25. USE CASE WITH NON-AUTOSAR SOFTWARE

Yêu cầu: Điều khiển Quạt Làm Mát Radiator

Yêu cầu:

- ECU phải bật quạt làm mát khi nhiệt độ của nước làm mát trong radiator vượt quá 50 độ C.
- Phần cứng bao gồm cảm biến nhiệt độ kết nối với chân analog của bộ điều khiển.
- Để điều khiển quạt làm mát, ECU có một H-Bridge IC kết nối với chân IO digital của bộ điều khiển.

Phần 1: Giải pháp Không Autosar

1. Cấu trúc phần mềm không Autosar:

- Không có cấu hình cụ thể, chỉ có mã C trực tiếp.
- **Mã C**:
  - Cấu hình các chân của bộ điều khiển ở mức register và khởi tạo ADC.
  - Cấu hình các chân IO để điều khiển H-Bridge và khởi tạo H-Bridge IC.
  - Vòng lặp vô hạn để thực hiện chức năng chính:
    - Bắt đầu chuyển đổi ADC và tính toán nhiệt độ.
    - Nếu nhiệt độ vượt quá 50 độ C, bật quạt bằng cách thiết lập chân IO của H-Bridge.
    - Nếu nhiệt độ thấp hơn, tắt quạt.

2. Mã C mẫu:

```
void setup() {
    // Cấu hình chân điều khiển và khởi tạo ADC
    configure_adc_pins();
    initialize_adc();

    // Cấu hình chân IO để điều khiển H-Bridge
    configure_io_pins();
    initialize_h_bridge();
}

void loop() {
    while(1) {
        // Đọc nhiệt độ từ cảm biến
        int temperature = read_temperature_from_adc();

        // Điều khiển quạt làm mát
        if (temperature > 50) {
            turn_on_cooling_fan();
        } else {
            turn_off_cooling_fan();
        }
    }
```

```
}
```

3. Nhược điểm của phần mềm không Autosar:

- **Phụ thuộc chặt chẽ vào phần cứng**:
  - Thay đổi bộ điều khiển vi mạch (microcontroller) đòi hỏi phải làm lại mã phần mềm liên quan đến cấu hình chân điều khiển và xử lý chuyển đổi ADC.
  - Thay đổi H-Bridge IC cũng đòi hỏi phải thay đổi mã phần mềm để tương thích với H-Bridge mới.
- **Khó khăn trong tái sử dụng và chuyển đổi**:
  - Khách hàng muốn tái sử dụng phần mềm đã phát triển trên các ECU khác sẽ gặp khó khăn vì phần cứng thay đổi đòi hỏi phải làm lại mã phần mềm.
  - Phụ thuộc hoàn toàn vào một nhà cung cấp phần cứng, không thể dễ dàng thay đổi nhà cung cấp nếu có vấn đề về chi phí hoặc chất lượng.

Phần 2: Giải pháp Autosar

1. Cấu trúc phần mềm Autosar:

- Sử dụng lớp trừu tượng hóa phần mềm (RTE) để tách biệt phần mềm ứng dụng khỏi phần cứng.
- Các thành phần phần mềm được phát triển và cấu hình theo tiêu chuẩn Autosar.

2. Thiết kế phần mềm Autosar:

Hệ thống gồm các thành phần:

- **Thành phần phần mềm điều khiển nhiệt độ (Temperature Control SWC)**: Đọc dữ liệu nhiệt độ từ cảm biến.
- **Thành phần phần mềm điều khiển quạt (Fan Control SWC)**: Quyết định bật/tắt quạt dựa trên dữ liệu nhiệt độ.

3. Cấu hình phần mềm:

- **Temperature Control SWC**:
  - **Ports**:
    - **ProvidePort (TemperatureData)**: Gửi dữ liệu nhiệt độ.
  - **Runnable**:
    - **ReadTemperature**: Đọc nhiệt độ từ cảm biến và gửi qua port.

```
void ReadTemperature() {
    int temperature = read_temperature_from_adc();
    Rte_Write_TemperatureData(temperature);
}
```

- **Fan Control SWC**:
  - **Ports**:

        - **RequirePort (TemperatureData)**: Nhận dữ liệu nhiệt độ.
    - **Runnable**:
        - **ControlFan**: Nhận dữ liệu nhiệt độ và điều khiển quạt.

```
void ControlFan() {
    int temperature;
    Rte_Read_TemperatureData(&temperature);
    if (temperature > 50) {
        turn_on_cooling_fan();
    } else {
        turn_off_cooling_fan();
    }
}
```

4. Lợi ích của phần mềm Autosar:

- **Tách biệt phần mềm và phần cứng**:
    - Thay đổi phần cứng (microcontroller, H-Bridge IC) không ảnh hưởng lớn đến mã phần mềm ứng dụng vì sự trừu tượng hóa của RTE.
    - Dễ dàng tái sử dụng phần mềm trên các ECU khác nhau mà không cần làm lại mã phần mềm.
- **Giảm chi phí và thời gian phát triển**:
    - Giảm đáng kể chi phí và thời gian cần thiết để phát triển lại phần mềm khi có thay đổi phần cứng.
    - Khách hàng có thể dễ dàng chuyển đổi nhà cung cấp phần cứng nếu cần thiết.
- **Tính linh hoạt và khả năng mở rộng**:
    - Tính linh hoạt trong phát triển và mở rộng phần mềm.
    - RTE xử lý giao tiếp giữa các thành phần phần mềm và các ECU khác nhau.

Tổng kết

Phần mềm không Autosar dễ bị ảnh hưởng bởi những thay đổi về phần cứng và khó khăn trong việc tái sử dụng và chuyển đổi. Trong khi đó, Autosar cung cấp một giải pháp linh hoạt, dễ dàng bảo trì và mở rộng, phù hợp với các yêu cầu phát triển phần mềm trong ngành công nghiệp ô tô hiện đại.

