01. AUTOSAR LÀ GÌ?

Bài này là bài viết mở đầu cho chuỗi bài giới thiệu về kiến trúc Autosar, một kiến trúc quen thuộc trong các lĩnh vực Automotive. Để hiểu và học tập chuỗi bài này, các bạn cần có kiến thức chung về **Embedded**, **lập trình C**, kiến trúc phân lớp, **một số kiến thức chung về vi điều khiển** như UART, CAN, I2C, ... Với chuỗi bài viết này, mình sẽ thống nhất sử dụng các thuật ngữ chung bằng tiếng anh, một vài thuật ngữ được giới thiệu ở bài viết **AUTOSAR Notes**.

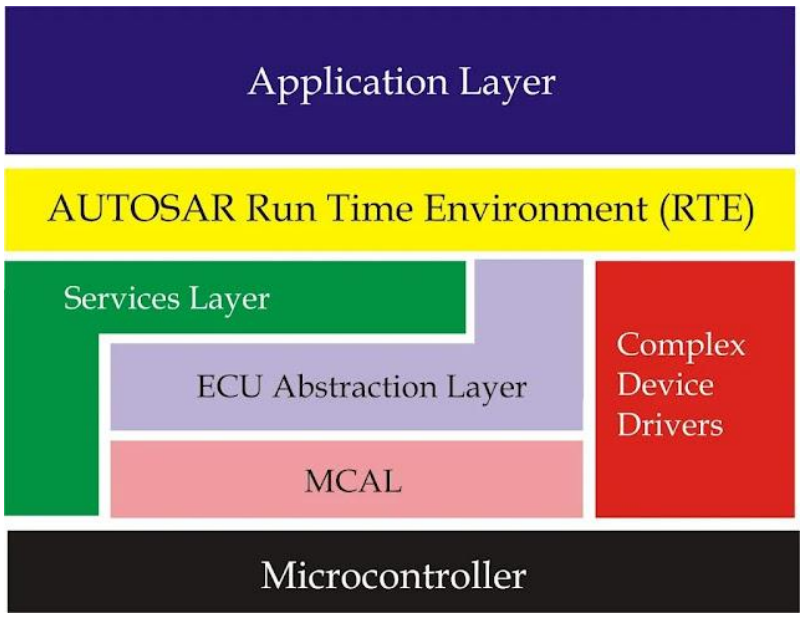
👉 AUTOSAR là gì?

AUTOSAR *- viết tắt của* ***AUT****omotive* ***O****pen* ***S****ystem* ***AR****chitecture*, là một kiến trúc phân lớp, với các thông số kỹ thuật tiêu chuẩn được quy định bởi một nhóm các công ty: BMW Group, BOSCH, Continental, Daimler, Ford, General Motors, PSA Group, Toyota, VOLKSWAGEN.

Trên đây là những thành viên chính (Core Members) của mối quan hệ đối tác AUTOSAR, những công ty/tập đoàn đã sáng lập ra chuẩn AUTOSAR. Trên thực tế có nhiều thể loại thành viên khác nhau đóng góp cho sự phát triển của AUTOSAR: *Core Members*, *Premium Members*, *Development Members*.

Ngoài các Core Members, có nhiều Premium Members cũng đang tham gia phát triển các công cụ phát triển (tool development), service providers, ... nhằm tiêu chuẩn hóa việc phát triển phần mềm cho ECU sử dụng trong các ứng dụng Automotive.

⏭ **AUTOSAR** được triển khai theo mô hình phân lớp, giống như mô hình OSI. Các lớp trong mô hình này dùng để xử lý và trừu tượng hóa các hoạt động khác nhau của code. AUTOSAR được sử dụng cho các vi điều khiển với mục đích sử dụng cho các ứng dụng automotive - CAN, FlexRay, Ethernet, ...



Hình trên là một kiến trúc phân lớp đơn giản nhất theo Autosar,

- **Application Layer:** Tầng này chứa code ứng dụng ở tầng cao nhất. Nó có thể bao gồm các khối ứng dụng khác nhau gọi là **SWCs - Software Components** cho từng tính năng mà ECU cần hỗ trợ tùy theo từng ứng dụng.
  Ví dụ, các chức năng như power window hay temperature measurement chính là các SWC riêng biệt.
- **AUTOSAR RTE:** Đây là một trong những layer quan trọng nhất của AUTOSAR, nó cung cấp khả năng giao tiếp giữa các SWC khác nhau và cả giữa các ECU khác nhau. Application layer sử dụng layer này trong khi giao tiếp với các layer bên dưới bằng các **ports**.
- **Services Layer:** Tầng nay cung cấp các dịch vụ khác nhau cho các application sử dụng. Các dịch vụ như: System Services, Memory Services, Crypto Services, Off board communication Services, Communication Services.
- **ECU Abstraction Layer:** Tầng này cung cấp các tính năng liên quan đến abstraction (Trừu tượng hóa). Nó bao gồm các tầng abstraction như: I/O Hardware Abstraction layer, On board device abstraction, Memory hardware Abstraction, Crypto hardware abstraction, ... Mục đích của tầng này nhằm độc lập phần mềm ứng dụng với phần cứng - Tức là code của tầng ứng dụng sẽ không cần thay đổi khi thay đổi phần cứng.
- **MCAL:** viết tắt của **M**icro **C**ontroller **A**bstraction **L**ayer, bao gồm các drivers giúp cho các lớp trên có thể giao tiếp với phần cứng (Các ngoại vi của Vi điều khiển).

👉 Lí do cần có AUTOSAR?

Thực tế thì làm gì cũng cần có tiêu chuẩn rõ ràng. Đối với việc viết phần mềm cho ECU, các developer có thể đối mặt với nhiều vấn đề khác nhau:

- Hệ thống nhúng là một lĩnh vực rất lớn và có rất nhiều nhà sản xuất chất bản dẫn khác nhau. Nền tảng phần cứng và phần mềm có thể được lựa chọn tùy thuộc vào từng yêu cầu ứng dụng khác nhau. Chính việc này có thể dẫn đến khó khăn trong việc phát triển cũng như tính di động của phần mềm.
  Chẳng hạn như việc phần cứng thay đổi, gần như toàn bộ phần mềm sẽ phải thay đổi theo, làm tăng chi phí và thời gian phát triển phần mềm.
- Automotive là một hệ thống phức tạp bao gồm nhiều hệ thống nhỏ hơn gọi là **ECUs - Electronic Control Unit**, nên việc bảo trì và phát triển phần mềm cho từng ECU là việc không dễ. Độ phức tạp càng tăng thêm khi các ECU khác nhau sử dụng các MCU khác nhau để đáp ứng các yêu cầu về mặt ứng dụng và chi phí.
  Vì vậy, có thể chúng ta sẽ cần phát triển rất nhiều phần mềm khác nhau cho các MCU này.
- Với cách viết phần mềm thông thường, để giao tiếp giữa nhiều ECU khác nhau, nhà phát triển sẽ cần tạo ra các giao thức, tiêu chuẩn giao tiếp, gọi là **Custom standard**. Việc này rất tốt, nhưng lần sau làm việc với chiếc xe khác, các ECU khác thì lại rất khó để maintain và tốn nhiều chi phí hơn.

