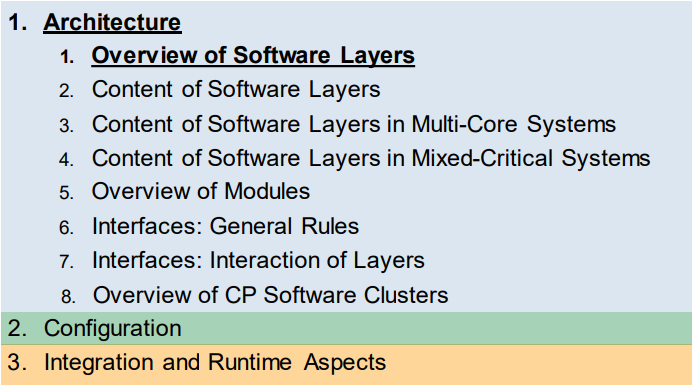
**Layered Software Architecture AUTOSAR**

***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**, với những đặc điểm nổi bật sau:

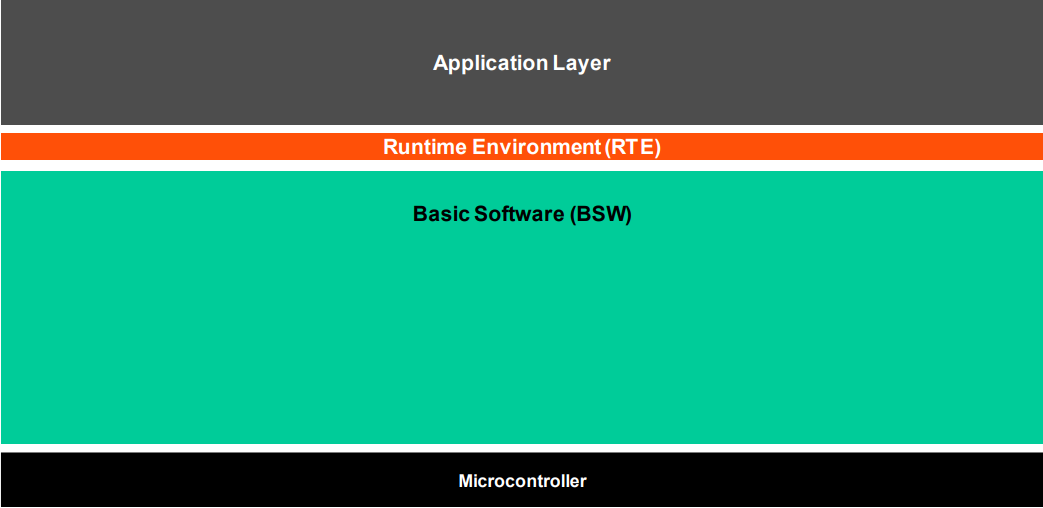
1. **Các mô-đun tiêu chuẩn có thể được mở rộng về chức năng, trong khi vẫn tuân thủ**:
   * Kiến trúc AUTOSAR cho phép mở rộng chức năng của các **mô-đun phần mềm tiêu chuẩn** mà không làm mất đi tính tuân thủ các chuẩn mực đã đặt ra. Tuy nhiên, trong quá trình mở rộng, **cấu hình** của các mô-đun này cần được **xem xét kỹ lưỡng 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 tương thích với các mô-đun khác trong hệ thống và không ảnh hưởng đến sự hoạt động ổn định của toàn bộ phần mềm.
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** vào hệ thống, nhưng chúng cần phải được tích hợp dưới dạng các **Trình điều khiển Phức tạp** (Complex Drivers). Những mô-đun này có thể bao gồm phần mềm chuyên biệt, không hoàn toàn phù hợp với các mô-đun tiêu chuẩn trong AUTOSAR, và cần các cơ chế đặc biệt để đảm bảo hoạt động trơn tru trong hệ thống.
3. **Không thể thêm các lớp bổ sung**:
   * Một nguyên tắc quan trọng trong **kiến trúc AUTOSAR** là không thể **thêm các lớp bổ sung** vào cấu trúc phân lớp đã được xác định. Điều này có nghĩa là hệ thống phần mềm trong AUTOSAR phải duy trì cấu trúc phân lớp rõ ràng và không được phép thay đổi hoặc mở rộng theo những cách có thể làm phức tạp hóa hệ thống. Điều này giúp đảm bảo tính ổn định, dễ bảo trì và khả năng mở rộng của hệ thống.

# ARCHITECTURE

## Tổng quan về Các lớp Phần mềm (Overview of Software Layers)

### Nhìn từ góc độ tổng quan (Top view)

**Kiến trúc AUTOSAR** phân biệt các lớp phần mềm ở mức trừu tượng cao nhất thành **ba lớp chính**: **Application**, **Runtime Environment (RTE)** và **Basic Software (BSW)**, tất cả đều chạy trên một **Bộ điều khiển vi xử lý (Microcontroller Unit - MCU)**.



Các lớp phần mềm trong AUTOSAR:

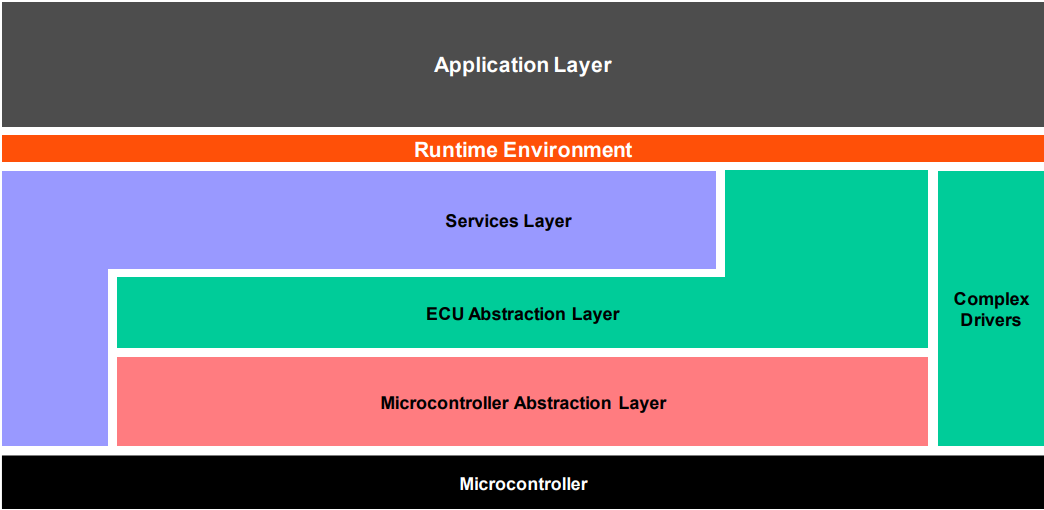
1. **Application Layer (Lớp Ứng dụng)**:
   * **Lớp ứng dụng** chứa các ứng dụng và phần mềm điều khiển đặc thù của từng hệ thống hoặc chức năng trong ô tô. Các ứng dụng này là những chương trình chạy trên ECU (Electronic Control Unit), thực hiện các chức năng như điều khiển động cơ, hệ thống phanh, hệ thống chiếu sáng, hoặc các ứng dụng khác trong hệ thống ô tô.
   * Lớp ứng dụng này không quan tâm đến việc quản lý tài nguyên phần cứng mà chủ yếu tập trung vào các thuật toán xử lý và logic điều khiển.
2. **Runtime Environment (RTE) (Môi trường Thời gian Thực)**:
   * **RTE** là lớp chịu trách nhiệm kết nối **lớp ứng dụng** với các dịch vụ phần mềm cung cấp bởi **Basic Software** và các tài nguyên phần cứng trên ECU. RTE đóng vai trò như một **giao diện phần mềm** cho phép các ứng dụng giao tiếp và tương tác với phần cứng thông qua các mô-đun trong Basic Software mà không cần phải biết cụ thể về phần cứng hoặc các chi tiết của các lớp thấp hơn.
   * RTE giúp tách biệt phần mềm ứng dụng với phần mềm cơ bản (Basic Software), giúp hệ thống dễ dàng mở rộng và bảo trì mà không làm ảnh hưởng đến các lớp khác.
3. **Basic Software (BSW) (Phần mềm Cơ bản)**:
   * **Basic Software** là lớp phần mềm cung cấp các dịch vụ cơ bản và giao tiếp trực tiếp với phần cứng. BSW bao gồm nhiều mô-đun, chẳng hạn như các trình điều khiển phần cứng, các giao thức truyền thông (CAN, LIN, Ethernet, v.v.), các dịch vụ hệ thống như quản lý bộ nhớ, bộ đếm thời gian, và nhiều chức năng hỗ trợ khác.
   * BSW giúp cung cấp một nền tảng ổn định và thống nhất cho các ứng dụng, làm việc chặt chẽ với các bộ điều khiển vi xử lý (MCU) và quản lý việc truy cập vào tài nguyên phần cứng.

**Bộ điều khiển vi xử lý (MCU)**:

* **MCU** là phần cứng cơ bản mà các lớp phần mềm AUTOSAR chạy trên đó. Bộ điều khiển này thực hiện các phép tính và điều khiển các chức năng của hệ thống, đồng thời kết nối với các cảm biến, bộ điều khiển và các thiết bị khác trong ô tô. Bộ điều khiển vi xử lý này thường có tài nguyên hạn chế về công suất tính toán và bộ nhớ, vì vậy các lớp phần mềm phải được tối ưu hóa cho môi trường này.

### Coarse view

Trong **phân tích tổng quan về phần mềm cơ bản AUTOSAR** (Basic Software (BSW)), phần mềm cơ bản được chia thành **bốn lớp chính**, mỗi lớp có vai trò cụ thể trong việc tương tác với phần cứng và các lớp phần mềm cao hơn. Các lớp này bao gồm:



1. **Services (Dịch vụ)**:

* **Lớp Dịch vụ** bao gồm các mô-đun phần mềm cung cấp các chức năng cơ bản mà các lớp phần mềm cao hơn có thể sử dụng. Các dịch vụ này không liên quan trực tiếp đến phần cứng mà tập trung vào việc cung cấp các **chức năng hệ thống**, chẳng hạn như quản lý bộ nhớ, đồng bộ hóa, xử lý sự kiện, hoặc quản lý thông báo giữa các mô-đun phần mềm.
* Các dịch vụ này đóng vai trò như các **tầng hỗ trợ** giúp các ứng dụng và các mô-đun khác có thể giao tiếp một cách hiệu quả mà không cần quan tâm đến phần cứng hoặc các chi tiết thực thi thấp hơn.

2. **ECU Abstraction (Trừu tượng ECU)**:

* **Lớp Trừu tượng ECU** cung cấp một lớp trừu tượng giữa phần mềm và phần cứng của **ECU (Electronic Control Unit)**. Nó đóng vai trò như một cầu nối giúp phần mềm có thể tương tác với phần cứng mà không cần phải biết chi tiết về phần cứng của ECU. Lớp này chuyển đổi các yêu cầu phần mềm thành các yêu cầu cụ thể cho phần cứng của ECU, đồng thời che giấu các chi tiết và sự khác biệt giữa các loại ECU khác nhau.
* Các mô-đun trong lớp này thực hiện các chức năng như điều khiển các cổng I/O, quản lý cảm biến, và truy cập vào các thiết bị ngoại vi mà không làm ảnh hưởng đến các mô-đun phần mềm cấp cao.

3. **Microcontroller Abstraction (Trừu tượng Bộ điều khiển Vi xử lý)**:

* **Lớp Trừu tượng Bộ điều khiển Vi xử lý** tương tự như lớp trừu tượng ECU nhưng có chức năng trừu tượng hóa các đặc điểm của **bộ điều khiển vi xử lý (MCU)**. Lớp này đảm nhiệm việc giao tiếp giữa phần mềm và các phần cứng cụ thể trong bộ điều khiển vi xử lý, chẳng hạn như **bộ vi xử lý, bộ nhớ, các bộ đếm thời gian và các giao diện phần cứng khác**.
* Mục tiêu của lớp này là tạo ra một giao diện phần mềm chung cho tất cả các bộ điều khiển vi xử lý mà không cần quan tâm đến các chi tiết về cấu trúc phần cứng cụ thể của từng loại bộ điều khiển vi xử lý.

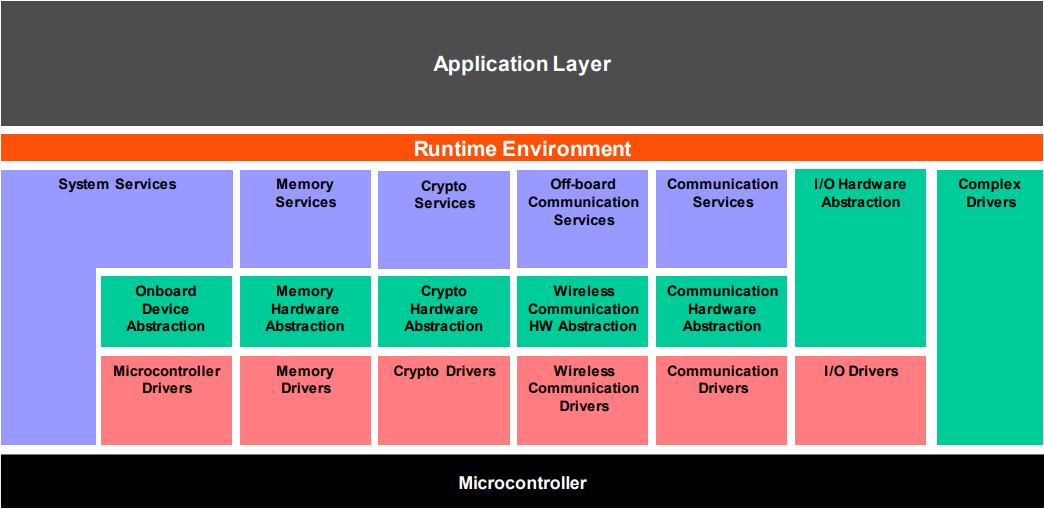
**4. Complex Drivers (Trình điều khiển Phức tạp):**

* **Trình điều khiển phức tạp** bao gồm các mô-đun phần mềm dành cho việc điều khiển các phần cứng đặc biệt hoặc các phần cứng không thể được xử lý thông qua các mô-đun chuẩn trong lớp trừu tượng ECU hoặc bộ điều khiển vi xử lý.
* Các trình điều khiển này có thể bao gồm các giao thức phức tạp như CAN, Ethernet, hoặc các hệ thống giao tiếp phức tạp khác mà yêu cầu các phần mềm điều khiển riêng biệt. Những mô-đun này thường tương tác trực tiếp với phần cứng và yêu cầu các trình điều khiển chuyên biệt để quản lý các tài nguyên phần cứng này.

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

### Detailed view

Trong **phần mềm cơ bản AUTOSAR**, các **Basic Software Layers** (BSW) được phân chia thành các **nhóm chức năng cụ thể** để hỗ trợ các yêu cầu phần mềm và phần cứng của hệ thống ô tô. Các nhóm chức năng này được thiết kế để xử lý các nhiệm vụ khác nhau và cung cấp các dịch vụ cần thiết cho các lớp phần mềm và ứng dụng khác.



***Các nhóm chức năng trong Basic Software:***

1. **System Services (Dịch vụ Hệ thống)**:
   * **Dịch vụ Hệ thống** cung cấp các chức năng nền tảng cho việc vận hành và điều khiển hệ thống. Các dịch vụ này bao gồm các tác vụ như:
     + **Quản lý lỗi**: Giám sát và báo cáo các lỗi hệ thống.
     + **Quản lý thời gian**: Bao gồm các dịch vụ như bộ đếm thời gian, đồng hồ hệ thống, v.v.
     + **Quản lý sự kiện**: Xử lý các sự kiện hệ thống, chẳng hạn như các tín hiệu từ các bộ cảm biến hoặc các kích hoạt từ phần mềm.
   * Đây là các chức năng quan trọng giúp duy trì sự ổn định và hiệu suất của hệ thống ô tô.
2. **Memory Services (Dịch vụ Bộ nhớ)**:
   * **Dịch vụ Bộ nhớ** quản lý việc truy cập và sử dụng bộ nhớ của hệ thống. Các chức năng chính bao gồm:
     + **Quản lý bộ nhớ flash**: Quản lý việc lưu trữ và truy cập dữ liệu trong bộ nhớ flash của ECU.
     + **Quản lý bộ nhớ RAM**: Xử lý việc phân bổ và giải phóng bộ nhớ trong các tình huống thời gian thực.
     + **Quản lý bộ nhớ EEPROM**: Xử lý việc lưu trữ dữ liệu cần bảo vệ và phục hồi sau khi tắt nguồn.
   * Các dịch vụ này quan trọng trong việc tối ưu hóa việc sử dụng bộ nhớ và đảm bảo sự bền vững của hệ thống.
3. **Communication Services (Dịch vụ Giao tiếp)**:
   * **Dịch vụ Giao tiếp** chịu trách nhiệm cho tất cả các chức năng liên quan đến giao tiếp giữa các ECU hoặc giữa các mô-đun trong hệ thống. Các chức năng giao tiếp này bao gồm:
     + **Giao thức truyền thông**: Xử lý các giao thức như CAN (Controller Area Network), LIN (Local Interconnect Network), Ethernet, FlexRay, và các giao thức khác.
     + **Quản lý các giao tiếp theo thời gian thực**: Đảm bảo các thông điệp được truyền tải chính xác và kịp thời trong hệ thống.
     + **Quản lý các kênh truyền thông**: Đảm bảo tính đồng bộ và quản lý các đường truyền dữ liệu giữa các ECU.
   * Các dịch vụ này rất quan trọng đối với sự vận hành của các hệ thống ô tô hiện đại, nơi các ECU cần phải giao tiếp liên tục và hiệu quả.

### Microcontroller Abstraction Layer

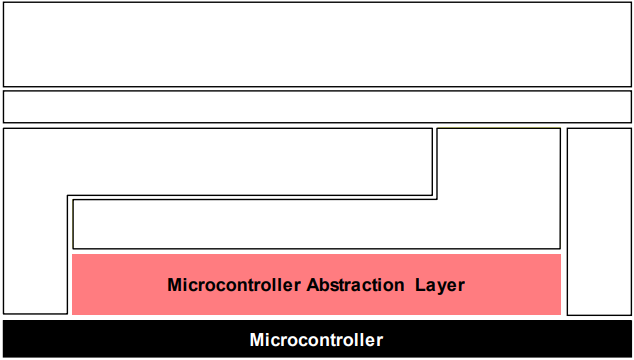
Lớp **Microcontroller Abstraction Layer (MCAL)** là lớp phần mềm thấp nhất trong **Basic Software** (Phần mềm cơ bản) trong kiến trúc **AUTOSAR**. MCAL có vai trò rất quan trọng trong việc **tách biệt phần mềm khỏi phần cứng**, giúp các mô-đun phần mềm ở các lớp cao hơn có thể hoạt động mà không cần quan tâm đến các chi tiết về bộ điều khiển vi xử lý (µC) cụ thể mà hệ thống đang sử dụng.

#### Nhiệm vụ của MCAL

* **Tạo các lớp phần mềm cao hơn độc lập với Bộ điều khiển vi xử lý (µC)**:
  + MCAL cung cấp một lớp trừu tượng giúp các mô-đun phần mềm không phải lo lắng về các chi tiết của phần cứng cụ thể của bộ điều khiển vi xử lý (microcontroller). Điều này có nghĩa là phần mềm ở các lớp cao hơn có thể tập trung vào các chức năng nghiệp vụ mà không cần quan tâm đến việc tương tác với các chi tiết phần cứng, chẳng hạn như các bộ đếm thời gian, bộ chuyển mạch, hoặc các giao thức giao tiếp cụ thể của bộ điều khiển vi xử lý.

#### Thuộc tính của MCAL

* **Triển khai: Phụ thuộc vào µC**:
  + MCAL được thiết kế để hoạt động cụ thể trên từng loại bộ điều khiển vi xử lý (µC). Điều này có nghĩa là **MCAL sẽ được cấu hình và triển khai khác nhau tùy vào loại bộ điều khiển vi xử lý** mà nó đang hỗ trợ. Các bộ điều khiển vi xử lý khác nhau có thể có các đặc điểm và chức năng phần cứng khác nhau, vì vậy MCAL phải được điều chỉnh để phù hợp với từng loại bộ vi xử lý này.
* **Giao diện trên cùng: Chuẩn hóa và độc lập với µC**:
  + Mặc dù MCAL phụ thuộc vào bộ điều khiển vi xử lý cụ thể trong phần cứng, nhưng giao diện mà nó cung cấp cho các mô-đun phần mềm ở các lớp cao hơn lại được **chuẩn hóa và độc lập với µC**. Điều này có nghĩa là các mô-đun phần mềm cấp cao sẽ tương tác với MCAL thông qua một giao diện chuẩn mà không cần quan tâm đến các chi tiết phần cứng của bộ điều khiển vi xử lý cụ thể.



**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***

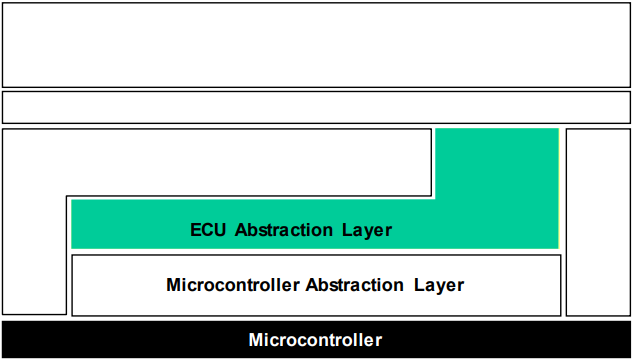
* Tạo các lớp phần mềm cao hơn độc lập với Bộ điều khiển vi xử lý

***Thuộc tính***

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

### ECU Abstraction Layer

Lớp **ECU Abstraction Layer (ECUAL)** là lớp phần mềm trong **Basic Software** của **AUTOSAR** có nhiệm vụ giao tiếp với các trình điều khiển của **Microcontroller Abstraction Layer (MCAL)** và điều khiển các thiết bị ngoại vi. ECUAL đóng vai trò trung gian giữa phần mềm cao hơn và phần cứng, giúp tách biệt phần mềm khỏi các chi tiết cụ thể về phần cứng của Đơn vị Điều khiển Điện tử (ECU).



#### Nhiệm vụ của ECU Abstraction Layer

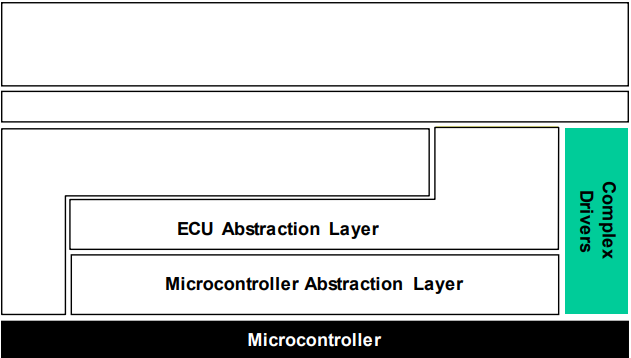
* **Tạo các lớp phần mềm cao hơn độc lập với bố trí phần cứng của Đơn vị Điều khiển Điện tử (ECU)**:
  + ECUAL cung cấp **một lớp trừu tượng** giúp phần mềm ở các lớp cao hơn không cần phải lo lắng về cách thức bố trí phần cứng của ECU (bao gồm các bộ vi xử lý và các thiết bị ngoại vi). Điều này giúp tăng tính linh hoạt của phần mềm, khi phần cứng ECU có thể thay đổi mà không ảnh hưởng đến phần mềm ứng dụng.

