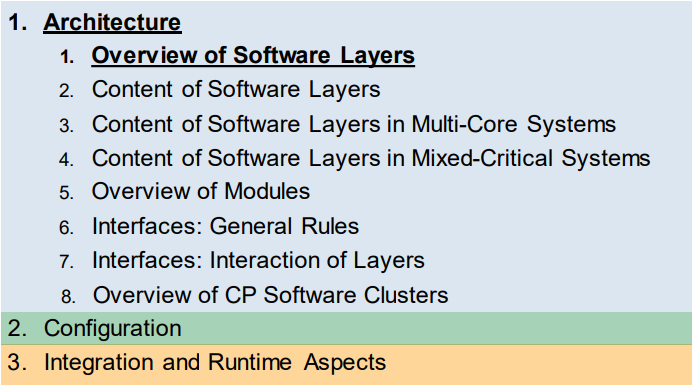
**Layered Software Architecture AUTOSAR**

***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:

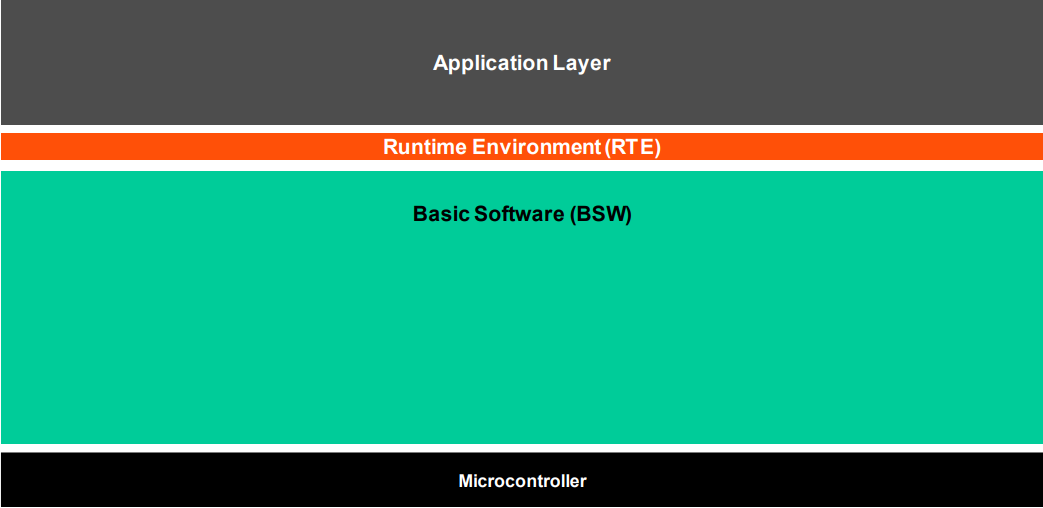
1. **Các mô-đun tiêu chuẩn có thể được mở rộng về chức năng, trong khi vẫn tuân thủ**:
   * Kiến trúc AUTOSAR cho phép mở rộng chức năng của các **mô-đun phần mềm tiêu chuẩn** mà không làm mất đi tính tuân thủ các chuẩn mực đã đặt ra. Tuy nhiên, trong quá trình mở rộng, **cấu hình** của các mô-đun này 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:

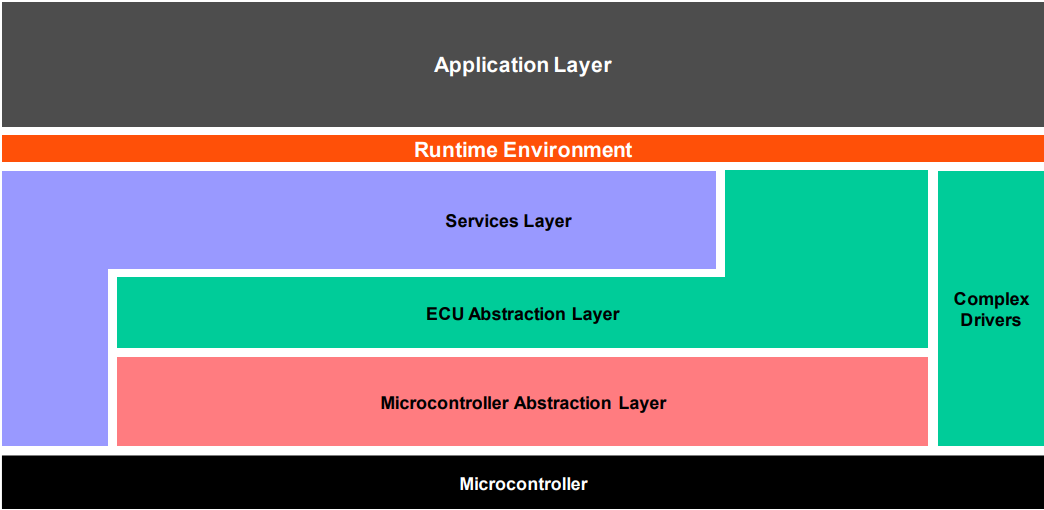
1. **Application Layer (Lớp Ứng dụng)**:
   * **Lớp ứng dụng** chứa các ứng dụng và phần mềm điều khiển đặc thù của từng hệ thống hoặc chức năng trong ô tô. Các ứng dụng này là những chương trình chạy trên ECU (Electronic Control Unit), thực hiện các chức năng như điều khiển động cơ, hệ thống phanh, 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.

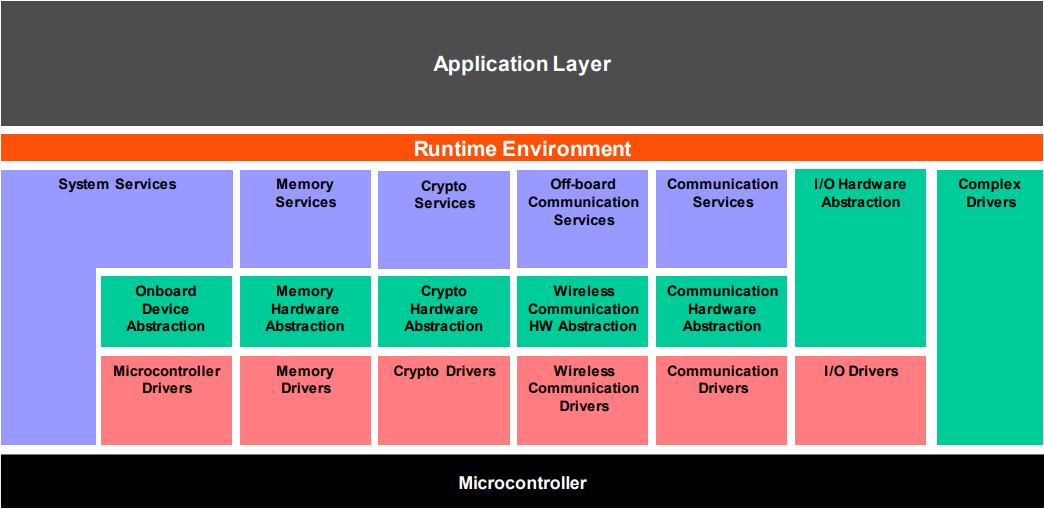
**4. Complex Drivers (Trình điều khiển Phức tạp):**

* **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.

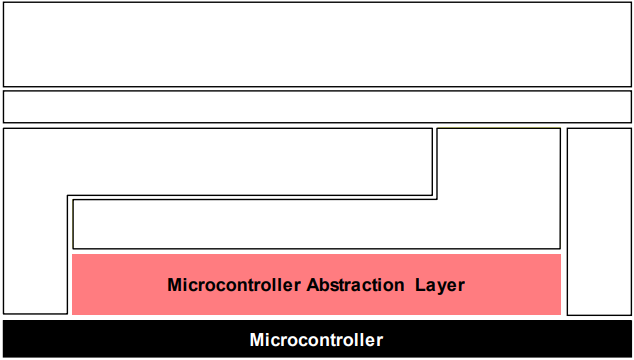


***Các nhóm chức năng trong Basic Software:***

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.



**Hình 1:** Microcontroller Abstraction Layer

#### Nhiệm vụ của MCAL

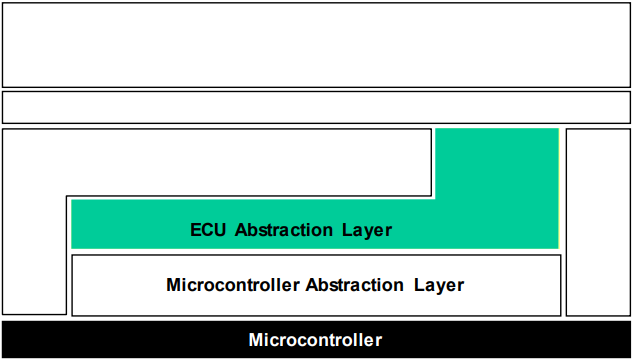
* **Tạo các lớp phần mềm cao hơn độc lập với Bộ điều khiển vi xử lý (µC)**:
  + MCAL cung cấp một lớp trừu tượng giúp các mô-đun phần mềm không phải lo lắng về các chi tiết của phần cứng cụ thể của bộ điều khiển vi xử lý (microcontroller). Điều này có nghĩa là phần mềm ở các lớp cao hơn có thể tập trung 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.

### 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).



**Hình 2:** ECU Abstraction Layer

#### Nhiệm vụ của ECU Abstraction Layer

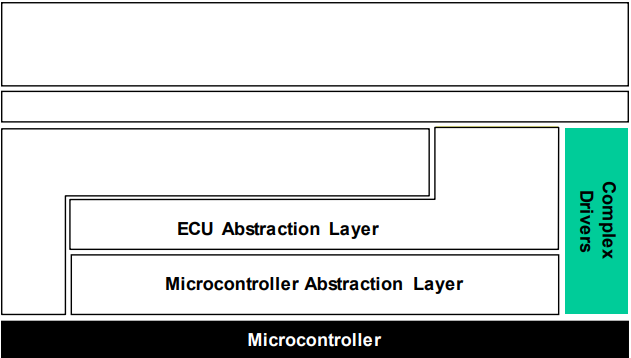
* **Tạo các lớp phần mềm cao hơn độc lập với bố trí phần cứng của Đơn vị Điều khiển Điện tử (ECU)**:
  + ECUAL cung cấp **một lớp trừu tượng** giúp phần mềm ở các lớp cao hơn không cần phải lo lắng về cách thức bố trí phần cứng của ECU (bao gồm các bộ vi xử lý và các thiết bị ngoại vi). Điều này giúp 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.



**Hình 3:** Complex Drivers

#### Nhiệm vụ của Complex Drivers Layer

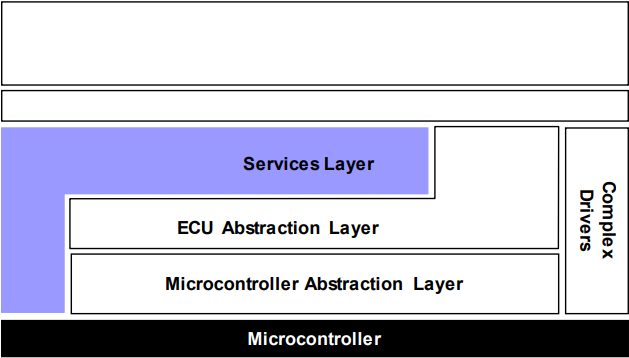
* **Cung cấp khả năng tích hợp chức năng đặc biệt**, ví dụ như:
  + **Trình điều khiển cho các thiết bị không được chỉ định trong AUTOSAR**: Lớp này hỗ trợ tích hợp các thiết bị hoặc phần cứng mà **AUTOSAR không định nghĩa**. Điều này đặc biệt hữu ích khi có phần cứng hoặc thiết bị đặc biệt mà không 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.



**Hình 4:** Services Layer

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

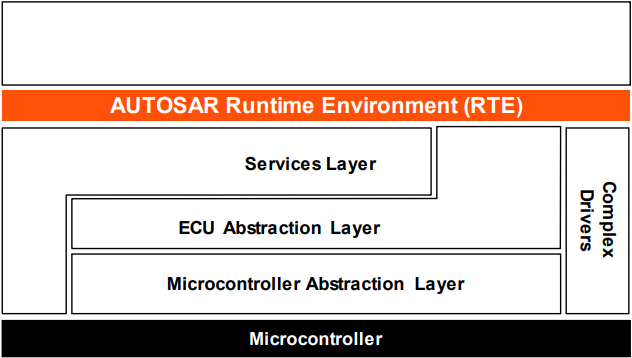
1. **Chức năng của hệ điều hành**:
   * Dịch vụ của hệ điều hành được cung cấp trong lớp này để hỗ trợ các ứng dụng và hệ thống trong việc quản lý tác vụ, luồng chương trình, và tài nguyên hệ thống.
2. **Dịch vụ giao tiếp và quản lý mạng xe**:
   * Services Layer hỗ trợ các giao thức mạng như **CAN**, **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).



**Hình 5:** **RTE**

#### 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****

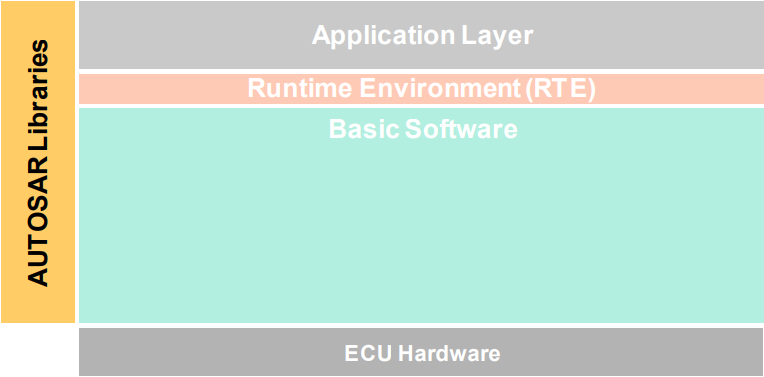
* Các **manager** thường được tích hợp vào **Services Layer** trong kiến trúc AUTOSAR. Services Layer cung cấp các dịch vụ cơ bản cho các phần mềm ứng dụng và các mô-đun cơ sở, và đây là nơi các **manager** như **NVRAM Manager**, **Memory Manager**, **Communication Manager**, v.v., được triển khai để 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.

### ****Libraries trong Kiến trúc AUTOSAR****

Trong **kiến trúc AUTOSAR**, **Libraries** (Thư viện) đóng vai trò quan trọng trong việc cung cấp các hàm chức năng tái sử dụng cho các mô-đun phần mềm khác. Các thư viện này giúp tăng cường tính hiệu quả trong phát triển phần mềm, tối ưu hóa quá trình bảo trì, và tạo ra các giải pháp phần mềm có thể tái sử dụng.



#### ****Đặc điểm của Libraries trong AUTOSAR****

1. **Có thể được gọi bởi các mô-đun BSW (bao gồm RTE), SW-Cs, thư viện hoặc mã tích hợp:**
   * Thư viện có thể được gọi bởi nhiều loại mô-đun phần mềm khác nhau, bao gồm cả các mô-đun **Basic Software (BSW)** như **RTE (Runtime Environment)**, các **Software Components (SW-Cs)**, các thư viện khác hoặc mã phần mềm tích hợp vào hệ thống. Điều này cho phép các hàm trong thư viện có thể tái sử dụng cho nhiều mục đích khác nhau trong toàn bộ hệ thống.
2. **Chạy trong ngữ cảnh của người gọi trong cùng một môi trường bảo vệ:**
   * Các thư viện chạy trong cùng ngữ cảnh bảo vệ với mô-đun gọi chúng, điều này có nghĩa là thư viện không thực thi trong không gian bảo vệ riêng biệt. Điều này giúp giảm độ trễ và tăng hiệu suất vì không cần phải chuyển đổi ngữ cảnh khi gọi các hàm trong thư viện.
3. **Chỉ có thể gọi thư viện:**
   * Thư viện được thiết kế để **chỉ được gọi**, không có khả năng gọi lại hoặc thay đổi ngữ cảnh khác. Điều này giúp đơn giản hóa việc sử dụng và đảm bảo tính nhất quán trong việc truy cập các hàm chức năng mà không gặp phải sự xung đột.
4. **Có thể được gọi lại:**
   * Các thư viện có thể được gọi lại trong bất kỳ lúc nào từ bất kỳ mô-đun nào, điều này đảm bảo tính **tái sử dụng** cao và khả năng tích hợp lại vào các mô-đun phần mềm mà không gặp phải xung đột hoặc sự phụ thuộc vào ngữ cảnh gọi.
5. **Không có trạng thái nội bộ:**
   * Thư viện không duy trì trạng thái của riêng mình giữa các lần gọi hàm. Mỗi lần thư viện được gọi, hàm thực thi độc lập mà không cần phải lưu trữ bất kỳ dữ liệu hay trạng thái nào từ các lần gọi trước đó. Điều này giúp các thư viện dễ dàng **tái sử dụng** mà không cần lo lắng về việc quản lý trạng thái.
6. **Không yêu cầu khởi tạo:**
   * Các thư viện **không yêu cầu bất kỳ bước khởi tạo nào** trước khi sử dụng. Điều này giúp đơn giản hóa việc tích hợp thư viện vào hệ thống, vì các mô-đun không cần phải lo lắng về việc khởi tạo thư viện trước khi sử dụng các hàm của nó.
7. **Được đồng bộ (Synchronous):**
   * Các hàm trong thư viện được **đồng bộ**, có nghĩa là chúng thực thi ngay lập tức và không có **độ trễ**. Điều này đảm bảo rằng các mô-đun sử dụng thư viện có thể nhận được kết quả của phép toán ngay lập tức mà không cần phải chờ đợi.

#### ****Ví dụ về Libraries trong AUTOSAR****

1. **Math Libraries (Thư viện toán học):**
   * Các thư viện toán học có thể bao gồm các hàm tính toán cho các phép toán cơ bản như cộng, trừ, nhân, chia và các phép toán phức tạp hơn như tính toán lượng giác hoặc các phép tính phân tích số học.
2. **Crypto Libraries (Thư viện mã hóa):**
   * Thư viện mã hóa có thể cung cấp các hàm mã hóa và giải mã dữ liệu sử dụng các thuật toán bảo mật như **AES**, **RSA**, v.v. Các thư viện này có thể được sử dụng bởi các mô-đun phần mềm để bảo vệ dữ liệu trong quá trình giao tiếp.
3. **Communication Libraries (Thư viện giao tiếp):**
   * Các thư viện giao tiếp có thể bao gồm các hàm giúp xử lý các giao thức giao tiếp như **CAN**, **Ethernet**, **LIN** và các giao thức khác. Các mô-đun trong hệ thống có thể gọi các hàm trong thư viện này để thực hiện giao tiếp với các hệ thống bên ngoài.
4. **Sensor/Actuator Libraries (Thư viện cảm biến/bộ truyền động):**
   * Các thư viện này có thể chứa các hàm giúp giao tiếp với cảm biến hoặc bộ truyền động. Ví dụ, thư viện có thể cung cấp các hàm cho việc đọc dữ liệu từ cảm biến, xử lý dữ liệu, hoặc điều khiển bộ truyền động.

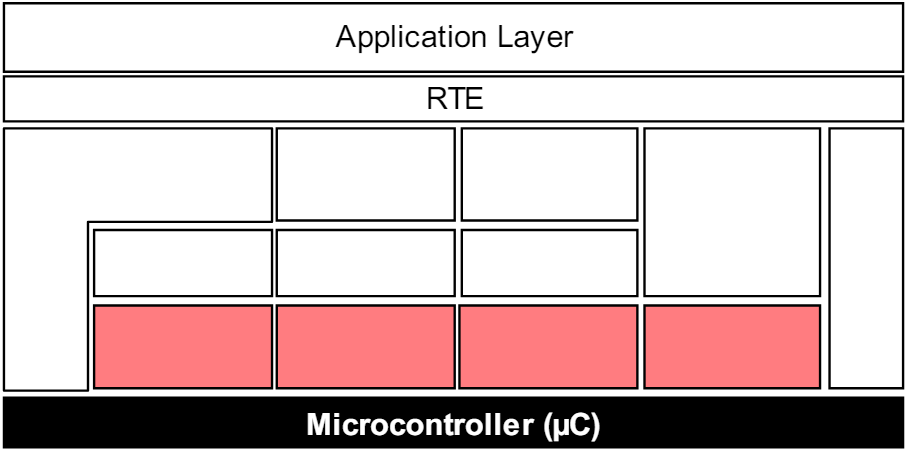
#### ****Tóm lại****

* **Libraries** trong **kiến trúc AUTOSAR** là các bộ sưu tập hàm chức năng tái sử dụng giúp giảm sự phụ thuộc giữa các mô-đun phần mềm khác nhau, đồng thời tối ưu hóa và tăng cường khả năng tái sử dụng mã nguồn.
* Các thư viện này có những đặc điểm như không có trạng thái nội bộ, không yêu cầu khởi tạo, và đồng bộ, đảm bảo tính hiệu quả cao trong việc thực thi mà không gặp phải độ trễ.
* **Libraries** có thể được sử dụng trong nhiều mô-đun trong hệ thống AUTOSAR, như các mô-đun BSW, RTE, các thành phần phần mềm, hoặc mã tích hợp. Các ví dụ điển hình bao gồm thư viện toán học, thư viện mã hóa, thư viện giao tiếp, và thư viện cảm biến/bộ truyền động.

|  |
| --- |
| * **Fixed Point Mathematical Library (Thư viện Toán học với Điểm Cố định)**:   + Cung cấp các hàm toán học sử dụng **số học điểm cố định (fixed point arithmetic)**. Đây là loại số học thường được sử dụng trong các hệ thống nhúng để tiết kiệm tài nguyên tính toán và bộ nhớ.   + Ứng dụng: Các phép tính trong thời gian thực trên ECU với tài nguyên hạn chế. * **Floating Point Mathematical Library (Thư viện Toán học với Điểm Dấu Phẩy Động)**:   + Cung cấp các hàm toán học sử dụng **số học dấu phẩy động (floating point arithmetic)**. Thư viện này phù hợp cho các phép tính phức tạp hoặc các ứng dụng yêu cầu độ chính xác cao hơn.   + Ứng dụng: Xử lý dữ liệu từ cảm biến hoặc thực hiện các phép tính phức tạp trong các hệ thống có vi xử lý hỗ trợ số học dấu phẩy động. * **Interpolation for Fixed Point Data (Nội suy với Dữ liệu Điểm Cố định)**:   + Hỗ trợ các phép toán **nội suy (interpolation)** trên dữ liệu số sử dụng điểm cố định. Nội suy là kỹ thuật tính toán giá trị trung gian dựa trên các dữ liệu đã biết.   + Ứng dụng: Hiệu chỉnh dữ liệu cảm biến hoặc hiệu suất động cơ trong các hệ thống yêu cầu tối ưu hóa sử dụng bộ nhớ. * **Interpolation for Floating Point Data (Nội suy với Dữ liệu Điểm Dấu Phẩy Động)**:   + Hỗ trợ các phép toán nội suy tương tự, nhưng dành cho dữ liệu sử dụng dấu phẩy động. Điều này đặc biệt hữu ích khi làm việc với dữ liệu yêu cầu độ chính xác cao.   + Ứng dụng: Các hệ thống điều khiển phức tạp hoặc dữ liệu đo từ cảm biến. * **Extended Functions Library (Thư viện Hàm Mở Rộng)**:   + Cung cấp các chức năng mở rộng, bao gồm:     - **Tính toán 64 bit**: Để xử lý các dữ liệu lớn hoặc cần độ chính xác cao hơn.     - **Lọc dữ liệu (filtering)**: Lọc các tín hiệu cảm biến để loại bỏ nhiễu hoặc xử lý tín hiệu.   + Ứng dụng: Các thuật toán phức tạp, ví dụ như lọc tín hiệu hoặc xử lý dữ liệu cảm biến tiên tiến. * **Bit Handling Library (Thư viện Xử lý Bit)**:   + Cung cấp các hàm xử lý bit, chẳng hạn như:     - Đọc và ghi từng bit.     - Thao tác dữ liệu ở cấp độ bit.   + Ứng dụng: Giao tiếp với phần cứng, tối ưu hóa việc sử dụng bộ nhớ và quản lý trạng thái hệ thống. * **E2E Communication Library (Thư viện Giao tiếp E2E - End-to-End)**:   + Hỗ trợ giao tiếp **end-to-end (E2E)**, đảm bảo dữ liệu được truyền đi và nhận lại một cách an toàn và chính xác.   + Ứng dụng: Đảm bảo tính toàn vẹn và bảo mật của dữ liệu trong các giao tiếp giữa các ECU hoặc các mạng xe. * **CRC Calculation Library (Thư viện Tính Toán CRC)**:   + Cung cấp các hàm để tính **mã kiểm tra tuần hoàn (CRC - Cyclic Redundancy Check)**, giúp phát hiện lỗi trong dữ liệu truyền hoặc lưu trữ.   + Ứng dụng: Kiểm tra tính toàn vẹn dữ liệu trong các giao tiếp CAN, LIN, Ethernet, hoặc trong bộ nhớ. * **Atomic Multicore Safe Operations Library (Thư viện Các Phép Toán An Toàn Đa Nhân)**:   + Hỗ trợ các thao tác **nguyên tử (atomic operations)** đảm bảo tính nhất quán và an toàn khi thực hiện trong môi trường đa nhân (multicore).   + Ứng dụng: Các hệ thống sử dụng vi xử lý đa lõi, nơi các tác vụ cần đảm bảo không bị xung đột khi truy cập tài nguyên chia sẻ. |

## Content of Software Layers

### Microcontroller Abstraction Layer



**Hình 6: Microcontroller Abstraction Layer (MCAL)**

**Microcontroller Abstraction Layer (MCAL)** là lớp phần mềm thấp nhất trong **Basic Software (BSW)** của kiến trúc **AUTOSAR**, cung cấp các trình điều khiển để giao tiếp trực tiếp với phần cứng của **bộ điều khiển vi xử lý (µC)**. Các trình điều khiển trong MCAL được chia thành các nhóm chính dựa trên chức năng của chúng. Dưới đây là chi tiết về từng nhóm:

#### ****Microcontroller Drivers (Trình điều khiển Microcontroller)****

* **Chức năng**:
  + Cung cấp trình điều khiển cho các **bộ ngoại vi nội** (on-chip peripherals) của bộ điều khiển vi xử lý.
  + Các chức năng này truy cập trực tiếp vào phần cứng µC mà không cần qua các lớp trừu tượng khác.
* **Ví dụ**:
  + **Watchdog**: Giám sát hệ thống, đảm bảo không bị treo bằng cách khởi động lại hệ thống nếu phát hiện lỗi.
  + **General Purpose Timer (GPT)**: Điều khiển các bộ đếm thời gian cho các ứng dụng yêu cầu độ chính xác cao.

