17. RUN TIME ENVIRONMENT (RTE)

Giới thiệu về RTE

- **RTE**: Là trung tâm của kiến trúc Autosar, chịu trách nhiệm cho tất cả các giao diện giao tiếp giữa các thành phần ứng dụng hoặc giữa ứng dụng và phần mềm cơ bản.
- **Vai trò của RTE**: Đóng vai trò như một trung tâm giao tiếp cấp hệ thống cho việc trao đổi thông tin giữa các thành phần trên cùng một ECU hoặc giữa các ECU khác nhau.

Virtual Function Bus (VFB)

- **VFB**: Là biểu diễn của RTE ở mức hệ thống, cho phép sự linh hoạt trong việc đặt các thành phần phần mềm trên bất kỳ ECU nào trong hệ thống xe.
- **Tầm quan trọng của VFB**: Đảm bảo rằng các thành phần phần mềm có thể được đặt trên các lõi hoặc phân vùng khác nhau một cách dễ dàng.

Lợi ích của RTE

1. **Tính linh hoạt**:
    - Đảm bảo các thành phần phần mềm có thể được phân phối dễ dàng trên các lõi hoặc phân vùng khác nhau của bộ điều khiển đa lõi.
    - Quản lý giao tiếp giữa các lõi một cách tối ưu.
2. **Tính nhất quán của thông điệp**:
    - Đảm bảo tính nhất quán của thông điệp khi truyền các giao diện giữa các ứng dụng, không phụ thuộc vào lõi hoặc độ ưu tiên của giao diện.
3. **Lập lịch runnable**:
    - RTE chịu trách nhiệm lập lịch các runnable dựa trên các sự kiện đã cấu hình.
    - Đảm bảo rằng các runnable được gọi theo cấu hình sự kiện.

Trách nhiệm của RTE

- **Giao tiếp giao diện**: Đảm bảo giao tiếp giữa các thành phần phần mềm.
- **Tính nhất quán của thông điệp**: Đảm bảo thông điệp nhất quán giữa các thành phần phần mềm.
- **Lập lịch runnable**: Dựa trên các sự kiện đã cấu hình.

Tài liệu tiêu chuẩn RTE của Autosar

1. **Tài liệu yêu cầu (Requirement Specification)**: Mô tả các yêu cầu của RTE.
2. **Tài liệu phần mềm (Software Specification)**: Mô tả chi tiết về cách thức RTE hoạt động.

API của RTE

- **RTE cung cấp các API**: Cho phép lớp ứng dụng giao tiếp với các giao diện và sử dụng các tính năng do RTE cung cấp.

- **Tóm tắt API**: Các API chính bao gồm giao diện gửi nhận và giao diện khách hàng-máy chủ, được sử dụng chủ yếu trong phát triển phần mềm.

Ví dụ về API RTE

- **RTE_Read API**: Được sử dụng để đọc dữ liệu từ các cổng.
- **Cách sử dụng**: API này sẽ được tham chiếu và sử dụng trong các quá trình phát triển phần mềm.

Ví dụ cấu hình API:

```
#include "RTE_ComponentName.h"

void MyFunction(void) {
    int data;
    RTE_Read_DataPort(&data);
    // Process data
}
```

Kết luận

- **RTE**: Là phần quan trọng trong kiến trúc Autosar, đảm bảo giao tiếp nhất quán và linh hoạt giữa các thành phần phần mềm.
- **Tài liệu tiêu chuẩn RTE**: Được cung cấp bởi Autosar, giúp hiểu rõ hơn về cách thức hoạt động và cấu hình của RTE.
- **API của RTE**: Cung cấp các chức năng cần thiết để giao tiếp và sử dụng các tính năng của RTE trong quá trình phát triển phần mềm.

Trong các phần tiếp theo, chúng ta sẽ đi sâu vào các API cụ thể của RTE và cách sử dụng chúng trong phát triển phần mềm Autosar.

18. RTE - SENDER RECEIVER INTERFACE

Giới thiệu

Để sử dụng các API của RTE trong quá trình phát triển tệp C, chúng ta phải dẫn xuất từ các cấu hình. Dưới đây là cách sử dụng các API này để viết hoặc nhận dữ liệu từ cổng cung cấp (provider port) hoặc cổng nhận (receiver port).

