01.GIỚI THIỆU VỀ AUTOSAR

1. Giới thiệu về AUTOSAR (AUTomotive Open System ARchitecture)

AUTOSAR là một tiêu chuẩn toàn cầu cho kiến trúc phần mềm trong ngành công nghiệp ô tô, được phát triển bởi một liên minh các công ty ô tô hàng đầu như BMW, Bosch, Continental, Daimler, Ford, GM, PSA, Toyota, và Volkswagen. Mục tiêu của AUTOSAR là cung cấp một nền tảng tiêu chuẩn hóa giúp các nhà sản xuất và nhà cung cấp phần mềm phát triển các hệ thống điều khiển điện tử (ECU) một cách hiệu quả và có tính tương thích cao.

Lý do cần có AUTOSAR

Trước đây, mỗi hãng xe thường tự phát triển hệ thống điều khiển của riêng mình, dẫn đến việc phần mềm và phần cứng của các hãng không thể tương thích với nhau. Điều này giống như việc bạn phải mua một bộ sạc riêng cho mỗi thiết bị điện tử mà bạn sở hữu, dẫn đến sự phức tạp và tốn kém khi cần thay thế hoặc nâng cấp.

AUTOSAR ra đời để giải quyết vấn đề này bằng cách tạo ra một tiêu chuẩn chung cho kiến trúc phần mềm, giúp tách biệt phần mềm và phần cứng. Điều này giúp việc thay đổi hoặc nâng cấp một phần của hệ thống trở nên dễ dàng và ít tốn kém hơn.

2. Các thành phần chính của AUTOSAR

AUTOSAR được chia thành hai nền tảng chính:

Classic Platform

Classic Platform là nền tảng ban đầu của AUTOSAR, được thiết kế cho các hệ thống điều khiển thời gian thực có yêu cầu cao về độ an toàn và tin cậy. Nền tảng này chủ yếu được sử dụng trong các ứng dụng như hệ thống điều khiển động cơ, hệ thống phanh ABS, và các hệ thống an toàn khác.

Các tính năng của Classic Platform:

- **An toàn (Safety)**: Hỗ trợ các tiêu chuẩn an toàn từ QM (Quality Management) đến các mức độ an toàn cao hơn như ASIL (Automotive Safety Integrity Level) từ A đến D.
- **Hiệu quả chi phí (Cost Efficiency)**: Giảm chi phí phát triển và bảo trì bằng cách sử dụng các module phần mềm tiêu chuẩn.
- **Độ tin cậy (Reliability)**: Đã được sử dụng và kiểm chứng qua nhiều năm trong ngành công nghiệp ô tô.
- **Hiệu suất cao (High Performance)**: Hỗ trợ các ứng dụng yêu cầu thời gian thực và hiệu suất cao.

Các layer của Classic Platform bao gồm:

- **Application Layer**: Chứa các ứng dụng cụ thể của ô tô, như hệ thống điều khiển động cơ, hệ thống phanh ABS.
- **Run-Time Environment (RTE)**: Là môi trường chạy thực thi giúp kết nối giữa Application Layer và Basic Software.
- **Basic Software (BSW)**: Gồm các module phần mềm cơ bản hỗ trợ các ứng dụng, như drivers cho các thiết bị phần cứng.
- **Microcontroller Abstraction Layer (MCAL)**: Tầng trừu tượng hóa vi điều khiển, giúp phần mềm tương tác với phần cứng một cách độc lập.

Các lớp trong Classic Platform

1. **Application Layer**
    - Là tầng chứa các ứng dụng cụ thể của hệ thống, ví dụ như hệ thống kiểm soát động cơ hoặc hệ thống điều khiển phanh ABS.
    - Các ứng dụng này thực hiện các chức năng cụ thể, giống như các ứng dụng trên máy tính của bạn, ví dụ như trình duyệt web hoặc phần mềm xử lý văn bản.
2. **Run-Time Environment (RTE)**
    - RTE là môi trường chạy thực thi kết nối giữa Application Layer và Basic Software.
    - Nó hoạt động như một người phiên dịch giữa hai tầng, đảm bảo rằng các ứng dụng có thể giao tiếp với các phần mềm cơ bản một cách mượt mà.
3. **Basic Software (BSW)**
    - BSW bao gồm các module phần mềm cơ bản hỗ trợ các ứng dụng, như drivers cho các thiết bị phần cứng.
    - Tương tự như các driver trên máy tính của bạn, BSW giúp phần mềm giao tiếp và điều khiển các thiết bị phần cứng.
4. **Microcontroller Abstraction Layer (MCAL)**
    - MCAL là tầng trừu tượng hóa vi điều khiển, giúp phần mềm tương tác với phần cứng một cách độc lập.
    - Điều này giống như việc bạn có thể sử dụng cùng một ứng dụng trên các loại máy tính khác nhau, miễn là có driver phù hợp.

Adaptive Platform

Adaptive Platform được thiết kế để hỗ trợ các tính năng mới hơn và linh hoạt hơn, như kết nối giữa các ECUs và khả năng cập nhật phần mềm qua mạng. Nền tảng này chủ yếu được sử dụng trong các ứng dụng hiện đại như hệ thống hỗ trợ lái xe tự động (ADAS), hệ thống giải trí trên xe, và các dịch vụ kết nối internet.

Các tính năng của Adaptive Platform:

- **An toàn và bảo mật (Safety and Security)**: Hỗ trợ các tiêu chuẩn an toàn và bảo mật hiện đại.
- **Khả năng kết nối (Connectivity)**: Hỗ trợ kết nối giữa các ECU và giữa các xe với nhau, ví dụ như giao tiếp giữa các xe để chia sẻ thông tin về tình hình giao thông.

- **Khả năng nâng cấp (Upgradeability)**: Cho phép cập nhật phần mềm dễ dàng qua mạng, giống như cập nhật hệ điều hành trên máy tính.

Các layer của Adaptive Platform bao gồm:

- **Application Layer**: Chứa các ứng dụng cụ thể, giống như trong Classic Platform.
- **Service-Oriented Middleware (SOM)**: Là tầng trung gian hỗ trợ các dịch vụ kết nối và truyền thông giữa các ứng dụng.
- **Basic Software (BSW)**: Tương tự như trong Classic Platform nhưng được tối ưu hóa cho các yêu cầu mới.

Các lớp trong Adaptive Platform

1. **Application Layer**
    - Chứa các ứng dụng cụ thể của hệ thống, như hệ thống hỗ trợ lái xe tự động (ADAS) hoặc hệ thống giải trí trên xe.
    - Các ứng dụng này có thể được cập nhật và thay đổi dễ dàng để phù hợp với các yêu cầu mới nhất.
2. **Service-Oriented Middleware (SOM)**
    - SOM là tầng trung gian hỗ trợ các dịch vụ kết nối và truyền thông giữa các ứng dụng.
    - Nó giống như một bộ điều phối, đảm bảo rằng các ứng dụng có thể giao tiếp với nhau và với các thiết bị bên ngoài một cách hiệu quả.
3. **Basic Software (BSW)**
    - Tương tự như trong Classic Platform nhưng được tối ưu hóa cho các yêu cầu mới.
    - Bao gồm các module phần mềm cơ bản như drivers cho các thiết bị phần cứng và các dịch vụ hệ thống.

3. Tầm quan trọng của AUTOSAR

AUTOSAR có tầm quan trọng lớn trong ngành công nghiệp ô tô vì những lý do sau:

1. **Tiêu chuẩn hóa**: AUTOSAR cung cấp một tiêu chuẩn chung cho kiến trúc phần mềm, giúp các nhà sản xuất và nhà cung cấp phần mềm phát triển các hệ thống một cách tương thích và hiệu quả.
2. **Tính linh hoạt**: Bằng cách tách biệt phần mềm và phần cứng, AUTOSAR giúp việc thay đổi hoặc nâng cấp các thành phần của hệ thống trở nên dễ dàng và ít tốn kém hơn.
3. **An toàn và bảo mật**: AUTOSAR hỗ trợ các tiêu chuẩn an toàn và bảo mật cao, giúp đảm bảo rằng các hệ thống điều khiển trong ô tô hoạt động một cách an toàn và bảo mật.
4. **Hiệu quả chi phí**: Sử dụng các module phần mềm tiêu chuẩn giúp giảm chi phí phát triển và bảo trì, đồng thời tăng hiệu quả hoạt động của hệ thống.

4. Các đối tác của AUTOSAR

AUTOSAR được phát triển và duy trì bởi một nhóm các công ty hàng đầu trong ngành công nghiệp ô tô, bao gồm các nhà sản xuất ô tô lớn và các nhà cung cấp phần mềm và dịch vụ. Các đối tác chính của AUTOSAR bao gồm:

1. **Core Partners**: BMW, Bosch, Continental, Daimler, Ford, GM, PSA, Toyota, Volkswagen.
    - Những công ty này đã thành lập liên minh AUTOSAR và đóng vai trò chủ chốt trong việc phát triển và duy trì tiêu chuẩn.
2. **Premium Partners**: Hơn 65 công ty, bao gồm cả các nhà sản xuất ô tô và các nhà cung cấp phần mềm và dịch vụ hàng đầu.
3. **Development Partners**: Hơn 80 công ty khác tham gia vào việc phát triển và ứng dụng tiêu chuẩn AUTOSAR.

5. Nền tảng phát triển của AUTOSAR

Hiện tại, AUTOSAR phát triển trên hai nền tảng chính:

Classic Platform

Classic Platform là nền tảng đầu tiên và chủ yếu của AUTOSAR, được sử dụng rộng rãi trong các hệ thống điều khiển điện tử (ECU) trong ô tô. Nó bao gồm các lớp sau:

- **Application Layer**: Chứa các ứng dụng cụ thể của hệ thống, như hệ thống kiểm soát động cơ hoặc hệ thống điều khiển phanh ABS.
- **Run-Time Environment (RTE)**: Là môi trường chạy thực thi kết nối giữa Application Layer và Basic Software.
- **Basic Software (BSW)**: Gồm các module phần mềm cơ bản hỗ trợ các ứng dụng, như drivers cho các thiết bị phần cứng.
- **Microcontroller Abstraction Layer (MCAL)**: Tầng trừu tượng hóa vi điều khiển, giúp phần mềm tương tác với phần cứng một cách độc lập.

Adaptive Platform

Adaptive Platform là nền tảng mới hơn của AUTOSAR, được thiết kế để hỗ trợ các tính năng hiện đại và linh hoạt hơn. Nó bao gồm các lớp sau:

- **Application Layer**: Chứa các ứng dụng cụ thể của hệ thống, như hệ thống hỗ trợ lái xe tự động (ADAS) hoặc hệ thống giải trí trên xe.
- **Service-Oriented Middleware (SOM)**: Là tầng trung gian hỗ trợ các dịch vụ kết nối và truyền thông giữa các ứng dụng.
- **Basic Software (BSW)**: Tương tự như trong Classic Platform nhưng được tối ưu hóa cho các yêu cầu mới.

6. Tầm nhìn và sứ mệnh của AUTOSAR

Sứ mệnh (Mission)

AUTOSAR hợp tác toàn cầu giữa các công ty hàng đầu trong ngành công nghệ ô tô và phần mềm để phát triển và thiết lập các phần mềm được tiêu chuẩn hóa theo kiến trúc hệ thống điện tử và điện tử.

Tầm nhìn (Vision)

AUTOSAR đặt mục tiêu trở thành tiêu chuẩn thiết lập toàn cầu cho phần mềm và phương pháp phát triển hệ thống điện tử di động thông minh trong tương lai, hỗ trợ độ tin cậy cao và đảm bảo an toàn và bảo mật.

Tổng kết

AUTOSAR là một tiêu chuẩn quan trọng trong ngành công nghiệp ô tô, giúp tăng cường tính linh hoạt, hiệu quả và an toàn của các hệ thống điện tử trong xe hơi. Nó cung cấp một nền tảng tiêu chuẩn giúp các công ty phát triển phần mềm và phần cứng một cách độc lập nhưng vẫn đảm bảo sự tương thích và hoạt động hiệu quả. Với sự hỗ trợ của các đối tác hàng đầu trong ngành, AUTOSAR đang ngày càng trở thành một tiêu chuẩn quan trọng và không thể thiếu trong việc phát triển các hệ thống điều khiển điện tử cho ô tô.

02AUTOSAR ARCHITECTURE LAYER VÀ MCAL LAYER

AUTOSAR Architecture được thiết kế để giúp các nhà phát triển phần mềm ô tô xây dựng các hệ thống phức tạp một cách có tổ chức và dễ quản lý. AUTOSAR chia kiến trúc phần mềm của hệ thống điều khiển điện tử (ECU) thành các lớp rõ ràng, mỗi lớp có một vai trò cụ thể và tách biệt. Điều này giúp tối ưu hóa sự phát triển, bảo trì và nâng cấp hệ thống. Chúng ta sẽ đi vào chi tiết từng lớp trong kiến trúc AUTOSAR, bao gồm:

1. Application Layer
2. Runtime Environment (RTE)
3. Basic Software (BSW)
4. Microcontroller Abstraction Layer (MCAL)

1. Application Layer

Định nghĩa

Application Layer là lớp cao nhất trong kiến trúc AUTOSAR và chứa các ứng dụng cụ thể của hệ thống ô tô. Đây là nơi các chức năng điều khiển chính được thực hiện.