#### ****Communication Drivers (Trình điều khiển Giao tiếp)****

* **Chức năng**:
  + Quản lý các giao tiếp nội bộ và giao tiếp giữa các ECU cũng như giữa các hệ thống trong xe.
  + Tương ứng với lớp **Data Link Layer** trong mô hình OSI.
* **Ví dụ**:
  + **SPI (Serial Peripheral Interface)**: Truyền thông nối tiếp giữa các thiết bị ngoại vi và µC.
  + **CAN (Controller Area Network)**: Giao tiếp trên mạng CAN trong xe.
  + **LIN (Local Interconnect Network)**: Truyền thông nối tiếp tốc độ thấp.

#### ****Memory Drivers (Trình điều khiển Bộ nhớ)****

* **Chức năng**:
  + Điều khiển các thiết bị bộ nhớ, bao gồm cả bộ nhớ nội bộ trên chip và bộ nhớ ngoại vi được ánh xạ bộ nhớ.
* **Ví dụ**:
  + **Flash internal**: Quản lý bộ nhớ flash bên trong µC.
  + **EEPROM internal**: Điều khiển bộ nhớ không thay đổi (non-volatile memory) bên trong.
  + **Flash external**: Điều khiển bộ nhớ flash bên ngoài được ánh xạ vào không gian địa chỉ bộ nhớ của µC.

#### ****I/O Drivers (Trình điều khiển I/O)****

* **Chức năng**:
  + Cung cấp trình điều khiển cho các thiết bị đầu vào/đầu ra (I/O) bao gồm cả analog và digital.
* **Ví dụ**:
  + **ADC (Analog-to-Digital Converter)**: Chuyển đổi tín hiệu analog từ cảm biến thành tín hiệu số.
  + **PWM (Pulse Width Modulation)**: Điều khiển các tín hiệu dạng xung, thường dùng trong điều khiển động cơ hoặc đèn.
  + **DIO (Digital Input/Output)**: Quản lý tín hiệu số đầu vào và đầu ra.

#### ****Crypto Drivers (Trình điều khiển Mã hóa)****

* **Chức năng**:
  + Điều khiển các thiết bị mã hóa trên chip, hỗ trợ các tác vụ bảo mật.
* **Ví dụ**:
  + **SHE (Secure Hardware Extension)**: Cung cấp các chức năng mã hóa cơ bản như quản lý khóa và bảo mật dữ liệu.
  + **HSM (Hardware Security Module)**: Hỗ trợ các tác vụ bảo mật tiên tiến, như mã hóa/giải mã dữ liệu và xác thực.

#### ****Wireless Communication Drivers (Trình điều khiển Giao tiếp Không dây)****

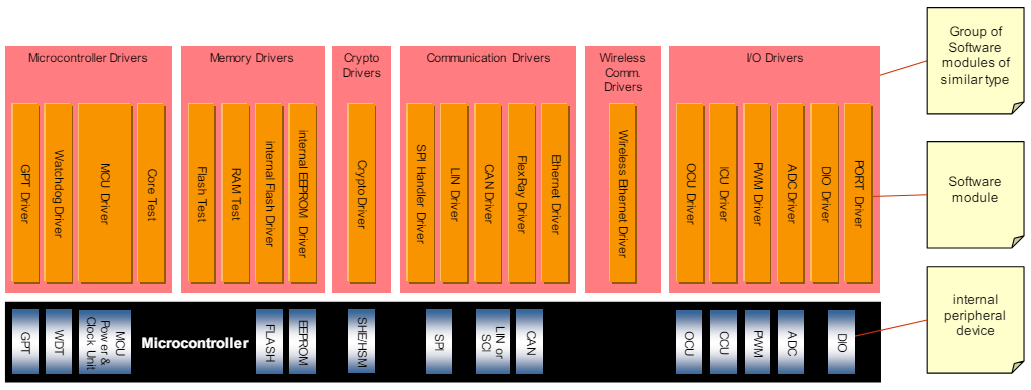
* **Chức năng**:
  + Điều khiển các thiết bị hỗ trợ giao tiếp không dây trong xe hoặc giữa xe và hệ thống bên ngoài.
* **Ví dụ**:
  + **Giao tiếp trong xe**: Hỗ trợ các hệ thống như Bluetooth hoặc Wi-Fi để giao tiếp giữa các thiết bị nội bộ trong xe.
  + **Giao tiếp ngoài ECU**: Truyền thông với các thiết bị bên ngoài, như các dịch vụ đám mây hoặc hệ thống IoT.

#### ****Tóm tắt****

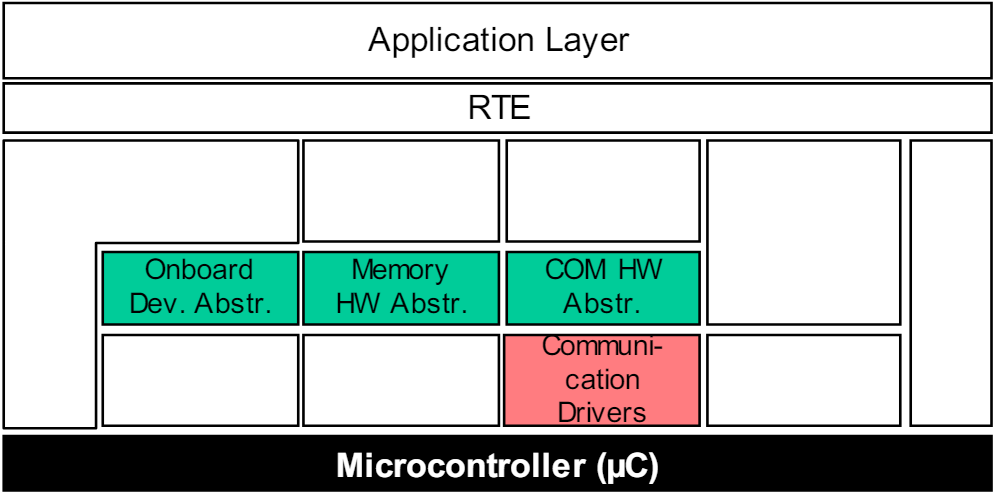
**Microcontroller Abstraction Layer (MCAL)** trong **AUTOSAR** cung cấp các trình điều khiển để truy cập và quản lý phần cứng của bộ điều khiển vi xử lý. Các nhóm mô-đun chính trong MCAL bao gồm:

* **Microcontroller Drivers**: Điều khiển các ngoại vi nội (Watchdog, Timer).
* **Communication Drivers**: Quản lý giao tiếp (CAN, SPI, LIN).
* **Memory Drivers**: Điều khiển bộ nhớ (Flash, EEPROM).
* **I/O Drivers**: Quản lý đầu vào/đầu ra (ADC, PWM, DIO).
* **Crypto Drivers**: Hỗ trợ mã hóa và bảo mật (SHE, HSM).
* **Wireless Communication Drivers**: Điều khiển giao tiếp không dây (Bluetooth, Wi-Fi).

Các trình điều khiển này tạo ra một nền tảng chuẩn hóa, giúp phần mềm ứng dụng tương tác hiệu quả và đồng nhất với phần cứng µC mà không cần lo lắng về sự phức tạp của phần cứng cụ thể.



### Microcontroller Abstraction Layer: SPIHandlerDriver



**SPIHandlerDriver** là một trình điều khiển (driver) đặc biệt trong **AUTOSAR**, có chức năng cho phép truy cập đồng thời từ nhiều **khách hàng** (client) vào một hoặc nhiều **bus SPI** (Serial Peripheral Interface). **SPI** là giao thức truyền thông nối tiếp được sử dụng để kết nối các thiết bị ngoại vi với **vi điều khiển (µC)**.

#### ****Chức Năng và Cách Thức Hoạt Động của SPIHandlerDriver****

**1. Truy Cập Đồng Thời:**

* **SPIHandlerDriver** cung cấp khả năng truy cập đồng thời cho nhiều **khách hàng** (các phần mềm hoặc ứng dụng yêu cầu giao tiếp với các thiết bị qua SPI).
* Điều này có nghĩa là, nhiều thành phần phần mềm (ví dụ: các phần mềm ứng dụng hoặc các mô-đun trong hệ thống) có thể sử dụng SPI cùng một lúc mà không gây xung đột, nhờ vào cơ chế **quản lý truy cập** và **phân phối tài nguyên** của SPIHandlerDriver.

**2. Quản Lý Các Chân Chip Select:**

* **Chip Select (CS)** là một tín hiệu trong giao thức SPI, dùng để xác định thiết bị ngoại vi nào sẽ nhận dữ liệu từ vi điều khiển. Mỗi thiết bị kết nối với SPI bus thường có một chân CS riêng biệt.
* Trong một hệ thống với nhiều thiết bị SPI, mỗi thiết bị cần một chân CS riêng biệt để vi điều khiển biết được thiết bị nào cần được truy cập.
* **SPIHandlerDriver** chịu trách nhiệm trực tiếp quản lý và xử lý các **chân CS** này. Điều này có nghĩa là **SPIHandlerDriver** sẽ kiểm soát các tín hiệu này và không để chúng bị truy cập hoặc thay đổi từ các driver khác như **DIO Driver** (Digital Input/Output Driver).

**3. Trừu Tượng Hóa Các Chức Năng Của SPI:**

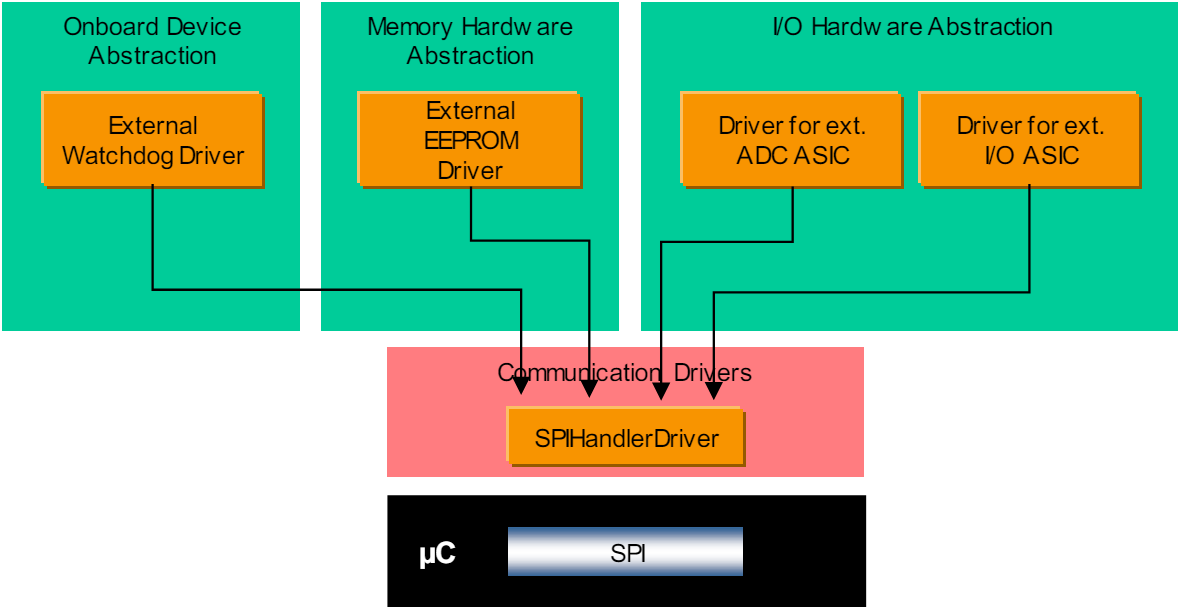
* **SPIHandlerDriver** đóng vai trò là lớp trừu tượng hóa (abstraction layer) giữa phần mềm ứng dụng và phần cứng SPI, giúp **phần mềm** không phải quan tâm đến cách thức thực hiện giao tiếp phần cứng qua các chân **CS**.
* **SPIHandlerDriver** cung cấp một giao diện chuẩn, cho phép các phần mềm hoặc ứng dụng chỉ cần gọi các chức năng mà không cần biết chi tiết phần cứng SPI của vi điều khiển. Điều này giúp phần mềm dễ dàng tương tác với SPI mà không cần phải xử lý trực tiếp từng chân CS.

**4. Sự Tách Biệt Với DIO Driver:**

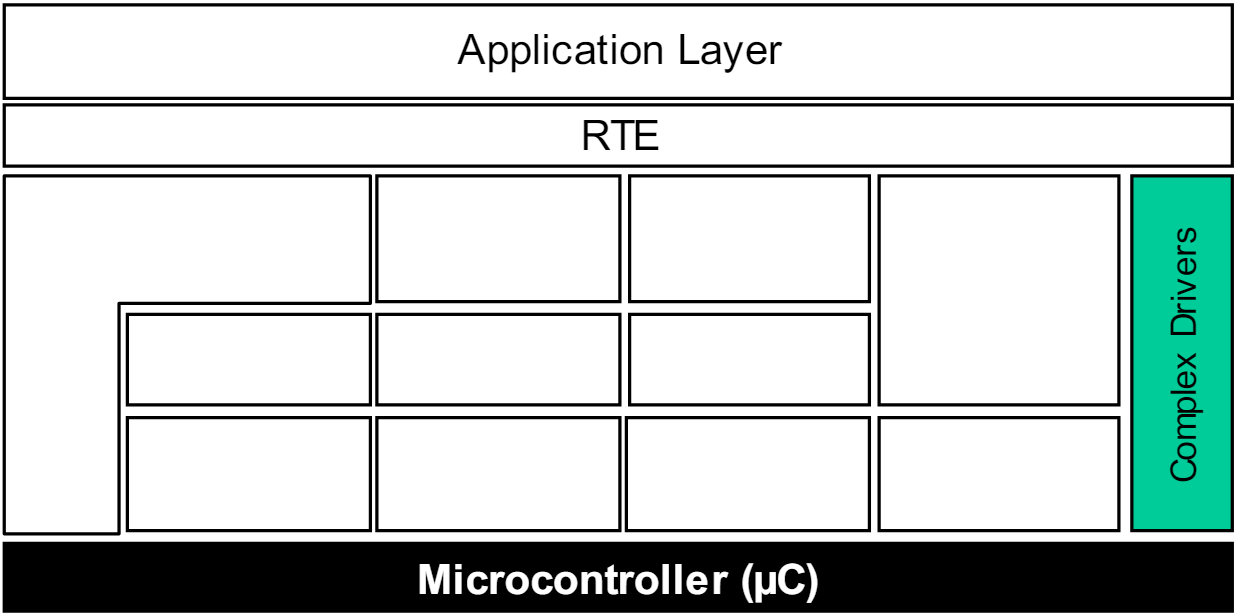
* **SPIHandlerDriver** kiểm soát các chân **Chip Select (CS)** trực tiếp, điều này có nghĩa là các chân này **không được sử dụng** hoặc **không có sẵn** cho **DIO Driver** (một trình điều khiển cho các chân I/O số).
* DIO Driver chỉ có thể quản lý các tín hiệu I/O số chung như **Digital Inputs/Outputs**, không bao gồm các chức năng quản lý chân **CS** riêng biệt của SPI. Điều này giúp tách biệt và duy trì sự nhất quán trong việc quản lý tài nguyên phần cứng.

#### ****Tóm Tắt****

* **SPIHandlerDriver** cho phép nhiều **khách hàng** sử dụng **SPI bus** đồng thời mà không gặp xung đột.
* **Chip Select (CS)** được quản lý trực tiếp bởi **SPIHandlerDriver**, không bị can thiệp bởi **DIO Driver**, giúp giữ sự phân tách giữa các chức năng.
* **SPIHandlerDriver** cung cấp lớp trừu tượng hóa để phần mềm không phải quan tâm đến chi tiết phần cứng SPI, chỉ cần sử dụng API chuẩn từ driver này.



### Complex Drivers



**Complex Driver** trong **AUTOSAR** là một mô-đun phần mềm thực hiện các chức năng không được chuẩn hóa trong **Basic Software (BSW)** stack. Các driver này được sử dụng trong các ứng dụng yêu cầu các xử lý đặc biệt và thời gian thực cho các cảm biến và **actuator** phức tạp, nơi cần truy cập trực tiếp vào **microcontroller (µC)** và các tính năng phần cứng đặc biệt.

#### ****Mô Tả và Chức Năng của Complex Driver****

**1. Chức Năng và Ứng Dụng:**

* **Complex Driver** được sử dụng trong các trường hợp **yêu cầu tính toán và xử lý phức tạp**, chẳng hạn như trong việc **đánh giá cảm biến** hoặc **kiểm soát actuator** (thiết bị thực thi như van, bơm, hoặc động cơ) mà không có chuẩn hóa sẵn trong **Basic Software Stack** của AUTOSAR.
* Các ví dụ bao gồm:
  + **Kiểm soát phun nhiên liệu:** Quản lý việc phun nhiên liệu trong động cơ dựa trên các dữ liệu cảm biến và yêu cầu thời gian thực.
  + **Kiểm soát van điện:** Điều khiển các van điện để điều chỉnh các yếu tố như nhiệt độ hoặc lưu lượng.
  + **Phát hiện vị trí tăng dần:** Nhận diện sự thay đổi vị trí của các bộ phận cơ khí trong hệ thống (ví dụ: piston trong động cơ) để điều khiển động cơ chính xác.

**2. Yêu Cầu Chức Năng và Thời Gian:**

* **Complex Driver** phải đáp ứng các yêu cầu đặc biệt về **chức năng** và **thời gian**:
  + **Chức năng:** Cung cấp khả năng xử lý các tác vụ phức tạp không được chuẩn hóa trước đó.
  + **Thời gian:** Đảm bảo tính đúng đắn trong các yêu cầu **thời gian thực**, với khả năng xử lý tín hiệu và phản hồi nhanh chóng từ các thiết bị cảm biến và actuator trong môi trường thời gian thực.

#### ****Thuộc Tính của Complex Driver****

**1. Triển Khai Phụ Thuộc Vào Vi Điều Khiển (µC), ECU và Ứng Dụng:**

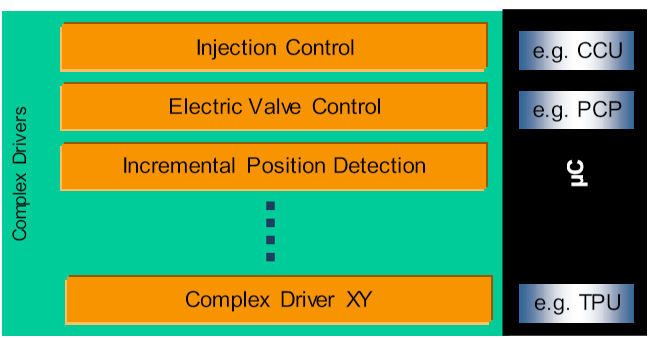
* **Complex Driver** được thiết kế và triển khai **riêng biệt cho từng vi điều khiển (µC)**, **ECU (Electronic Control Unit)** và ứng dụng cụ thể. Điều này có nghĩa là các driver này cần phải tùy chỉnh và tối ưu hóa để làm việc với các đặc điểm phần cứng của vi điều khiển và ECU cụ thể trong hệ thống.

**2. Giao Diện Trên Cùng và Dưới Cùng:**

* **Giao diện trên cùng (Upward Interface):**
  + Giao diện của **Complex Driver** với các **Software Components (SW-Cs)** được xác định và triển khai theo **AUTOSAR Interface** chuẩn. Điều này giúp **Complex Driver** giao tiếp hiệu quả với các mô-đun phần mềm khác trong hệ thống.
* **Giao diện dưới cùng (Downward Interface):**
  + Giao diện này **truy cập vào các giao diện chuẩn hóa (Standardized Interfaces)** nhưng chỉ có thể truy cập ở mức độ **hạn chế**. Điều này có nghĩa là **Complex Driver** không sử dụng tất cả các giao diện chuẩn hóa mà chỉ truy cập những phần cần thiết trong phần cứng hoặc các mô-đun phần mềm để thực hiện các chức năng đặc biệt của nó.

#### ****Tóm Tắt****

* **Complex Driver** là mô-đun phần mềm phức tạp trong **AUTOSAR**, thực hiện các chức năng đặc biệt cho các cảm biến và actuator phức tạp.
* Các driver này đáp ứng các yêu cầu về **chức năng** và **thời gian** đặc biệt trong các ứng dụng yêu cầu truy cập trực tiếp vào phần cứng của **µC**.
* **Triển khai và giao diện** của **Complex Driver** phụ thuộc vào đặc thù của vi điều khiển, ECU và ứng dụng.
* Chúng có **giao diện trên cùng** tuân thủ **AUTOSAR Interface** và **giao diện dưới cùng** chỉ truy cập hạn chế vào các giao diện chuẩn hóa.



### ECU Abstraction: I/O Hardware Abstraction



**I/O Hardware Abstraction** trong **AUTOSAR** là một nhóm các mô-đun phần mềm có nhiệm vụ trừu tượng hóa và cung cấp một lớp giao tiếp giữa phần mềm và phần cứng của ECU, đặc biệt là các thiết bị **Input/Output (I/O)**, bao gồm cả các thiết bị ngoại vi và bố cục phần cứng của ECU. Mục tiêu chính của lớp này là giúp phần mềm không phải quan tâm đến chi tiết phần cứng của ECU, đồng thời hỗ trợ việc quản lý và truy cập các tín hiệu I/O từ các thiết bị ngoại vi một cách dễ dàng và linh hoạt.

#### ****Chức Năng và Nhiệm Vụ của I/O Hardware Abstraction****

**1. Đại Diện cho Tín Hiệu I/O:**

* **I/O Hardware Abstraction** cung cấp một lớp trừu tượng để đại diện các tín hiệu I/O, chẳng hạn như dòng điện, điện áp, tần số, v.v. mà các thiết bị I/O sử dụng để giao tiếp với hệ thống phần mềm. Điều này cho phép phần mềm ứng dụng không phải lo lắng về cách các tín hiệu này được truyền và xử lý trong phần cứng của ECU.
* Mô-đun này cung cấp một cách thức chuẩn hóa để các tín hiệu I/O có thể được kết nối và sử dụng trên phần cứng ECU mà không cần biết chi tiết phần cứng của từng loại thiết bị I/O (ví dụ: cảm biến hoặc bộ điều khiển).

**2. Ẩn Thuộc Tính Bố Cục và Phần Cứng ECU:**

* Một trong những nhiệm vụ quan trọng của **I/O Hardware Abstraction** là **ẩn đi các thuộc tính phần cứng** của ECU từ các lớp phần mềm phía trên (ví dụ: các mô-đun trong **BSW** hoặc **SW-Cs**). Điều này giúp phần mềm có thể làm việc một cách linh hoạt và độc lập với từng loại **microcontroller (µC)** hoặc thiết bị phần cứng cụ thể.
* Cụ thể, các thông tin về cách các chân vi điều khiển được kết nối, cách tín hiệu được đảo ngược, hay các chi tiết phần cứng như cấu trúc của **ECU hardware layout** đều được **ẩn đi**, giúp phần mềm không bị phụ thuộc vào phần cứng cụ thể.

#### ****Thuộc Tính và Triển Khai của I/O Hardware Abstraction****

**1. Triển Khai Phụ Thuộc vào ECU Hardware:**

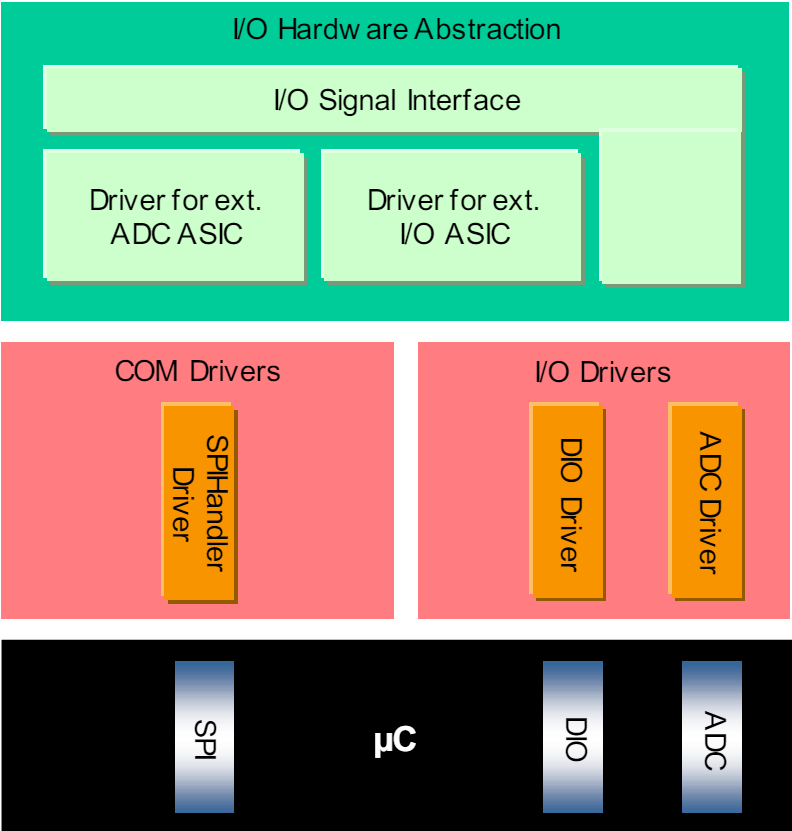
* **I/O Hardware Abstraction** là một mô-đun **phụ thuộc vào phần cứng của ECU** nhưng **độc lập với µC**. Điều này có nghĩa là, trong khi mô-đun này có thể hoạt động với nhiều loại vi điều khiển khác nhau, nó lại có sự phụ thuộc vào cấu trúc phần cứng của **ECU** cụ thể mà nó đang chạy.
* Các tín hiệu I/O có thể thay đổi tùy theo đặc điểm của từng **ECU hardware**, như số lượng chân, loại tín hiệu, hay khả năng xử lý của bộ vi xử lý.