- Một chiếc ô tô có nhiều bộ phận được sản xuất bởi các công ty khác nhau được gọi là công ty tier 1, cung cấp phụ tùng cho các OEM như BMW, Volkswagen, ... Ngày nay, hầu hết các bộ phận cơ khí đều trở nên thông minh hơn bằng cách thêm các ECU vào. Vì vậy cũng cần có các tiêu chuẩn để giao tiếp với cả những ECU của OEM khác nhau này.

⏭ Với những lý do nêu trên, cần có một số tiêu chuẩn thiết kế phần mềm để có thể thống nhất việc phát triển và giao tiếp giữa các ECU khác nhau, của các OEM khác nhau. Và AUTOSAR sinh ra từ đây để giải quyết những vấn đề trên.

AUTOSAR được thiết kế theo kiến trúc phân lớp, và tầng ứng dụng sẽ được viết độc lập với phần cứng, để cùng một code ứng dụng có thể chạy trên các nền tảng phần cứng khác nhau. AUTOSAR cung cấp một layer riêng để giao tiếp với phần cứng gọi là **MCAL** (Micro Controller Abstraction Layer). AUTOSAR còn cung cấp tiêu chuẩn để giao tiếp giữa các ECU của các nhà phát triển khác nhau (Cả OEM và Tier 1), từ đó giảm chi phí khi maintain custom standard.

Các ECU sử dụng AUTOSAR có thể giao tiếp với nhau bất kể sự khác biệt về phần cứng. Hầu hết các nhà sản xuất chip đều cung cấp tầng MCAL của AUTOSAR, nhưng nếu họ không cung cấp thì developer cần phải viết tầng MCAL này hoặc thuê các công ty dịch vụ viết.

❓ Đây có phải là một hạn chế của AUTOSAR

Trên thực tế đây không phải là một hạn chế, trong thế giới thay đổi nhanh chóng hiện nay, việc đáp ứng hoàn thành các dự án đúng deadline ngày càng nghiêm ngặt. Nên khi có AUTOSAR, việc đáp ứng nhu cầu phát triển phần mềm sẽ dễ dàng hơn. Mặc dùng AUTOSAR triển khai theo kiến trúc phân lớp và mọi thứ trông có vẻ có sẵn, nhưng chúng ta vẫn cần phải triển khai viết code cho các chức năng của SWC trong một **Runnable** của SWC.

❓ Nếu muốn sử dụng các thiết bị ngoài không hỗ AUTOSAR thì sao?

Nếu sử dụng các thiết bị không được AUTOSAR hỗ trợ, chúng ta có thể sử dụng tầng **CDD - Complex Device Drivers**. Tầng này cho phép truy cập trực tiếp vào lớp MCAL từ tầng Application và chúng ta có thể giao tiếp trực tiếp giữa thiết bị ngoài với ECU. Chúng ta có thể tự phát triển phần mềm cho các thiết bị này, tuy nhiên nó phụ thuộc vào phần cứng, nên không có khả năng tái sử dụng như SWC.

👉 Các phiên bản AUTOSAR

Có 2 loại kiến trúc AUTOSAR là: **Classic** và **Adaptive**. Bản Classic có tất cả các module cơ bản cần có cho các ứng dụng, trong khi bản Adaptive có thể config được và điều chỉnh theo các ứng dụng bằng cách loại bỏ các module không cần thiết.

Bản Classic hiện tại đang là **4.7.0**, còn bản Adaptive hiện tại là **21.11** ⇨ Phiên bản này chỉ đúng tại thời điểm viết bài và có thể không còn đúng vào thời điểm hiện tại.

Past Releases

AUTOSAR Foundation Release R21-11

AUTOSAR Foundation Release R20-11

AUTOSAR Foundation Release R19-11

AUTOSAR Foundation Release 1.5.1

AUTOSAR Foundation Release 1.5.0

AUTOSAR Foundation Release 1.4.0

AUTOSAR Foundation Release 1.3

AUTOSAR Foundation Release 1.2

AUTOSAR Foundation Release 1.1

AUTOSAR Foundation Release 1.0

02. BẮT ĐẦU DỰ ÁN AUTOSAR

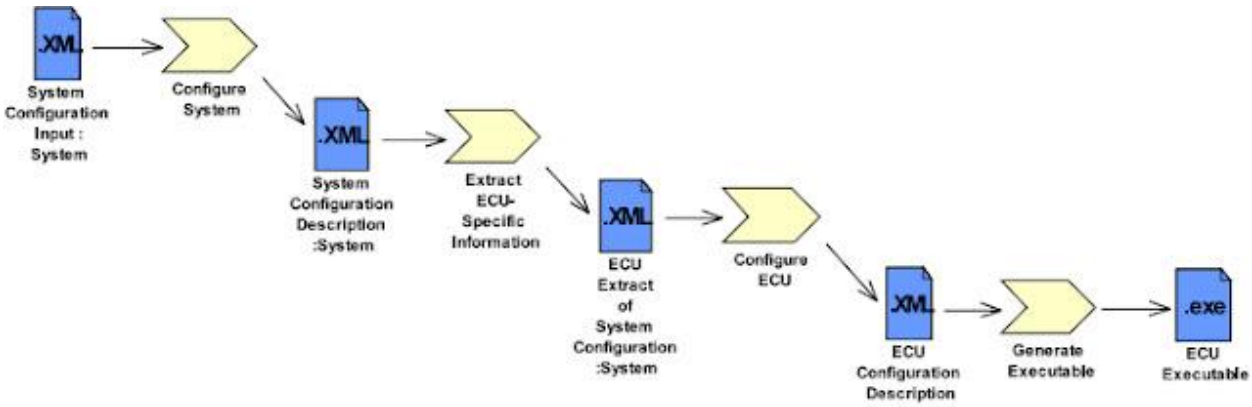
AUTOSAR khá khó tiếp cận với một lập trình viên nhúng mới bắt đầu, vì vậy, để nhanh tiếp cận hơn với khái niệm này, chúng ta có thể bắt đầu với các project với AUTOSAR. Như **bài viết trước** đã giới thiệu, AUTOSAR là một kiến trúc phân lớp, phần mềm cho ECU dựa trên AUTOSAR dựa trên các thông số kỹ thuật tiêu chuẩn.

Phần mềm có thể được phát triển bởi các software developer, một số code config có thể được tạo ra bằng cách sử dụng một số phần mềm cấu hình ECU khác nhau như Vector's DaVinci Configurator and Developer , K-SAR , v.v..