Đặc điểm

- **Không phụ thuộc vào phần cứng**: Ứng dụng trong lớp này không tương tác trực tiếp với phần cứng mà thông qua các tầng dưới.
- **Chứa các chức năng cụ thể**: Ví dụ, hệ thống kiểm soát động cơ (Engine Control Unit - ECU), hệ thống phanh ABS (Anti-lock Braking System).

Ví dụ thực tế

Hãy tưởng tượng Application Layer như các ứng dụng trên điện thoại di động của bạn, như trình duyệt web hay ứng dụng nhắn tin. Các ứng dụng này sử dụng các dịch vụ của hệ điều hành (Runtime Environment) để tương tác với phần cứng (CPU, bộ nhớ).

2. Runtime Environment (RTE)

Định nghĩa

Runtime Environment (RTE) là lớp trung gian giữa Application Layer và Basic Software. Nó cung cấp một môi trường chạy thực thi và dịch vụ giao tiếp để các ứng dụng có thể tương tác với phần mềm cơ bản.

Đặc điểm

- **Kết nối các lớp**: RTE là cầu nối giữa Application Layer và Basic Software, giúp các ứng dụng có thể sử dụng các dịch vụ của Basic Software mà không cần biết chi tiết phần cứng.

- **Tách biệt ứng dụng và phần mềm cơ bản**: Giúp ứng dụng không bị phụ thuộc vào các thay đổi của Basic Software.

Ví dụ thực tế

Hãy tưởng tượng RTE như một hệ điều hành trên máy tính của bạn. Nó giúp các ứng dụng (như trình duyệt web) có thể sử dụng các dịch vụ cơ bản (như quản lý tệp tin, giao tiếp mạng) mà không cần phải tự mình xử lý các chi tiết này.

3. Basic Software (BSW)

Định nghĩa

Basic Software là tầng phần mềm cơ bản, cung cấp các dịch vụ cần thiết cho các ứng dụng trong Application Layer. Nó bao gồm các module phần mềm được chia thành ba lớp chính: System Services, ECU Abstraction, và Microcontroller Drivers.

Các lớp của Basic Software

1. **System Services Layer**
    - **Communication Services**: Quản lý giao tiếp giữa các ECU và các thiết bị ngoại vi.
    - **System Services**: Cung cấp các dịch vụ hệ thống như quản lý bộ nhớ, quản lý thời gian thực, và các dịch vụ an ninh.
    - **Memory Services**: Quản lý các hoạt động liên quan đến bộ nhớ như lưu trữ dữ liệu và truy xuất dữ liệu.
2. **ECU Abstraction Layer**
    - **Onboard Device Abstraction**: Trừu tượng hóa các thiết bị onboard, giúp các ứng dụng tương tác với các thiết bị này một cách dễ dàng.
    - **Hardware Abstraction**: Trừu tượng hóa phần cứng, giúp phần mềm có thể sử dụng các dịch vụ của phần cứng mà không cần biết chi tiết về cấu trúc phần cứng.
3. **Microcontroller Drivers**
    - **Microcontroller Abstraction Layer (MCAL)**: Tầng trừu tượng hóa vi điều khiển, cung cấp các driver trực tiếp để tương tác với phần cứng vi điều khiển.
    - **Peripheral Drivers**: Các driver cho các thiết bị ngoại vi như giao tiếp CAN, LIN, và các bộ điều khiển khác.

Ví dụ thực tế

Hãy tưởng tượng Basic Software như các thư viện và driver trong hệ điều hành của bạn. Các thư viện này cung cấp các hàm và dịch vụ cần thiết để các ứng dụng có thể hoạt động. Ví dụ, thư viện giao tiếp mạng giúp ứng dụng web có thể gửi và nhận dữ liệu qua internet.

4. Microcontroller Abstraction Layer (MCAL)

Định nghĩa

MCAL là tầng trừu tượng hóa vi điều khiển, cung cấp các driver để tương tác trực tiếp với phần cứng vi điều khiển. Đây là tầng thấp nhất trong kiến trúc AUTOSAR.

Đặc điểm

- **Tương tác trực tiếp với phần cứng**: MCAL cung cấp các hàm API để truy cập trực tiếp vào các thanh ghi của vi điều khiển.
- **Tách biệt phần mềm và phần cứng**: Giúp các tầng phần mềm cao hơn không cần biết chi tiết về phần cứng.

Các module của MCAL

1. **Microcontroller Drivers**: Quản lý các chức năng cơ bản của vi điều khiển như quản lý thời gian, watchdog, và quản lý ngắt.
2. **Memory Drivers**: Quản lý các thiết bị bộ nhớ như Flash và RAM.
3. **Communication Drivers**: Quản lý giao tiếp với các thiết bị ngoại vi như CAN, LIN, và Ethernet.
4. **IO Drivers**: Quản lý các thiết bị vào/ra như ADC, GPIO.

Ví dụ thực tế

Hãy tưởng tượng MCAL như các driver phần cứng trên máy tính của bạn, như driver card mạng hoặc driver card đồ họa. Các driver này giúp hệ điều hành và các ứng dụng có thể sử dụng các thiết bị phần cứng mà không cần biết chi tiết về cách chúng hoạt động.

Tổng kết về AUTOSAR Architecture Layer

AUTOSAR Architecture Layer được chia thành các lớp rõ ràng, mỗi lớp có một vai trò cụ thể và tách biệt. Điều này giúp tối ưu hóa sự phát triển, bảo trì và nâng cấp hệ thống. Các lớp trong kiến trúc AUTOSAR bao gồm:

1. **Application Layer**: Chứa các ứng dụng cụ thể của hệ thống, không phụ thuộc vào phần cứng.
2. **Runtime Environment (RTE)**: Cầu nối giữa Application Layer và Basic Software, cung cấp môi trường chạy thực thi và dịch vụ giao tiếp.
3. **Basic Software (BSW)**: Tầng phần mềm cơ bản, cung cấp các dịch vụ cần thiết cho các ứng dụng trong Application Layer.
4. **Microcontroller Abstraction Layer (MCAL)**: Tầng trừu tượng hóa vi điều khiển, cung cấp các driver để tương tác trực tiếp với phần cứng vi điều khiển.

Đào sâu vào Basic Software (BSW)

Basic Software (BSW) là một trong những phần phức tạp nhất và quan trọng nhất của AUTOSAR. Nó được chia thành nhiều module và mỗi module có một vai trò cụ thể. Hãy cùng đi sâu vào chi tiết của từng lớp trong Basic Software.

1. System Services Layer

System Services Layer là lớp cung cấp các dịch vụ hệ thống chung, giúp các ứng dụng có thể hoạt động một cách hiệu quả và an toàn.

1. **Communication Services**
    - **CAN, LIN, FlexRay, Ethernet**: Các giao thức truyền thông khác nhau được sử dụng để giao tiếp giữa các ECU và giữa ECU với các thiết bị ngoại vi.
    - **Network Management**: Quản lý trạng thái của mạng, bao gồm khởi động, tắt và quản lý lỗi.
2. **System Services**
    - **OS (Operating System)**: Quản lý các nhiệm vụ, tài nguyên và dịch vụ thời gian thực.
    - **Watchdog**: Giám sát và khởi động lại hệ thống khi xảy ra lỗi.
    - **Diagnostics**: Thu thập và phân tích dữ liệu chẩn đoán từ các ECU để phát hiện và xử lý lỗi.
3. **Memory Services**
    - **NVRAM Management**: Quản lý bộ nhớ không bay hơi (Non-Volatile RAM) để lưu trữ dữ liệu quan trọng.
    - **Flash EEPROM Emulation**: Giả lập EEPROM trên bộ nhớ Flash để lưu trữ các cấu hình và dữ liệu không thay đổi thường xuyên.

2. ECU Abstraction Layer

ECU Abstraction Layer cung cấp một tầng trừu tượng hóa giữa phần cứng và phần mềm, giúp các ứng dụng có thể tương tác với phần cứng một cách dễ dàng và độc lập.

1. **Onboard Device Abstraction**
    - **ADC (Analog-to-Digital Converter)**: Chuyển đổi tín hiệu analog từ các cảm biến sang tín hiệu số để ECU có thể xử lý.
    - **PWM (Pulse Width Modulation)**: Điều khiển các thiết bị như động cơ và đèn thông qua tín hiệu PWM.
2. **Hardware Abstraction**
    - **I/O Hardware Abstraction**: Quản lý các thiết bị vào/ra như công tắc, đèn và cảm biến.
    - **Timer**: Quản lý các bộ định thời để hỗ trợ các tác vụ cần thời gian chính xác.

3. Microcontroller Drivers

Microcontroller Drivers cung cấp các driver để tương tác trực tiếp với phần cứng vi điều khiển. Các driver này bao gồm:

1. **Microcontroller Abstraction Layer (MCAL)**
    - **MCU Driver**: Quản lý các chức năng cơ bản của vi điều khiển như quản lý thời gian, watchdog và quản lý ngắt.
    - **Memory Driver**: Quản lý các thiết bị bộ nhớ như Flash và RAM.

2. **Peripheral Drivers**
    - **Communication Drivers**: Quản lý giao tiếp với các thiết bị ngoại vi như CAN, LIN và Ethernet.
    - **IO Drivers**: Quản lý các thiết bị vào/ra như ADC, GPIO.

Đào sâu vào Microcontroller Abstraction Layer (MCAL)

MCAL là lớp trừu tượng hóa vi điều khiển, cung cấp các driver để tương tác trực tiếp với phần cứng vi điều khiển. Nó giúp phần mềm có thể sử dụng các dịch vụ của phần cứng mà không cần biết chi tiết về cấu trúc phần cứng.

Các module của MCAL

1. **Microcontroller Drivers**
    - **MCU Driver**: Quản lý các chức năng cơ bản của vi điều khiển như quản lý thời gian, watchdog và quản lý ngắt.
    - **Timer Driver**: Quản lý các bộ định thời để hỗ trợ các tác vụ cần thời gian chính xác.
2. **Memory Drivers**
    - **Flash Driver**: Quản lý bộ nhớ Flash để lưu trữ các cấu hình và dữ liệu quan trọng.
    - **EEPROM Driver**: Quản lý bộ nhớ EEPROM để lưu trữ các cấu hình và dữ liệu không thay đổi thường xuyên.
3. **Communication Drivers**
    - **CAN Driver**: Quản lý giao tiếp CAN để truyền dữ liệu giữa các ECU.
    - **LIN Driver**: Quản lý giao tiếp LIN để truyền dữ liệu giữa các ECU và các thiết bị ngoại vi.
4. **IO Drivers**
    - **ADC Driver**: Chuyển đổi tín hiệu analog từ các cảm biến sang tín hiệu số để ECU có thể xử lý.
    - **PWM Driver**: Điều khiển các thiết bị như động cơ và đèn thông qua tín hiệu PWM.

Tổng kết về Microcontroller Abstraction Layer (MCAL)

MCAL là một trong những lớp quan trọng nhất của AUTOSAR, cung cấp các driver để tương tác trực tiếp với phần cứng vi điều khiển. Điều này giúp phần mềm có thể sử dụng các dịch vụ của phần cứng một cách dễ dàng và hiệu quả.

03.ECU ABSTRACTION LAYER

ECU Abstraction Layer (ECU AL) là một phần của kiến trúc phần mềm AUTOSAR, nằm trên MCAL (Microcontroller Abstraction Layer). Mục tiêu của lớp này là cung cấp một tầng trừu tượng hóa giúp tách biệt phần cứng và phần mềm, giúp phần mềm ứng dụng có thể hoạt động trên nhiều nền tảng phần cứng khác nhau mà không cần thay đổi lớn.

Chức năng của ECU Abstraction Layer

ECU Abstraction Layer đóng vai trò là cầu nối giữa các driver của MCAL và các lớp phần mềm bên trên. Nó giúp phần mềm ứng dụng và các dịch vụ hệ thống sử dụng các chức năng phần cứng một cách dễ dàng mà không cần phải biết chi tiết về phần cứng cụ thể. Điều này giúp giảm sự phụ thuộc của phần mềm vào phần cứng, tăng tính linh hoạt và khả năng bảo trì của hệ thống.

Các thành phần của ECU Abstraction Layer

ECU Abstraction Layer được chia thành nhiều nhóm module khác nhau, mỗi nhóm phục vụ một mục đích cụ thể và tương tác với một phần của MCAL. Chúng ta sẽ đi sâu vào từng nhóm module của lớp này.

1. Onboard Device Abstraction

Nhóm module này chứa các driver cho các thiết bị trên bo mạch (onboard) không phải là cảm biến hay bộ truyền động (actuators).

- **Watchdog Interface**: Driver quản lý watchdog timer, giúp hệ thống khởi động lại khi phát hiện lỗi.
- **DIO Interface**: Driver quản lý các tín hiệu vào/ra số (Digital Input/Output).

2. Memory Hardware Abstraction

Nhóm module này quản lý các thiết bị bộ nhớ, bao gồm bộ nhớ trong và bộ nhớ ngoài của vi điều khiển.

