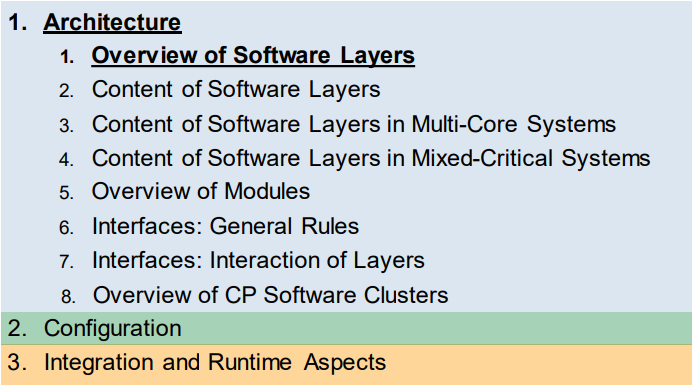
**Layered Software Architecture AUTOSAR**

***Autosar Classic Platform***



## Giới thiệu

### Mục đích của tài liệu này

**Layered Software Architecture** là một phần quan trọng trong việc mô tả **kiến trúc phần mềm** của **AUTOSAR**. Cụ thể, kiến trúc này bao gồm các nội dung chính sau:

1. **Mô tả cấu trúc phân cấp của phần mềm AUTOSAR theo cách tiếp cận từ trên xuống**:
   * Phương pháp này bắt đầu bằng cái nhìn tổng quan về toàn bộ hệ thống, sau đó dần đi sâu vào chi tiết các thành phần cụ thể.
   * Cấu trúc phân cấp được xây dựng dựa trên việc phân tách các nhiệm vụ và chức năng, tạo thành các lớp phần mềm độc lập nhưng có tính kết nối.
2. **Ánh xạ các Mô-đun Phần mềm Cơ bản (Basic Software Modules) vào các lớp phần mềm**:
   * Các **Mô-đun Phần mềm Cơ bản (BSW Modules)**, như **ECU Abstraction Layer**, **Microcontroller Abstraction Layer**, và **Services Layer**, được phân bổ vào các lớp phần mềm tương ứng.
   * Việc ánh xạ này giúp định nghĩa rõ ràng nhiệm vụ của từng mô-đun trong từng lớp, đảm bảo tính mô-đun hóa và khả năng tái sử dụng.
3. **Thể hiện mối quan hệ giữa các lớp phần mềm và mô-đun**:
   * Các mối quan hệ này mô tả cách các lớp phần mềm tương tác với nhau, ví dụ: lớp ứng dụng (Application Layer) sẽ giao tiếp với lớp dịch vụ (Services Layer) thông qua các API chuẩn hóa.
   * Mối quan hệ này đảm bảo sự phối hợp hoạt động hiệu quả giữa các thành phần phần mềm, giúp hệ thống đạt được mục tiêu tích hợp và vận hành ổn định.

Tài liệu này **không chứa các yêu cầu cụ thể** mà chỉ có tính chất **thông tin**. Những **ví dụ** được đưa ra trong tài liệu không nhất thiết phải hoàn chỉnh ở tất cả các khía cạnh, nghĩa là chúng chỉ mang tính minh họa và không yêu cầu phải bao quát mọi chi tiết.

Mục tiêu của tài liệu là tập trung vào các **góc nhìn tĩnh** của **kiến trúc phần mềm lớp mô hình**. Cụ thể, tài liệu không đi vào chỉ định một **kiến trúc phần mềm cấu trúc** (hay thiết kế phần mềm), cũng như không cung cấp các mô tả chi tiết về **giao diện tĩnh và động** của hệ thống.

* **Kiến trúc phần mềm cấu trúc** (software architecture design) bao gồm các mô tả về cách thức các thành phần trong hệ thống tương tác với nhau, cả ở mức **tĩnh** (các thành phần phần mềm và giao diện của chúng) và **động** (các quy trình, luồng dữ liệu, và sự kiện trong hệ thống).
* **Thông tin chi tiết về giao diện tĩnh và động** không được đề cập trong tài liệu này, thay vào đó, các mô tả này sẽ được **bao gồm trong các đặc tả** của các **mô-đun phần mềm cơ bản chính** (Basic Software Modules specifications).

### Đầu vào

Tài liệu này dựa trên các tài liệu đặc tả và yêu cầu của AUTOSAR.

## Phạm vi và Khả năng Mở rộng

### Phạm vi ứng dụng của AUTOSAR

**AUTOSAR** (Automotive Open System Architecture) được phát triển đặc biệt dành cho các **Đơn vị Điều khiển Điện tử (ECU - Electronic Control Units)** trong ngành công nghiệp ô tô. Các **ECU** này có các đặc tính kỹ thuật sau:

1. **Tương tác mạnh mẽ với phần cứng (cảm biến và bộ điều khiển)**:
   * Các ECU trong ô tô có nhiệm vụ xử lý dữ liệu từ nhiều cảm biến (như cảm biến tốc độ, cảm biến nhiệt độ, cảm biến ánh sáng, v.v.) và gửi các tín hiệu điều khiển đến các bộ điều khiển khác (ví dụ như bộ điều khiển động cơ, hệ thống phanh, v.v.). Điều này đòi hỏi khả năng giao tiếp mạnh mẽ và liên tục giữa phần mềm và phần cứng.
2. **Kết nối với các mạng xe như CAN, LIN, FlexRay hoặc Ethernet**:
   * Các ECU thường phải kết nối và trao đổi dữ liệu qua các **mạng xe** như **CAN (Controller Area Network)**, **LIN (Local Interconnect Network)**, **FlexRay**, hoặc **Ethernet**. Những mạng này cung cấp cơ sở hạ tầng truyền thông để các ECU có thể giao tiếp với nhau và đồng bộ hóa hoạt động trong hệ thống ô tô.
3. **Bộ điều khiển vi xử lý với tài nguyên hạn chế**:
   * Các ECU sử dụng **bộ điều khiển vi xử lý (Microcontroller)** thường có cấu trúc 16 bit hoặc 32 bit và bị giới hạn về **công suất tính toán** và **bộ nhớ**. Điều này khác biệt so với các giải pháp doanh nghiệp, nơi các hệ thống có thể có tài nguyên mạnh mẽ hơn. Do đó, phần mềm trong AUTOSAR phải được tối ưu hóa để hoạt động hiệu quả trên các vi xử lý có tài nguyên hạn chế.
4. **Hệ thống Thời gian Thực (Real-Time System)**:
   * Các ECU yêu cầu hệ thống phải hoạt động trong **thời gian thực**, tức là phần mềm phải có khả năng phản hồi nhanh và chính xác với các sự kiện trong hệ thống (ví dụ: khi xe dừng đột ngột, hệ thống phanh phải phản ứng ngay lập tức). Điều này đòi hỏi các ứng dụng trong AUTOSAR phải tuân thủ các yêu cầu nghiêm ngặt về độ trễ và độ tin cậy.
5. **Thực thi chương trình từ bộ nhớ flash nội hoặc ngoại**:
   * Phần mềm của các ECU thường được lưu trữ và thực thi từ **bộ nhớ flash** (có thể là bộ nhớ flash nội của ECU hoặc bộ nhớ flash ngoại gắn ngoài). Điều này giúp tiết kiệm không gian bộ nhớ và tăng tốc độ truy xuất dữ liệu.

**Lưu ý:** Trong ngữ cảnh của AUTOSAR, AUTOSAR không quan tâm đến cấu trúc vật lý của ECU, mà chỉ tập trung vào việc mô tả và cấu hình các ECU từ góc độ phần mềm và các mô-đun phần mềm tương ứng. Mỗi bộ điều khiển vi xử lý trong một vỏ (housing) đều phải được xem như một ECU riêng biệt với mô tả đầy đủ về phần mềm và cấu hình của nó trong hệ thống AUTOSAR.

### Khả năng Mở rộng của AUTOSAR

**Kiến trúc Phần mềm AUTOSAR** là một phương pháp **tổng quát**, có nghĩa là nó được thiết kế để hỗ trợ tính **mở rộng** và **tích hợp** các thành phần một cách linh hoạt trong khi vẫn duy trì tính chuẩn hóa. Các đặc điểm quan trọng của kiến trúc này bao gồm:

1. **Các mô-đun tiêu chuẩn có thể được mở rộng về chức năng, trong khi vẫn tuân thủ**:
   * Kiến trúc AUTOSAR cho phép các mô-đun phần mềm tiêu chuẩn (Standard Software Modules) có thể được **mở rộng** để thực hiện thêm các chức năng cụ thể mà không làm mất đi tính tương thích với các tiêu chuẩn đã thiết lập.
   * Tuy nhiên, khi mở rộng các mô-đun này, **cấu hình của chúng** phải được **xem xét trong quá trình cấu hình Tự động Phần mềm Cơ bản (Basic Software Configuration)**. Điều này đảm bảo rằng các mô-đun mở rộng vẫn tuân thủ đúng các quy chuẩn của AUTOSAR, không làm ảnh hưởng đến các phần mềm khác trong hệ thống, và hệ thống vẫn hoạt động ổn định và hiệu quả.
2. **Các mô-đun không tiêu chuẩn có thể được tích hợp vào hệ thống dưới dạng các Trình điều khiển Phức tạp**:
   * AUTOSAR cho phép tích hợp các **mô-đun không tiêu chuẩn** (Non-Standard Modules) vào hệ thống, nhưng chúng phải được thực hiện dưới dạng **Trình điều khiển Phức tạp (Complex Drivers)**. Những trình điều khiển này có thể bao gồm các phần mềm đặc thù cho phần cứng hoặc các ứng dụng không hoàn toàn phù hợp với mô-đun phần mềm chuẩn trong AUTOSAR.
   * Việc tích hợp các mô-đun này yêu cầu các biện pháp đặc biệt để đảm bảo rằng chúng không gây xung đột hoặc ảnh hưởng đến các mô-đun phần mềm khác trong hệ thống.
3. **Không thể thêm các lớp bổ sung**:
   * Trong kiến trúc AUTOSAR, không được phép thêm **các lớp bổ sung** vào giữa các lớp phần mềm đã được định nghĩa. Điều này có nghĩa là các lớp phần mềm trong hệ thống phải tuân theo một cấu trúc phân lớp rõ ràng và không thể thay đổi hoặc mở rộng theo cách mà tạo ra sự phức tạp không cần thiết.
   * Điều này giúp duy trì tính ổn định và khả năng mở rộng của hệ thống, đồng thời đảm bảo rằng các mô-đun và lớp phần mềm trong AUTOSAR tương thích và có thể tích hợp một cách hiệu quả.

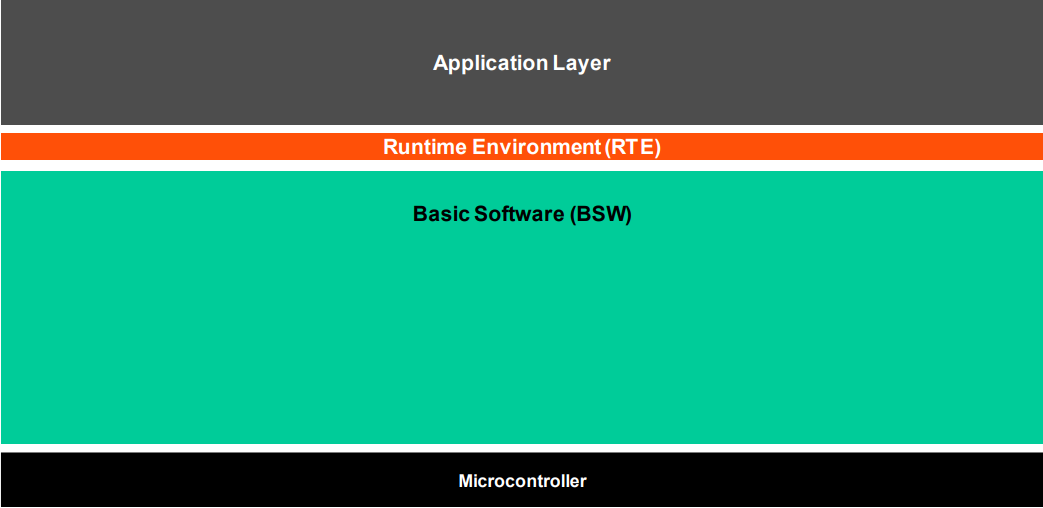
Kiến trúc Phần mềm AUTOSAR là một phương pháp tổng quát: ➢ các mô-đun tiêu chuẩn có thể được mở rộng về chức năng, trong khi vẫn tuân thủ, ◼ tuy nhiên, cấu hình của chúng phải được xem xét trong quá trình cấu hình Tự động Phần mềm Cơ bản! ➢ các mô-đun không tiêu chuẩn có thể được tích hợp vào hệ thống dựa trên AUTOSAR dưới dạng các Trình điều khiển Phức tạp và ➢ không thể thêm các lớp bổ sung.

# ARCHITECTURE

## Overview of Software Layers

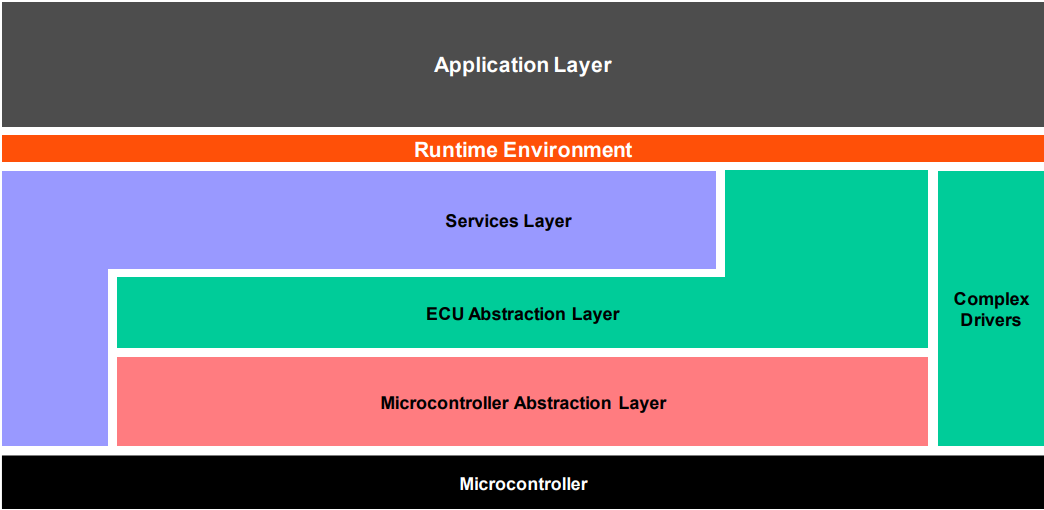
### Top view

AUTOSAR Architecture phân biệt ở mức trừu tượng cao nhất giữa ba lớp phần mềm: Application, Runtime Environment và Basic Software, chạy trên một Bộ điều khiển vi xử lý.



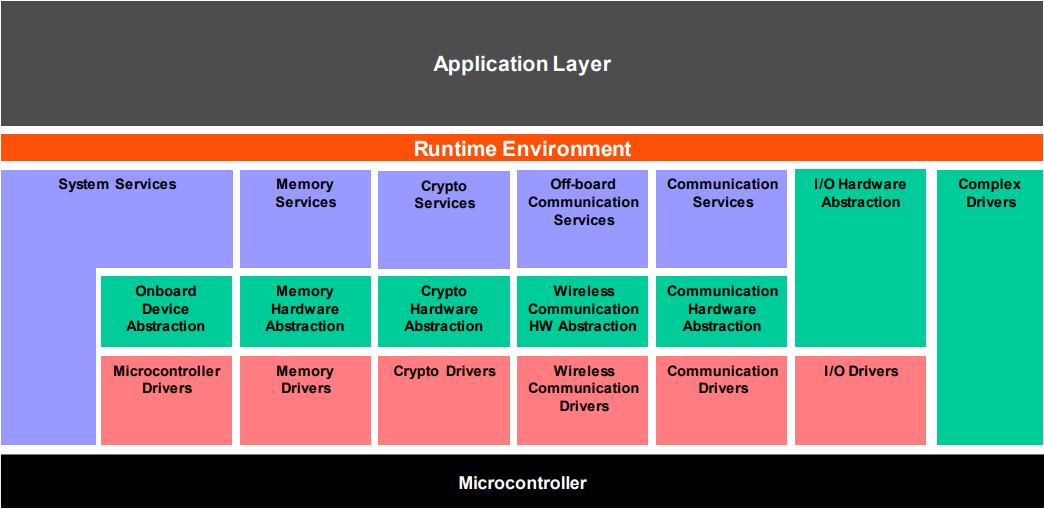
### Coarse view

Phần mềm cơ bản AUTOSAR được chia thành các lớp: Services, ECU Abstraction, Microcontroller Abstraction và Complex Drivers.

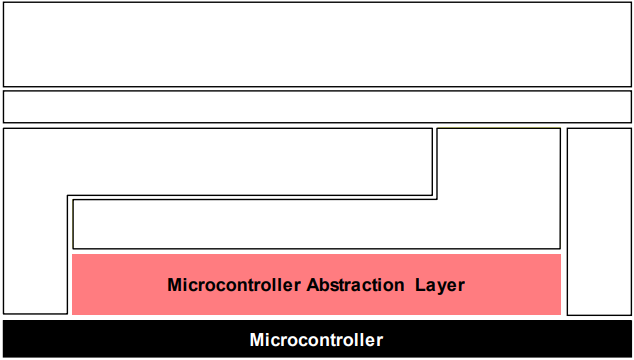


### Detailed view

Các Basic Software Layers được chia thành các nhóm chức năng. Ví dụ về Services của System, Memory và Communication Services.



### Microcontroller Abstraction Layer



**Microcontroller Abstraction Layer** là lớp phần mềm thấp nhất của Basic Software.

Nó chứa các trình điều khiển nội bộ, là các mô-đun phần mềm có truy cập trực tiếp đến µC và các thiết bị nội bộ.

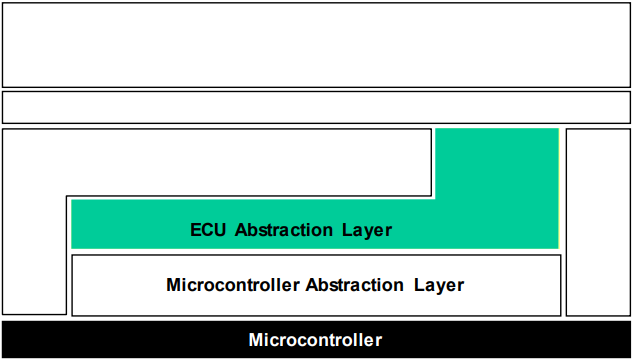
***Nhiệm vụ***

* Tạo các lớp phần mềm cao hơn độc lập với Bộ điều khiển vi xử lý

***Thuộc tính***

* Triển khai: Phụ thuộc vào µC
* Giao diện trên cùng: Chuẩn hóa và độc lập với µC

### ECU Abstraction Layer



**ECU Abstraction Layer** giao tiếp với các trình điều khiển của Microcontroller Abstraction Layer (MCAL). Nó cũng bao gồm các trình điều khiển cho các thiết bị ngoại vi.

Nó cung cấp một API để truy cập vào các ngoại vi và thiết bị bất kể vị trí của chúng (µC internal/external) và kết nối của chúng với µC (chân cổng, loại giao diện).

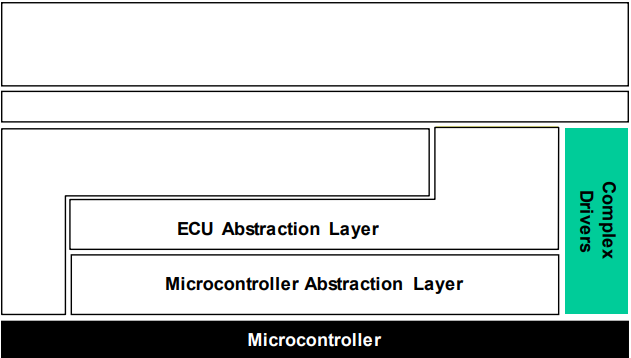
***Nhiệm vụ***

* Tạo các lớp phần mềm cao hơn độc lập với bố trí phần cứng của Đơn vị Điều khiển Điện tử

***Thuộc tính***

* Triển khai: Độc lập với µC, phụ thuộc vào ECU hardware.
* Giao diện trên cùng: Độc lập với µC và ECU hardware.

### Complex Drivers



**Complex Drivers Layer** bao gồm từ phần cứng đến RTE.

***Nhiệm vat***

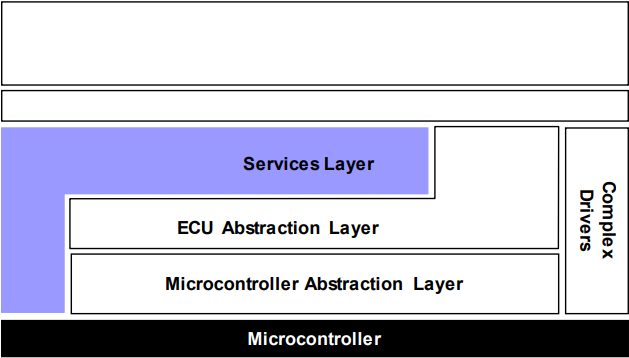
Cung cấp khả năng tích hợp chức năng đặc biệt, ví dụ: Trình điều khiển cho các thiết bị:

* Không được chỉ định trong AUTOSAR,
* Có ràng buộc thời gian rất cao hoặc
* Cho mục đích di chuyển và cấu hình lại v.v.