****** Lưu ý bài viết này chỉ đưa ra cái nhìn tổng quan về một project cơ bản theo AUTOSAR, chứ không nêu chi tiết từng bước thực hiện.***

👉 Các thành phần trong AUTOSAR Project

Developer sẽ phát triển các phần mềm (giống như driver) có khả năng tái sử dụng, để xử lý luồng code chính. Cùng với đó, họ sẽ phát triển một phần dựa trên một phần mềm cụ thể, phần mềm này cung cấp cho người dùng một giao diện (GUI) để người dùng có thể config data theo ý mình.



Hình trên chỉ ra quy trình phát triển một project AUTOSAR từ System Configuration đến file thực thi cuối cùng (.elf, .hex, .bin, ...). Các file định dạng **.XML** trong hình có thể hiểu như các file cấu hình giao diện phần mềm cho người dùng sử dụng, trong AUTOSAR, các file này có định dạng **.arxml (AUTOSAR Extensible Markup Language)**.

Một số bước trong quá trình config:

1. **System Configuration**
   Các ECU trong system và phần cứng của chúng được cấu hình trong *SWC - Software Component* và *Compostion SWC* được ánh xạ tới các ECU tương ứng.

2. **Generate System Configuration Description**
   Đầu ra của bước một là một file có định dạng .arxml, gọi là *System Configuration Description*. File này chứa tất cả thông tin của toàn bộ ECUs trong hệ thống.
3. **ECU Extract generation**
   Bước này sử dụng đầu ra của bước hai làm đầu vào, và gen ra một file được gọi là ECU *Extract arxml file*. File này chứa thông tin của các ECU đơn lẻ.
4. **ECU Configuration**
   Trong bước này, thông tin ECU ở bước trên được sử dụng làm đầu vào và ECU tương ứng được cấu hình theo yêu cầu của ứng dụng, ví dụ như *BSW - Basic Software*, hay cấu hình hệ điều hành OS, ...
5. **Generate ECU Configuration Description**
   Bước này dùng để tạo đầu ra của bước 4, gen ra file *ECU Configuration Description arxml*, file này được sử dụng trong quá trình tạo ra file thực thi (.elf, .hex, ...).

👉 Các thông tin trong file System Configuration Description

- Các ECU có trong hệ thống
- Hệ thống mạng giao tiếp giữa các ECU và cấu hình của chúng
- AUTOSAR nhằm mục đích chuẩn hóa toàn bộ quá trình phát triển tất cả dữ liệu, kích thước, ... sẽ được truyền hoặc nhận cần phải được định cấu hình tại thời điểm cấu hình.
- Định nghĩa SWC - Software Component với các khái niệm: *ports, interfaces, connections*.
- Ánh xạ SWC - Software Component tới ECU.

Tệp trên đóng vai trò là đầu vào cho cấu hình ECU, các khái niệm sẽ được giới thiệu ở những bài viết sau này.

👉 Các thông tin trong file ECU Configuration Description

ECU Extract chỉ xác định các thành phần cấu hình cần được tất cả các ECU trong hệ thống đồng ý, nhưng điều này không đủ để gen ra code có thể chạy trên phần cứng. Vì tệp này không có bất kỳ thông tin cấu hình cấp thấp (low-level configuration) nào có thể được sử dụng để định cấu hình các low layer của kiến trúc AUTOSAR.

Vì vậy, *ECU Configuration description* có thông tin mà phần mềm cấu hình sử dụng và tạo các tệp .c và .h được biên dịch thêm và chạy trên phần cứng.

👉 Mô phỏng một dự án giống như AUTOSAR

Các bài toán về Automotive (AUTOSAR) đa phần đều là các bài toán lớn, và thường chỉ có thể gặp ở các công ty lớn về mảng này. Cùng với đó là chi phí cho các phần mềm kể trên cũng khá đắt đỏ cho việc học tập. Đối với các bạn mới tiếp xúc và chưa có cơ hội làm việc thực tế trong các dự án AUTOSAR, chúng ta có thể bắt đầu với một vài ví dụ nhỏ tự xây dựng tương tự với các dự án AUTOSAR.

Trên blog mình đã xây dựng một bài toán nhỏ, mô phỏng về cách triển khai một driver, với file configuration để gen ra user config!!!

✅ **Ý tưởng**: Thiết kế driver cho ngoại vi MPU. Cùng với đó, bài toán này mình thiết kế giao diện để user có thể config được và gen ra code .c/.h.