26. THEORY - MCAL LAYER DEVELOPMENT

Phân tích Yêu cầu và Phân chia Kiến trúc

Yêu cầu:

- ECU phải bật quạt làm mát khi nhiệt độ của nước làm mát trong radiator vượt quá 50 độ C.

Phân chia Kiến trúc:

- **Lớp ứng dụng (Application Layer)**: Thành phần phần mềm để điều khiển quạt làm mát.
- **Lớp trừu tượng hóa IO (IO Abstraction Layer)**: Lớp trung gian giữa phần mềm ứng dụng và lớp điều khiển phần cứng.
- **Lớp MCAL (Microcontroller Abstraction Layer)**: Truy cập trực tiếp vào các chân IO của bộ điều khiển vi mạch.

Cấu hình yêu cầu:

- **3 chân IO**: Để điều khiển H-Bridge IC.
- **1 chân ADC**: Để đọc dữ liệu từ cảm biến nhiệt độ.

Thiết kế Chi tiết và Cấu hình

1. Lớp MCAL IO Driver:

- **DIO Driver**: Đọc và ghi chân IO.
- **Port Driver**: Cấu hình các chân của bộ điều khiển.
- **ADC Driver**: Xử lý chuyển đổi ADC.

2. Lớp trừu tượng hóa IO:

- **Abstraction cho cảm biến nhiệt độ**: Đọc dữ liệu nhiệt độ.
- **Abstraction cho H-Bridge**: Điều khiển H-Bridge.

3. Lớp ứng dụng:

- **Thành phần phần mềm quạt làm mát**: Đọc dữ liệu từ cảm biến và điều khiển quạt.

Cấu hình Chi tiết

1. Cấu hình Chân Điều khiển (Port Driver):

```
// Cấu hình chân ADC
PortConfigType PortConfig[] = {
    {PORT_PIN_ADC_CHANNEL, PORT_PIN_IN, PORT_PIN_MODE_ADC},
    {PORT_PIN_HBRIDGE_CONTROL_1, PORT_PIN_OUT, PORT_PIN_MODE_DIO},
```

```
    {PORT_PIN_HBRIDGE_CONTROL_2, PORT_PIN_OUT, PORT_PIN_MODE_DIO},
    {PORT_PIN_HBRIDGE_CONTROL_3, PORT_PIN_OUT, PORT_PIN_MODE_DIO}
};
```

2. Triển khai Driver IO (DIO Driver):

```
// Đọc chân IO
Dio_LevelType Dio_ReadChannel(Dio_ChannelType ChannelId) {
    // Đọc trạng thái chân IO và trả về STD_HIGH hoặc STD_LOW
}

// Ghi chân IO
void Dio_WriteChannel(Dio_ChannelType ChannelId, Dio_LevelType Level) {
    // Ghi trạng thái cho chân IO
}
```

3. Triển khai Driver ADC (ADC Driver):

```
// Bắt đầu chuyển đổi ADC
void Adc_StartGroupConversion(Adc_GroupType Group) {
    // Bắt đầu chuyển đổi ADC cho nhóm đã chỉ định
}

// Đọc dữ liệu từ ADC
Std_ReturnType Adc_ReadGroup(Adc_GroupType Group, Adc_ValueGroupType*
DataBufferPtr) {
    // Đọc dữ liệu đã chuyển đổi và ghi vào buffer
}
```

4. Lớp trừu tượng hóa IO (Abstraction Layer):

- **Abstraction cho cảm biến nhiệt độ**:
  - **Đọc nhiệt độ từ cảm biến**:

```
int ReadTemperature() {
    int temperature;
    Adc_StartGroupConversion(TEMP_SENSOR_GROUP);
    Adc_ReadGroup(TEMP_SENSOR_GROUP, &temperature);
    return temperature;
}
```

- **Abstraction cho H-Bridge**:
  - **Điều khiển quạt:**

```
void ControlFan(Dio_LevelType state) {
    Dio_WriteChannel(HBRIDGE_CONTROL_PIN, state);
}
```

5. Thành phần phần mềm ứng dụng (Application Component):

```
void CoolingFanController() {
    int temperature = ReadTemperature();
```

```
    if (temperature > 50) {
        ControlFan(STD_HIGH); // Bật quạt
    } else {
        ControlFan(STD_LOW); // Tắt quạt
    }
}
```

6. Cấu hình RTE và sự kiện:

- **Runnable**:
  - **CoolingFanController**: Được ánh xạ tới sự kiện thời gian (periodic event) hoặc sự kiện dữ liệu nhận (data received event).

Quy trình Tổng quát

1. **Cấu hình Port Driver**: Cấu hình chân ADC và chân IO điều khiển H-Bridge.
2. **Triển khai DIO và ADC Driver**: Viết các hàm để đọc và ghi chân IO, chuyển đổi ADC.
3. **Lớp trừu tượng hóa IO**: Tạo các hàm để đọc nhiệt độ và điều khiển H-Bridge.
4. **Thành phần phần mềm ứng dụng**: Tạo hàm điều khiển quạt dựa trên nhiệt độ.
5. **Cấu hình RTE và sự kiện**: Ánh xạ hàm điều khiển quạt tới sự kiện thời gian hoặc sự kiện dữ liệu nhận.

Kết luận

Việc triển khai phần mềm theo tiêu chuẩn Autosar giúp tách biệt phần cứng và phần mềm, dễ dàng bảo trì và nâng cấp. RTE đảm bảo rằng các giao tiếp giữa các thành phần phần mềm được quản lý và lên lịch một cách chính xác, giảm thiểu sự phụ thuộc giữa phần mềm ứng dụng và phần cứng. Điều này không chỉ giảm chi phí và thời gian phát triển mà còn cung cấp tính linh hoạt và khả năng mở rộng cho hệ thống.