#### Thuộc tính của ECU Abstraction Layer

* **Triển khai: Độc lập với µC, phụ thuộc vào ECU hardware**:
  + Mặc dù ECUAL phụ thuộc vào phần cứng của **Đơn vị Điều khiển Điện tử (ECU)**, nó lại không phụ thuộc vào bộ điều khiển vi xử lý cụ thể (µC). Điều này có nghĩa là ECUAL có thể được triển khai trên các ECU sử dụng các loại bộ vi xử lý khác nhau, miễn là chúng có phần cứng tương thích. ECUAL có thể tương tác với nhiều loại phần cứng khác nhau trong hệ thống ECU.
* **Giao diện trên cùng: Độc lập với µC và ECU hardware**:
  + Giao diện mà ECUAL cung cấp cho các mô-đun phần mềm ở các lớp cao hơn được chuẩn hóa và **độc lập với µC và phần cứng của ECU**. Điều này có nghĩa là các mô-đun phần mềm ở lớp cao hơn có thể truy cập các thiết bị ngoại vi và các tài nguyên phần cứng của ECU mà không cần quan tâm đến cách thức chúng được kết nối hay vị trí của chúng (nội bộ hay bên ngoài bộ vi xử lý, chân cổng, hoặc loại giao diện).

### Complex Drivers

Lớp **Complex Drivers** trong **AUTOSAR** là một phần của **Basic Software**, có nhiệm vụ cung cấp khả năng tích hợp các chức năng đặc biệt mà các mô-đun phần mềm khác không thể hỗ trợ trực tiếp. Các **trình điều khiển phức tạp** này có thể được sử dụng cho các thiết bị không được chỉ định trong **AUTOSAR**, hoặc các thiết bị yêu cầu **ràng buộc thời gian rất cao**, hoặc các thiết bị cần được di chuyển và cấu hình lại trong hệ thống.



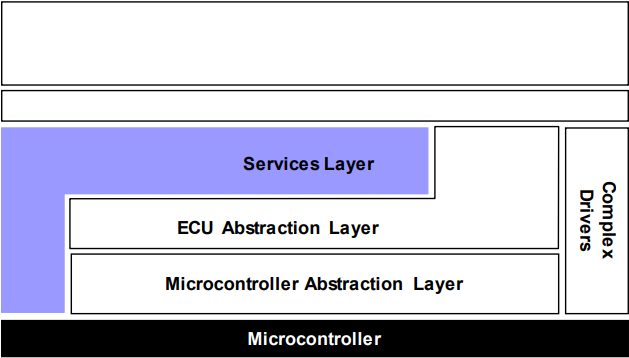
#### Nhiệm vụ của Complex Drivers Layer

* **Cung cấp khả năng tích hợp chức năng đặc biệt**, ví dụ như:
  + **Trình điều khiển cho các thiết bị không được chỉ định trong AUTOSAR**: Lớp này hỗ trợ tích hợp các thiết bị hoặc phần cứng mà **AUTOSAR không định nghĩa**. Điều này đặc biệt hữu ích khi có phần cứng hoặc thiết bị đặc biệt mà không được mô tả trong các mô-đun phần mềm tiêu chuẩn của AUTOSAR.
  + **Trình điều khiển có ràng buộc thời gian rất cao**: Các thiết bị yêu cầu **thời gian phản hồi nhanh**, chẳng hạn như cảm biến hoặc bộ điều khiển phức tạp, cần một lớp trình điều khiển có thể xử lý các yêu cầu thời gian thực rất khắt khe.
  + **Chức năng di chuyển và cấu hình lại**: Complex Drivers cũng có thể xử lý các tình huống yêu cầu **cấu hình lại phần cứng hoặc di chuyển thiết bị** trong quá trình hoạt động của hệ thống.

#### Thuộc tính của Complex Drivers Layer

* **Triển khai**:
  + **Có thể là ứng dụng**, phụ thuộc vào bộ điều khiển vi xử lý (µC) và phần cứng ECU. Lớp **Complex Drivers** có thể được triển khai dưới dạng ứng dụng phần mềm tùy chỉnh, có thể phụ thuộc vào các yêu cầu phần cứng cụ thể của bộ điều khiển vi xử lý và ECU. Điều này cho phép tùy chỉnh các trình điều khiển để phù hợp với phần cứng và yêu cầu hệ thống.
* **Giao diện trên cùng**:
  + **Có thể là ứng dụng**, phụ thuộc vào bộ điều khiển vi xử lý (µC) và phần cứng ECU. Các mô-đun phần mềm ở lớp trên có thể tương tác với **Complex Drivers** thông qua giao diện chuẩn, nhưng giao diện này cũng có thể thay đổi tùy thuộc vào ứng dụng và phần cứng của hệ thống.

### Services Layer



Lớp **Services** là lớp cao nhất trong **Basic Software** của **AUTOSAR** và có vai trò quan trọng trong việc cung cấp các dịch vụ cơ bản cho các ứng dụng, môi trường thực thi (RTE), và các mô-đun phần mềm trong **Basic Software**. Lớp này thực hiện các chức năng hỗ trợ cho toàn bộ hệ thống, bao gồm quản lý tài nguyên hệ thống, giao tiếp mạng, chẩn đoán lỗi, và giám sát các trạng thái của hệ thống.

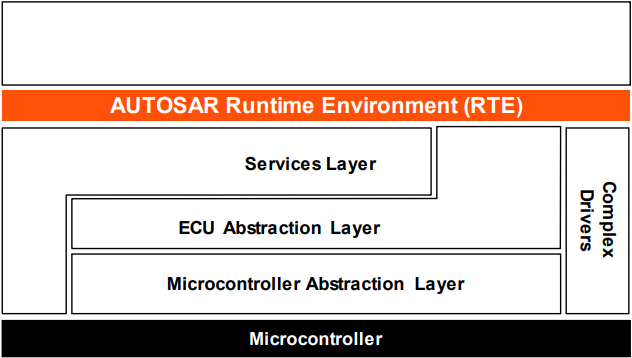
#### Các chức năng mà Services Layer cung cấp:

1. **Chức năng của hệ điều hành**:
   * Dịch vụ của hệ điều hành được cung cấp trong lớp này để hỗ trợ các ứng dụng và hệ thống trong việc quản lý tác vụ, luồng chương trình, và tài nguyên hệ thống.
2. **Dịch vụ giao tiếp và quản lý mạng xe**:
   * Services Layer hỗ trợ các giao thức mạng như **CAN**, **LIN**, **FlexRay**, và **Ethernet**. Lớp này cung cấp các dịch vụ để đảm bảo giao tiếp hiệu quả và chính xác giữa các ECUs (Đơn vị Điều khiển Điện tử) trong hệ thống.
3. **Dịch vụ bộ nhớ (quản lý NVRAM)**:
   * Dịch vụ này giúp quản lý **Non-Volatile RAM (NVRAM)**, nơi lưu trữ các dữ liệu không thay đổi sau khi hệ thống tắt nguồn, chẳng hạn như thông tin cấu hình và dữ liệu chẩn đoán.
4. **Dịch vụ chẩn đoán**:
   * Bao gồm các chức năng như **giao tiếp UDS (Unified Diagnostic Services)**, **bộ nhớ lỗi**, và **xử lý lỗi**. Lớp này hỗ trợ phát hiện và xử lý các lỗi trong hệ thống, đảm bảo tính ổn định và độ tin cậy cao.
5. **Quản lý trạng thái ECU và chế độ**:
   * Services Layer cung cấp các dịch vụ quản lý trạng thái hoạt động của ECU, giúp điều khiển các chế độ khác nhau của ECU (chế độ khởi động, hoạt động, tắt máy, v.v.).
6. **Giám sát dòng chương trình logic và thời gian (quản lý Wdg)**:
   * Lớp này thực hiện **giám sát dòng chương trình logic** và **quản lý thời gian** để đảm bảo chương trình không bị lỗi logic hoặc lỗi thời gian (thời gian thực). Chức năng này cũng bao gồm việc **giám sát watchdog (Wdg)**, bảo vệ hệ thống khỏi tình trạng không phản hồi hoặc bị treo.

#### Nhiệm vụ của Services Layer

* **Cung cấp các dịch vụ cơ bản cho các Applications, RTE và các mô-đun phần mềm của Basic Software**:
  + Services Layer cung cấp các dịch vụ nền tảng để hỗ trợ hoạt động của các ứng dụng và mô-đun khác trong hệ thống, đảm bảo các chức năng thiết yếu của hệ thống như điều khiển tác vụ, giao tiếp mạng, và chẩn đoán lỗi.

### Runtime Environment (RTE)



**RTE** (Môi trường Thực thi) là lớp trung gian quan trọng trong kiến trúc **AUTOSAR**, cung cấp dịch vụ giao tiếp cho **Application Software**, bao gồm các **AUTOSAR Software Components** và **AUTOSAR Sensor/Actuator components**. RTE là cầu nối giúp các thành phần phần mềm trong hệ thống AUTOSAR có thể giao tiếp với nhau, đồng thời tương tác với các dịch vụ và thành phần bên ngoài ECU (Electronic Control Unit).

#### Mô hình Kiến trúc: Từ ****Layered**** đến ****Component Style****

Phía trên **RTE**, kiến trúc phần mềm của AUTOSAR chuyển từ mô hình **"layered"** (lớp phần mềm phân tầng) sang **"component style"** (kiến trúc kiểu thành phần), tức là các thành phần phần mềm sẽ giao tiếp trực tiếp với nhau qua **RTE**, thay vì chỉ đơn thuần theo các lớp phân cấp. Điều này làm tăng tính linh hoạt và khả năng tái sử dụng của các thành phần phần mềm trong môi trường hệ thống.

#### Nhiệm vụ của ****RTE****

* **Độc lập với phép ánh xạ đến ECU cụ thể**: **RTE** đảm bảo rằng các **AUTOSAR Software Components** không bị ràng buộc vào các đặc tính phần cứng hay cấu hình của một **ECU** cụ thể. Điều này giúp phần mềm trở nên **độc lập** và có thể tái sử dụng cho nhiều loại ECU khác nhau mà không cần phải thay đổi mã nguồn ứng dụng. Các thành phần phần mềm có thể giao tiếp với nhau qua **RTE** mà không phụ thuộc vào phần cứng của ECU.

#### Thuộc tính của ****RTE****

* **Triển khai**:
  + **Đặc thù cho ECU và ứng dụng**: **RTE** được triển khai riêng biệt cho mỗi ECU và ứng dụng, với mục đích tối ưu hóa cho các điều kiện cụ thể của phần cứng và ứng dụng đó. Tuy nhiên, mỗi ECU có thể có một **RTE** riêng biệt phù hợp với ứng dụng của nó.
* **Giao diện trên cùng**:
  + **Độc lập hoàn toàn với ECU**: Mặc dù **RTE** được triển khai riêng biệt cho mỗi ECU, nhưng giao diện của **RTE** với các thành phần phần mềm và ứng dụng **độc lập hoàn toàn với phần cứng ECU**. Điều này có nghĩa là giao tiếp giữa các **AUTOSAR Software Components** qua **RTE** không bị phụ thuộc vào phần cứng của ECU, tạo ra một hệ thống phần mềm linh hoạt và dễ dàng tương thích với các ECU khác nhau.

### Các Loại Dịch Vụ trong ****Basic Software (BSW)**** của AUTOSAR

Trong kiến trúc **AUTOSAR**, **Basic Software (BSW)** đóng vai trò là lớp phần mềm cơ bản giúp kết nối các thành phần phần mềm ứng dụng với phần cứng của ECU. **BSW** cung cấp một loạt các dịch vụ quan trọng giúp quản lý phần cứng, bộ nhớ, giao tiếp và bảo mật, từ đó hỗ trợ hoạt động của các **AUTOSAR Software Components** (ASC). Các dịch vụ này có thể được chia thành nhiều nhóm chính như sau:

#### ****Input/Output (I/O)****

* **Mô tả**: Cung cấp khả năng truy cập chuẩn hóa đến các cảm biến, bộ điều khiển và các thiết bị ngoại vi trên **ECU**. Dịch vụ I/O đóng vai trò quan trọng trong việc kết nối phần mềm với thế giới bên ngoài thông qua các thiết bị vật lý. Các cảm biến và bộ điều khiển thường phải giao tiếp với phần mềm ứng dụng để thu thập hoặc xử lý dữ liệu từ môi trường xung quanh xe (ví dụ như cảm biến nhiệt độ, bộ điều khiển động cơ, etc).
* **Chức năng chính**:
  + Truy cập và điều khiển các thiết bị ngoại vi.
  + Chuẩn hóa các giao diện và phương thức truy cập.

#### ****Memory****

* **Mô tả**: Cung cấp truy cập chuẩn hóa đến các bộ nhớ nội bộ và ngoại vi, bao gồm bộ nhớ không thay đổi (non-volatile memory). Các dịch vụ này quan trọng trong việc lưu trữ dữ liệu lâu dài (ví dụ như dữ liệu cấu hình, trạng thái hệ thống) mà không bị mất khi mất điện.
* **Chức năng chính**:
  + Quản lý việc đọc/ghi vào bộ nhớ không thay đổi (non-volatile memory).
  + Cung cấp dịch vụ cho phép ứng dụng lưu trữ dữ liệu bền vững.

#### ****Crypto (Mã hóa)****

* **Mô tả**: Cung cấp quyền truy cập chuẩn hóa vào các nguyên lý mã hóa, bao gồm các phần cứng mã hóa bên trong và bên ngoài. Dịch vụ mã hóa bảo vệ các giao tiếp và dữ liệu trong hệ thống từ các tấn công bên ngoài, đảm bảo tính bảo mật cho hệ thống.
* **Chức năng chính**:
  + Cung cấp các thuật toán mã hóa và giải mã.
  + Quản lý các khóa bảo mật cho giao tiếp an toàn.

#### ****Giao tiếp****

* **Mô tả**: Dịch vụ giao tiếp chuẩn hóa giúp kết nối các hệ thống mạng trong xe, các hệ thống giao tiếp trên **ECU** và phần mềm bên trong **ECU**. Các dịch vụ giao tiếp này cung cấp các phương thức chuẩn để các **AUTOSAR Software Components** và các hệ thống mạng có thể giao tiếp với nhau.
* **Chức năng chính**:
  + Quản lý các giao tiếp nội bộ giữa các thành phần của ECU.
  + Giao tiếp giữa các hệ thống mạng trên xe, ví dụ như **CAN**, **Ethernet**, **LIN**.

#### ****Giao tiếp ngoài ECU****

* **Mô tả**: Cung cấp quyền truy cập chuẩn hóa vào giao tiếp giữa xe với các hệ thống bên ngoài (X), thông qua các hệ thống mạng không dây hoặc giao tiếp bên ngoài ECU. Điều này bao gồm việc giao tiếp với các thiết bị bên ngoài như điện thoại di động, cơ sở hạ tầng giao thông, hoặc các dịch vụ đám mây.
* **Chức năng chính**:
  + Kết nối với các thiết bị hoặc hệ thống bên ngoài thông qua mạng không dây (ví dụ **Wi-Fi**, **Bluetooth**, **5G**).
  + Tích hợp với các dịch vụ bên ngoài xe (ví dụ như cập nhật phần mềm từ xa).

#### ****System****

* **Mô tả**: Dịch vụ hệ thống cung cấp các chức năng chuẩn hóa như hệ điều hành, bộ đếm thời gian, bộ nhớ lỗi và các chức năng thư viện. Hệ thống này cũng bao gồm các dịch vụ đặc thù cho ECU như quản lý trạng thái của ECU và **watchdog**. Các dịch vụ hệ thống này đảm bảo sự ổn định và quản lý hiệu quả các tài nguyên của ECU.
* **Chức năng chính**:
  + Cung cấp các dịch vụ hệ điều hành như lập lịch tác vụ, điều phối đa nhiệm, và quản lý bộ nhớ.
  + Quản lý **watchdog** để đảm bảo hệ thống không rơi vào tình trạng không đáp ứng.
  + Quản lý trạng thái của ECU (ví dụ như tắt/bật chế độ tiết kiệm năng lượng).
  + Chức năng xử lý lỗi và báo cáo (bao gồm bộ nhớ lỗi).

### ****Driver trong Kiến trúc AUTOSAR:****

Trong **kiến trúc AUTOSAR**, driver là các thành phần phần mềm chịu trách nhiệm điều khiển và truy cập các thiết bị phần cứng (cả bên trong và bên ngoài bộ điều khiển vi xử lý). Các driver giúp tách biệt phần mềm ứng dụng khỏi phần cứng, tạo điều kiện cho sự linh hoạt và khả năng tái sử dụng phần mềm, đồng thời hỗ trợ việc thay đổi phần cứng mà không cần thay đổi phần mềm ứng dụng.

#### ****Internal Driver (Trình điều khiển nội bộ)****

* **Thiết bị bên trong (Internal Devices)**:
  + Các thiết bị nằm bên trong **bộ điều khiển vi xử lý (µC)**, không yêu cầu kết nối bên ngoài.
  + Ví dụ về các thiết bị bên trong:
    - **Bộ nhớ EEPROM bên trong**: Lưu trữ dữ liệu không thay đổi (non-volatile).
    - **Bộ điều khiển CAN bên trong**: Điều khiển giao tiếp mạng CAN trong xe.
    - **Bộ chuyển đổi ADC bên trong**: Chuyển đổi tín hiệu analog từ cảm biến thành tín hiệu số.
* **Internal Driver (Trình điều khiển nội bộ)**:
  + Là các driver chịu trách nhiệm điều khiển các thiết bị bên trong bộ điều khiển vi xử lý.
  + Các driver này được đặt trong **Microcontroller Abstraction Layer (MCAL)** của **Basic Software**.
  + **Nhiệm vụ**: Các **internal driver** giúp phần mềm ứng dụng truy cập các thiết bị bên trong mà không cần biết chi tiết về phần cứng cụ thể của **µC**.
  + **Vị trí**: Được triển khai trong **MCAL**, giúp tách biệt phần mềm ứng dụng khỏi phần cứng của bộ điều khiển vi xử lý.

#### ****External Driver (Trình điều khiển bên ngoài)****

* **Thiết bị bên ngoài (External Devices)**:
  + Các thiết bị này không nằm trong bộ điều khiển vi xử lý mà được kết nối với ECU (Electronic Control Unit) qua các giao diện và cổng kết nối bên ngoài.
  + Ví dụ về các thiết bị bên ngoài:
    - **EEPROM bên ngoài**: Bộ nhớ không thay đổi được lưu trữ ngoài ECU, dùng để lưu trữ dữ liệu lâu dài.
    - **Bộ đếm watchdog bên ngoài**: Giám sát và đảm bảo hoạt động của ECU.
    - **Bộ nhớ flash bên ngoài**: Bộ nhớ lưu trữ cho phép ghi và xóa dữ liệu.
* **External Driver (Trình điều khiển bên ngoài)**:
  + Là các driver chịu trách nhiệm điều khiển và truy cập các thiết bị bên ngoài.
  + Các **external driver** được triển khai trong **ECU Abstraction Layer (ECU AL)**, giúp tách biệt phần mềm khỏi phần cứng bên ngoài.
  + **Nhiệm vụ**: Các **external driver** cho phép phần mềm ứng dụng truy cập các thiết bị ngoại vi một cách chuẩn hóa, chẳng hạn như bộ nhớ flash, bộ đếm watchdog hoặc EEPROM.
* **Kết nối với MCAL**:
  + Mặc dù các thiết bị bên ngoài có thể nằm ngoài ECU, các **external driver** này vẫn có thể tương tác với **Microcontroller Abstraction Layer (MCAL)** khi truy cập phần cứng thông qua các trình điều khiển của **MCAL**. Ví dụ, một driver cho EEPROM bên ngoài có thể sử dụng giao diện SPI để giao tiếp với bộ nhớ flash bên ngoài thông qua bus SPI, trong khi **MCAL** cung cấp driver cho bus SPI.

**Ví dụ về Driver bên ngoài:**

* **Driver cho EEPROM bên ngoài**:
  + EEPROM bên ngoài có thể được kết nối qua giao diện **SPI** (Serial Peripheral Interface).
  + Trình điều khiển sẽ thực hiện các thao tác đọc/ghi vào EEPROM bên ngoài thông qua giao diện SPI, sử dụng các chức năng do **MCAL** cung cấp cho giao diện SPI.

#### ****Trường hợp ngoại lệ (Exception):****

* Trong một số trường hợp đặc biệt, các thiết bị ngoại vi như bộ nhớ flash ngoại hoặc các thiết bị khác có thể được ánh xạ bộ nhớ trực tiếp. Điều này có nghĩa là chúng có thể được truy cập trực tiếp từ bộ điều khiển vi xử lý mà không cần qua các trình điều khiển của **ECU Abstraction Layer**.
* **Trình điều khiển trực tiếp này** sẽ được đặt trong **Microcontroller Abstraction Layer (MCAL)** vì chúng phụ thuộc vào cấu trúc phần cứng của **µC**.

### Interface

Trong **kiến trúc AUTOSAR**, **Interface** (mô-đun giao diện) đóng vai trò quan trọng trong việc cung cấp khả năng trừu tượng hóa giữa phần mềm và phần cứng. Interface không chỉ giúp tách biệt các mô-đun phần mềm mà còn hỗ trợ phần mềm ứng dụng giao tiếp với phần cứng mà không cần phải quan tâm đến chi tiết triển khai của phần cứng cụ thể.

#### ****Khái niệm Interface (Giao diện)****

* **Định nghĩa**:
  + Một **interface module** là một mô-đun phần mềm chịu trách nhiệm trừu tượng hóa các mô-đun hoặc thiết bị dưới nó.
  + Nó cung cấp một **API chuẩn** giúp phần mềm truy cập và tương tác với các thiết bị mà không phụ thuộc vào cấu hình phần cứng cụ thể hoặc số lượng thiết bị có trong hệ thống.
  + Mục tiêu của interface là **ẩn đi sự phức tạp** của phần cứng, từ đó cung cấp một cách tiếp cận nhất quán và dễ dàng cho các mô-đun phần mềm khác.
* **Chức năng**:
  + Interface cung cấp một **API chung** cho phép phần mềm ứng dụng truy cập vào các thiết bị hoặc hệ thống mà không cần quan tâm đến số lượng thiết bị trong hệ thống hoặc các khác biệt trong triển khai phần cứng.
  + Ví dụ, giao diện cho một **hệ thống CAN** sẽ cung cấp một API chung cho việc giao tiếp với các bộ điều khiển CAN, bất kể có bao nhiêu bộ điều khiển CAN trong ECU và bất kể cách thức thực hiện phần cứng của chúng (trong chip hay ngoài chip).
* **Đặc điểm quan trọng**:
  + **Không thay đổi nội dung dữ liệu**: Interface không thay đổi hoặc xử lý nội dung của dữ liệu, nó chỉ định hướng và cung cấp các dịch vụ truy cập.
  + **Trừu tượng hóa phần cứng**: Interface che giấu các chi tiết cụ thể về phần cứng và cung cấp một cách tiếp cận chung để giao tiếp với các thiết bị.

#### ****Vị trí của Interface trong Kiến trúc AUTOSAR****

* **ECU Abstraction Layer (ECU AL)**: Các interface thường được đặt trong lớp **ECU Abstraction Layer (ECU AL)** của **Basic Software**. Lớp này chịu trách nhiệm trừu tượng hóa các chi tiết phần cứng của ECU và cung cấp các dịch vụ chuẩn hóa cho các mô-đun phần mềm ứng dụng.
* **Ví dụ về Interface**:
  + **Giao diện CAN**: Một **giao diện cho hệ thống giao tiếp CAN** sẽ cung cấp một API chuẩn để các mô-đun phần mềm giao tiếp với mạng CAN mà không cần biết chi tiết về cách thức triển khai phần cứng của các bộ điều khiển CAN.
  + Giao diện này sẽ hỗ trợ việc truy cập tới tất cả các bộ điều khiển CAN trong ECU, bất kể có bao nhiêu bộ CAN controllers và bất kể bộ điều khiển CAN được triển khai như thế nào (bên trong hay bên ngoài chip).

#### ****Ưu điểm của Interface trong AUTOSAR****

* **Tính linh hoạt**: Nhờ vào việc trừu tượng hóa phần cứng, các interface cho phép phần mềm ứng dụng có thể hoạt động trên nhiều phần cứng khác nhau mà không cần thay đổi.
* **Đơn giản hóa phát triển phần mềm**: Các mô-đun phần mềm ứng dụng có thể sử dụng API của các interface mà không cần phải lo lắng về chi tiết phần cứng của thiết bị.
* **Khả năng tái sử dụng**: Phần mềm có thể tái sử dụng giữa các ECU khác nhau hoặc giữa các thiết bị khác nhau mà không phải thay đổi mã nguồn.