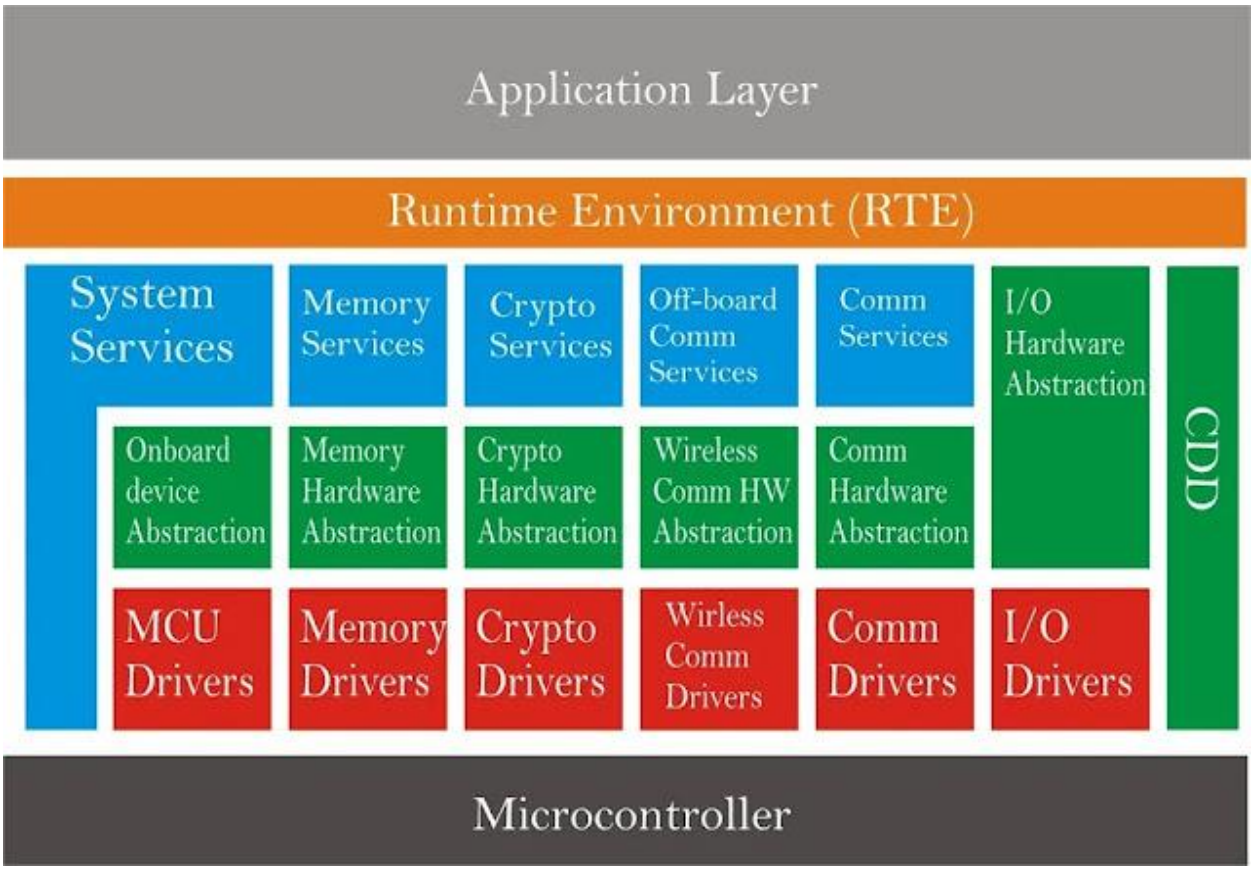
03. BSW - BASIC SOFTWARE

Để hiểu những nội dung của bài viết này, các bạn cần đọc hiểu các nội dung trong **bài viết mở đầu về AUTOSAR**. Bài viết này chúng ta sẽ tìm hiểu về layer AUTOSAR BSW - Basic Software (Layer dưới của RTE), đây là một layer quan trọng giúp cho tầng Application có thể sử dụng để giao tiếp với các peripheral khác nhau của MCU.

👉 BSW - Basic Software là gì?

BSW - Basic Software bao gồm các module Basic Software (BSWM) dưới dạng tập hợp các file phần mềm (code hoặc description), những file code này triển khai các hàm cơ bản của một ECU.

BSW được cấu hình bằng các phần mềm cấu hình như ***Vector Davinci Configurator*** là các cấu hình nằm dưới tầng RTE (*Như trong hình*).



Hình trên là một mô hình chi tiết về kiến trúc phân lớp của AUTOSAR:

- **Micro-Controller**
  Tầng màu đen dưới cùng là phần cứng Micro-Controller.

- **MCAL - Microcontroller Abstraction Layer**
  Các khối màu đỏ đại diện cho tầng MCAL (Microcontroller Abstraction Layer), bao gồm các drivers của các peripherals. MCAL là *tầng thấp nhất của Basic Software*. Tầng này trực tiếp truy cập vào phần cứng nên rất phụ thuộc vào phần cứng, và sẽ cần thay đổi đối với các phần cứng khác nhau.
- **ECU Abstraction Layer**
  Các lớp màu xanh green (ngoại trừ CDD), được hiểu là ECU Abstraction Layer. *Abstraction - Trừu tượng*, trong bài toán này có thể hiểu là làm giảm tối đa sự phụ thuộc vào phần cứng. Tức là cung cấp các API chung nhất cho người dùng, để khi phần cứng có thay đổi thì các API này cũng ít bị ảnh hưởng. (**Tham khảo thư viện HAL của hãng ST - chip STM32**).
- **CDD - Complex Device Drivers**
  Tầng CDD kết nối trực tiếp SWC - tầng Application với phần cứng MCU thông qua tầng RTE. Tầng này hữu ích cho việc viết các *functions/drivers* của các *peripheral/external devices* mà không được định nghĩa trong AUTOSAR, hoặc những functions yêu cầu ràng buộc cao về mặt thời gian.
  Khác với tầng MCAL kết hợp với ECUAL, tầng CDD phụ thuộc nhiều vào phần cứng và khả năng sử dụng lại code sẽ kém hơn.
- **Service Layer**
  Đây là tầng trên cùng của BSW, tầng này cung cấp các dịch vụ (service) cơ bản cho các Application, RTE, BSW. Các service có thể là: Các function liên quan đến OS, Communication Services, Memory Service (NVRAM), ECU state management, ...

👉 Internal/External Driver trong AUTOSAR

Phụ thuộc vào loại peripheral được sử dụng, AUTOSAR có 2 loại driver: **Internal** và **External**.

Internal Driver được sử dụng để truy cập tới các thiết bị ngoại vi nội bên trong vi điều khiển như EEPROM, ADC, GPIO, ... Trong khi đó External Driver sử dụng để truy cập tới các thiết bị ngoại vi kết nối bên ngoài Vi điều khiển, như External Flash, SD Card, ...

Các Internal Driver nằm trong tầng MCAL, trong khi External Driver nằm trong tầng ECU Abstraction Layer. Một vài trường hợp ngoại lệ, với External Device như memory mapped memories, nó có thể được truy cập trực tiếp bằng vi điều khiển và nó nằm trong tầng MCAL.

👉 Khái niệm Interfaces, Handlers, Manager

➤ **Interfaces** là khái niệm nằm trong tầng ***ECU Abstraction***, cung cấp một chức năng để trừu tượng hóa các module cấp thấp (Low-level module) và cung cấp một số API có thể được sử dụng ở các lớp trên.

Interfaces còn cung cấp khả năng truy cập vào một số thiết bị cụ thể bất kể số lượng thiết bị hiện có cùng loại, và không phụ thuộc vào Hardware.

➤ **Handlers** có nhiệm vụ kiểm soát các truy cập đồng thời, các truy cập liên tiếp và bất đồng bộ của một hay nhiều clients, Khi có nhiều truy cập xảy ra thì Handlers sẽ đưa chúng vào các cơ chế như Buffer, **Queue**, ... để tránh mất sự kiện.

Các chức năng của ***Handlers*** thường được triển khai trong Driver hoặc Interfaces.

➤ **Manager** là khái niệm nằm trong tầng ***Services***, nó cung cung các dịch vụ cụ thể cho nhiều clients khác nhau. Nó được yêu cầu trong mọi trường hợp mà chỉ mình ***Handlers*** là không đủ để xử lý các yêu cầu.

Ví dụ. ***NVRAM Manager*** điều khiển việc truy cập vào bộ nhớ **Internal/External Flash**.

👉 Chi tiết các tầng của BSW

① MCAL (Microcontroller Abstraction Layer)

Tầng MCAL bao gồm những Layer sau:

- ***Microcontroller Drivers*** - Cung cấp các function để truy cập đến các ngoại vi nội như Watchdog, Timer, ...
- ***Memory Drivers*** - Cung cấp các function để truy cập đến các bộ nhớ nội như Internal Flash, Internal EEPROM, và các bộ nhớ ngoài như External Flash.
- ***Crypto Drivers*** - Cung cấp các function để truy cập đến Internal Crypto như SHE, HSM, ...
- ***Wireless Communication Drivers*** - Cung cấp các function cho các hệ thống mạng không dây (Giao tiếp trong hoặc ngoài xe).
- ***Communication Drivers*** - Cung cấp các function cho các giao tiếp on board như **UART**, **I2C**, **SPI**, ... và các giao tiếp Vehicle như CAN, LIN, FlexRay, ...
- ***I/O Drivers*** - Cung cấp các function để truy cập và sử dụng các chân I/O của MCU như chân Digital, Analog, PWM, ...