- **Memory Interface**: Quản lý các thao tác đọc/ghi dữ liệu từ/đến bộ nhớ.
- **Flash Driver**: Driver quản lý bộ nhớ Flash, lưu trữ các chương trình và dữ liệu quan trọng.
- **EEPROM Driver**: Driver quản lý bộ nhớ EEPROM, lưu trữ các cấu hình và dữ liệu không thay đổi thường xuyên.

3. Crypto Hardware Abstraction

Nhóm module này quản lý các chức năng mã hóa và giải mã, đảm bảo an toàn cho dữ liệu và các giao tiếp của hệ thống.

- **Crypto Interface**: Cung cấp các hàm API cho các chức năng mã hóa và giải mã.

- **Crypto Driver**: Driver quản lý các thiết bị mã hóa trên phần cứng.

4. Communication Hardware Abstraction

Nhóm module này quản lý giao tiếp giữa các ECU và giữa ECU với các thiết bị ngoại vi.

- **CAN Interface**: Driver quản lý giao tiếp CAN (Controller Area Network).
- **LIN Interface**: Driver quản lý giao tiếp LIN (Local Interconnect Network).
- **Ethernet Interface**: Driver quản lý giao tiếp Ethernet.

5. Wireless Communication Hardware Abstraction

Nhóm module này quản lý giao tiếp không dây giữa các thiết bị.

- **Wireless Interface**: Driver quản lý các giao tiếp không dây như Bluetooth, Wi-Fi.

6. I/O Hardware Abstraction

Nhóm module này quản lý các tín hiệu vào/ra của hệ thống.

- **ADC (Analog-to-Digital Converter) Driver**: Chuyển đổi tín hiệu analog từ các cảm biến sang tín hiệu số.
- **PWM (Pulse Width Modulation) Driver**: Điều khiển các thiết bị như động cơ và đèn thông qua tín hiệu PWM.
- **DIO (Digital Input/Output) Driver**: Quản lý các tín hiệu vào/ra số.

Mối quan hệ giữa MCAL và ECU Abstraction Layer

MCAL cung cấp các driver để tương tác trực tiếp với phần cứng, còn ECU Abstraction Layer trừu tượng hóa các driver này để các lớp phần mềm bên trên có thể sử dụng một cách dễ dàng và độc lập với phần cứng.

Ví dụ về ADC (Analog-to-Digital Converter)

1. **MCAL Layer**:
    - **ADC Driver**: Tương tác trực tiếp với thanh ghi của vi điều khiển để đọc giá trị analog và chuyển đổi thành giá trị số.
2. **ECU Abstraction Layer**:
    - **ADC Interface**: Cung cấp hàm API cho các lớp phần mềm bên trên để đọc giá trị số từ ADC mà không cần biết chi tiết về cách ADC hoạt động.

Quá trình xử lý tín hiệu từ cảm biến nhiệt độ

1. **MCAL Layer**:
    - ADC Driver đọc giá trị analog từ cảm biến nhiệt độ và chuyển đổi thành giá trị số.
2. **ECU Abstraction Layer**:

    - ADC Interface cung cấp giá trị số này cho các lớp phần mềm bên trên.
    - Các lớp phần mềm bên trên có thể sử dụng giá trị này để thực hiện các tính toán và điều khiển cần thiết.

Tại sao không có lớp IO Hardware Service

Lớp IO Hardware Service không tồn tại vì tín hiệu từ các thiết bị IO thường được xử lý trực tiếp và không cần qua một lớp dịch vụ trung gian. Điều này giúp giảm độ trễ và tăng hiệu quả xử lý tín hiệu.

Lý do

- **Tín hiệu IO cần phản ứng nhanh**: Các tín hiệu từ cảm biến và bộ truyền động thường yêu cầu phản ứng nhanh và liên tục, do đó việc thêm một lớp dịch vụ trung gian có thể gây ra độ trễ không cần thiết.
- **Tránh sự phụ thuộc vào phần cứng cụ thể**: Mỗi cảm biến và bộ truyền động có các yêu cầu và chức năng riêng, do đó không thể xây dựng một lớp dịch vụ chung cho tất cả các thiết bị IO.

Kết luận

ECU Abstraction Layer là một phần quan trọng của kiến trúc AUTOSAR, giúp tách biệt phần cứng và phần mềm, tăng tính linh hoạt và khả năng bảo trì của hệ thống. Lớp này cung cấp các driver và hàm API cho các lớp phần mềm bên trên, giúp chúng có thể sử dụng các chức năng của phần cứng một cách dễ dàng mà không cần biết chi tiết về phần cứng.

Tiếp theo

Trong phần tiếp theo, chúng ta sẽ đi sâu vào lớp Basic Software Service Layer, bao gồm các dịch vụ hệ thống chung, dịch vụ giao tiếp và quản lý bộ nhớ, giúp hệ thống hoạt động một cách hiệu quả và an toàn.

04.BSW SERVICES LAYER VÀ CDD LAYER

Basic Software (BSW) Service Layer là lớp cao nhất trong Basic Software của AUTOSAR. Lớp này cung cấp các dịch vụ cần thiết cho các lớp phần mềm khác, như Runtime Environment (RTE) và Application Layer. Mục tiêu của BSW Service Layer là cung cấp các dịch vụ hệ thống và các dịch vụ phần mềm cơ bản để đảm bảo hệ thống hoạt động một cách hiệu quả và an toàn.

Các thành phần chính của BSW Service Layer

1. **System Services**
2. **Memory Services**
3. **Crypto Services**
4. **Onboard Communication Services**
5. **Communication Services**

1. System Services

System Services là nhóm các module và chức năng cung cấp các dịch vụ hệ thống cơ bản cho toàn bộ hệ thống phần mềm. Chúng bao gồm các dịch vụ như quản lý hệ điều hành, quản lý nguồn lực và giám sát hệ thống.

- **Operating System (OS)**: Quản lý các nhiệm vụ, tài nguyên và dịch vụ thời gian thực. OS cần có timer từ vi điều khiển để hoạt động.
- **Watchdog Manager**: Quản lý Watchdog Timer, giúp hệ thống khởi động lại khi phát hiện lỗi.
- **Diagnostic Event Manager (DEM)**: Thu thập và phân tích dữ liệu chẩn đoán từ các ECU để phát hiện và xử lý lỗi.
- **ECU State Manager**: Quản lý trạng thái của ECU, bao gồm việc khởi động, tắt và chuyển đổi trạng thái.

2. Memory Services

Memory Services quản lý các hoạt động liên quan đến bộ nhớ như lưu trữ, truy xuất và bảo vệ dữ liệu.

- **NVRAM Manager (NvM)**: Quản lý dữ liệu không bay hơi (Non-Volatile RAM) để lưu trữ dữ liệu quan trọng. Nó cung cấp cơ chế lưu trữ đáng tin cậy, bảo vệ dữ liệu và xác minh tính toàn vẹn của dữ liệu.
- **Flash EEPROM Emulation (Fee)**: Giả lập EEPROM trên bộ nhớ Flash để lưu trữ các cấu hình và dữ liệu không thay đổi thường xuyên.
- **Memory Interface (MemIf)**: Cung cấp các hàm API để truy cập bộ nhớ một cách thống nhất, bất kể vị trí vật lý của bộ nhớ.

3. Crypto Services

Crypto Services quản lý các chức năng mã hóa và giải mã, đảm bảo an toàn cho dữ liệu và các giao tiếp của hệ thống.

- **Crypto Service Manager (CrySM)**: Quản lý các hoạt động liên quan đến mã hóa, lưu trữ và sử dụng các khóa mã hóa.
- **Crypto Interface (CryIf)**: Cung cấp các hàm API cho các lớp phần mềm bên trên để thực hiện các chức năng mã hóa và giải mã.

4. Onboard Communication Services

Onboard Communication Services quản lý giao tiếp nội bộ giữa các module phần mềm trong cùng một ECU.

- **Inter-ECU Communication (I-ECU)**: Quản lý giao tiếp giữa các module phần mềm trong cùng một ECU, bao gồm việc truyền và nhận dữ liệu.
- **Signal Gateway (SigGW)**: Chuyển tiếp các tín hiệu giữa các module phần mềm và đảm bảo dữ liệu được truyền tải một cách chính xác.

5. Communication Services

Communication Services quản lý giao tiếp giữa các ECU và giữa ECU với các thiết bị ngoại vi.

- **CAN Communication Service (CanCom)**: Quản lý giao tiếp CAN (Controller Area Network).
- **LIN Communication Service (LinCom)**: Quản lý giao tiếp LIN (Local Interconnect Network).
- **Ethernet Communication Service (EthCom)**: Quản lý giao tiếp Ethernet.
- **FlexRay Communication Service (FrCom)**: Quản lý giao tiếp FlexRay.

CDD Layer (Complex Device Driver Layer)

CDD (Complex Device Driver) là một lớp đặc biệt trong Basic Software của AUTOSAR. Nó cung cấp các driver phức tạp cho các thiết bị cụ thể mà không được chỉ định trong các lớp khác. CDD thường được sử dụng cho các driver mà yêu cầu hiệu suất cao và tương tác trực tiếp với phần cứng mà không thông qua các lớp trung gian khác.

Chức năng của CDD

- **Trực tiếp và hiệu quả**: CDD cho phép các ứng dụng tương tác trực tiếp với phần cứng một cách nhanh chóng và hiệu quả, đáp ứng các yêu cầu về thời gian thực.
- **Đáp ứng yêu cầu cụ thể**: CDD cung cấp các driver cho các thiết bị đặc biệt hoặc các chức năng đặc biệt mà không có trong các module chuẩn của AUTOSAR.
- **Không thông qua các lớp trung gian**: Để đảm bảo hiệu suất và thời gian phản hồi nhanh, CDD thường không thông qua các lớp trung gian như RTE.

Ví dụ về CDD

1. **ICU (Input Capture Unit) Module**: Được sử dụng để đo thời gian hoặc tần số của các tín hiệu vào.
2. **MCL (Motor Control Layer)**: Điều khiển các động cơ trong hệ thống ô tô.

Mối quan hệ giữa các lớp trong BSW

BSW Service Layer nằm trên cùng của Basic Software và cung cấp các dịch vụ cần thiết cho các lớp phần mềm khác, bao gồm RTE và Application Layer. Các lớp trong BSW Service Layer tương tác với các module phần mềm khác thông qua các hàm API, giúp giảm sự phụ thuộc và tăng tính linh hoạt của hệ thống.

Ví dụ về mối quan hệ giữa các lớp

1. **Memory Services và RTE**:
    - RTE sử dụng các dịch vụ của Memory Services để lưu trữ và truy xuất dữ liệu quan trọng, đảm bảo tính toàn vẹn và an toàn của dữ liệu.
2. **Communication Services và Application Layer**:
    - Application Layer sử dụng các dịch vụ của Communication Services để truyền và nhận dữ liệu giữa các ECU và các thiết bị ngoại vi.

Tổng kết

AUTOSAR Basic Software (BSW) Service Layer cung cấp các dịch vụ hệ thống và phần mềm cơ bản để đảm bảo hệ thống hoạt động một cách hiệu quả và an toàn. Các dịch vụ này bao gồm quản lý bộ nhớ, mã hóa dữ liệu, giao tiếp nội bộ và giao tiếp giữa các ECU. Lớp CDD cung cấp các driver phức tạp cho các thiết bị đặc biệt, đảm bảo hiệu suất cao và thời gian phản hồi nhanh.

05.APPLICATION LAYER VÀ EXAMPLE CÁCH SỬ DỤNG SWCS

Application Layer là tầng cao nhất trong kiến trúc AUTOSAR, nơi chứa các ứng dụng và chức năng cụ thể cho các hệ thống trên xe. Tầng này thực hiện các nhiệm vụ cụ thể như điều khiển túi khí, hệ thống phanh ABS, hệ thống trợ lái và radar. Ngoài ra, nó cũng bao gồm các tính năng giải trí, kết nối OTA (Over-The-Air) và nhiều hơn nữa.

Các thành phần chính của Application Layer

Ứng dụng trong Application Layer được chia thành các **Software Component (SWC)**, mỗi SWC chứa các chức năng cụ thể và có thể tương tác với các SWC khác để hoàn thiện hệ thống.

Các loại Software Component (SWC) trong AUTOSAR

1. **Application Software Component**
2. **Complex Device Driver (CDD)**
3. **Service Component**
4. **ECU Abstraction Component**
5. **Sensor Actuator Component**
6. **Parameter Component**
7. **NV Block Component**

Chi tiết về các loại SWC

1. Application Software Component

- **Chức năng**: Chứa các chức năng chính của ứng dụng, như điều khiển động cơ, phanh ABS.
- **Tương tác**: Tương tác với các SWC khác để hoàn thiện hệ thống ứng dụng.
- **Ví dụ**: Hệ thống kiểm soát động cơ, hệ thống phanh ABS.

2. NV Block Component

- **Chức năng**: Quản lý các giao tiếp với bộ nhớ không bay hơi (NVRAM).
- **Tương tác**: Tương tác với NVRAM để lưu trữ và truy xuất dữ liệu.
- **Ví dụ**: Lưu trữ dữ liệu cấu hình, thông tin chẩn đoán.