**2. Giao Diện Trên Cùng Độc Lập với µC và ECU Hardware:**

* Giao diện phía trên của **I/O Hardware Abstraction** hoàn toàn **độc lập với µC và ECU hardware**. Điều này có nghĩa là phần mềm ứng dụng không cần phải quan tâm đến việc tín hiệu I/O được xử lý như thế nào trong phần cứng.
* Tuy nhiên, giao diện này sẽ phụ thuộc vào loại tín hiệu I/O mà nó hỗ trợ (như tín hiệu điện áp, dòng điện, tần số). Giao diện này được triển khai theo **AUTOSAR interface**, giúp duy trì sự chuẩn hóa và tính linh hoạt trong việc quản lý tín hiệu I/O.

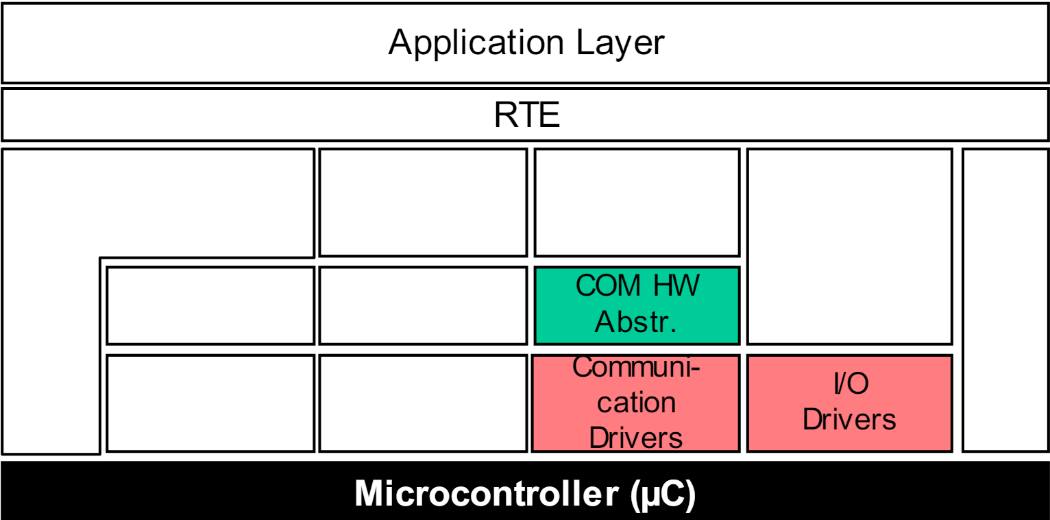
#### ****Tóm Tắt****

* **I/O Hardware Abstraction** là một lớp phần mềm trong AUTOSAR giúp trừu tượng hóa các tín hiệu I/O và cấu trúc phần cứng của ECU.
* Mô-đun này đại diện cho các tín hiệu I/O (như điện áp, dòng điện, tần số) mà các thiết bị ngoại vi sử dụng và giúp phần mềm ứng dụng không phải lo lắng về các chi tiết phần cứng của ECU.
* **Triển khai** của mô-đun này phụ thuộc vào **ECU hardware**, nhưng **độc lập với µC**, giúp phần mềm dễ dàng tương tác với nhiều loại vi điều khiển mà không bị ràng buộc vào phần cứng cụ thể.
* Giao diện trên cùng của **I/O Hardware Abstraction** hoàn toàn **độc lập với phần cứng**, tạo ra sự chuẩn hóa và linh hoạt trong việc truy cập và quản lý tín hiệu I/O.



### ECU Abstraction: Communication Hardware Abstraction

**Lớp Trừu tượng Phần cứng Giao tiếp (Communication Hardware Abstraction Layer)** là một nhóm các mô-đun phần mềm trong **AUTOSAR** có nhiệm vụ trừu tượng hóa và quản lý các bộ điều khiển giao tiếp và bố cục phần cứng của Đơn vị Điều khiển Điện tử (ECU). Lớp này đóng vai trò quan trọng trong việc cung cấp giao diện truy cập thống nhất cho các hệ thống giao tiếp, bất kể vị trí của các bộ điều khiển giao tiếp (trên chip hay trên bo mạch). Điều này cho phép các mô-đun phần mềm không cần phải lo lắng về cách thức hay nơi các bộ điều khiển giao tiếp được triển khai.



#### ****Chức Năng và Nhiệm Vụ của Lớp Trừu Tượng Phần Cứng Giao Tiếp****

**1. Cung Cấp Cơ Chế Bình Đẳng Để Truy Cập Kênh Bus:**

* **Lớp Trừu tượng Phần cứng Giao tiếp** đảm bảo rằng tất cả các kênh giao tiếp (bus) có thể được truy cập một cách **bình đẳng**, không phân biệt vị trí của chúng trong hệ thống (ví dụ: có thể là kênh trên vi điều khiển hoặc kênh trên bo mạch).
* Các hệ thống giao tiếp phổ biến như **CAN**, **LIN**, **FlexRay** yêu cầu các cơ chế này để quản lý giao tiếp giữa các thiết bị trong ECU mà không cần phải quan tâm đến chi tiết phần cứng cụ thể của các kênh bus.

**2. Trừu Tượng Hóa Các Bộ Điều Khiển Giao Tiếp:**

* Mô-đun trong lớp này sẽ **trừu tượng hóa các bộ điều khiển giao tiếp** trên vi điều khiển hoặc ASIC, đảm bảo rằng các mô-đun phần mềm trên các lớp phần mềm cao hơn có thể truy cập vào chúng thông qua các giao diện bus chuẩn hóa mà không cần phải biết về cách thức phần cứng thực hiện giao tiếp.
* Điều này giúp đơn giản hóa quá trình phát triển phần mềm và giảm sự phụ thuộc vào cấu trúc phần cứng cụ thể của **ECU**.

#### ****Thuộc Tính và Triển Khai của Lớp Trừu Tượng Phần Cứng Giao Tiếp****

**1. Triển Khai Phụ Thuộc vào Phần Cứng của ECU:**

* Lớp này **phụ thuộc vào phần cứng của ECU** và các **thiết bị ngoại vi** (ví dụ: ASIC, bộ điều khiển CAN trên bo mạch, v.v.). Tuy nhiên, **lớp trừu tượng phần cứng giao tiếp này độc lập với vi điều khiển (µC)**.
* Các bộ điều khiển giao tiếp có thể được triển khai dưới dạng **vi điều khiển tích hợp (internal)** hoặc **ASIC (external)**, nhưng lớp này sẽ quản lý việc truy cập cho cả hai loại.

**2. Giao Diện Trên Cùng Phụ Thuộc vào Bus:**

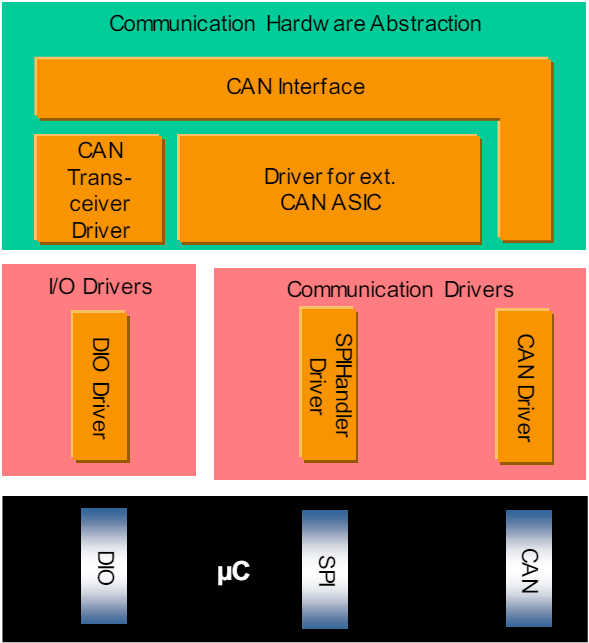
* Giao diện **trên cùng** của lớp này **phụ thuộc vào loại bus giao tiếp** (ví dụ: CAN, LIN, FlexRay). Tuy nhiên, giao diện này sẽ **độc lập với vi điều khiển** và **độc lập với phần cứng của ECU**. Điều này có nghĩa là phần mềm không cần phải biết về việc bộ điều khiển giao tiếp cụ thể được tích hợp trong vi điều khiển hay được cung cấp bởi một ASIC riêng biệt.
* Các giao diện chuẩn hóa này sẽ giúp các mô-đun phần mềm có thể tương tác với các kênh giao tiếp mà không gặp phải vấn đề về sự khác biệt phần cứng.

#### ****Ví Dụ Cụ Thể về Triển Khai****

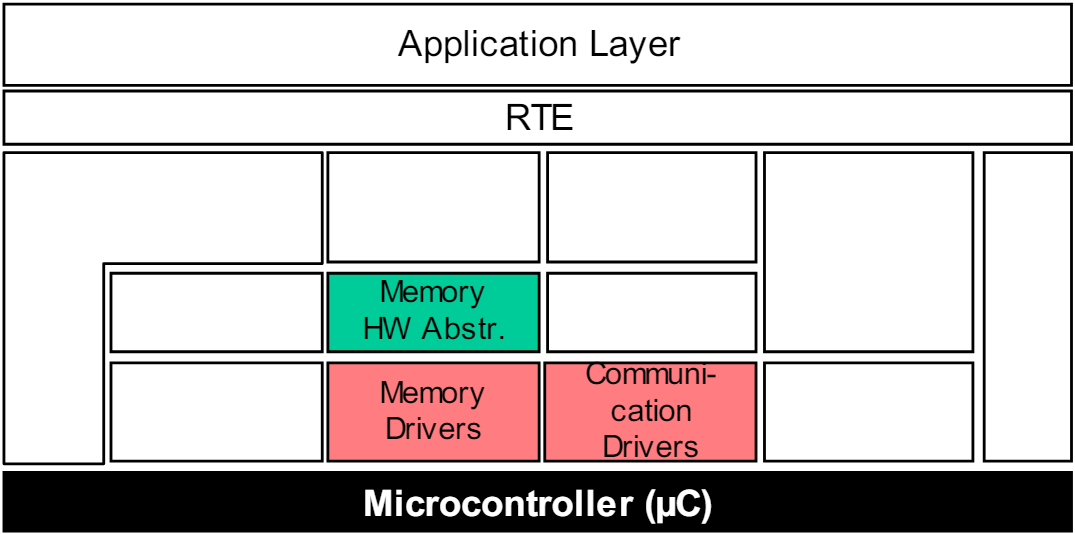
Giả sử một **ECU** có một **vi điều khiển với hai kênh CAN** tích hợp sẵn và một **ASIC trên bo mạch với bốn bộ điều khiển CAN**. **ASIC CAN** này kết nối với vi điều khiển qua giao tiếp **SPI**. Trong trường hợp này, **Lớp Trừu Tượng Phần Cứng Giao Tiếp** sẽ đảm bảo rằng phần mềm có thể truy cập vào các kênh CAN (dù chúng là kênh vi điều khiển tích hợp hay kênh trên ASIC) một cách đồng nhất và không cần quan tâm đến vị trí hoặc cách thức kết nối của chúng.

#### ****Tóm Tắt****

* **Lớp Trừu Tượng Phần Cứng Giao Tiếp** trong **AUTOSAR** cung cấp một cơ chế để truy cập vào các bộ điều khiển giao tiếp mà không phân biệt chúng được tích hợp trong vi điều khiển hay ASIC.
* Lớp này giúp ẩn đi các chi tiết phức tạp của phần cứng giao tiếp, từ đó giúp phần mềm có thể làm việc một cách linh hoạt và đồng nhất trên các hệ thống giao tiếp như **CAN**, **LIN**, **FlexRay**.
* Triển khai của lớp này **phụ thuộc vào phần cứng của ECU** nhưng **độc lập với vi điều khiển**, giúp dễ dàng quản lý các kênh giao tiếp trong hệ thống mà không cần phải thay đổi phần mềm khi thay đổi phần cứng.



### Scope: Memory Hardware Abstraction



**Lớp Trừu tượng Phần cứng Bộ nhớ (Memory Hardware Abstraction Layer)** là một nhóm các mô-đun trong **AUTOSAR** có nhiệm vụ trừu tượng hóa việc truy cập và quản lý 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 (ECU). Lớp này giúp các mô-đun phần mềm có thể truy cập vào các thiết bị bộ nhớ như **EEPROM** hoặc **Flash memory** mà không cần phải lo lắng về chi tiết phần cứng của chúng, tạo ra một cơ chế đồng nhất cho việc truy cập bộ nhớ từ các nguồn khác nhau.

#### ****Chức Năng và Nhiệm Vụ của Lớp Trừu Tượng Phần Cứng Bộ Nhớ****

**1. Cung Cấp Cơ Chế Bình Đẳng Để Truy Cập Các Thiết Bị Bộ Nhớ:**

* **Lớp Trừu tượng Bộ nhớ** cung cấp cơ chế truy cập đồng nhất và **bình đẳng** cho các thiết bị bộ nhớ cả **nội bộ (on-chip)** và **ngoại vi (off-chip)**, chẳng hạn như **EEPROM** và **Flash**. Điều này giúp đơn giản hóa việc phát triển phần mềm, vì các mô-đun phần mềm không cần phải thay đổi khi phần cứng bộ nhớ thay đổi (ví dụ: thay đổi từ **EEPROM trên chip** sang **EEPROM ngoại vi**).

**2. Trừu Tượng Hóa Các Thiết Bị Bộ Nhớ Khác Nhau:**

* Các thiết bị bộ nhớ như **EEPROM** trên chip và **EEPROM ngoại vi** có thể truy cập thông qua một giao diện trừu tượng chung mà lớp này cung cấp.
* Để làm điều này, lớp **Memory Hardware Abstraction** có thể **giả lập** một thiết bị bộ nhớ (ví dụ: **EEPROM**) trên các thiết bị bộ nhớ khác nhau, như **Flash**, và cung cấp một giao diện thống nhất để phần mềm có thể truy cập tất cả các loại bộ nhớ này một cách đồng nhất.

**3. Giao Diện Trừu Tượng Bộ Nhớ:**

* Các **trình điều khiển bộ nhớ** sẽ truy cập thông qua các **mô-đun trừu tượng/giả lập** của bộ nhớ, ví dụ như **trừu tượng EEPROM**. Điều này giúp chuẩn hóa việc truy cập vào bộ nhớ bất kể thiết bị bộ nhớ cụ thể nào đang được sử dụng (EEPROM, Flash).

#### ****Thuộc Tính và Triển Khai của Lớp Trừu Tượng Phần Cứng Bộ Nhớ****

**1. Triển Khai Phụ Thuộc vào Thiết Bị Ngoại Vi:**

* Lớp trừu tượng bộ nhớ này **độc lập với vi điều khiển (µC)**, nhưng **phụ thuộc vào thiết bị ngoại vi**. Điều này có nghĩa là lớp trừu tượng không quan tâm đến loại vi điều khiển đang sử dụng, nhưng nó sẽ phụ thuộc vào phần cứng bộ nhớ ngoại vi (ví dụ, **Flash memory** hay **EEPROM**).

**2. Giao Diện Trên Cùng Độc Lập Với Vi Điều Khiển và Phần Cứng ECU:**

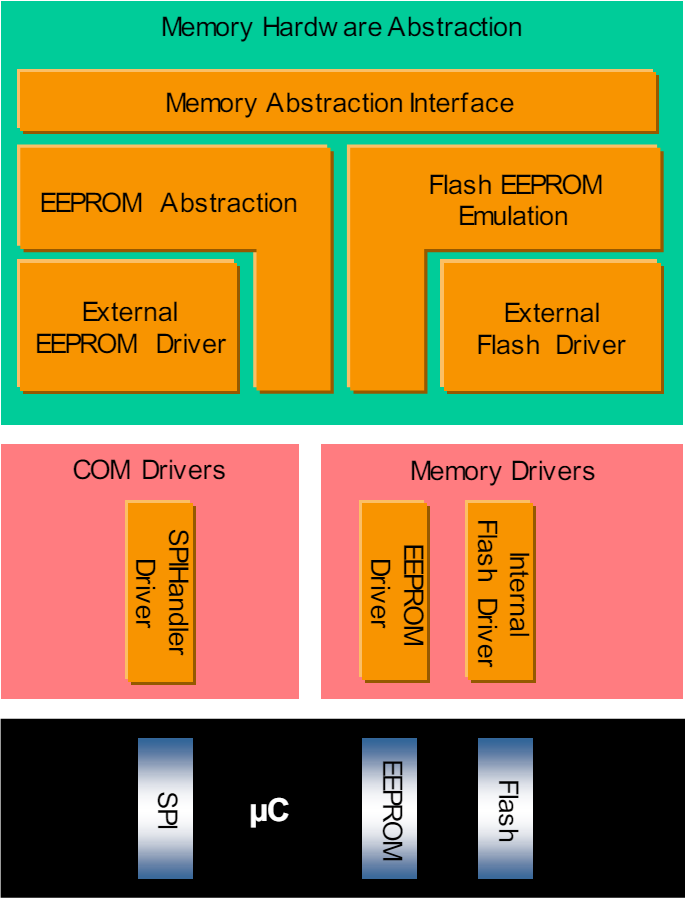
* Giao diện trên cùng của lớp trừu tượng bộ nhớ này là **độc lập với vi điều khiển và phần cứng của Đơn vị Điều khiển Điện tử (ECU)**. Điều này có nghĩa là phần mềm có thể sử dụng các giao diện này để truy cập bộ nhớ mà không phải thay đổi khi phần cứng (như vi điều khiển hoặc bộ điều khiển bộ nhớ) thay đổi.
* Tuy nhiên, giao diện trên cùng sẽ **phụ thuộc vào loại bộ nhớ** (EEPROM, Flash) mà hệ thống đang sử dụng.

#### ****Ví Dụ Cụ Thể về Triển Khai****

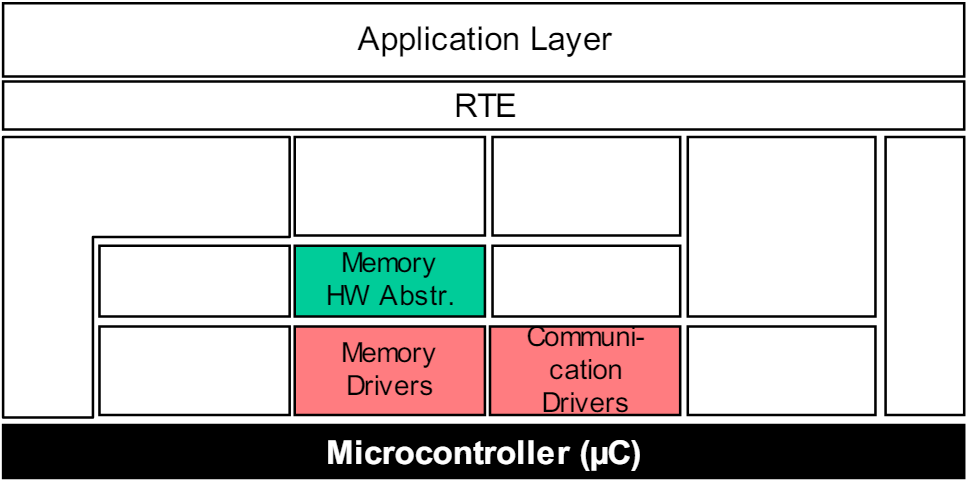
* Một ví dụ về **Lớp Trừu Tượng Bộ Nhớ** có thể là **EEPROM trên chip** và **EEPROM ngoại vi** (có thể là Flash) có thể truy cập thông qua cùng một giao diện trừu tượng bộ nhớ. Khi phần mềm yêu cầu truy cập vào bộ nhớ, lớp trừu tượng này sẽ điều hướng và thực hiện việc truy cập này một cách phù hợp mà không cần phần mềm phải biết bộ nhớ nào được sử dụng.
* Chẳng hạn, nếu phần cứng sử dụng **EEPROM Flash ngoại vi**, lớp này sẽ cung cấp giao diện truy cập đồng nhất mà không cần thay đổi trong mã phần mềm.

#### ****Tóm Tắt****

* **Lớp Trừu Tượng Phần Cứng Bộ Nhớ** trong **AUTOSAR** cung cấp một cách tiếp cận thống nhất để truy cập các thiết bị bộ nhớ như **EEPROM** và **Flash**, bất kể chúng được tích hợp trên chip hay là bộ nhớ ngoại vi.
* Lớp này **giúp trừu tượng hóa việc truy cập bộ nhớ**, ẩn đi các chi tiết phần cứng, và giúp phần mềm có thể làm việc với các loại bộ nhớ khác nhau mà không cần quan tâm đến các đặc tính phần cứng cụ thể.
* **Triển khai lớp này** là **độc lập với vi điều khiển**, nhưng **phụ thuộc vào thiết bị bộ nhớ ngoại vi**. Giao diện trừu tượng của lớp này **độc lập với vi điều khiển và phần cứng của ECU**, tạo ra sự linh hoạt và dễ dàng khi thay đổi phần cứng mà không làm ảnh hưởng đến phần mềm.



### 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.

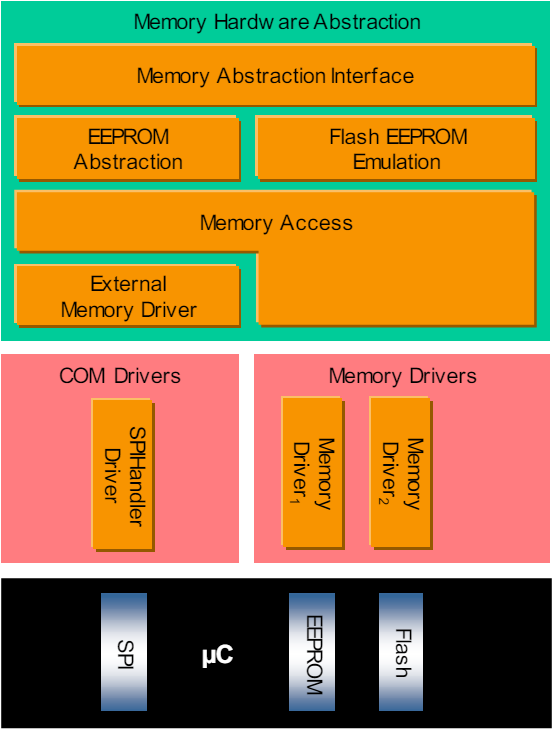
Nhiệm vụ:

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).

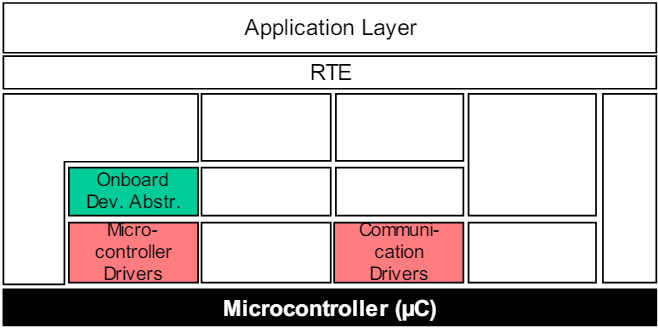
Thuộc tính:

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 (On-Board Device Abstraction Layer)** là một nhóm mô-đun trong **AUTOSAR** có chức năng trừu tượng hóa và quản lý các thiết bị trên bo mạch của Đơn vị Điều khiển Điện tử (ECU). Các thiết bị này thường không phải là cảm biến hay actuator, ví dụ như các **watchdog** (cả nội bộ và ngoại vi), và chúng cần được truy cập thông qua các **Lớp Trừu tượng Bộ điều khiển vi xử lý (MCU Abstraction Layer)**.

#### ****Chức Năng và Nhiệm Vụ của Lớp Trừu Tượng Thiết bị Trên Bo Mạch****

**1. Trừu Tượng Hóa Các Thiết Bị Trên Bo Mạch:**

* **Lớp Trừu tượng Thiết bị Trên bo mạch** chịu trách nhiệm trừu tượng hóa các thiết bị trên bo mạch cụ thể của **Đơn vị Điều khiển Điện tử (ECU)**. Điều này có nghĩa là lớp này cung cấp giao diện phần mềm để truy cập các thiết bị phần cứng như **watchdog**, nhưng không phải cảm biến hoặc actuator. Những thiết bị này có thể bao gồm các **watchdog nội bộ (internal watchdog)**, **watchdog ngoại vi (external watchdog)**, hoặc các thiết bị khác cần được quản lý mà không liên quan đến việc thu thập dữ liệu từ cảm biến hoặc điều khiển actuator.
* Các thiết bị trên bo mạch này có thể truy cập thông qua các giao diện phần mềm của lớp **MCU Abstraction Layer**, làm cho việc giao tiếp với phần cứng trên bo mạch trở nên dễ dàng và thống nhất.

**2. Cung Cấp Giao Diện Đơn Giản Để Truy Cập Các Thiết Bị Trên Bo Mạch:**

* Lớp trừu tượng này không yêu cầu phần mềm phải biết chi tiết về phần cứng cụ thể của các thiết bị. Ví dụ, trong trường hợp **watchdog**, lớp này sẽ xử lý các yêu cầu từ phần mềm để kiểm tra hoặc khởi động lại hệ thống mà không cần phải biết liệu **watchdog** đang được xử lý trên một **vi điều khiển cụ thể** hay là một **thiết bị ngoại vi**.