### ****Handler trong Kiến trúc AUTOSAR****

Trong **kiến trúc AUTOSAR**, **Handler** là một thành phần phần mềm quan trọng trong việc quản lý và điều khiển quá trình truy cập đến các thiết bị hoặc tài nguyên, đặc biệt là khi có nhiều khách hàng (client) yêu cầu truy cập cùng một lúc. Handler đảm nhận chức năng quản lý truy cập đồng thời, đa luồng và không đồng bộ đến các **driver** hoặc **interface** trong hệ thống.

#### ****Khái niệm về Handler****

* **Định nghĩa**:
  + **Handler** là một giao diện cụ thể giúp **quản lý truy cập đồng thời** giữa nhiều khách hàng (client) đến một hoặc nhiều trình điều khiển (driver). Các khách hàng có thể yêu cầu truy cập vào các tài nguyên phần cứng hoặc các dịch vụ phần mềm như bộ điều khiển SPI, ADC, CAN, v.v.
  + Nó thực hiện các **chức năng quản lý tài nguyên** như **lưu trữ tạm thời**, **queuing** (đặt vào hàng đợi), **arbitration** (phân bổ tài nguyên), và **multiplexing** (điều phối truy cập). Các chức năng này giúp đảm bảo rằng các yêu cầu truy cập được xử lý một cách hợp lý và công bằng, đặc biệt khi nhiều yêu cầu đến cùng một lúc.
* **Chức năng chính**:
  + **Lưu trữ tạm thời**: Handler có thể lưu trữ tạm thời các yêu cầu truy cập nếu tài nguyên hiện tại đang bận hoặc không sẵn sàng.
  + **Queuing (Xử lý hàng đợi)**: Các yêu cầu truy cập sẽ được xếp vào hàng đợi, chờ đến lượt để được xử lý.
  + **Arbitration (Trọng tài)**: Khi có nhiều khách hàng yêu cầu truy cập vào tài nguyên cùng lúc, handler sẽ quyết định thứ tự ưu tiên và cho phép tài nguyên được cấp phát một cách hợp lý.
  + **Multiplexing**: Quản lý việc chia sẻ tài nguyên giữa nhiều khách hàng một cách hiệu quả, đảm bảo rằng tài nguyên được sử dụng tối ưu mà không gây xung đột.
* **Không thay đổi nội dung dữ liệu**:
  + Handler không thay đổi nội dung của dữ liệu mà chỉ kiểm soát việc truy cập và phân bổ tài nguyên. Việc xử lý dữ liệu và các phép toán tính toán thường là nhiệm vụ của **driver** hoặc **giao diện**.

### ****Ứng dụng của Handler trong AUTOSAR****

Các **handlers** thường được tích hợp trực tiếp vào các **driver** hoặc **giao diện** cho các hệ thống như **SPI (Serial Peripheral Interface)**, **ADC (Analog-to-Digital Converter)**, **CAN (Controller Area Network)**, v.v. Cụ thể:

1. **SPIHandler**:
   * Trong **giao diện SPI**, handler có thể quản lý việc gửi và nhận dữ liệu giữa các bộ vi xử lý và các thiết bị ngoại vi như EEPROM, bộ điều khiển CAN, v.v. Nếu nhiều thiết bị cùng yêu cầu truy cập vào giao diện SPI, handler sẽ quyết định thứ tự xử lý các yêu cầu này.
2. **ADC Driver**:
   * Trong trường hợp của **ADC (Bộ chuyển đổi tín hiệu tương tự sang số)**, một **ADC Handler** sẽ quản lý các yêu cầu đọc tín hiệu từ các kênh đầu vào ADC. Khi có nhiều cảm biến hoặc tín hiệu tương tự cần được chuyển đổi, handler sẽ xếp các yêu cầu vào hàng đợi và xử lý lần lượt.
3. **CAN Driver**:
   * Trong trường hợp **CAN (Controller Area Network)**, handler sẽ giúp quản lý các thông điệp giao tiếp giữa các ECU và các thiết bị trong mạng CAN. Các thông điệp này có thể đến từ các nguồn khác nhau và handler sẽ quyết định cách thức xử lý chúng, tránh các tình huống xung đột.

### ****Tích hợp Handler vào Driver hoặc Interface****

* **SPIHandlerDriver**: Trong hệ thống AUTOSAR, **SPI Handler** có thể được tích hợp vào driver của SPI. Driver này sẽ chịu trách nhiệm quản lý việc đọc/ghi dữ liệu từ và đến các thiết bị qua giao thức SPI. Handler trong trường hợp này đảm nhận việc xử lý các yêu cầu đồng thời từ nhiều nguồn.
* **ADC Driver**: Tương tự, trong **driver ADC**, handler sẽ quản lý các yêu cầu truy cập ADC từ các phần mềm ứng dụng. Handler đảm bảo rằng các tín hiệu được đọc và chuyển đổi chính xác mà không bị xung đột khi nhiều yêu cầu đến cùng một lúc.

### ****Manager trong Kiến trúc AUTOSAR****

Trong **kiến trúc AUTOSAR**, một **Manager** là một thành phần phần mềm cao cấp cung cấp các dịch vụ cho nhiều khách hàng, đặc biệt trong các tình huống mà chức năng của **Handler** không đủ để đáp ứng các yêu cầu. Manager đóng vai trò quan trọng trong việc **trừu tượng hóa các dịch vụ**, quản lý và điều chỉnh **dữ liệu** để phục vụ các yêu cầu đồng thời từ nhiều khách hàng khác nhau.

#### ****Khái niệm về Manager:****

* **Định nghĩa**:
  + **Manager** là một mô-đun phần mềm chuyên cung cấp các dịch vụ cho các thành phần phần mềm khác (khách hàng) trong hệ thống. Nó có thể điều khiển việc truy cập tài nguyên, đồng thời thực hiện việc xử lý và điều chỉnh dữ liệu mà các khách hàng yêu cầu.
  + **Chức năng** của manager vượt ra ngoài khả năng của **handler**: ngoài việc **quản lý truy cập** đồng thời, manager còn có thể **đánh giá**, **thay đổi**, hoặc **điều chỉnh dữ liệu** theo các quy trình cụ thể. Điều này đặc biệt hữu ích trong các tình huống mà nhiều khách hàng yêu cầu truy cập vào các tài nguyên cùng lúc, và dữ liệu cần được xử lý trước khi trả về cho khách hàng.
* **Chức năng chính**:
  + **Quản lý đồng thời**: Manager giúp quản lý việc truy cập tài nguyên hoặc thiết bị từ nhiều khách hàng, đồng thời đảm bảo rằng các yêu cầu được xử lý một cách hợp lý và công bằng.
  + **Đánh giá và điều chỉnh dữ liệu**: Ngoài việc cung cấp dịch vụ, manager có thể thực hiện các hành động cần thiết như kiểm tra tính hợp lệ của dữ liệu, thay đổi dữ liệu khi cần thiết, hoặc điều chỉnh thông tin trước khi cung cấp cho khách hàng.
  + **Lưu trữ và cung cấp dữ liệu**: Manager cũng có thể thực hiện các dịch vụ lưu trữ phân tán, cung cấp giá trị mặc định, và đảm bảo tính toàn vẹn của dữ liệu.

#### ****Ví dụ về Manager trong AUTOSAR****

1. **NVRAM Manager**:
   * **NVRAM Manager** là một ví dụ điển hình về manager trong AUTOSAR. Nó quản lý việc truy cập đồng thời vào các thiết bị bộ nhớ như **flash memory** và **EEPROM** (cả nội bộ và ngoại vi). NVRAM Manager thực hiện một loạt các dịch vụ, bao gồm:
     + **Lưu trữ dữ liệu phân tán và đáng tin cậy**: Dữ liệu có thể được lưu trữ vào các thiết bị bộ nhớ không bay hơi (NVRAM) và phải được đảm bảo rằng chúng không bị mất khi hệ thống mất điện.
     + **Kiểm tra dữ liệu**: Trước khi dữ liệu được trả về cho khách hàng, NVRAM Manager có thể kiểm tra tính hợp lệ của nó để đảm bảo rằng không có lỗi hoặc sự cố trong dữ liệu.
     + **Cung cấp các giá trị mặc định**: Khi không có dữ liệu sẵn có, manager có thể cung cấp các giá trị mặc định cho các thành phần yêu cầu.
   * Các chức năng này rất quan trọng trong các hệ thống xe hơi, nơi việc quản lý dữ liệu bộ nhớ là cực kỳ quan trọng đối với hiệu suất và độ tin cậy của hệ thống.
2. **Memory Manager**:
   * **Memory Manager** là một manager khác chịu trách nhiệm quản lý truy cập vào bộ nhớ. Các dịch vụ có thể bao gồm:
     + **Quản lý phân bổ bộ nhớ**: Memory Manager có thể quản lý việc phân bổ và giải phóng bộ nhớ cho các ứng dụng khác nhau trong hệ thống.
     + **Kiểm tra bộ nhớ**: Đảm bảo rằng các khu vực bộ nhớ không bị hỏng hoặc bị ghi đè ngoài ý muốn.
     + **Cung cấp bộ nhớ tạm thời**: Cung cấp bộ nhớ tạm thời cho các tác vụ có tính chất tạm thời trong hệ thống.
3. **Communication Manager**:
   * **Communication Manager** có thể quản lý tất cả các giao tiếp trong một ECU, từ giao tiếp giữa các phần mềm trong ECU đến giao tiếp giữa ECU với các hệ thống khác trong xe.
     + **Quản lý các giao thức giao tiếp**: Đảm bảo rằng các giao thức như CAN, LIN, Ethernet, v.v., được quản lý một cách hiệu quả và dữ liệu có thể được truyền tải một cách chính xác và an toàn.
     + **Kiểm tra tính toàn vẹn của dữ liệu**: Kiểm tra và xác minh các gói dữ liệu trước khi truyền đi và sau khi nhận.

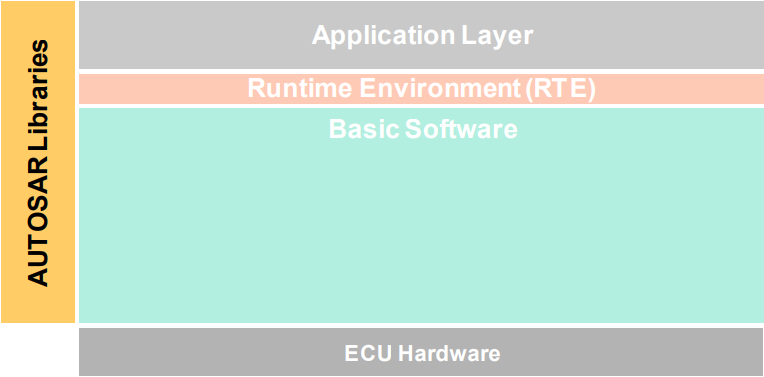
#### ****Tích hợp Manager vào Kiến trúc AUTOSAR****:

* Các **manager** thường được tích hợp vào **Services Layer** trong kiến trúc AUTOSAR. Services Layer cung cấp các dịch vụ cơ bản cho các phần mềm ứng dụng và các mô-đun cơ sở, và đây là nơi các **manager** như **NVRAM Manager**, **Memory Manager**, **Communication Manager**, v.v., được triển khai để hỗ trợ các yêu cầu quản lý tài nguyên và xử lý dữ liệu.
* Các **manager** này giúp nâng cao tính khả dụng, hiệu quả và độ tin cậy của hệ thống phần mềm, đồng thời đảm bảo rằng các tài nguyên được sử dụng tối ưu và an toàn.

#### ****Tóm lại****

* **Manager** trong **kiến trúc AUTOSAR** là một mô-đun phần mềm chịu trách nhiệm cung cấp các dịch vụ cụ thể cho nhiều khách hàng trong hệ thống. Bên cạnh việc quản lý truy cập đồng thời (như **handler**), manager còn có thể **đánh giá**, **thay đổi** hoặc **điều chỉnh dữ liệu**. Điều này làm cho manager trở thành thành phần không thể thiếu trong các hệ thống phức tạp, nơi nhiều khách hàng yêu cầu truy cập và thao tác với dữ liệu hoặc tài nguyên.
* Các **manager** như **NVRAM Manager**, **Memory Manager**, và **Communication Manager** cung cấp các dịch vụ quan trọng để quản lý bộ nhớ, dữ liệu và giao tiếp trong hệ thống, đảm bảo tính toàn vẹn và hiệu quả hoạt động của ECU.

### Giới thiệu về Thư viện



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

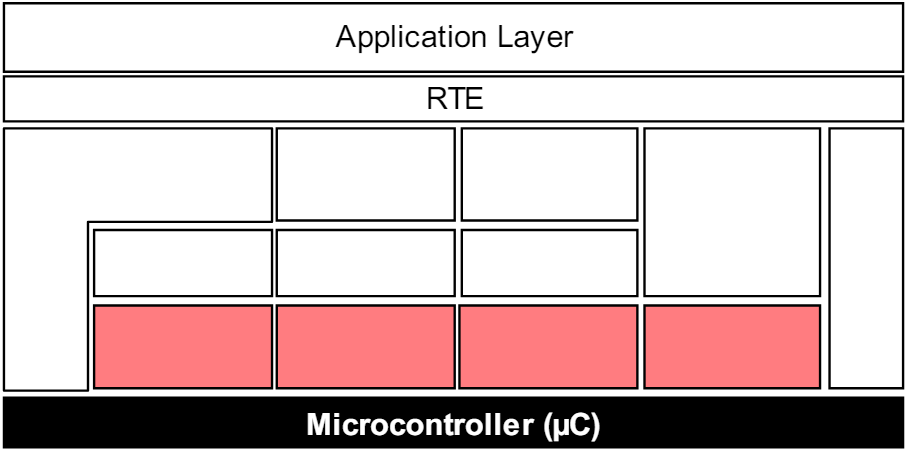
Thư viện:

* 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**

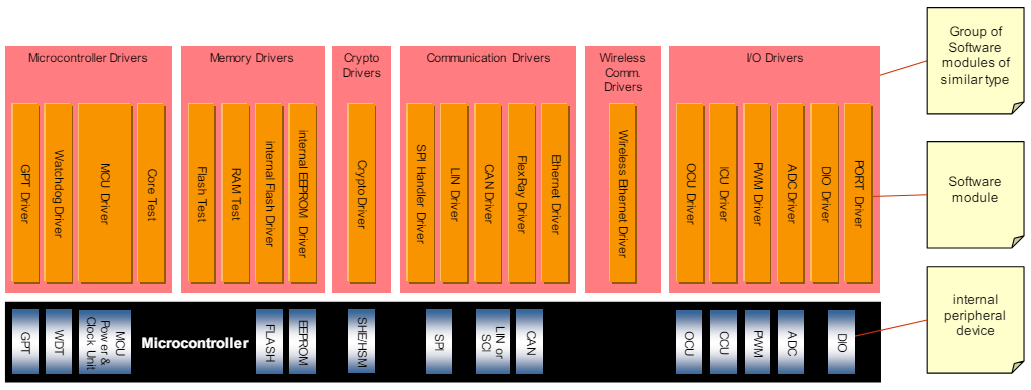
Trình điều khiển cho các thiết bị bộ nhớ trên chip (ví dụ: Flash internal, EEPROM internal) và các thiết bị bộ nhớ ngoại ánh xạ bộ nhớ (ví dụ: Flash external)

**➢ I/O Drivers**

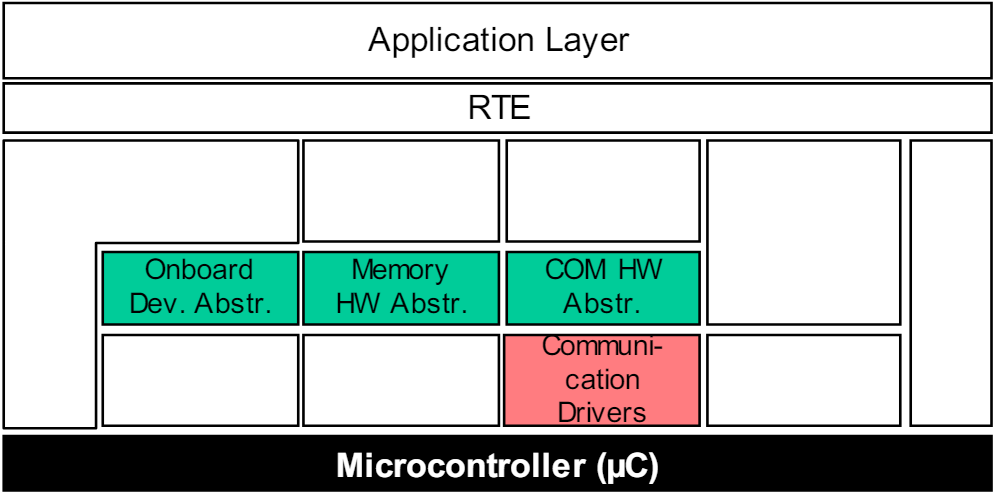
Trình điều khiển cho analog và digital I/O (ví dụ: ADC, PWM, DIO)

**➢ 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)

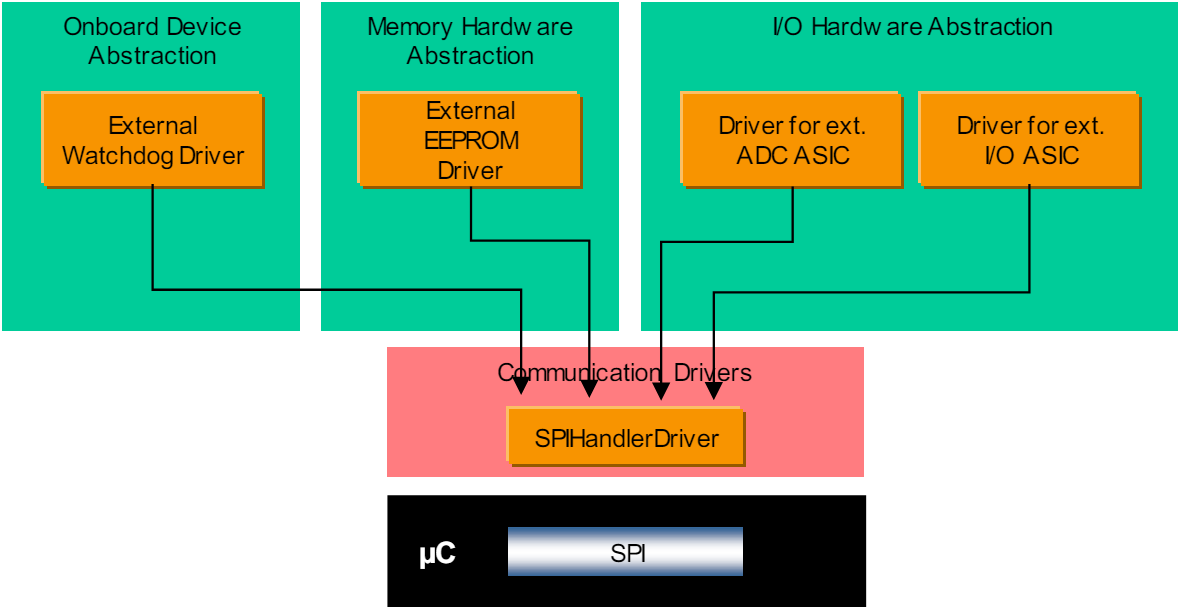


### Microcontroller Abstraction Layer: SPIHandlerDriver

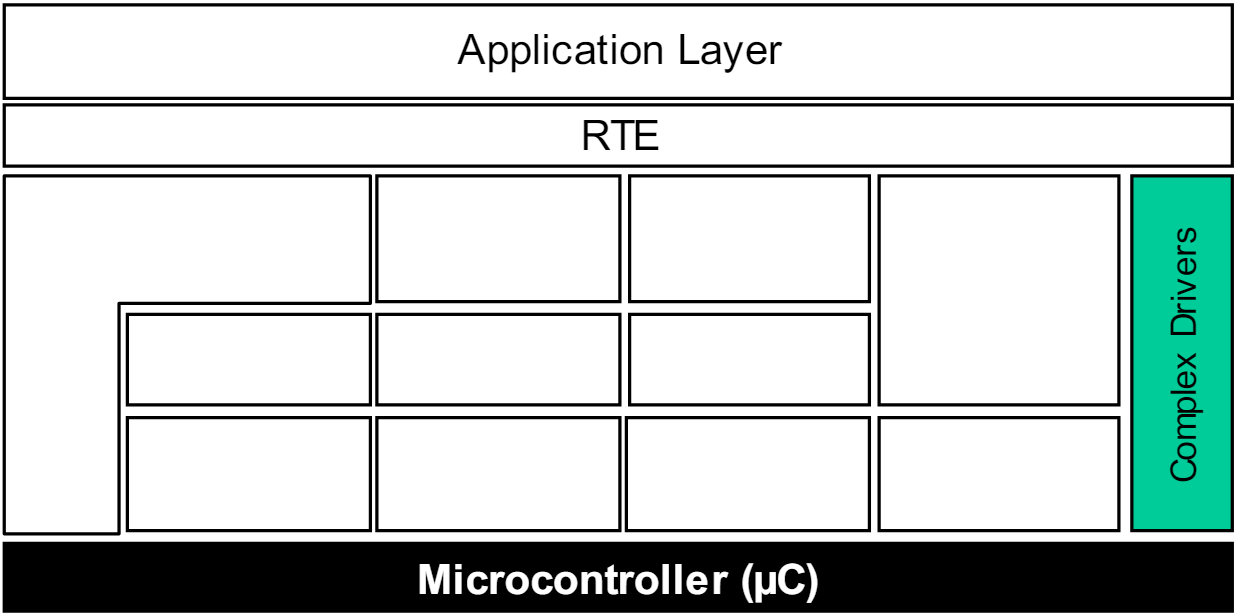


**SPIHandlerDriver** cho phép truy cập đồng thời của nhiều khách hàng vào một hoặc nhiều bus SPI.

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:

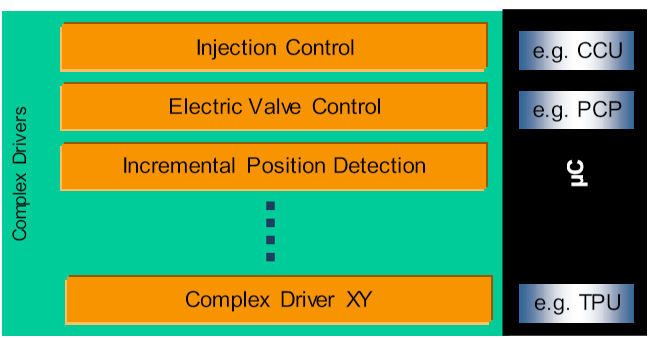
* Kiểm soát phun nhiên liệu
* Kiểm soát van điện
* Phát hiện vị trí tăng dần

***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!

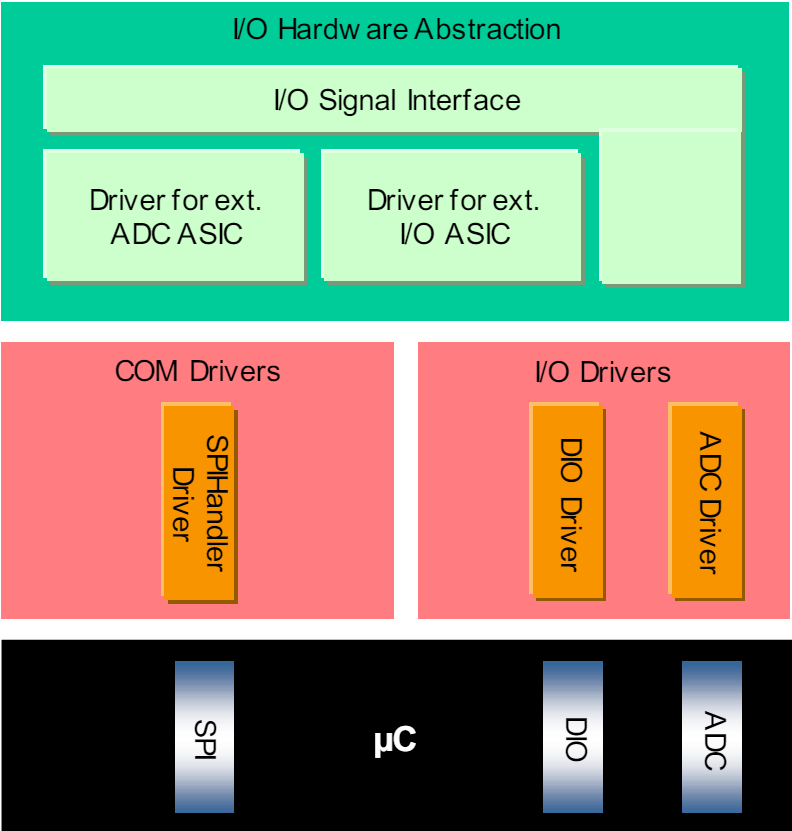
Các thiết bị I/O khác nhau có thể được truy cập thông qua một giao diện tín hiệu I/O.