3. Complex Device Driver (CDD)

- **Chức năng**: Cung cấp các driver đặc biệt cho các thiết bị phần cứng yêu cầu hiệu suất cao.
- **Tương tác**: Tương tác trực tiếp với phần cứng để đáp ứng các yêu cầu về thời gian thực.
- **Ví dụ**: Điều khiển trực tiếp các cảm biến hoặc actuator đặc biệt.

4. Service Component

- **Chức năng**: Cấu hình và cung cấp các dịch vụ hệ thống cho các đơn vị điều khiển.
- **Tương tác**: Cung cấp các dịch vụ như quản lý bộ nhớ, chẩn đoán, quản lý mạng.
- **Ví dụ**: NVRAM Service, Diagnostic Service.

5. Service Proxy Component

- **Chức năng**: Cung cấp dịch vụ truy cập từ các đơn vị điều khiển khác.
- **Tương tác**: Cho phép các ECUs khác truy cập các dịch vụ trên ECU hiện tại.
- **Ví dụ**: Truy cập giá trị cảm biến từ ECU khác và lưu trữ vào NVRAM.

6. ECU Abstraction Component

- **Chức năng**: Hoạt động như một lớp trung gian giữa phần cứng và phần mềm.
- **Tương tác**: Tương tác với các cảm biến và actuator thông qua lớp MCAL.
- **Ví dụ**: Quản lý giao tiếp với các cảm biến nhiệt độ, áp suất.

7. Sensor Actuator Component

- **Chức năng**: Quản lý các giao tiếp với các cảm biến và actuator.
- **Tương tác**: Tương tác trực tiếp với phần mềm và phần cứng thông qua các lớp trung gian.
- **Ví dụ**: Đọc giá trị từ cảm biến, điều khiển actuator.

8. Parameter Component

- **Chức năng**: Cung cấp các giá trị cấu hình và hiệu chuẩn cho các SWC.
- **Tương tác**: Tương tác với các SWC để cung cấp các giá trị cấu hình cần thiết.
- **Ví dụ**: Cung cấp giá trị ngưỡng cho hệ thống kiểm soát tốc độ.

Ví dụ chi tiết về cách các SWC hoạt động cùng nhau

Tình huống: Đọc giá trị tốc độ từ cảm biến và kiểm tra ngưỡng an toàn

1. **Đọc giá trị từ cảm biến tốc độ**
    - **Sensor Actuator Component** sẽ đọc giá trị từ cảm biến tốc độ qua lớp ECU Abstraction.
    - **ECU Abstraction Component** đảm bảo giá trị được chuyển đổi và truyền đến các SWC khác.
2. **Lấy giá trị ngưỡng từ Parameter Component**
    - **Parameter Component** cung cấp giá trị ngưỡng an toàn cho ứng dụng.
3. **Kiểm tra giá trị tốc độ so với ngưỡng**
    - **Application Software Component** so sánh giá trị tốc độ từ cảm biến với giá trị ngưỡng.
    - Nếu giá trị tốc độ vượt quá ngưỡng, hệ thống sẽ thực hiện các hành động cần thiết.
4. **Tắt kim xăng và lưu giá trị tốc độ tối đa**

    - **Complex Device Driver (CDD)** sẽ thực hiện việc tắt kim xăng ngay lập tức để đảm bảo an toàn.
    - **Service Component** sẽ lưu giá trị tốc độ tối đa vào NVRAM thông qua **NV Block Component**.
    - **Diagnostic Service** sẽ ghi lại lỗi và lưu thông tin chẩn đoán.

Mối quan hệ giữa các lớp trong AUTOSAR

- **Application Layer** tương tác trực tiếp với **Runtime Environment (RTE)** và **Basic Software (BSW)**.
- **RTE** cung cấp môi trường thực thi cho các ứng dụng và quản lý giao tiếp giữa các SWC.
- **BSW** cung cấp các dịch vụ hệ thống cơ bản, driver phần cứng và các dịch vụ quản lý bộ nhớ, giao tiếp.

Tổng kết

AUTOSAR Application Layer chứa các ứng dụng và chức năng cụ thể cho hệ thống ô tô, được chia thành nhiều **Software Component (SWC)** để đảm bảo tính mô-đun và linh hoạt. Mỗi SWC có chức năng cụ thể và tương tác với các SWC khác để hoàn thiện hệ thống. Các loại SWC bao gồm Application Software Component, Complex Device Driver, Service Component, ECU Abstraction Component, Sensor Actuator Component, Parameter Component và NV Block Component.

06.AUTOSAR INTERFACE, PORT VÀ PORT INTERFACE

Trong kiến trúc AUTOSAR, các **Software Component (SWC)** tương tác với nhau thông qua các **Interfaces** và **Ports**. Để hiểu rõ hơn về cách các SWC giao tiếp, chúng ta cần đi sâu vào các loại interfaces và ports được sử dụng.

Các loại Interfaces trong AUTOSAR

AUTOSAR định nghĩa ba loại interface chính:

1. **AUTOSAR Interface**
2. **Sender-Receiver Interface**
3. **Client-Server Interface**

1. AUTOSAR Interface

- **Mục đích**: Giao tiếp giữa các lớp phần mềm khác nhau.
- **Vị trí**: Giữa lớp Application Software, RTE (Runtime Environment) và Basic Software.
- **Chức năng**:
    - Kết nối Application Software với RTE.
    - Kết nối RTE với Basic Software.
    - Kết nối RTE với các driver CDD (Complex Device Driver).

2. Sender-Receiver Interface

- **Mục đích**: Truyền và nhận dữ liệu giữa các SWC.
- **Loại dữ liệu**: Số nguyên, số thực, chuỗi ký tự, hoặc mảng.
- **Chức năng**: Cho phép một SWC gửi dữ liệu và một hoặc nhiều SWC khác nhận dữ liệu đó.

3. Client-Server Interface

- **Mục đích**: Gọi các hàm từ các SWC khác.
- **Chức năng**: Cho phép một SWC (Client) gọi hàm và nhận dữ liệu từ một SWC khác (Server).

Các loại Ports trong AUTOSAR

Để các SWC có thể giao tiếp với nhau, chúng cần sử dụng **Ports**. Có ba loại ports chính:

1. **Provide Port (P-Port)**
2. **Require Port (R-Port)**
3. **Provide-Require Port (PR-Port)**

1. Provide Port (P-Port)

- **Chức năng**: Cung cấp dữ liệu hoặc dịch vụ.
- **Vị trí**: Ở các SWC cung cấp dữ liệu cho các SWC khác.
- **Ví dụ**: Một SWC cung cấp dữ liệu nhiệt độ cho SWC khác.

2. Require Port (R-Port)

- **Chức năng**: Yêu cầu và nhận dữ liệu hoặc dịch vụ.
- **Vị trí**: Ở các SWC cần dữ liệu từ các SWC khác.
- **Ví dụ**: Một SWC yêu cầu dữ liệu nhiệt độ từ SWC khác.

3. Provide-Require Port (PR-Port)

- **Chức năng**: Cả cung cấp và yêu cầu dữ liệu hoặc dịch vụ.
- **Vị trí**: Ở các SWC cần cung cấp và yêu cầu dữ liệu từ các SWC khác.
- **Ví dụ**: Một SWC có thể vừa cung cấp dữ liệu về nhiệt độ vừa yêu cầu dữ liệu từ các SWC khác.

Chi tiết về các loại Interfaces

1. Sender-Receiver Interface

- **Chức năng**: Giao tiếp dữ liệu giữa các SWC theo kiểu bất đồng bộ.
- **Loại**:
    - Một gửi - một nhận.
    - Một gửi - nhiều nhận.
    - Nhiều gửi - một nhận.

  Ví dụ:

    - Một cảm biến nhiệt độ gửi dữ liệu cho một hệ thống kiểm soát nhiệt độ.
    - Nhiều cảm biến gửi dữ liệu cho một hệ thống quản lý trung tâm.

2. Client-Server Interface

- **Chức năng**: Gọi hàm từ SWC khác.
- **Loại**:
    - Một client - một server.
    - Một client - nhiều server.

  Ví dụ:

    - Một hệ thống kiểm soát yêu cầu dữ liệu từ một cảm biến cụ thể.
    - Một hệ thống kiểm soát yêu cầu dịch vụ từ nhiều cảm biến.

Ví dụ về cách sử dụng Ports và Interfaces

Giả sử chúng ta có hai SWC: **Temperature Control** và **LED Control**.

1. **Temperature Control** cần dữ liệu nhiệt độ từ cảm biến và sẽ kiểm tra nếu nhiệt độ vượt quá ngưỡng, nó sẽ gửi lệnh tắt LED.

Setup các Ports và Interfaces:

- **Temperature Control SWC**:
  - **Require Port (R-Port)** để nhận dữ liệu nhiệt độ từ **Sensor SWC**.
  - **Provide Port (P-Port)** để gửi lệnh tắt LED cho **LED Control SWC**.
- **Sensor SWC**:
  - **Provide Port (P-Port)** để cung cấp dữ liệu nhiệt độ cho **Temperature Control SWC**.
- **LED Control SWC**:
  - **Require Port (R-Port)** để nhận lệnh tắt LED từ **Temperature Control SWC**.

Thiết lập cụ thể:

1. **Sensor SWC**:
   - **P-Port**: Cung cấp dữ liệu nhiệt độ.
2. **Temperature Control SWC**:
   - **R-Port**: Nhận dữ liệu nhiệt độ từ **Sensor SWC**.
   - **P-Port**: Gửi lệnh tắt LED cho **LED Control SWC**.
3. **LED Control SWC**:
   - **R-Port**: Nhận lệnh tắt LED từ **Temperature Control SWC**.

Quá trình giao tiếp:

1. **Sensor SWC** gửi dữ liệu nhiệt độ qua **P-Port**.
2. **Temperature Control SWC** nhận dữ liệu qua **R-Port** và kiểm tra ngưỡng nhiệt độ.
3. Nếu nhiệt độ vượt quá ngưỡng, **Temperature Control SWC** gửi lệnh tắt LED qua **P-Port**.
4. **LED Control SWC** nhận lệnh qua **R-Port** và tắt LED.

Tổng kết

AUTOSAR Software Component Interfaces và Ports cho phép các SWC giao tiếp và tương tác với nhau một cách hiệu quả và linh hoạt. Các loại interface như **Sender-Receiver Interface** và **Client-Server Interface** cung cấp các phương thức giao tiếp khác nhau để đáp ứng các yêu cầu cụ thể của hệ thống. Các Ports như **Provide Port (P-Port)**, **Require Port (R-Port)** và **Provide-Require Port (PR-Port)** đảm bảo rằng dữ liệu và dịch vụ được trao đổi một cách chính xác và hiệu quả giữa các SWC.

07.COMPOSITION AND CONNECTOR

Trong kiến trúc AUTOSAR, **Composition** và **Connector** là hai khái niệm quan trọng giúp tổ chức và quản lý các **Software Components (SWC)**. Hãy cùng đi sâu vào từng khái niệm này để hiểu rõ hơn về cách chúng hoạt động và cách chúng giúp xây dựng các hệ thống phức tạp trong ô tô.

Composition

Composition là một tập hợp các SWC hoặc các **Composition** khác. Nó giúp tổ chức và quản lý các SWC có mối liên hệ chặt chẽ với nhau, làm cho hệ thống dễ hiểu và dễ quản lý hơn.

Đặc điểm của Composition:

- **Chứa nhiều SWC**: Một Composition có thể chứa nhiều SWC.
- **Chứa các Composition khác**: Một Composition có thể chứa các Composition khác, tạo ra cấu trúc phân cấp.
- **Gom nhóm các SWC liên quan**: Composition giúp gom nhóm các SWC có liên quan với nhau, làm cho việc quản lý và bảo trì hệ thống dễ dàng hơn.

Ví dụ về Composition:

Giả sử chúng ta có một hệ thống kiểm soát động cơ, trong đó bao gồm các thành phần như kiểm soát nhiên liệu, kiểm soát khí thải và kiểm soát nhiệt độ. Các thành phần này có thể được gom nhóm vào một Composition duy nhất gọi là **Engine Control Composition**.

Connector

Connector là các kết nối giữa các SWC hoặc giữa các Composition. Có ba loại Connector chính trong AUTOSAR:

1. **Assembly Connector**
2. **Delegation Connector**
3. **Pass-Through Software Connector**

1. Assembly Connector

- **Chức năng**: Kết nối trực tiếp giữa các SWC với nhau.
- **Ứng dụng**: Dùng để kết nối các SWC trong cùng một Composition hoặc giữa các Composition khác nhau.

Ví dụ về Assembly Connector:

Giả sử chúng ta có hai SWC: **Temperature Sensor** và **Temperature Controller**. Assembly Connector sẽ kết nối **Provide Port** của **Temperature Sensor** với **Require Port** của **Temperature Controller** để truyền dữ liệu nhiệt độ.

2. Delegation Connector

- **Chức năng**: Kết nối các Ports của một Composition với các Ports của SWC bên trong Composition đó.
- **Ứng dụng**: Dùng để đưa các Ports từ bên trong Composition ra bên ngoài, cho phép giao tiếp với các SWC hoặc Composition khác.

Ví dụ về Delegation Connector:

Giả sử chúng ta có một Composition **Engine Control** chứa SWC **Fuel Control**. Delegation Connector sẽ kết nối Port của **Engine Control Composition** với Port của **Fuel Control** để truyền dữ liệu ra ngoài Composition.