***Thuộc tính***

* Triển khai: có thể là ứng dụng, phụ thuộc vào µC và ECU hardware.
* Giao diện trên cùng: có thể là ứng dụng, phụ thuộc vào µC và ECU hardware.

### Services Layer



**Services Layer** là lớp cao nhất của Basic Software, cũng như liên quan đến tính ứng dụng của nó: khi truy cập đến các tín hiệu I/O được hỗ trợ trong ECU Abstraction Laye, Services Layer cung cấp:

* Chức năng của hệ điều hành.
* Dịch vụ giao tiếp và quản lý mạng xe
* Dịch vụ bộ nhớ (quản lý NVRAM)
* Dịch vụ chẩn đoán (bao gồm giao tiếp UDS, bộ nhớ lỗi và xử lý lỗi)
* Quản lý trạng thái ECU, quản lý chế độ
* Giám sát dòng chương trình logic và thời gian (quản lý Wdg)

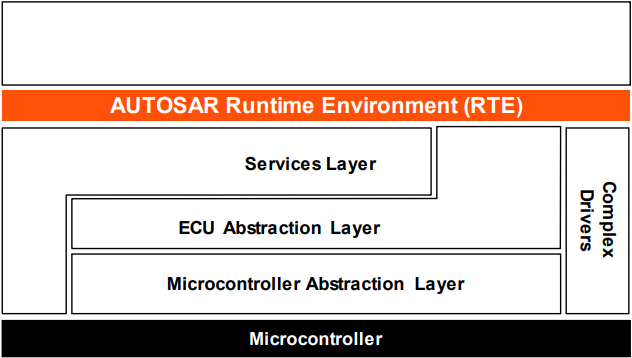
***Nhiệm vụ***

* Cung cấp các dịch vụ cơ bản cho Applications, RTE và Basic Software modules.

***Thuộc tính***

* Triển Khai: chủ yếu độc lập với µC và ECU hardware.
* Giao diện trên cùng: độc lập với µC và ECU hardware.

### Runtime Environment (RTE)



**RTE** là một lớp cung cấp dịch vụ giao tiếp cho **Application Software** (AUTOSAR Software Components and/or AUTOSAR Sensor/Actuator components).

Phía trên RTE, kiến trúc phần mềm thay đổi từ " layered" sang " “component style".

AUTOSAR Software Components giao tiếp với các thành phần khác (bên trong và / bên ngoài ECU) và/ dịch vụ thông qua RTE.

**Nhiệm vụ**

* Làm cho AUTOSAR Software Components độc lập với phép ánh xạ đến một ECU cụ thể.

**Thuộc tính**

* Triển khai: Đặc thù cho ECU và ứng dụng (tạo ra riêng lẻ cho mỗi ECU)
* Giao diện trên cùng: độc lập hoàn toàn với ECU.

### Introduction to types of services

**Basic Software** có thể được chia thành các loại dịch vụ sau:

**➢ Input/Output (I/O)**

Truy cập chuẩn hóa đến cảm biến, bộ điều khiển và các thiết bị ngoại vi trên ECU

**➢ Memory**

Truy cập chuẩn hóa đến bộ nhớ nội/ ngoại vi (non-volatile memory)

**➢ Crypto (Mã hóa)**

Quyền truy cập được tiêu chuẩn hóa vào các nguyên tắc mã hóa bao gồm các phần cứng bên trong/bên ngoài

**➢ Giao tiếp**

Chuẩn hóa truy cập: các hệ thống mạng trong xe, các hệ thống giao tiếp trên ECU và phần mềm bên trong ECU.

➢ **Giao tiếp ngoài ECU**

Quyền truy cập được tiêu chuẩn hóa: Giao tiếp giữa xe với X, trong hệ thống mạng không dây trên xe, Hệ thống giao tiếp bên ngoài ECU.

**➢ System**

Cung cấp các dịch vụ chuẩn hóa (hệ điều hành, bộ đếm thời gian, bộ nhớ lỗi) và dịch vụ đặc thù của ECU (quản lý trạng thái ECU, quản lý watchdog) và các chức năng thư viện.

### Driver (internal)

Một **driver** chứa chức năng để kiểm soát và truy cập vào một thiết bị bên trong hoặc bên ngoài.

Các thiết bị bên trong được đặt bên trong bộ điều khiển vi xử lý. Ví dụ về các thiết bị bên trong bao gồm:

* Bộ nhớ EEPROM bên trong.
* Bộ điều khiển CAN bên trong.
* Bộ chuyển đổi ADC bên trong.

Một trình điều khiển bên trong của một thiết bị được gọi là **internal driver** và được đặt trong Microcontroller Abstraction Layer.

### Driver (external)

External devices được đặt bên ngoài ECU hardware của microcontroller. Ví dụ về các thiết bị ngoại bao gồm:

* EEPROM bên ngoài.
* Bộ đếm watchdog bên ngoài.
* Bộ nhớ flash bên ngoài.

Một trình điều khiển bên ngoài cho một thiết bị được gọi là external driver và được đặt trong ECU Abstraction Layer. Nó truy cập vào thiết bị ngoại thông qua các trình điều khiển của Microcontroller Abstraction Layer.

Cách này cũng hỗ trợ các thành phần tích hợp trong System Basis Chips (SBCs) như bộ chuyển đổi và bộ đếm watchdog bởi AUTOSAR.

➢ Ví dụ: một trình điều khiển cho EEPROM bên ngoài với giao diện SPI truy cập vào EEPROM bên ngoài thông qua bộ xử lý/trình điều khiển cho bus SPI.

**Exception:**

Các trình điều khiển cho các thiết bị ngoại **được ánh xạ bộ nhớ** (ví dụ: bộ nhớ flash ngoại) có thể truy cập trực tiếp vào microcontroller. Các trình điều khiển bên ngoài này được đặt trong Microcontroller Abstraction Layer vì chúng phụ thuộc vào microcontroller.

### Interface

Một **Interface (interface module)** chứa chức năng để trừu tượng hóa các mô-đun được đặt dưới mức kiến ​​trúc của chúng. Ví dụ, một mô-đun giao diện trừu tượng hóa từ việc thực hiện phần cứng của một thiết bị cụ thể. Nó cung cấp một API chung để truy cập vào một loại thiết bị cụ thể không phụ thuộc vào số lượng thiết bị tồn tại của loại đó và không phụ thuộc vào việc thực hiện phần cứng của các thiết bị khác nhau.

Giao diện không thay đổi nội dung của dữ liệu.

Nhìn chung, các giao diện được đặt trong **ECU Abstraction Layer.**

Example: một giao diện cho hệ thống giao tiếp CAN cung cấp một API chung để truy cập vào các mạng giao tiếp CAN không phụ thuộc vào số lượng CAN Controllers trong một ECU và không phụ thuộc vào việc thực hiện phần cứng (bên trong chip, bên ngoài chip).

### Handler

Một **handler** là một giao diện cụ thể điều khiển việc truy cập đồng thời, đa luồng và không đồng bộ của một hoặc nhiều khách hàng đến một hoặc nhiều trình điều khiển. Nó thực hiện việc lưu trữ tạm thời, queuing, arbitration, multiplexing.

Bộ xử lý không thay đổi nội dung của dữ liệu.

Chức năng của bộ xử lý thường được tích hợp trong trình điều khiển hoặc giao diện SPIHandlerDriver, ADC Driver).

### Manager

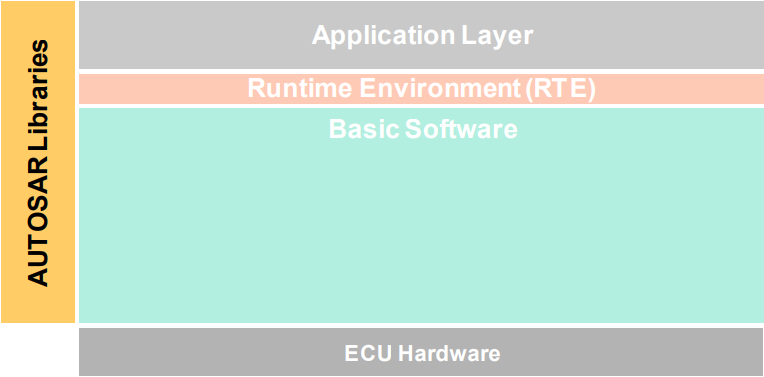
Một **manager** cung cấp các dịch vụ cụ thể cho nhiều khách hàng. Nó cần thiết trong tất cả các trường hợp mà chức năng của bộ xử lý không đủ để trừu tượng hóa từ nhiều khách hàng.

Ngoài chức năng của bộ xử lý, một quản lý có thể đánh giá và thay đổi hoặc điều chỉnh nội dung của dữ liệu.

Nhìn chung, các quản lý được đặt trong **Services Layer.**

Example: Bộ quản lý NVRAM quản lý việc truy cập đồng thời vào các thiết bị bộ nhớ nội và/hoặc ngoại như bộ nhớ flash và EEPROM. Nó cũng thực hiện việc lưu trữ dữ liệu phân tán và đáng tin cậy, kiểm tra dữ liệu, cung cấp các giá trị mặc định v.v.

### Giới thiệu về Thư viện



**Libraries** là một bộ sưu tập các hàm cho các mục đích liên quan.

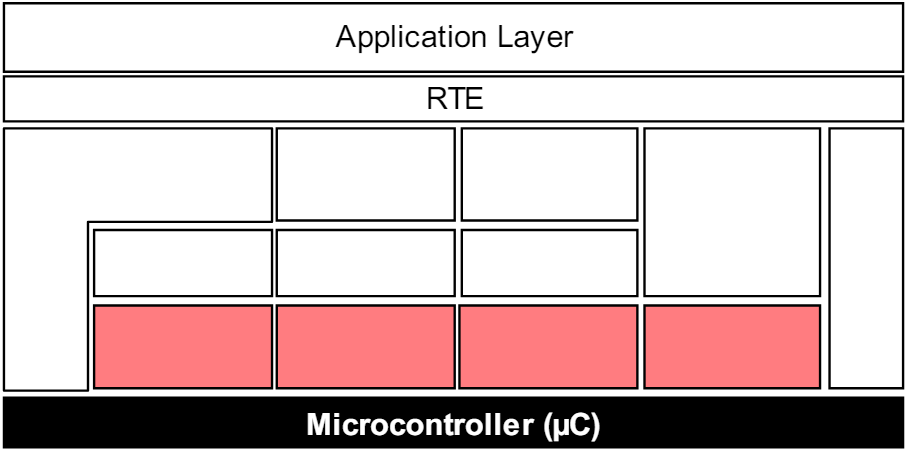
Thư viện:

* Có thể được gọi bởi các mô-đun BSW (bao gồm RTE), SW-Cs, thư viện hoặc mã tích hợp
* Chạy trong ngữ cảnh của người gọi trong cùng một môi trường bảo vệ
* Chỉ có thể gọi thư viện
* Có thể được gọi lại
* Không có trạng thái nội bộ
* Không yêu cầu bất kỳ sự khởi tạo nào
* Được đồng bộ, tức là chúng không có delay.

|  |
| --- |
| **The following libraries are specified within AUTOSAR:**   * Fixed point mathematical * Floating point mathematical * Interpolation for fixed point data * Interpolation for floating point data * Extended functions (e.g. 64bits calculation, filtering, etc.) * Bit handling * E2E communication * CRC calculation * Atomic multicore safe operations |

## Content of Software Layers

### Microcontroller Abstraction Layer



**µC Abstraction Layer** (MCAL) bao gồm các nhóm mô-đun sau:

➢ **Microcontroller Drivers**

Trình điều khiển cho các bộ ngoại vi nội (ví dụ: Watchdog, General Purpose Timer)

Các chức năng có truy cập trực tiếp đến µC.

➢ **Communication Drivers**

Trình điều khiển cho việc giao tiếp trên ECU (ví dụ: SPI) và giao tiếp xe (ví dụ: CAN).

OSI-Layer: Data Link Layer.

**➢ Memory Drivers**

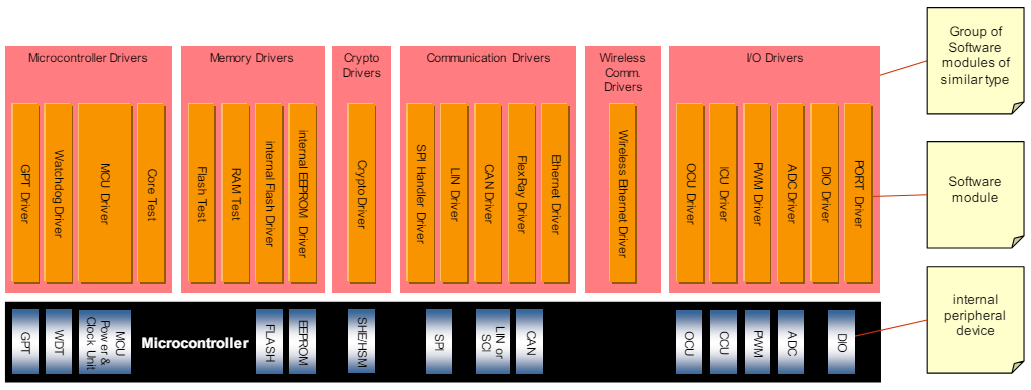
Trình điều khiển cho các thiết bị bộ nhớ trên chip (ví dụ: Flash internal, EEPROM internal) và các thiết bị bộ nhớ ngoại ánh xạ bộ nhớ (ví dụ: Flash external)

**➢ I/O Drivers**

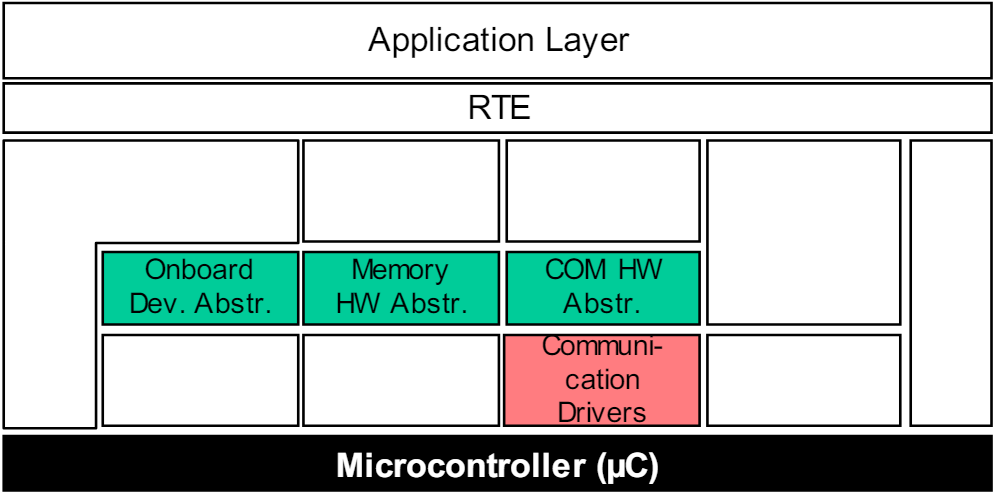
Trình điều khiển cho analog và digital I/O (ví dụ: ADC, PWM, DIO)

**➢ Crypto Drivers** Mã hóa Trình điều khiển cho các thiết bị mã hóa trên chip như SHE hoặc HSM

➢ **Wireless Communication Drivers**: Trình điều khiển cho các hệ thống mạng không dây (giao tiếp trong xe hoặc giao tiếp ngoài ECU)

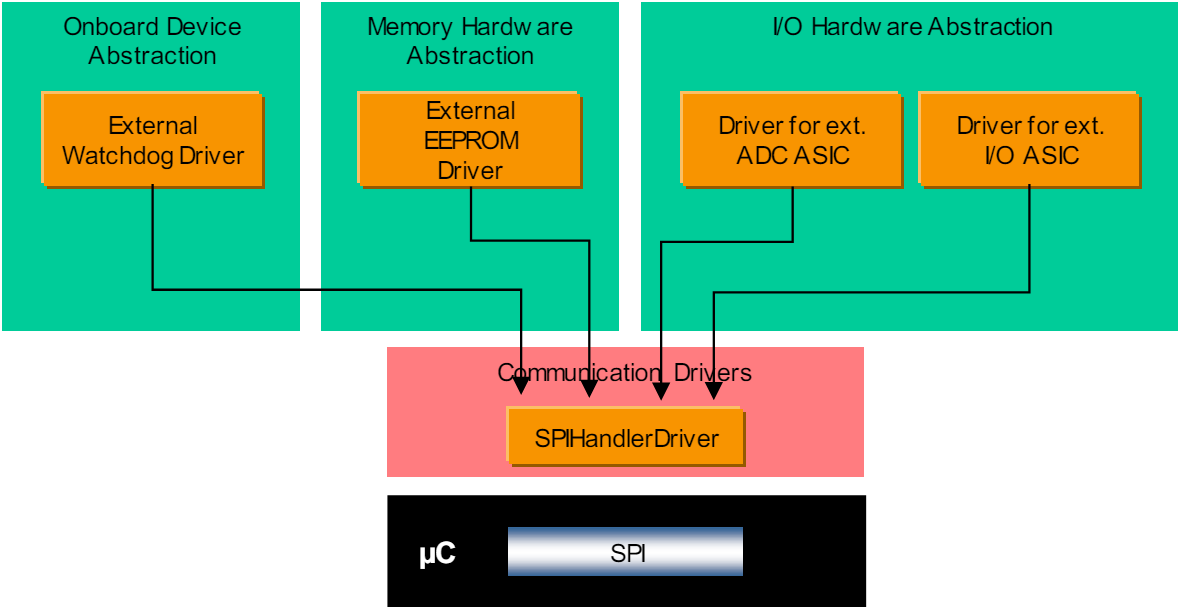


### Microcontroller Abstraction Layer: SPIHandlerDriver

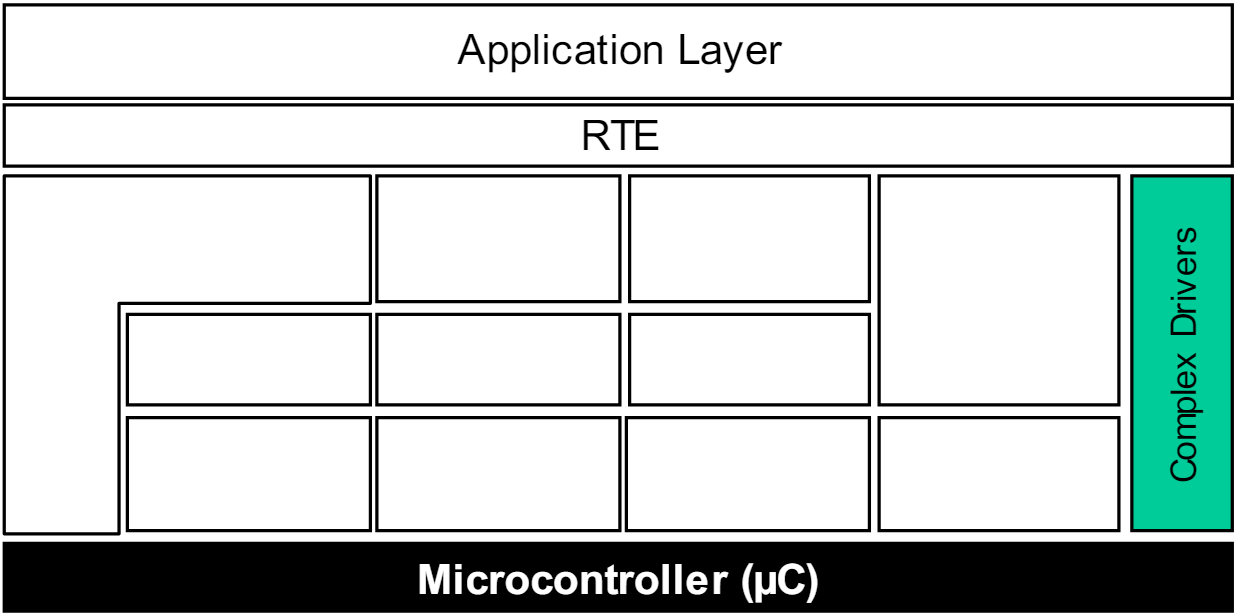


**SPIHandlerDriver** cho phép truy cập đồng thời của nhiều khách hàng vào một hoặc nhiều bus SPI.

Để trừu tượng hóa tất cả các tính năng của một vi điều khiển SPI có các chân được dành riêng cho Chip Select, những chân đó sẽ được xử lý trực tiếp bởi SPIHandlerDriver. Điều đó có nghĩa là những chân đó sẽ không có sẵn trong DIO Driver.



### Complex Drivers



Một **Complex Driver** là một mô-đun thực hiện chức năng không được chuẩn hóa trong basic software stack.

Một ví dụ là triển khai đánh giá cảm biến phức tạp và kiểm soát actuator với truy cập trực tiếp đến µC bằng các ngắt cụ thể và/hoặc các µC phức tạp (như PCP, TPU), ví dụ:

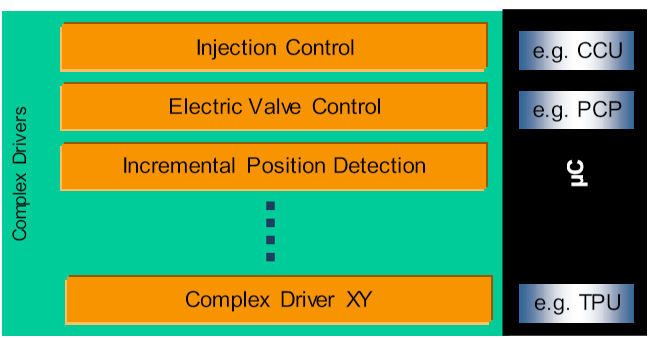
* Kiểm soát phun nhiên liệu
* Kiểm soát van điện
* Phát hiện vị trí tăng dần

***Nhiệm vụ:***

* Đáp ứng các yêu cầu chức năng và thời gian đặc biệt cho việc xử lý cảm biến và actuator phức tạp

***Thuộc tính:***

* Triển khai: phụ thuộc nhiều vào µC, ECU và Application
* Giao diện trên cùng đến SW-Cs: được chỉ định và triển khai theo AUTOSAR (AUTOSAR interface)
* Giao diện dưới cùng: truy cập hạn chế vào các Giao diện Chuẩn hóa (Standardized Interfaces).