#### ****Thuộc Tính và Triển Khai của Lớp Trừu Tượng Thiết bị Trên Bo Mạch****

**1. Triển Khai Độc Lập với Vi Điều Khiển, Phụ Thuộc vào Thiết Bị Ngoại Vi:**

* Lớp trừu tượng thiết bị trên bo mạch này được thiết kế **độc lập với vi điều khiển (µC)**, nhưng lại **phụ thuộc vào thiết bị ngoại vi**. Điều này có nghĩa là lớp này sẽ hoạt động với các thiết bị phần cứng cụ thể, chẳng hạn như **watchdog** hay các thiết bị trên bo mạch, mà không liên quan đến loại vi điều khiển mà hệ thống đang sử dụng. Điều này giúp giữ cho phần mềm không bị ảnh hưởng khi thay đổi vi điều khiển, nhưng vẫn có thể hỗ trợ các thiết bị ngoại vi khác nhau.

**2. Giao Diện Trên Cùng Độc Lập Với Vi Điều Khiển và Phụ Thuộc Một Phần Vào Phần Cứng của ECU:**

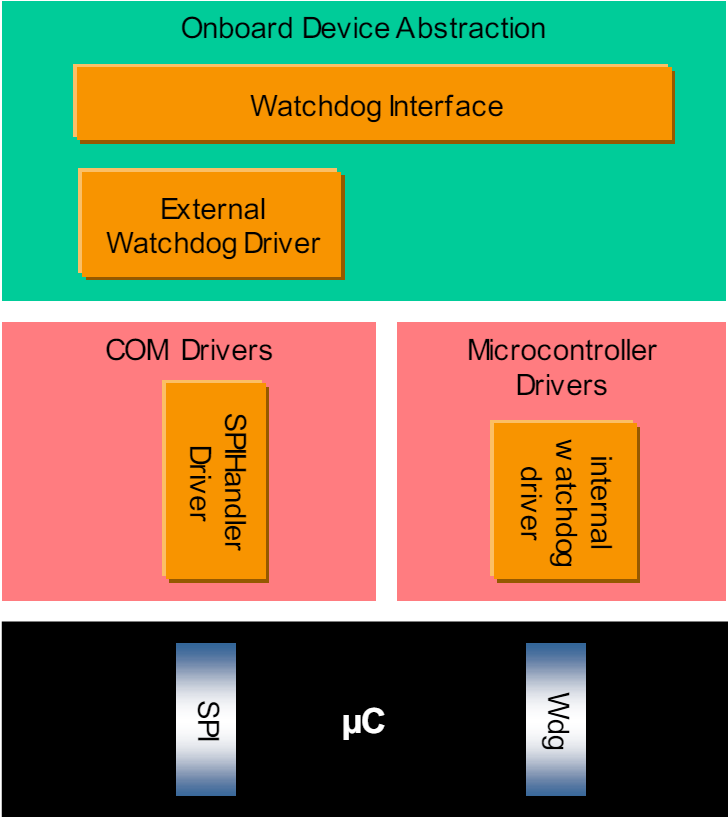
* Giao diện trên cùng của lớp này **độc lập với vi điều khiển**, nhưng lại **phụ thuộc vào phần cứng của ECU**. Điều này có nghĩa là lớp này sẽ cung cấp một giao diện thống nhất cho phần mềm để tương tác với các thiết bị trên bo mạch, nhưng phần giao diện này vẫn sẽ phải điều chỉnh nếu có thay đổi về phần cứng của ECU (ví dụ như thay đổi loại **watchdog** hoặc các thiết bị ngoại vi khác).

#### ****Ví Dụ Cụ Thể****

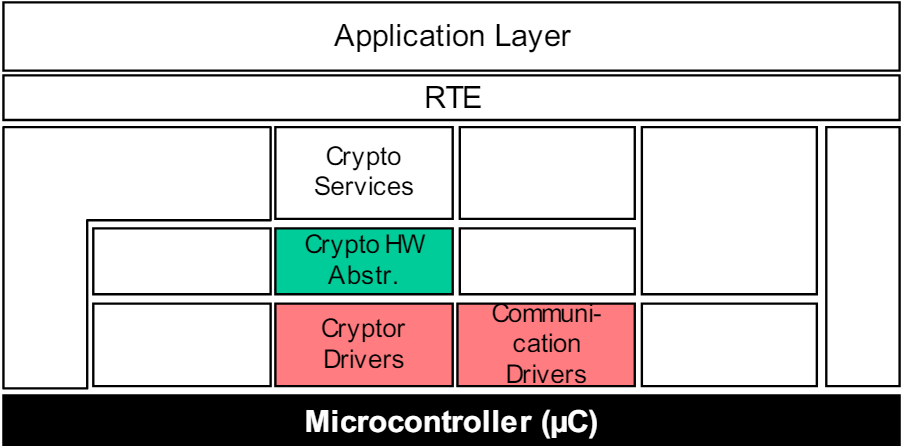
* **Watchdog Nội Bộ:** Một **watchdog nội bộ** trong hệ thống có thể được quản lý qua lớp trừu tượng này, nơi phần mềm chỉ cần sử dụng giao diện chung để kiểm tra và kích hoạt watchdog mà không phải biết về chi tiết phần cứng của watchdog.
* **Watchdog Ngoại Vi:** Trong trường hợp có một **watchdog ngoại vi** được kết nối qua một giao diện như **SPI**, lớp trừu tượng thiết bị trên bo mạch sẽ cung cấp cơ chế để phần mềm có thể làm việc với cả **watchdog nội bộ** và **watchdog ngoại vi** thông qua giao diện phần mềm chung.

#### ****Tóm Tắt****

* **Lớp Trừu Tượng Thiết bị Trên Bo Mạch** cung cấp một cơ chế để phần mềm truy cập và tương tác với các thiết bị trên bo mạch của **Đơn vị Điều khiển Điện tử** mà không cần phải hiểu chi tiết về phần cứng của chúng.
* Lớp này **độc lập với vi điều khiển** và chỉ phụ thuộc vào các thiết bị ngoại vi của hệ thống. Điều này giúp hệ thống trở nên linh hoạt và dễ dàng thay đổi phần cứng mà không làm ảnh hưởng đến phần mềm.
* Giao diện của lớp này được **độc lập với vi điều khiển** nhưng vẫn có thể phụ thuộc vào **phần cứng ECU** để phù hợp với các thiết bị ngoại vi khác nhau.



### Scope: Crypto Hardware Abstraction



#### ****Lớp Trừu tượng Phần cứng Mã hóa (Crypto Hardware Abstraction Layer)****

**Lớp Trừu tượng Phần cứng Mã hóa** trong **AUTOSAR** là một nhóm các mô-đun phần mềm có nhiệm vụ trừu tượng hóa và cung cấp giao diện truy cập thống nhất cho các **nguyên tắc mật mã**. Các nguyên tắc này có thể được triển khai dưới dạng **phần cứng nội (on-chip)**, **phần cứng ngoại vi (off-chip)** hoặc **thư viện phần mềm**. Mục tiêu của lớp này là tạo ra một cơ chế đồng nhất và chuẩn hóa để phần mềm truy cập vào các dịch vụ mã hóa mà không cần quan tâm đến cách thức chúng được triển khai.

#### ****Chức năng và Nhiệm vụ****

**1. Trừu tượng hóa vị trí triển khai mã hóa:**

* Lớp này **trừu tượng hóa các nguyên tắc mật mã** (cryptographic primitives) bất kể chúng được thực hiện trong phần cứng nội (trên chip), phần cứng ngoại vi hoặc được cung cấp dưới dạng thư viện phần mềm.
* Ví dụ:
  + **AES (Advanced Encryption Standard)**: Có thể được thực hiện trong một module phần cứng như **SHE (Secure Hardware Extension)** hoặc thông qua một **thư viện phần mềm mã hóa**.
  + **HSM (Hardware Security Module)**: Một phần cứng chuyên dụng trên ECU thực hiện các chức năng mã hóa/giải mã.

**2. Cung cấp cơ chế truy cập bình đẳng:**

* Lớp này cung cấp một cơ chế **truy cập bình đẳng** vào các dịch vụ mật mã, bất kể chúng được triển khai bằng phần cứng hay phần mềm. Điều này đảm bảo rằng các mô-đun phần mềm ứng dụng không cần phải lo lắng về việc nguyên tắc mật mã được thực hiện như thế nào, mà chỉ cần sử dụng giao diện chuẩn hóa.

#### ****Thuộc tính và Triển khai****

**1. Triển khai Độc lập với Vi điều khiển (µC):**

* Lớp trừu tượng này **độc lập với vi điều khiển**. Điều này có nghĩa là việc triển khai mã hóa có thể hoạt động trên nhiều loại vi điều khiển khác nhau mà không cần phải thay đổi lớp này.
* Việc triển khai cụ thể phụ thuộc vào thiết bị mật mã (phần cứng hoặc thư viện phần mềm) mà lớp này cần hỗ trợ.

**2. Giao diện Trên Cùng Độc lập với Vi điều khiển và Phần cứng:**

* Giao diện trên cùng của lớp này là **độc lập với vi điều khiển**, **phần cứng của ECU**, và **thiết bị mật mã**.
* Phần mềm sử dụng các giao diện chuẩn hóa mà lớp này cung cấp để thực hiện các thao tác mã hóa mà không cần quan tâm đến cách thức hoặc nơi các chức năng mã hóa được triển khai.

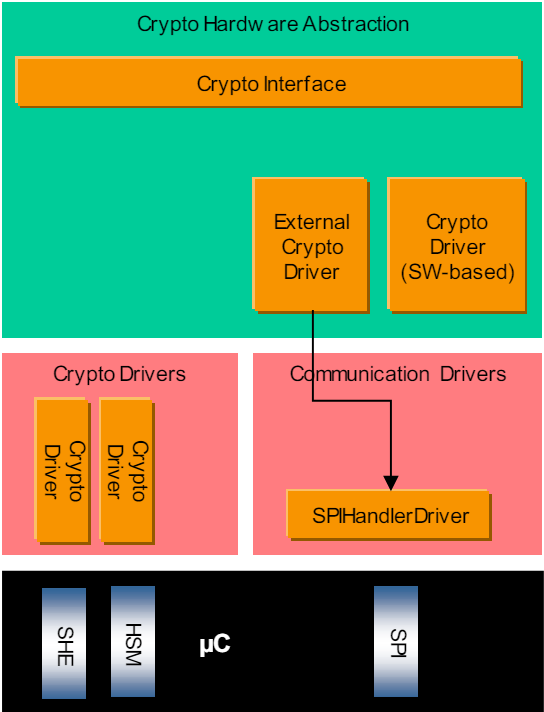
#### ****Ví dụ về Ứng dụng****

1. **Mã hóa và giải mã AES:**
   * Nếu **AES** được thực hiện trong một module phần cứng như **SHE**, lớp trừu tượng sẽ quản lý việc truy cập vào module phần cứng này.
   * Nếu **AES** được cung cấp dưới dạng một thư viện phần mềm, lớp này sẽ cung cấp cùng một giao diện chuẩn để sử dụng các dịch vụ mã hóa, bất kể phần cứng hay phần mềm thực hiện.
2. **Quản lý khóa mật mã:**
   * Lớp trừu tượng có thể quản lý các thao tác liên quan đến **quản lý khóa mật mã** (key management), như lưu trữ, truy xuất, hoặc hủy khóa, bất kể chúng được triển khai bằng phần cứng hoặc phần mềm.
3. **Xác thực và mã hóa dữ liệu:**
   * Trong các hệ thống sử dụng giao tiếp an toàn, lớp này có thể cung cấp giao diện chuẩn hóa để thực hiện các thao tác xác thực (authentication) và mã hóa dữ liệu, đảm bảo tính bảo mật và tính toàn vẹn của dữ liệu.

#### ****Tóm Tắt****

* **Lớp Trừu tượng Phần cứng Mã hóa** cung cấp một cơ chế đồng nhất để truy cập các nguyên tắc mật mã, bất kể chúng được triển khai dưới dạng phần cứng nội (on-chip), phần cứng ngoại vi (off-chip), hay thư viện phần mềm.
* **Thuộc tính:**
  + **Triển khai**: Độc lập với vi điều khiển, phù hợp với nhiều loại thiết bị mật mã.
  + **Giao diện trên cùng**: Độc lập với vi điều khiển, phần cứng ECU và thiết bị mật mã, đảm bảo phần mềm ứng dụng có thể sử dụng các dịch vụ mã hóa mà không quan tâm đến chi tiết triển khai.
* **Ví dụ**: Thực hiện các thao tác mã hóa/giải mã AES, quản lý khóa mật mã, hoặc xác thực dữ liệu thông qua các giao diện chuẩn hóa.

Lớp này tạo điều kiện thuận lợi cho việc sử dụng các dịch vụ mã hóa trong các hệ thống ô tô, đảm bảo tính bảo mật, tính linh hoạt và khả năng tái sử dụng của phần mềm.



### Services: Crypto Services

**Dịch vụ Mã hóa (Crypto Services)** trong AUTOSAR bao gồm ba mô-đun chính, mỗi mô-đun có nhiệm vụ và chức năng riêng biệt như sau:

1. **Crypto Service Manager (Trình quản lý dịch vụ mã hóa)**:
   * **Nhiệm vụ**: Quản lý các công việc mã hóa (cryptographic jobs). Trình quản lý này chịu trách nhiệm tổ chức và điều phối các tác vụ mã hóa, đảm bảo các công việc được xử lý một cách hiệu quả và đúng đắn.
2. **Key Manager (Trình quản lý khóa)**:
   * **Nhiệm vụ**: Tương tác với hệ thống cung cấp khóa (key provisioning master), có thể lưu trữ khóa trong bộ nhớ không bay hơi (NVM) hoặc thông qua Crypto Driver. Trình quản lý khóa chịu trách nhiệm lưu trữ và xác minh các chuỗi chứng chỉ (certificate chains). Điều này rất quan trọng trong việc bảo mật các thông tin và xác thực giao dịch.
3. **Intrusion Detection System Manager (Trình quản lý hệ thống phát hiện xâm nhập)**:
   * **Nhiệm vụ**: Xử lý các sự kiện bảo mật được báo cáo từ các mô-đun phần mềm cơ sở (BSW) hoặc các SW-C (Software Component). Hệ thống này giúp phát hiện và phản ứng kịp thời với các cuộc tấn công bảo mật hoặc sự cố bảo mật trong hệ thống.

#### ****Nhiệm vụ chung****:

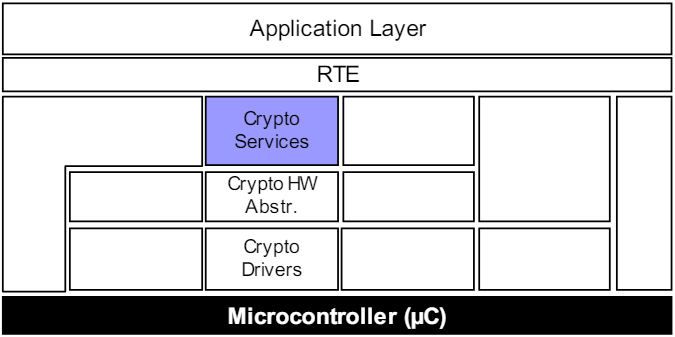
* Cung cấp các nguyên lý mã hóa (cryptographic primitives), dịch vụ phát hiện xâm nhập (IDS services), và lưu trữ khóa cho ứng dụng một cách đồng nhất.
* Trừu tượng hóa việc truy cập vào các thiết bị phần cứng và các tính chất phần cứng, giúp ứng dụng không bị phụ thuộc vào phần cứng cụ thể của vi điều khiển (µC) hoặc ECU.

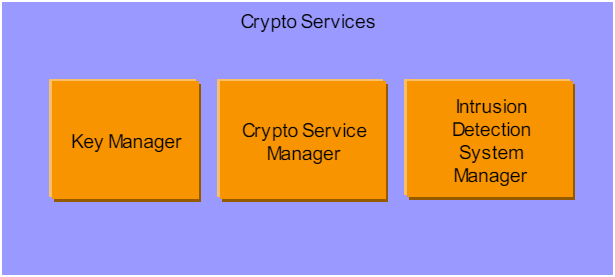
#### ****Thuộc tính****:

* **Triển khai**: Độc lập với phần cứng của vi điều khiển (µC) và ECU, có tính linh hoạt cao.
* **Giao diện trên cùng**: Độc lập với phần cứng vi điều khiển và ECU, giao diện được chỉ định và triển khai theo chuẩn AUTOSAR (giao diện AUTOSAR).

#### Tóm lại:

Dịch vụ mã hóa trong AUTOSAR cung cấp các cơ chế bảo mật mạnh mẽ cho hệ thống, bao gồm mã hóa dữ liệu, quản lý khóa, và phát hiện các cuộc tấn công hoặc sự xâm nhập bất hợp pháp. Điều này được thực hiện thông qua các mô-đun riêng biệt giúp ứng dụng có thể hoạt động một cách an toàn mà không cần phải phụ thuộc vào phần cứng cụ thể.





### Communication Services – General

**Dịch vụ Giao tiếp (Communication Services)** là một nhóm các mô-đun được sử dụng để giao tiếp mạng xe (CAN, LIN, FlexRay và Ethernet). Chúng giao tiếp với các trình điều khiển giao tiếp thông qua lớp trừu tượng phần cứng giao tiếp.

#### ****Nhiệm vụ****:

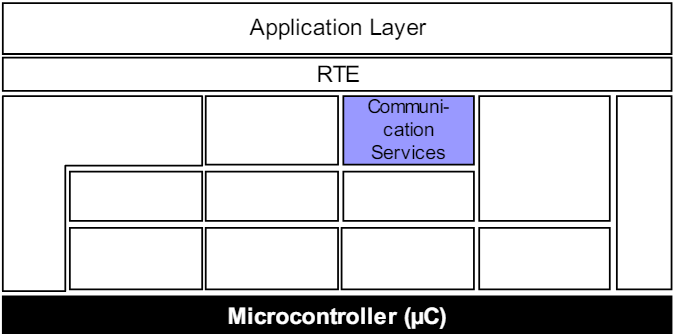
1. **Cung cấp giao diện đồng nhất cho mạng xe**:
   * Dịch vụ giao tiếp này đảm bảo các ứng dụng có thể giao tiếp với mạng xe mà không cần quan tâm đến các chi tiết kỹ thuật của từng loại mạng cụ thể (CAN, LIN, FlexRay, Ethernet).
2. **Cung cấp các dịch vụ quản lý mạng đồng nhất**:
   * Dịch vụ quản lý mạng giúp giám sát, kiểm soát và duy trì các trạng thái của mạng xe, bao gồm việc quản lý các tài nguyên mạng và bảo đảm rằng các hệ thống có thể giao tiếp một cách hiệu quả và an toàn.
3. **Cung cấp giao diện đồng nhất cho giao tiếp chẩn đoán**:
   * Dịch vụ giao tiếp chẩn đoán giúp các thiết bị trong xe có thể trao đổi thông tin chẩn đoán và báo cáo sự cố về hệ thống thông qua các giao thức như UDS (Unified Diagnostic Services).
4. **Ẩn các thuộc tính của giao thức và tin nhắn khỏi ứng dụng**:
   * Giao diện này giúp trừu tượng hóa các chi tiết của giao thức (như cách thức truyền tin) và các thông điệp giữa các thiết bị, làm cho ứng dụng không cần phải xử lý các vấn đề phức tạp này.

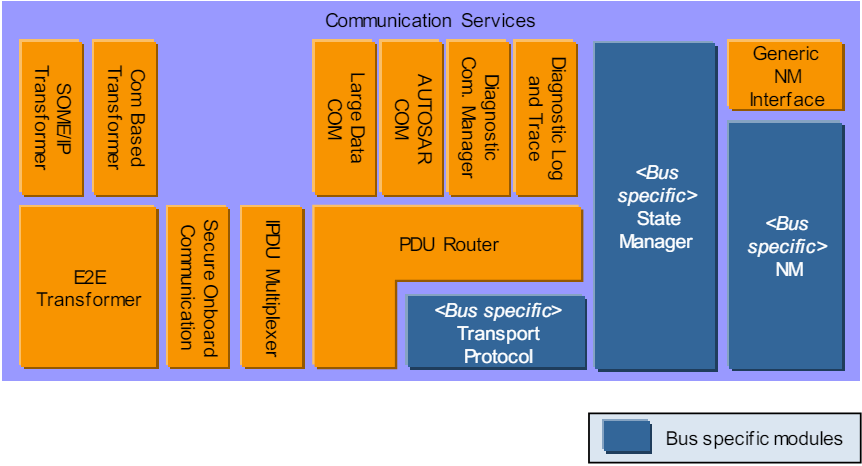
#### ****Thuộc tính****:

* **Triển khai**: Độc lập với phần cứng của vi điều khiển (µC) và ECU, tuy nhiên có sự phụ thuộc vào loại bus (CAN, LIN, FlexRay, Ethernet) vì mỗi bus sẽ có những yêu cầu và cách thức giao tiếp riêng biệt.
* **Giao diện trên cùng**: Độc lập với phần cứng của vi điều khiển (µC), phần cứng ECU và loại bus, giúp đơn giản hóa việc phát triển và triển khai giao tiếp mạng.

#### ****Tóm tắt****:

Dịch vụ giao tiếp trong AUTOSAR cung cấp các giải pháp chuẩn hóa cho việc giao tiếp trong mạng xe, giúp các hệ thống có thể giao tiếp và quản lý mạng một cách đồng nhất và hiệu quả mà không cần phải quan tâm đến các chi tiết phức tạp của từng giao thức hoặc cấu trúc tin nhắn cụ thể. Các dịch vụ này đóng vai trò quan trọng trong việc quản lý mạng và hỗ trợ giao tiếp chẩn đoán, giúp hệ thống dễ dàng bảo trì và mở rộng.





### Communication Stack – CAN

**Dịch vụ Giao tiếp CAN (CAN Communication Services)** là một nhóm các mô-đun dành cho giao tiếp mạng xe với hệ thống giao tiếp CAN (Controller Area Network).

#### ****Nhiệm vụ****:

1. **Cung cấp giao diện đồng nhất cho mạng CAN**:
   * Dịch vụ giao tiếp này giúp tạo ra một giao diện chuẩn để các ứng dụng có thể giao tiếp với mạng CAN mà không cần phải hiểu và xử lý các chi tiết phức tạp của giao thức CAN.
2. **Ẩn các thuộc tính giao thức và thông điệp khỏi ứng dụng**:
   * Dịch vụ này ẩn đi các chi tiết liên quan đến giao thức và các thông điệp, giúp các ứng dụng chỉ tập trung vào việc giao tiếp mà không phải lo lắng về cách thức truyền tải và mã hóa thông điệp.

#### ****Hỗ trợ trong CAN Communication Stack****:

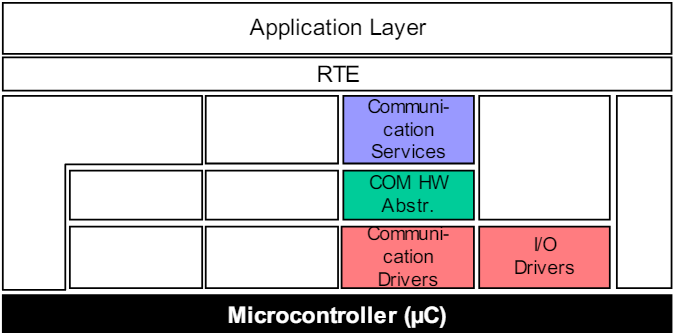
1. **Giao tiếp CAN truyền thống (CAN 2.0)**:
   * Hỗ trợ các giao thức CAN 2.0 truyền thống, vốn là tiêu chuẩn lâu đời nhất và phổ biến trong các hệ thống CAN.
2. **Giao tiếp CAN FD (Flexible Data Rate)**:
   * CAN FD cho phép truyền tải dữ liệu lớn hơn trong một chu kỳ truyền, giúp tăng hiệu suất và tốc độ truyền tải khi phần cứng hỗ trợ.
3. **Giao tiếp CAN XL**:
   * CAN XL là một phiên bản mới hơn của CAN, được thiết kế để hỗ trợ tốc độ truyền cao hơn và băng thông lớn hơn, nhưng chỉ hoạt động nếu phần cứng hỗ trợ.

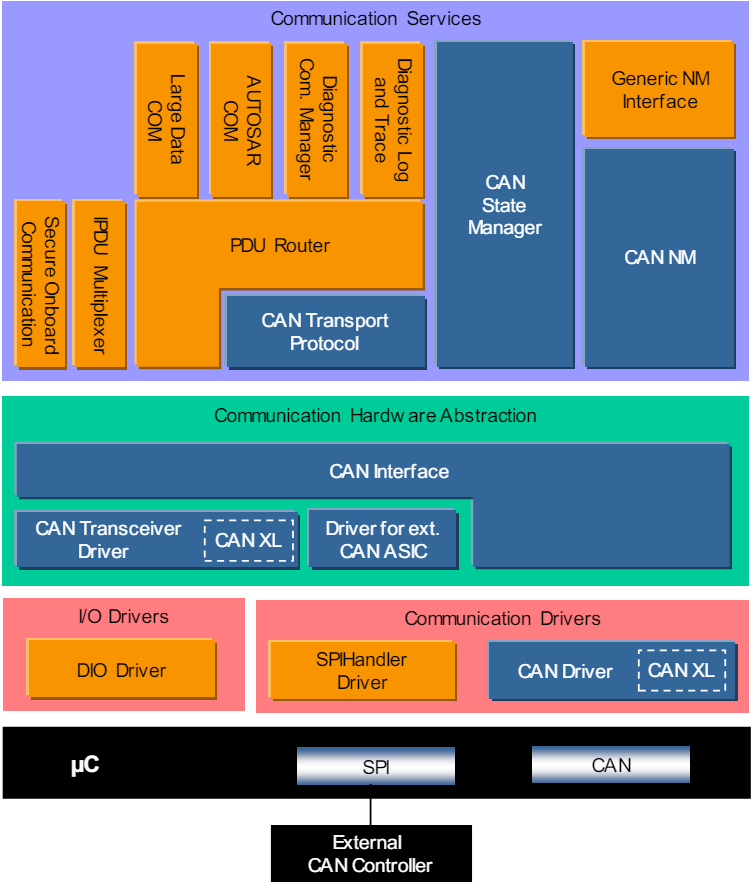
#### ****Thuộc tính****:

* **Triển khai**: Độc lập với phần cứng vi điều khiển (µC) và ECU, tuy nhiên có sự phụ thuộc một phần vào hệ thống CAN.
* **Các thành phần chung cho tất cả các hệ thống mạng xe**:
  + **AUTOSAR COM**: Cung cấp giao diện cơ bản cho giao tiếp giữa các mô-đun phần mềm trong ECU.
  + **Generic NM (Network Management)**: Quản lý mạng chung cho các hệ thống mạng xe. Chỉ có một instance (thể hiện) trên mỗi ECU và nó sẽ bao gồm một dispatcher (chuyển tiếp) cho việc điều phối các thông điệp.
  + **Diagnostic Communication Manager**: Cung cấp dịch vụ giao tiếp chẩn đoán trong hệ thống.
* **Giao diện Generic NM** chỉ bao gồm một dispatcher, không có chức năng bổ sung nào. Tuy nhiên, trong trường hợp các ECU gateway, nó cũng có thể bao gồm chức năng NM coordinator, cho phép đồng bộ nhiều mạng khác nhau (cùng loại hoặc khác loại) để đồng bộ hóa việc đánh thức hoặc tắt chúng.
* **CAN NM (Network Management)** là một phần đặc thù của mạng CAN và sẽ được khởi tạo riêng biệt cho mỗi mạng CAN trong xe.
* **CAN State Manager**: Mô-đun này chịu trách nhiệm quản lý trạng thái hệ thống CAN, bao gồm các tính năng như khởi động và tắt hệ thống. Nó cũng kiểm soát các tùy chọn giao tiếp trong COM để gửi PDUs (Protocol Data Units) và giám sát các sự cố về thời gian chờ tín hiệu (signal timeouts).