27. DEMO - MCAL LAYER DEVELOPMENT

Phân tích Yêu cầu và Phân chia Kiến trúc

Yêu cầu:

- ECU phải bật quạt làm mát khi nhiệt độ của nước làm mát trong radiator vượt quá 50 độ C.

Phân chia Kiến trúc:

- **Lớp ứng dụng (Application Layer)**: Thành phần phần mềm để điều khiển quạt làm mát dựa trên nhiệt độ.
- **Lớp trừu tượng hóa IO (IO Abstraction Layer)**: Lớp trung gian giữa phần mềm ứng dụng và lớp điều khiển phần cứng.
- **Lớp MCAL (Microcontroller Abstraction Layer)**: Truy cập trực tiếp vào các chân IO của bộ điều khiển vi mạch.

Cấu hình yêu cầu:

- **3 chân IO**: Để điều khiển H-Bridge IC.
- **1 chân ADC**: Để đọc dữ liệu từ cảm biến nhiệt độ.

Thiết kế Chi tiết và Cấu hình

1. Lớp MCAL IO Driver:

- **DIO Driver**: Đọc và ghi chân IO.
- **Port Driver**: Cấu hình các chân của bộ điều khiển.
- **ADC Driver**: Xử lý chuyển đổi ADC.

2. Lớp trừu tượng hóa IO:

- **Abstraction cho cảm biến nhiệt độ**: Đọc dữ liệu nhiệt độ.
- **Abstraction cho H-Bridge**: Điều khiển H-Bridge.

3. Lớp ứng dụng:

- **Thành phần phần mềm quạt làm mát**: Đọc dữ liệu từ cảm biến và điều khiển quạt.

Chúng ta sẽ triển khai từng lớp theo thứ tự.

Cấu hình Chi tiết

1. Cấu hình Chân Điều khiển (Port Driver):

```
// Cấu hình chân ADC
PortConfigType PortConfig[] = {
    {PORT_PIN_ADC_CHANNEL, PORT_PIN_IN, PORT_PIN_MODE_ADC},
    {PORT_PIN_HBRIDGE_CONTROL_1, PORT_PIN_OUT, PORT_PIN_MODE_DIO},
    {PORT_PIN_HBRIDGE_CONTROL_2, PORT_PIN_OUT, PORT_PIN_MODE_DIO},
    {PORT_PIN_HBRIDGE_CONTROL_3, PORT_PIN_OUT, PORT_PIN_MODE_DIO}
};
```

2. Triển khai Driver IO (DIO Driver):

```
// Đọc chân IO
Dio_LevelType Dio_ReadChannel(Dio_ChannelType ChannelId) {
    // Đọc trạng thái chân IO và trả về STD_HIGH hoặc STD_LOW
}

// Ghi chân IO
void Dio_WriteChannel(Dio_ChannelType ChannelId, Dio_LevelType Level)
{
    // Ghi trạng thái cho chân IO
}
```

3. Triển khai Driver ADC (ADC Driver):

```
// Bắt đầu chuyển đổi ADC
void Adc_StartGroupConversion(Adc_GroupType Group) {
    // Bắt đầu chuyển đổi ADC cho nhóm đã chỉ định
}

// Đọc dữ liệu từ ADC
Std_ReturnType Adc_ReadGroup(Adc_GroupType Group, Adc_ValueGroupType*
DataBufferPtr) {
    // Đọc dữ liệu đã chuyển đổi và ghi vào buffer
}
```

4. Lớp trừu tượng hóa IO (Abstraction Layer):

- **Abstraction cho cảm biến nhiệt độ**:
  - **Đọc nhiệt độ từ cảm biến**:

```
int ReadTemperature() {
    int temperature;
    Adc_StartGroupConversion(TEMP_SENSOR_GROUP);
    Adc_ReadGroup(TEMP_SENSOR_GROUP, &temperature);
    return temperature;
}
```

- **Abstraction cho H-Bridge**:
  - **Điều khiển quạt**:

```
void ControlFan(Dio_LevelType state) {
    Dio_WriteChannel(HBRIDGE_CONTROL_PIN, state);
}
```

5. Thành phần phần mềm ứng dụng (Application Component):

```
void CoolingFanController() {
    int temperature = ReadTemperature();
    if (temperature > 50) {
        ControlFan(STD_HIGH); // Bật quạt
    } else {
        ControlFan(STD_LOW); // Tắt quạt
    }
}
```

6. Cấu hình RTE và sự kiện:

- **Runnable**:
    - **CoolingFanController**: Được ánh xạ tới sự kiện thời gian (periodic event) hoặc sự kiện dữ liệu nhận (data received event).

Quy trình Tổng quát

1. **Cấu hình Port Driver**: Cấu hình chân ADC và chân IO điều khiển H-Bridge.
2. **Triển khai DIO và ADC Driver**: Viết các hàm để đọc và ghi chân IO, chuyển đổi ADC.
3. **Lớp trừu tượng hóa IO**: Tạo các hàm để đọc nhiệt độ và điều khiển H-Bridge.
4. **Thành phần phần mềm ứng dụng**: Tạo hàm điều khiển quạt dựa trên nhiệt độ.
5. **Cấu hình RTE và sự kiện**: Ánh xạ hàm điều khiển quạt tới sự kiện thời gian hoặc sự kiện dữ liệu nhận.

Kết luận

Việc triển khai phần mềm theo tiêu chuẩn Autosar giúp tách biệt phần cứng và phần mềm, dễ dàng bảo trì và nâng cấp. RTE đảm bảo rằng các giao tiếp giữa các thành phần phần mềm được quản lý và lên lịch một cách chính xác, giảm thiểu sự phụ thuộc giữa phần mềm ứng dụng và phần cứng. Điều này không chỉ giảm chi phí và thời gian phát triển mà còn cung cấp tính linh hoạt và khả năng mở rộng cho hệ thống.