3. Pass-Through Software Connector

- **Chức năng**: Kết nối giữa các Composition hoặc giữa SWC trong Application Layer và Basic Software.
- **Ứng dụng**: Dùng để kết nối giữa các Composition hoặc giữa SWC trong các tầng khác nhau của phần mềm.

Ví dụ về Pass-Through Software Connector:

Giả sử chúng ta có hai Composition **Engine Control** và **Transmission Control**. Pass-Through Software Connector sẽ kết nối các Composition này để chúng có thể trao đổi dữ liệu với nhau.

Ví dụ cụ thể về Composition và Connector

Tình huống: Hệ thống kiểm soát động cơ và kiểm soát nhiệt độ

1. **Engine Control Composition**:
    - **Temperature Sensor SWC** (Provide Port)
    - **Temperature Controller SWC** (Require Port)
2. **Temperature Control Composition**:
    - **LED Control SWC** (Require Port)
    - **Temperature Display SWC** (Provide Port)

Thiết lập các Connectors:

1. **Assembly Connector**:
    - Kết nối **Temperature Sensor SWC** với **Temperature Controller SWC**.
    - Kết nối **Temperature Controller SWC** với **LED Control SWC**.
2. **Delegation Connector**:
    - Kết nối Port của **Engine Control Composition** với Port của **Temperature Sensor SWC** để đưa dữ liệu nhiệt độ ra ngoài Composition.
3. **Pass-Through Software Connector**:

- Kết nối giữa **Engine Control Composition** và **Temperature Control Composition** để trao đổi dữ liệu giữa hai Composition.

Quá trình giao tiếp:

1. **Temperature Sensor SWC** đo nhiệt độ và gửi dữ liệu qua **Provide Port**.
2. **Assembly Connector** truyền dữ liệu đến **Require Port** của **Temperature Controller SWC**.
3. **Temperature Controller SWC** xử lý dữ liệu và quyết định nếu nhiệt độ vượt quá ngưỡng.
4. Nếu nhiệt độ vượt ngưỡng, **Temperature Controller SWC** gửi lệnh tắt đèn qua **Provide Port**.
5. **Assembly Connector** truyền lệnh đến **LED Control SWC**.
6. **LED Control SWC** nhận lệnh qua **Require Port** và tắt đèn.
7. **Delegation Connector** truyền dữ liệu từ **Engine Control Composition** đến hệ thống bên ngoài nếu cần.

Tổng kết

Composition và **Connector** trong AUTOSAR giúp tổ chức và quản lý các SWC một cách hiệu quả. Composition giúp gom nhóm các SWC liên quan lại với nhau, trong khi Connector cho phép các SWC giao tiếp với nhau thông qua các kết nối rõ ràng và có cấu trúc.

08.RUNNABLE VÀ EVENTS

Trong kiến trúc AUTOSAR, **Runnable** và **Event** là hai khái niệm quan trọng giúp tổ chức và quản lý việc thực thi các đoạn mã trong hệ thống phần mềm của ô tô. Runnable là các đơn vị mã nhỏ nhất có thể thực thi được, trong khi Event là các cơ chế giúp kích hoạt hoặc lên lịch thực thi các Runnable.

Runnable

Runnable là các đơn vị mã nhỏ nhất trong hệ thống AUTOSAR. Chúng tương tự như các hàm (function) trong lập trình truyền thống và được thực thi bởi hệ điều hành (OS) hoặc Runtime Environment (RTE).

Đặc điểm của Runnable:

- **Chứa các đoạn mã có thể thực thi**: Runnable chứa các đoạn mã cần thực thi để hoàn thành một nhiệm vụ cụ thể.
- **Có thể được kích hoạt hoặc lên lịch**: Runnable có thể được kích hoạt hoặc lên lịch thực thi bởi các Event.
- **Chứa các biến và giao tiếp**: Runnable có thể chứa các biến và giao tiếp với các phần khác của hệ thống thông qua các Interface.

Các loại Runnable:

1. **Init Runnable**
    - **Mô tả**: Được thực thi trong quá trình khởi tạo hệ thống.
    - **Nhiệm vụ**: Thiết lập các tài nguyên phần cứng và phần mềm, cấu hình các kênh giao tiếp.
    - **Trigger**: Có thể được kích hoạt bởi Event khởi tạo hoặc các điều kiện đặc biệt.
2. **Periodic Runnable**
    - **Mô tả**: Được thực thi định kỳ theo một khoảng thời gian cụ thể.
    - **Nhiệm vụ**: Thực hiện các nhiệm vụ định kỳ, như cập nhật cảm biến hoặc kiểm tra trạng thái hệ thống.
    - **Trigger**: Có thể được lên lịch thực thi định kỳ với khoảng thời gian cố định hoặc thay đổi.
3. **Server Runnable**
    - **Mô tả**: Được sử dụng để cung cấp các dịch vụ cho các SWC khác trong hệ thống.
    - **Nhiệm vụ**: Xử lý các yêu cầu từ các SWC khác và cung cấp các dịch vụ cần thiết.
    - **Trigger**: Có thể được kích hoạt bởi các yêu cầu dịch vụ từ các SWC khác.

Event

Event là các cơ chế giúp kích hoạt hoặc lên lịch thực thi các Runnable. Event cung cấp các thông tin cho hệ điều hành (OS) hoặc RTE về cách thức và thời điểm kích hoạt Runnable.

Đặc điểm của Event:

- **Chỉ định cách kích hoạt Runnable**: Event chỉ định cách thức và thời điểm kích hoạt Runnable.
- **Có thể được map với các RTE Event**: Event có thể được ánh xạ với các RTE Event để đảm bảo rằng các Runnable được thực thi đúng theo mong đợi.
- **Hỗ trợ nhiều loại Event**: AUTOSAR hỗ trợ nhiều loại Event để đáp ứng các nhu cầu khác nhau của hệ thống.

Các loại Event:

1. **General Event**
   - **Loại**: Init Event, Timing Event, External Trigger Event, Internal Trigger Event, Background Event.
   - **Mô tả**: Gồm các Event chung, như Event khởi tạo, Event định kỳ, Event kích hoạt bên ngoài, Event kích hoạt bên trong, và Event nền.
2. **Client-Server Event**
   - **Loại**: Operation Invoke Event, Sync Client Call Event.
   - **Mô tả**: Gồm các Event liên quan đến cơ chế Client-Server, như Event gọi hàm và Event đồng bộ Client-Server.
3. **Data Event**
   - **Loại**: Data Receive Event, Data Send Event.
   - **Mô tả**: Gồm các Event liên quan đến dữ liệu, như Event nhận dữ liệu và Event gửi dữ liệu.
4. **Mode Event**
   - **Loại**: Mode Switch Event, Mode Manager Error Event.
   - **Mô tả**: Gồm các Event liên quan đến chế độ, như Event chuyển đổi chế độ và Event lỗi của Mode Manager.

Ví dụ về Runnable và Event

Giả sử chúng ta có một SWC cần thực hiện các chức năng tính toán cơ bản như cộng, trừ, nhân và chia. Mỗi chức năng này sẽ được thực hiện bởi một Runnable khác nhau.

1. **Software Component (SWC)**:
   - **Runnable1**: Thực hiện phép cộng.
   - **Runnable2**: Thực hiện phép trừ.
   - **Runnable3**: Thực hiện phép nhân.
   - **Runnable4**: Thực hiện phép chia.

Thiết lập Event cho các Runnable:

1. **Init Event**:
   - **Trigger**: Khi hệ thống khởi động.
   - **Runnable**: **Runnable1** (Thực hiện phép cộng).
2. **Periodic Event**:
   - **Trigger**: Mỗi 1 giây.
   - **Runnable**: **Runnable2** (Thực hiện phép trừ).

3. **Server Event**:
    - **Trigger**: Khi có yêu cầu từ SWC khác.
    - **Runnable**: **Runnable3** (Thực hiện phép nhân).
4. **Data Event**:
    - **Trigger**: Khi nhận được dữ liệu mới.
    - **Runnable**: **Runnable4** (Thực hiện phép chia).

Quá trình giao tiếp:

1. **Init Event** kích hoạt **Runnable1** khi hệ thống khởi động.
2. **Periodic Event** kích hoạt **Runnable2** mỗi 1 giây để thực hiện phép trừ.
3. **Server Event** kích hoạt **Runnable3** khi có yêu cầu từ SWC khác để thực hiện phép nhân.
4. **Data Event** kích hoạt **Runnable4** khi nhận được dữ liệu mới để thực hiện phép chia.

Tổng kết

Runnable và **Event** trong kiến trúc AUTOSAR giúp quản lý và tổ chức việc thực thi các đoạn mã trong hệ thống phần mềm ô tô. Runnable là các đơn vị mã nhỏ nhất có thể thực thi được, trong khi Event là các cơ chế giúp kích hoạt hoặc lên lịch thực thi các Runnable. Bằng cách sử dụng Runnable và Event, hệ thống AUTOSAR có thể đảm bảo rằng các chức năng được thực thi đúng theo yêu cầu và lịch trình.

09.VIRTUAL FUNCTION BUS VÀ RTE LAYER

Trong hệ thống AUTOSAR, **Virtual Function Bus (VFB)** và **Runtime Environment (RTE)** là hai thành phần quan trọng giúp quản lý và tổ chức giao tiếp giữa các **Software Components (SWC)** trong và giữa các ECU khác nhau. Hãy đi sâu vào từng khái niệm này để hiểu rõ cách chúng hoạt động và cách chúng hỗ trợ việc xây dựng các hệ thống phần mềm ô tô phức tạp.

Virtual Function Bus (VFB)

Virtual Function Bus (VFB) là một khái niệm quan trọng trong AUTOSAR, cung cấp cơ chế giao tiếp ảo giữa các SWC. VFB giúp các SWC giao tiếp với nhau mà không cần biết chi tiết về phần cứng và cấu trúc của hệ thống.

Đặc điểm của VFB:

- **Giao tiếp ảo**: VFB cung cấp một lớp giao tiếp ảo giữa các SWC, không yêu cầu kết nối vật lý trực tiếp.
- **Giao tiếp giữa các ECU**: VFB cho phép các SWC trên các ECU khác nhau giao tiếp với nhau.
- **Tầng giao tiếp trừu tượng**: VFB xử lý các chi tiết giao tiếp ở tầng thấp, như giao tiếp CAN, FlexRay, giúp SWC tập trung vào chức năng cốt lõi của mình.

Ví dụ về VFB:

Giả sử chúng ta có hai ECU, **ECU1** và **ECU2**, với các SWC như sau:

- **ECU1**: SWC1 (Cảm biến nhiệt độ)
- **ECU2**: SWC2 (Điều khiển quạt làm mát)

Khi SWC1 trên ECU1 muốn gửi dữ liệu nhiệt độ cho SWC2 trên ECU2, VFB sẽ xử lý giao tiếp này thông qua các giao thức tầng thấp như CAN hoặc FlexRay.

Runtime Environment (RTE)

Runtime Environment (RTE) là thành phần cung cấp môi trường thực thi cho các SWC và quản lý giao tiếp giữa chúng. RTE đảm bảo rằng các SWC có thể tương tác với nhau một cách hiệu quả và nhất quán.

Đặc điểm của RTE:

- **Quản lý giao tiếp**: RTE quản lý giao tiếp giữa các SWC và giữa các SWC với Basic Software (BSW).
- **Lập lịch thực thi**: RTE lập lịch thực thi các Runnable trong SWC, đảm bảo chúng được thực thi đúng thời gian.
- **Tương tác giữa các ECU**: RTE cũng chịu trách nhiệm quản lý giao tiếp giữa các SWC trên các ECU khác nhau.

Ví dụ về RTE:

Giả sử chúng ta có hai SWC trong cùng một ECU:

- **SWC1**: Đọc dữ liệu cảm biến.
- **SWC2**: Xử lý và hiển thị dữ liệu cảm biến.

RTE sẽ quản lý giao tiếp giữa SWC1 và SWC2, đảm bảo dữ liệu từ SWC1 được truyền đến SWC2 để xử lý và hiển thị.

Các API của RTE

RTE cung cấp nhiều API để hỗ trợ việc giao tiếp và quản lý thực thi của các SWC. Một số API phổ biến bao gồm:

1. **RTE_Read**: Đọc dữ liệu từ một nguồn cụ thể.
2. **RTE_Write**: Ghi dữ liệu vào một nguồn cụ thể.
3. **RTE_Call**: Gọi một hàm từ một SWC khác.

Cú pháp của API RTE

1. **RTE_Read**

   ```
   Std_ReturnType RTE_Read_<pPort>_<oVariable>(<DataType>* data);
   ```

   - **pPort**: Tên của Provide Port.
   - **oVariable**: Tên của biến dữ liệu.
   - **data**: Con trỏ đến dữ liệu sẽ được đọc.
2. **RTE_Write**

   ```
   Std_ReturnType RTE_Write_<pPort>_<oVariable>(<DataType> data);
   ```

   - **pPort**: Tên của Provide Port.
   - **oVariable**: Tên của biến dữ liệu.
   - **data**: Dữ liệu sẽ được ghi.
3. **RTE_Call**

   ```
   Std_ReturnType RTE_Call_<pPort>_<oOperation>(<InputType> input,
   <OutputType>* output);
   ```

   - **pPort**: Tên của Provide Port.
   - **oOperation**: Tên của hàm.
   - **input**: Dữ liệu đầu vào cho hàm.
   - **output**: Con trỏ đến dữ liệu đầu ra của hàm.