② CDD - Complex Device Drivers

Module này thích hợp trong việc triển khai các chức năng ***non-standard*** (không theo tiêu chuẩn của AUTOSAR) trong BSW stack. Trên thực tế AUTOSAR là một tiêu chuẩn chung cho phần mềm phục vụ cho lĩnh vực Automotive, nên nó không thể cover hết được những trường hợp riêng biệt của từng hãng làm phần mềm.

Chính vì vậy có nhiều trường hợp mà các hãng sẽ triển khai một số chức năng mà AUTOSAR không hỗ trợ, CDD sẽ được sử dụng trong những trường hợp như vậy.

Một trường hợp khác, một số chức năng yêu cầu ràng buộc về mặt thời gian, và yêu cầu thời gian thấp hơn thời gian tối thiểu của AUTOSAR OS cũng cần đặt vào tầng CDD, vì CDD giúp kết nối MCU với ứng dụng.

➤ Nhược điểm lớn nhất của tầng CDD là không theo tiêu chuẩn, và phụ thuộc vào application nhiều hơn là MCU, vì vậy code ở tầng này sẽ ***khó porting*** giữa các dòng chip khác nhau.

③ ECU Abstraction Layer

Tầng Abstraction giúp việc viết phần mềm cho ECU sẽ độc lập với phần cứng (Cả về các thành phần MCU cũng như các thiết bị ngoại vi bên ngoài kết nối với ECU). Tức là Application sẽ được Abstraction cung cấp các API để sử dụng và không cần quan tâm đến sự thay đổi của phần cứng (ECU hay MCU).

- ***I/O Hardware Abstraction*** - Các thiết bị ngoại vi I/O có thể nằm trên chip hoặc trên board, việc cấu hình các chân của MCU cũng khá phức tạp. Tuy nhiên tầng Abstraction giúp đơn giản hóa việc cấu hình của tầng Application, bằng cách cung cấp các API để sử dụng I/O, Ví dụ - ***Pin_SetMode(pin, mode)***.
  Tầng này sẽ cung cấp các API không đổi như trên, nhưng nội dung của nó sẽ thay đổi theo phần cứng của ECU (Ví dụ các MCU khác nhau thì sẽ có số lượng chân khác nhau, tên các module quản lý chân khác nhau như GPIO, PORT, DIO, ...).
- ***Communication Hardware Abstraction*** - Đối với các module truyền thông (như CAN, LIN, FlexRay, ...), tầng Application chỉ cần quan tâm việc truyền nhận dữ liệu, không cần quan tâm chân nào được sử dụng để kết nối, kết nối trên chip hay trên board, thậm chí không cần quan tâm loại bus giao tiếp nào được sử dụng.
  Module này đảm bảo việc độc lập phần cứng như nói trên, cung cấp các API cho việc truyền/nhận dữ liệu.
- ***Crypto Hardware Abstraction*** - Module này trừu tượng hóa chức năng Crypto bằng cách ẩn các thông tin về Crypto được sử dụng (Các thiết bị Internal/External hoặc các phần mềm cơ sở). Bởi vì Application không cần quan tâm đến việc loại Crypto nào được sử dụng, trên chip, board hay mã hóa bằng phần mềm.
- ***Memory Hardware Abstraction*** - Module này trừu tượng hóa vị trí của thiết bị bộ nhớ được sử dụng. Application chỉ cần quan tâm đến dữ liệu, không có quyền kiểm soát đối với thiết bị bộ nhớ được chọn là Internal/External, ROM/Flash/SDCard, ...
- ***Onboard Device Abstraction*** - Module này trừu tượng hóa các thiết bị trên board mạch cụ thể, cung cấp các API để giao tiếp với các thiết bị trên Board như **Sensors**, **Actuators**.

④ Services Layer

- ***Communication Services*** - Đây là một nhóm các module dành cho các giao thức truyền thông mạng trên xe, cung cấp giao diện thống nhất cho mạng trên xe để giao tiếp dữ liệu, đồng thời ẩn các thuộc tính giao thức và thuộc tính message khỏi tầng Application. Communication Service Interfaces với Communication Drivers (MCAL), cùng với sự giúp đỡ của Communication Hardware Abstraction. Đây là sự độc lập phần cứng với ECU và MCU, nhưng phụ thuộc vào bus type. Vì vậy, một phần của tầng Service có thể thay đổi nếu bus type thay đổi (CAN, FlexRay, ...).
- ***Off board Communication Services*** - Đây là một nhóm các module dành cho phương tiện để giao tiếp với các clients bên ngoài thông qua mạng không dây. Nó bao gồm 3 khối

được sử dụng với những chức năng khác nhau. Module này cung cấp một giao diện thống nhất cho mạng Ethernet không dây bằng cách ẩn các thuộc tính giao thức và message.

- ***Memory Services*** - Service này bao gồm một module, *NVRAM Manager*. Nó chịu trách nhiệm quản lý dữ liệu non-volatile (đọc/ghi từ các memory drivers khác nhau).
  Application yêu cầu lưu trữ dữ liệu trong bộ nhớ để sử dụng sau này, vì vậy module này được sử dụng để triển khai điều này theo một cách thống nhất và cung cấp khả năng trừu tượng hóa từ các vị trí bộ nhớ cấp thấp hơn.
  Memory Services cung cấp cơ chế để lưu trữ, load, checksum, ... dữ liệu non-volatile, có khả năng cấu hình cao và độc lập với phần cứng ECU.
- ***System Services*** - Đây là một nhóm các module có thể được sử dụng bởi các module của tất cả các layer, ví dụ như RTOS, Error Messager. Các services này có thể phụ thuộc vào một số MCU hoặc có thể hỗ trợ các tính năng đặc biệt của MCU (như Time Service), một phần phụ thuộc vào phần cứng ECU và Application.

Trên đây là các thuật ngữ về BSW, có một số thuật ngữ sẽ được giải thích chi tiết sau này.

NOTES. CÁC THUẬT NGỮ CHUNG TRONG AUTOSAR

AUTOSAR được xây dựng để tiêu chuẩn hóa việc phát triển phần mềm cho các ECU trong các ứng dụng Automotive. Do đó, có nhiều thuật ngữ mà có thể những người học nhúng thuông thường chưa nghe đến bao giờ. Trong chuỗi bài viết về AUTOSAR, mình sẽ sử dụng lặp lại các thuật ngữ này và nó cũng có thể xuất hiện trong rất nhiều tài liệu khác nữa về AUTOSAR! Vì vậy, trong bài viết này và cả series, mình sẽ giữ các thuật ngữ ở dạng tiếng anh để tránh sai lệch về mặt ý nghĩa.