28. THEORY - ECU ABSTRACTION LAYER DEVELOPMENT

Thành phần Trừu tượng cảm biến nhiệt độ (Temperature Sensor Abstraction Component)

Chức năng:

- Tương tác với driver ADC để thực hiện chuyển đổi tín hiệu và đọc lại dữ liệu từ cảm biến nhiệt độ.
- Thực hiện các điều chỉnh offset, chuyển đổi nội bộ liên quan đến nhiệt độ.

Triển khai Mã C:

```
#include "Adc.h"
#include "Dio.h"

#define TEMP_SENSOR_CHANNEL 0 // Kênh ADC cho cảm biến nhiệt độ

// Hàm đọc nhiệt độ
int ReadTemperature() {
    Adc_ValueGroupType result;

    // Bắt đầu chuyển đổi ADC
    Adc_StartGroupConversion(TEMP_SENSOR_CHANNEL);

    // Đọc dữ liệu đã chuyển đổi từ ADC
    Adc_ReadGroup(TEMP_SENSOR_CHANNEL, &result);

    // Thực hiện các chuyển đổi và điều chỉnh cần thiết
    int temperature = (int)result * 0.1; // Giả sử có hệ số chuyển đổi đơn
giản

    return temperature;
}
```

Thành phần Trừu tượng H-Bridge (H-Bridge Abstraction Component)

Chức năng:

- Điều khiển H-Bridge IC để bật/tắt quạt làm mát bằng cách sử dụng các chân IO của bộ điều khiển.

Triển khai Mã C:

```
#include "Dio.h"

#define HBRIDGE_CONTROL_PIN_1 0 // Chân điều khiển H-Bridge 1
#define HBRIDGE_CONTROL_PIN_2 1 // Chân điều khiển H-Bridge 2
#define HBRIDGE_CONTROL_PIN_3 2 // Chân điều khiển H-Bridge 3

// Hàm khởi động động cơ quạt
void StartMotor() {
    Dio_WriteChannel(HBRIDGE_CONTROL_PIN_1, STD_HIGH);
```

```
    Dio_WriteChannel(HBRIDGE_CONTROL_PIN_2, STD_HIGH);
    Dio_WriteChannel(HBRIDGE_CONTROL_PIN_3, STD_HIGH);
}

// Hàm dừng động cơ quạt
void StopMotor() {
    Dio_WriteChannel(HBRIDGE_CONTROL_PIN_1, STD_LOW);
    Dio_WriteChannel(HBRIDGE_CONTROL_PIN_2, STD_LOW);
    Dio_WriteChannel(HBRIDGE_CONTROL_PIN_3, STD_LOW);
}
```

Cấu hình thành phần trừu tượng IO

Loại thành phần:

- ECU abstraction software component type.

Cấu hình các Port và Giao diện Port:

- Thành phần trừu tượng cảm biến nhiệt độ:
  - Sử dụng giao diện sender-receiver để truyền dữ liệu nhiệt độ thô lên lớp ứng dụng.
- Thành phần trừu tượng H-Bridge:
  - Sử dụng giao diện client-server để ứng dụng có thể gọi hàm khởi động/dừng quạt.

Cấu hình ARXML:

```
<!-- Cấu hình ARXML cho thành phần trừu tượng cảm biến nhiệt độ -->
<ECUAbstractionComponent>
    <SHORT-NAME>TemperatureSensorAbstraction</SHORT-NAME>
    <PORTS>
        <P-PORT>
            <SHORT-NAME>PP_Temperature</SHORT-NAME>
            <REQUIRED-INTERFACE>
                <SENDER-RECEIVER-INTERFACE>
                    <SHORT-NAME>TemperatureData</SHORT-NAME>
                    <DATA-ELEMENTS>
                        <DATA-ELEMENT>
                            <SHORT-NAME>Temperature</SHORT-NAME>
                            <TYPE>int</TYPE>
                        </DATA-ELEMENT>
                    </DATA-ELEMENTS>
                </SENDER-RECEIVER-INTERFACE>
            </REQUIRED-INTERFACE>
        </P-PORT>
    </PORTS>
</ECUAbstractionComponent>

<!-- Cấu hình ARXML cho thành phần trừu tượng H-Bridge -->
<ECUAbstractionComponent>
    <SHORT-NAME>HBridgeAbstraction</SHORT-NAME>
    <PORTS>
        <P-PORT>
```

```
            <SHORT-NAME>PP_HBridgeControl</SHORT-NAME>
            <REQUIRED-INTERFACE>
                <CLIENT-SERVER-INTERFACE>
                    <SHORT-NAME>HBridgeControlInterface</SHORT-NAME>
                    <OPERATIONS>
                        <OPERATION>
                            <SHORT-NAME>StartMotor</SHORT-NAME>
                            <ARGUMENTS>
                                <ARGUMENT>
                                    <SHORT-NAME>start</SHORT-NAME>
                                    <DIRECTION>in</DIRECTION>
                                    <TYPE>void</TYPE>
                                </ARGUMENT>
                            </ARGUMENTS>
                        </OPERATION>
                        <OPERATION>
                            <SHORT-NAME>StopMotor</SHORT-NAME>
                            <ARGUMENTS>
                                <ARGUMENT>
                                    <SHORT-NAME>stop</SHORT-NAME>
                                    <DIRECTION>in</DIRECTION>
                                    <TYPE>void</TYPE>
                                </ARGUMENT>
                            </ARGUMENTS>
                        </OPERATION>
                    </OPERATIONS>
                </CLIENT-SERVER-INTERFACE>
            </REQUIRED-INTERFACE>
        </P-PORT>
    </PORTS>
</ECUAbstractionComponent>
```

Kết luận

Trong phần này, chúng ta đã cấu hình và triển khai các thành phần trừu tượng cho cảm biến nhiệt độ và H-Bridge. Những thành phần này tương tác với các driver lớp MCAL để thực hiện các chức năng cần thiết. Tiếp theo, chúng ta sẽ tiến hành cấu hình lớp ứng dụng và tích hợp các thành phần này với nhau thông qua RTE để hoàn thành yêu cầu điều khiển quạt làm mát.