***Nhiệm vụ:***

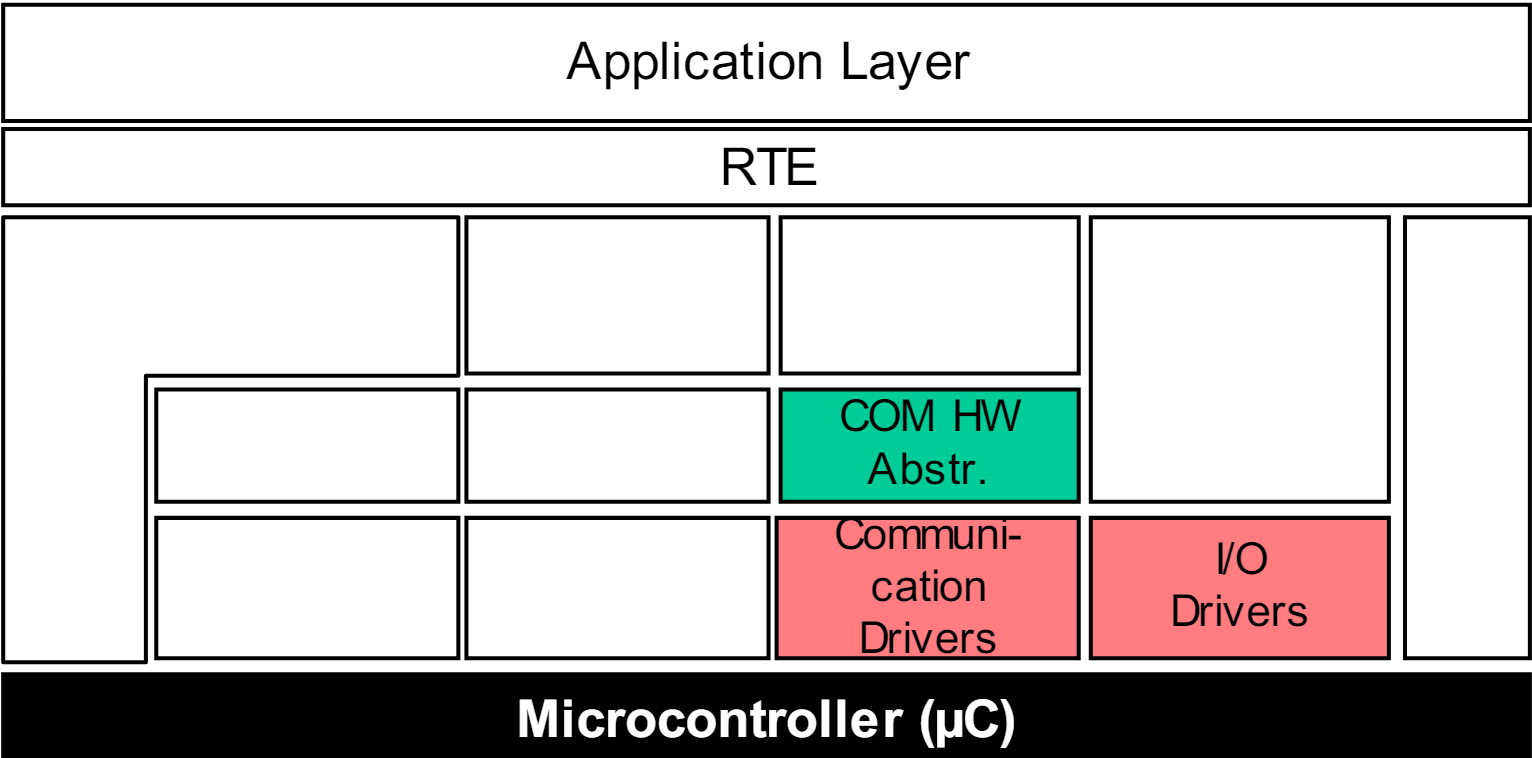
* Đại diện cho các tín hiệu I/O như chúng được kết nối với phần cứng của ECU hardware (e.g. current, voltage, frequency).
* Ẩn các thuộc tính bố cục và phần cứng ECU từ mức cao hơn lớp phần mềm

***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).

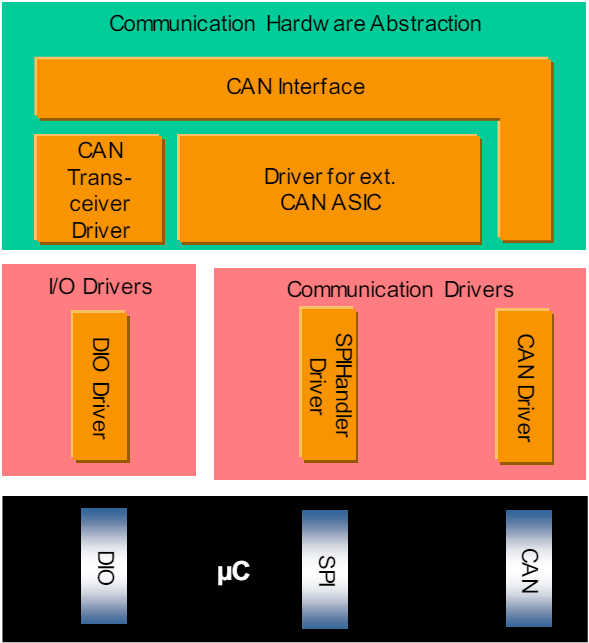
Nhiệm vụ:

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)

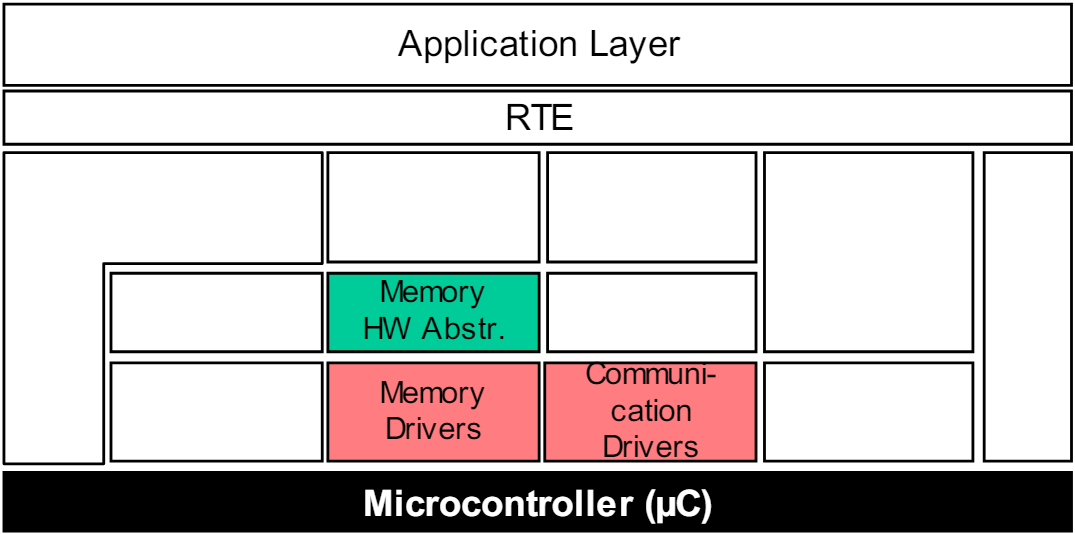
Thuộc tính:

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.

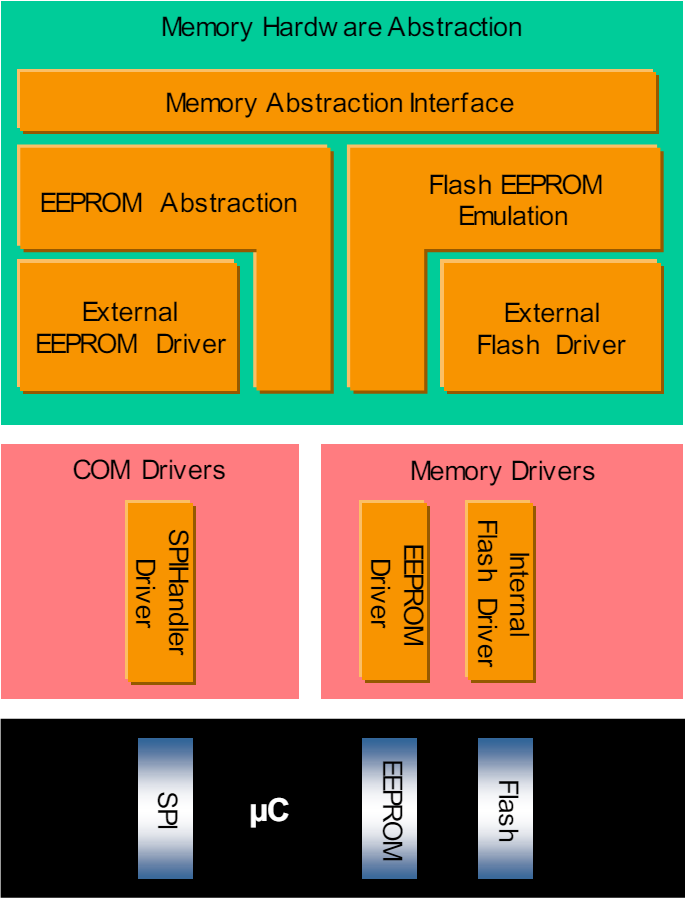
Nhiệm vụ:

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).

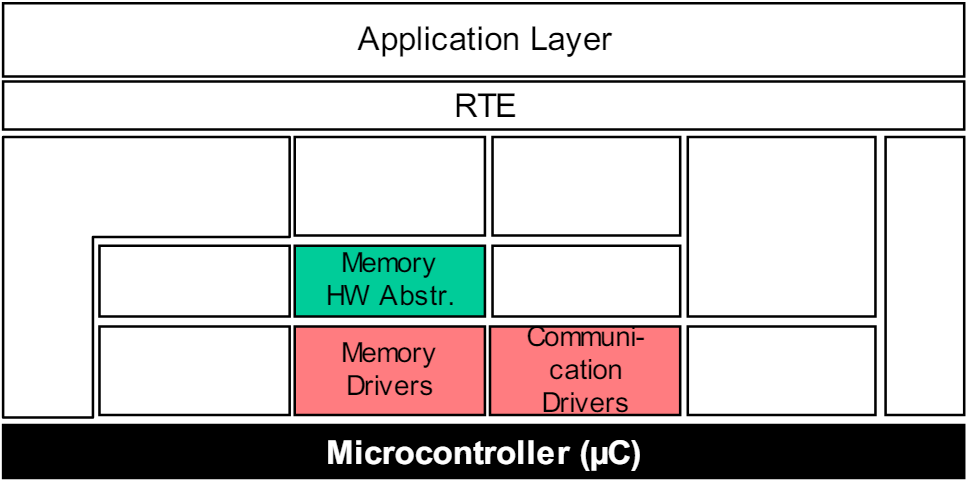
Thuộc tính:

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.

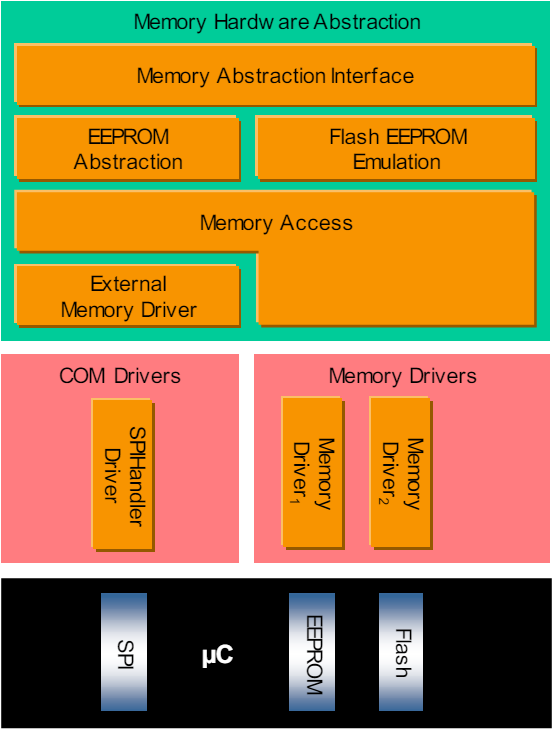
Nhiệm vụ:

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).

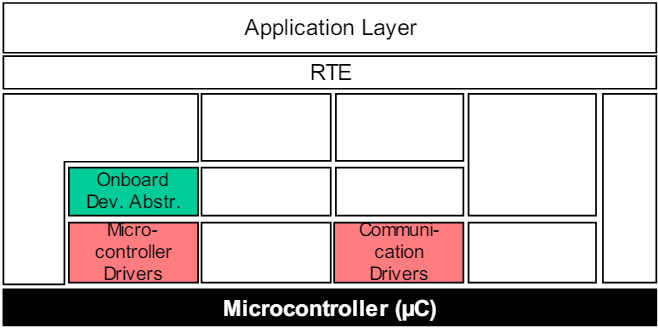
Thuộc tính:

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.

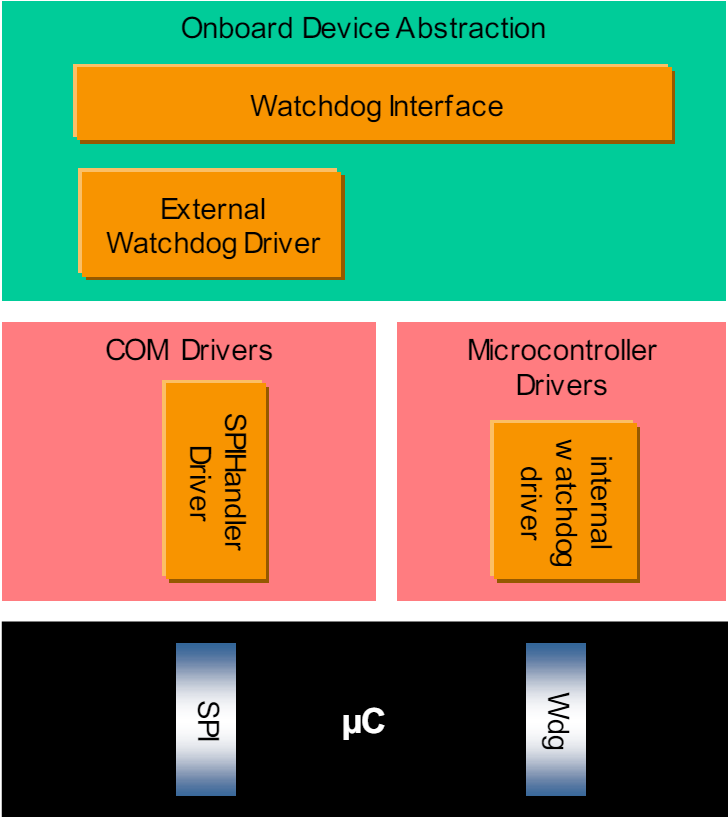
Nhiệm vụ:

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

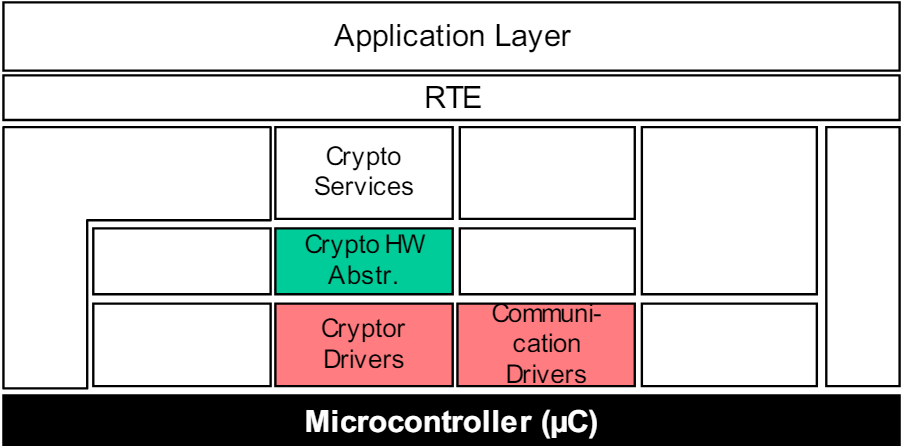
Thuộc tính:

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.

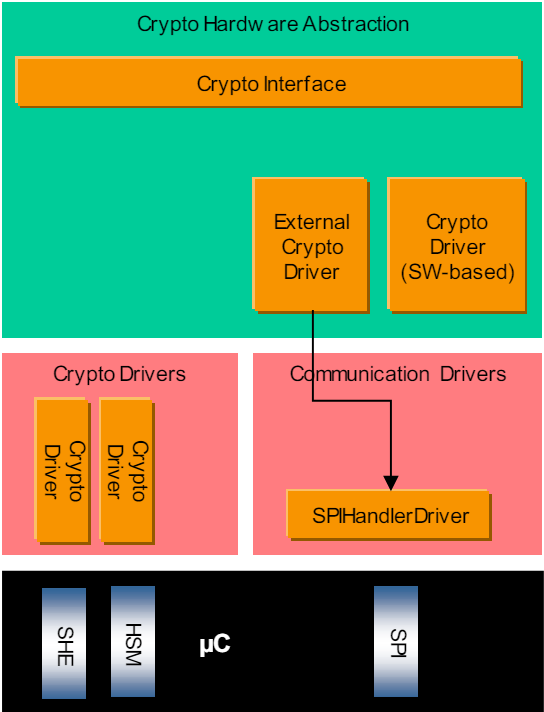
Nhiệm vụ:

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

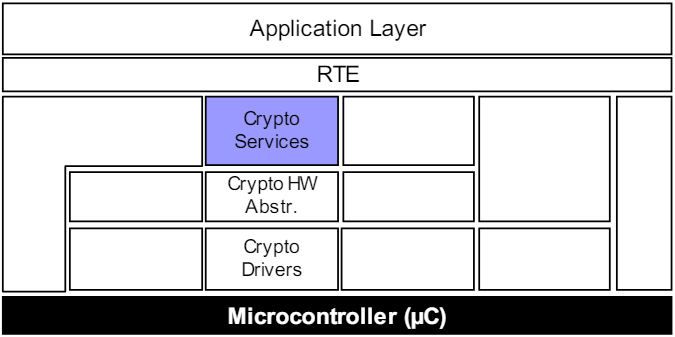
Thuộc tính:

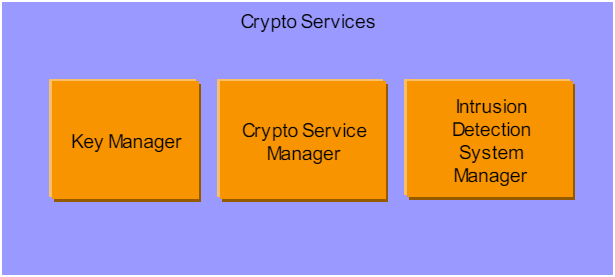
Triển khai: độc lập với vi điều khiển

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

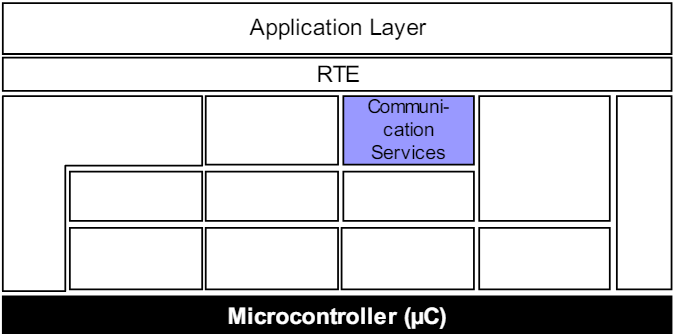


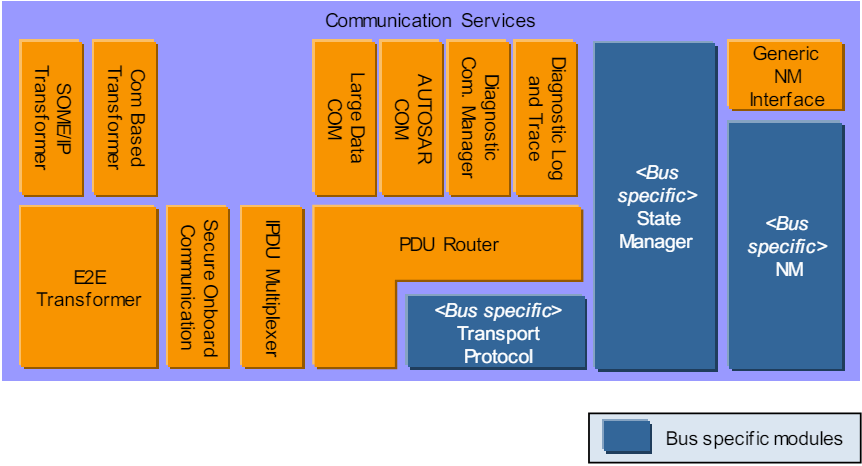
### Services: Crypto Services



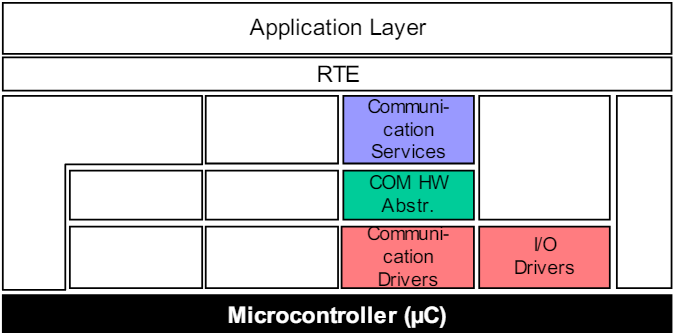


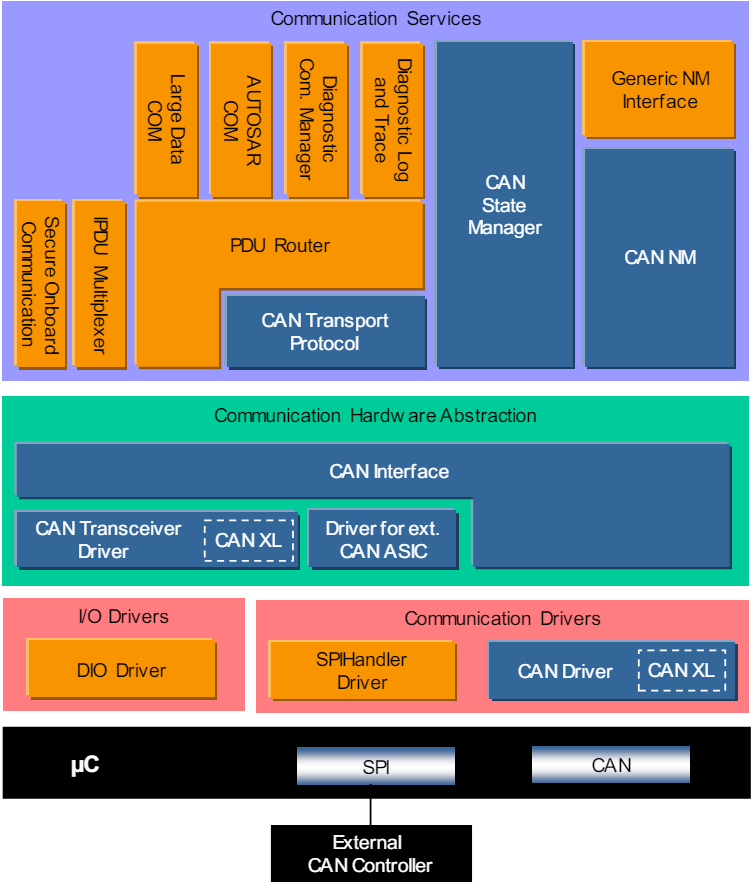
### Communication Services – General





### Communication Stack – CAN

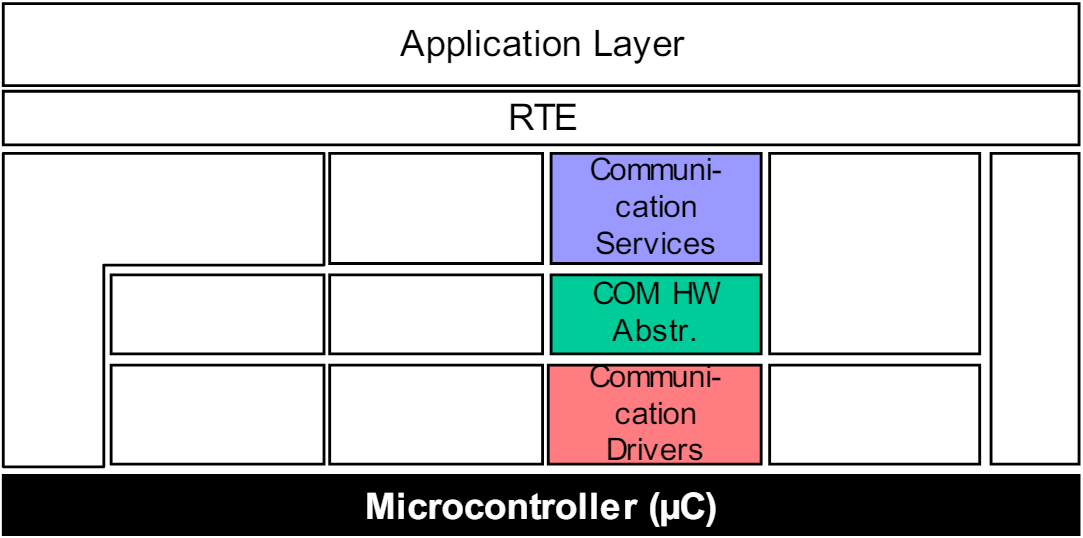


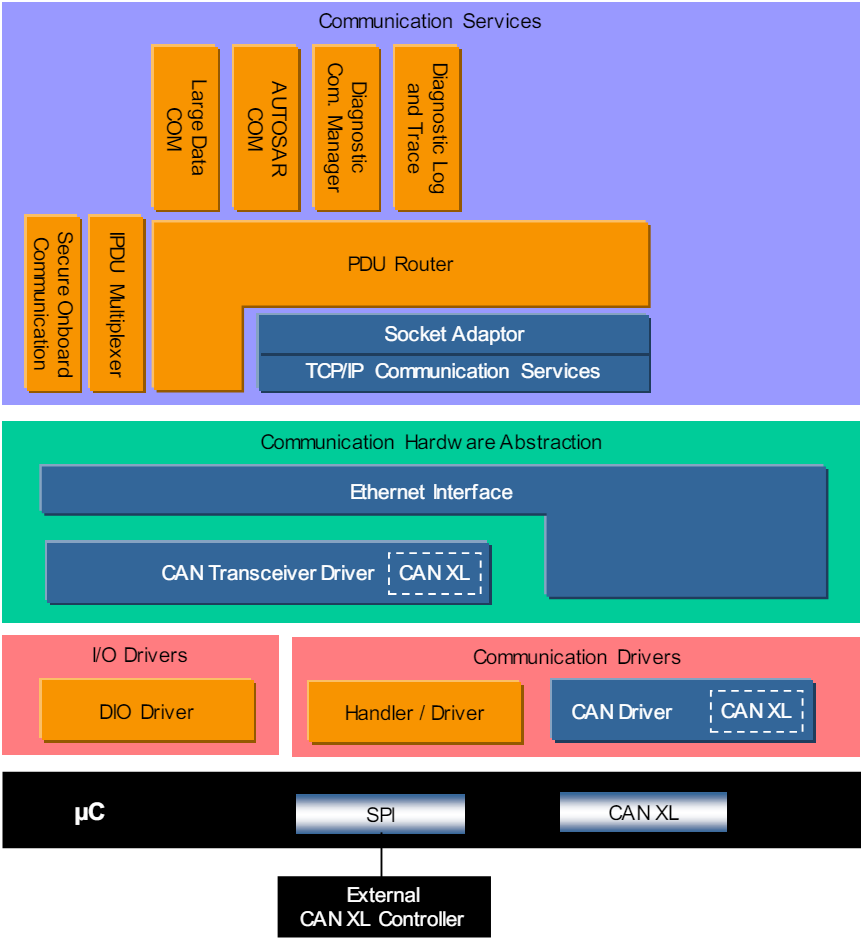


### Communication Stack – CAN

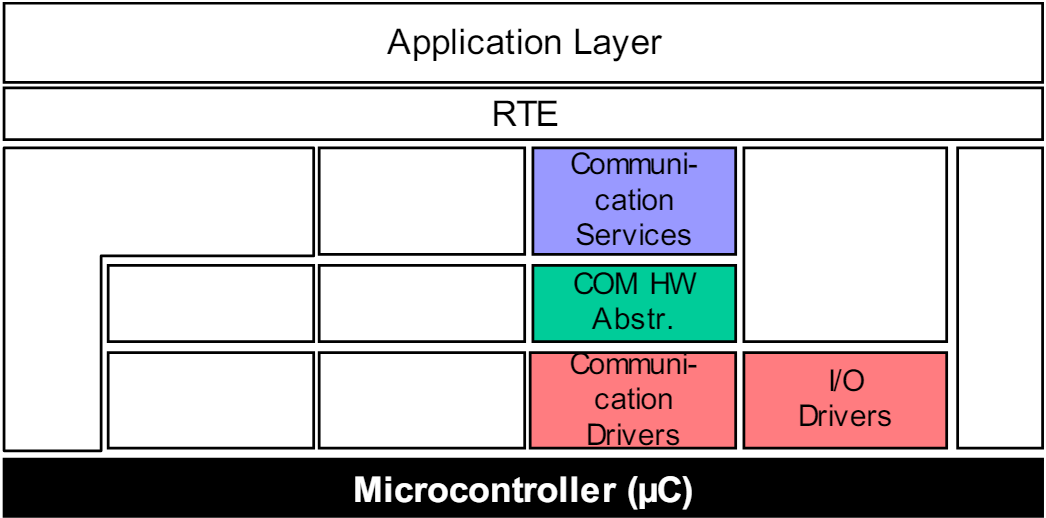


### Communication Stack – Ethernet/CAN XL

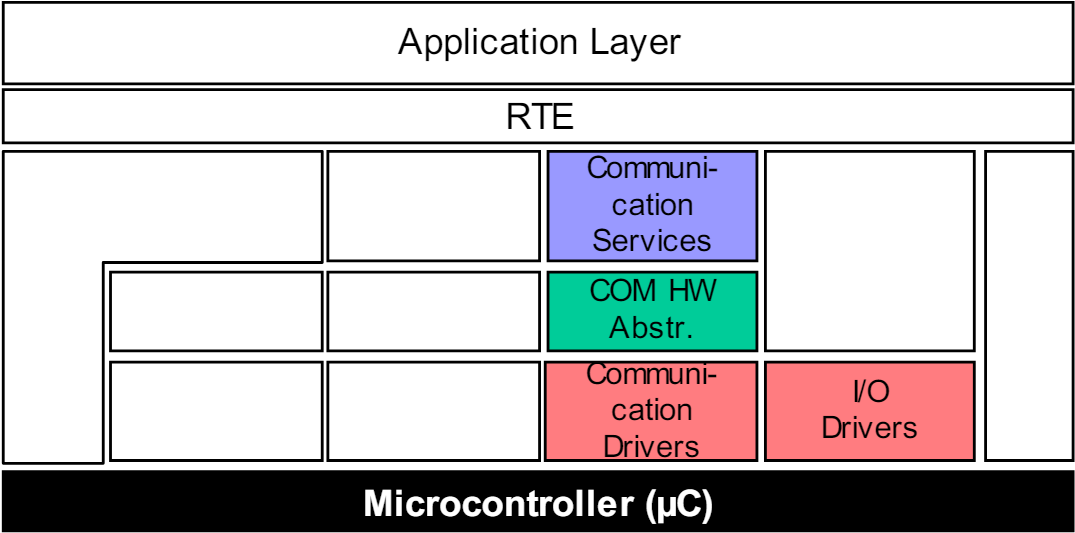


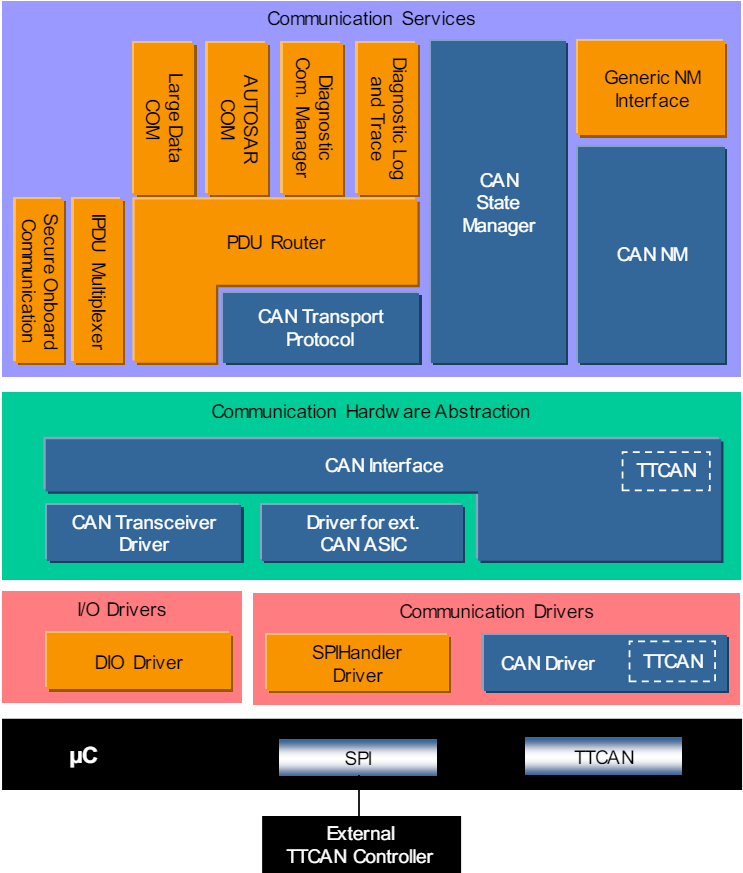


### Communication Stack Extension – CAN XL

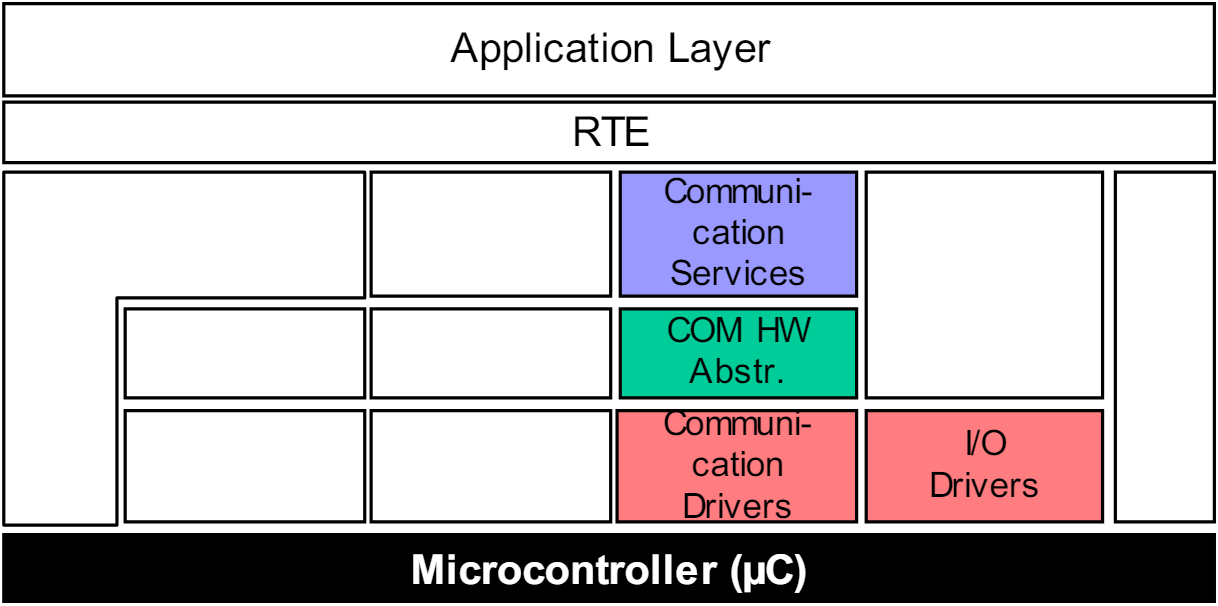


### Communication Stack Extension – TTCAN

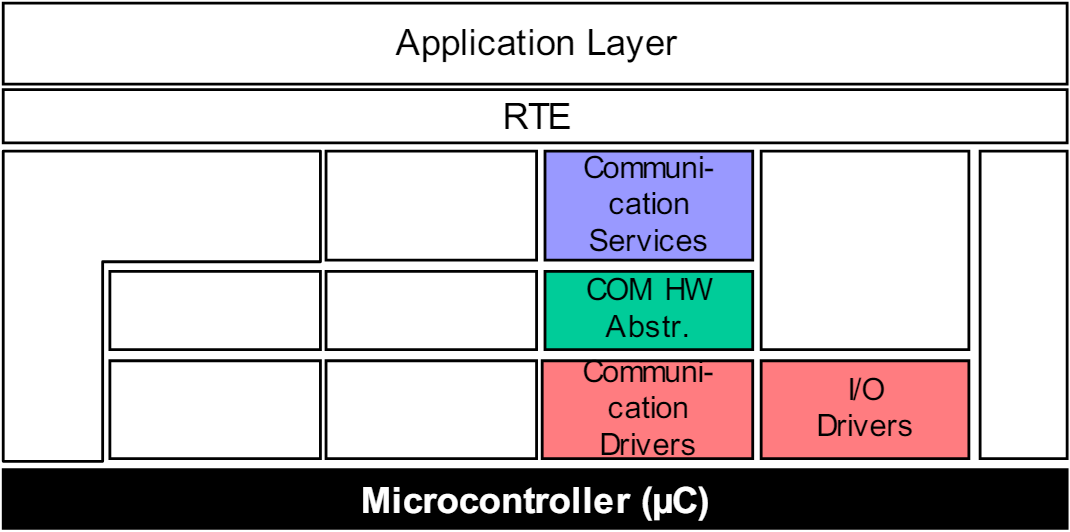


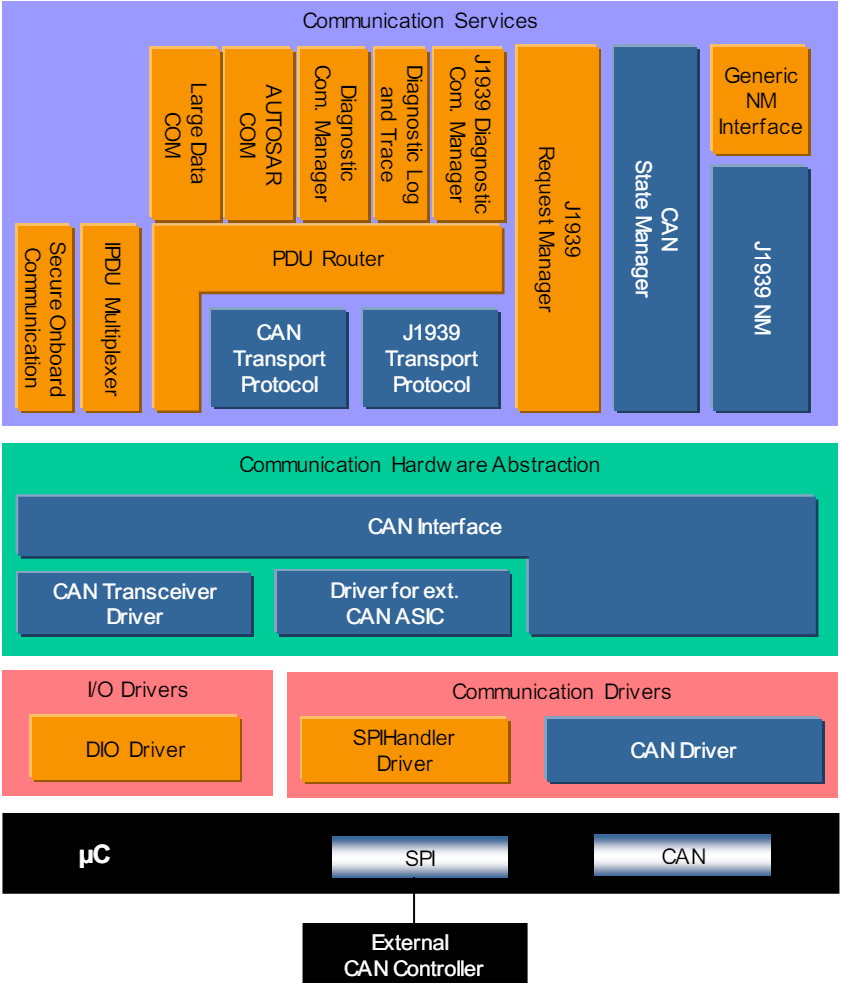


### Communication Stack Extension – TTCAN

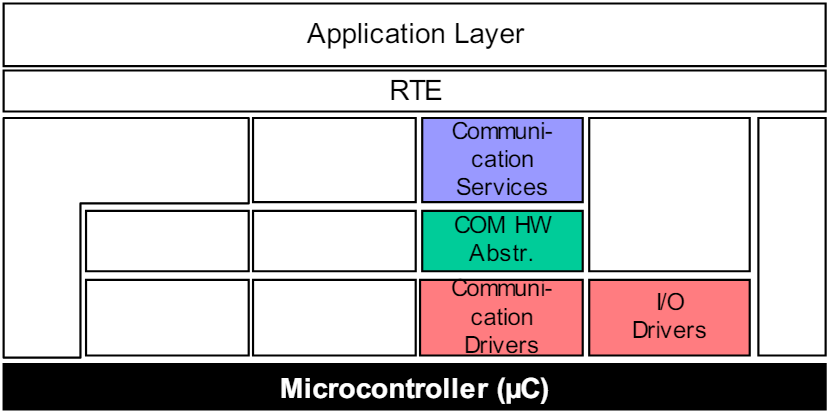


### Communication Stack Extension – J1939

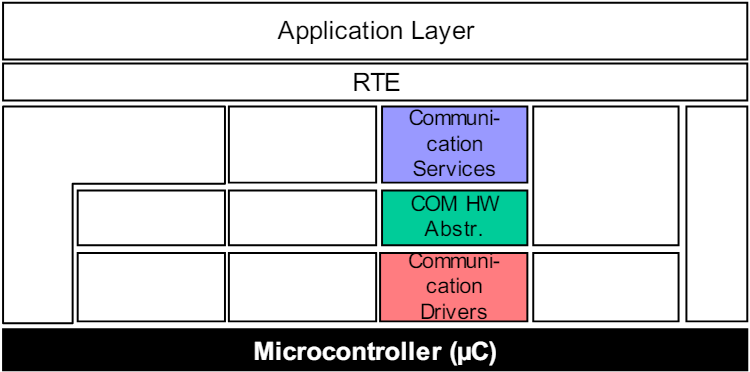


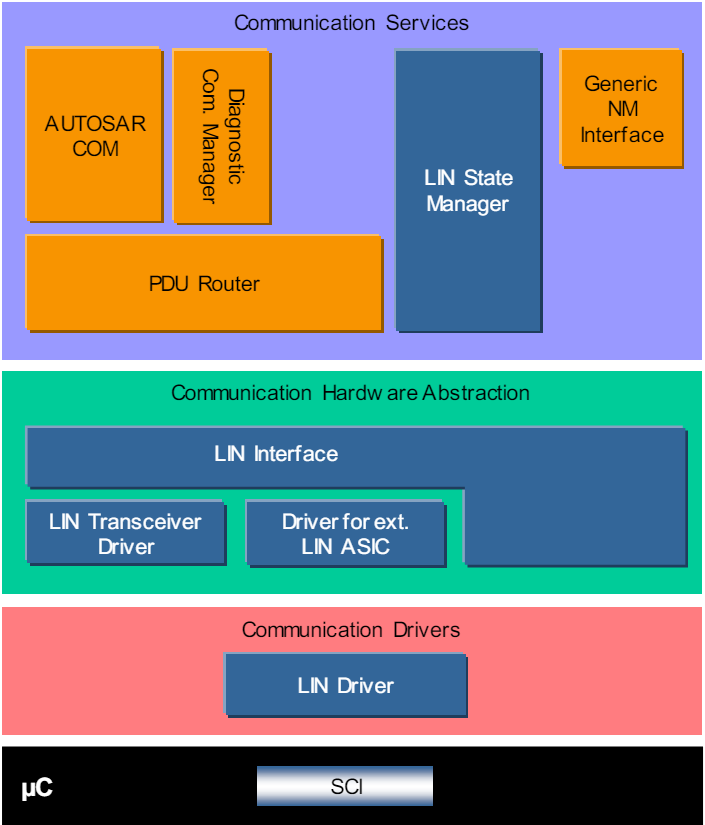