Ví dụ Cấu hình

Giả sử chúng ta có 2 cổng, một cổng cung cấp và một cổng nhận, được định nghĩa trong cấu hình của chúng ta và chúng được ánh xạ đến giao diện gửi nhận.

Cách viết API RTE_Write

1. **API RTE_Write**: Trả về trạng thái của hoạt động sử dụng kiểu trả về chuẩn.
2. **Định danh hàm**:
    - **RTE_Write_**: Tiền tố của hàm.
    - **[component_name]_**: Tùy chọn, có thể bỏ qua.
    - **PortName**: Tên của cổng.
    - **VariableDataPrototype**: Tên của biến dữ liệu từ giao diện gửi nhận.

Ví dụ Cụ thể

- **Tên cổng cung cấp (Provider Port)**: `PP_Speed`
- **Tên biến dữ liệu (Variable Data Prototype)**: `Speed`

Cấu hình hàm RTE_Write:

```
#include "RTE_ComponentName.h"

void Function_1() {
    float speed = 60.5;
    Std_ReturnType status = RTE_Write_PP_Speed_Speed(speed);
    // Kiểm tra trạng thái và xử lý tiếp theo
}
```

- **RTE_Write_PP_Speed_Speed**:
    - **RTE_Write**: Tiền tố hàm.
    - **PP_Speed**: Tên cổng cung cấp.
    - **Speed**: Tên biến dữ liệu.

Cách viết API RTE_Read

1. **API RTE_Read**: Tương tự như RTE_Write, nhưng hướng của tham số là đầu vào kiểu con trỏ để RTE có thể ghi dữ liệu vào đó.
2. **Định danh hàm**:

- **RTE_Read_**: Tiền tố của hàm.
- **[component_name]_**: Tùy chọn, có thể bỏ qua.
- **PortName**: Tên của cổng.
- **VariableDataPrototype**: Tên của biến dữ liệu từ giao diện gửi nhận.

Cấu hình hàm RTE_Read:

```
c
Sao chép mã
#include "RTE_ComponentName.h"

void Function_2() {
    float speed;
    Std_ReturnType status = RTE_Read_RP_Speed_Speed(&speed);
    // Sử dụng dữ liệu speed cho các tính toán tiếp theo
}
```

- **RTE_Read_RP_Speed_Speed**:
  - **RTE_Read**: Tiền tố hàm.
  - **RP_Speed**: Tên cổng nhận.
  - **Speed**: Tên biến dữ liệu.

Tóm tắt

- **RTE_Write**: Dùng để ghi dữ liệu vào cổng cung cấp. Hàm được cấu hình với tên cổng và biến dữ liệu.
- **RTE_Read**: Dùng để nhận dữ liệu từ cổng nhận. Hàm được cấu hình tương tự nhưng tham số là kiểu con trỏ.

Bằng cách sử dụng các API RTE_Write và RTE_Read, chúng ta có thể giao tiếp hiệu quả giữa các thành phần phần mềm trong Autosar. Các hàm này đảm bảo rằng dữ liệu được truyền tải nhất quán và đúng cách giữa các cổng đã cấu hình.

19. RTE - CLIENT SERVER INTERFACE

Giới thiệu

Giao diện khách hàng-máy chủ (Client Server Interface) được sử dụng để gọi hàm hoặc dịch vụ từ một module khác. Trong ví dụ này, chúng ta có một giao diện khách hàng-máy chủ được định nghĩa với một thao tác gọi là "Sum".

Định nghĩa API RTE_Call

1. **Tiền tố (Prefix)**: `RTE_Call`
2. **Tên cổng nhận (Receiver Port Name)**: Tên của cổng nhận trên client.
3. **Tên thao tác khách hàng-máy chủ (Client Server Operation Name)**: Tên của thao tác được cấu hình trong giao diện khách hàng-máy chủ.
4. **Các tham số (Arguments)**: Có thể thuộc ba loại dựa trên hướng (IN, OUT, INOUT).