### ECU Abstraction: I/O Hardware Abstraction



**I/O Hardware Abstraction** là một nhóm các mô-đun trừu tượng hóa **location** của các thiết bị I/O ngoại vi (trên chip hoặc trên bo mạch) và ECU hardware layout (ví dụ: kết nối chân vi điều khiển và đảo ngược mức tín hiệu). **I/O Hardware Abstraction** không trừu tượng hóa từ các cảm biến/bộ điều khiển!

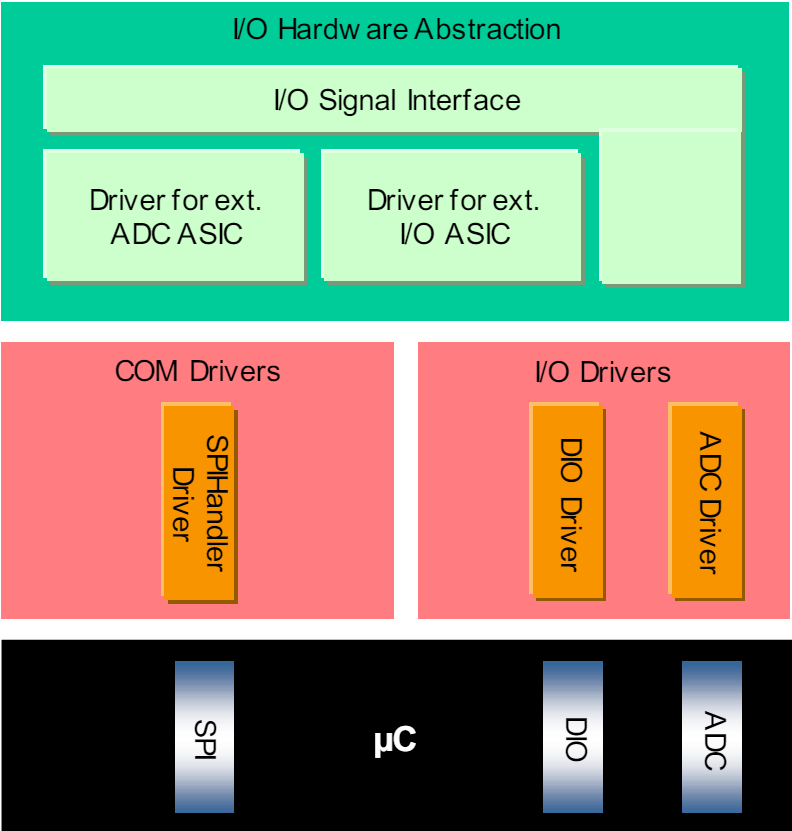
Các thiết bị I/O khác nhau có thể được truy cập thông qua một giao diện tín hiệu I/O.

***Nhiệm vụ:***

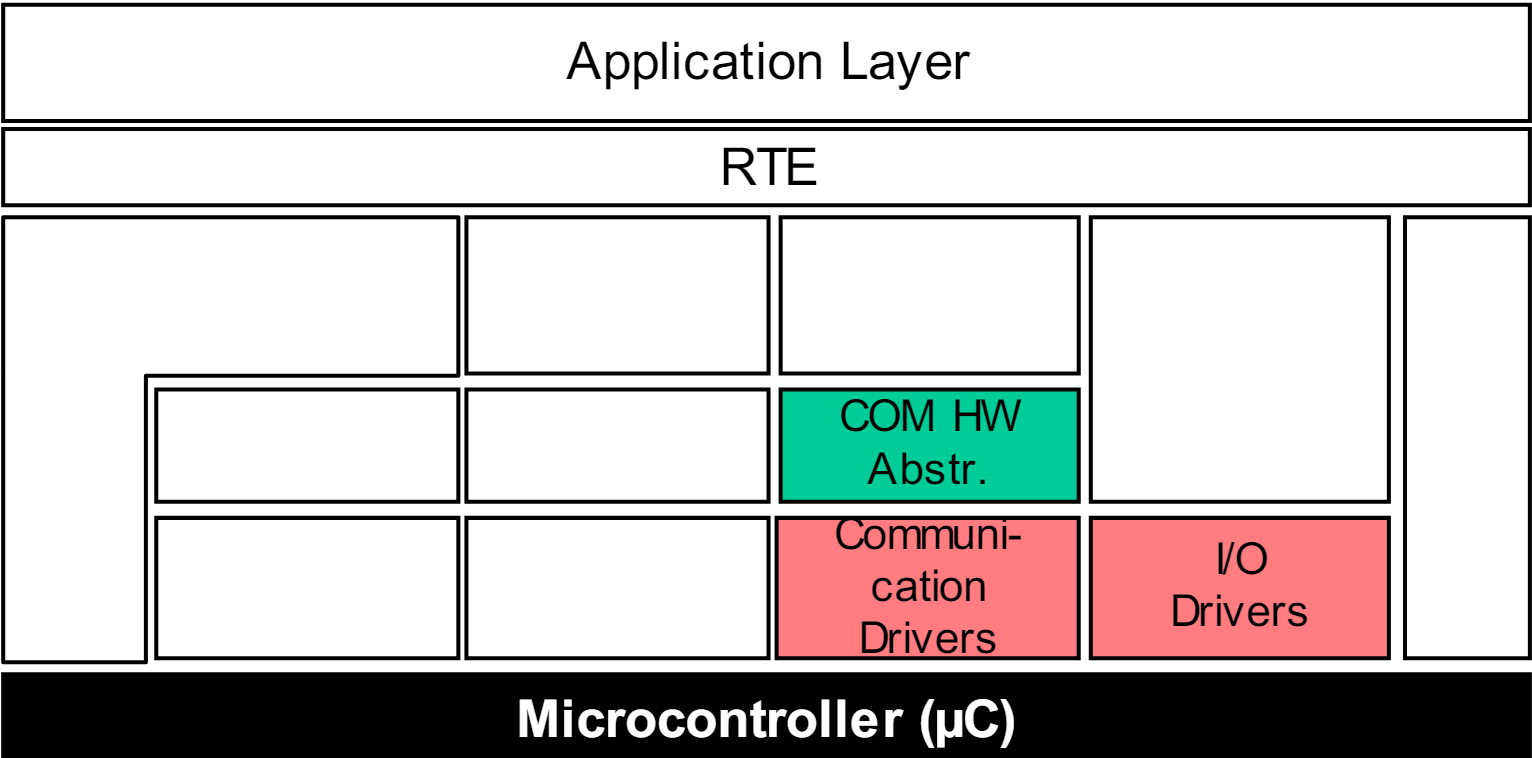
* Đại diện cho các tín hiệu I/O như chúng được kết nối với phần cứng của ECU hardware (e.g. current, voltage, frequency).
* Ẩn các thuộc tính bố cục và phần cứng ECU từ mức cao hơn lớp phần mềm

***Thuộc tính:***

* Triển khai: độc lập µC, phụ thuộc vào ECU hardware.
* Giao diện trên cùng: độc lập với µC and ECU hardware, phụ thuộc vào loại tín hiệu được chỉ định và triển khai theo AUTOSAR (AUTOSAR interface).



### ECU Abstraction: Communication Hardware Abstraction



Lớp Trừu tượng Phần cứng Giao tiếp là một nhóm các mô-đun trừu tượng hóa vị trí của các bộ điều khiển giao tiếp và bố cục phần cứng của Đơn vị Điều khiển Điện tử. Đối với tất cả các hệ thống giao tiếp, một Lớp Trừu tượng Phần cứng Giao tiếp cụ thể là cần thiết (ví dụ: cho LIN, CAN, FlexRay).

Ví dụ: Một Đơn vị Điều khiển Điện tử có một vi điều khiển vi xử lý với 2 kênh CAN nội và một ASIC trên bo mạch với 4 bộ điều khiển CAN. ASIC CAN được kết nối với vi điều khiển vi xử lý qua giao tiếp SPI.

Các trình điều khiển giao tiếp được truy cập thông qua các giao diện cụ thể của bus (ví dụ: Giao diện CAN).

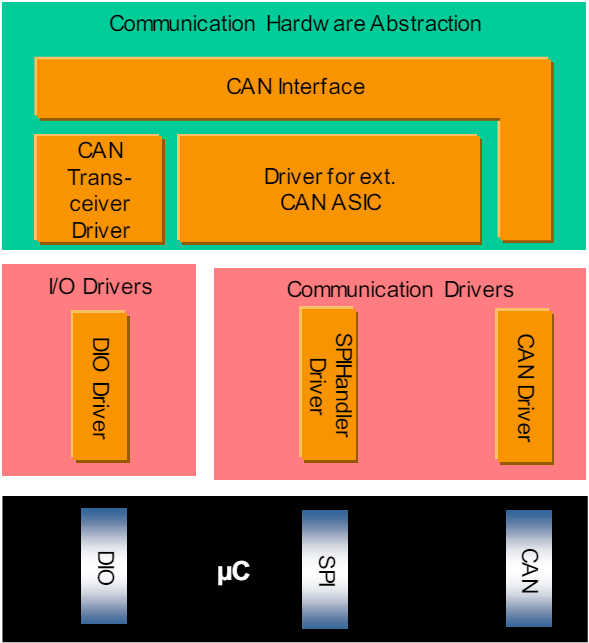
Nhiệm vụ:

Cung cấp cơ chế bình đẳng để truy cập một kênh bus mà không phụ thuộc vào vị trí của nó (trên chip / trên bo mạch)

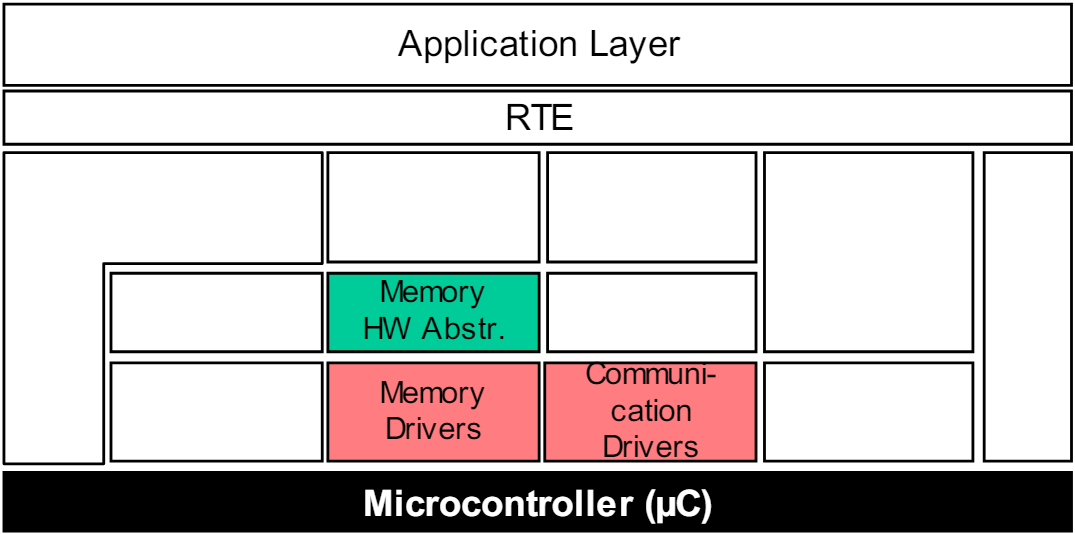
Thuộc tính:

Triển khai: độc lập với vi điều khiển, phụ thuộc vào phần cứng của Đơn vị Điều khiển Điện tử và thiết bị ngoại vi

Giao diện trên cùng: phụ thuộc vào bus, độc lập với vi điều khiển và phần cứng của Đơn vị Điều khiển Điện tử



### Scope: Memory Hardware Abstraction



Lớp Trừu tượng Phần cứng Bộ nhớ là một nhóm các mô-đun trừu tượng hóa vị trí của các thiết bị bộ nhớ ngoại vi (trên chip hoặc trên bo mạch) và bố cục phần cứng của Đơn vị Điều khiển Điện tử.

Ví dụ: các thiết bị EEPROM trên chip và EEPROM ngoại vi có thể truy cập thông qua cùng một cơ chế.

Các trình điều khiển bộ nhớ được truy cập thông qua các mô-đun trừu tượng/giả lập cụ thể của bộ nhớ (ví dụ: Trừu tượng EEPROM).

Bằng cách giả lập một trừu tượng EEPROM trên các đơn vị phần cứng Flash, một cách truy cập chung thông qua Giao diện Trừu tượng Bộ nhớ đến cả hai loại phần cứng được kích hoạt.

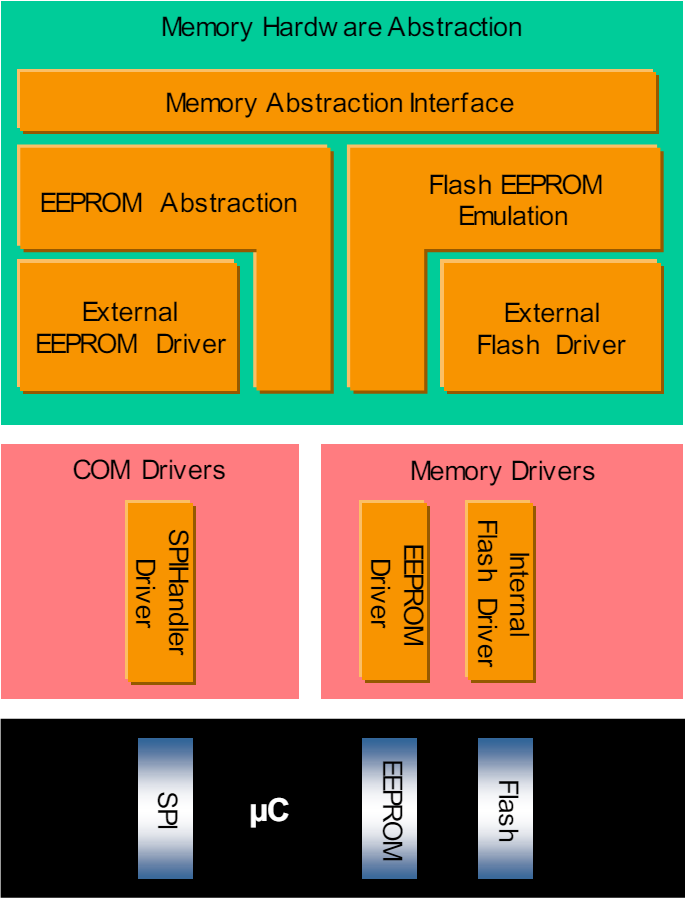
Nhiệm vụ:

Cung cấp cơ chế bình đẳng để truy cập các thiết bị bộ nhớ nội (trên chip) và ngoại (trên bo mạch) và loại phần cứng bộ nhớ (EEPROM, Flash).

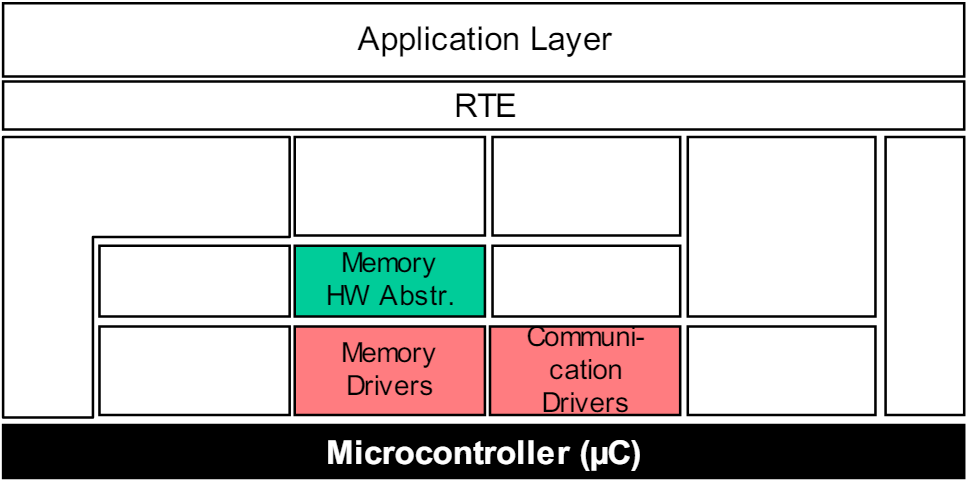
Thuộc tính:

Triển khai: độc lập với vi điều khiển, phụ thuộc vào thiết bị ngoại vi

Giao diện trên cùng: độc lập với vi điều khiển, phần cứng của Đơn vị Điều khiển Điện tử và thiết bị bộ nhớ



### Scope: Memory Hardware Abstraction {DRAFT}



Lớp Trừu tượng Phần cứng Bộ nhớ là một nhóm các mô-đun trừu tượng hóa vị trí của các thiết bị bộ nhớ ngoại vi (trên chip hoặc trên bo mạch) và bố cục phần cứng của Đơn vị Điều khiển Điện tử.

Ví dụ: các thiết bị EEPROM trên chip và EEPROM ngoại vi có thể truy cập thông qua cùng một cơ chế.

Các trình điều khiển bộ nhớ được truy cập thông qua các mô-đun trừu tượng/giả lập cụ thể của bộ nhớ (ví dụ: Trừu tượng EEPROM).

Bằng cách giả lập một trừu tượng EEPROM trên các đơn vị phần cứng Flash, một cách truy cập chung thông qua Giao diện Trừu tượng Bộ nhớ đến cả hai loại phần cứng được kích hoạt.

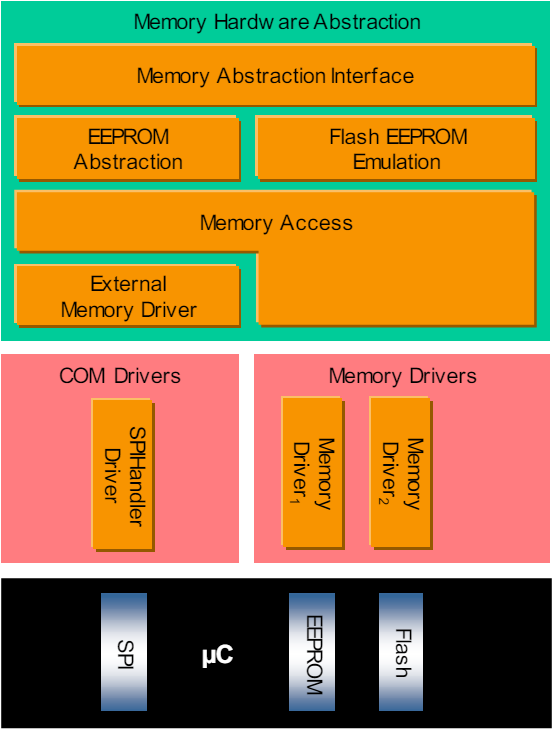
Nhiệm vụ:

Cung cấp cơ chế bình đẳng để truy cập các thiết bị bộ nhớ nội (trên chip) và ngoại (trên bo mạch) và loại phần cứng bộ nhớ (EEPROM, Flash).

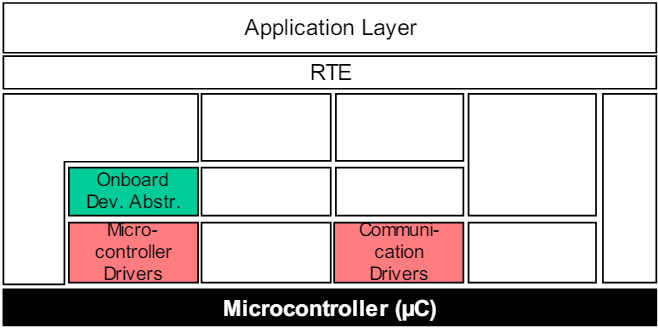
Thuộc tính:

Triển khai: độc lập với vi điều khiển, phụ thuộc vào thiết bị ngoại vi

Giao diện trên cùng: độc lập với vi điều khiển, phần cứng của Đơn vị Điều khiển Điện tử và thiết bị bộ nhớ



### Onboard Device Abstraction



Lớp Trừu tượng Thiết bị Trên bo mạch chứa các trình điều khiển cho các thiết bị trên bo mạch của Đơn vị Điều khiển Điện tử mà không thể coi là cảm biến hoặc actuator như các watchdog nội hoặc ngoại. Những trình điều khiển đó truy cập các thiết bị trên bo mạch của Đơn vị Điều khiển Điện tử thông qua Lớp Trừu tượng Bộ điều khiển vi xử lý.