29. DEMO-UNDERSTANDING AN ARXML CONFIGURATION

Để bắt đầu, chúng ta sẽ hiểu cấu trúc cấu hình ARXML và các thẻ trong đó. Như đã giải thích, các công cụ Autosar cho cấu hình rất đắt đỏ, và nếu không phải là một phần của các công ty ô tô hàng đầu, chúng ta không thể sử dụng những phần mềm phức tạp này từ bên ngoài. Dù có các công cụ này để cấu hình, việc hiểu cấu hình trực tiếp là rất tốt để có kiến thức đầy đủ về Autosar và các cấu hình của nó.

Cấu trúc ARXML

Cấu hình ARXML mẫu

Chúng ta sẽ sử dụng hai công cụ:

- Notepad++
- XML Spy

Bạn có thể chọn bất kỳ trình soạn thảo XML chung nào, nhưng nên chọn trình hỗ trợ xác thực schema để giúp biết được các thẻ và quy tắc được phép khi cấu hình.

Dưới đây là cấu hình ARXML mẫu để giải thích các khái niệm chung:

```
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns="http://autosar.org/schema/r4.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://autosar.org/schema/r4.0 AUTOSAR_4-3-0.xsd">
    <AR-PACKAGES>
        <AR-PACKAGE>
            <SHORT-NAME>RootPackage</SHORT-NAME>
            <ELEMENTS>
                <!-- Nội dung cấu hình sẽ được đặt ở đây -->
            </ELEMENTS>
        </AR-PACKAGE>
    </AR-PACKAGES>
</AUTOSAR>
```

- **AUTOSAR Tag**: Tất cả các cấu hình liên quan đến Autosar bắt đầu với thẻ `AUTOSAR`.
- **AR-PACKAGES**: Tất cả các cấu hình được đặt trong các gói AR-PACKAGES.
- **AR-PACKAGE**: Mỗi gói phải có một tên ngắn (SHORT-NAME) để tham chiếu.

Tham chiếu giữa các tệp ARXML

Chúng ta có thể phân chia cấu hình thành nhiều tệp ARXML và tham chiếu chéo giữa chúng. Dưới đây là ví dụ về cách thực hiện tham chiếu:

```
<AR-PACKAGE>
    <SHORT-NAME>ExamplePackage</SHORT-NAME>
    <ELEMENTS>
```

```
        <P-PORT-PROTOTYPE>
            <SHORT-NAME>ExamplePort</SHORT-NAME>
            <REQUIRED-INTERFACE-TREF DEST="SENDER-RECEIVER-
INTERFACE">/OtherPackage/ExampleInterface</REQUIRED-INTERFACE-TREF>
        </P-PORT-PROTOTYPE>
    </ELEMENTS>
</AR-PACKAGE>
```

Trong ví dụ này:

- `DEST` chỉ định loại đích là `SENDER-RECEIVER-INTERFACE`.
- Đường dẫn tham chiếu đến `ExampleInterface` trong `OtherPackage`.

Ví dụ cụ thể về Cấu hình ARXML

Cấu hình Driver DIO

Chúng ta sẽ cấu hình các chân IO số (DIO) và ADC của bộ điều khiển.

```
<AR-PACKAGES>
    <AR-PACKAGE>
        <SHORT-NAME>MCAL_Config</SHORT-NAME>
        <ELEMENTS>
            <PORT-DRIVER>
                <SHORT-NAME>PortDriverConfig</SHORT-NAME>
                <PORT-CONFIGURATION>
                    <PORT-CONFIGURATION-SET>
                        <SHORT-NAME>PortConfigSet</SHORT-NAME>
                        <PORTS>
                            <PORT>
                                <SHORT-NAME>Port_10</SHORT-NAME>
                                <PORT-PIN-DIRECTION>PORT_PIN_IN</PORT-PIN-
DIRECTION>
                                <PORT-PIN-MODE>ADC</PORT-PIN-MODE>
                            </PORT>
                            <PORT>
                                <SHORT-NAME>Port_20</SHORT-NAME>
                                <PORT-PIN-DIRECTION>PORT_PIN_OUT</PORT-PIN-
DIRECTION>
                                <PORT-PIN-MODE>DIGITAL_IO</PORT-PIN-MODE>
                            </PORT>
                        </PORTS>
                    </PORT-CONFIGURATION-SET>
                </PORT-CONFIGURATION>
            </PORT-DRIVER>
        </ELEMENTS>
    </AR-PACKAGE>
</AR-PACKAGES>
```

Trong ví dụ này:

- `Port_10` được cấu hình như một chân đầu vào ADC.

- `Port_20` được cấu hình như một chân đầu ra số.

Cấu hình ADC Driver

```
<AR-PACKAGES>
    <AR-PACKAGE>
        <SHORT-NAME>MCAL_Config</SHORT-NAME>
        <ELEMENTS>
            <ADC-DRIVER>
                <SHORT-NAME>AdcDriverConfig</SHORT-NAME>
                <ADC-CHANNELS>
                    <ADC-CHANNEL>
                        <SHORT-NAME>AdcChannel_1</SHORT-NAME>
                        <ADC-GROUP-CHANNELS>
                            <SHORT-NAME>AdcGroup_1</SHORT-NAME>
                            <CHANNELS>
                                <CHANNEL-REF DEST="ADC-
CHANNEL">/MCAL_Config/AdcDriverConfig/AdcChannel_1</CHANNEL-REF>
                            </CHANNELS>
                        </ADC-GROUP-CHANNELS>
                    </ADC-CHANNEL>
                </ADC-CHANNELS>
            </ADC-DRIVER>
        </ELEMENTS>
    </AR-PACKAGE>
</AR-PACKAGES>
```

Kết Luận

Chúng ta đã thấy cách cấu hình các driver cơ bản của MCAL cho các chân IO số và ADC. Trong phần tiếp theo, chúng ta sẽ tiếp tục với cấu hình lớp trừu tượng IO và tích hợp các thành phần này với nhau thông qua RTE để hoàn thành yêu cầu điều khiển quạt làm mát.