Ví dụ Cấu hình

- **Receiver Port**: `RP_Calculator`
- **Client Server Operation**: `Sum`
- **Tham số**: `X` và `Y` (đầu vào IN), `SumResult` (đầu ra OUT)

Viết mã C cho Client

```
#include "RTE_ComponentName.h"

void Function_1() {
    int X = 5;
    int Y = 10;
    int SumResult;

    // Gọi hàm server thông qua API RTE_Call
    Std_ReturnType status = RTE_Call_RP_Calculator_Sum(X, Y,
&SumResult);

    // Kiểm tra trạng thái và xử lý kết quả
    if (status == RTE_E_OK) {
        // Sử dụng SumResult cho các xử lý tiếp theo
    } else {
        // Xử lý lỗi nếu có
    }
}
```

Phân tích mã C

1. **Tiền tố hàm**: `RTE_Call`
2. **Tên cổng nhận**: `RP_Calculator`
3. **Tên thao tác khách hàng-máy chủ**: `Sum`
4. **Tham số**:

- `X` và `Y` là các tham số đầu vào (IN), được truyền trực tiếp.
- `SumResult` là tham số đầu ra (OUT), được truyền dưới dạng con trỏ để server có thể ghi kết quả vào đó.

Mã RTE được tạo

Khi gọi hàm `RTE_Call_RP_Calculator_Sum`, RTE sẽ tạo mã tương ứng để gọi hàm trên server. Ví dụ về mã RTE được tạo:

```
Std_ReturnType RTE_Call_RP_Calculator_Sum(int X, int Y, int *SumResult)
{
    // Gọi hàm server thực tế
    return Sum(X, Y, SumResult);
}
```

Trong mã trên:

- Hàm `Sum` trên server được gọi với các tham số `X`, `Y` và `SumResult`.
- Trả về trạng thái `RTE_E_OK` nếu thành công hoặc lỗi nếu có vấn đề.

Tóm tắt

- **RTE_Call API**: Được sử dụng để gọi hàm hoặc dịch vụ từ một module khác trong giao diện khách hàng-máy chủ.
- **Cấu hình hàm**: Bao gồm tiền tố `RTE_Call`, tên cổng nhận, tên thao tác khách hàng-máy chủ và các tham số.
- **Tham số**: Được truyền dựa trên hướng (IN, OUT, INOUT).

Bằng cách sử dụng API RTE_Call, chúng ta có thể dễ dàng thực hiện các cuộc gọi hàm giữa các module phần mềm trong Autosar. Các API này đảm bảo rằng giao tiếp giữa các thành phần phần mềm diễn ra một cách nhất quán và hiệu quả.

20. RTE - COMMUNICATION

Ví dụ điển hình

Giả sử chúng ta có hai thành phần phần mềm (SWC) cần giao tiếp với nhau thông qua RTE:

- **Component-1 (SWC1)**: Có cổng cung cấp để gửi dữ liệu ra ngoài.
- **Component-2 (SWC2)**: Có cổng nhận để nhận dữ liệu.

Giao tiếp từ Component-1 (SWC1)

1. **Tệp C của SWC1**:
    - Chứa một runnable hoặc hàm tính toán dữ liệu cần gửi và viết nó vào RTE thông qua giao diện Autosar hoặc API RTE.
    - Ví dụ: `RTE_Write_PP_Speed_Speed(data);`
2. **RTE từ phía Component-1**:
    - RTE định nghĩa các API và thực hiện chức năng cho chúng.
    - RTE tạo một giao diện từ phía mình và sao chép giao diện này từ Component-1.

Giao tiếp từ Component-2 (SWC2)

1. **Tệp C của SWC2**:
    - Chứa runnable hoặc hàm để đọc dữ liệu được gửi từ Component-1.
    - Ví dụ: `RTE_Read_RP_Speed_Speed(&data);`
2. **RTE từ phía Component-2**:
    - RTE thực hiện API đọc để sao chép dữ liệu từ giao diện trung gian của RTE vào biến con trỏ mà ứng dụng truyền vào.

Tại sao lại cần RTE?

Lý do không giao tiếp trực tiếp

- Trước đây, giao tiếp trực tiếp từ một module sang module khác là dễ dàng, nhưng khi độ phức tạp của phần mềm tăng lên với hàng ngàn thành phần trong một hệ thống, việc quản lý trở nên khó khăn.