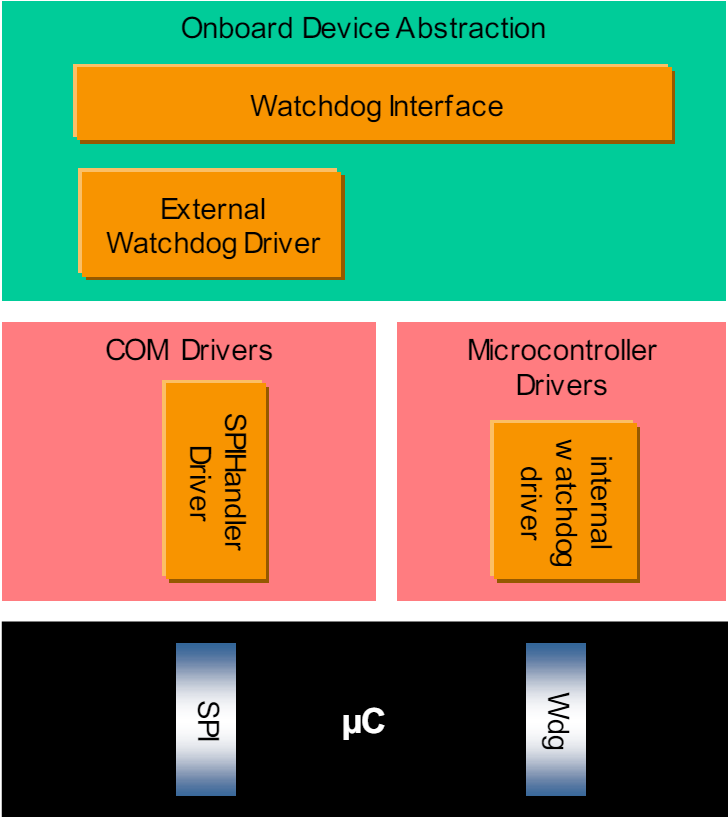
Nhiệm vụ:

Từ trừu tượng hóa các thiết bị trên bo mạch cụ thể của Đơn vị Điều khiển Điện tử.

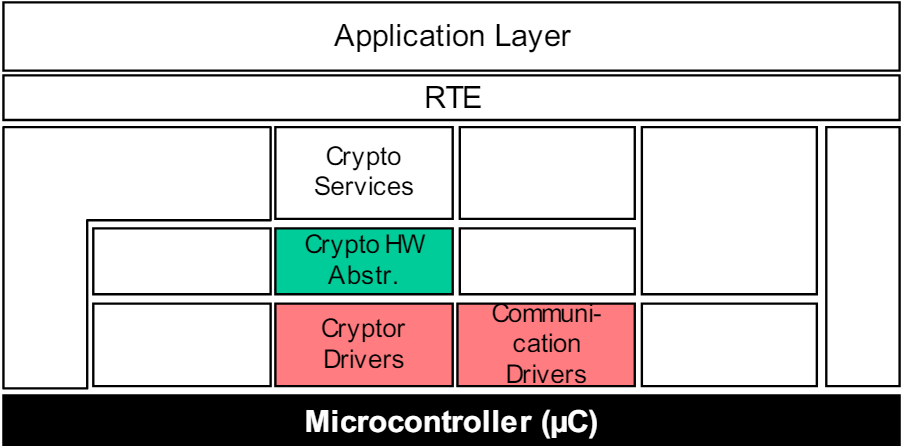
Thuộc tính:

Triển khai: độc lập với vi điều khiển, phụ thuộc vào thiết bị ngoại vi

Giao diện trên cùng: độc lập với vi điều khiển, một phần phụ thuộc vào phần cứng của Đơn vị Điều khiển Điện tử



### Scope: Crypto Hardware Abstraction



Lớp Trừu tượng Phần cứng Mã hóa là một nhóm các mô-đun trừu tượng hóa vị trí của các nguyên tắc mật mã (phần cứng nội hoặc ngoại vi hoặc dựa trên phần mềm).

Ví dụ: Nguyên tắc AES được thực hiện trong SHE hoặc được cung cấp dưới dạng thư viện phần mềm.

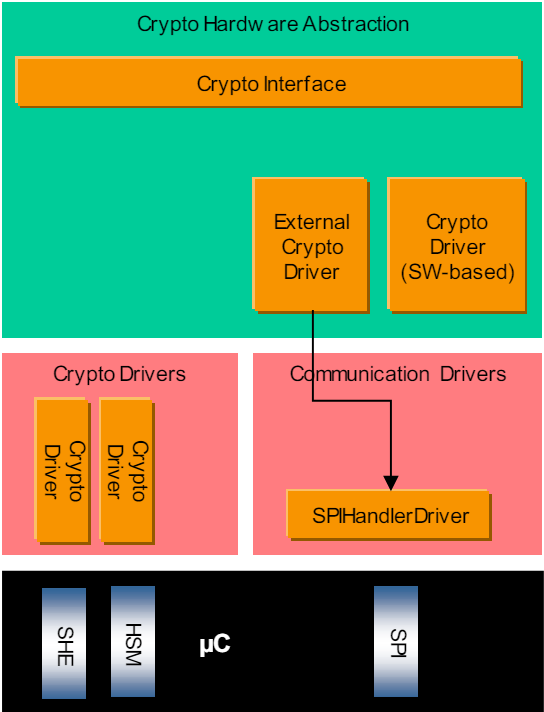
Nhiệm vụ:

Cung cấp cơ chế bình đẳng để truy cập các thiết bị mật mã nội (trên chip) và phần mềm.

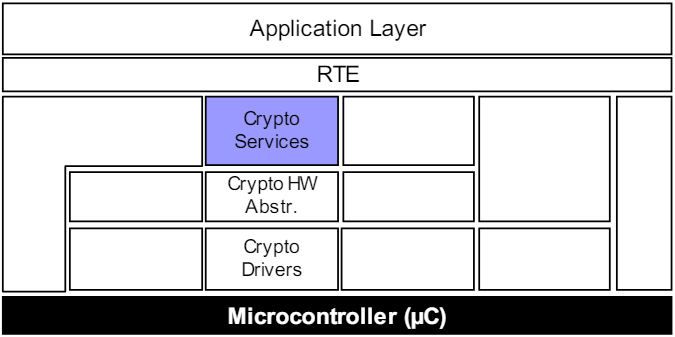
Thuộc tính:

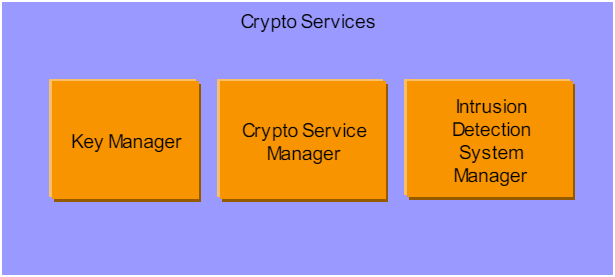
Triển khai: độc lập với vi điều khiển

Giao diện trên cùng: độc lập với vi điều khiển, phần cứng của Đơn vị Điều khiển Điện tử và thiết bị mật mã