### Communication Stack Extension – J1939

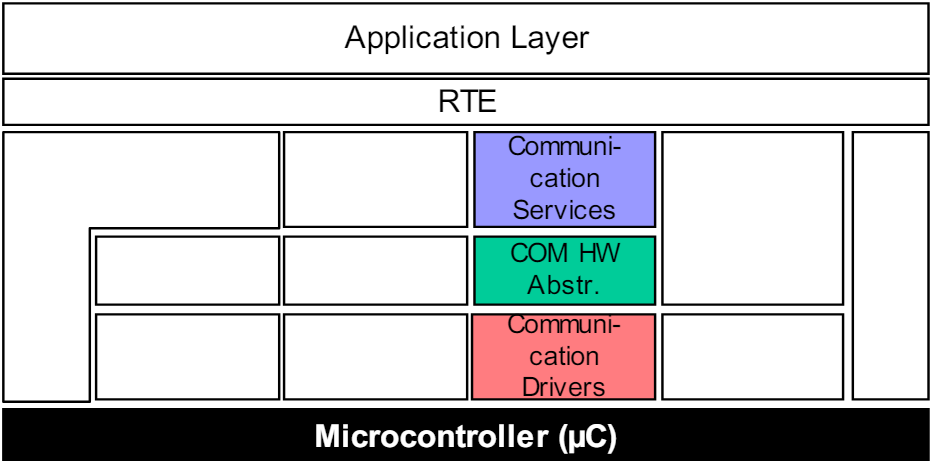


### Communication Stack – LIN

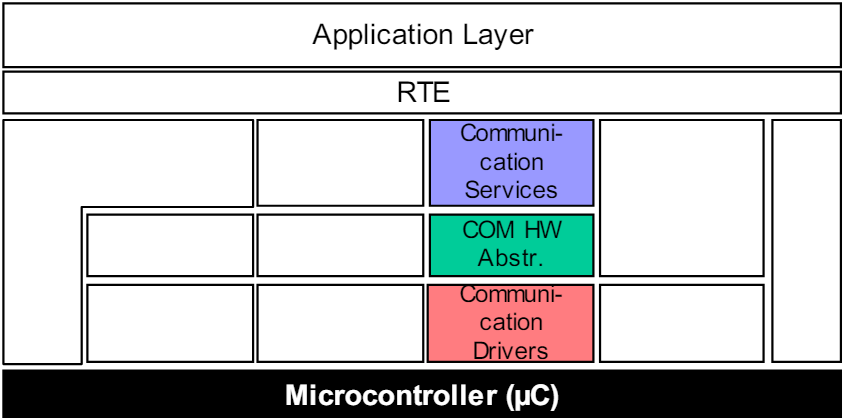


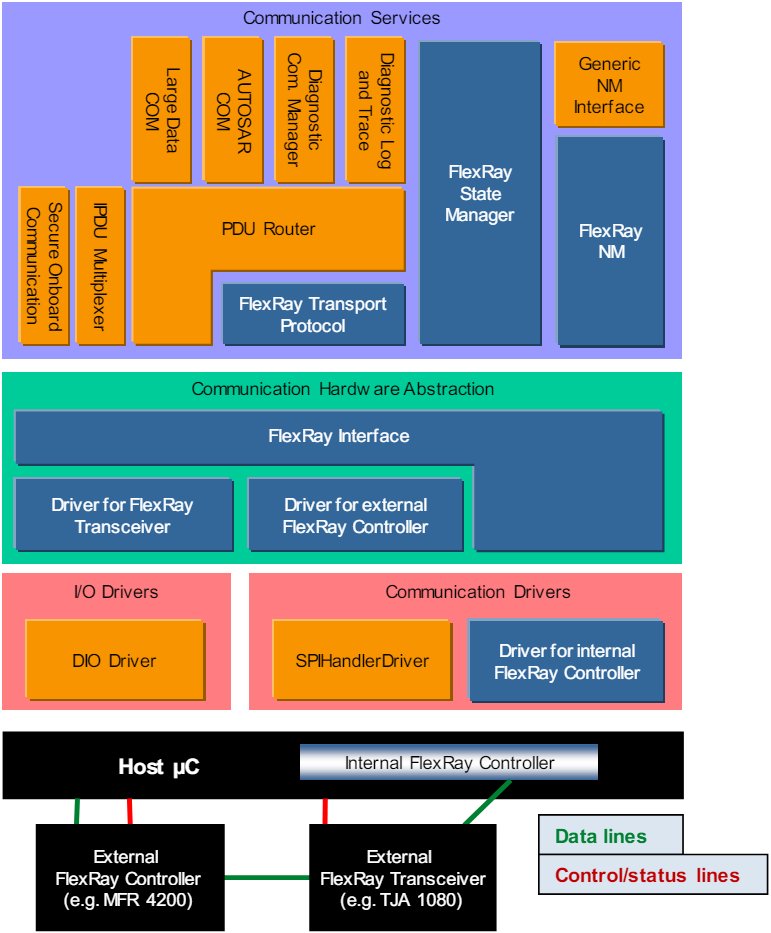


### Communication Stack – LIN

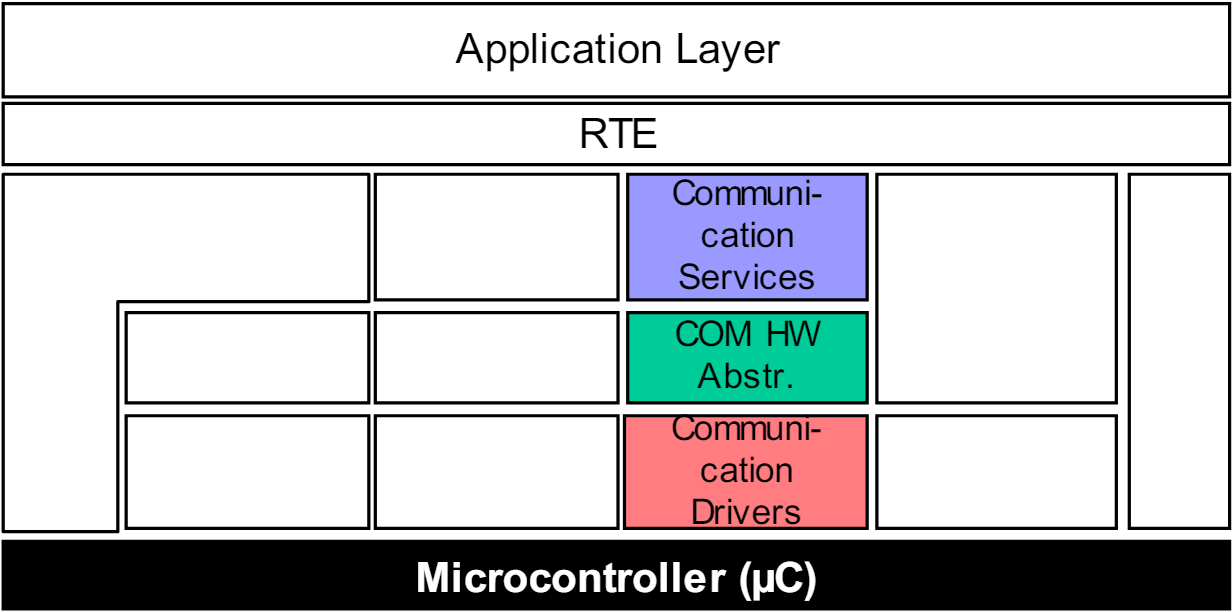


### Communication Stack – FlexRay



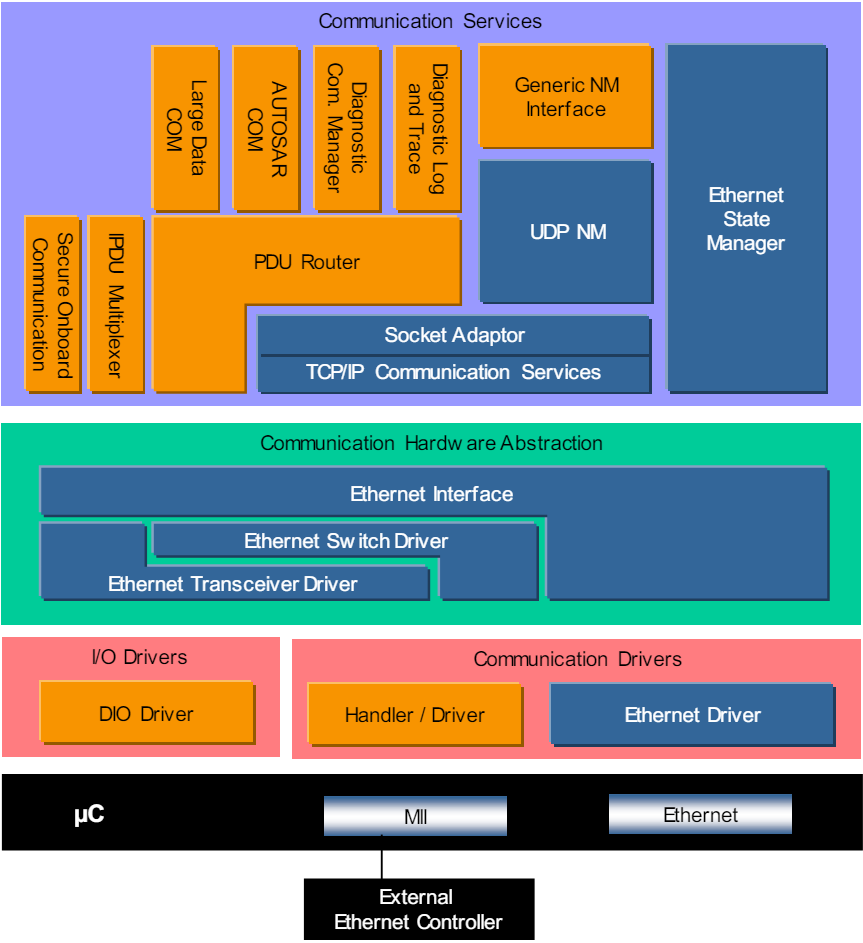


### Communication Stack – FlexRay

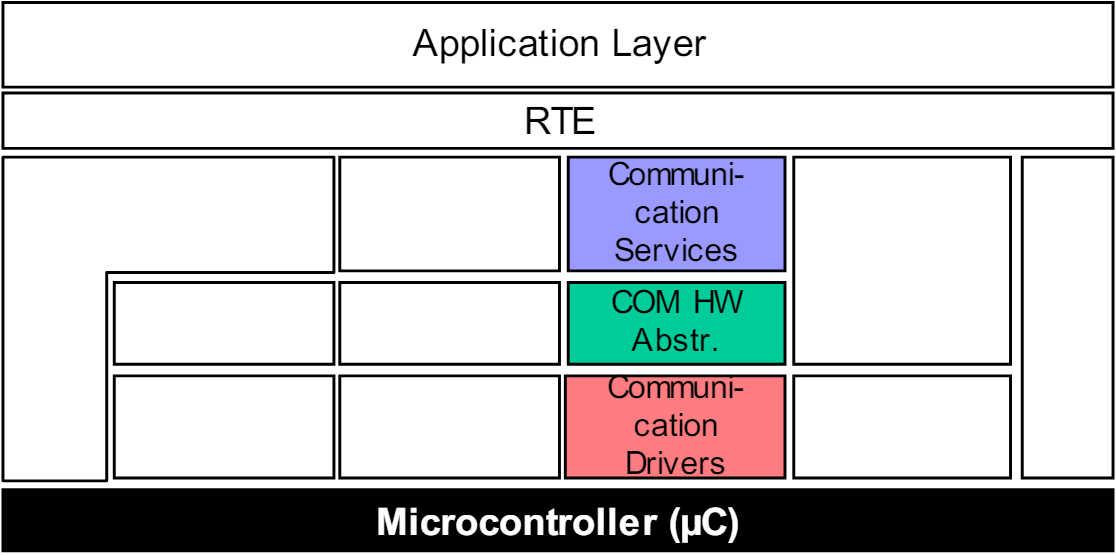


### Communication Stack – TCP/IP

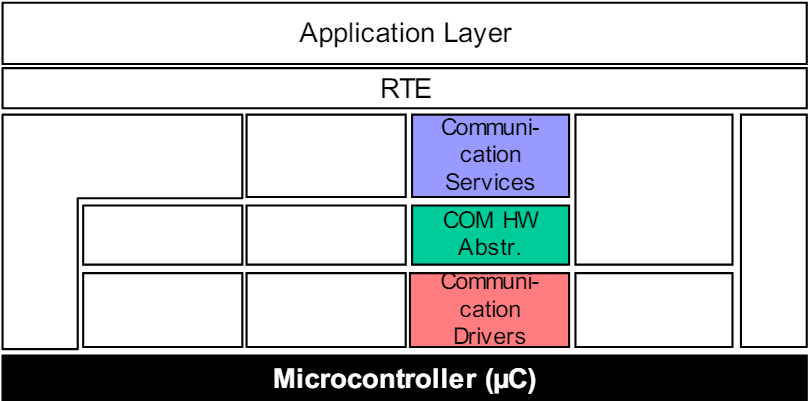


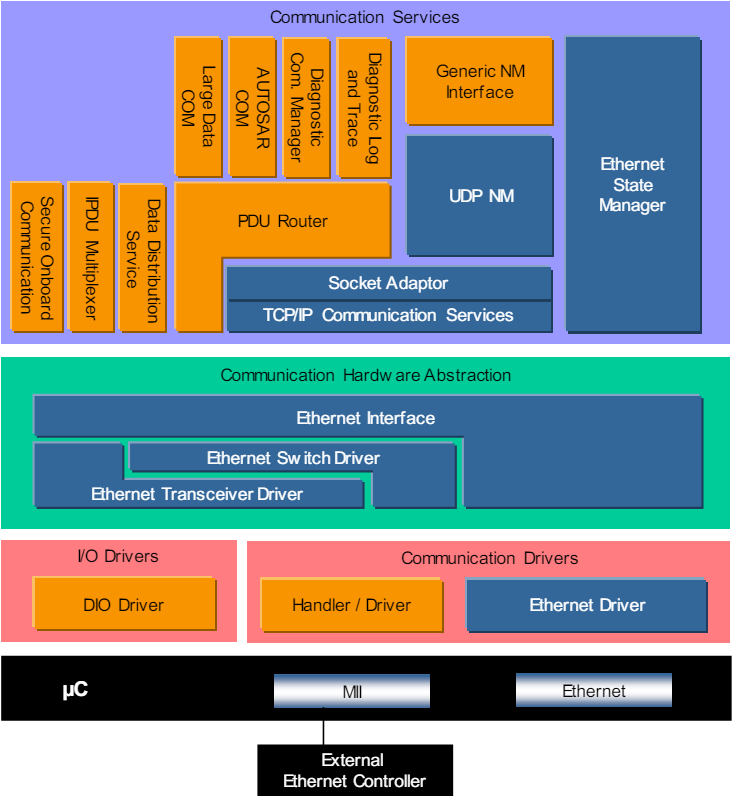


### Communication Stack – TCP/IP

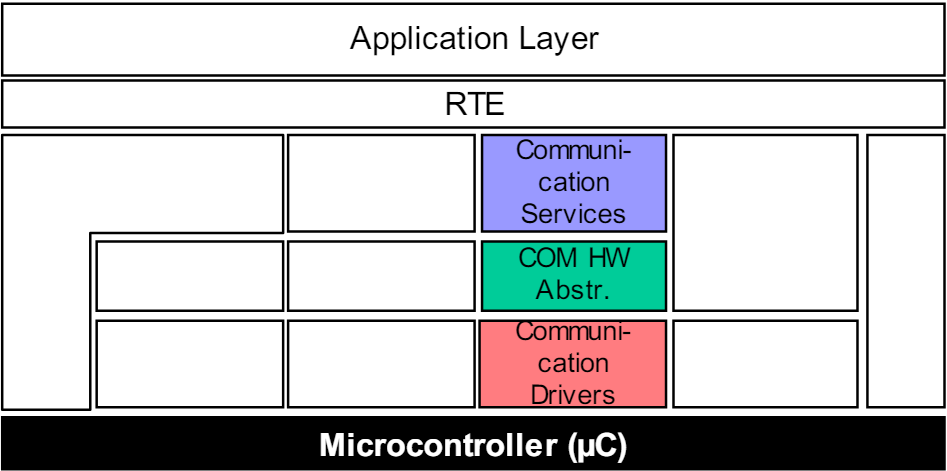


### Communication Stack – DDS

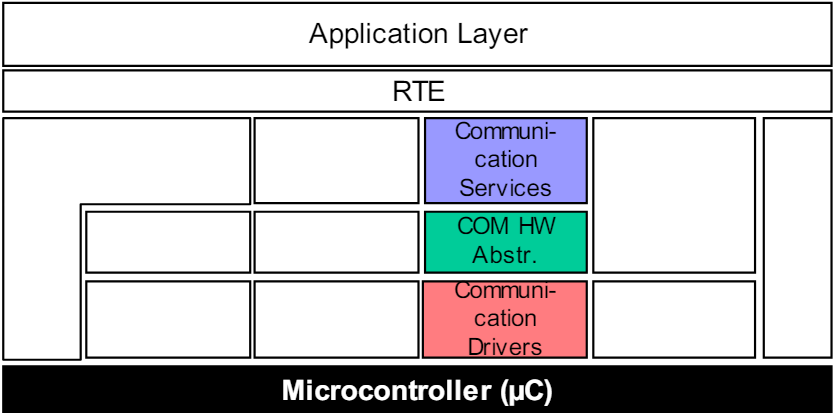




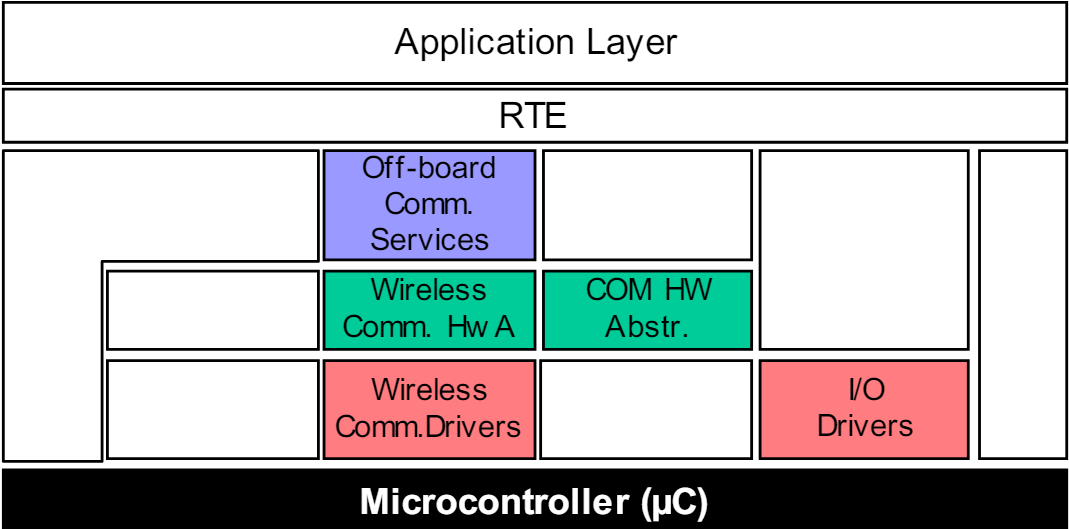
### Communication Stack – DDS

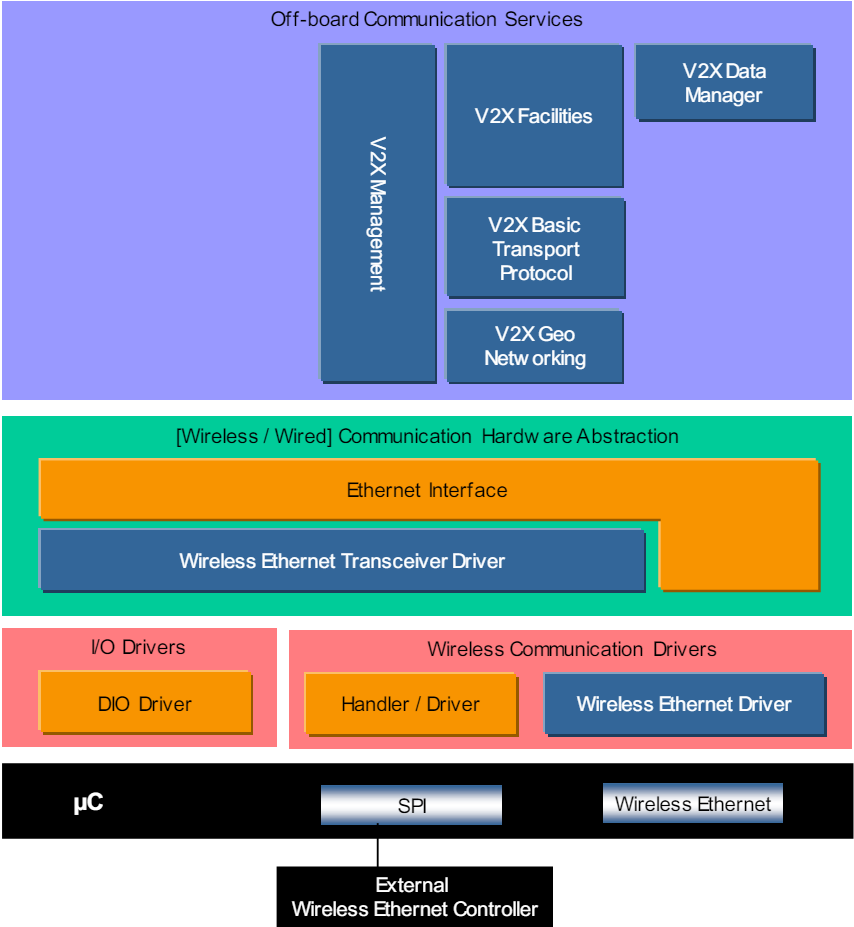


### Communication Stack – General

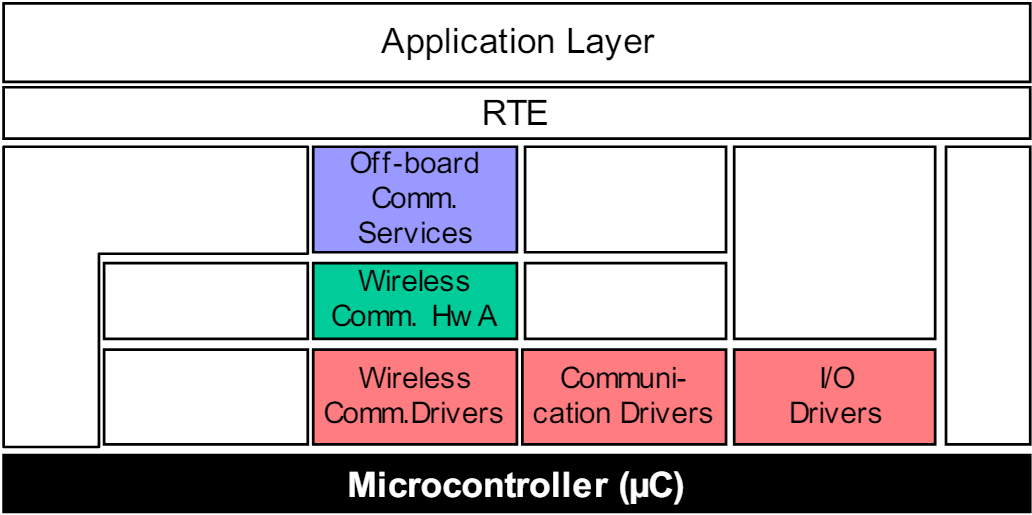


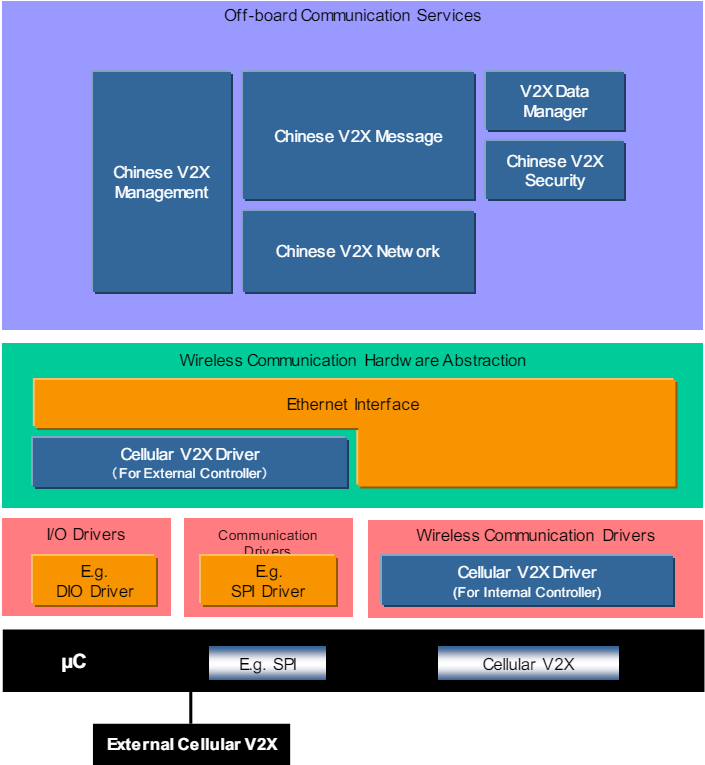
### Off-board Communication Stack – European Vehicle-2-X



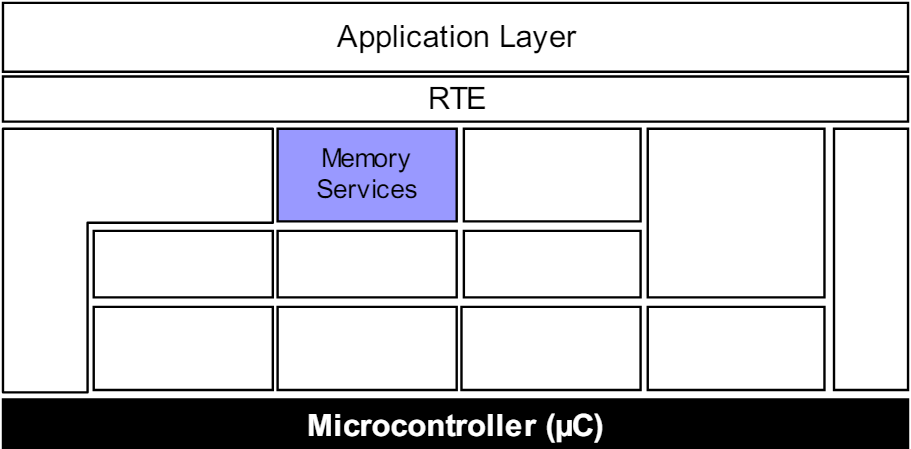


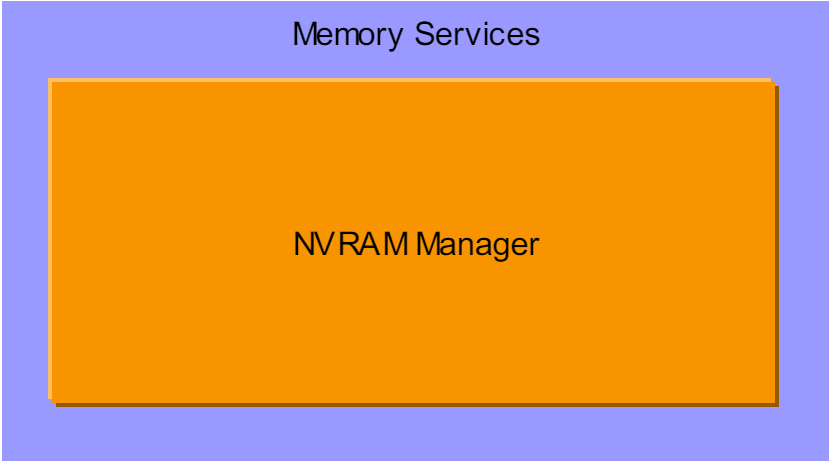
### Off-board Communication Stack – Chinese Vehicle-2-X