#### ****Tóm tắt****:

Dịch vụ Giao tiếp CAN trong AUTOSAR cung cấp các giải pháp chuẩn hóa và đồng nhất cho việc giao tiếp trong mạng CAN của xe, bao gồm CAN 2.0, CAN FD, và CAN XL (nếu phần cứng hỗ trợ). Các dịch vụ này giúp ứng dụng giao tiếp hiệu quả mà không cần quan tâm đến chi tiết của giao thức và thông điệp. Hệ thống mạng được quản lý bởi các mô-đun như NM (Network Management) và Can State Manager, đảm bảo rằng giao tiếp mạng được thực hiện một cách đáng tin cậy và đồng bộ trong toàn bộ mạng xe.







### Communication Stack – Ethernet/CAN XL

**Dịch vụ Giao tiếp CAN XL (CAN XL Communication Services)** mở rộng khả năng của giao thức CAN truyền thống bằng cách hỗ trợ trực tiếp việc truyền tải các frame Ethernet IEEE 802.3, cho phép tham gia giao tiếp IP (Internet Protocol). Điều này giúp mở rộng khả năng kết nối mạng xe, đồng thời giữ nguyên các nguyên lý cơ bản của CAN trong khi hỗ trợ các chuẩn giao tiếp hiện đại như Ethernet.

#### ****Nhiệm vụ****:

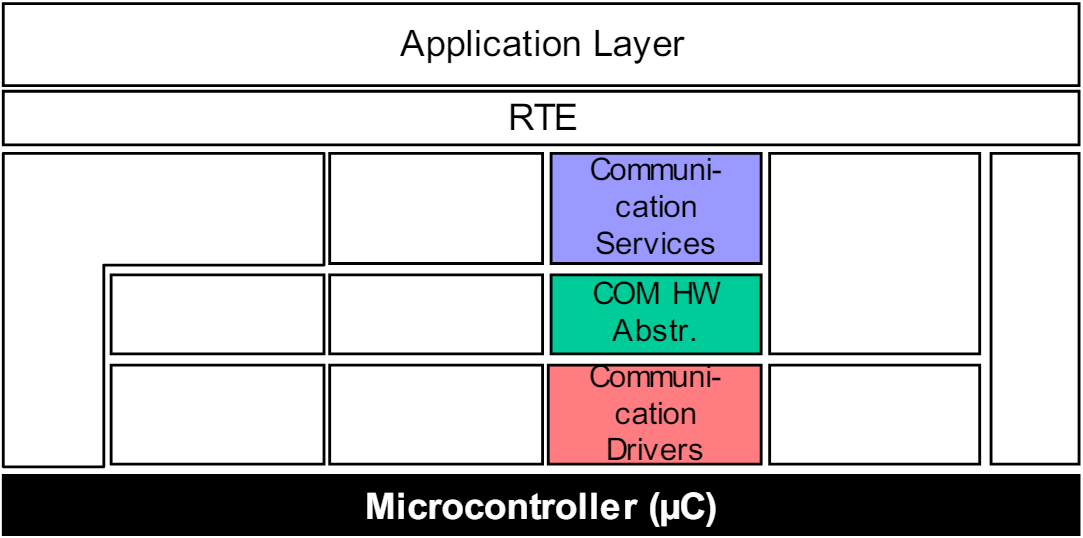
1. **Cung cấp giao tiếp rộng khắp xe** với cùng một ý nghĩa (semantic) sử dụng ở mọi nơi, bất kể kết nối vật lý (CAN XL / Ethernet) hay phương thức giao tiếp (dựa trên tín hiệu hoặc dịch vụ).
   * Mục tiêu là cung cấp một giao diện mạng đồng nhất và dễ hiểu cho các ứng dụng trong xe, cho dù chúng sử dụng CAN XL, Ethernet, hay bất kỳ dạng giao tiếp nào khác.
2. **Tunneling IEEE 802.3 Ethernet frames**:
   * CAN XL hỗ trợ việc đóng gói các frame Ethernet IEEE 802.3 trực tiếp vào các gói CAN, cho phép giao tiếp IP mà không cần phải chuyển sang các giao thức mạng khác, tạo sự tương thích cao giữa CAN và Ethernet.

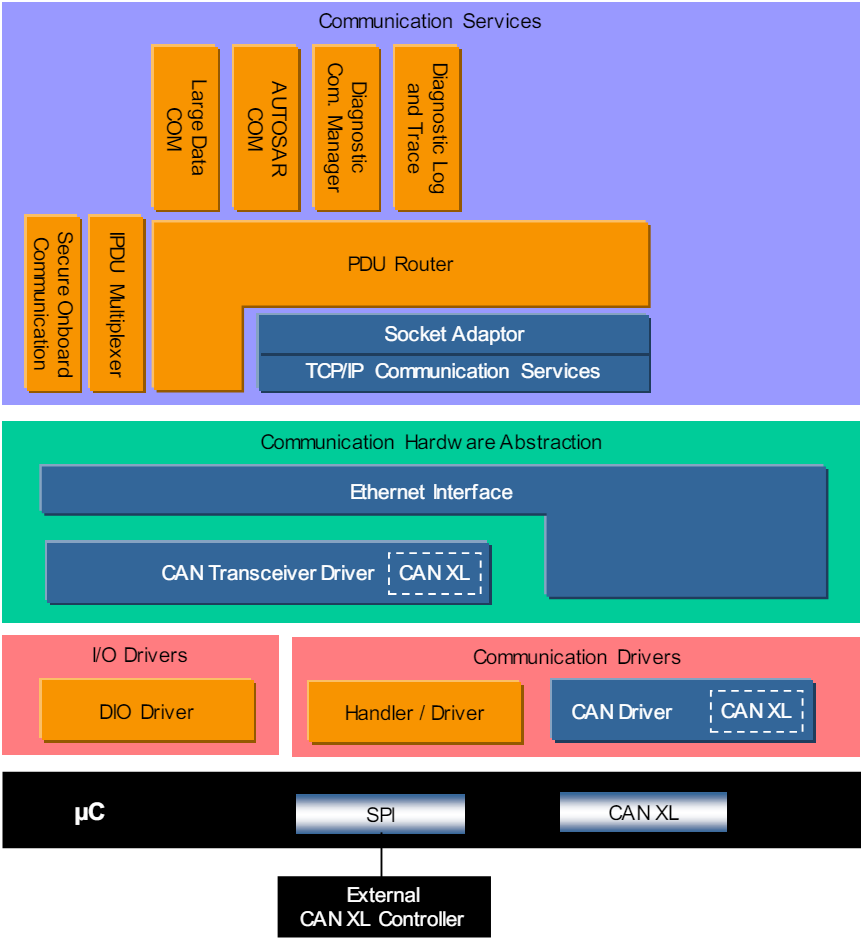
#### ****Thuộc tính****:

1. **CAN XL là một superset của CAN**:
   * Điều này có nghĩa là bất kỳ hệ thống CAN nào hỗ trợ CAN XL có thể phục vụ cả mạng CAN thông thường và mạng CAN XL. Tức là, CAN XL không chỉ là một giao thức mở rộng mà còn là một sự thay thế linh hoạt cho hệ thống CAN truyền thống.
2. **Các mô-đun cần được mở rộng để hỗ trợ CAN XL**:
   * **CanIf (CAN Interface)**, **CanTrcvDrv (CAN Transceiver Driver)** và **CanDrv (CAN Driver)** là các mô-đun cần được mở rộng để hỗ trợ giao tiếp CAN XL.
   * Việc mở rộng các mô-đun này cho phép chúng tương thích với các khái niệm mới của CAN XL, bao gồm việc truyền tải frame Ethernet và hỗ trợ giao tiếp IP.
3. **Tính năng của giao tiếp CAN** cũng được duy trì trong CAN XL:
   * Mọi tính năng của giao tiếp CAN truyền thống (CAN 2.0 và CAN FD) cũng được duy trì khi sử dụng CAN XL. Điều này bao gồm việc hỗ trợ các kiểu truyền tải dữ liệu, kiểm soát lỗi và các quy trình giao tiếp khác mà CAN đã phát triển qua nhiều năm.

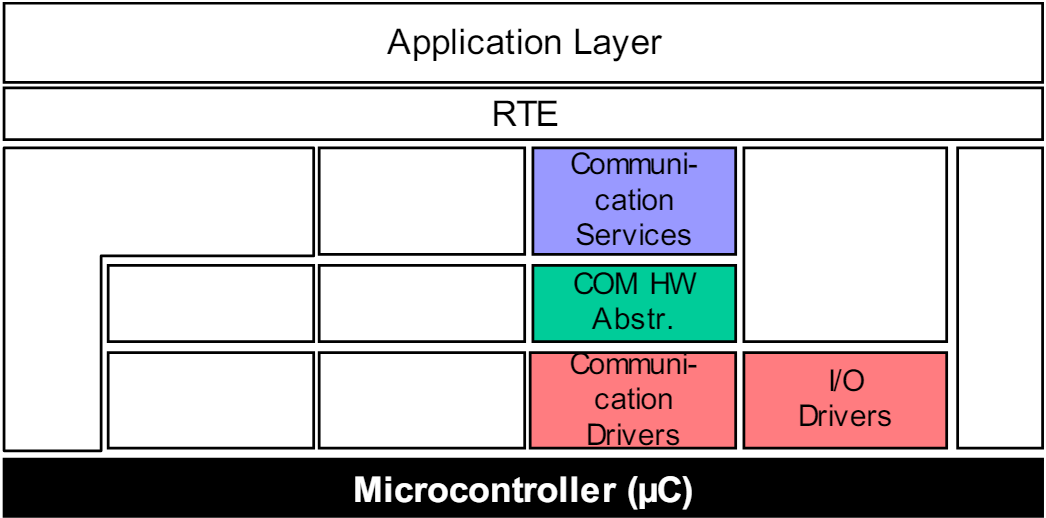
#### ****Tóm tắt****:

CAN XL là một sự mở rộng mạnh mẽ của giao thức CAN, cho phép tích hợp giao tiếp Ethernet và IP vào hệ thống mạng xe, tạo ra một hệ thống mạng xe linh hoạt hơn. Bằng cách hỗ trợ việc đóng gói trực tiếp các frame Ethernet, CAN XL mang lại sự liên kết mạnh mẽ giữa các hệ thống CAN và Ethernet, đồng thời duy trì khả năng giao tiếp của CAN trong các môi trường phức tạp. Các mô-đun giao tiếp chính của CAN như **CanIf**, **CanTrcvDrv**, và **CanDrv** cần được mở rộng để hỗ trợ các tính năng này, nhưng các thuộc tính và tính năng cơ bản của giao thức CAN vẫn giữ nguyên khi triển khai CAN XL.





### Communication Stack Extension – CAN XL



### Communication Stack Extension – TTCAN

**Dịch vụ Giao tiếp TTCAN (TTCAN Communication Services)** là các mở rộng tùy chọn cho mô-đun **CAN Interface** và **CAN Driver** trong hệ thống giao tiếp mạng xe với giao thức **TTCAN**. TTCAN (Time-Triggered CAN) là một phiên bản mở rộng của giao thức CAN, được thiết kế để cung cấp khả năng giao tiếp chính xác theo thời gian, điều này rất quan trọng trong các ứng dụng xe tự lái hoặc hệ thống điều khiển an toàn.

#### ****Nhiệm vụ****:

1. **Cung cấp giao diện đồng nhất cho mạng TTCAN**:
   * Mục tiêu chính của TTCAN Communication Services là cung cấp một giao diện mạng đồng nhất và dễ sử dụng cho các ứng dụng mà không cần phải quan tâm đến các chi tiết phức tạp của giao thức TTCAN. Điều này bao gồm việc **ẩn các thuộc tính giao thức và tin nhắn** để các ứng dụng chỉ cần tương tác với giao diện đơn giản mà không cần hiểu sâu về hoạt động bên trong của TTCAN.

#### ****Thông tin cần lưu ý****:

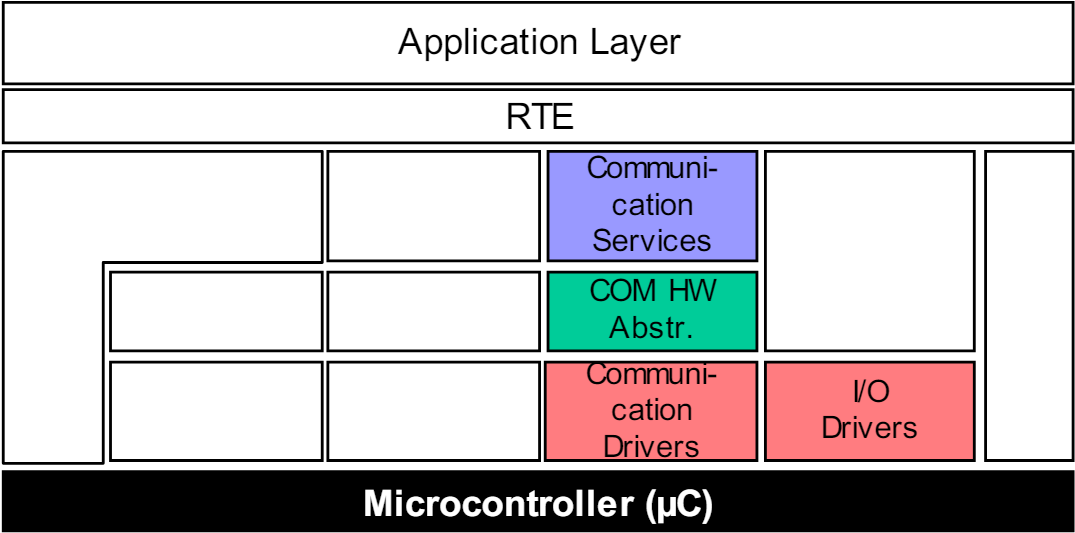
1. **CAN Interface với TTCAN**:
   * **CAN Interface** có thể phục vụ cả **CAN Driver** thông thường và **CAN Driver TTCAN**, điều này có nghĩa là một phần mềm có thể hoạt động với cả hai loại giao thức mà không cần thay đổi nhiều trong phần mềm.

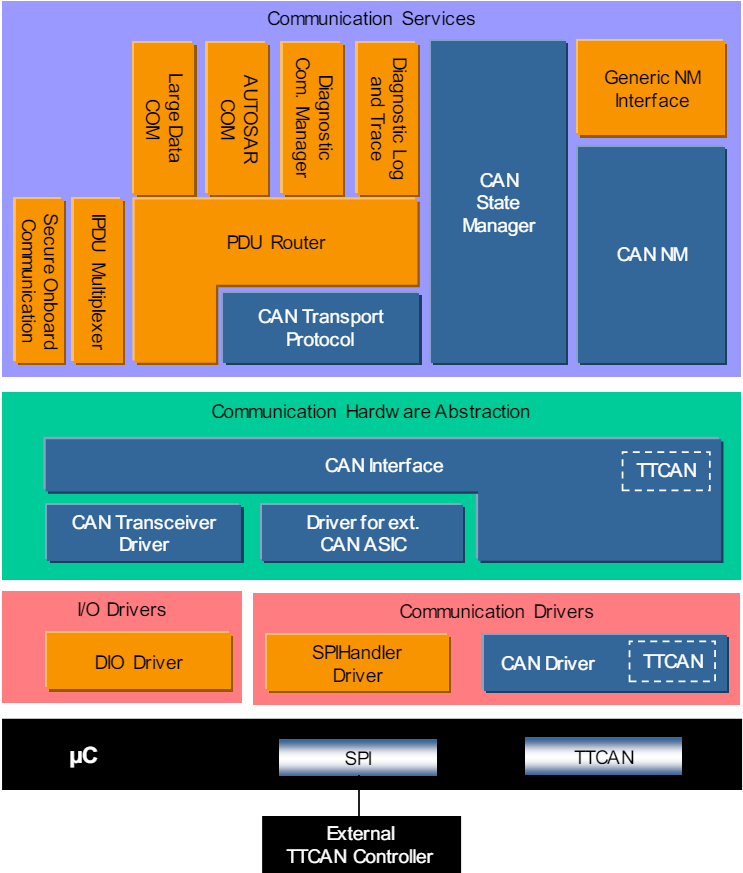
#### ****Thuộc tính****:

1. **TTCAN là một superset hoàn toàn của CAN**:
   * Điều này có nghĩa là TTCAN bao gồm tất cả các tính năng của CAN và bổ sung thêm các tính năng mới để hỗ trợ các ứng dụng yêu cầu giao tiếp chính xác về thời gian. Một hệ thống giao tiếp hỗ trợ TTCAN có thể làm việc với cả mạng CAN và TTCAN, mà không gặp phải các vấn đề tương thích.
2. **Các mô-đun cần mở rộng để hỗ trợ TTCAN**:
   * **CanIf** và **CanDrv** là những mô-đun cần được mở rộng để hỗ trợ giao tiếp TTCAN. Điều này cho phép hệ thống giao tiếp CAN truyền thống cũng có thể hỗ trợ các tính năng đặc biệt của TTCAN như đồng bộ hóa thời gian, thời gian kích hoạt chính xác, và ưu tiên các tin nhắn theo thời gian.
3. **Các thuộc tính của giao tiếp CAN cũng đúng với CAN có tính năng TTCAN**:
   * Các tính năng cơ bản của giao tiếp CAN, chẳng hạn như việc truyền tải dữ liệu, xử lý lỗi, và các giao thức quản lý mạng (Network Management) vẫn được duy trì trong hệ thống TTCAN. Điều này giúp bảo đảm rằng TTCAN sẽ hoạt động theo cách tương tự CAN, nhưng với những tính năng bổ sung mạnh mẽ hơn.

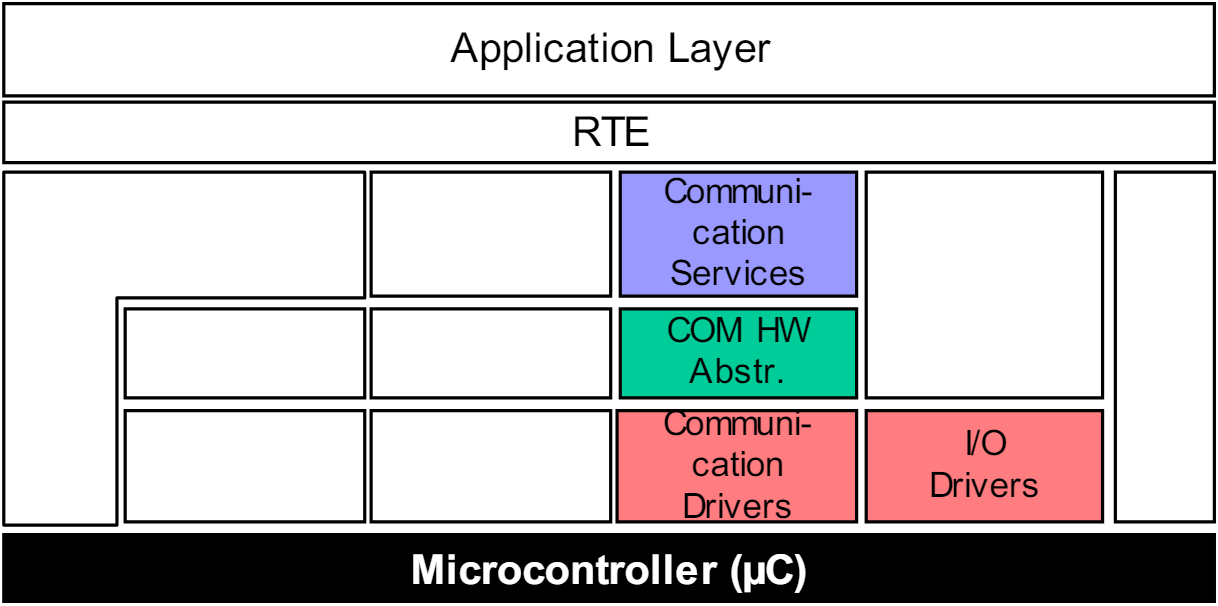
#### ****Tóm tắt****:

TTCAN là một sự mở rộng mạnh mẽ của giao thức CAN, đặc biệt hữu ích cho các ứng dụng yêu cầu đồng bộ hóa và thời gian chính xác. Bằng cách hỗ trợ các tính năng như "Time-Triggered" communication, TTCAN mang lại khả năng quản lý thời gian và thứ tự tin nhắn vượt trội trong các môi trường phức tạp, ví dụ như hệ thống xe tự lái, giao tiếp an toàn trong xe, hoặc các ứng dụng xe cần xử lý thời gian thực chính xác. Các mô-đun **CanIf** và **CanDrv** cần được mở rộng để hỗ trợ TTCAN, nhưng các đặc tính cơ bản của giao tiếp CAN vẫn giữ nguyên, giúp hệ thống có khả năng tương thích ngược với CAN.