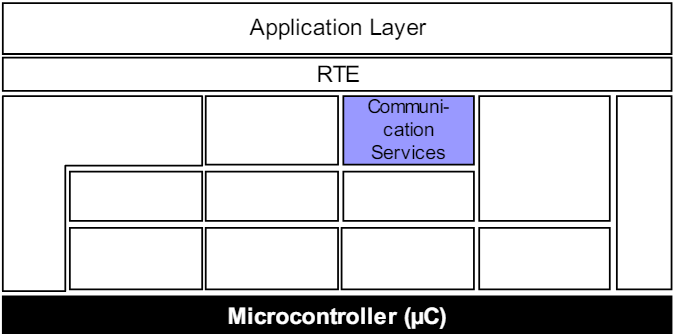


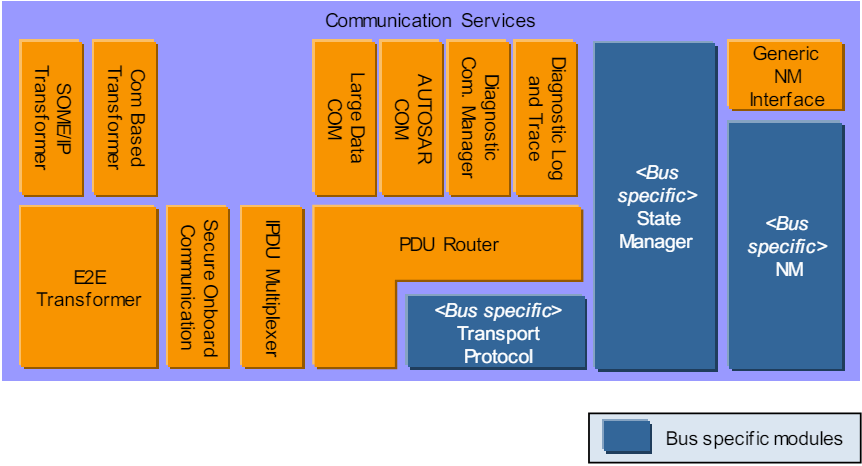
### Services: Crypto Services



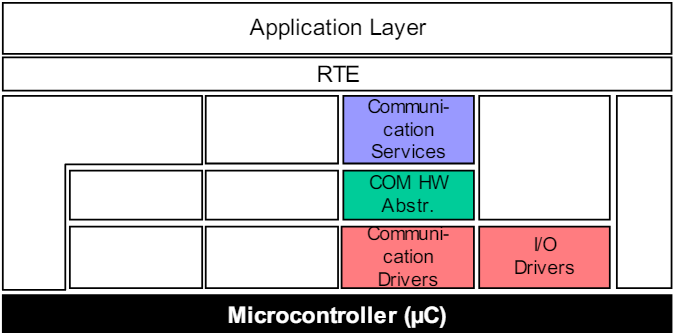


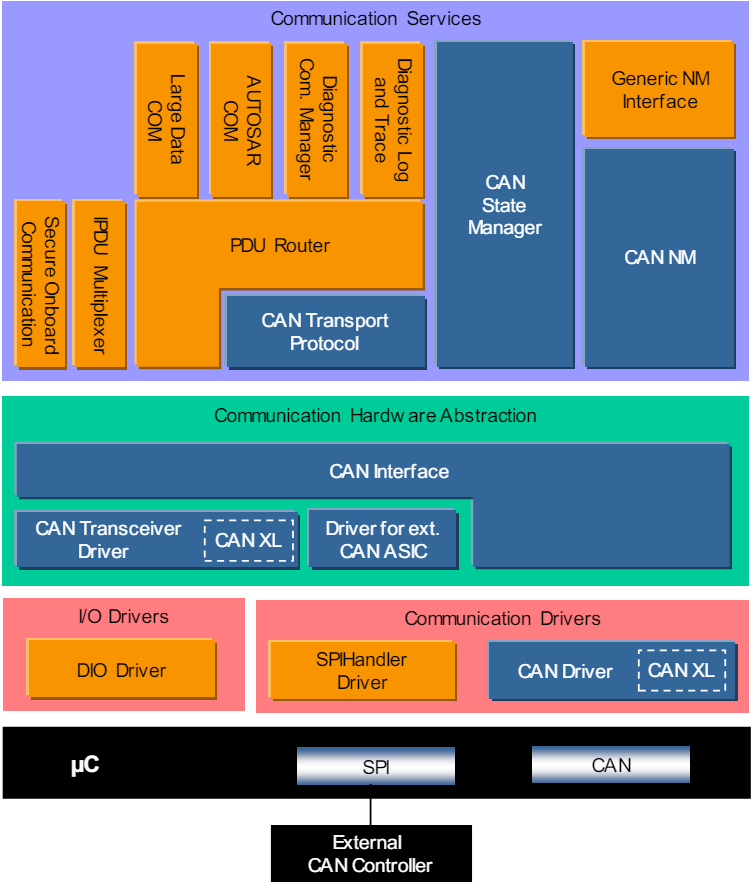
### Communication Services – General





### Communication Stack – CAN

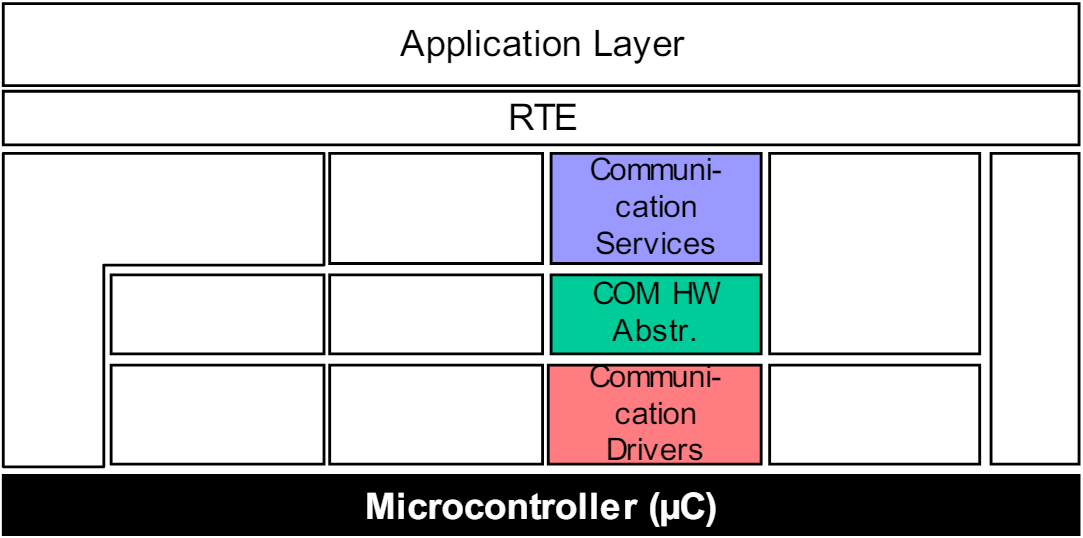


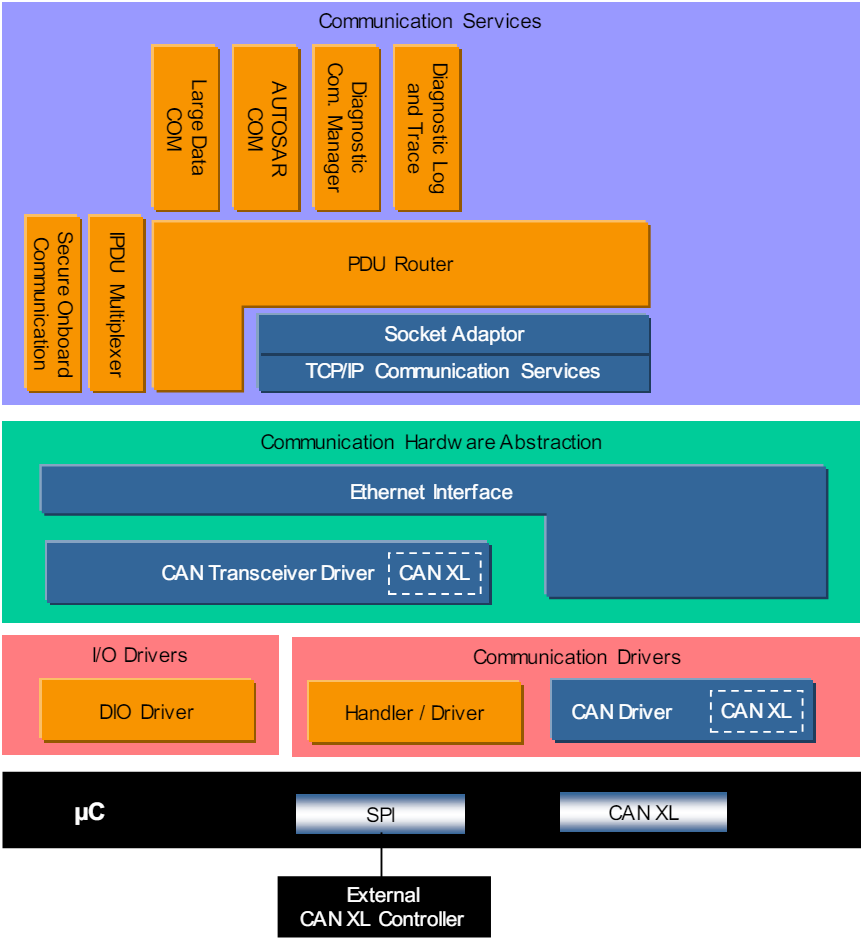


### Communication Stack – CAN

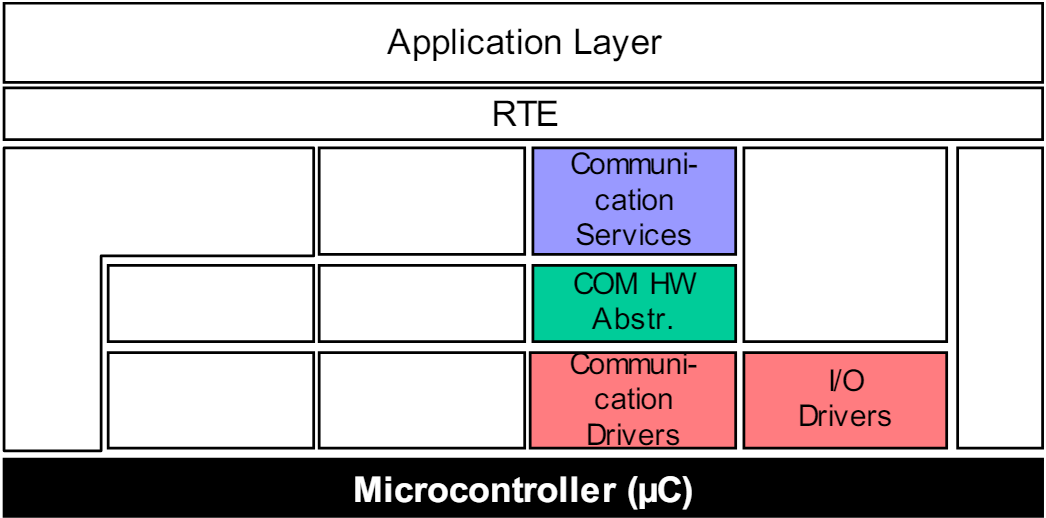


### Communication Stack – Ethernet/CAN XL

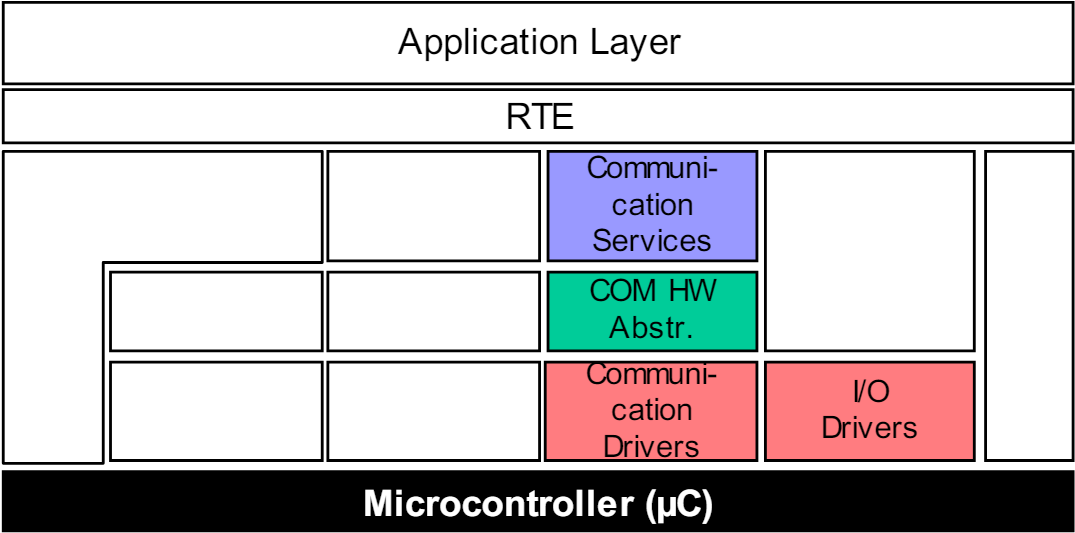


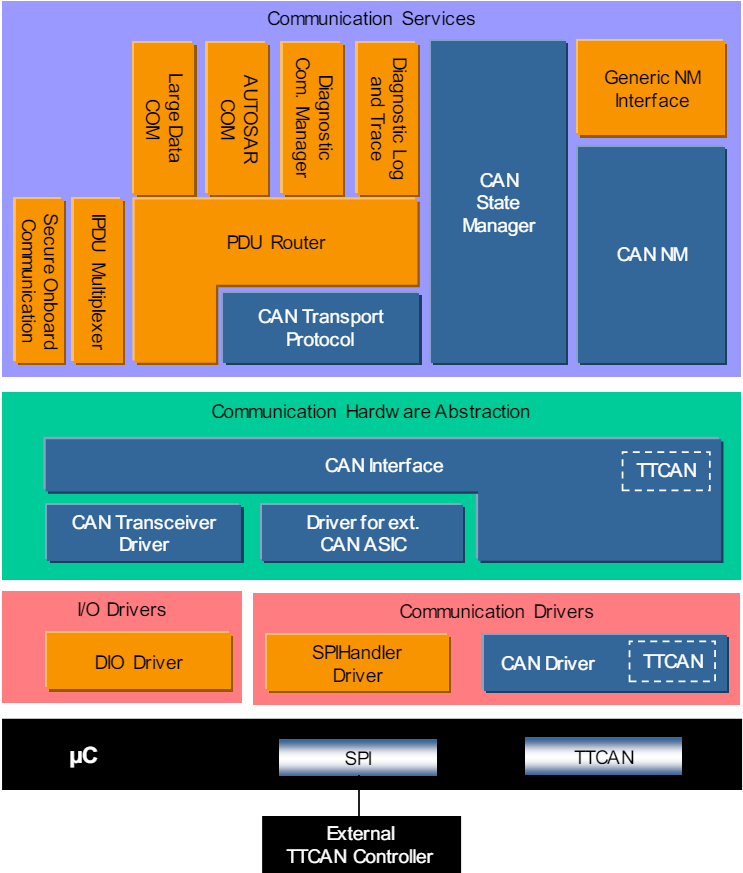


### Communication Stack Extension – CAN XL

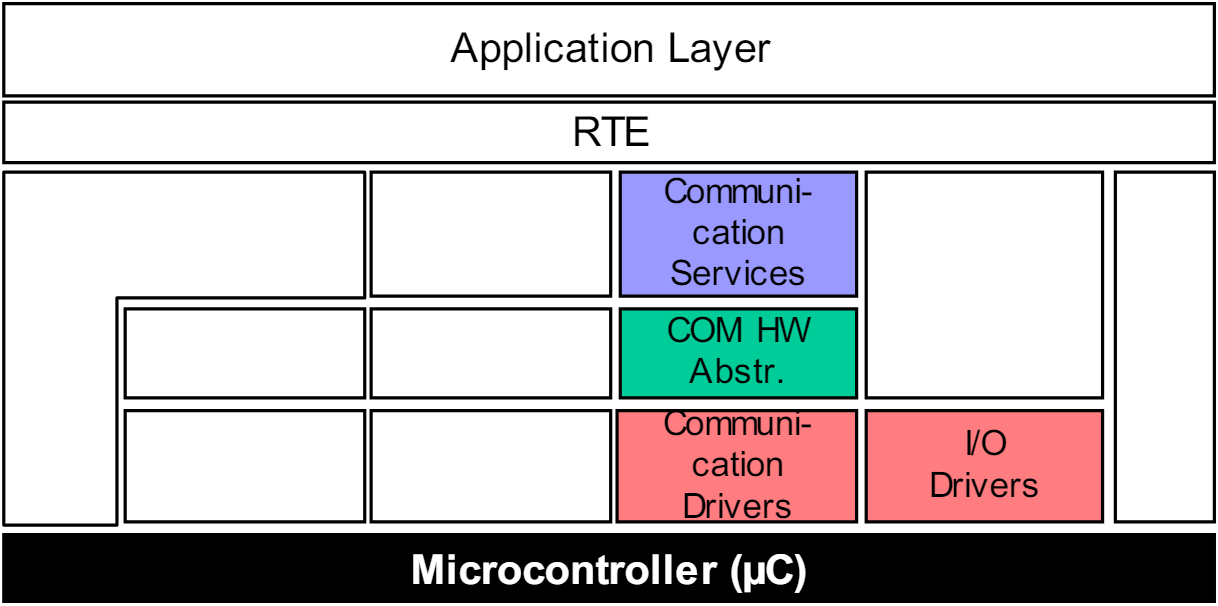


### Communication Stack Extension – TTCAN

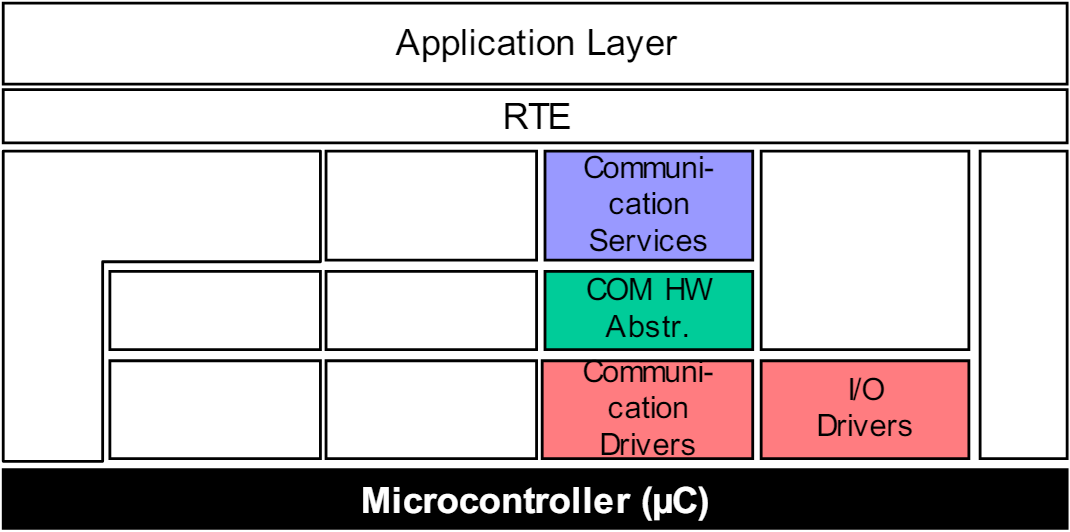


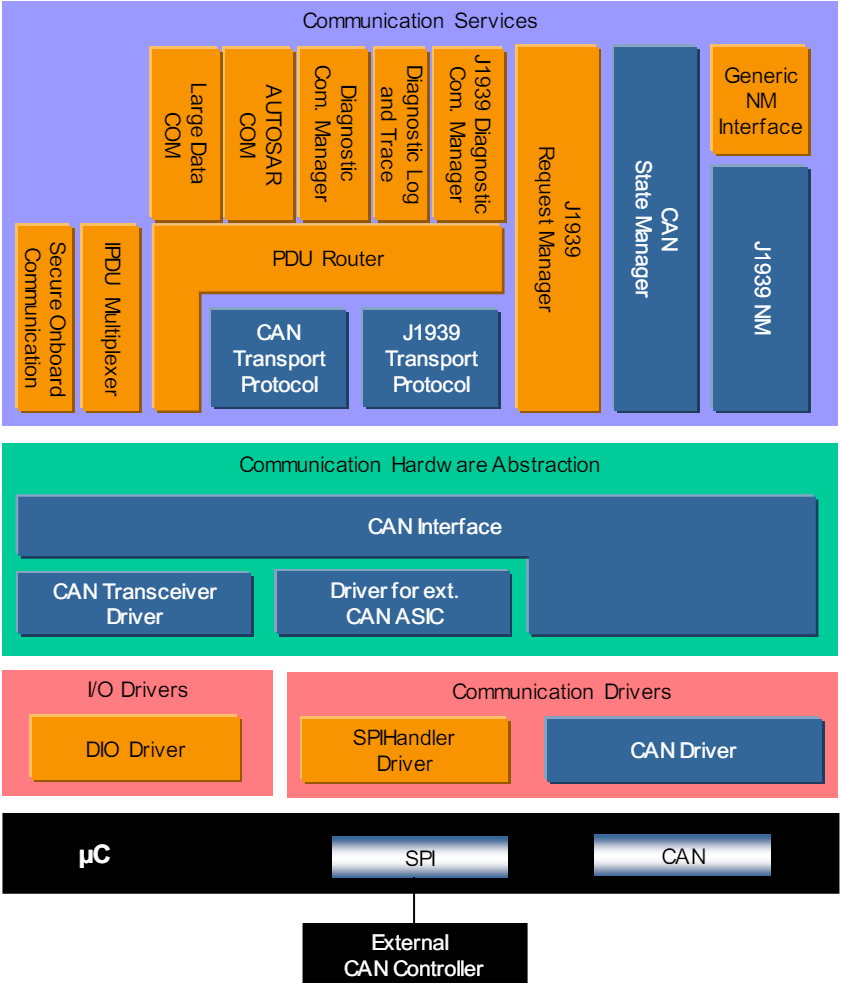


### Communication Stack Extension – TTCAN

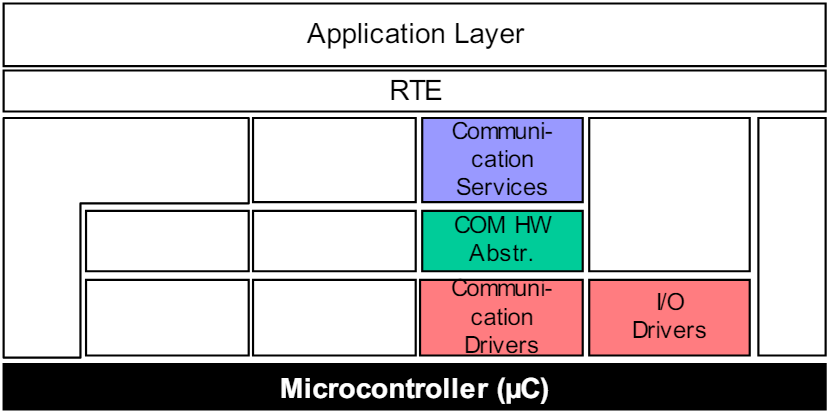


### Communication Stack Extension – J1939

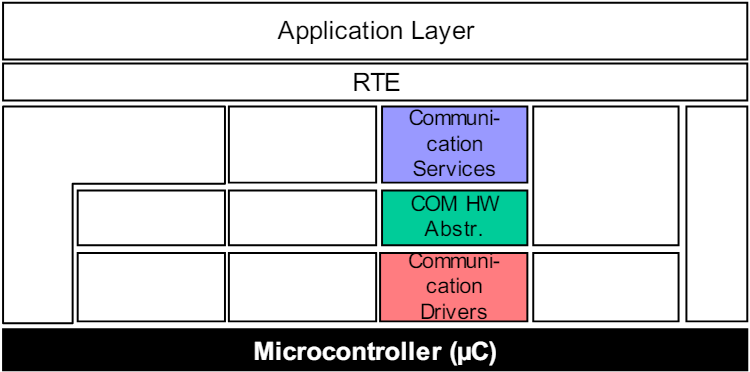


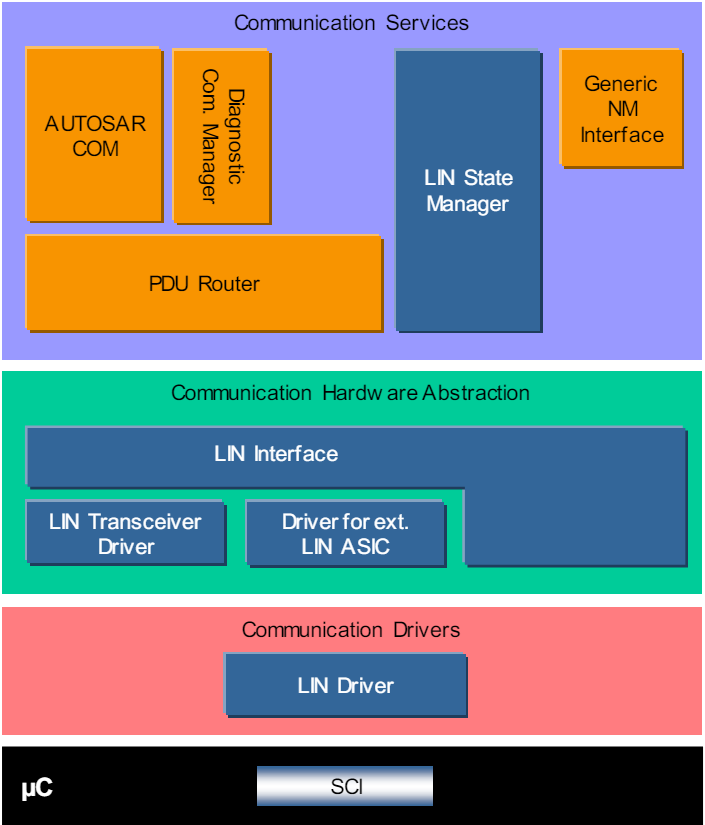


### Communication Stack Extension – J1939

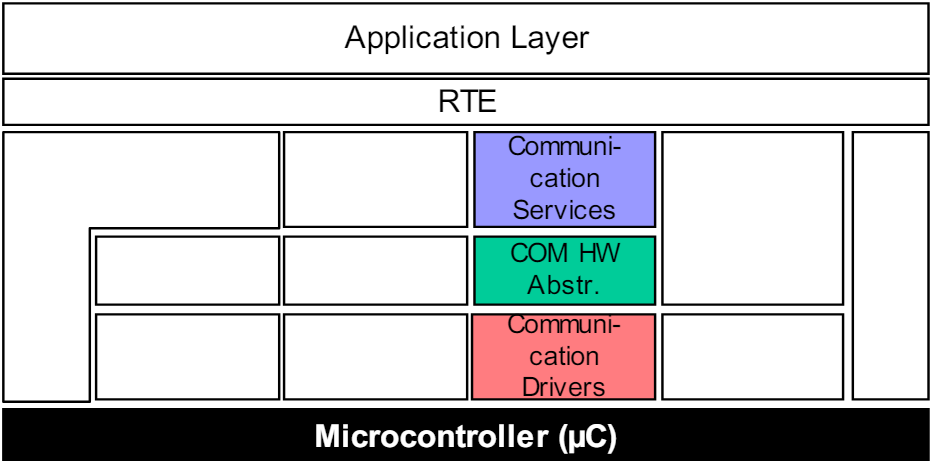