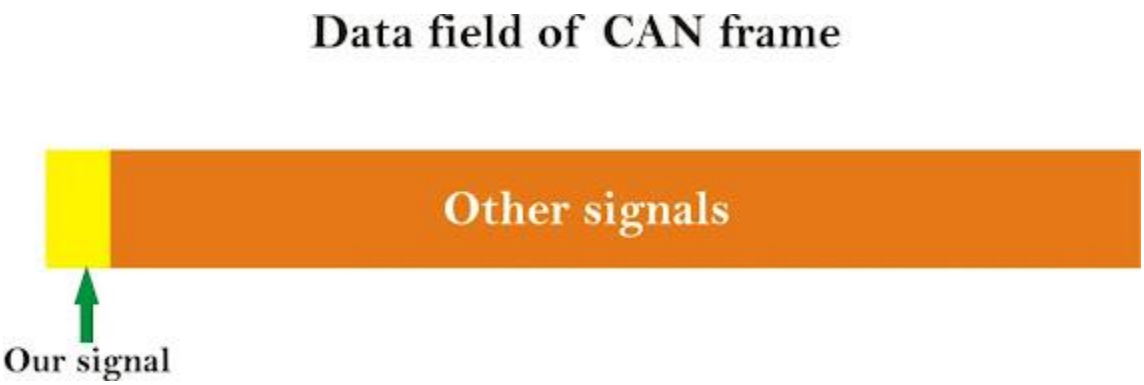
👉 Integrator

Với Autosar, khái niệm Integrator là cấu hình và tạo dự án AUTOSAR bằng giao diện phần mềm, bằng các **phần mềm GUI cấu hình bộ config** như đã giới thiệu trước đây. Trong dự án AUTOSAR, developer triển khai SWC bằng cách viết code **Runnable**.

👉 Signal

AUTOSAR thực hiện giao tiếp dựa trên các **signal**. Signal là định lượng thông tin nhỏ nhất mà một bản tin CAN có thể có. Một Signal có thể có bất kỳ kích thước nào từ 1 đến 64 bits, tùy thuộc vào bản tin CAN. Khái niệm Signal cũng có thể sử dụng cho FlexRay và các Bus khác, chỉ khác ở chỗ số lượng bit tối đa của nó.

Một ví dụ thực tế, giả sử một ECU cần biết trạng thái của cửa (locked/un-locked). Trong một chương trình C, chúng ta có thể triển khai một biến cờ (Flag) để hiển thị trạng thái của cửa. Trong AUTOSAR, chỉ cần tín hiệu **1 bit** được sử dụng để biểu thị điều này.



Tương ứng với các tín hiệu có kích thước lớn hơn, giả sử rải giá trị từ 0-7 sẽ cần 3 bit để hiển thị.

➤ Cách làm này giúp tiết kiệm tối đa không gian bộ nhớ.

SWC sử dụng Signal để giao tiếp với nhau bằng cách sử dụng **VFB-Virtual Function Bus** qua tầng **RTE**. Signal được triển khai và chỉ được hiểu bởi các layer từ COM đến Application.

Các signal có thể được nhóm khi chúng cần được giữ mối liên hệ với nhau, hoặc nhóm với nhau thành một **struct**. Ví dụ như struct gồm các bits của một thanh ghi.

👉 PDU / Message

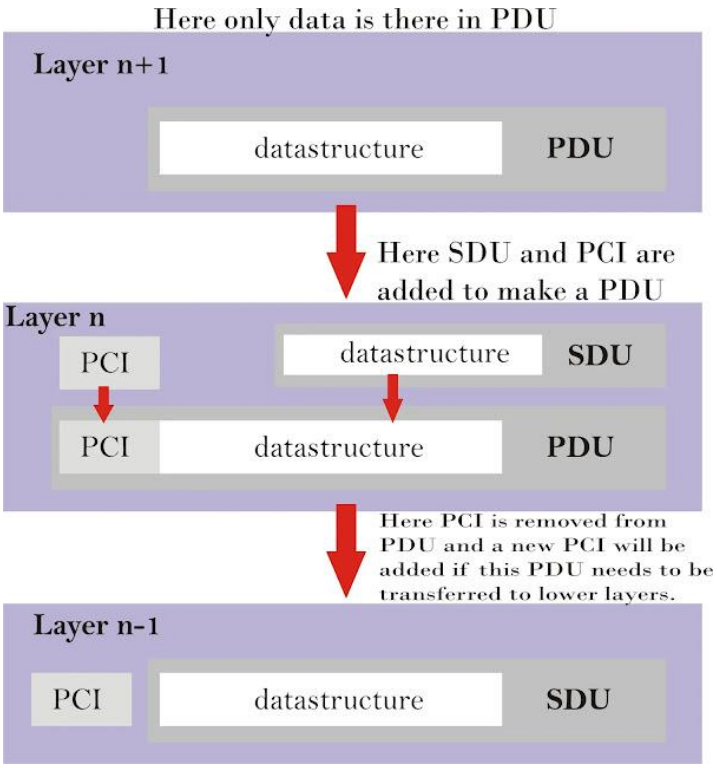
Trong AUTOSAR, một ***message*** có thể được gọi là ***PDU (Protocol Data Unit)***. PDU chứa thông tin khác với dữ liệu được sử dụng hoặc trích xuất bởi các lớp dưới hoặc lớp trên trong quá trình truyền/nhận. Có thể có n số PDU với kích thước khác nhau. Về cơ bản, PDU là một nhóm các Signal được đóng gói cùng với thông tin lớp thấp hơn.

AUTOSAR COM thực hiện đóng gói hoặc giải nén tín hiệu vào hoặc từ PDU trong khi truyền/nhận. Mỗi PDU có một ID duy nhất.

PDU chứa SDU (Service Data Unit) và PCI (Protocol Control Infomation).

- ***SDU*** là dữ liệu cần được truyền đi. Trong khi truyền, SDU được truyền từ các lớp trên xuống lớp dưới cùng với PCI. Trong quá trình nhận, SDU là dữ liệu được trích xuất từ các lớp dưới và được chuyển lên các lớp trên.
- ***PCI*** chứa thông tin cho biết điểm đến tiếp theo của SDU. Về cơ bản, nó chứa thông tin nguồn và đích của SDU, trong trường hợp này là lớp tiếp theo mà PDU cần được chuyển đến.

➤ Nói đơn giản, PDU được chuyển từ các lớp trên xuống lớp dưới và ngược lại, nơi chứa SDU và PCI.



Việc đóng gói SDU và PCI trong một PDU khi truyền từ lớp cao xuống lớp thấp hơn

Việc truyền PDU từ lớp này sang lớp khác được gọi bằng tên có liên quan đến lớp mà nó nằm trong đó. Ví dụ **I-PDU** (Interaction Layer PDU), **N-PDU** (Network Layer PDU), **L-PDU** (Data Link Layer PDU).

👉 Computation Methods

Compu Methods được sử dụng để chuyển đổi các giá trị cố định trong phần mềm thành các giá trị vật lý có thể là giá trị dấu phẩy động.

Compu Methods xác định mối quan hệ chuyển đổi các giá trị bên trong của SWC thành giá trị vật lý. Compu Methods được định nghĩa cho một **Signal**.

