number MOST-108-2636-E-002-011.

References

1. AUTOSAR. http://www.autosar.org
2. Robert Bosch GmbH. (1991). CAN specification (Version 2.0).
3. Davare, A., Zhu, Q., Di Natale, M., Pinello, C., Kanajan, S., & Sangiovanni-Vincentelli, A. (2007, June). Period optimization for hard real-time distributed automotive systems. In *Design Automation Conference (DAC'07)*.
4. Deng, P., Cremona, F., Zhu, Q., Di Natale, M., & Zeng, H. (2015, April). A model-based synthesis flow for automotive CPS. In *2015 ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)* (pp. 198–207).
5. Deng, P., Zhu, Q., Davare, A., Mourikis, A., Liu, X., & Natale, M. D. (2016, December). An efficient control-driven period optimization algorithm for distributed real-time systems. *IEEE Transactions on Computers, 65*(12), 3552–3566.
6. Deng, P., Zhu, Q., Di Natale, M., & Zeng, H. (2014, June). Task Synthesis for latency-sensitive synchronous block diagram. In *2014 9th IEEE International Symposium on Industrial Embedded Systems (SIES)* (pp. 112–121).
7. FlexRay Consortium. (2010, October). FlexRay communications system protocol specification (Version 3.0.1).
8. IEEE. (2011, March). IEEE standard for local and metropolitan area networks — timing and synchronization for time-sensitive applications in bridged local area networks. In *IEEE Std 802.1AS-2011* (pp. 1–292).
9. Liang, H., Jagielski, M., Zheng, B., Lin, C., Kang, E., Shiraishi, S., et al. (2018, November). Network and system level security in connected vehicle applications. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*.
10. Liang, H., Wang, Z., Zheng, B., & Zhu, Q. (2017, November). Addressing extensibility and fault tolerance in can-based automotive systems. In *2017 IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*.
11. Lin, C.-W., Rao, L., Giusto, P., D'Ambrosio, J., & Sangiovanni-Vincentelli, A. (2015, November). Efficient wire routing and wire sizing for weight minimization of automotive systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 34*(11), 1730–1741.
12. Lin, C.-W., Zheng, B., Zhu, Q., & Sangiovanni-Vincentelli, A. (2015, December). Security-aware design methodology and optimization for automotive systems. *ACM Transactions on Design Automation of Electronic Systems, 21*(1), 18:1–18:26.
13. Lin, C.-W., Zhu, Q., Phung, C., & Sangiovanni-Vincentelli, A. (2013). Security-aware mapping for CAN-based real-time distributed automotive systems. In *2013 IEEE/ACM International Conference on Computer-Aided Design* (pp. 115–121)
14. Lin, C.-W., Zhu, Q., & Sangiovanni-Vincentelli, A. (2014, November). Security-aware mapping for TDMA-based real-time distributed systems. In *2014 IEEE/ACM International Conference on Computer-Aided Design* (pp. 24–31)

15. Lin, C.-W., Zhu, Q., & Sangiovanni-Vincentelli, A. (2015, March). Security-aware modeling and efficient mapping for CAN-based real-time distributed automotive systems. *IEEE Embedded Systems Letters, 7*(1), 11–14.
16. SAE. (2011, November). Time-triggered ethernet. *SAE Standard AS6802*.
17. SAE. (2011, February). TTP communication protocol. *SAE Standard AS6003*.
18. SUMO. (2017). http://www.dlr.de/ts/en/desktopdefault.aspx/tabid-9883/16931_read-41000/
19. UPPAAL. (2017). https://www.uppaal.org/
20. Zheng, B., Gao, Y., Zhu, Q., & Gupta, S. (2015, October). Analysis and optimization of soft error tolerance strategies for real-time systems. In *2015 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)* (pp. 55–64).
21. Zheng, B., Lin, C. W., Liang, H., Shiraishi, S., Li, W., & Zhu, Q. (2017, May). Delay-aware design, analysis and verification of intelligent intersection management. In *2017 IEEE International Conference on Smart Computing (SMARTCOMP)* (pp. 1–8).
22. Zheng, B., Lin, C.-W., Yu, H., Liang, H., & Zhu, Q. (2016, November). CONVINCE: A cross-layer modeling, exploration and validation framework for next-generation connected vehicles. In *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*.
23. Zheng, B., Sayin, M. O., Lin, C. W., Shiraishi, S., & Zhu, Q. (2017, November). Timing and security analysis of VANET-based intelligent transportation systems: (invited paper). In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)* (pp. 984–991).
24. Zheng, W., Zhu, Q., Natale, M. D., & Sangiovanni-Vincentelli, A. (2007). Definition of task allocation and priority assignment in hard real-time distributed systems. In *RTSS ’07: Proceedings of the 28th IEEE International Real-Time Systems Symposium* (pp. 161–170)
25. Zhu, Q., Deng, P., Di Natale, M., & Zeng, H. (2013, March). Robust and extensible task implementations of synchronous finite state machines. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2013* (pp. 1319–1324)
26. Zhu, Q., Liang, H., Zhang, L., Roy, D., Li, W., & Chakraborty, S. (2017, June). Extensibility-driven automotive in-vehicle architecture design. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)* (pp. 1–6)
27. Zhu, Q., & Sangiovanni-Vincentelli, A. (2018, Sept). Codesign methodologies and tools for cyber–physical systems. *Proceedings of the IEEE, 106*(9), 1484–1500.
28. Zhu, Q., Yang, Y., Natale, M. D., Scholte, E., & Sangiovanni-Vincentelli, A. (2010). Optimizing the software architecture for extensibility in hard real-time distributed systems. *IEEE Transactions on Industrial Informatics, 6*(4):621–636.
29. Zhu, Q., Yang, Y., Scholte, E., Natale, M. D., & Sangiovanni-Vincentelli, A. (2009). Optimizing extensibility in hard real-time distributed systems. In *RTAS ’09: Proceedings of the 2009 15th IEEE Real-Time and Embedded Technology and Applications Symposium* (pp. 275–284).
30. Zhu, Q., Zeng, H., Zheng, W., Di Natale, M., & Sangiovanni-Vincentelli, A. (2012). Optimization of task allocation and priority assignment in hard real-time distributed systems. *ACM Transactions on Embedded Computing Systems, 11*(4), 85:1–85:30.