### Communication Stack – LIN

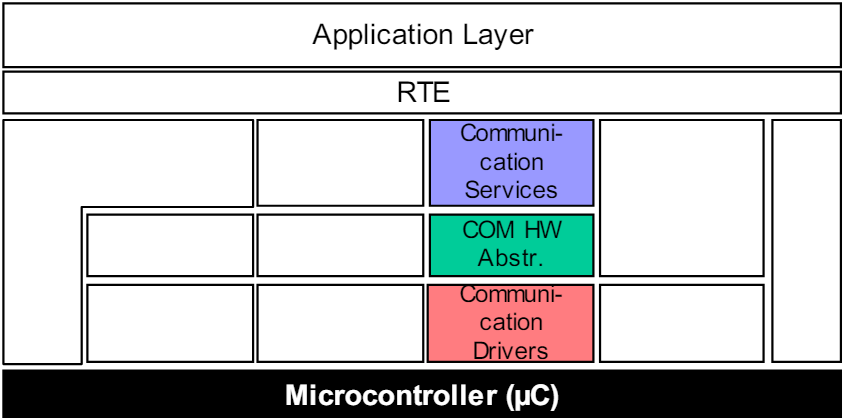


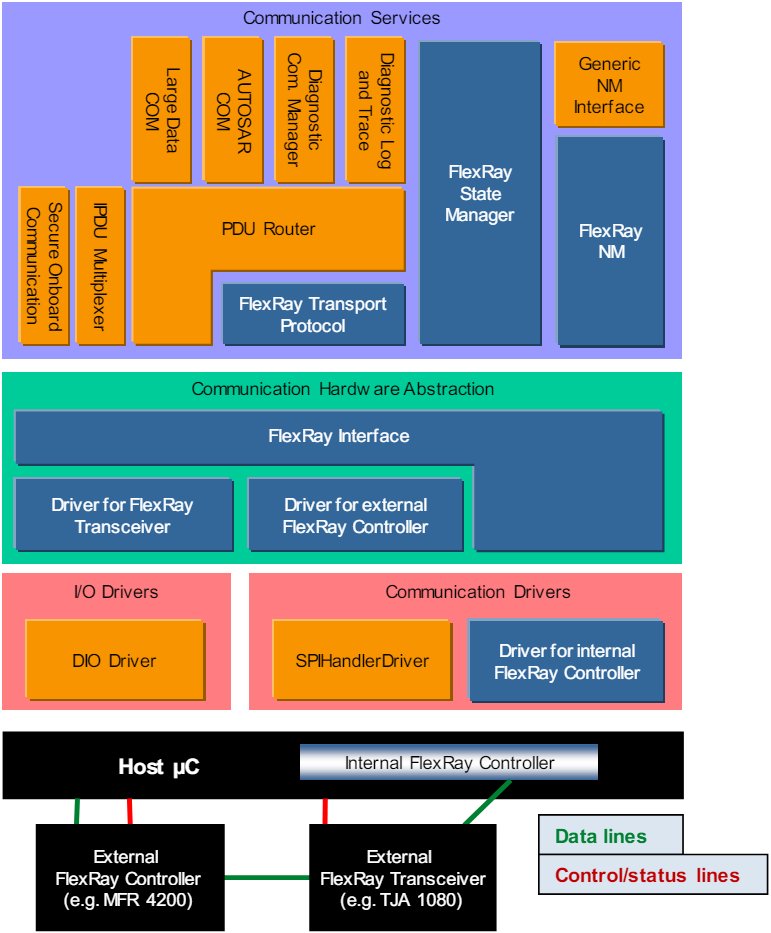


### Communication Stack – LIN

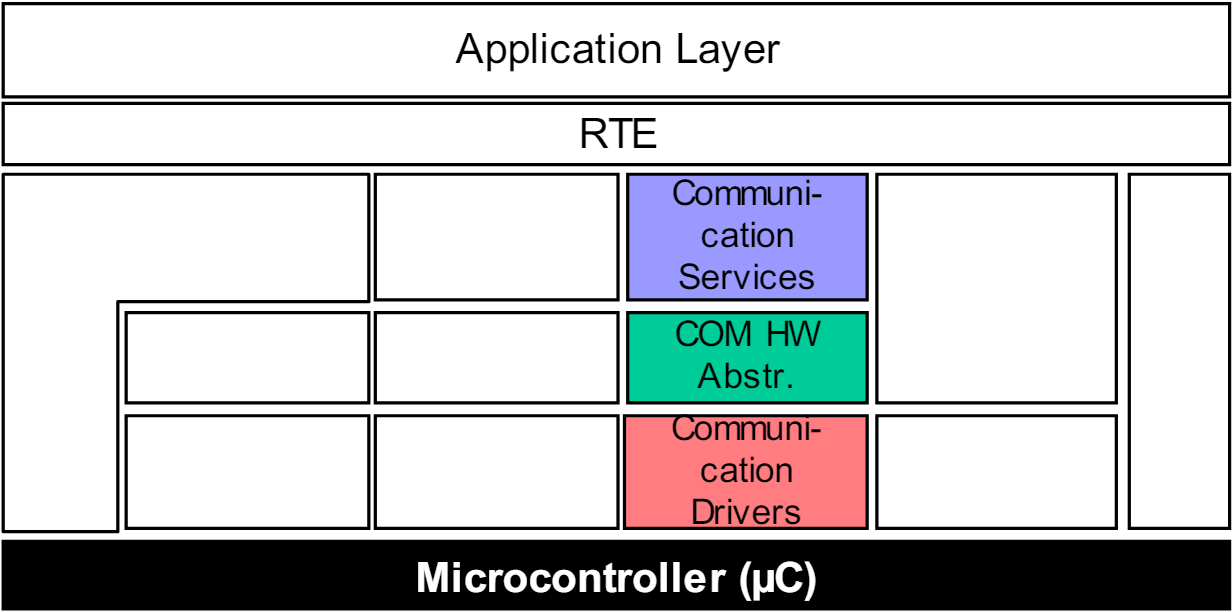


### Communication Stack – FlexRay



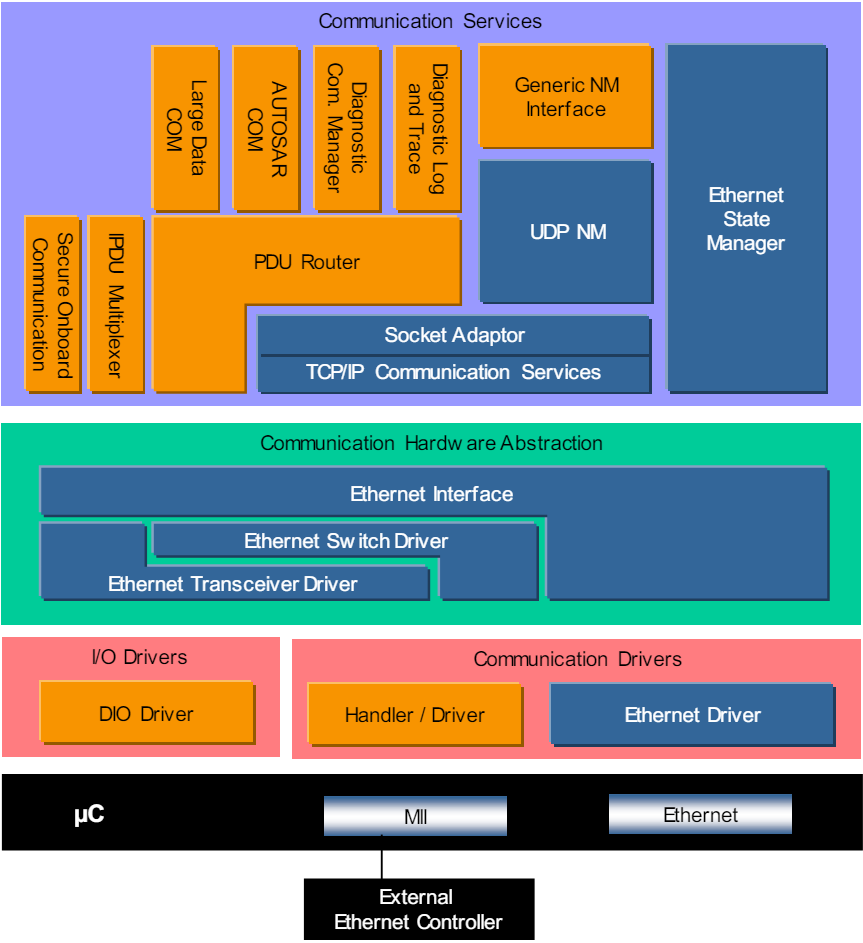


### Communication Stack – FlexRay

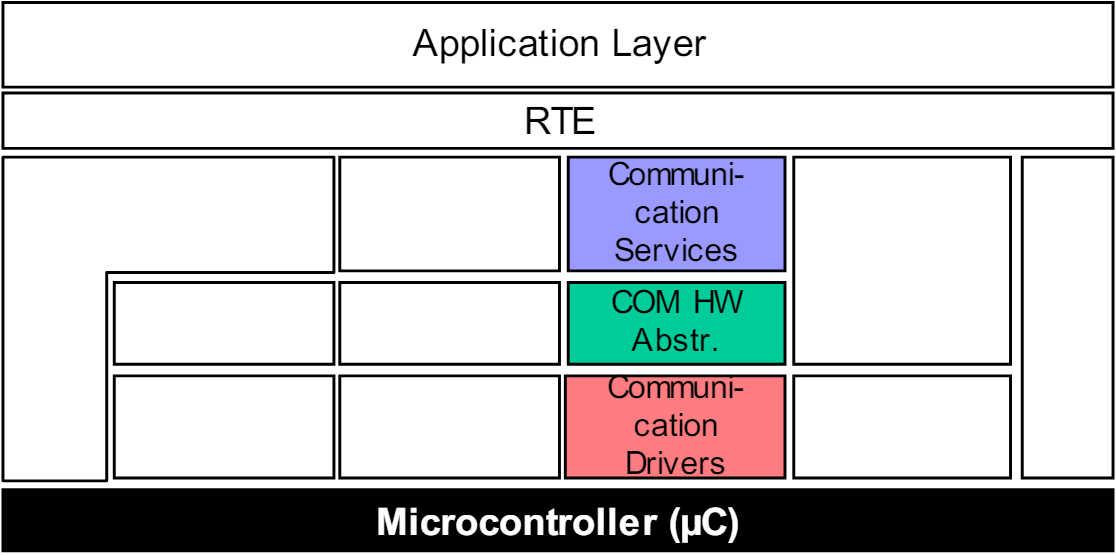


### Communication Stack – TCP/IP

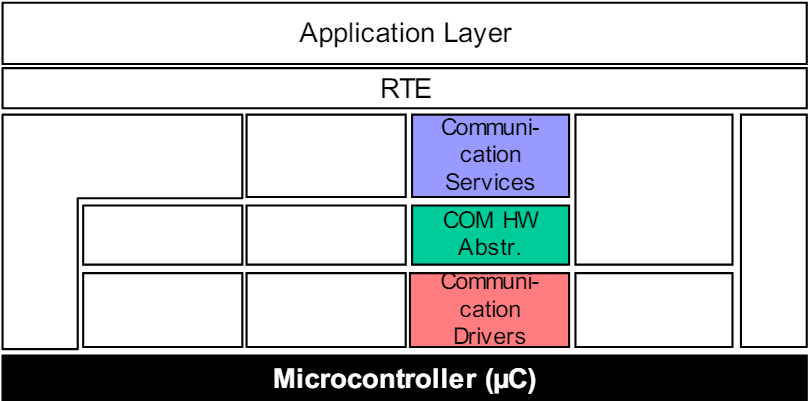


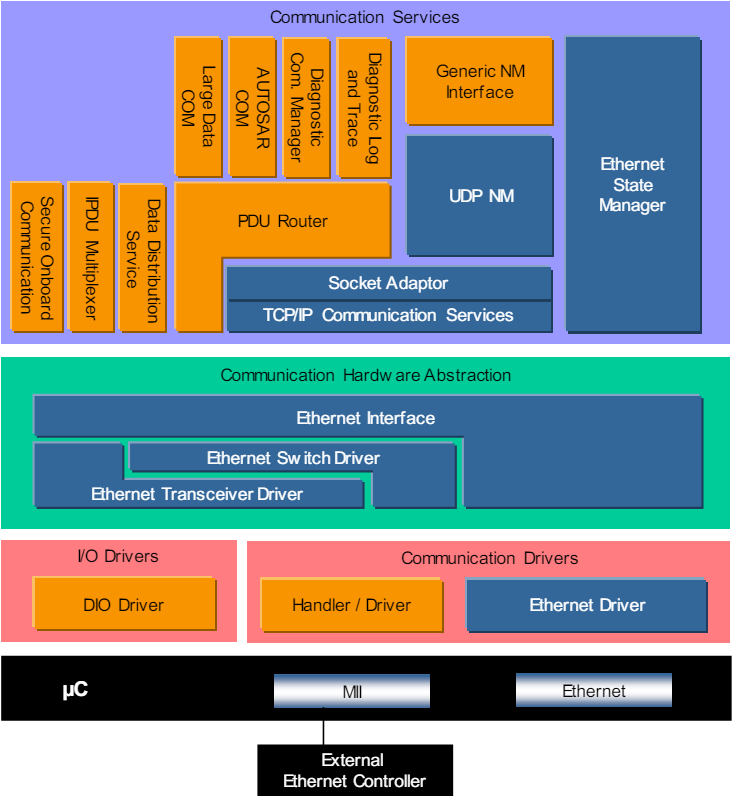


### Communication Stack – TCP/IP

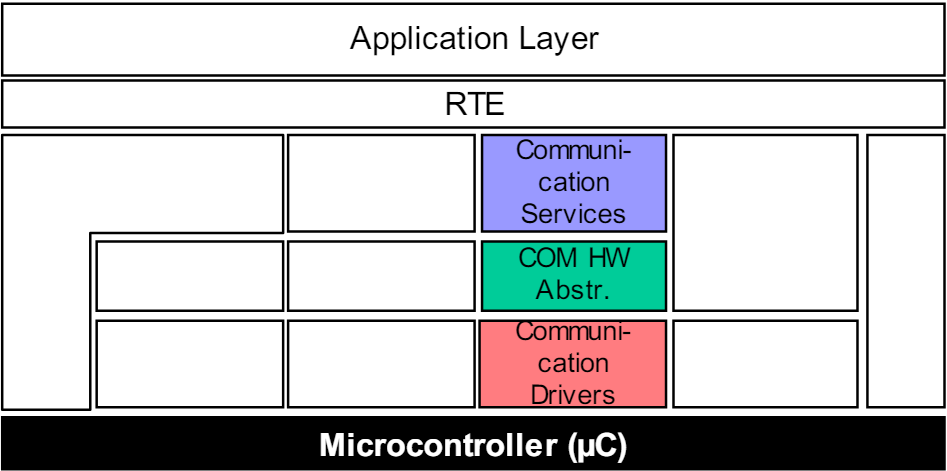


### Communication Stack – DDS

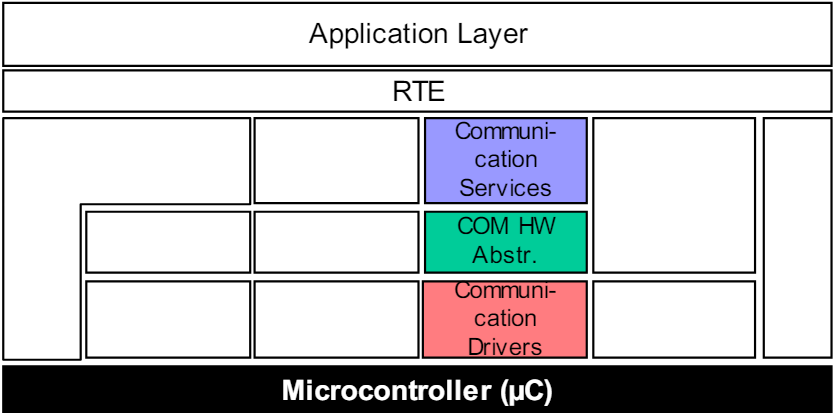




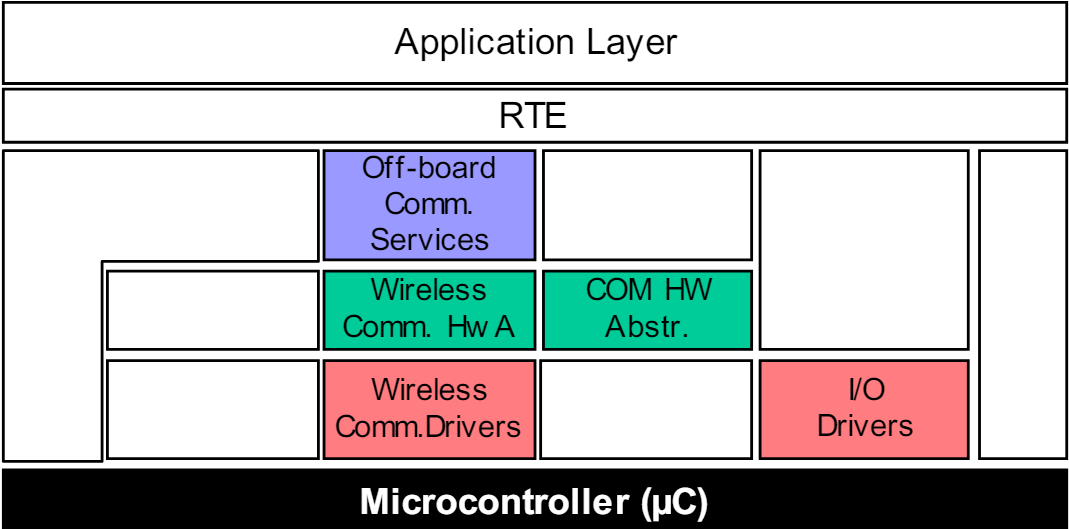
### Communication Stack – DDS

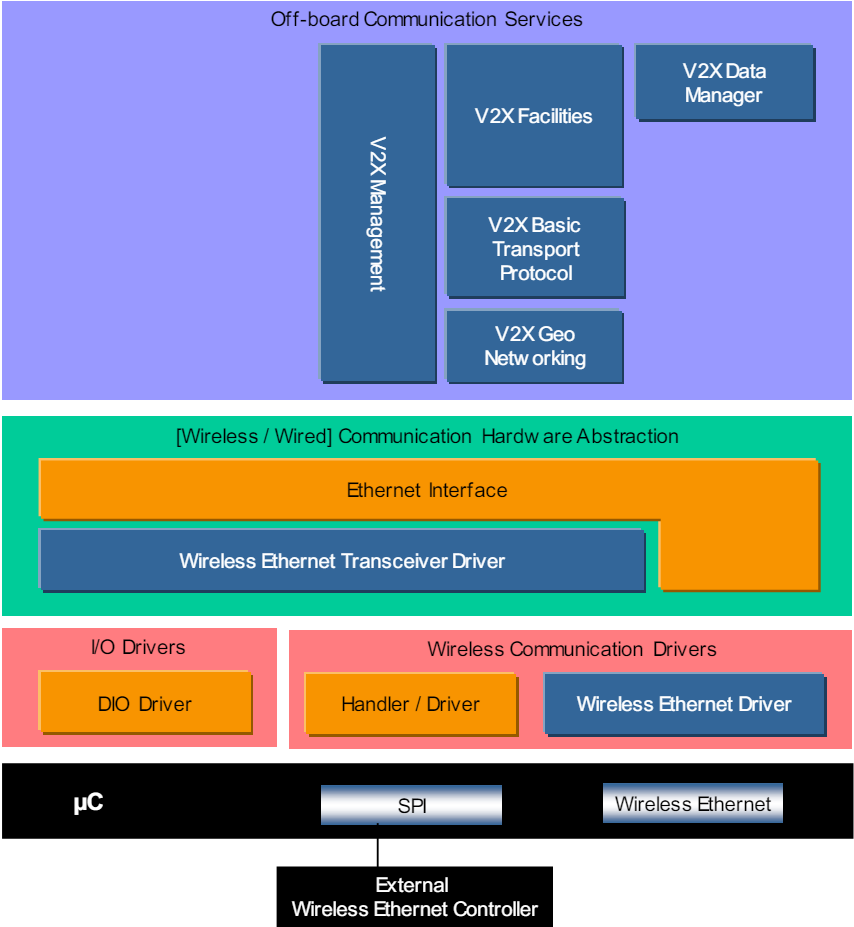


### Communication Stack – General

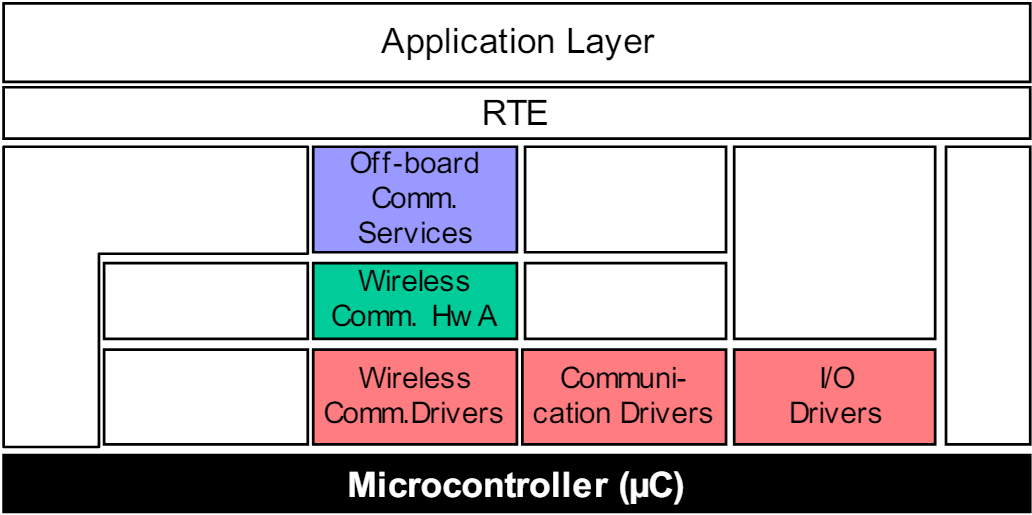


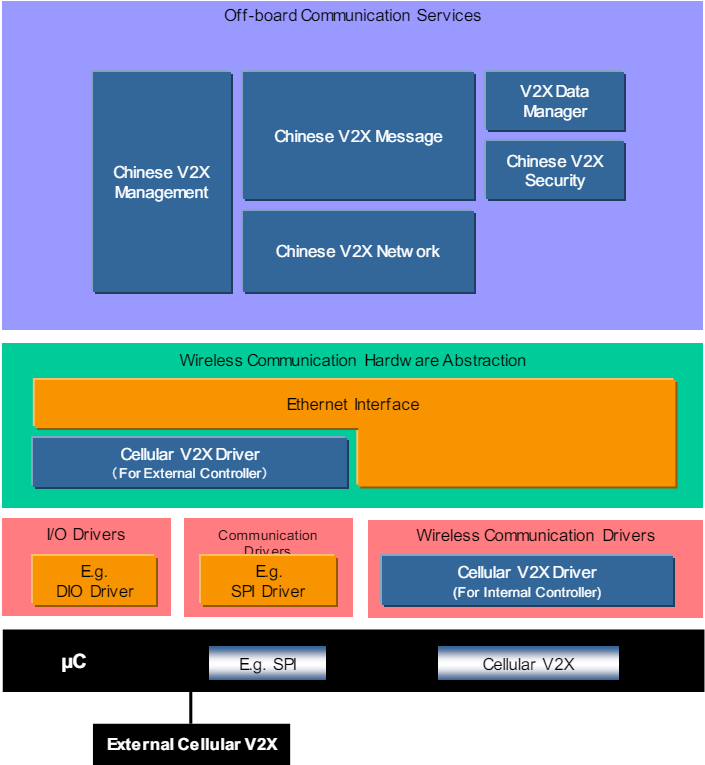
### Off-board Communication Stack – European Vehicle-2-X



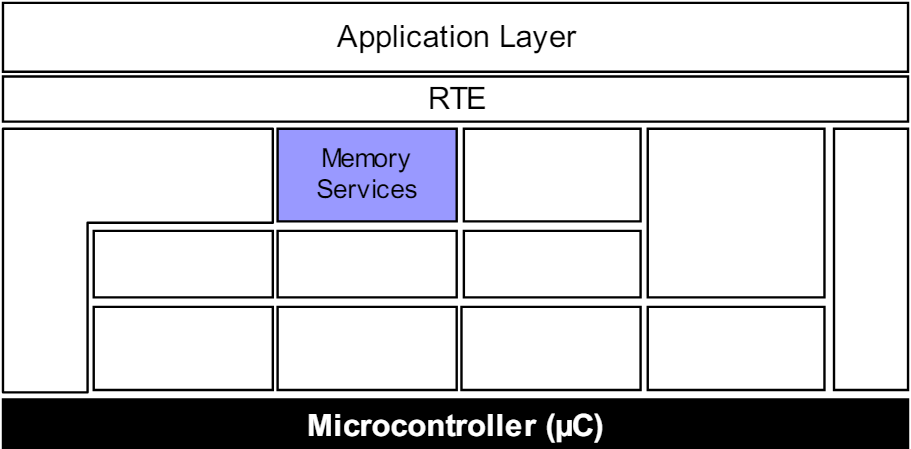


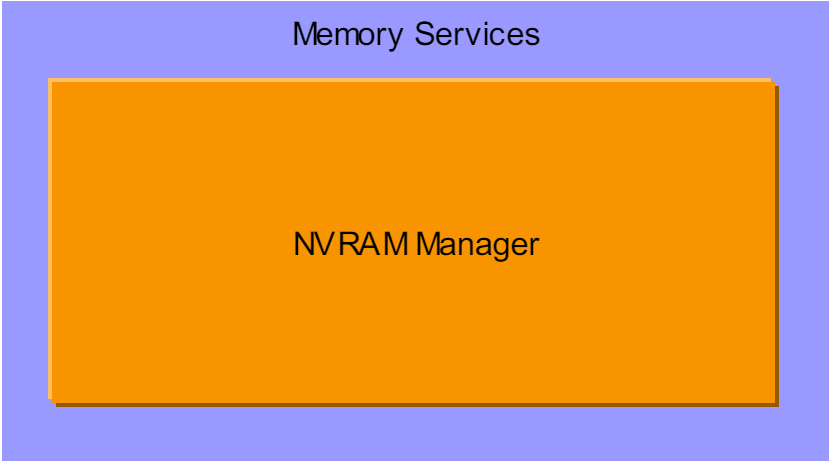
### Off-board Communication Stack – Chinese Vehicle-2-X