30. DEMO - PROJECT SETUP

Chúng ta sẽ tiếp tục với ví dụ của chúng ta bằng cách tạo lớp trừu tượng ECU.

Tạo cấu trúc thư mục

Trước tiên, chúng ta sẽ tạo thư mục để chứa các tệp liên quan.

Bên trong, chúng ta sẽ tạo một tệp C gọi là `ECUAbstraction.c` và một tệp ARXML để viết cấu hình cho thành phần này.

Thiết lập schema

Để chỉnh sửa các tệp ARXML, chúng ta cần có tệp schema để hỗ trợ. Tệp này có thể tải về từ tiêu chuẩn Autosar 4.4.0. Bên trong thư mục tải về, chúng ta có thể thấy tệp zip `methodology and templates`.

Bên trong zip này, chúng ta sẽ tìm thấy tệp XML schema (`AUTOSAR_4-3-0.xsd`) chứa schema cho phiên bản Autosar tương ứng.

Chúng ta sẽ sao chép toàn bộ schema này vào dự án để sử dụng.

Tạo tệp ARXML cho định nghĩa giao diện

Tiếp theo, chúng ta sẽ tạo một thư mục khác để chứa tất cả các định nghĩa giao diện chung cho dự án.

Bên trong, chúng ta sẽ tạo một tệp ARXML gọi là `Interfaces.arxml` để định nghĩa tất cả các giao diện.

Không bắt buộc phải luôn có các giao diện trong một tệp riêng biệt. Tất cả cấu hình có thể được gộp chung trong các gói AR (AR-Packages) hoặc chia ra thành bất kỳ số lượng tệp ARXML nào.

Sử dụng các tệp nền tảng

Tiếp theo, chúng ta sẽ cần một tệp ARXML chứa các loại dữ liệu nền tảng. Chúng ta có thể tái sử dụng tệp này từ tiêu chuẩn. Bên trong thư mục `Modelling Showcase`, trong `Measurements and Calibration`, chúng ta có thể chọn bất kỳ thư mục nào (introductory hoặc advanced), và tìm thấy tệp `PlatformTypes.arxml`.

Chúng ta sẽ sao chép tệp này vào dự án của mình.

Thiết lập cấu hình ARXML

Chúng ta sẽ sử dụng phần mềm XML Spy để chỉnh sửa các tệp ARXML. Như đã đề cập, không bắt buộc phải sử dụng cùng một phần mềm, bất kỳ trình chỉnh sửa XML nào hỗ trợ trợ giúp schema đều có thể sử dụng.

Phần mềm này đi kèm với giấy phép dùng thử một tháng, chúng ta có thể sử dụng cho các thử nghiệm của mình.

Bắt đầu dự án mới trong XML Spy

Chúng ta sẽ tạo một dự án mới trong XML Spy và nhập các thư mục dự án của mình vào đó.

Chúng ta sẽ thêm dự án dưới thư mục XML files.

Cấu hình Platform Types ARXML

Mở tệp `PlatformTypes.arxml` và thiết lập schema cho tệp này. Chúng ta sẽ sử dụng tùy chọn `Assign Schema` để liên kết với tệp schema `AUTOSAR_4-3-0.xsd` đã sao chép từ tiêu chuẩn.

Sau khi thiết lập schema, chúng ta sẽ lưu tệp và xác nhận rằng cấu hình ARXML là hợp lệ.

Các bước tiếp theo

Sử dụng các tùy chọn và tính năng của XML Spy, chúng ta có thể dễ dàng viết cấu hình của mình dựa trên schema.

Tạo cấu hình lớp Trừu Tượng ECU (ECU Abstraction Layer)

Cấu hình tệp `ECUAbstraction.arxml`

Chúng ta sẽ bắt đầu với cấu hình của tệp `ECUAbstraction.arxml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns="http://autosar.org/schema/r4.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://autosar.org/schema/r4.0 AUTOSAR_4-3-0.xsd">
    <AR-PACKAGES>
        <AR-PACKAGE>
            <SHORT-NAME>ECUAbstraction</SHORT-NAME>
            <ELEMENTS>
                <ECU-ABSTRACTION-SW-COMPONENT-TYPE>
                    <SHORT-NAME>TemperatureSensorAbstraction</SHORT-NAME>
                    <PORTS>
                        <P-PORT-PROTOTYPE>
                            <SHORT-NAME>TempSensorPort</SHORT-NAME>
                            <PROVIDED-INTERFACE-TREF DEST="SENDER-RECEIVER-
INTERFACE">/Interfaces/TemperatureSensorInterface</PROVIDED-INTERFACE-TREF>
                        </P-PORT-PROTOTYPE>
                    </PORTS>
```

```
                </ECU-ABSTRACTION-SW-COMPONENT-TYPE>
                <ECU-ABSTRACTION-SW-COMPONENT-TYPE>
                    <SHORT-NAME>HBridgeAbstraction</SHORT-NAME>
                    <PORTS>
                        <P-PORT-PROTOTYPE>
                            <SHORT-NAME>HBridgeStartPort</SHORT-NAME>
                            <PROVIDED-INTERFACE-TREF DEST="CLIENT-SERVER-
INTERFACE">/Interfaces/HBridgeStartInterface</PROVIDED-INTERFACE-TREF>
                        </P-PORT-PROTOTYPE>
                        <P-PORT-PROTOTYPE>
                            <SHORT-NAME>HBridgeStopPort</SHORT-NAME>
                            <PROVIDED-INTERFACE-TREF DEST="CLIENT-SERVER-
INTERFACE">/Interfaces/HBridgeStopInterface</PROVIDED-INTERFACE-TREF>
                        </P-PORT-PROTOTYPE>
                    </PORTS>
                </ECU-ABSTRACTION-SW-COMPONENT-TYPE>
            </ELEMENTS>
        </AR-PACKAGE>
    </AR-PACKAGES>
</AUTOSAR>
```

Tạo các giao diện

Trong tệp `Interfaces.arxml`, chúng ta sẽ định nghĩa các giao diện:

```
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns="http://autosar.org/schema/r4.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://autosar.org/schema/r4.0 AUTOSAR_4-3-0.xsd">
    <AR-PACKAGES>
        <AR-PACKAGE>
            <SHORT-NAME>Interfaces</SHORT-NAME>
            <ELEMENTS>
                <SENDER-RECEIVER-INTERFACE>
                    <SHORT-NAME>TemperatureSensorInterface</SHORT-NAME>
                    <DATA-ELEMENTS>
                        <VARIABLE-DATA-PROTOTYPE>
                            <SHORT-NAME>TemperatureData</SHORT-NAME>
                            <TYPE-TREF DEST="IMPLEMENTATION-DATA-
TYPE">/PlatformTypes/uint8</TYPE-TREF>
                        </VARIABLE-DATA-PROTOTYPE>
                    </DATA-ELEMENTS>
                </SENDER-RECEIVER-INTERFACE>
                <CLIENT-SERVER-INTERFACE>
                    <SHORT-NAME>HBridgeStartInterface</SHORT-NAME>
                    <OPERATIONS>
                        <CLIENT-SERVER-OPERATION>
                            <SHORT-NAME>StartMotor</SHORT-NAME>
                            <ARGUMENTS>
                                <ARGUMENTS>
                                    <SHORT-NAME>void</SHORT-NAME>
                                    <TYPE-TREF DEST="IMPLEMENTATION-DATA-
TYPE">/PlatformTypes/void</TYPE-TREF>
                                </ARGUMENTS>
                            </ARGUMENTS>
```

```
                        </CLIENT-SERVER-OPERATION>
                    </OPERATIONS>
                </CLIENT-SERVER-INTERFACE>
                <CLIENT-SERVER-INTERFACE>
                    <SHORT-NAME>HBridgeStopInterface</SHORT-NAME>
                    <OPERATIONS>
                        <CLIENT-SERVER-OPERATION>
                            <SHORT-NAME>StopMotor</SHORT-NAME>
                            <ARGUMENTS>
                                <ARGUMENTS>
                                    <SHORT-NAME>void</SHORT-NAME>
                                    <TYPE-TREF DEST="IMPLEMENTATION-DATA-
TYPE">/PlatformTypes/void</TYPE-TREF>
                                </ARGUMENTS>
                            </ARGUMENTS>
                        </CLIENT-SERVER-OPERATION>
                    </OPERATIONS>
                </CLIENT-SERVER-INTERFACE>
            </ELEMENTS>
        </AR-PACKAGE>
    </AR-PACKAGES>
</AUTOSAR>
```

Kết Luận

Chúng ta đã thiết lập cấu hình cho lớp Trừu Tượng ECU và định nghĩa các giao diện cần thiết. Trong phần tiếp theo, chúng ta sẽ tiếp tục với việc cấu hình lớp Ứng dụng và hoàn thành yêu cầu điều khiển quạt làm mát.

31. DEMO - CREATING INTERFACES IN ARXML

1. Định nghĩa các giao diện (Interfaces)

Đầu tiên, chúng ta sẽ định nghĩa các giao diện (port interfaces) trong `Interfaces.arxml`.

Chúng ta bắt đầu với một tệp mới và thêm các phần tử cần thiết.

```
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns="http://autosar.org/schema/r4.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://autosar.org/schema/r4.0 AUTOSAR_4-3-0.xsd">
    <AR-PACKAGES>
        <AR-PACKAGE>
            <SHORT-NAME>Interfaces</SHORT-NAME>
            <ELEMENTS>
                <!-- Sender Receiver Interface for Temperature -->
                <SENDER-RECEIVER-INTERFACE>
                    <SHORT-NAME>I_Temperature</SHORT-NAME>
                    <DATA-ELEMENTS>
                        <VARIABLE-DATA-PROTOTYPE>
                            <SHORT-NAME>Temperature</SHORT-NAME>
                            <TYPE-TREF DEST="IMPLEMENTATION-DATA-
TYPE">/PlatformTypes/ImplementationDataTypes/float32</TYPE-TREF>
                        </VARIABLE-DATA-PROTOTYPE>
                    </DATA-ELEMENTS>
                </SENDER-RECEIVER-INTERFACE>
                <!-- Client Server Interface for Start Motor -->
                <CLIENT-SERVER-INTERFACE>
                    <SHORT-NAME>I_StartMotor</SHORT-NAME>
                    <OPERATIONS>
                        <CLIENT-SERVER-OPERATION>
                            <SHORT-NAME>StartMotor</SHORT-NAME>
                            <ARGUMENTS>
                                <ARGUMENT-DATA-PROTOTYPE>
                                    <SHORT-NAME>Fan_Speed</SHORT-NAME>
                                    <TYPE-TREF DEST="IMPLEMENTATION-DATA-
TYPE">/PlatformTypes/ImplementationDataTypes/uint8</TYPE-TREF>
                                    <DIRECTION>IN</DIRECTION>
                                </ARGUMENT-DATA-PROTOTYPE>
                            </ARGUMENTS>
                            <POSSIBLE-ERRORS>
                                <APPLICATION-ERROR>
                                    <SHORT-NAME>E_OK</SHORT-NAME>
                                    <ERROR-CODE>0</ERROR-CODE>
                                </APPLICATION-ERROR>
                                <APPLICATION-ERROR>
                                    <SHORT-NAME>E_NOTOK</SHORT-NAME>
                                    <ERROR-CODE>1</ERROR-CODE>
                                </APPLICATION-ERROR>
                            </POSSIBLE-ERRORS>
                        </CLIENT-SERVER-OPERATION>
                    </OPERATIONS>
                </CLIENT-SERVER-INTERFACE>
                <!-- Client Server Interface for Stop Motor -->
```