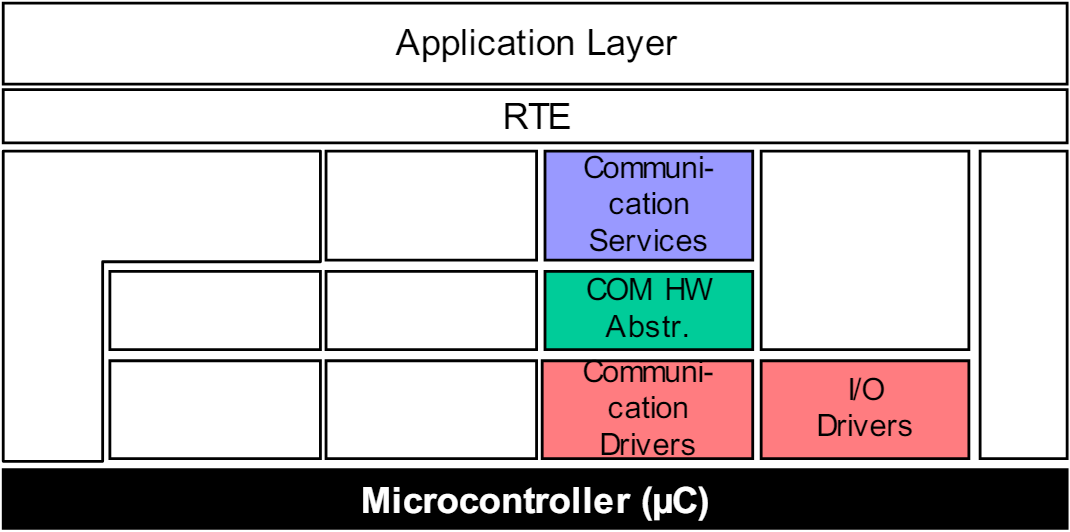


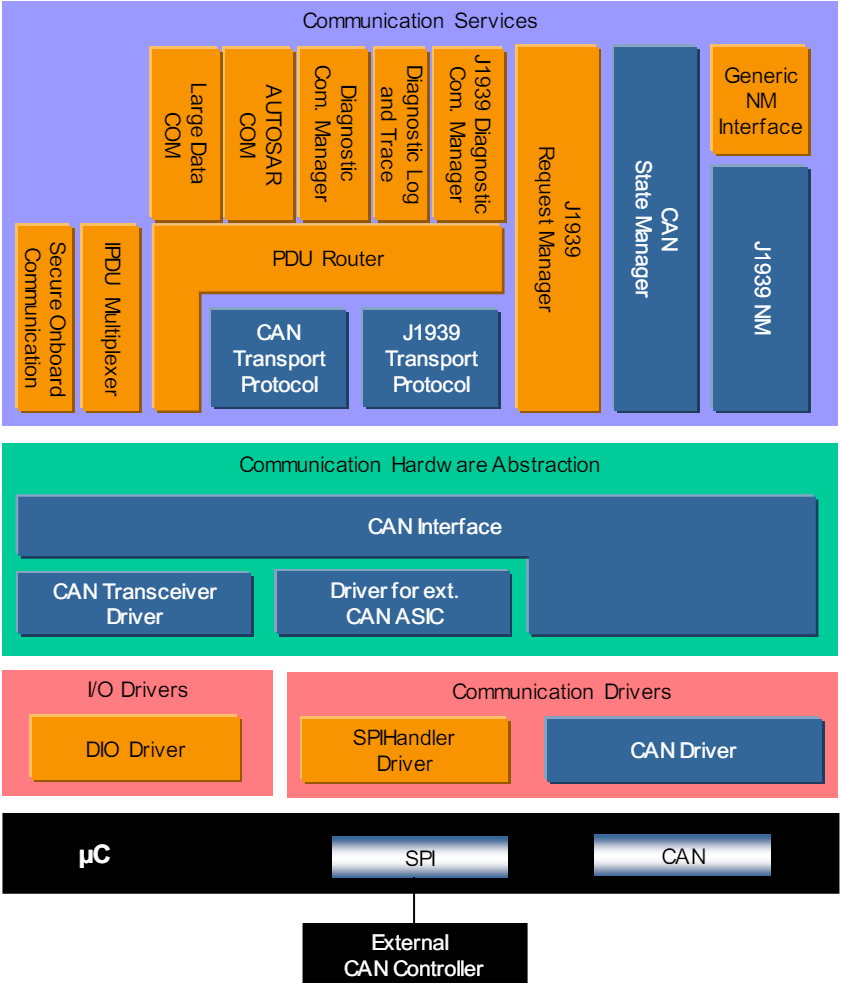


### Communication Stack Extension – TTCAN

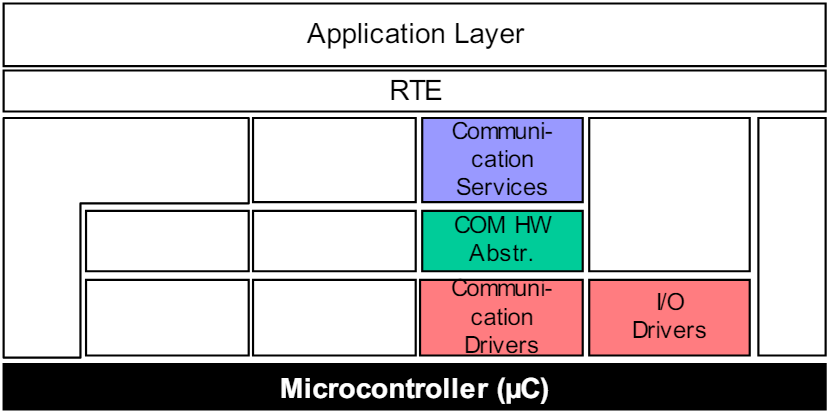


### Communication Stack Extension – J1939

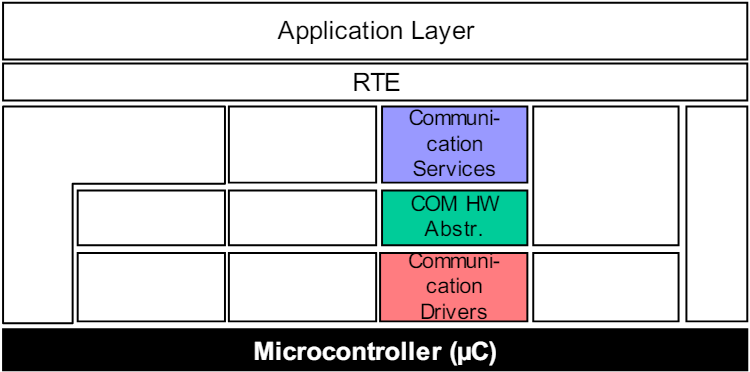


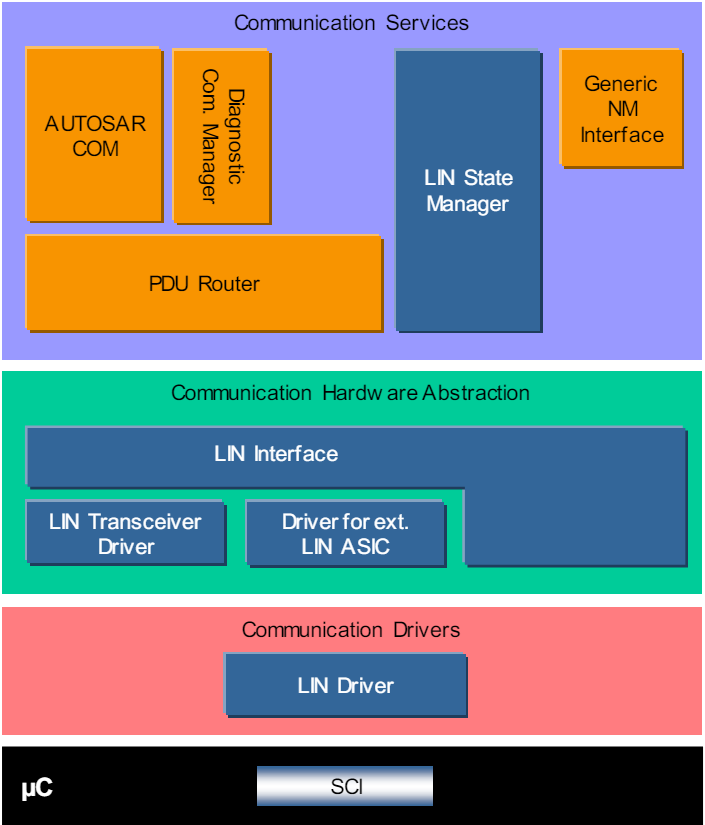


### Communication Stack Extension – J1939

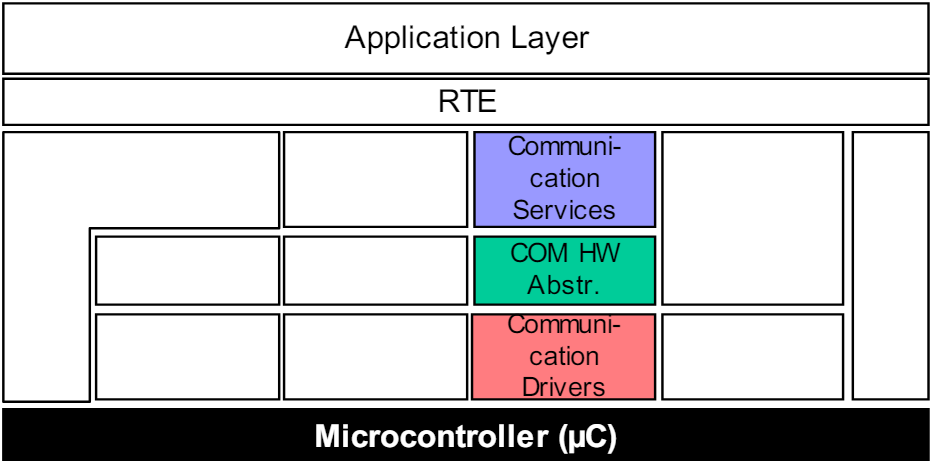


### Communication Stack – LIN

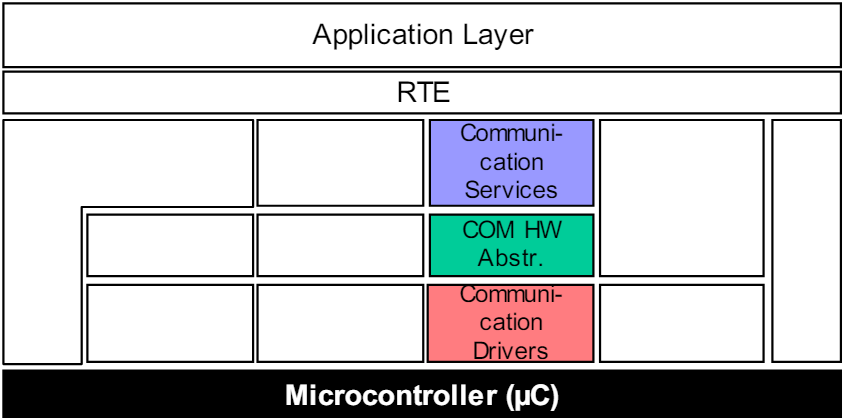


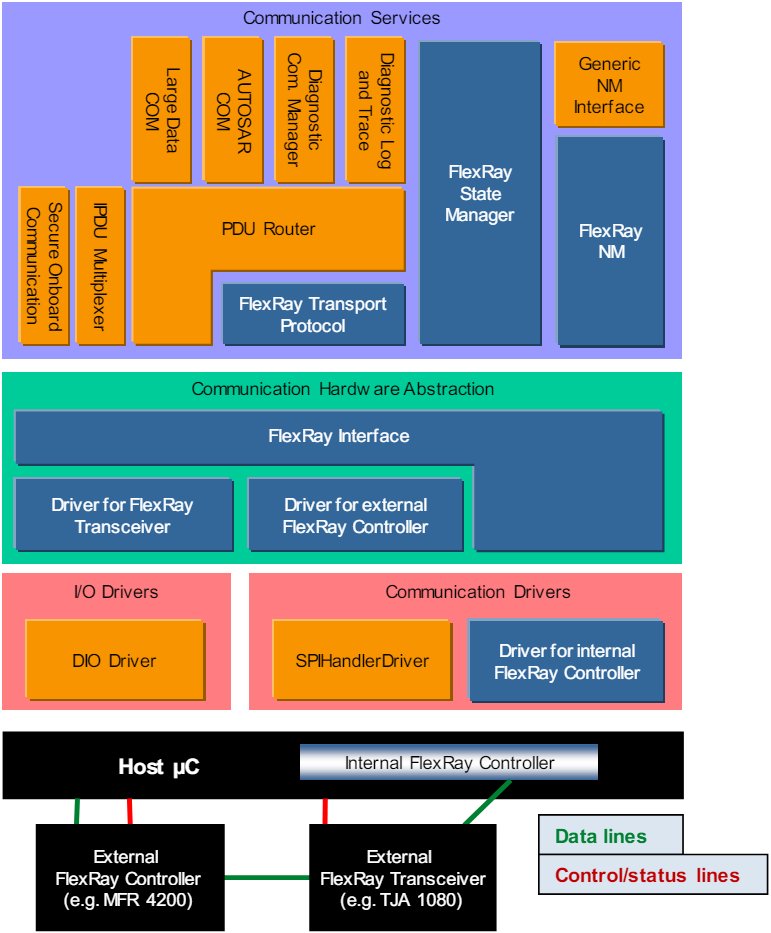


### Communication Stack – LIN

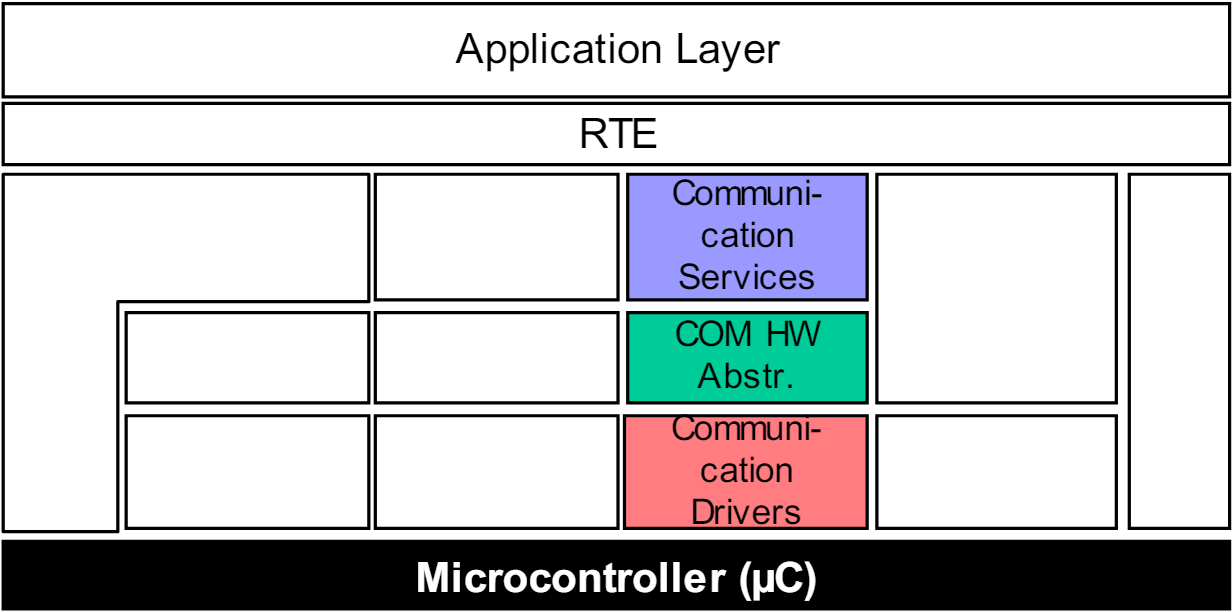


### Communication Stack – FlexRay



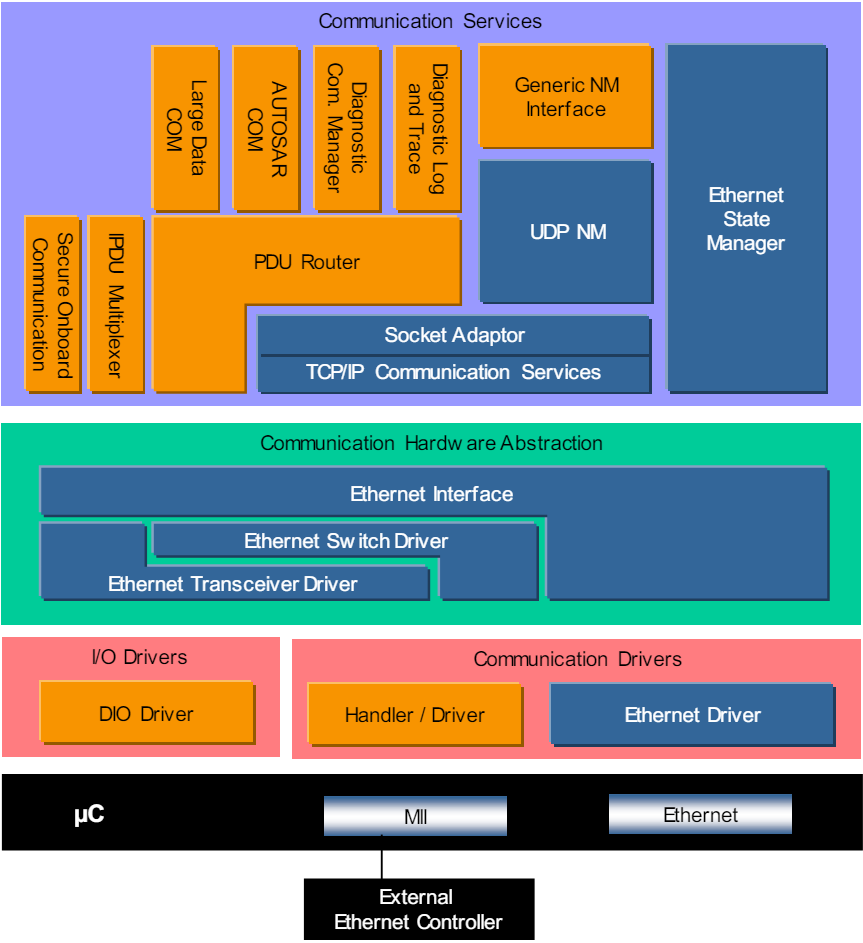


### Communication Stack – FlexRay

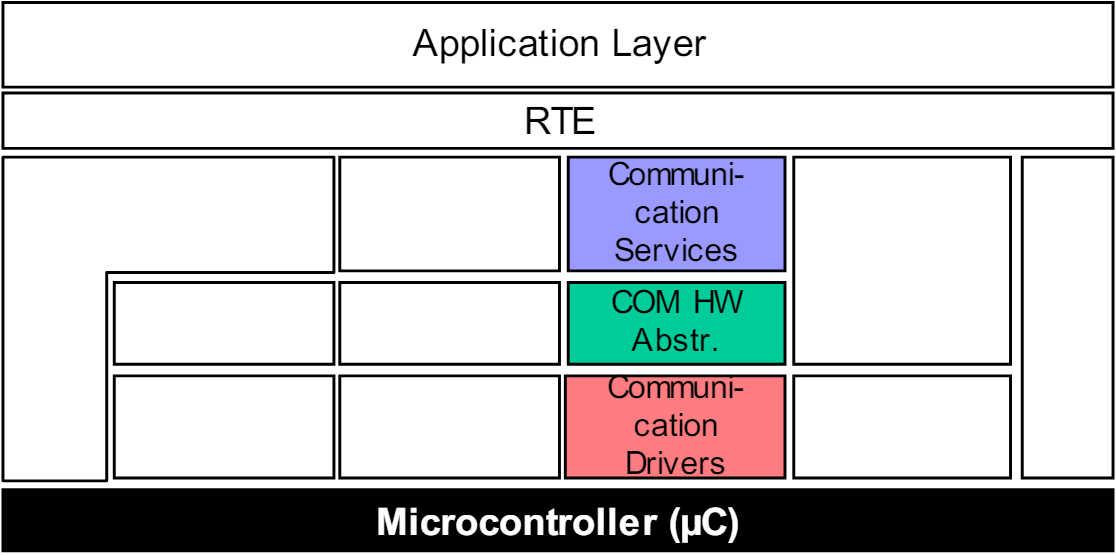


### Communication Stack – TCP/IP

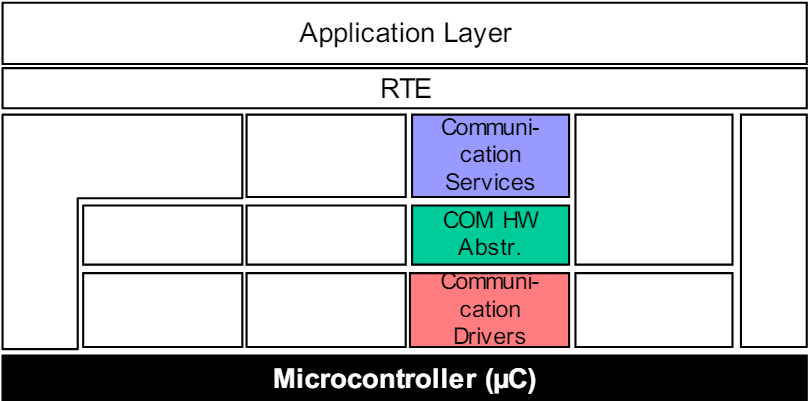


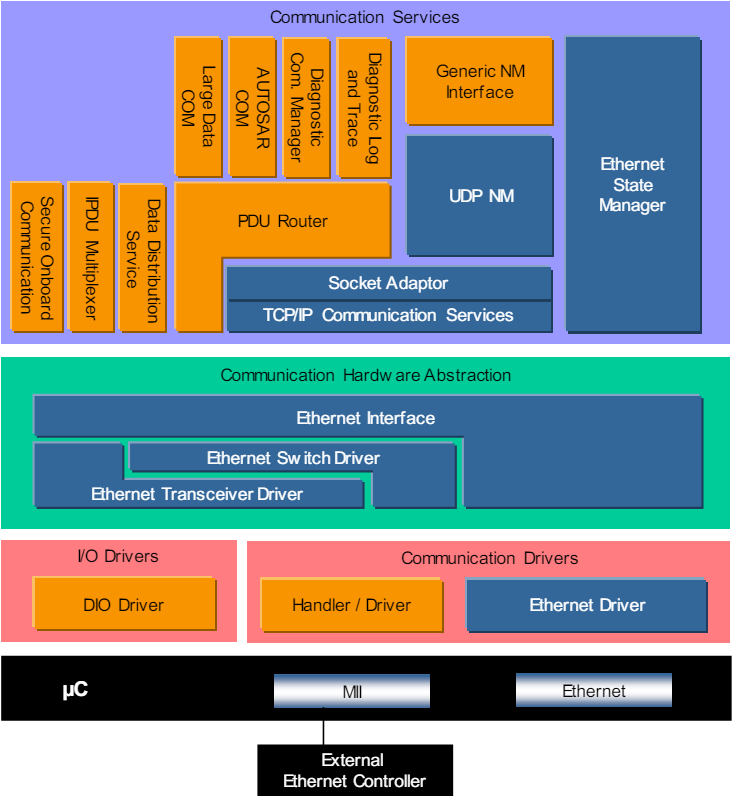


### Communication Stack – TCP/IP

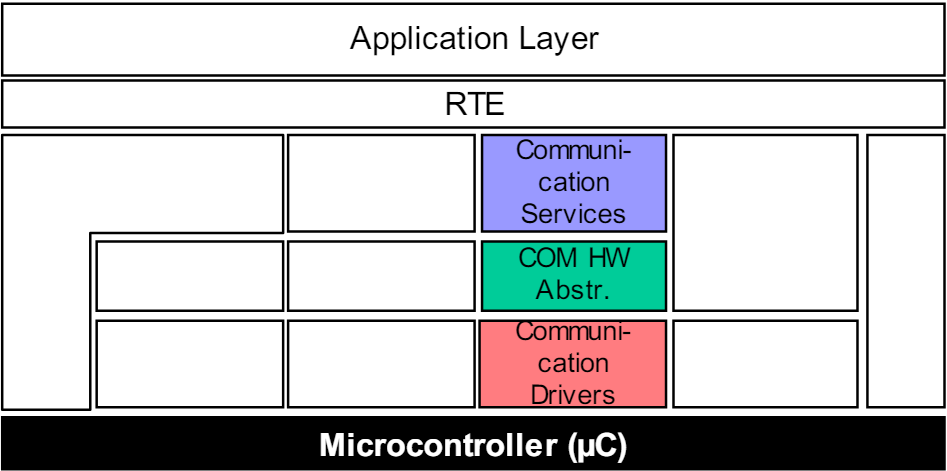


### Communication Stack – DDS

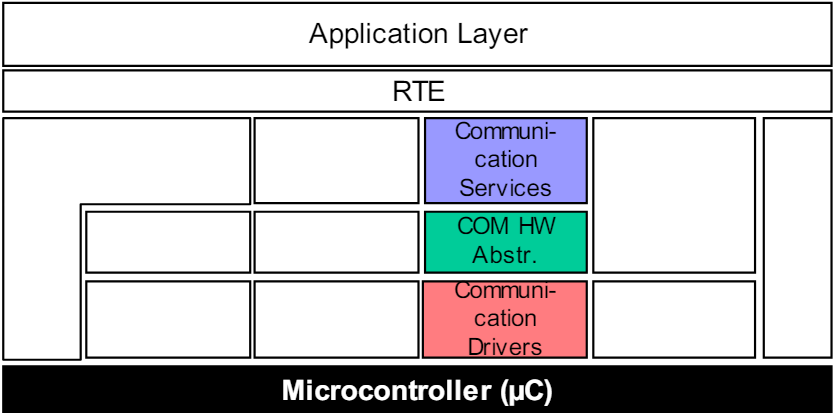




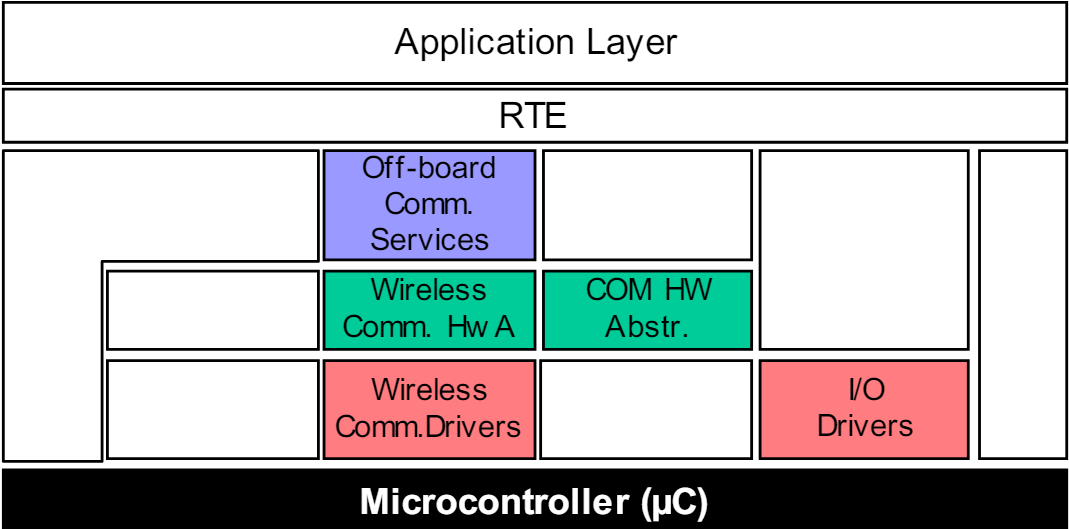
### Communication Stack – DDS

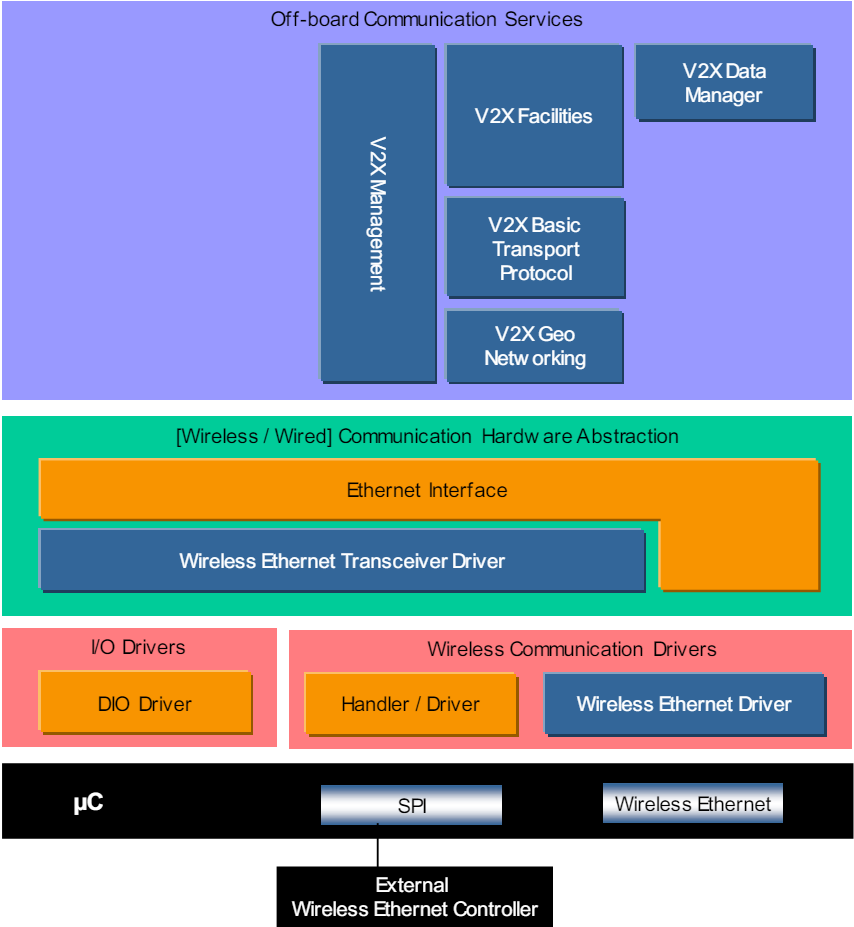