Ví dụ cụ thể về VFB và RTE

Giả sử chúng ta có một hệ thống gồm hai ECU, mỗi ECU có một SWC như sau:

- **ECU1**: SWC1 (Cảm biến nhiệt độ)
- **ECU2**: SWC2 (Điều khiển quạt làm mát)

1. **Giao tiếp giữa các SWC qua VFB**:
    - **SWC1** đọc dữ liệu nhiệt độ và gửi qua VFB.
    - **SWC2** nhận dữ liệu từ VFB và quyết định điều khiển quạt làm mát.
2. **Quản lý giao tiếp và thực thi bởi RTE**:
    - **RTE_Read** được sử dụng trong **SWC2** để đọc dữ liệu nhiệt độ từ **SWC1**.
    - **RTE_Call** được sử dụng để gọi hàm điều khiển quạt từ **SWC2** khi nhiệt độ vượt ngưỡng.

Tổng kết

Virtual Function Bus (VFB) và **Runtime Environment (RTE)** là hai thành phần quan trọng trong kiến trúc AUTOSAR giúp quản lý và tổ chức giao tiếp giữa các SWC. VFB cung cấp một lớp giao tiếp ảo, cho phép các SWC giao tiếp với nhau mà không cần biết chi tiết về phần cứng. RTE cung cấp môi trường thực thi cho các SWC, quản lý giao tiếp và lập lịch thực thi, đảm bảo các SWC hoạt động hiệu quả và nhất quán.

10.EXAMPLE TỔNG QUAN CÁC LỚP

Trong hệ thống AUTOSAR, **RTE Generator** là công cụ quan trọng giúp tạo ra môi trường thực thi và các file cấu hình cần thiết để các **Software Component (SWC)** có thể giao tiếp và hoạt động một cách trơn tru. Phần này sẽ tập trung vào việc giải thích quá trình cấu hình và tạo ra RTE, cũng như cung cấp một ví dụ cụ thể để minh họa.

RTE Generator

RTE Generator là công cụ tạo ra mã nguồn cho môi trường thực thi của các SWC dựa trên các file cấu hình AUTOSAR XML (ARXML). Quá trình này bao gồm hai pha chính: **Contract Phase** và **Generation Phase**.

Contract Phase

Trong pha này, RTE Generator sử dụng các file ARXML chứa định nghĩa về các SWC và các giao diện của chúng. Kết quả của pha này là các file tiêu đề (header files) và định nghĩa các hàm cho các SWC.

- **Input**: File ARXML định nghĩa các SWC.
- **Output**: Header files và các định nghĩa hàm.

Generation Phase

Trong pha này, RTE Generator dựa vào các file ARXML, cấu hình của OS, và các file cấu hình khác để tạo ra mã nguồn đầy đủ cho RTE.

- **Input**: File ARXML, cấu hình OS, cấu hình ECU.
- **Output**: Mã nguồn đầy đủ cho RTE, bao gồm các file mã nguồn C/C++.

Các công cụ sử dụng trong AUTOSAR

Có nhiều công cụ khác nhau có thể được sử dụng để cấu hình và tạo ra RTE. Một số công cụ phổ biến bao gồm:

- **Vector DaVinci Developer**
- **Electrobit Tresos Studio**
- **Artop Tool Platform**

Trong ví dụ này, chúng ta sẽ tập trung vào việc sử dụng **Vector DaVinci Developer** để cấu hình và tạo RTE.

Ví dụ về Cấu Hình AUTOSAR

Hãy xem xét một ví dụ đơn giản để minh họa cách cấu hình các thành phần AUTOSAR và tạo ra RTE.

Mô tả hệ thống

Giả sử chúng ta có một hệ thống đơn giản bao gồm các thành phần sau:

1. **ADC Driver**: Đọc giá trị từ cảm biến nhiệt độ.
2. **DIO Driver**: Điều khiển động cơ (motor).
3. **Temperature Control Application**: Đọc giá trị nhiệt độ và điều khiển động cơ dựa trên giá trị nhiệt độ.

Các bước cấu hình

1. **Tạo các SWC và cấu hình của chúng trong ARXML**:
    - Định nghĩa các SWC như **ADC Driver**, **DIO Driver**, và **Temperature Control Application**.
    - Định nghĩa các giao diện (interfaces) và các cổng (ports) cho các SWC này.
2. **Cấu hình các Runnable và Event**:
    - Xác định các runnable cho các chức năng như đọc giá trị ADC, điều khiển DIO.
    - Định nghĩa các event để kích hoạt các runnable này.
3. **Tạo cấu hình cho OS và RTE**:
    - Cấu hình hệ điều hành (OS) cho các nhiệm vụ (tasks) và các lịch trình (schedules).
    - Tạo cấu hình cho RTE dựa trên các định nghĩa SWC và OS.
4. **Chạy RTE Generator**:
    - Sử dụng công cụ như **Vector DaVinci Developer** để chạy RTE Generator và tạo ra mã nguồn cần thiết.

Ví dụ cụ thể

1. Tạo các SWC

- **ADC Driver**: Đọc giá trị từ cảm biến nhiệt độ.
    - Provide Port: `P_ADC_Value`
    - Runnable: `Read_ADC`
- **DIO Driver**: Điều khiển động cơ.
    - Provide Port: `P_Motor_Control`
    - Runnable: `Control_Motor`
- **Temperature Control Application**: Đọc giá trị nhiệt độ và điều khiển động cơ.
    - Require Port: `R_ADC_Value`
    - Require Port: `R_Motor_Control`
    - Runnable: `Monitor_Temperature`

2. Cấu hình các Runnable và Event

- **ADC Driver**:
    - Runnable: `Read_ADC`
    - Event: `Periodic_Event` (Kích hoạt định kỳ để đọc giá trị ADC)

- **DIO Driver**:
  - Runnable: `Control_Motor`
  - Event: `Direct_Call_Event` (Kích hoạt khi có yêu cầu điều khiển động cơ)
- **Temperature Control Application**:
  - Runnable: `Monitor_Temperature`
  - Event: `Data_Receive_Event` (Kích hoạt khi nhận được dữ liệu nhiệt độ mới)

3. Tạo cấu hình cho OS và RTE

- Cấu hình OS:
  - Task: `Task_ADC_Read` (Chạy định kỳ để đọc giá trị ADC)
  - Task: `Task_Temperature_Control` (Chạy khi có dữ liệu nhiệt độ mới)
- Cấu hình RTE:
  - Tạo ra các API cho RTE như `RTE_Read_ADC`, `RTE_Control_Motor`, `RTE_Monitor_Temperature`.

4. Chạy RTE Generator

Sử dụng công cụ **Vector DaVinci Developer** để chạy RTE Generator, tạo ra các file mã nguồn cần thiết.

Tổng kết

RTE Generator là công cụ quan trọng trong hệ thống AUTOSAR, giúp tạo ra mã nguồn cho môi trường thực thi của các SWC. Quá trình cấu hình và tạo RTE bao gồm hai pha: **Contract Phase** và **Generation Phase**, với các đầu vào là các file cấu hình ARXML và các file cấu hình OS, ECU. Các công cụ như **Vector DaVinci Developer** giúp tự động hóa quá trình này, đảm bảo các SWC có thể giao tiếp và hoạt động một cách trơn tru.

11.PRACTISE PROJECT CREATE PROJECT VÀ BASE DATA TYPES

Trong bài thực hành này, chúng ta sẽ áp dụng các kiến thức đã học về các **Ports**, **Interfaces**, **Software Components (SWCs)**, **Connectors**, **Compositions**, và các khái niệm khác để xây dựng một hệ thống kiểm soát nhiệt độ đơn giản trong AUTOSAR. Hệ thống sẽ bao gồm ba SWCs chính trong lớp Application: **Temperature Control**, **LED Control**, và **Heater Control**. Mục tiêu của hệ thống là kiểm soát nhiệt độ và điều khiển LED và Heater dựa trên giá trị nhiệt độ.

Bước 1: Tạo Các Data Types Cơ Bản

Trước khi bắt đầu tạo các SWCs, chúng ta cần tạo các **Data Types** cơ bản trong AUTOSAR. Điều này là cần thiết để định nghĩa các biến và kiểu dữ liệu mà các SWCs sẽ sử dụng.

Các bước thực hiện:

1. Mở công cụ AUTOSAR (như Vector DaVinci Developer).
2. Tạo một dự án mới.
3. Tạo các Data Types cơ bản:

- **Signed Integers**:
  - `sint8` (8-bit signed integer)
  - `sint16` (16-bit signed integer)
  - `sint32` (32-bit signed integer)
  - `sint64` (64-bit signed integer)
- **Unsigned Integers**:
  - `uint8` (8-bit unsigned integer)
  - `uint16` (16-bit unsigned integer)
  - `uint32` (32-bit unsigned integer)
  - `uint64` (64-bit unsigned integer)
- **Floating Point**:
  - `float32` (32-bit floating point)
  - `float64` (64-bit floating point)

Cấu hình chi tiết:

1. **Data Type Configuration**:
   - Mở **Data Type Editor** trong công cụ AUTOSAR.
   - Thêm các Data Types như sau:

```
Name: sint8, Size: 8, Type: Signed Integer
Name: sint16, Size: 16, Type: Signed Integer
Name: sint32, Size: 32, Type: Signed Integer
Name: sint64, Size: 64, Type: Signed Integer

Name: uint8, Size: 8, Type: Unsigned Integer
Name: uint16, Size: 16, Type: Unsigned Integer
```

```
Name: uint32, Size: 32, Type: Unsigned Integer
Name: uint64, Size: 64, Type: Unsigned Integer

Name: float32, Size: 32, Type: Floating Point
Name: float64, Size: 64, Type: Floating Point
```

Bước 2: Tạo Các Software Components (SWCs)

1. Temperature Control SWC:

- **Purpose**: Đọc giá trị nhiệt độ và điều khiển LED và Heater dựa trên giá trị nhiệt độ.
- **Ports**:
  - **Require Port**: `R_Temperature` (để nhận dữ liệu nhiệt độ từ ADC)
  - **Provide Port**: `P_LED_Control` (để điều khiển LED)
  - **Provide Port**: `P_Heater_Control` (để điều khiển Heater)

2. LED Control SWC:

- **Purpose**: Điều khiển LED dựa trên tín hiệu từ Temperature Control.
- **Ports**:
  - **Require Port**: `R_LED_Command` (để nhận lệnh từ Temperature Control)

3. Heater Control SWC:

- **Purpose**: Điều khiển Heater dựa trên tín hiệu từ Temperature Control.
- **Ports**:
  - **Require Port**: `R_Heater_Command` (để nhận lệnh từ Temperature Control)

Bước 3: Tạo các Runnable và Event

1. Temperature Control SWC:

- **Runnable**: `Monitor_Temperature`
  - **Event**: `Periodic_Event` (Kích hoạt định kỳ mỗi 1 giây để kiểm tra nhiệt độ)

2. LED Control SWC:

- **Runnable**: `Control_LED`
  - **Event**: `Data_Receive_Event` (Kích hoạt khi nhận được lệnh từ Temperature Control)

3. Heater Control SWC:

- **Runnable**: `Control_Heater`
  - **Event**: `Data_Receive_Event` (Kích hoạt khi nhận được lệnh từ Temperature Control)

Bước 4: Tạo Composition và Kết Nối Các SWCs

1. Tạo Composition:

- Tạo một **Composition** chứa các SWC:
  - `TemperatureControlComposition`
    - Chứa: `TemperatureControlSWC`, `LEDControlSWC`, `HeaterControlSWC`

2. Kết nối các SWCs:

- Sử dụng **Assembly Connectors** để kết nối các ports của các SWC.

Ví dụ:

- **TemperatureControlSWC.P_LED_Control** -> **LEDControlSWC.R_LED_Command**
- **TemperatureControlSWC.P_Heater_Control** -> **HeaterControlSWC.R_Heater_Command**

Bước 5: Cấu hình OS và RTE

1. Cấu hình OS:

- Tạo các Task cho các Runnable:
  - `Task_Monitor_Temperature` (Chạy định kỳ mỗi 1 giây)
  - `Task_Control_LED` (Chạy khi có lệnh điều khiển LED)
  - `Task_Control_Heater` (Chạy khi có lệnh điều khiển Heater)

2. Cấu hình RTE:

- Sử dụng RTE Generator để tạo các file cấu hình và mã nguồn cần thiết.
- Đảm bảo rằng các API như `RTE_Read`, `RTE_Write`, `RTE_Call` được cấu hình đúng cách.

Bước 6: Kiểm tra và Gỡ lỗi

- **Compile và build** hệ thống.
- **Test** hệ thống bằng cách kiểm tra các hành vi của Temperature Control, LED Control và Heater Control.
- **Debug** nếu có lỗi xảy ra.