Lợi ích của RTE

1. **Khả năng thay thế linh hoạt**:
    - Khách hàng có thể thay thế một thành phần phần mềm mà không cần thay đổi toàn bộ hệ thống.
    - Ví dụ: Khách hàng không hài lòng với Component-2 có thể dễ dàng thay thế bằng một thành phần mới từ nhà cung cấp khác.
2. **Dễ dàng cắm và chạy (Plug and Play)**:
    - RTE cho phép các thành phần phần mềm được cắm vào và sử dụng mà không cần phải thay đổi các thành phần khác.

    - Việc trao đổi các thành phần chỉ cần điều chỉnh RTE mà không ảnh hưởng đến phần mềm còn lại.
3. **Tính nhất quán của thông điệp**:
    - RTE đảm bảo tính nhất quán của thông điệp khi truyền giữa các thành phần phần mềm.
4. **Quản lý phức tạp**:
    - RTE xử lý phần phức tạp của việc đảm bảo tính nhất quán của thông điệp, giảm bớt gánh nặng cho nhà phát triển ứng dụng.

Ví dụ minh họa

Giả sử SWC1 có hàm tính toán tốc độ và gửi nó ra ngoài:

```
#include "RTE_Component1.h"

void Function_1() {
    int speed = calculateSpeed();
    RTE_Write_PP_Speed_Speed(speed);
}
```

RTE từ phía Component-1 sẽ định nghĩa API và xử lý viết dữ liệu:

```
Std_ReturnType RTE_Write_PP_Speed_Speed(int speed) {
    // Thực hiện việc sao chép dữ liệu và quản lý tính nhất quán
    RTE_SpeedValue = speed;
    return RTE_E_OK;
}
```

Tương tự, SWC2 có hàm đọc dữ liệu từ RTE:

```
#include "RTE_Component2.h"

void Function_2() {
    int speed;
    RTE_Read_RP_Speed_Speed(&speed);
    processSpeed(speed);
}
```

RTE từ phía Component-2 sẽ thực hiện API đọc:

```
Std_ReturnType RTE_Read_RP_Speed_Speed(int* speed) {
    *speed = RTE_SpeedValue;
    return RTE_E_OK;
}
```

Kết luận

- **RTE**: Đảm bảo giao tiếp giữa các thành phần phần mềm một cách nhất quán và hiệu quả.
- **Lợi ích**: Tính linh hoạt, dễ dàng thay thế thành phần, đảm bảo tính nhất quán của thông điệp, và quản lý phần mềm phức tạp.

- **API**: Sử dụng các API như `RTE_Write` và `RTE_Read` để giao tiếp giữa các thành phần phần mềm mà không cần phải lo lắng về chi tiết quản lý thông điệp.

RTE giúp đơn giản hóa việc phát triển và quản lý phần mềm trong hệ thống Autosar, cho phép nhà phát triển tập trung vào các chức năng chính mà không phải lo lắng về việc đảm bảo tính nhất quán và giao tiếp giữa các thành phần.

21. RTE - SCHEDULING OF EVENTS

Giới thiệu

RTE không chỉ đảm bảo giao tiếp giữa các thành phần phần mềm mà còn quản lý việc lập lịch (scheduling) của các Runnable. Hãy xem cách RTE xử lý việc lập lịch này thông qua ví dụ trước.

Ví dụ minh họa

Giả sử chúng ta có hai chức năng:

- **Function_1**: Mapped to a Timing Event (lập lịch theo thời gian).
- **Function_2**: Mapped to a Data Received Event (lập lịch khi nhận dữ liệu mới).

Cấu hình RTE cho Lập lịch

Cần có cấu hình RTE riêng cho hoạt động lập lịch này, được thực hiện ở giai đoạn cuối của quá trình phát triển hệ thống, gọi là Event To Task mapping trong cấu hình RTE, cùng với cấu hình bộ chứa tác vụ (task container) trong cấu hình OS ECU.

Ví dụ về Mapping

- **Function_1** được ánh xạ tới một sự kiện thời gian và được ánh xạ tới "CalcTask".
- **Function_2** được ánh xạ tới một sự kiện nhận dữ liệu và được ánh xạ tới "ReadTask".