### Communication Stack – General

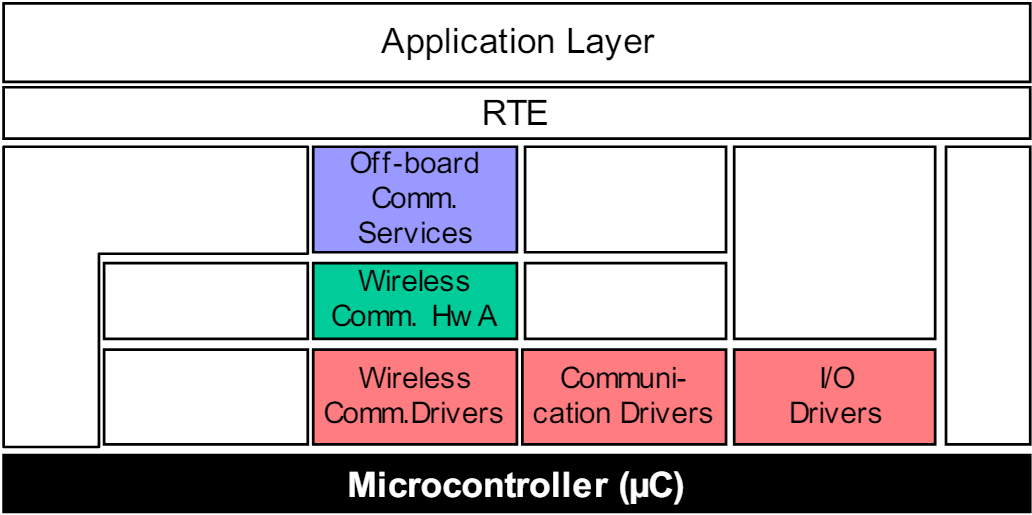


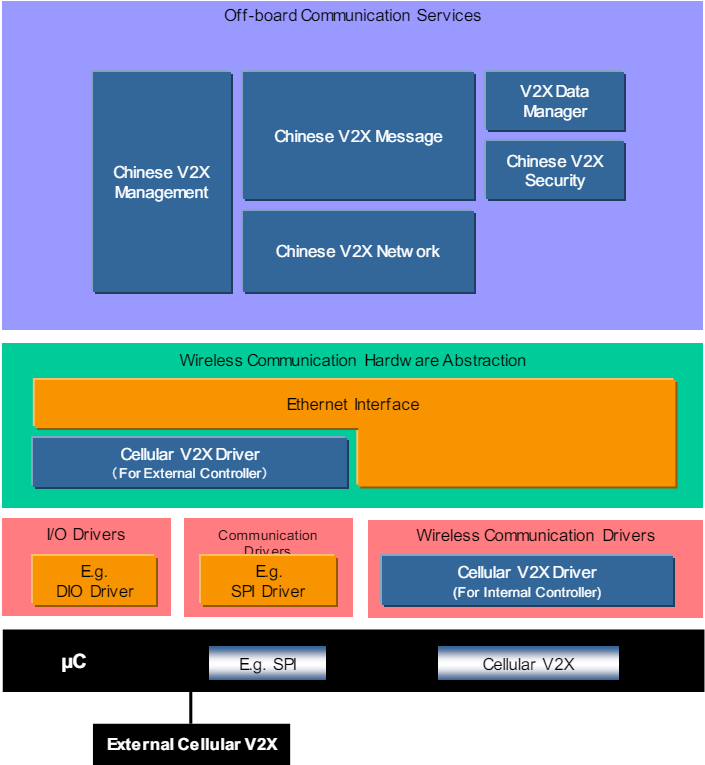
### Off-board Communication Stack – European Vehicle-2-X



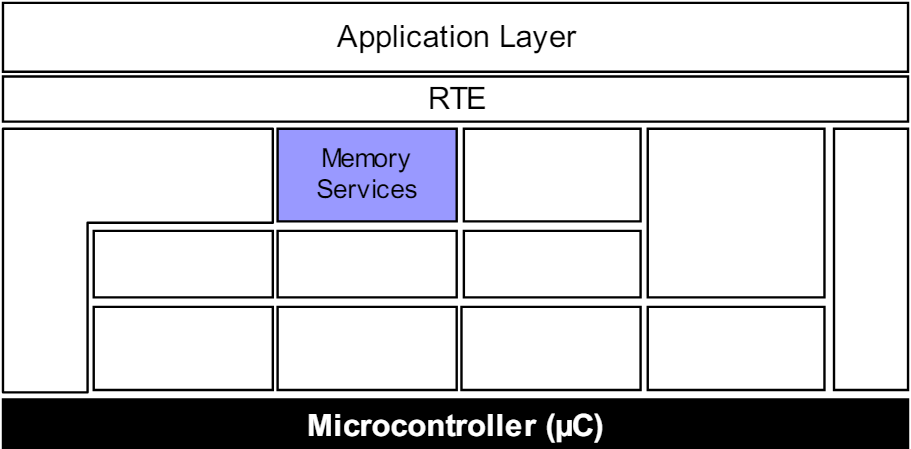


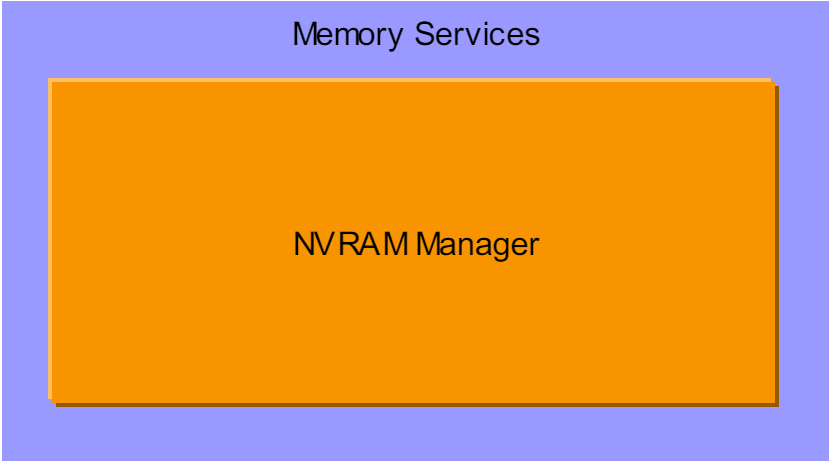
### Off-board Communication Stack – Chinese Vehicle-2-X



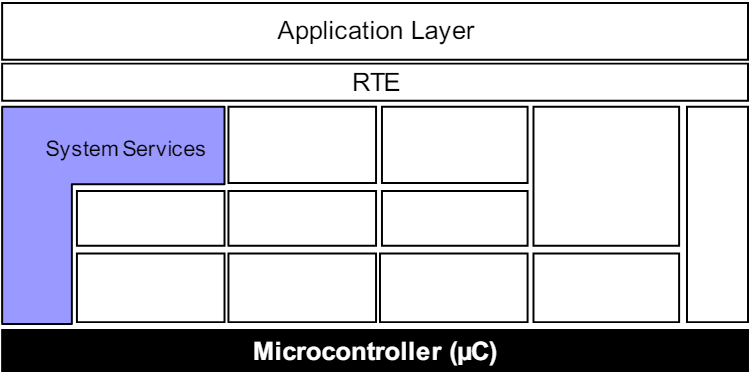


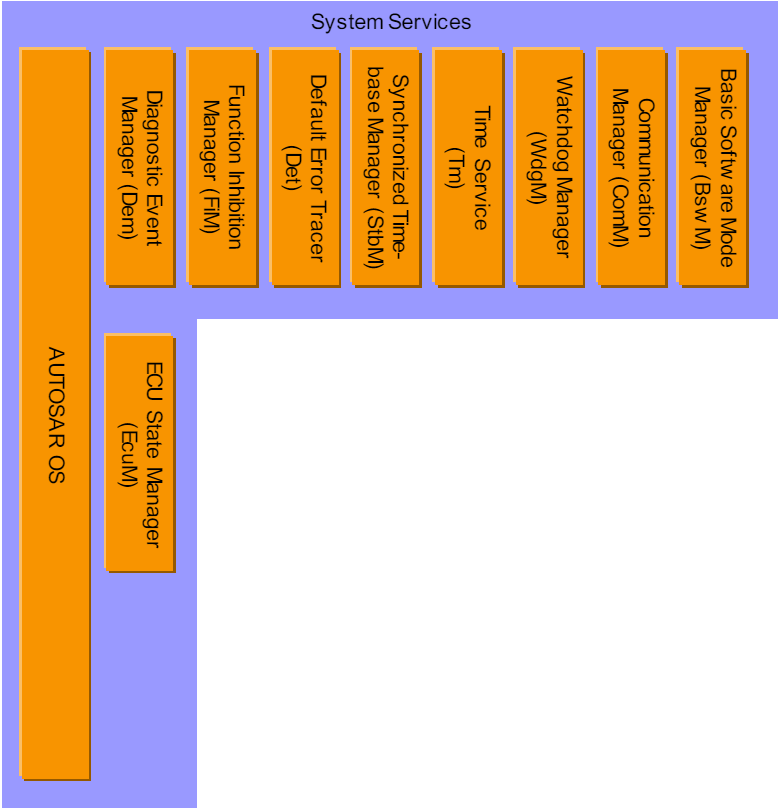
### Services: Memory Services



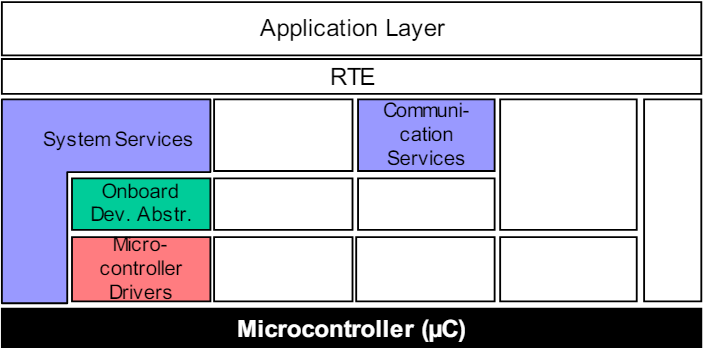


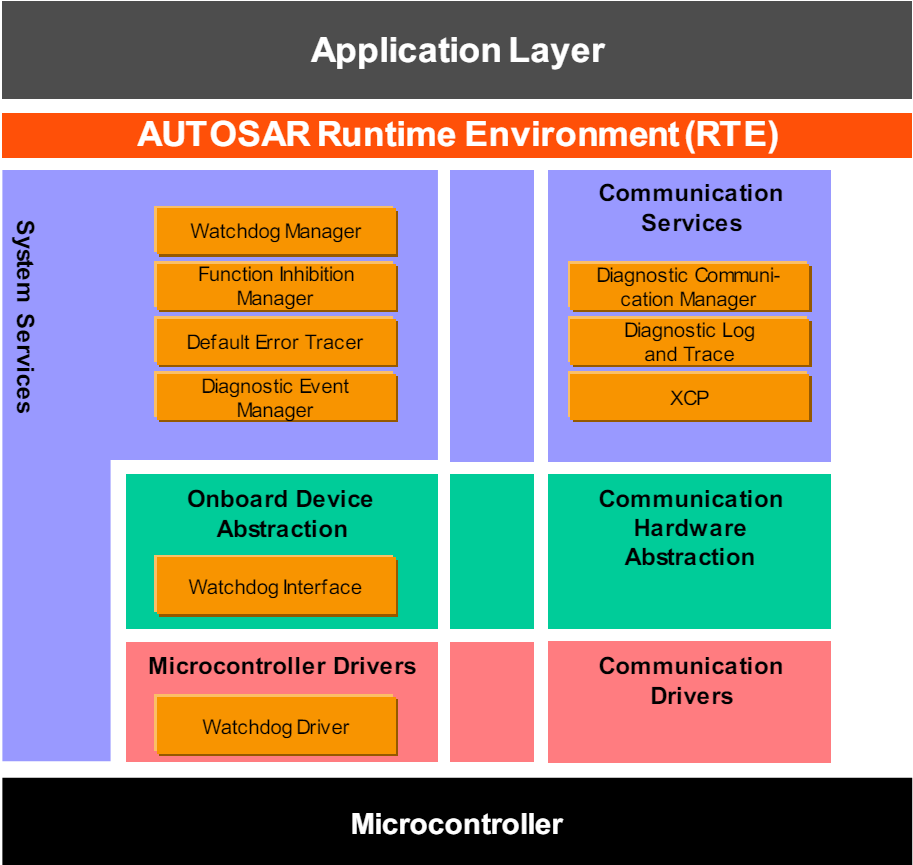
### Services: System Services



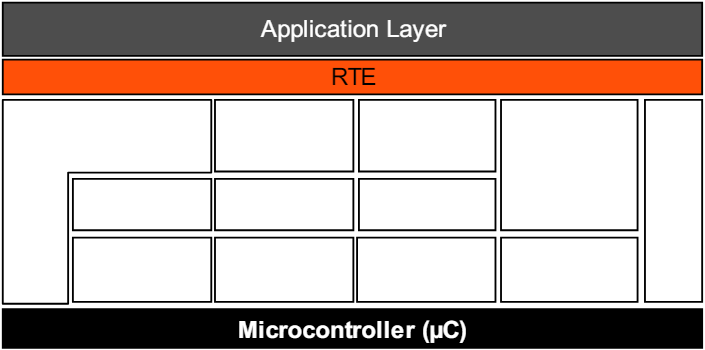


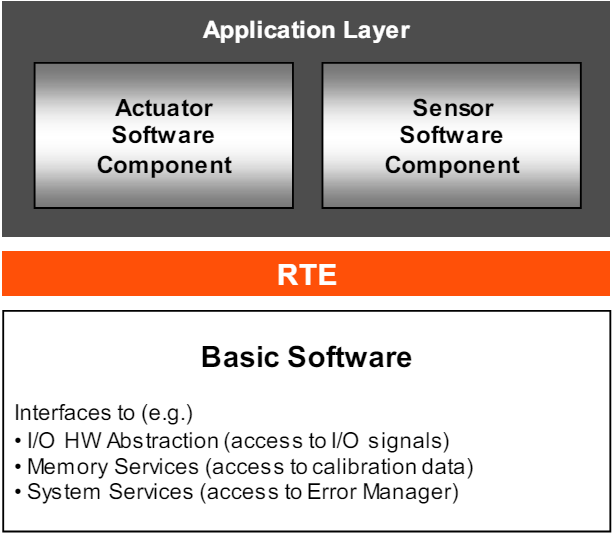
### Error Handling, Reporting and Diagnostic





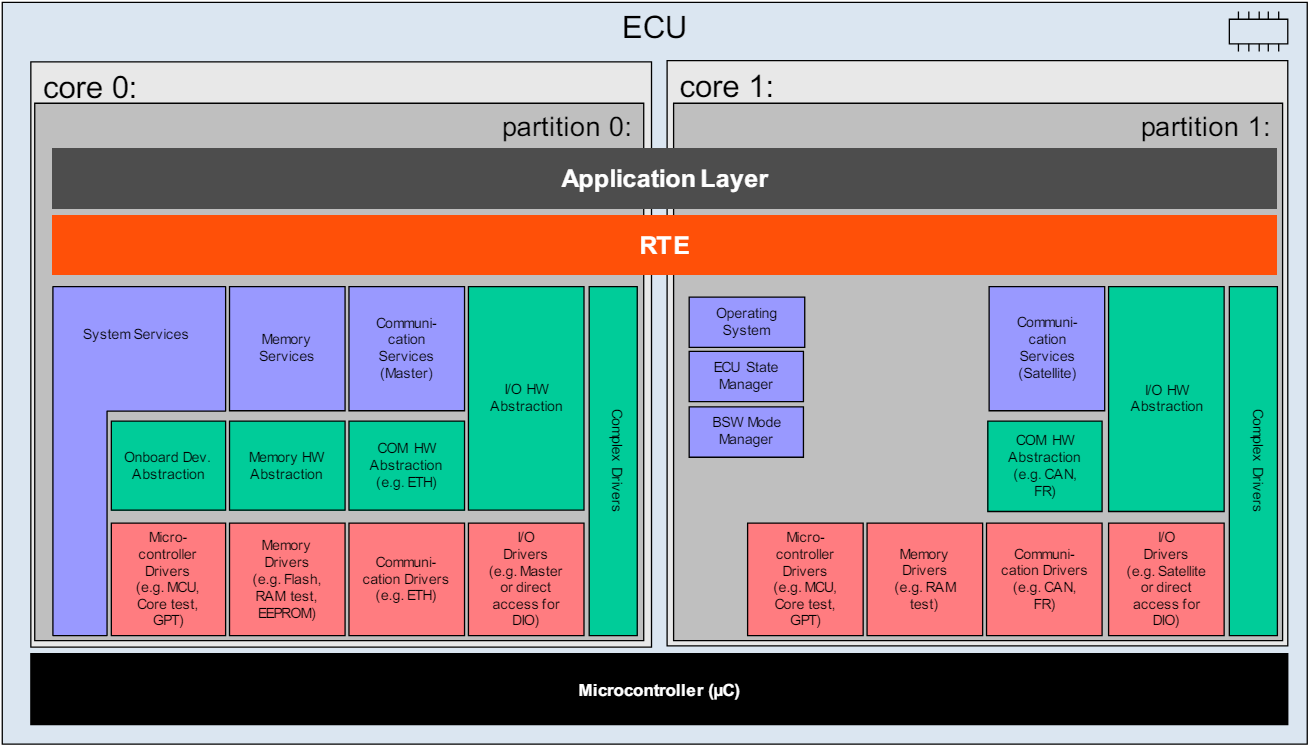
### Application Layer: Sensor/Actuator Software Components



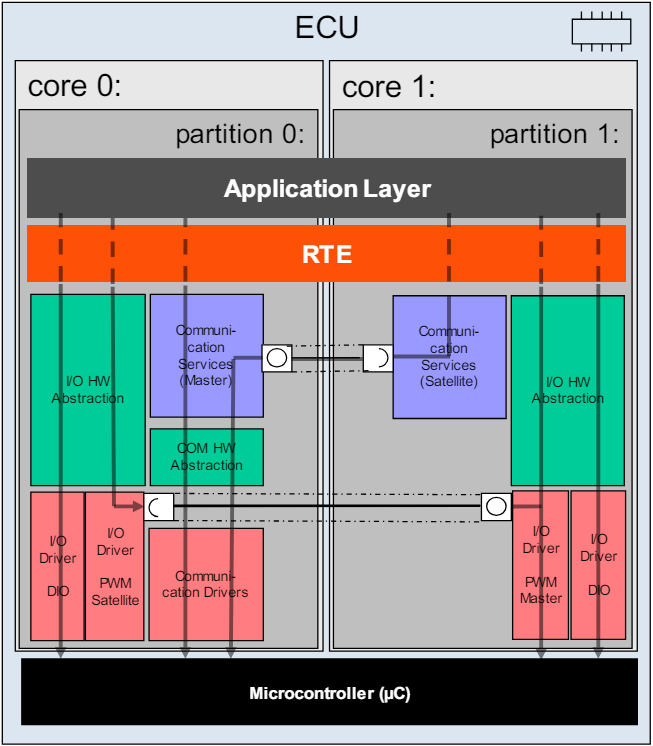


## Content of Software Layers in Multi-Core Systems

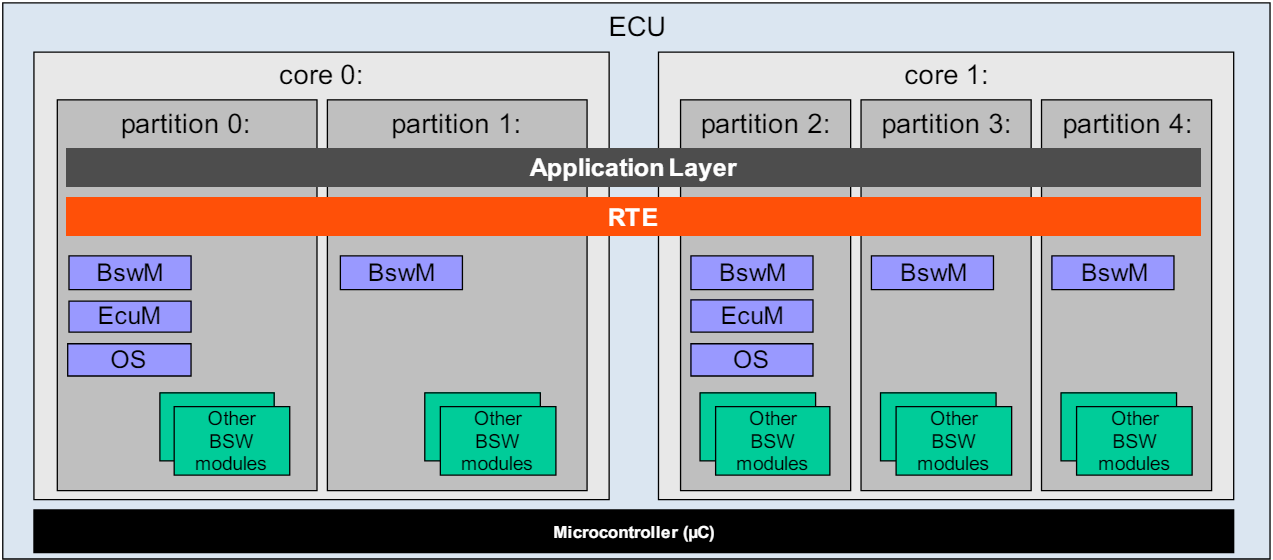
### Example of a Layered Software Architecture for Multi-Core Microcontroller



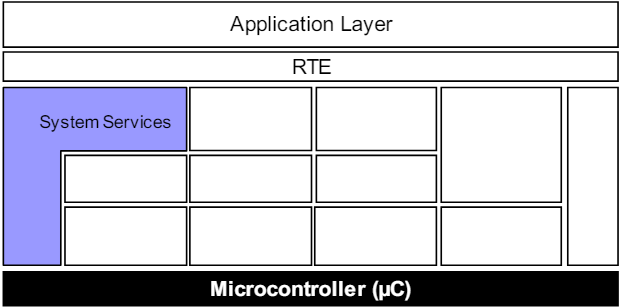
### Detailed View of Distributed BSW Modules