Có 3 loại Compu Methods chính:

- ***Linear - Tuyến tính.*** Sử dụng khi giá trị được chuyển đổi thuộc loại tuyến tính. Trong quá trình cấu hình, cần phải đưa ra phạm vi giá trị thô (Min và Max của giá trị, Gain, Offset Value).
- ***Text Table.*** Đây là Compu Methods đơn giản nhất. Nó là một bảng các giá trị số đại diện cho một số văn bản có ý nghĩa nào đó.
- ***Scale-Linear.*** Đây là một bảng các Compu Methods kiểu Linear.

👉 .cdd file

File này không liên quan gì đến tầng CDD đã từng nói ở các bài trước 😅

.cdd file là file cụ thể của Vector chứa thông tin liên quan đến cấu hình Diagnostics configuration. Mặc dù nó khá specific nhưng nhiều tool/ide vẫn dùng nó làm tiêu chuẩn.

👉 OSEK/VDX

Thuật ngữ này là viết tắt của một từ tiếng Đức hơi dài chút (**O**ffene **S**ysteme und deren Schnittstellen für die **E**lektronik in **K**raftfahrzeugen), và có thể hiểu đơn giản theo tiếng anh là ***Open Systems and their Interfaces for the Electronics in Motor Vehicles***. Nó cung cấp một số đặc điểm kỹ thuật của hệ điều hành (OS), communication stack, và giao thức quản lý mạng được sử dụng trong các ứng dụng ô tô, được triển khai bởi một tập đoàn gộp nhiều công ty ô tô lớn của Đức như **BMW**, **Bosch**, **Opel**, **Siemens**, **Volkswagen**.

Dự án này sau đó có sự tham gia của một số công ty ô tô của Pháp như **Renault** và **PSA Peugeot**, những công ty có dự án tương tự gọi là **VDX (Vehicle Distributed Executive**). Vì vậy, cái tên **OSEK/VDX** được sử dụng kết hợp.

👉 Composition SWC

Composition là một nhóm các Software Component (SWC), được gán cho một ECU duy nhất trong System Configuration, tức là ECU này sẽ đảm nhiệm một số chức năng (của từng SWC)

trong application. Việc phân nhóm này giúp trừu tượng hóa các Software Component và tiêu chuẩn hóa quá trình phát triển phần mềm, đó là điều mà AUTOSAR hướng tới.

👉 SWC

Trong AUTOSAR, application được phân phối trong các SWC khác nhau. ***SWC - Software Component*** là một thành phần có logic ứng dụng, trong AUTOSAR, một chức năng sẽ được đóng gói bởi SWC.

Ví dụ. Hoạt động cửa cửa sổ điện trên ô tô, một SWC chuyên dụng sẽ được đóng gói để thực hiện chức năng này.

Các SWC giao tiếp với nhau, hoặc sử dụng các lớp thấp hơn bằng cách sử dụng các Ports, với sự trợ giúp của RTE.

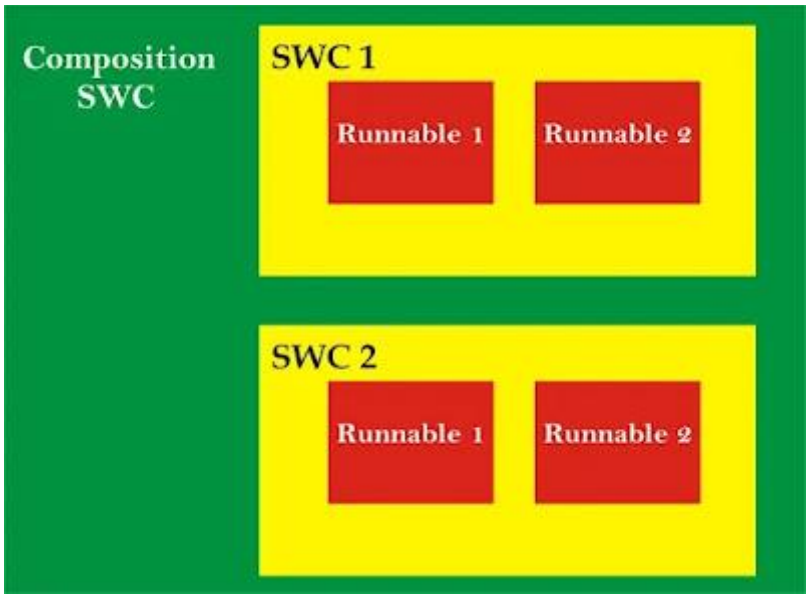
➤ AUTOSAR phân loại SWC dựa trên công dụng của nó:

- ***Application SWC.***
- ***SensorActuator SWC.*** Xử lý các **Cảm biến và thiết bị chấp hành**.
- ***Parameter SWC.*** Sử dụng để chia sẻ các tham số hiệu chuẩn của ECU với các thiết bị bên ngoài.
- ***Composition SWC.*** Đã nói ở trên.
- ***Service Proxy SWC.*** Hoạt động như một proxy, cung cấp dịch vụ nội bộ cho một hoặc nhiều ECU từ xa, phân phối thông tin chế độ hoạt động của xe cho toàn bộ hệ thống.
- ***Service SWC.*** Cung cấp các dịch vụ AUTOSAR cụ thể của **module BSW**.
- ***ECU Abstraction SWC.*** Cung cấp quyền truy cập vào các I/O bằng cách tương tác trực tiếp với các module BSW cụ thể (Chỉ có thể sử dụng SWC này để truy cập I/O).
- ***Complex Device Driver SWC.*** Được sử dụng để phát triển CDD cho các thiết bị bên ngoài mà AUTOSAR không hỗ trợ, hoặc một số hoạt động quan trọng.
- ***Nvblock SWC.*** Tương tác với NVRAM hoặc memory.

👉 Runnable Entity

Runnable Entity là một thành phần của SWC, nơi viết các logic hành vi của tầng Application. Có thể hiểu Runnable tương tự các function trong C. Trong AUTOSAR, Runnable được tạo ra trong một SWC trong quá trình config, và Runnable hay function đó được tạo ra trong các file source code của SWC.

Tên của function được người dùng cấu hình trong giao diện cấu hình, và sẽ được thực thi bởi AUTOSAR OS. Người dùng sẽ cần viết nội dung của các function này. Các function này có thể sử dụng khái niệm "***Trigger Point***" - tức là hàm sẽ được gọi khi thỏa mãn một số điều kiện nhất định.



Composition SWC and Runnables

➤ Có 3 loại Runnable:

- ***Init Runnable.*** Được gọi khi khởi tạo ECU.
- ***Periodic Runnable.*** Sử dụng khi cần trigger function này để thực thi một số hoạt động một cách có chu kỳ.
- ***Server Runnable.*** Sử dụng để triển khai server của giao diện port client/server.