Giả sử OS đã được cấu hình để có hai tác vụ gọi là "CalcTask" và "ReadTask".

Tạo tác vụ từ cấu hình

Dựa trên cấu hình này, RTE sẽ tạo các thân tác vụ (task bodies) cho từng tác vụ cá nhân mà chúng ta đã cấu hình.

- RTE sẽ thêm tiền tố `RTE_Task` cho mỗi tác vụ mà nó tạo ra, tiếp theo là tên của tác vụ từ cấu hình của chúng ta.

Ví dụ về tên tác vụ:

- Nếu chúng ta lấy tên là "CalcTask", tác vụ cuối cùng mà RTE tạo ra sẽ có tên `RTE_Task_CalcTask`.

Quản lý tác vụ theo thời gian

Function_1 được ánh xạ tới sự kiện thời gian (Timing Event)

- Tên tác vụ: `RTE_Task_CalcTask`
- Tác vụ này được thêm vào bộ lập lịch của hệ điều hành.

```
void RTE_Task_CalcTask() {
    // Gọi Function_1 định kỳ dựa trên thời gian cấu hình
    Function_1();
}
```

- `Function_1` sẽ được gọi định kỳ dựa trên khoảng thời gian được cấu hình trong sự kiện thời gian.

Quản lý sự kiện nhận dữ liệu

Function_2 được ánh xạ tới sự kiện nhận dữ liệu (Data Received Event)

- Tên tác vụ: `RTE_Task_ReadTask`

```
void RTE_Task_ReadTask() {
    if (RTE_Flag_DataReceived) {
        Function_2();
    }
}
```

- RTE thêm một cơ chế cờ để đảm bảo rằng hàm này chỉ được gọi khi dữ liệu mới được nhận trên cổng này.
- Cờ `RTE_Flag_DataReceived` được đặt tại API `RTE_Write` mỗi khi có dữ liệu mới.

Cập nhật API RTE_Write để đặt cờ

```
Std_ReturnType RTE_Write_PP_Speed_Speed(int speed) {
    // Sao chép dữ liệu và quản lý tính nhất quán
    RTE_SpeedValue = speed;
    // Đặt cờ khi dữ liệu mới được nhận
    RTE_Flag_DataReceived = 1;
    return RTE_E_OK;
}
```

Tóm tắt

- **RTE quản lý lịch trình**: Dựa trên loại sự kiện, RTE đảm bảo rằng các runnables được gọi đúng cách theo cấu hình sự kiện.
- **Cấu hình tác vụ và lập lịch**: RTE tạo các tác vụ và quản lý việc gọi hàm dựa trên các cấu hình này.
- **Lợi ích của RTE**: Cung cấp khả năng quản lý linh hoạt và hiệu quả cho việc lập lịch và giao tiếp giữa các thành phần phần mềm.

Câu hỏi tiếp theo

Với việc thấy các mã của RTE cho đến bây giờ, bạn có thể tự hỏi ai viết mã RTE này hoặc mã này đến từ đâu. Đây là điều chúng ta sẽ xem xét tiếp theo.

Chúng ta sẽ xem xét nguồn gốc của mã RTE và cách nó được tạo ra trong các phần tiếp theo.

22. RTE GENERATOR

Giới thiệu

Lớp RTE trong kiến trúc Autosar được tạo ra chủ yếu từ các cấu hình và cần có các công cụ đặc biệt để tạo ra lớp này. Những công cụ này thường được cấp phép và không dễ dàng tiếp cận. Tuy nhiên, việc hiểu cách chúng hoạt động và cách chúng xử lý đầu vào và đầu ra là rất quan trọng.