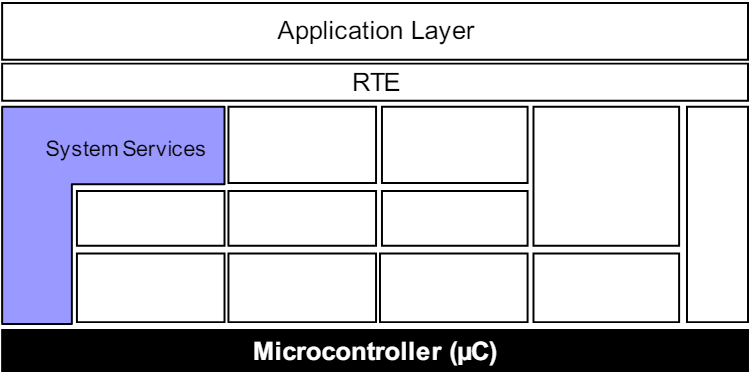


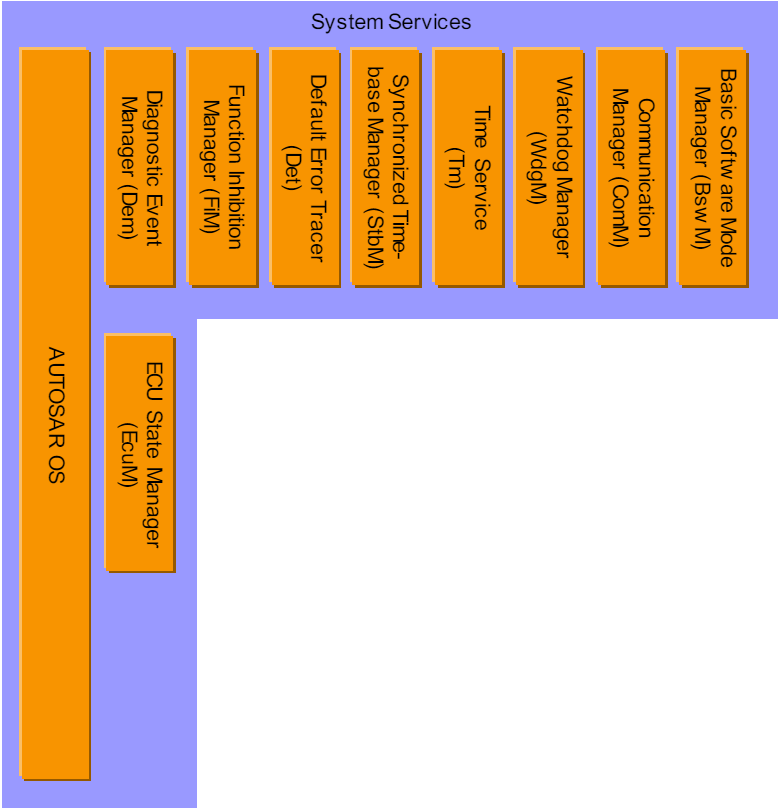
### Services: Memory Services



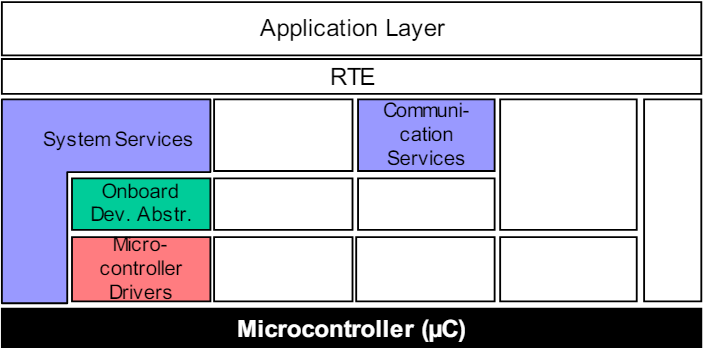


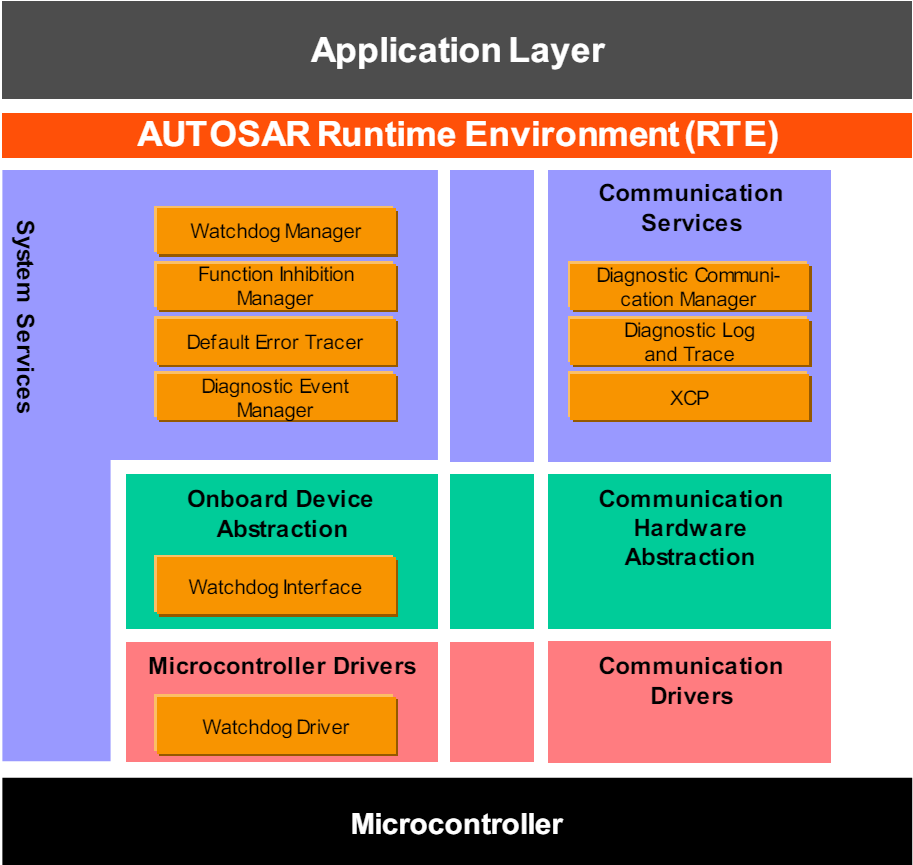
### Services: System Services



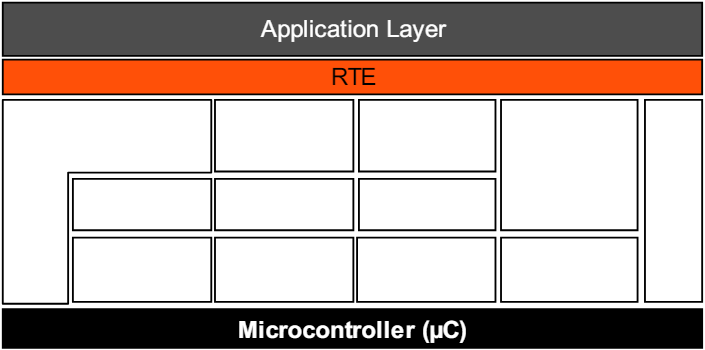


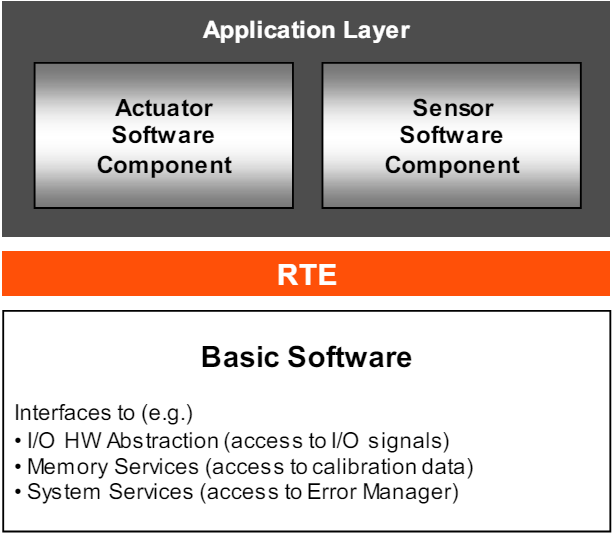
### Error Handling, Reporting and Diagnostic





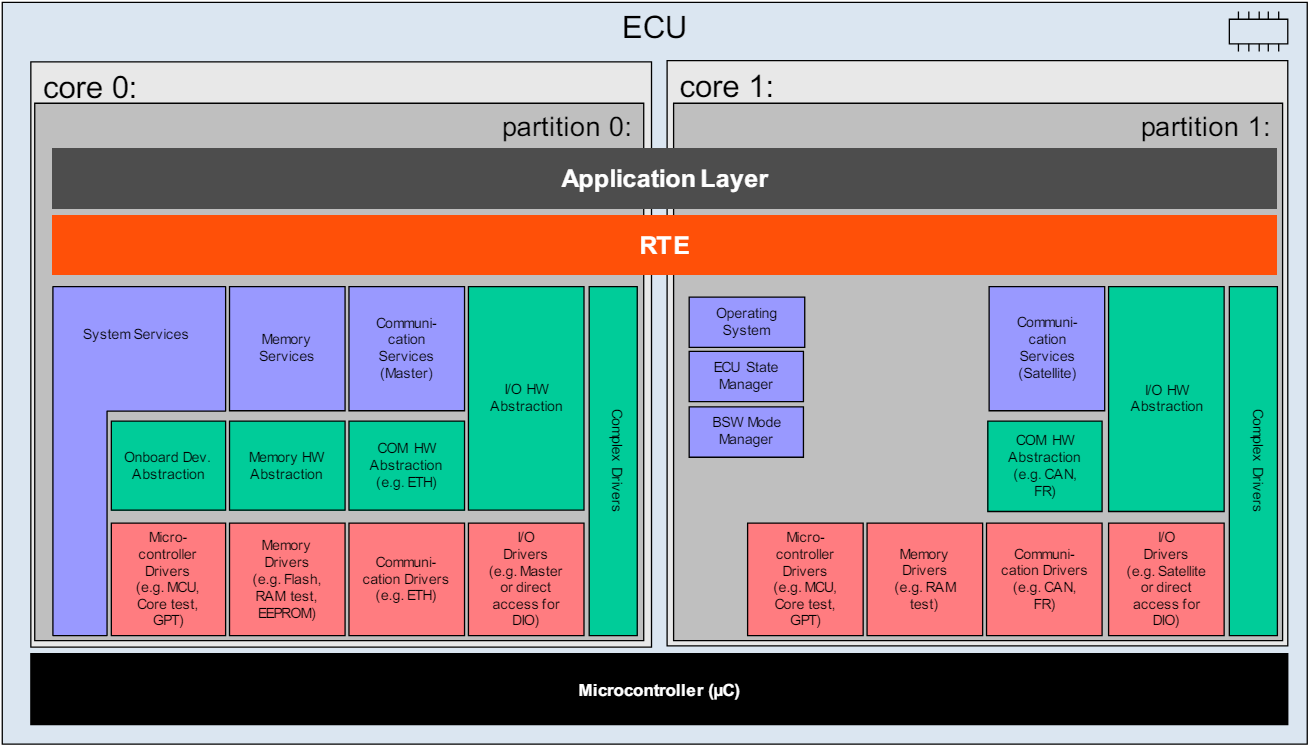
### Application Layer: Sensor/Actuator Software Components



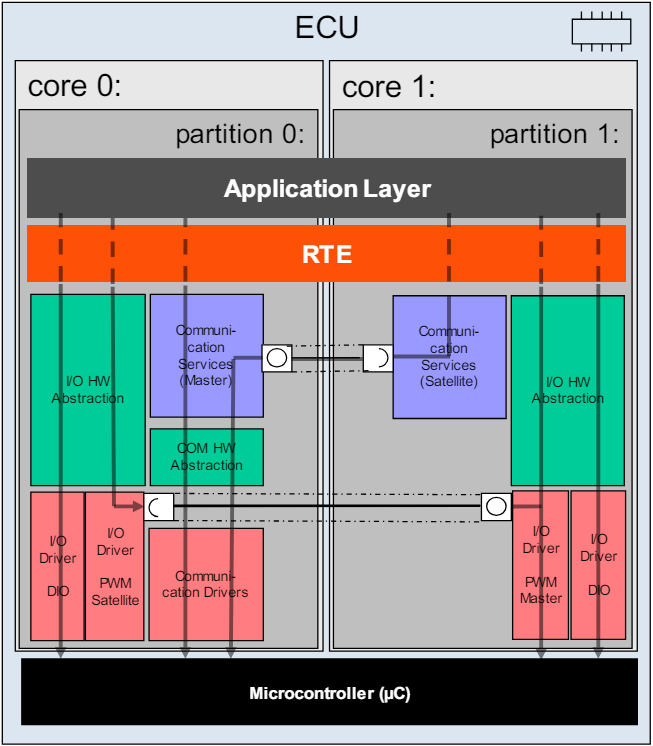


## Content of Software Layers in Multi-Core Systems

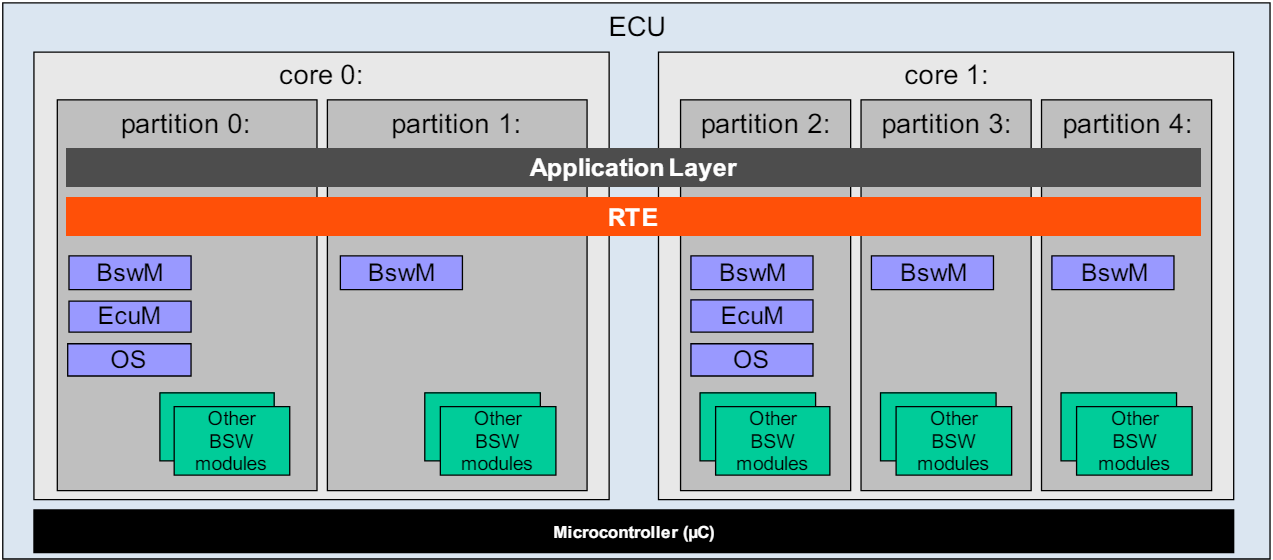
### Example of a Layered Software Architecture for Multi-Core Microcontroller



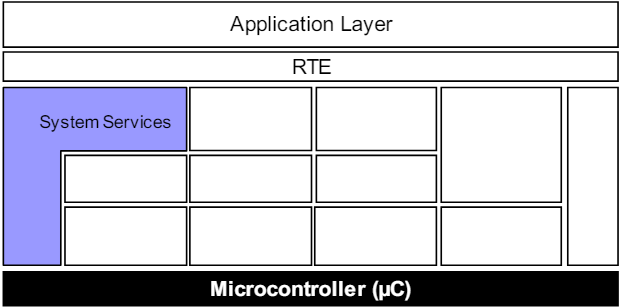
### Detailed View of Distributed BSW Modules