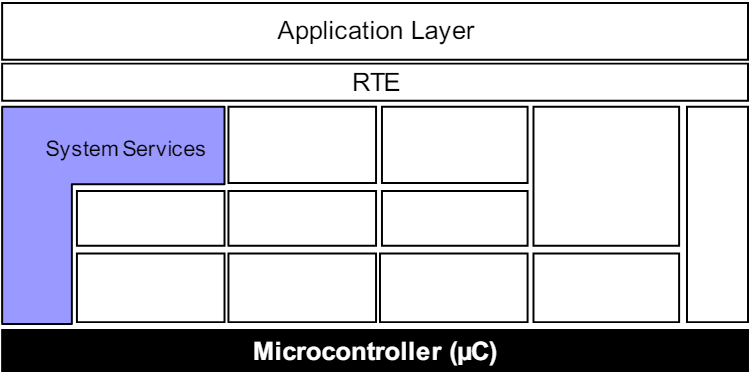


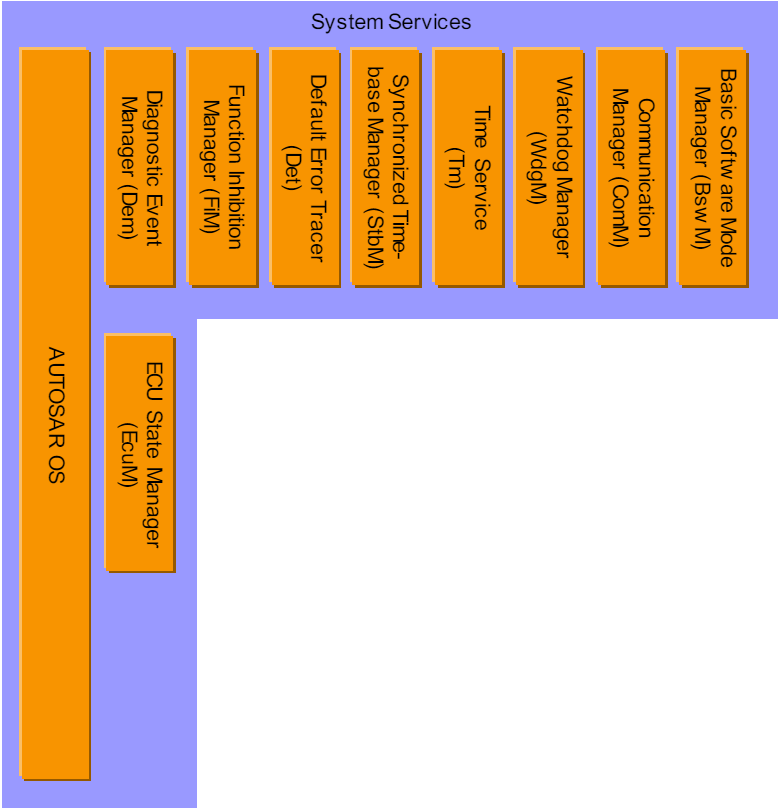
### Services: Memory Services



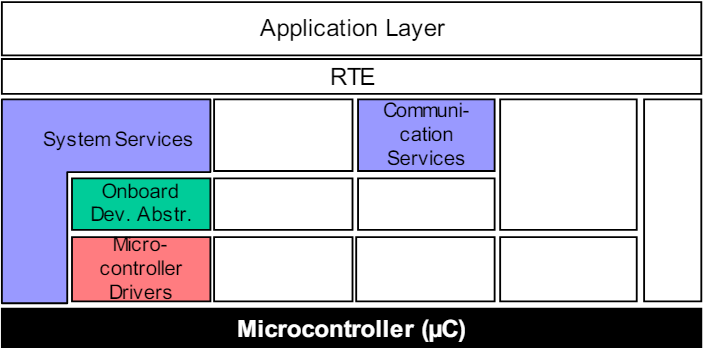


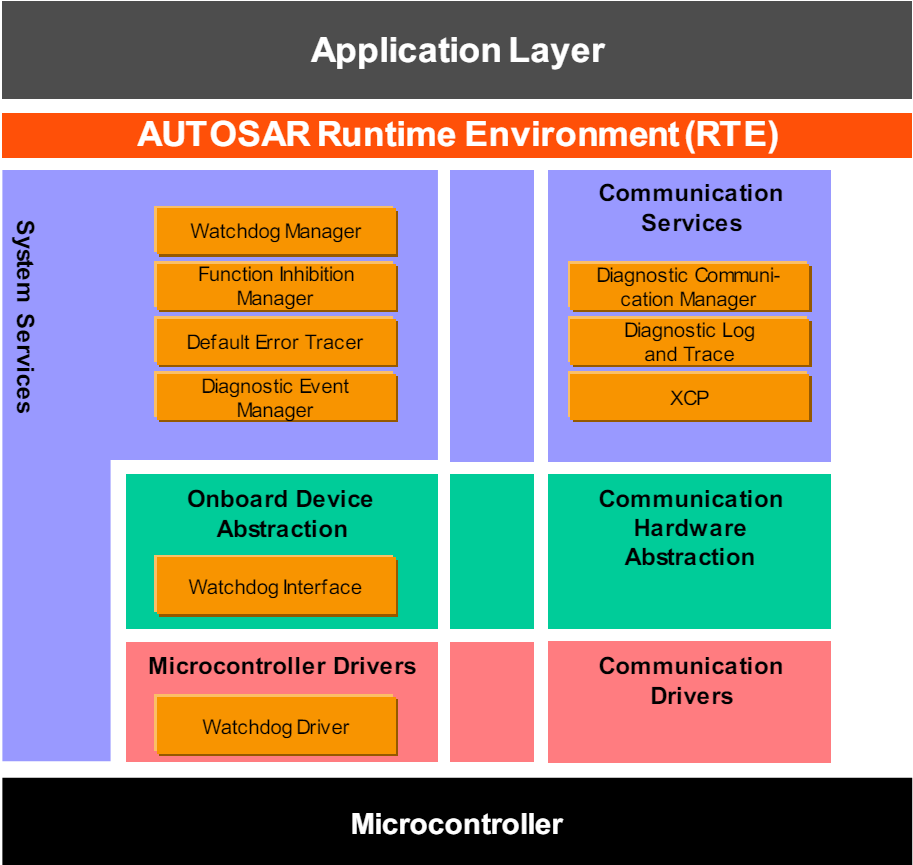
### Services: System Services



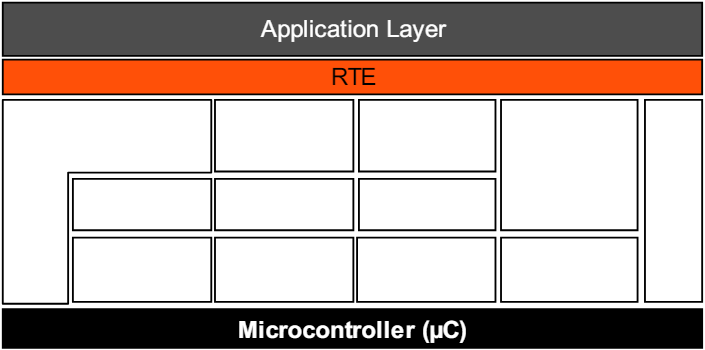


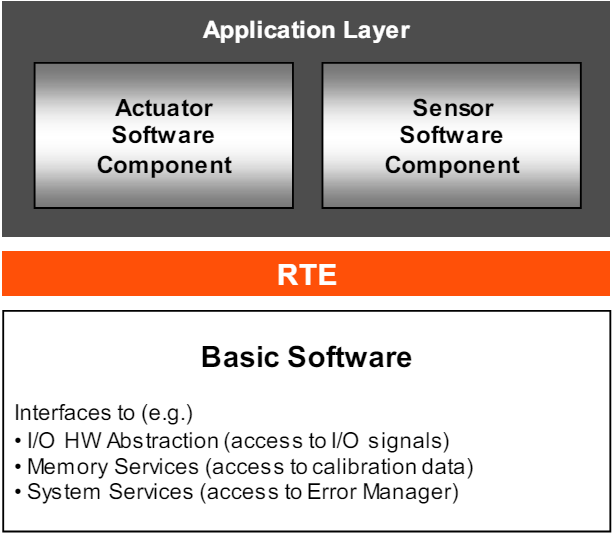
### Error Handling, Reporting and Diagnostic





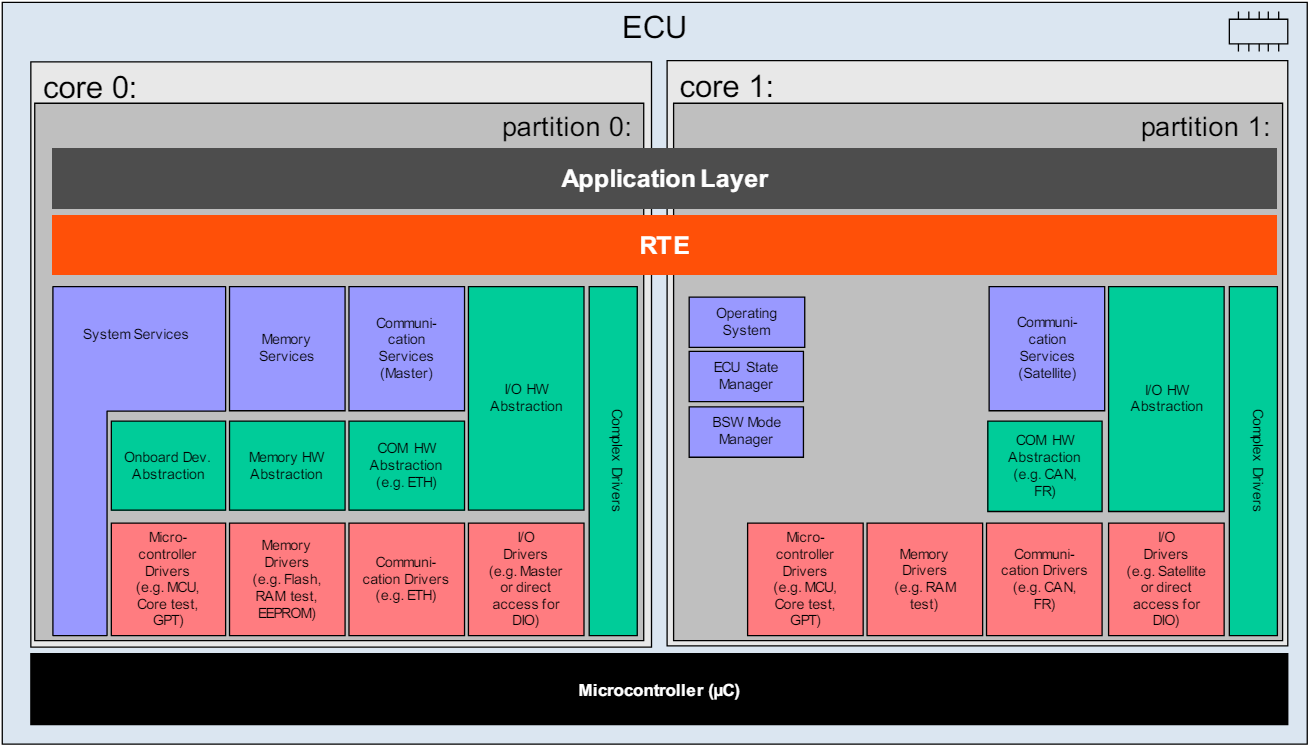
### Application Layer: Sensor/Actuator Software Components



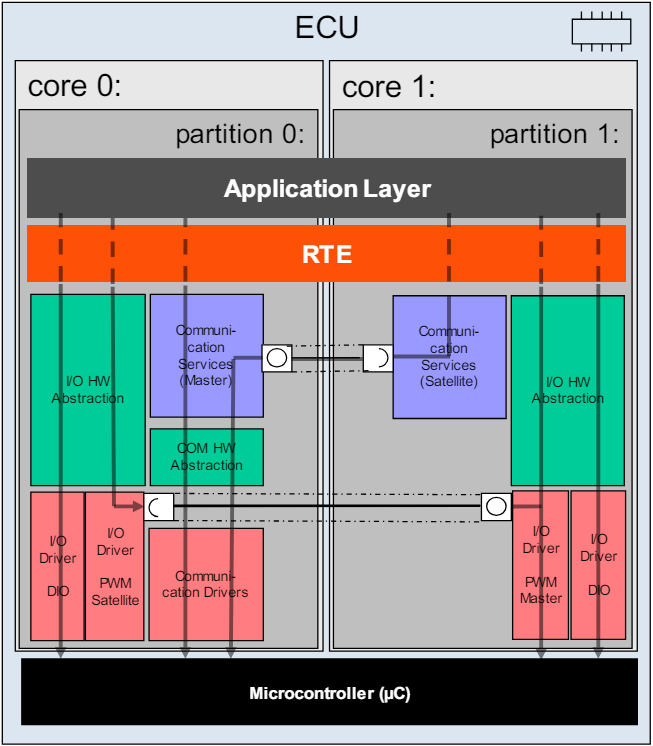


## Content of Software Layers in Multi-Core Systems

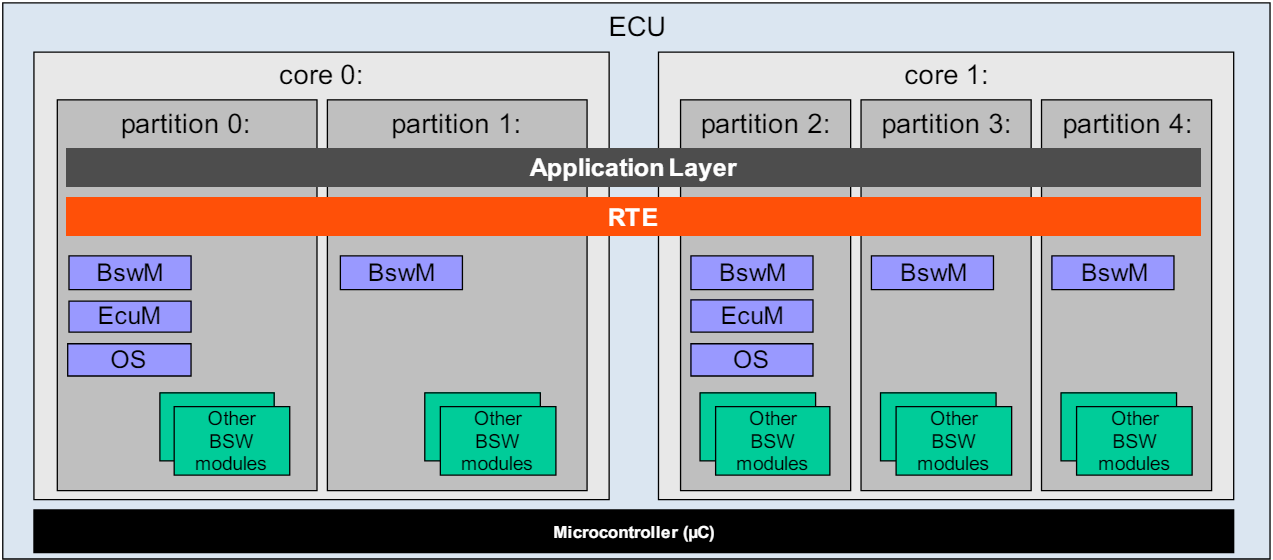
### Example of a Layered Software Architecture for Multi-Core Microcontroller



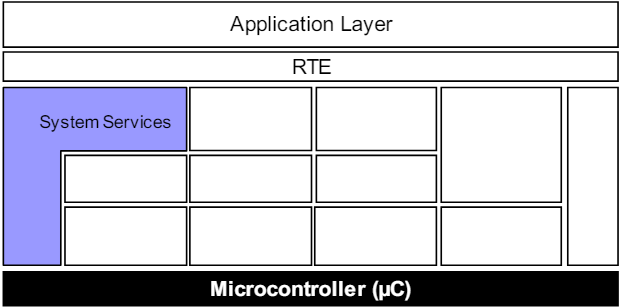
### Detailed View of Distributed BSW Modules