Tổng kết

Bài thực hành này giúp bạn hiểu rõ hơn về cách áp dụng các khái niệm AUTOSAR như **Ports**, **Interfaces**, **SWCs**, **Connectors**, **Compositions**, **Runnable** và **Event** vào một bài toán thực tế. Bằng cách thực hiện từng bước từ việc tạo các Data Types cơ bản đến cấu hình các SWC và

RTE, bạn sẽ có một cái nhìn tổng quan về quy trình phát triển và cấu hình một hệ thống phần mềm AUTOSAR.

12.PRACTISE PROJECT CREATE VARIABLE ĐỂ GIAO TIẾP GIỮA CÁC SWCS

Trong phần này, chúng ta sẽ tạo các biến cần thiết để các SWCs có thể giao tiếp với nhau. Các biến này sẽ đại diện cho các giá trị dữ liệu được truyền giữa các SWCs, như điều khiển LED, điều khiển Heater, và đọc giá trị nhiệt độ.

Bước 1: Tạo Các Biến (Variables)

1. LED On/Off Variable:

- **Purpose**: Điều khiển trạng thái bật/tắt của LED từ `Temperature Control` đến `LED Control`.
- **Type**: `uint8`
- **Variable Name**: `LED_On_Off`

2. Heater On/Off Variable:

- **Purpose**: Điều khiển trạng thái bật/tắt của Heater từ `Temperature Control` đến `Heater Control`.
- **Type**: `uint8`
- **Variable Name**: `Heater_On_Off`

3. Temperature Value Variable:

- **Purpose**: Lưu trữ giá trị nhiệt độ đọc được từ cảm biến ADC.
- **Type**: `uint16`
- **Variable Name**: `Temperature_Value`

4. LED Control IO Hardware Abstraction Layer:

- **Purpose**: Điều khiển LED thông qua lớp `IO Hardware Abstraction Layer`.
- **Type**: `uint8`
- **Variable Name**: `LED_Control_IOHAL`

5. Heater Control IO Hardware Abstraction Layer:

- **Purpose**: Điều khiển Heater thông qua lớp `IO Hardware Abstraction Layer`.
- **Type**: `uint8`
- **Variable Name**: `Heater_Control_IOHAL`

Cách Tạo Các Biến trong AUTOSAR Tool

1. **Mở AUTOSAR Tool (Vector DaVinci Developer hoặc tương tự)**.
2. **Chọn Dự Án Hiện Tại**.
3. **Đi tới Phần Implement Data Type**:
    - Mở `Implement Data Type Editor`.

- Tạo mới các biến cần thiết bằng cách thêm `Variables`.

Cấu hình chi tiết các biến:

1. **Tạo Biến `LED_On_Off`:**
   - Tên biến: `LED_On_Off`
   - Kiểu dữ liệu: `uint8`
   - Referenced Data Type: `uint8`

```
Name: LED_On_Off
Data Type: uint8
```

2. **Tạo Biến `Heater_On_Off`:**
   - Tên biến: `Heater_On_Off`
   - Kiểu dữ liệu: `uint8`
   - Referenced Data Type: `uint8`

```
Name: Heater_On_Off
Data Type: uint8
```

3. **Tạo Biến `Temperature_Value`:**
   - Tên biến: `Temperature_Value`
   - Kiểu dữ liệu: `uint16`
   - Referenced Data Type: `uint16`

```
Name: Temperature_Value
Data Type: uint16
```

4. **Tạo Biến `LED_Control_IOHAL`:**
   - Tên biến: `LED_Control_IOHAL`
   - Kiểu dữ liệu: `uint8`
   - Referenced Data Type: `uint8`

```
Name: LED_Control_IOHAL
Data Type: uint8
```

5. **Tạo Biến `Heater_Control_IOHAL`:**
   - Tên biến: `Heater_Control_IOHAL`
   - Kiểu dữ liệu: `uint8`
   - Referenced Data Type: `uint8`

```
text
Sao chép mã
Name: Heater_Control_IOHAL
Data Type: uint8
```

Kết quả sau khi tạo các biến:

Chúng ta đã tạo thành công năm biến cần thiết để giao tiếp giữa các SWC và các lớp IO Hardware Abstraction Layer.

Tiếp Theo: Tạo Interface cho SWCs

Trong video tiếp theo, chúng ta sẽ tạo các **Interfaces** để các SWCs có thể giao tiếp với nhau thông qua các biến mà chúng ta vừa tạo. Chúng ta sẽ cấu hình các Interfaces trong AUTOSAR Tool và liên kết chúng với các Ports của SWC.

Tổng kết

Trong phần này, chúng ta đã học cách tạo các biến cơ bản trong AUTOSAR để giao tiếp giữa các SWC. Việc tạo các biến là bước đầu tiên quan trọng để đảm bảo rằng các SWC có thể truyền và nhận dữ liệu một cách chính xác.

13.PRACTISE PROJECT CREATE PORT INTERFACE CHO CÁC SWCS

Trong phần này, chúng ta sẽ tạo các **Interfaces** cần thiết để các **Software Components (SWCs)** có thể giao tiếp với nhau trong hệ thống AUTOSAR. Chúng ta sẽ xác định các loại interface và cấu hình chúng trong công cụ AUTOSAR.

Các Loại Interfaces

1. **Client-Server Interface (CS)**
2. **Sender-Receiver Interface (SR)**
3. **Parameter Interface**
4. **Mode-Switch Interface**
5. **Trigger Interface**

Xác Định Các Interfaces Giữa Các SWCs

1. **Temperature Control** và **LED Control**:
   - **Type**: Sender-Receiver
   - **Interface**: `SR_LED_On_Off`
   - **Variable**: `LED_On_Off`
2. **Temperature Control** và **Heater Control**:
   - **Type**: Sender-Receiver
   - **Interface**: `SR_Heater_On_Off`
   - **Variable**: `Heater_On_Off`
3. **Temperature Control** và **IO Hardware Abstraction Layer**:
   - **Type**: Sender-Receiver
   - **Interface**: `SR_Temperature_Value`
   - **Variable**: `Temperature_Value`
4. **LED Control** và **IO Hardware Abstraction Layer**:
   - **Type**: Sender-Receiver
   - **Interface**: `SR_LED_Control_IOHAL`
   - **Variable**: `LED_Control_IOHAL`
5. **Heater Control** và **IO Hardware Abstraction Layer**:
   - **Type**: Sender-Receiver
   - **Interface**: `SR_Heater_Control_IOHAL`
   - **Variable**: `Heater_Control_IOHAL`
6. **Temperature Control** và **Line Control**:
   - **Type**: Client-Server
   - **Interface**: `CS_Line_Control`
   - **Variable**: `Line_On_Off`

Tạo Interfaces Trong AUTOSAR Tool

Bước 1: Mở Tab Interfaces

1. Mở công cụ AUTOSAR (Vector DaVinci Developer hoặc tương tự).

2. Mở dự án hiện tại và đi tới tab `Interfaces`.

Bước 2: Tạo Client-Server Interface

1. Tạo một **Client-Server Interface** mới.
2. Đặt tên: `CS_Line_Control`.
3. Kích đúp vào `CS_Line_Control` để mở cấu hình chi tiết.
4. Thêm một operation:
    - Tên: `Operation_Line_Control`
    - Referenced Data Type: `Line_On_Off`
    - Data Element: `Line_On_Off`

```
Name: CS_Line_Control
Operation: Operation_Line_Control
Referenced Data Type: Line_On_Off
```

Bước 3: Tạo Sender-Receiver Interfaces

1. **SR_LED_On_Off**:
    - Tên: `SR_LED_On_Off`
    - Data Element: `LED_On_Off`
2. **SR_Heater_On_Off**:
    - Tên: `SR_Heater_On_Off`
    - Data Element: `Heater_On_Off`
3. **SR_Temperature_Value**:
    - Tên: `SR_Temperature_Value`
    - Data Element: `Temperature_Value`
4. **SR_LED_Control_IOHAL**:
    - Tên: `SR_LED_Control_IOHAL`
    - Data Element: `LED_Control_IOHAL`
5. **SR_Heater_Control_IOHAL**:
    - Tên: `SR_Heater_Control_IOHAL`
    - Data Element: `Heater_Control_IOHAL`

```
Name: SR_LED_On_Off
Data Element: LED_On_Off

Name: SR_Heater_On_Off
Data Element: Heater_On_Off

Name: SR_Temperature_Value
Data Element: Temperature_Value

Name: SR_LED_Control_IOHAL
Data Element: LED_Control_IOHAL

Name: SR_Heater_Control_IOHAL
Data Element: Heater_Control_IOHAL
```

Tạo Các Interfaces

1. Mở tab `Interfaces` và chọn `New Interface`.
2. Chọn loại interface (Client-Server hoặc Sender-Receiver).
3. Điền các thông tin cần thiết và liên kết với các data element đã tạo trước đó.

```
Name: CS_Line_Control
Type: Client-Server
Operation: Operation_Line_Control
Referenced Data Type: Line_On_Off

Name: SR_LED_On_Off
Type: Sender-Receiver
Data Element: LED_On_Off

Name: SR_Heater_On_Off
Type: Sender-Receiver
Data Element: Heater_On_Off

Name: SR_Temperature_Value
Type: Sender-Receiver
Data Element: Temperature_Value

Name: SR_LED_Control_IOHAL
Type: Sender-Receiver
Data Element: LED_Control_IOHAL

Name: SR_Heater_Control_IOHAL
Type: Sender-Receiver
Data Element: Heater_Control_IOHAL
```

Tổng kết

Trong phần này, chúng ta đã tạo các interfaces cần thiết để các SWCs có thể giao tiếp với nhau. Chúng ta đã tạo cả Client-Server Interface và Sender-Receiver Interface dựa trên các biến đã tạo trước đó. Các interfaces này sẽ giúp truyền tải dữ liệu giữa các thành phần trong hệ thống AUTOSAR.

14.PRACTISE PROJECT CREATE SWCS VÀ AUTOSAR PORT CHO SWC

Trong phần này, chúng ta sẽ tạo các SWCs và cấu hình các ports để giao tiếp giữa các SWCs trong hệ thống AUTOSAR.

Bước 1: Tạo Các SWCs

Chúng ta sẽ tạo các SWCs sau đây:

- **Application Layer**:
    1. Temp Control
    2. LED Control
    3. Heater Control
- **ECU Abstraction Layer**:
    1. IO Hardware Layer (LED Control)
    2. IO Hardware Layer (Heater Control)
    3. IO Hardware Layer (Temperature Sensor)

1. Tạo SWCs cho Application Layer

1. **Temp Control**:
    - Vào `Software Component` → `New Component` → `Application Layer`
    - Tên: `Temp_Control`
    - Các Ports:
        - `Provide Port`: `Prov_LED_On_Off` (SR Interface)
        - `Require Port`: `Req_LED_On_Off` (SR Interface)
        - `Require Port`: `Req_Heater_On_Off` (SR Interface)
        - `Require Port`: `Req_Temp_Value` (SR Interface)
2. **LED Control**:
    - Vào `Software Component` → `New Component` → `Application Layer`
    - Tên: `LED_Control`
    - Các Ports:
        - `Provide Port`: `Prov_LED_On_Off` (SR Interface)
        - `Require Port`: `Req_Temp_Value` (SR Interface)
3. **Heater Control**:
    - Vào `Software Component` → `New Component` → `Application Layer`
    - Tên: `Heater_Control`
    - Các Ports:
        - `Provide Port`: `Prov_Heater_On_Off` (SR Interface)
        - `Require Port`: `Req_Temp_Value` (SR Interface)

2. Tạo SWCs cho ECU Abstraction Layer

1. **IO Hardware Layer (LED Control)**:
    - Vào `Software Component` → `New Component` → `ECU Abstraction Layer`
    - Tên: `IO_LED_Control`

- Các Ports:
  - `Require Port`: `Req_LED_Control_IOHAL` (SR Interface)

2. **IO Hardware Layer (Heater Control)**:
   - Vào `Software Component` → `New Component` → `ECU Abstraction Layer`
   - Tên: `IO_Heater_Control`
   - Các Ports:
     - `Require Port`: `Req_Heater_Control_IOHAL` (SR Interface)
3. **IO Hardware Layer (Temperature Sensor)**:
   - Vào `Software Component` → `New Component` → `ECU Abstraction Layer`
   - Tên: `IO_Temperature_Sensor`
   - Các Ports:
     - `Provide Port`: `Prov_Temp_Value` (SR Interface)

Bước 2: Cấu Hình Ports cho SWCs

1. Temp Control

1. Mở `Temp_Control` → `Ports`
2. Thêm các Ports:
   - `Provide Port`:
     - Tên: `Prov_LED_On_Off`
     - Interface: `SR_LED_On_Off`
   - `Require Port`:
     - Tên: `Req_LED_On_Off`
     - Interface: `SR_LED_On_Off`
   - `Require Port`:
     - Tên: `Req_Heater_On_Off`
     - Interface: `SR_Heater_On_Off`
   - `Require Port`:
     - Tên: `Req_Temp_Value`
     - Interface: `SR_Temperature_Value`

2. LED Control

1. Mở `LED_Control` → `Ports`
2. Thêm các Ports:
   - `Provide Port`:
     - Tên: `Prov_LED_On_Off`
     - Interface: `SR_LED_On_Off`
   - `Require Port`:
     - Tên: `Req_Temp_Value`
     - Interface: `SR_Temperature_Value`

3. Heater Control

1. Mở `Heater_Control` → `Ports`

2. Thêm các Ports:
    - `Provide Port`:
        - Tên: `Prov_Heater_On_Off`
        - Interface: `SR_Heater_On_Off`
    - `Require Port`:
        - Tên: `Req_Temp_Value`
        - Interface: `SR_Temperature_Value`

4. IO Hardware Layer (LED Control)

1. Mở `IO_LED_Control` → `Ports`
2. Thêm các Ports:
    - `Require Port`:
        - Tên: `Req_LED_Control_IOHAL`
        - Interface: `SR_LED_Control_IOHAL`

5. IO Hardware Layer (Heater Control)

1. Mở `IO_Heater_Control` → `Ports`
2. Thêm các Ports:
    - `Require Port`:
        - Tên: `Req_Heater_Control_IOHAL`
        - Interface: `SR_Heater_Control_IOHAL`

6. IO Hardware Layer (Temperature Sensor)

1. Mở `IO_Temperature_Sensor` → `Ports`
2. Thêm các Ports:
    - `Provide Port`:
        - Tên: `Prov_Temp_Value`
        - Interface: `SR_Temperature_Value`

Kết Luận

Trong phần này, chúng ta đã tạo các SWCs và cấu hình các ports cần thiết để giao tiếp giữa các SWCs trong hệ thống AUTOSAR. Chúng ta đã xác định và thiết lập các interfaces để truyền tải dữ liệu giữa các thành phần khác nhau.