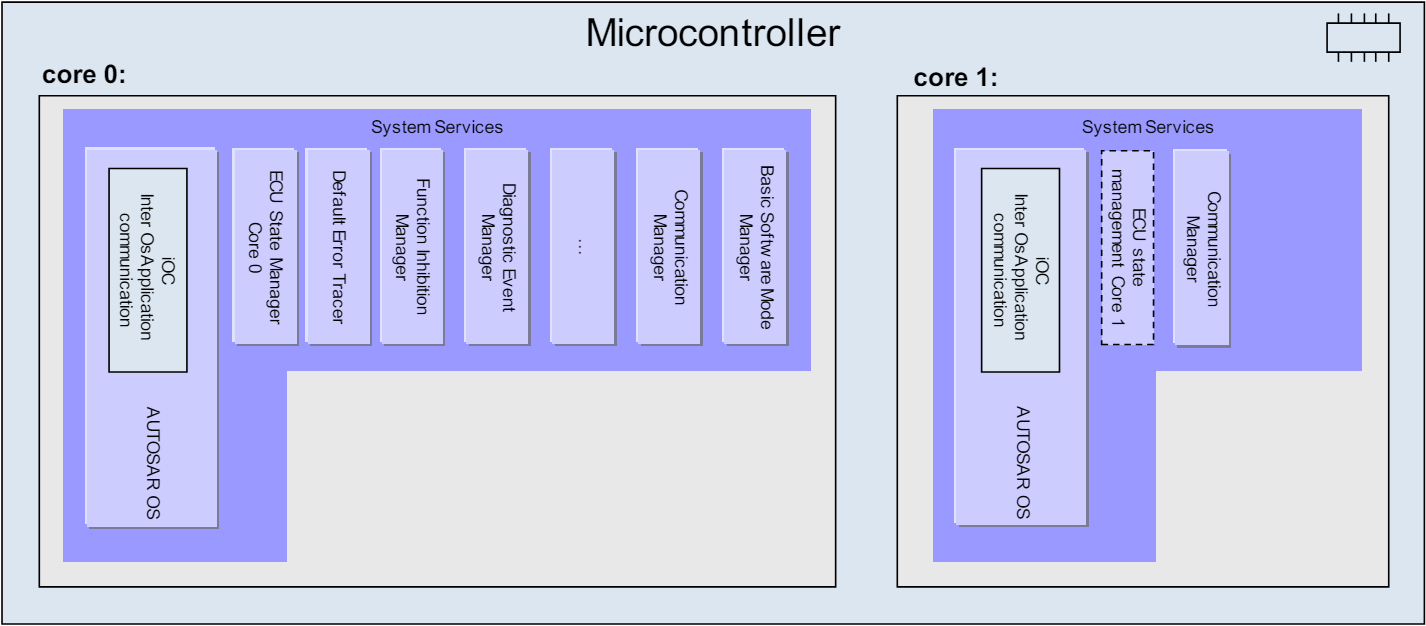


### Overview of BSW Modules, OS, BswM and EcuM on Multiple Partitions



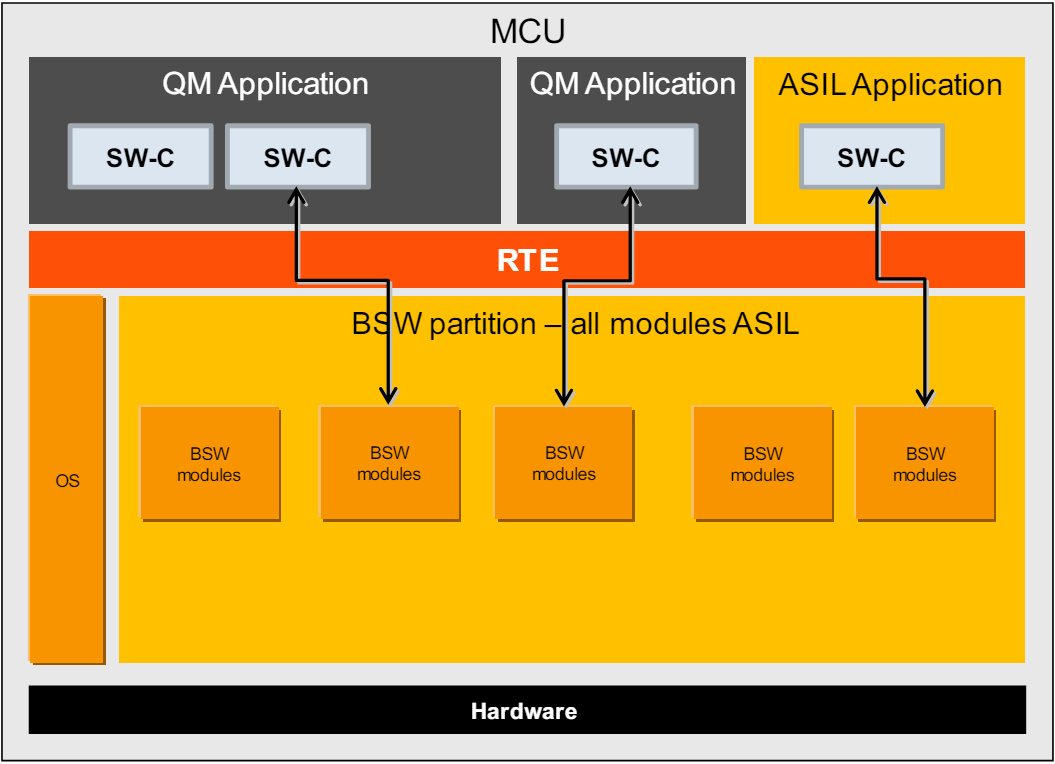
### Scope: Multi-Core System Services



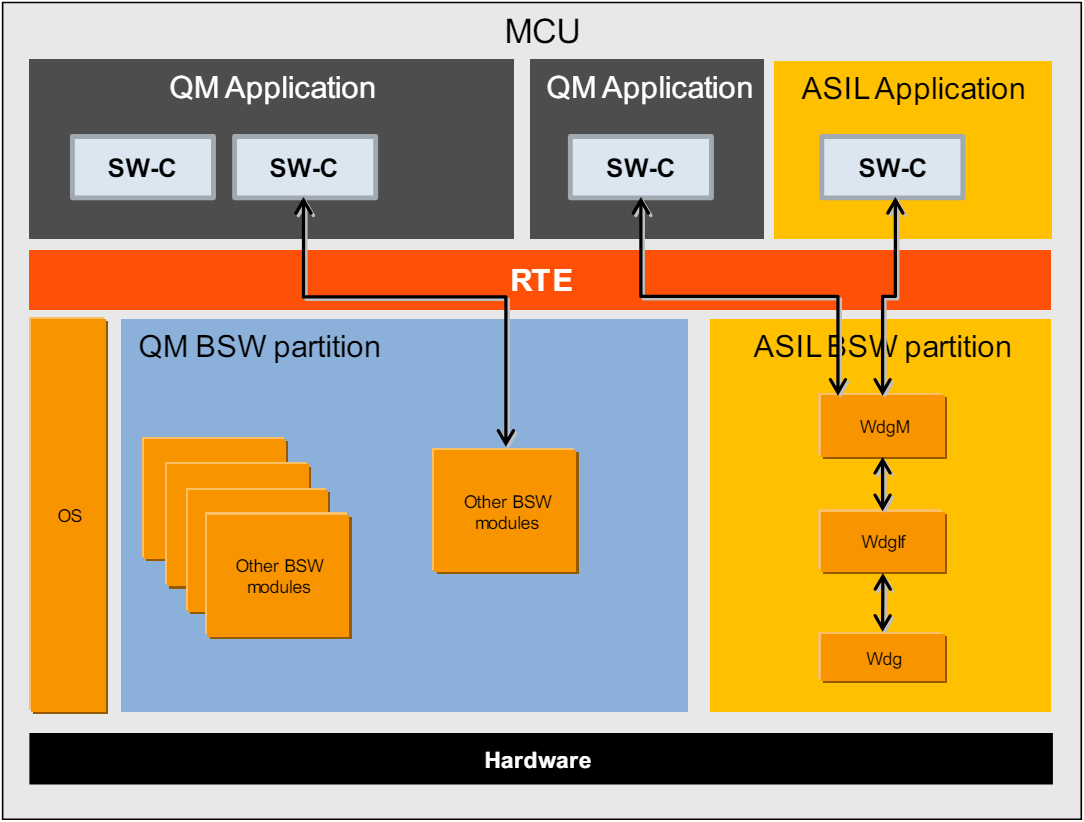


## Content of Software Layers in Mixed-Critical Systems

### Overview of AUTOSAR safety handling

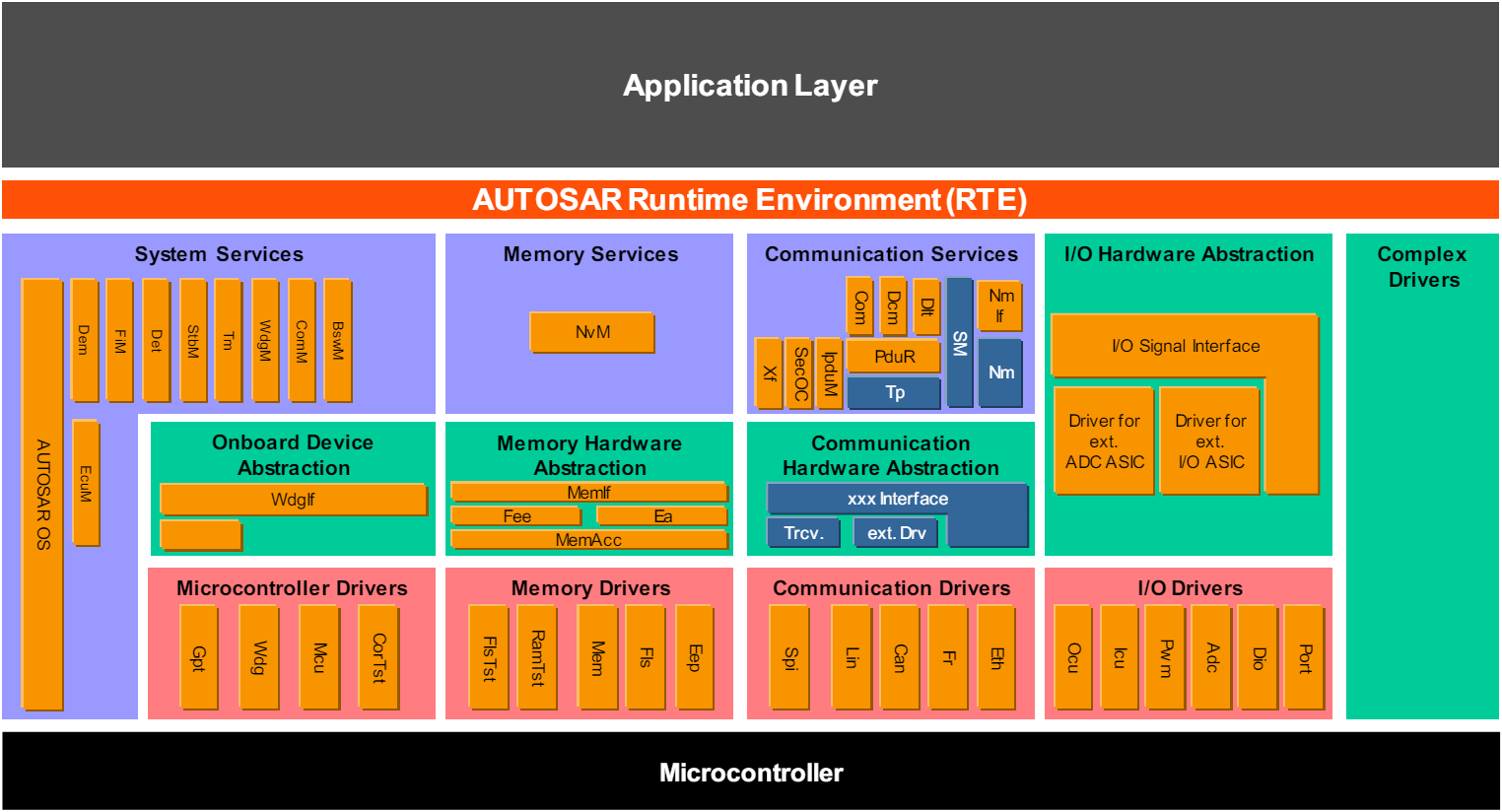


### AUTOSAR BSW distribution for safety systems



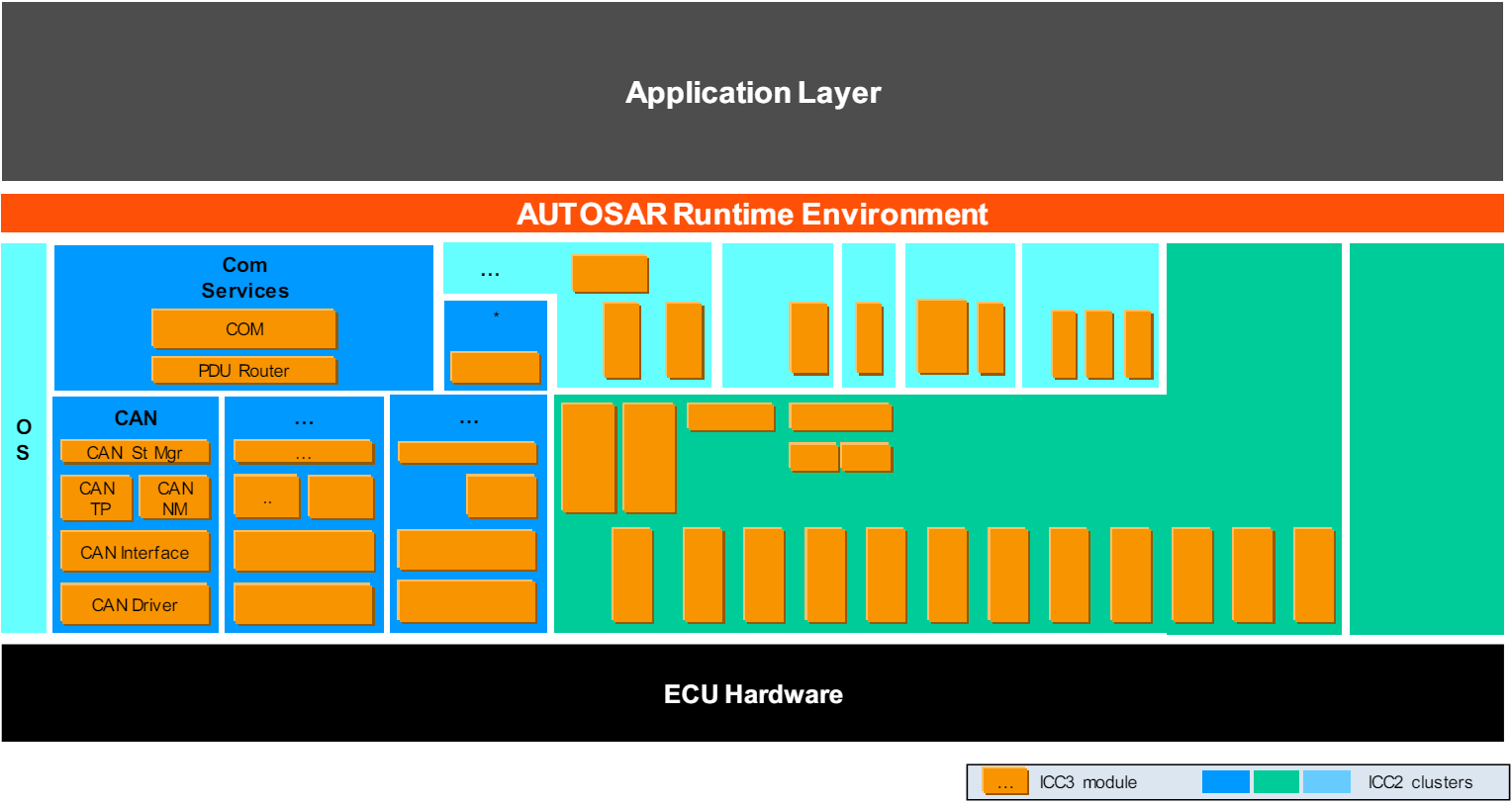
## Overview of Modules

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

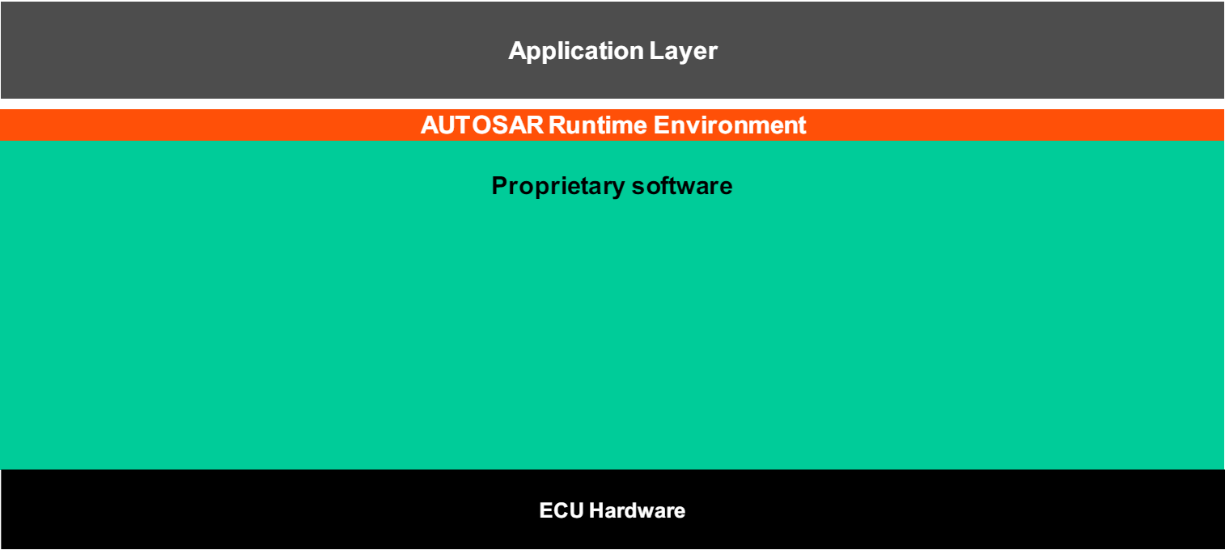




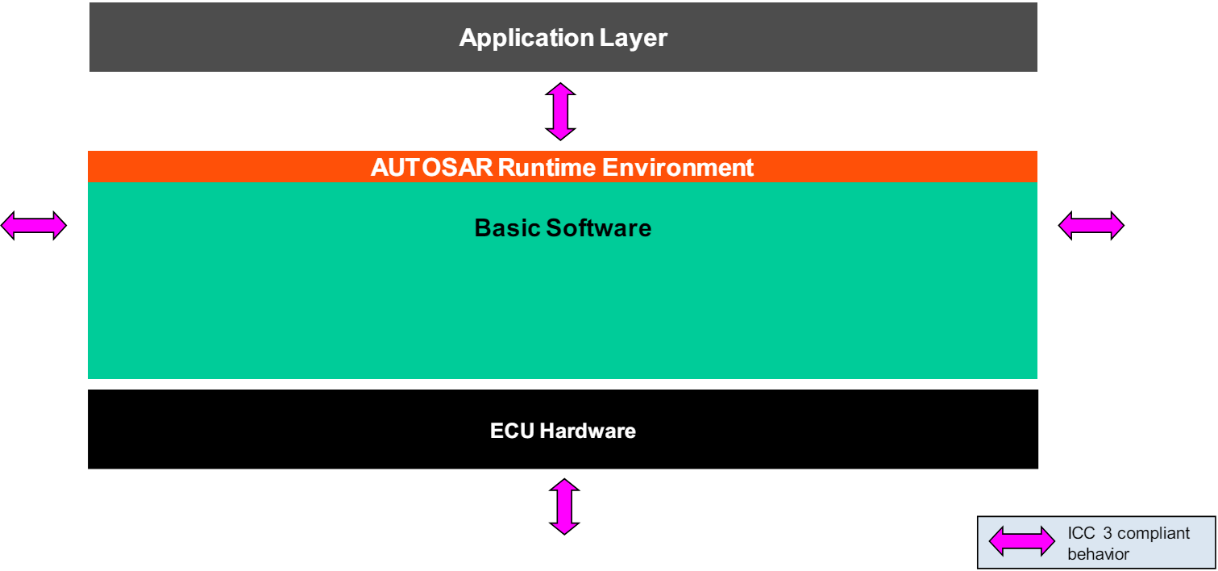
### Overview of Modules – Implementation Conformance Classes – ICC2



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



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

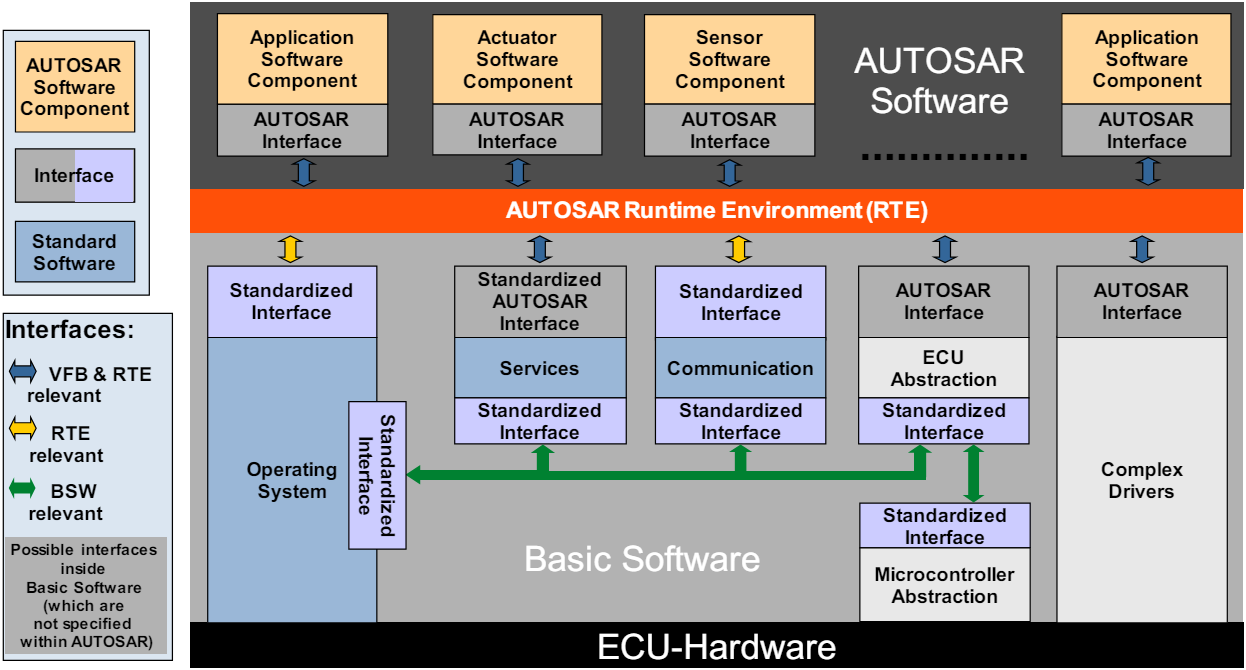


## Interfaces: General Rules

### Interfaces Type of Interfaces in AUTOSAR

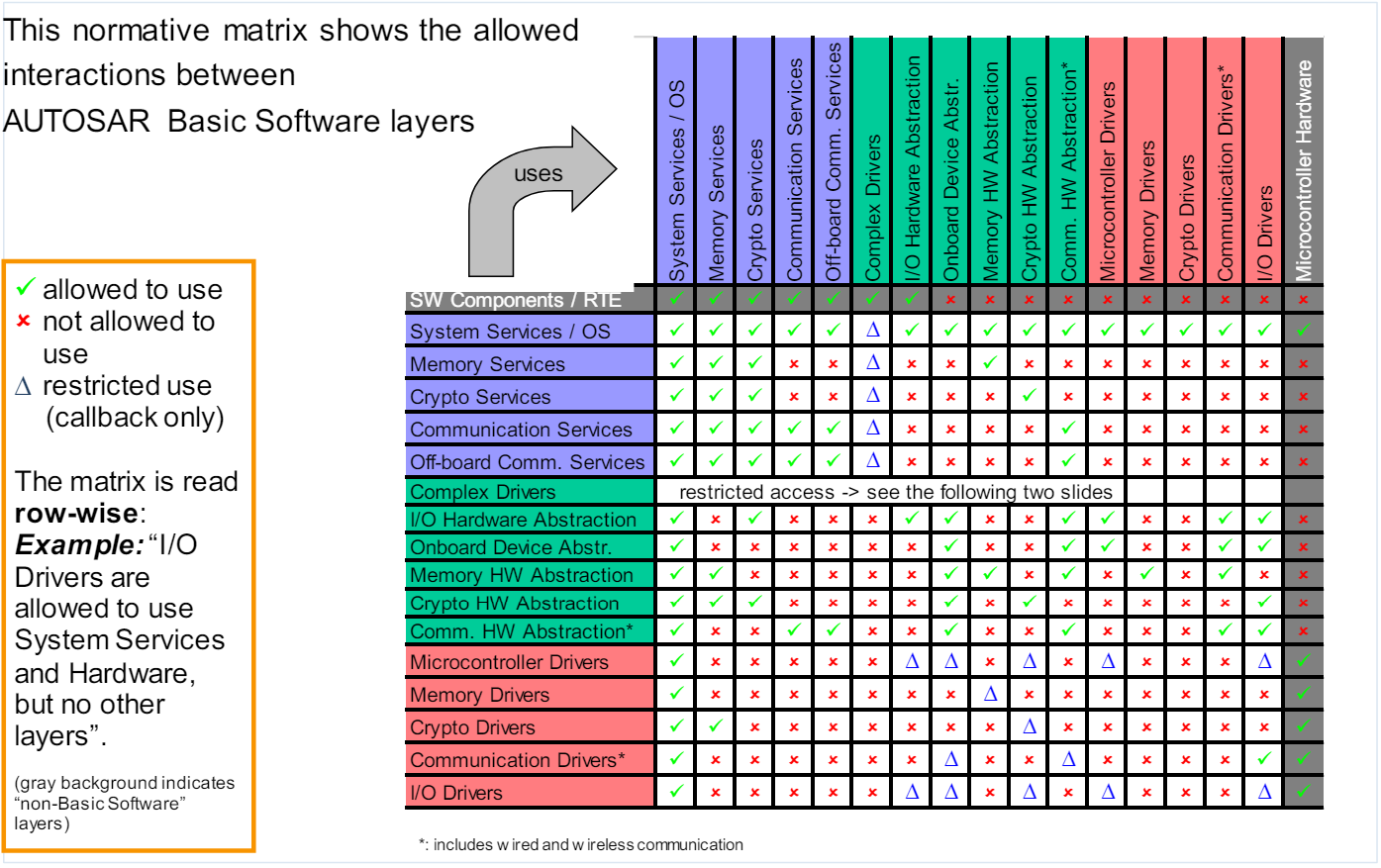
|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

### Components and interfaces view (simplified)

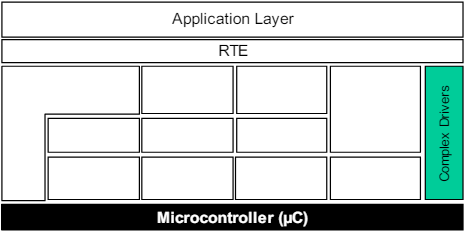


### General Interfacing Rules

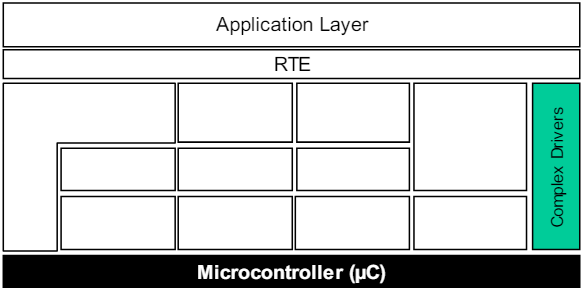
### Layer Interaction Matrix



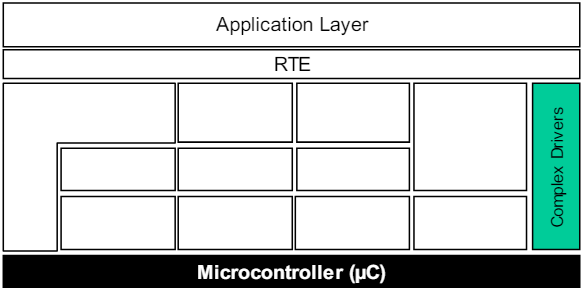
### Interfacing with Complex Drivers (1)



### Interfacing with Complex Drivers (2)



### Interfacing with Complex Drivers (3)



## Interfaces: Interaction of Layers

### Introduction