Runnable có thể được cấu hình để hoạt động trên các sự kiện RTE như:

- ***Timing Event.*** Như giải thích ở trên, bất cứ khi nào đạt đến thời gian đã set từ trước, một sự kiện sẽ kích hoạt Periodic Runnable. Việc này có thể triển khai bằng các ngắt Timer.
- ***Data Received Event.*** Sự kiện được kích hoạt khi có data được nhận bởi Ports.
- ***Operation Invoked Event.*** Sự kiện được gọi bởi Client khi một Server có thể chạy được bằng
- ***Mode Switch Event.*** Bất cứ khi nào chế độ hoạt động của ECU thay đổi, Runnable có thể được trigger để hoạt động. Ví dụ. ECU ở chế độ shutdown, nếu ECU cần thực hiện một số hoạt động trước khi shutdown, sự kiện đó sẽ được nối với Runnable và thực hiện trước khi shutdown.
- ***Data Received Error Event.*** Xảy ra khi có bất kỳ lỗi xảy ra trong quá trình nhận dữ liệu.
- ***Data Send Completed Event.*** Runnable được trigger khi dữ liệu được gửi thành công.

👉 File MemMap

Phần này khá hay và dài nên mình sẽ giới thiệu riêng ở bài sau.

👉 Port and Port Interfaces

Trong AUTOSAR, mọi giao tiếp giữa SWC với các lớp thấp hơn được thực hiện bằng cách sử dụng **Ports**. Port là một channel hoặc connection sử dụng truyền nhận data giữa các SWC, hoặc các module BSW.

Như mục đích của AUTOSAR là tiêu chuẩn hóa, data sẽ được truyền nhận giữa các thực thể cần được biết đến tại thời điểm cấu hình, Port cũng không phải ngoại lệ.

Các Port thuộc về một SWC tại mỗi thời điểm. Port có thể hoặc không được kết nối với mỗi đầu. Có 2 loại Port:

- ***Required Port.*** Sử dụng khi dữ liệu được nhận hoặc được yêu cầu từ các thực thể khác.
- ***Provider Port.*** Sử dụng khi dữ liệu được truyền hoặc SWC là nơi cung cấp các dịch vụ đến các thực thể khác.

Port Interfaces là giao diện định nghĩa loại thông tin được truyền nhận giữa 2 Ports. Port Interfaces định nghĩa một protocol tuân theo port của SWC. Port Interface có thể được tái sử dụng bởi nhiều port khác nhau.

Port Interface được cấu hình tại thời điểm system configuration và port sẽ hoạt động theo port interface được cấu hình tại thời điểm đó.

➤ AUTOSAR có 3 loại **Port Interfaces**:

- ***AUTOSAR Interface.*** Một giao diện chung được tạo cho các Port của SWC, sử dụng để tương tác với các SWC khác, hoặc ECU Abstraction Layer.
- ***Standardized AUTOSAR Interface.*** Giao diện này được AUTOSAR định nghĩa trước, được ứng dụng SWC sử dụng khi tương tác với các **BSW Services** như ECU Manager, ...
- ***Standardized Interface.*** Giao diện này cũng được định nghĩa trước bởi AUTOSAR, nó giống như các API C, sử dụng giữa các module BSW, giữa RTE và OS, ...

👉 Hardware Objects / Hardware Object Handle

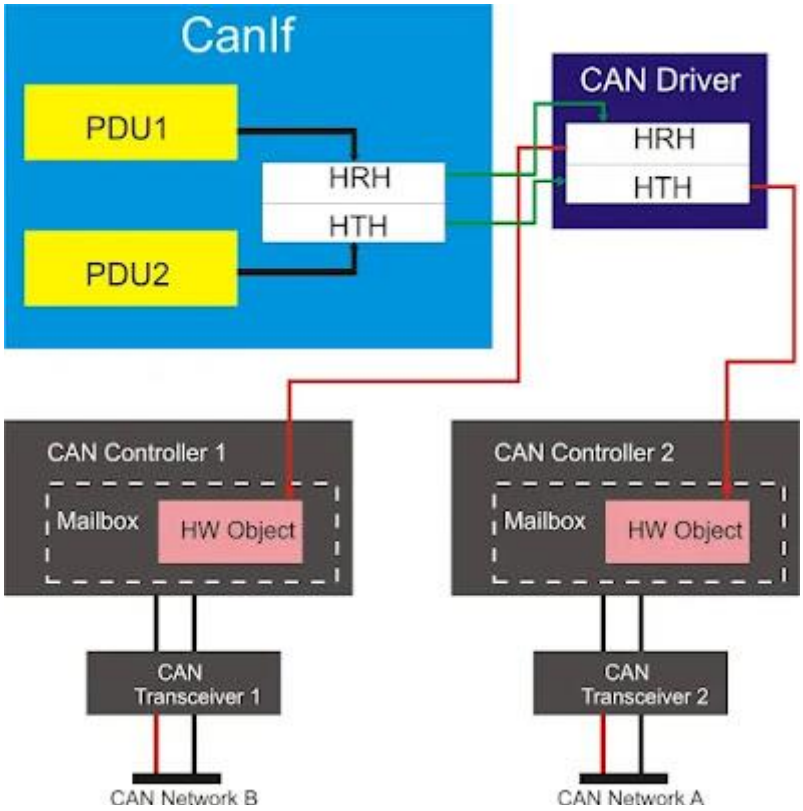
Thuật ngữ Hardware Object được sử dụng liên quan đến CAN Bus, nó là một vùng nhớ thuộc RAM của CAN Controller, nơi chứa PDU. Khi PDU nằm trong RAM của CAN Controller, nó được gọi là ***Harware Object***.

HOH - Hardware Object Handle đại diện cho một tham chiếu trừu tượng của CAN Mailbox, nơi có tất cả tham số của CAN Frame như DLC, CANID, CAN Data. Các lớp trên không thể trực tiếp tạo ra CAN Frame với data và điều này mâu thuẫn với mục tiêu độc lập phần cứng của AUTOSAR, thay vào đó, một tham chiếu trừu tượng được sử dụng để đảm bảo sự độc lập phần cứng này.

Có 2 loại HOH:

- ***HTH - Hardware Transmit Handle.*** Sử dụng trong quá trình truyền của CAN Frame.
- ***HRH - Hardware Receive Handle.*** Sử dụng trong quá trình nhận của CAN Frame.

HOH được sử dụng bởi **CanIf** và được tham chiếu dựa trên bộ đệm phần cứng CAN. HOH được sử dụng làm parameter khi gọi tầng thấp hơn - CAN Driver.



Hardware Object Handle references in different modules

Trên đây là một số khái niệm/thuật ngữ cơ bản trong AUTOSAR mà mình tìm hiểu được, thực tế còn rất nhiều thuật ngữ khác nữa. Nhưng trong phạm vi bài viết có giới hạn nên chỉ giới thiệu được những thuật ngữ chính và có thể chưa dễ hiểu lắm. Các bạn có thể search các keyword trên Internet để có thể tìm hiểu sâu hơn.