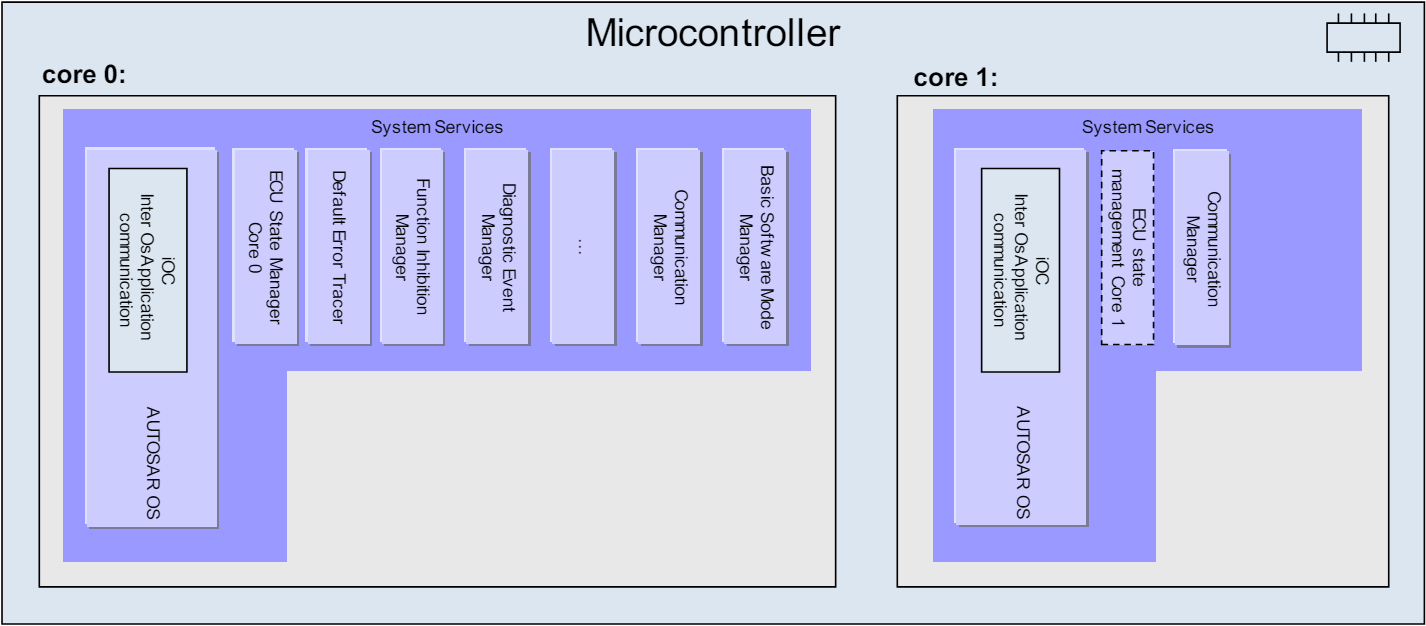


### Overview of BSW Modules, OS, BswM and EcuM on Multiple Partitions



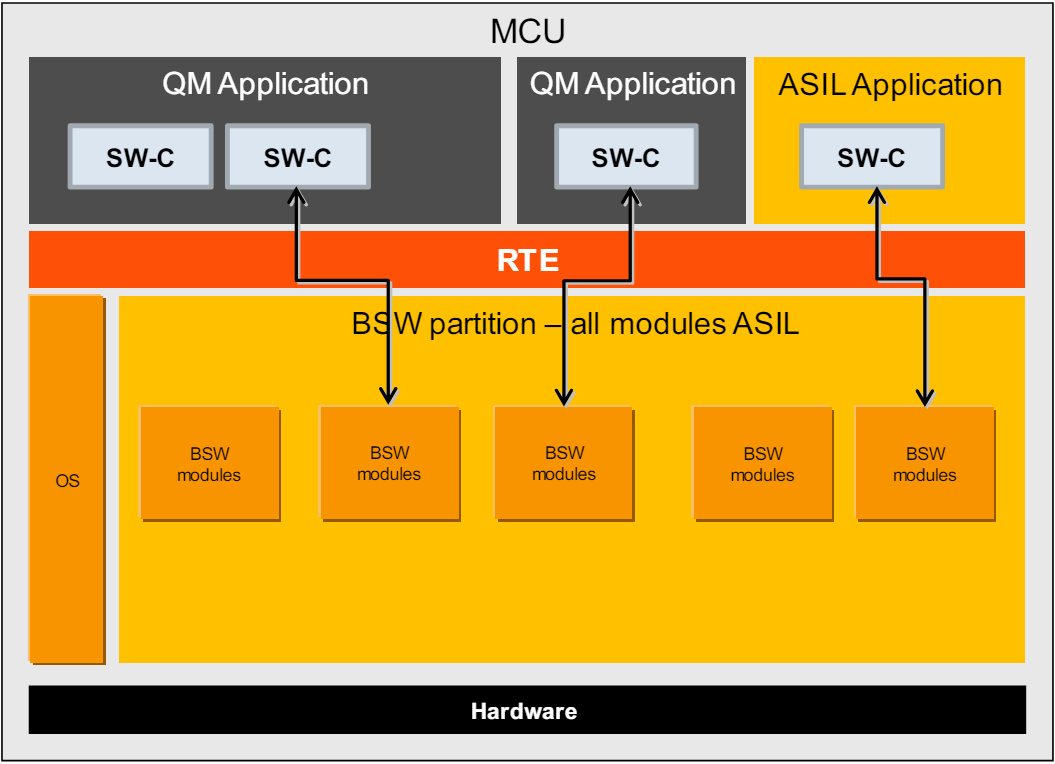
### Scope: Multi-Core System Services



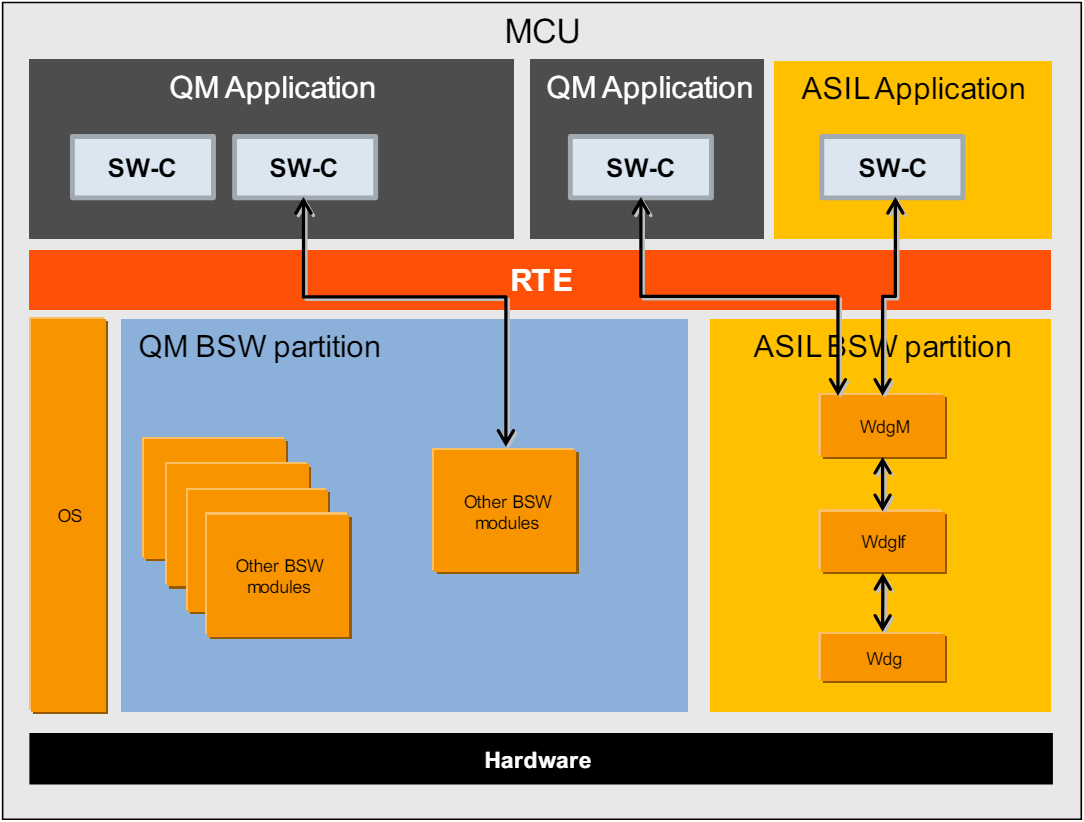


## Content of Software Layers in Mixed-Critical Systems

### Overview of AUTOSAR safety handling

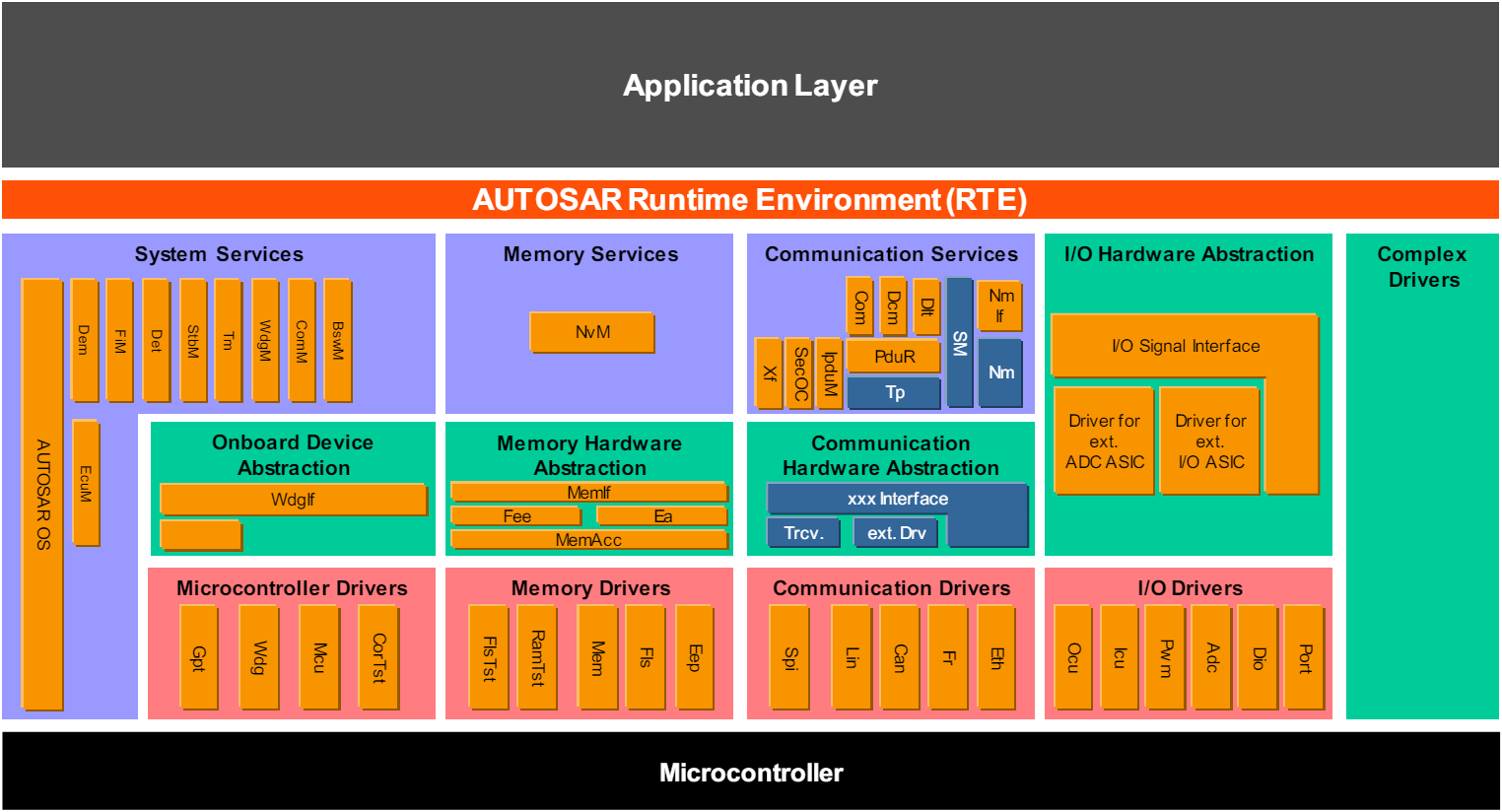


### AUTOSAR BSW distribution for safety systems



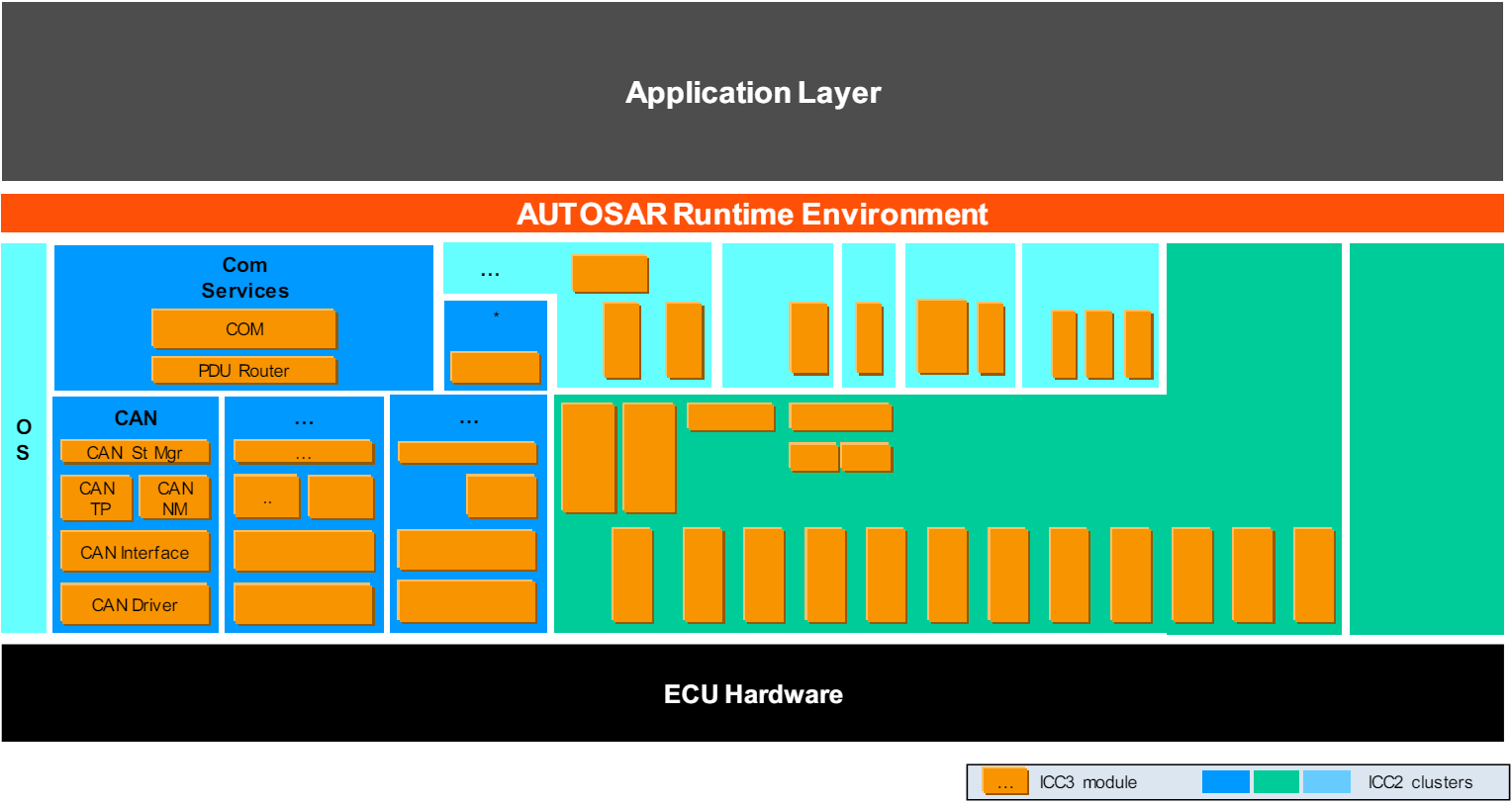
## Overview of Modules

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

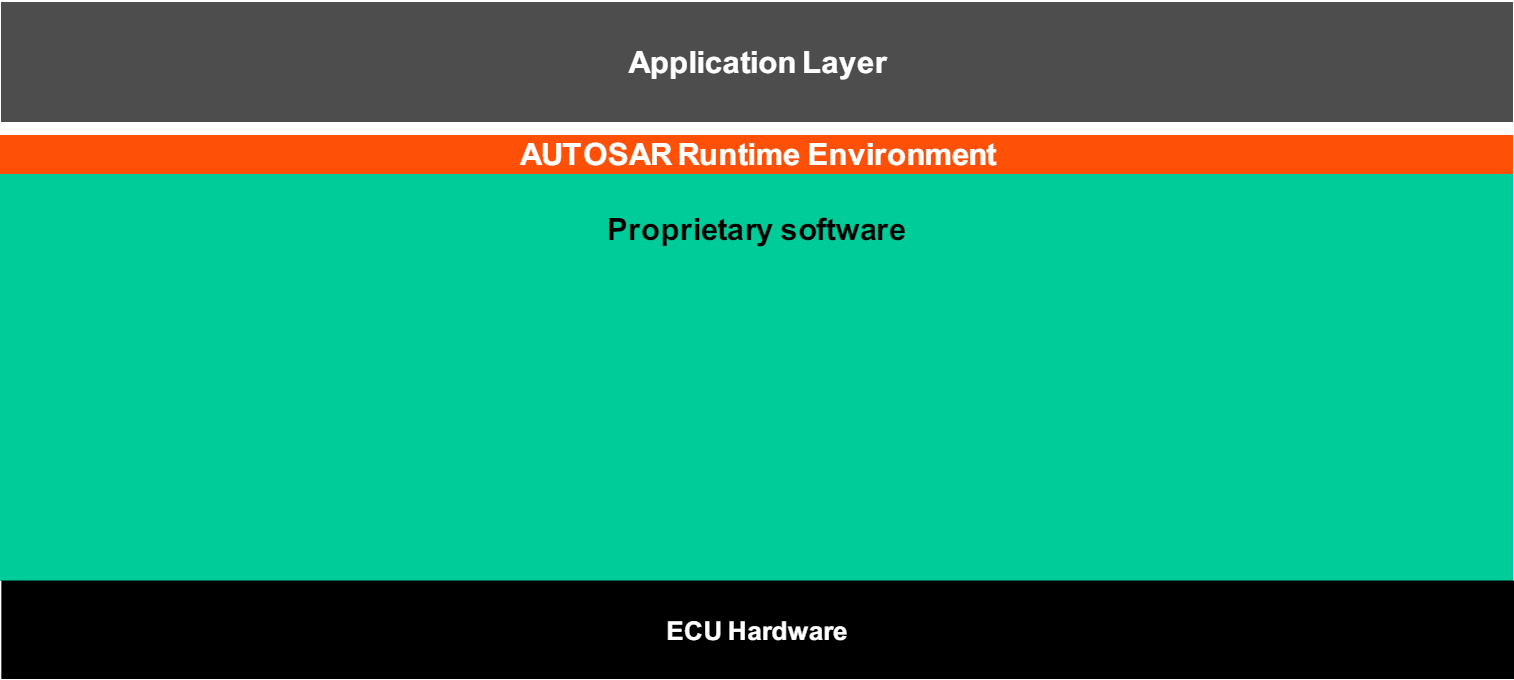




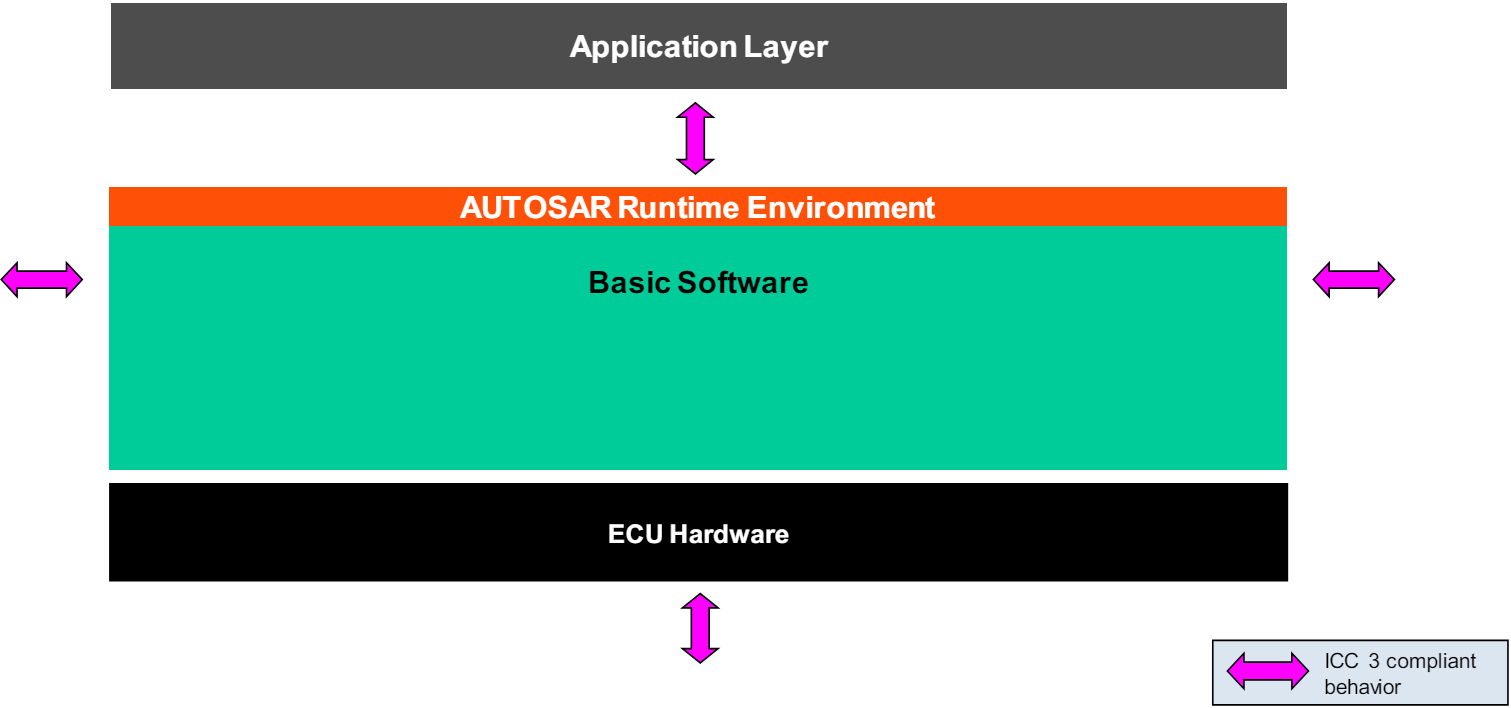
### Overview of Modules – Implementation Conformance Classes – ICC2



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



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

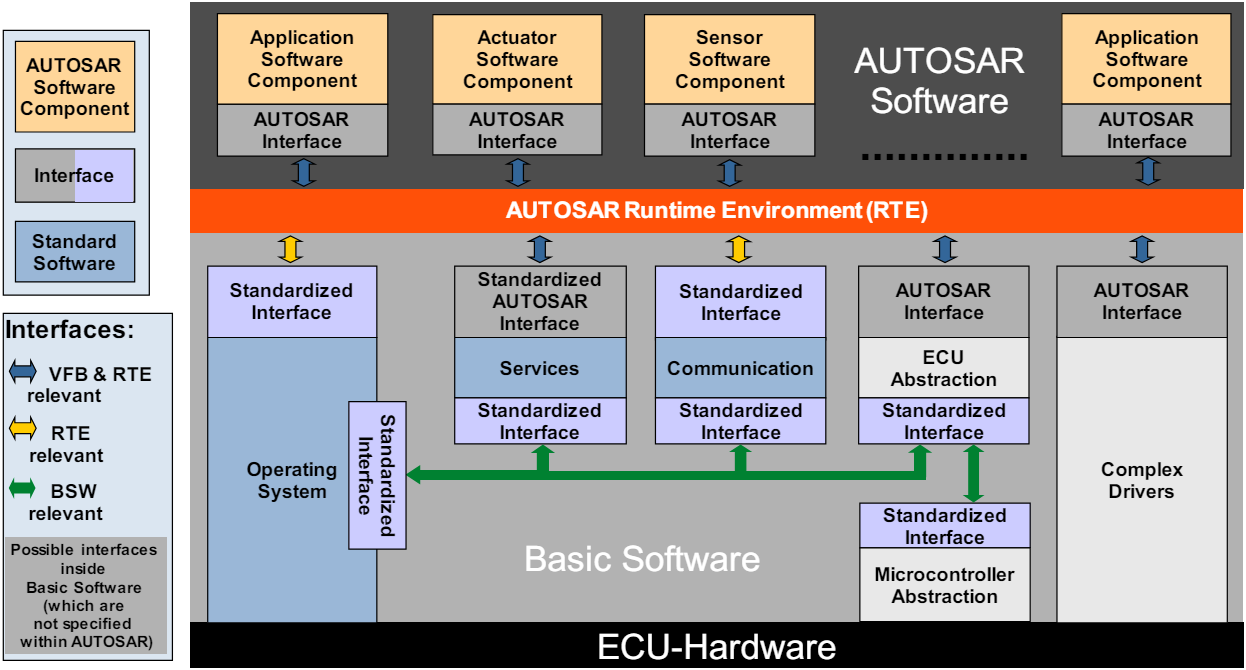


## Interfaces: General Rules

### Interfaces Type of Interfaces in AUTOSAR

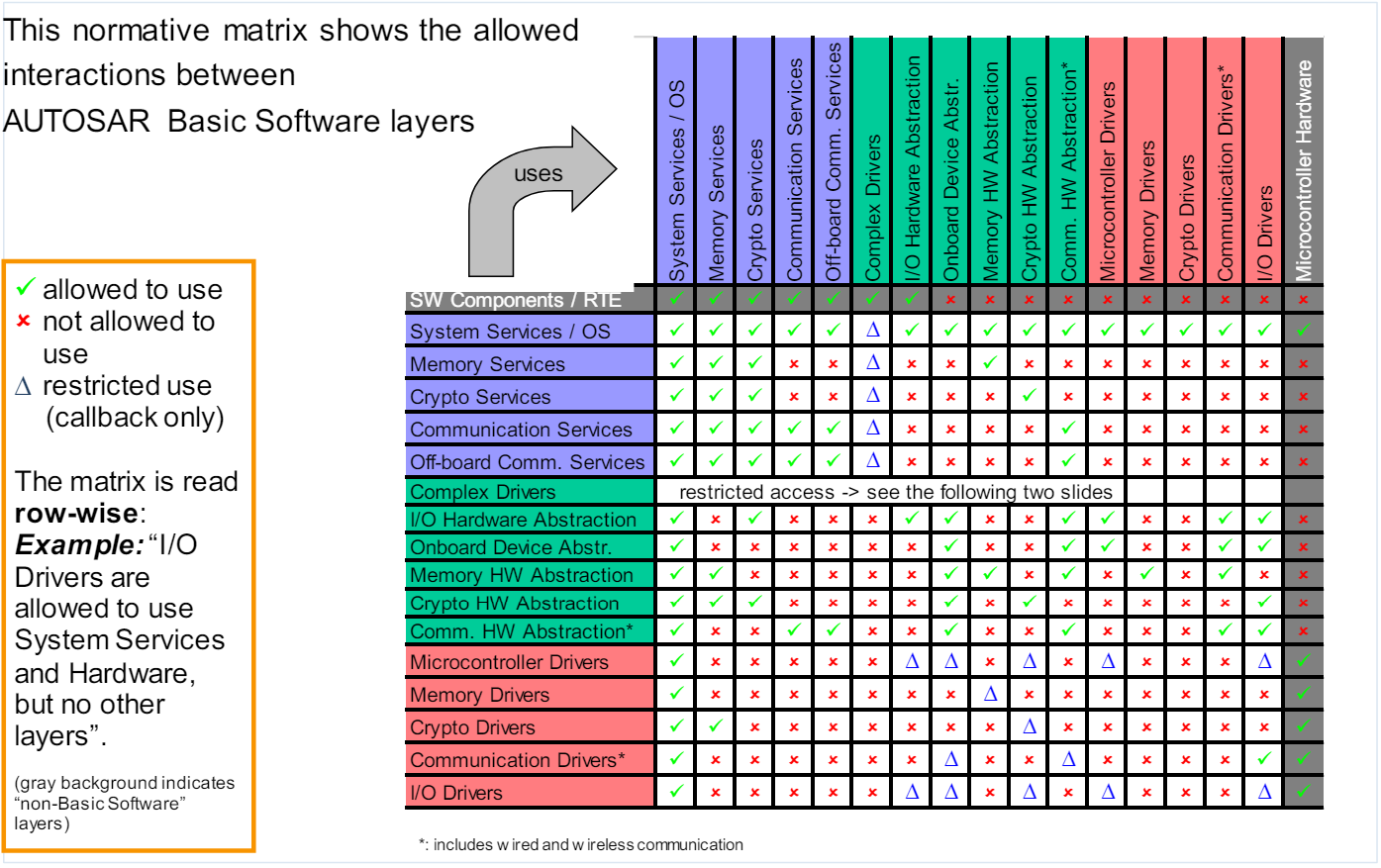
|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

### Components and interfaces view (simplified)

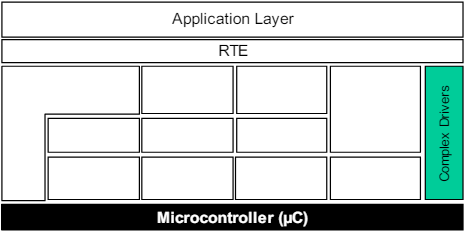


### General Interfacing Rules

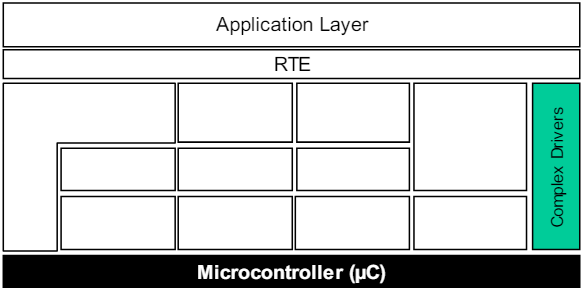
### Layer Interaction Matrix



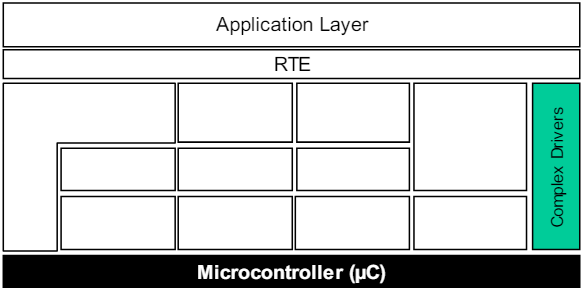
### Interfacing with Complex Drivers (1)



### Interfacing with Complex Drivers (2)



### Interfacing with Complex Drivers (3)



## Interfaces: Interaction of Layers

### Introduction