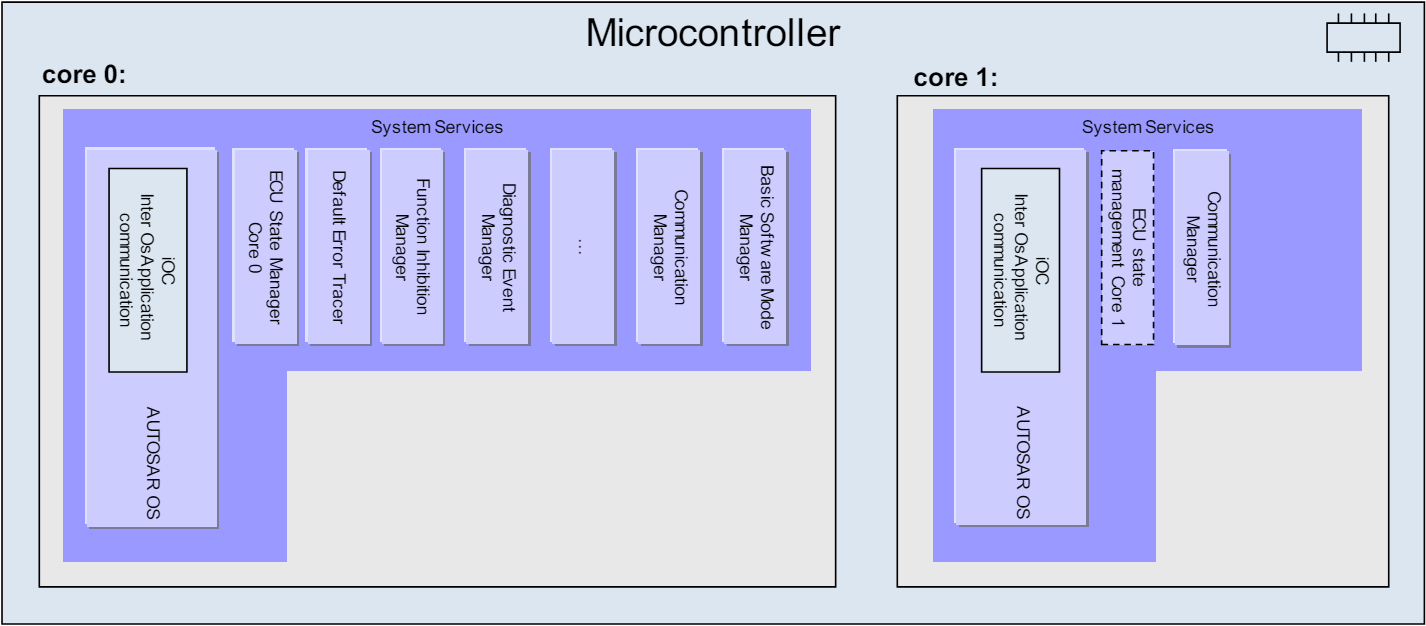


### Overview of BSW Modules, OS, BswM and EcuM on Multiple Partitions



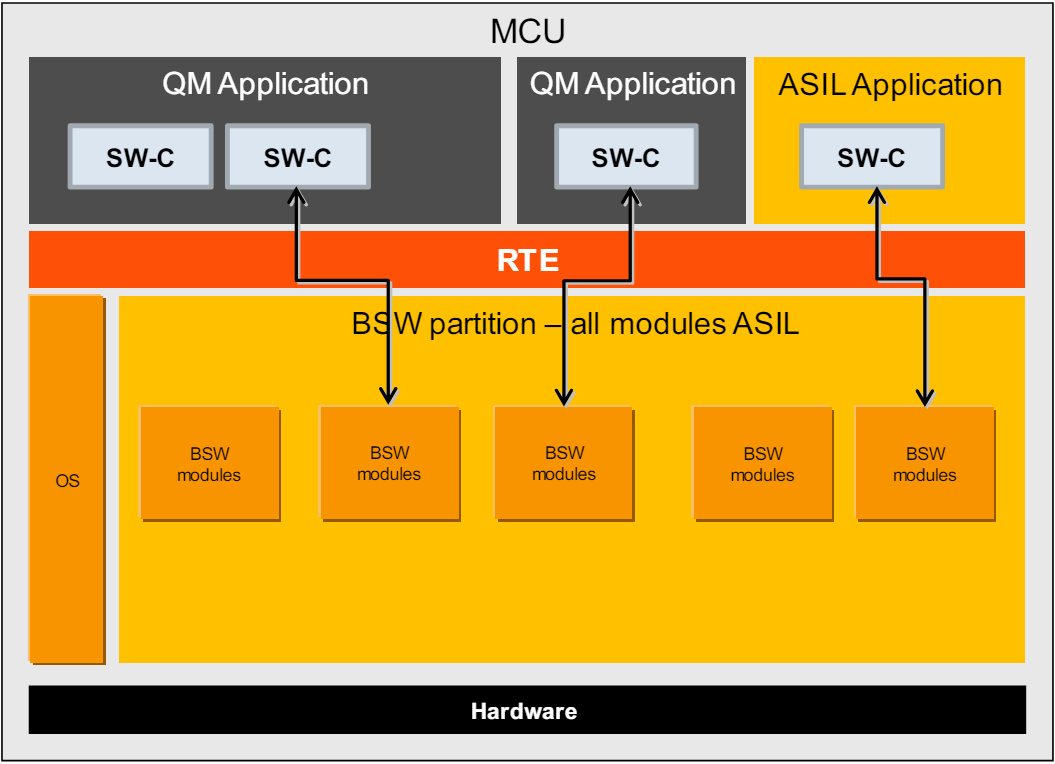
### Scope: Multi-Core System Services



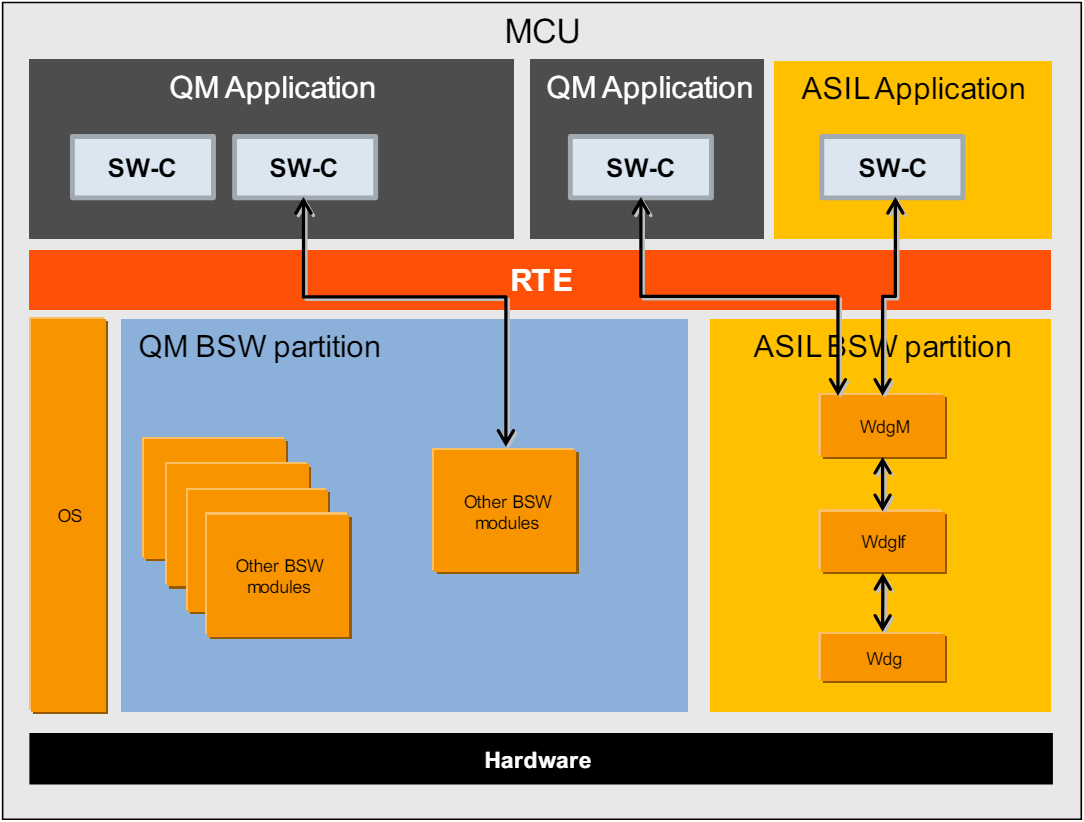


## Content of Software Layers in Mixed-Critical Systems

### Overview of AUTOSAR safety handling

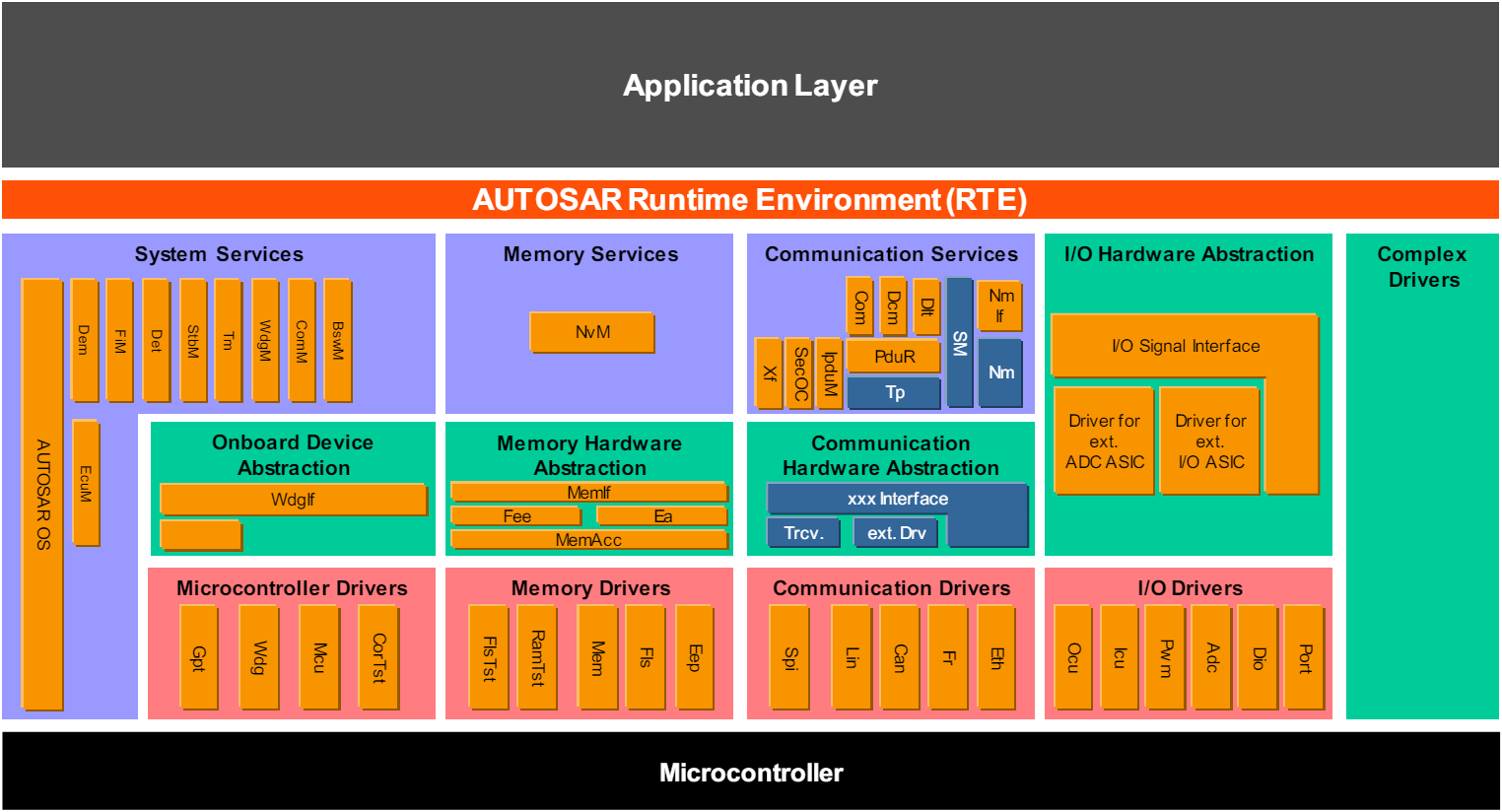


### AUTOSAR BSW distribution for safety systems



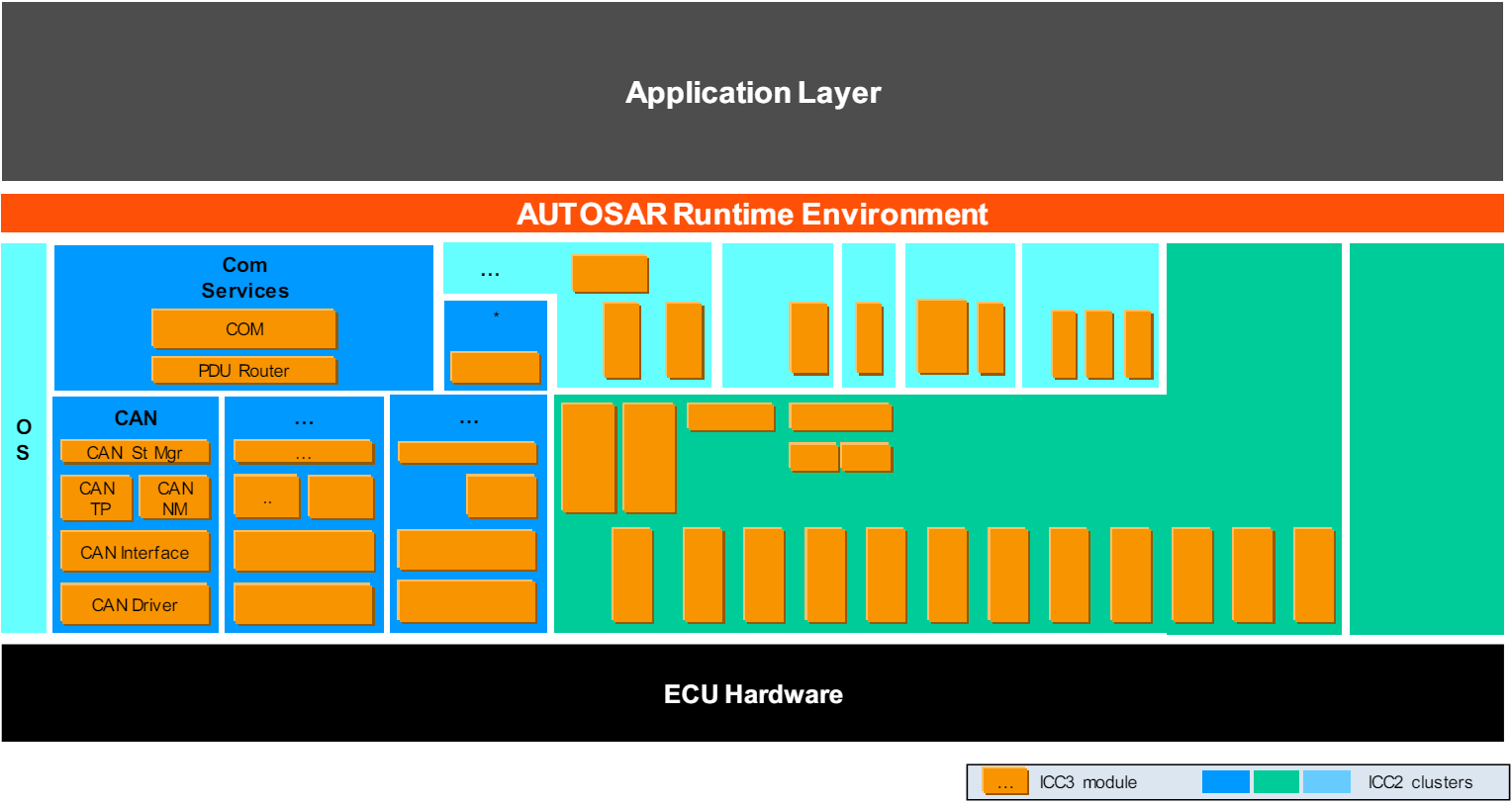
## Overview of Modules

### Overview of Modules – Implementation Conformance Class 3 - ICC3

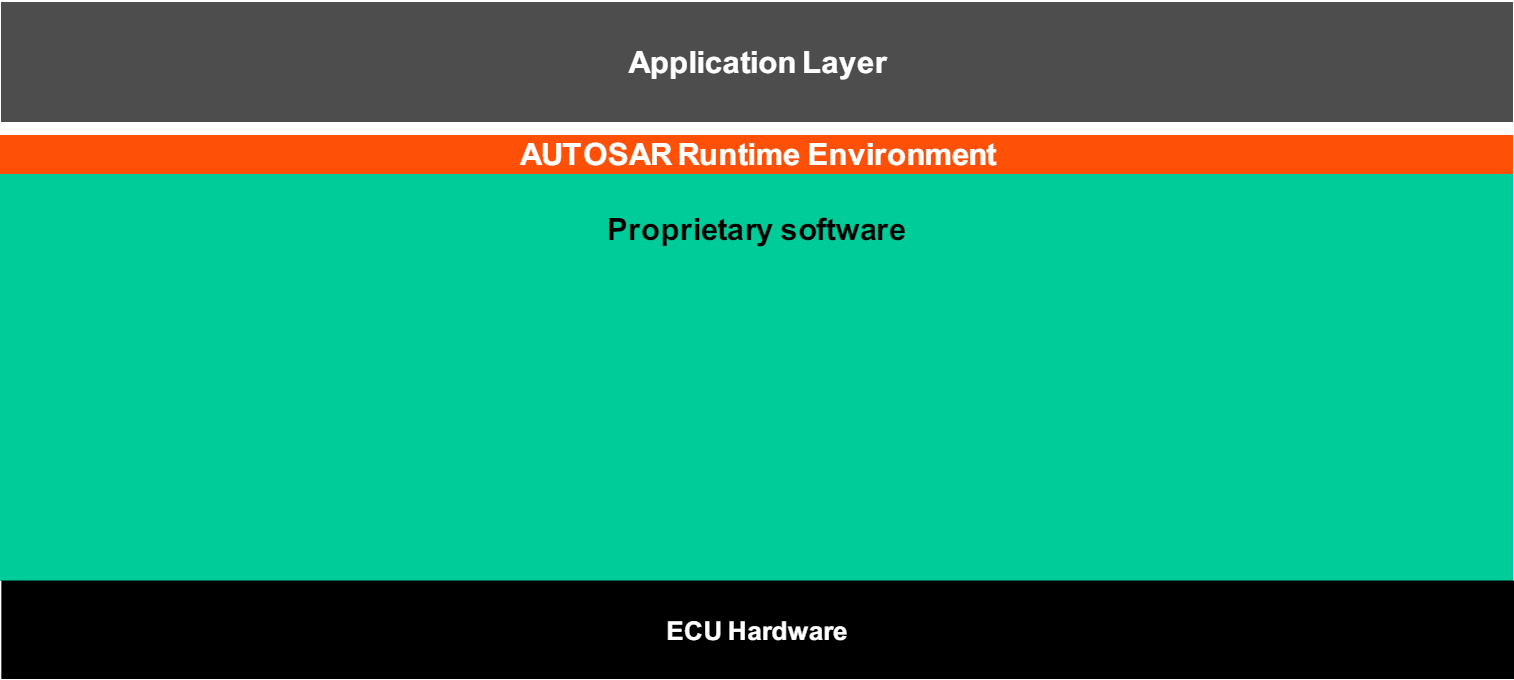




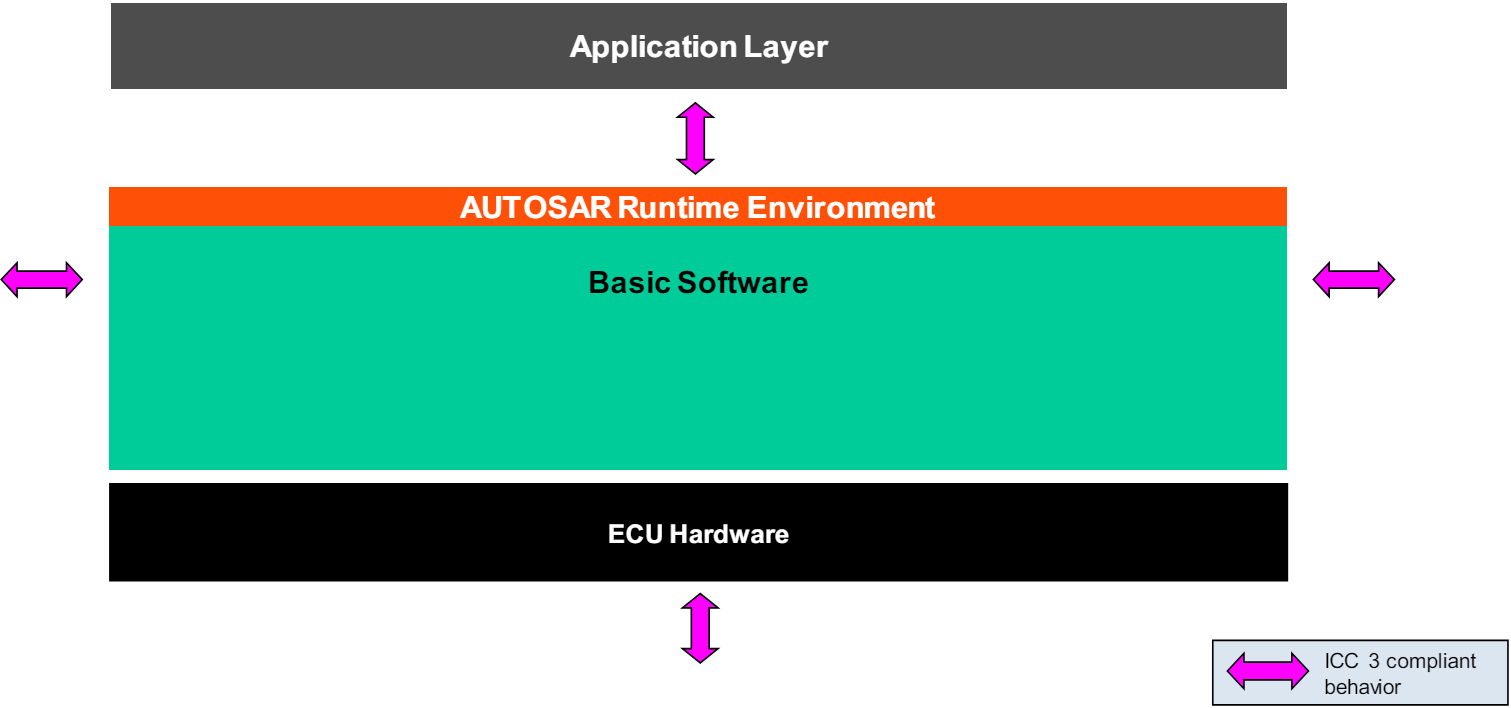
### Overview of Modules – Implementation Conformance Classes – ICC2



### Overview of Modules – Implementation Conformance Classes – ICC1



### Overview of Modules – Implementation Conformance Classes – behavior to the outside

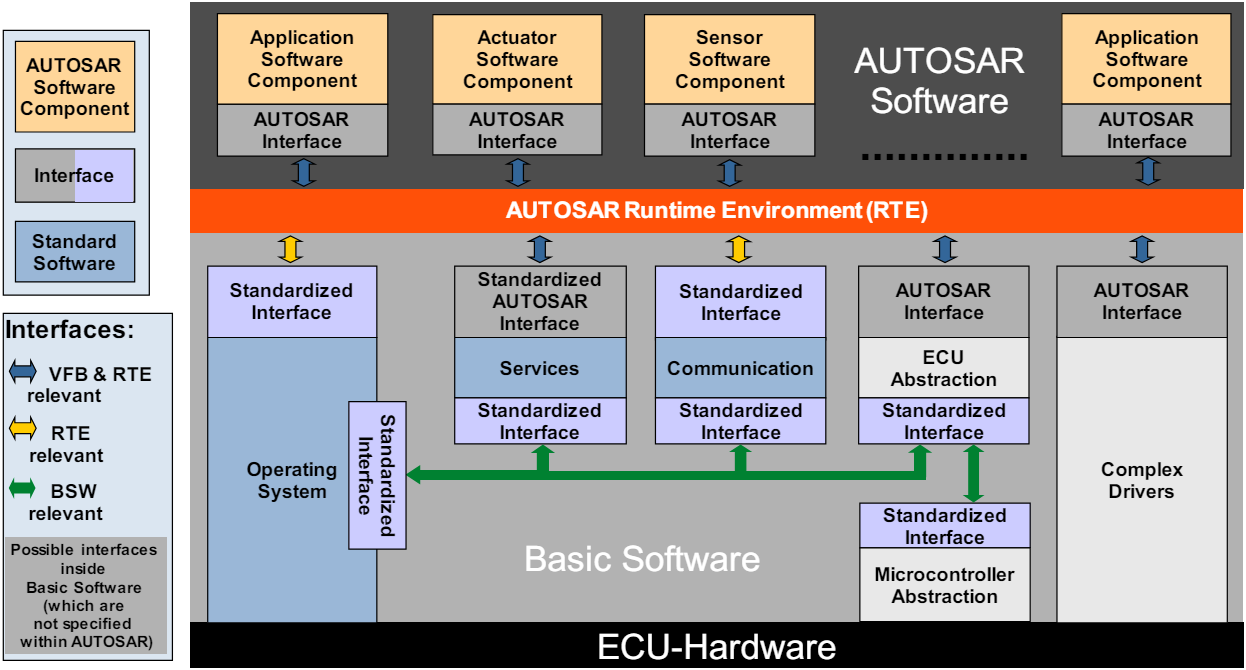


## Interfaces: General Rules

### Interfaces Type of Interfaces in AUTOSAR

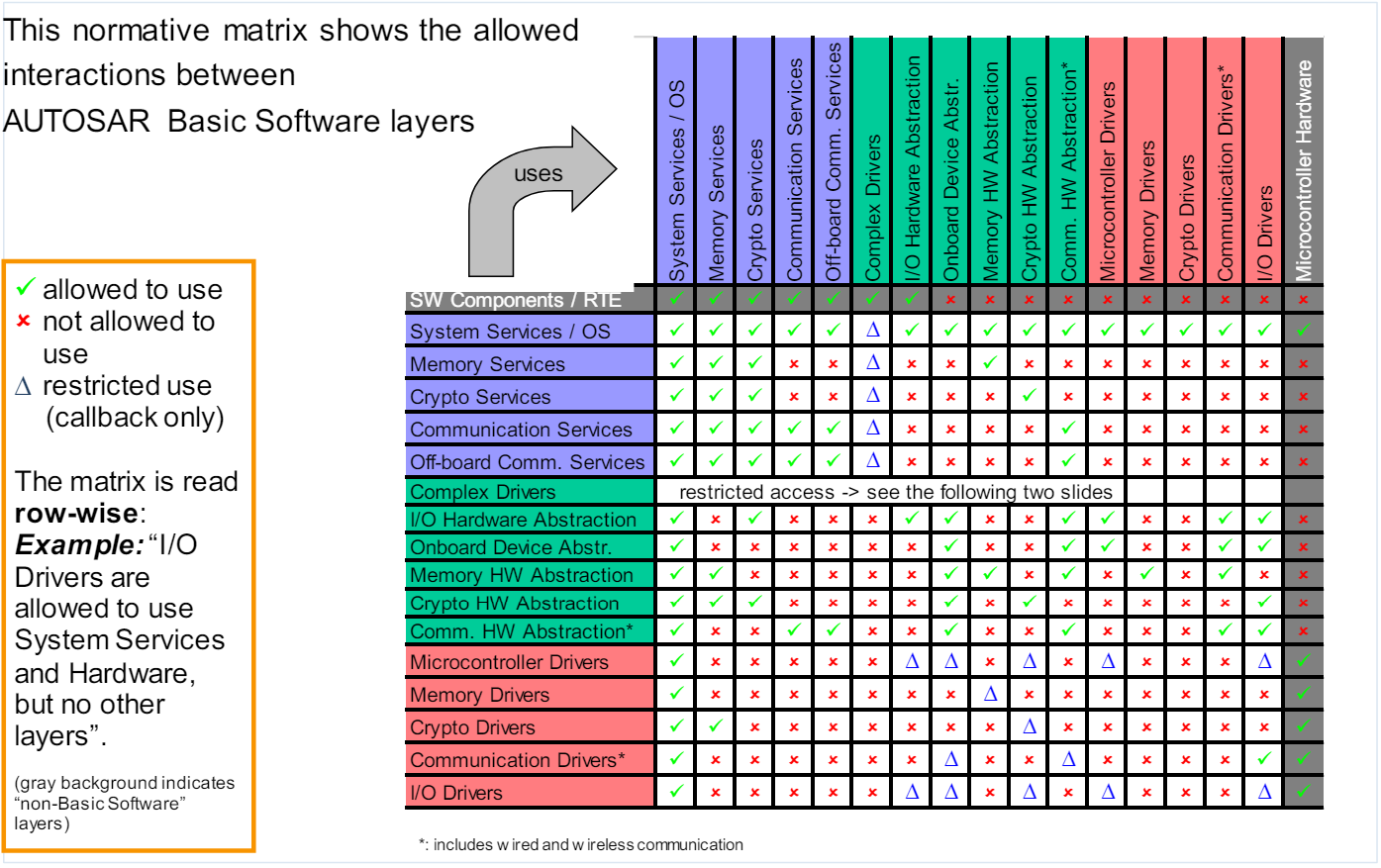
|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

### Components and interfaces view (simplified)

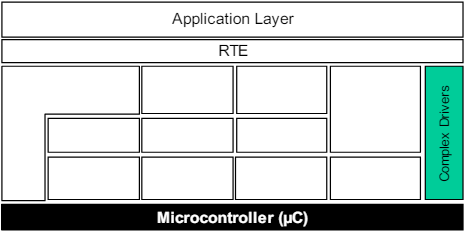


### General Interfacing Rules

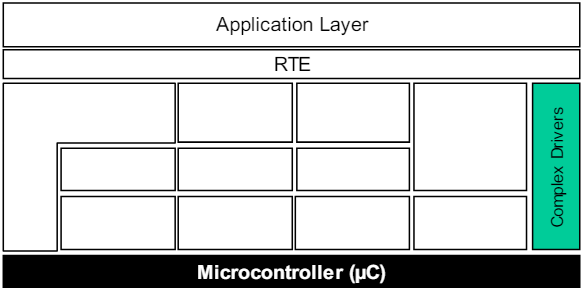
### Layer Interaction Matrix



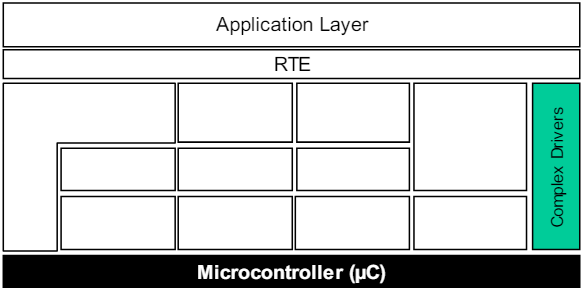
### Interfacing with Complex Drivers (1)



### Interfacing with Complex Drivers (2)



### Interfacing with Complex Drivers (3)



## Interfaces: Interaction of Layers

### Introduction