Quy trình Tạo RTE

1. **Đầu vào của Generator**: Các cấu hình ARXML bao gồm:
    - **Mô tả thành phần phần mềm (Software Component Description)**: Các cấu hình từ quá trình phát triển thành phần phần mềm.
    - **Cấu hình trích xuất ECU (ECU Extract Configuration)**: Các cấu hình liên quan đến một ECU cụ thể.
    - **Cấu hình RTE và OS ECU**: Định nghĩa các bộ chứa tác vụ OS và ánh xạ sự kiện đến các tác vụ.
    - **Cấu hình phiên bản ECU (ECU Instance Configuration)**: Mô tả của đơn vị điều khiển tương ứng.
    - **Cấu hình phần mềm cơ bản (BSW)**: Các mô-đun phần mềm cơ bản.
2. **Đầu ra của Generator**:
    - **Tệp Rte.c**: Chứa mã đầy đủ cho giao tiếp giữa các thành phần phần mềm, định nghĩa giao diện RTE, v.v.
    - **Tệp tiêu đề RTE cho từng thành phần phần mềm**: Định dạng là `Rte_ + Tên thành phần + ".h"`.
    - **Các tệp hỗ trợ khác**:
        - **OS related ARXML**: Thông tin lập lịch của các tác vụ được tạo.
        - **IOC (Inter OS Communication) Extract File**: Thông tin giao tiếp qua các lõi hoặc phân vùng.
        - **MCU support data**: Thông tin đo lường và hiệu chuẩn (calibration).
        - **BSWMD file**: Thông tin ánh xạ bộ nhớ cho các giao diện RTE được tạo.

Ví dụ về Quy trình Tạo RTE

1. **Cấu hình thành phần phần mềm (SWC1 và SWC2)**:
    - SWC1 có một runnable `Function_1` được ánh xạ tới sự kiện thời gian.
    - SWC2 có một runnable `Function_2` được ánh xạ tới sự kiện nhận dữ liệu.
2. **Tạo mã RTE**:
    - **RTE_Write API**: Được tạo để ghi dữ liệu vào cổng cung cấp của SWC1.
    - **RTE_Read API**: Được tạo để đọc dữ liệu từ cổng nhận của SWC2.
    - **Task Functions**:

```
void RTE_Task_CalcTask() {
    Function_1();
}

void RTE_Task_ReadTask() {
```

```
        if (RTE_Flag_DataReceived) {
            Function_2();
        }
    }
```

3. **Tệp tiêu đề RTE**

```
#include "Rte_SWC1.h"
#include "Rte_SWC2.h"
```

4. **Các tệp hỗ trợ**:
    - **OS related ARXML**: Chứa thông tin lập lịch cho các tác vụ.
    - **IOC Extract File**: Thông tin giao tiếp qua các lõi hoặc phân vùng.
    - **MCU support data**: Thông tin đo lường và hiệu chuẩn.
    - **BSWMD file**: Thông tin ánh xạ bộ nhớ.

Các Công Cụ Phát Triển Autosar

Có nhiều công cụ phát triển Autosar trên thị trường, được phân loại theo các mục đích khác nhau:

1. **Công cụ thực hiện BSW hoặc MCAL**.
2. **Công cụ cấu hình BSW**.
3. **Công cụ tạo RTE**: Tạo lớp RTE từ tất cả các cấu hình.
4. **Trình chỉnh sửa XML đơn giản**: Sử dụng để tạo và chỉnh sửa các tệp ARXML.

Ví dụ về Các Công Cụ Autosar

- **Vector DaVinci Developer**
- **ETAS ISOLAR**
- **Mentor Graphics Volcano**
- **Electrobit Tresos Studio**

Tóm tắt

- **Lớp RTE**: Được tạo ra chủ yếu từ các cấu hình ARXML bằng cách sử dụng các công cụ tạo đặc biệt.
- **Đầu vào của RTE Generator**: Bao gồm các cấu hình phần mềm, ECU, OS, và BSW.
- **Đầu ra của RTE Generator**: Gồm các tệp mã nguồn và tiêu đề RTE, cùng với các tệp hỗ trợ khác.
- **Các công cụ phát triển Autosar**: Cung cấp khả năng tạo và cấu hình các thành phần Autosar, tuy nhiên, chúng thường có chi phí cao và không dễ tiếp cận.

Hy vọng rằng việc hiểu quy trình này giúp bạn có cái nhìn rõ ràng hơn về cách RTE layer trong Autosar được tạo ra từ các cấu hình đầu vào và các lợi ích của việc sử dụng công cụ tạo RTE. Trong các phần tiếp theo, chúng ta sẽ sử dụng trình chỉnh sửa XML đơn giản để thực hiện các ví dụ và cấu hình.