15.PRACTISE PROJECT CREATE RUNNABLES VÀ DATAACCESS CHO SWC

Trong phần này, chúng ta sẽ tạo các runnables (r) cho từng software component (SC) và cấu hình data access để đảm bảo các SC có thể giao tiếp với nhau một cách hiệu quả.

Bước 1: Tạo Runnables cho Các SC

1. Temp Control

1. **Tạo Runnables**:
   - Mở `Temp_Control` → `Internal Behavior` → `Runnables`
   - Tạo ba runnables:
     - `Temp_Control_R1_LED_On`
     - `Temp_Control_R2_Heater_On`
     - `Temp_Control_R3_Read_Temp`
2. **Cấu Hình Data Access**:
   - `Temp_Control_R1_LED_On`:
     - Port: `Prov_LED_On_Off`
     - Interface: `SR_LED_On_Off`
     - Data Access Mode: `Write`
   - `Temp_Control_R2_Heater_On`:
     - Port: `Prov_Heater_On_Off`
     - Interface: `SR_Heater_On_Off`
     - Data Access Mode: `Write`
   - `Temp_Control_R3_Read_Temp`:
     - Port: `Req_Temp_Value`
     - Interface: `SR_Temperature_Value`
     - Data Access Mode: `Read`

2. LED Control

1. **Tạo Runnables**:
   - Mở `LED_Control` → `Internal Behavior` → `Runnables`
   - Tạo hai runnables:
     - `LED_Control_R4_Temp_Control`
     - `LED_Control_R6_IO_Hardware`
2. **Cấu Hình Data Access**:
   - `LED_Control_R4_Temp_Control`:
     - Port: `Req_Temp_Value`
     - Interface: `SR_Temperature_Value`
     - Data Access Mode: `Read`
   - `LED_Control_R6_IO_Hardware`:
     - Port: `Prov_LED_On_Off`
     - Interface: `SR_LED_Control_IOHAL`
     - Data Access Mode: `Write`

3. Heater Control

1. **Tạo Runnables**:
   - Mở `Heater_Control` → `Internal Behavior` → `Runnables`
   - Tạo hai runnables:
     - `Heater_Control_R5_Temp_Control`
     - `Heater_Control_R7_IO_Hardware`
2. **Cấu Hình Data Access**:
   - `Heater_Control_R5_Temp_Control`:
     - Port: `Req_Temp_Value`
     - Interface: `SR_Temperature_Value`
     - Data Access Mode: `Read`
   - `Heater_Control_R7_IO_Hardware`:
     - Port: `Prov_Heater_On_Off`
     - Interface: `SR_Heater_Control_IOHAL`
     - Data Access Mode: `Write`

4. IO Hardware Layer (LED Control)

1. **Tạo Runnables**:
   - Mở `IO_LED_Control` → `Internal Behavior` → `Runnables`
   - Tạo một runnable:
     - `IO_LED_Control_R8`
2. **Cấu Hình Data Access**:
   - `IO_LED_Control_R8`:
     - Port: `Req_LED_Control_IOHAL`
     - Interface: `SR_LED_Control_IOHAL`
     - Data Access Mode: `Read`

5. IO Hardware Layer (Heater Control)

1. **Tạo Runnables**:
   - Mở `IO_Heater_Control` → `Internal Behavior` → `Runnables`
   - Tạo một runnable:
     - `IO_Heater_Control_R9`
2. **Cấu Hình Data Access**:
   - `IO_Heater_Control_R9`:
     - Port: `Req_Heater_Control_IOHAL`
     - Interface: `SR_Heater_Control_IOHAL`
     - Data Access Mode: `Read`

6. IO Hardware Layer (Temperature Sensor)

1. **Tạo Runnables**:
   - Mở `IO_Temperature_Sensor` → `Internal Behavior` → `Runnables`
   - Tạo một runnable:

        - `IO_Temperature_Sensor_R10`
2. **Cấu Hình Data Access**:
    - `IO_Temperature_Sensor_R10`:
        - Port: `Prov_Temp_Value`
        - Interface: `SR_Temperature_Value`
        - Data Access Mode: `Write`

Kết Luận

Chúng ta đã tạo các runnables cho từng SC và cấu hình data access để các SC có thể giao tiếp với nhau. Đây là bước quan trọng để đảm bảo rằng các SC trong hệ thống AUTOSAR có thể tương tác và thực hiện các chức năng một cách chính xác.

16.PRACTISE PROJECT CREATE EVENT CHO RUNNABLES CỦA SWC

Mục Tiêu

Trong bài này, chúng ta sẽ tạo các event cho các runnables đã tạo ở các bước trước. Mỗi event sẽ kích hoạt các runnables tương ứng với các điều kiện hoặc thời gian cụ thể.

Bước 1: Tạo Event Cho `Temp Control`

1. **Mở `Temp Control`**:
   - Mở project và chọn `Temp Control` từ danh sách các software component.
2. **Tạo Event `Temp_Read`**:
   - Chọn tab `Events`.
   - Tạo event mới:
     - Name: `Temp_Read_Event`
     - Type: `Periodic`
     - Interval: `1s`
     - Associated Runnable: `Temp_Control_R3_Read_Temp`
3. **Tạo Event `LED_On`**:
   - Tạo event mới:
     - Name: `LED_On_Event`
     - Type: `Periodic`
     - Interval: `1s`
     - Associated Runnable: `Temp_Control_R1_LED_On`
4. **Tạo Event `Heater_On`**:
   - Tạo event mới:
     - Name: `Heater_On_Event`
     - Type: `Periodic`
     - Interval: `1s`
     - Associated Runnable: `Temp_Control_R2_Heater_On`

Bước 2: Tạo Event Cho `LED Control`

1. **Mở `LED Control`**:
   - Chọn `LED Control` từ danh sách các software component.
2. **Tạo Event `Control_From_Temp`**:
   - Chọn tab `Events`.
   - Tạo event mới:
     - Name: `Control_From_Temp_Event`
     - Type: `Periodic`
     - Interval: `1s`
     - Associated Runnable: `LED_Control_R4_Temp_Control`
3. **Tạo Event `Control_To_IOHAL`**:
   - Tạo event mới:
     - Name: `Control_To_IOHAL_Event`

        - Type: `Periodic`
        - Interval: `1s`
        - Associated Runnable: `LED_Control_R6_IO_Hardware`

Bước 3: Tạo Event Cho `Heater Control`

1. **Mở `Heater Control`**:
    - Chọn `Heater Control` từ danh sách các software component.
2. **Tạo Event `Control_From_Temp`**:
    - Chọn tab `Events`.
    - Tạo event mới:
        - Name: `Control_From_Temp_Event`
        - Type: `Periodic`
        - Interval: `1s`
        - Associated Runnable: `Heater_Control_R5_Temp_Control`
3. **Tạo Event `Control_To_IOHAL`**:
    - Tạo event mới:
        - Name: `Control_To_IOHAL_Event`
        - Type: `Periodic`
        - Interval: `1s`
        - Associated Runnable: `Heater_Control_R7_IO_Hardware`

Bước 4: Tạo Event Cho `IO Hardware Layer (LED Control)`

1. **Mở `IO_LED_Control`**:
    - Chọn `IO_LED_Control` từ danh sách các software component.
2. **Tạo Event `Control_From_LED`**:
    - Chọn tab `Events`.
    - Tạo event mới:
        - Name: `Control_From_LED_Event`
        - Type: `Periodic`
        - Interval: `1s`
        - Associated Runnable: `IO_LED_Control_R8`

Bước 5: Tạo Event Cho `IO Hardware Layer (Heater Control)`

1. **Mở `IO_Heater_Control`**:
    - Chọn `IO_Heater_Control` từ danh sách các software component.
2. **Tạo Event `Control_From_Heater`**:
    - Chọn tab `Events`.
    - Tạo event mới:
        - Name: `Control_From_Heater_Event`
        - Type: `Periodic`
        - Interval: `1s`
        - Associated Runnable: `IO_Heater_Control_R9`

Bước 6: Tạo Event Cho `IO Hardware Layer (Temperature Sensor)`

1. **Mở `IO_Temperature_Sensor`:**
    - Chọn `IO_Temperature_Sensor` từ danh sách các software component.
2. **Tạo Event `Provide_Temp`:**
    - Chọn tab `Events`.
    - Tạo event mới:
        - Name: `Provide_Temp_Event`
        - Type: `Periodic`
        - Interval: `1s`
        - Associated Runnable: `IO_Temperature_Sensor_R10`

Tổng Kết

Chúng ta đã tạo các event cho từng runnables trong các software component. Mỗi event sẽ kích hoạt các runnables tương ứng để thực hiện các chức năng cần thiết trong hệ thống.

17.PRACTISE PROJECT CREATE SOFTWARE COMPOSITIONS VÀ CONNECTIONS

Mục Tiêu

Trong bài này, chúng ta sẽ tạo Software Composition cho dự án, bao gồm việc gom các Software Components lại và kết nối chúng với nhau thông qua các Connectors.

Bước 1: Tạo Software Composition

1. **Mở Dự Án**:
   - Mở project trong công cụ cấu hình AUTOSAR của bạn.
2. **Tạo Software Composition**:
   - Tạo một Software Composition mới với tên `Software_Comp`.
3. **Thêm Các Software Components**:
   - Thêm các Software Components đã tạo vào Composition:
     - `Temp_Control`
     - `LED_Control`
     - `Heater_Control`
     - `IO_LED_Control`
     - `IO_Heater_Control`
     - `IO_Temperature_Sensor`

Bước 2: Kết Nối Các Software Components Bằng Connectors

1. **Temp_Control và LED_Control**:
   - **Connector Type**: Assembly
   - **Source Component**: `Temp_Control`
     - **Port**: Provide Port `Temp_Control.PPort_LED_OnOff`
   - **Target Component**: `LED_Control`
     - **Port**: Require Port `LED_Control.RPort_LED_OnOff`
2. **Temp_Control và Heater_Control**:
   - **Connector Type**: Assembly
   - **Source Component**: `Temp_Control`
     - **Port**: Provide Port `Temp_Control.PPort_Heater_OnOff`
   - **Target Component**: `Heater_Control`
     - **Port**: Require Port `Heater_Control.RPort_Heater_OnOff`
3. **Temp_Control và IO_Temperature_Sensor**:
   - **Connector Type**: Assembly
   - **Source Component**: `Temp_Control`
     - **Port**: Require Port `Temp_Control.RPort_Temperature`
   - **Target Component**: `IO_Temperature_Sensor`
     - **Port**: Provide Port `IO_Temperature_Sensor.PPort_Temperature`
4. **LED_Control và IO_LED_Control**:
   - **Connector Type**: Assembly
   - **Source Component**: `LED_Control`
     - **Port**: Provide Port `LED_Control.PPort_LED_Command`
   - **Target Component**: `IO_LED_Control`

        - **Port**: Require Port `IO_LED_Control.RPort_LED_Command`
5. **Heater_Control và IO_Heater_Control**:
    - **Connector Type**: Assembly
    - **Source Component**: `Heater_Control`
        - **Port**: Provide Port `Heater_Control.PPort_Heater_Command`
    - **Target Component**: `IO_Heater_Control`
        - **Port**: Require Port `IO_Heater_Control.RPort_Heater_Command`

Bước 3: Kiểm Tra Và Hoàn Tất

1. **Kiểm Tra Cấu Hình**:
    - Đảm bảo rằng tất cả các ports được kết nối đúng và không có lỗi.
2. **Lưu Lại Cấu Hình**:
    - Lưu lại cấu hình của bạn sau khi hoàn tất.

Tổng Kết

Chúng ta đã tạo và kết nối các Software Components thành một Software Composition. Bằng cách này, chúng ta có thể quản lý và tương tác giữa các components một cách dễ dàng và hiệu quả.