```
                <CLIENT-SERVER-INTERFACE>
                    <SHORT-NAME>I_StopMotor</SHORT-NAME>
                    <OPERATIONS>
                        <CLIENT-SERVER-OPERATION>
                            <SHORT-NAME>StopMotor</SHORT-NAME>
                            <POSSIBLE-ERRORS>
                                <APPLICATION-ERROR>
                                    <SHORT-NAME>E_OK</SHORT-NAME>
                                    <ERROR-CODE>0</ERROR-CODE>
                                </APPLICATION-ERROR>
                                <APPLICATION-ERROR>
                                    <SHORT-NAME>E_NOTOK</SHORT-NAME>
                                    <ERROR-CODE>1</ERROR-CODE>
                                </APPLICATION-ERROR>
                            </POSSIBLE-ERRORS>
                        </CLIENT-SERVER-OPERATION>
                    </OPERATIONS>
                </CLIENT-SERVER-INTERFACE>
            </ELEMENTS>
        </AR-PACKAGE>
    </AR-PACKAGES>
</AUTOSAR>
```

2. Tạo lớp trừu tượng ECU

2.1 Cấu hình ARXML cho lớp trừu tượng ECU (ECUAbstraction.arxml)

Chúng ta sẽ bắt đầu với cấu hình ARXML cho lớp trừu tượng ECU:

```
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns="http://autosar.org/schema/r4.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://autosar.org/schema/r4.0 AUTOSAR_4-3-0.xsd">
    <AR-PACKAGES>
        <AR-PACKAGE>
            <SHORT-NAME>ECUAbstraction</SHORT-NAME>
            <ELEMENTS>
                <ECU-ABSTRACTION-SW-COMPONENT-TYPE>
                    <SHORT-NAME>TemperatureSensorAbstraction</SHORT-NAME>
                    <PORTS>
                        <P-PORT-PROTOTYPE>
                            <SHORT-NAME>TempSensorPort</SHORT-NAME>
                            <PROVIDED-INTERFACE-TREF DEST="SENDER-RECEIVER-
INTERFACE">/Interfaces/I_Temperature</PROVIDED-INTERFACE-TREF>
                        </P-PORT-PROTOTYPE>
                    </PORTS>
                </ECU-ABSTRACTION-SW-COMPONENT-TYPE>
                <ECU-ABSTRACTION-SW-COMPONENT-TYPE>
                    <SHORT-NAME>HBridgeAbstraction</SHORT-NAME>
                    <PORTS>
                        <P-PORT-PROTOTYPE>
                            <SHORT-NAME>HBridgeStartPort</SHORT-NAME>
                            <PROVIDED-INTERFACE-TREF DEST="CLIENT-SERVER-
INTERFACE">/Interfaces/I_StartMotor</PROVIDED-INTERFACE-TREF>
                        </P-PORT-PROTOTYPE>
```

```
                        <P-PORT-PROTOTYPE>
                            <SHORT-NAME>HBridgeStopPort</SHORT-NAME>
                            <PROVIDED-INTERFACE-TREF DEST="CLIENT-SERVER-
INTERFACE">/Interfaces/I_StopMotor</PROVIDED-INTERFACE-TREF>
                        </P-PORT-PROTOTYPE>
                    </PORTS>
                </ECU-ABSTRACTION-SW-COMPONENT-TYPE>
            </ELEMENTS>
        </AR-PACKAGE>
    </AR-PACKAGES>
</AUTOSAR>
```

3. Mã C cho lớp trừu tượng ECU

3.1 Tệp `ECUAbstraction.c`

Tạo tệp `ECUAbstraction.c` để chứa mã C cho lớp trừu tượng ECU.

```
#include "Rte_ECUAbstraction.h"

void ReadTemperature(void) {
    Std_ReturnType result;
    uint16 adcResult;

    result = Adc_StartGroupConversion(ADC_GROUP_TEMPERATURE);
    if (result == E_OK) {
        result = Adc_ReadGroup(ADC_GROUP_TEMPERATURE, &adcResult);
        if (result == E_OK) {
            float temperature = (float)adcResult * 0.1; // Example conversion
            Rte_Write_TempSensorPort_Temperature(temperature);
        }
    }
}

void StartMotor(uint8 Fan_Speed) {
    if (Fan_Speed > 0) {
        Dio_WriteChannel(DIO_CHANNEL_HBRIDGE, STD_HIGH);
        // Additional logic to set fan speed
    } else {
        Dio_WriteChannel(DIO_CHANNEL_HBRIDGE, STD_LOW);
    }
}

void StopMotor(void) {
    Dio_WriteChannel(DIO_CHANNEL_HBRIDGE, STD_LOW);
}
```

Kết luận

Chúng ta đã hoàn thành việc định nghĩa các giao diện (port interfaces) và cấu hình lớp trừu tượng ECU. Chúng ta cũng đã tạo mã C cho lớp trừu tượng ECU. Trong phần tiếp theo, chúng ta sẽ cấu hình lớp ứng dụng và hoàn thành yêu cầu điều khiển quạt làm mát.

32. DEMO - ECU ABSTRACTION DEVELOPMENT (ARXML)

1. Tổng quan về ARXML của thành phần phần mềm

Bây giờ chúng ta sẽ viết cấu hình ARXML cho lớp trừu tượng ECU (ECU Abstraction Layer). Chúng ta sẽ định nghĩa các cổng (ports), hành vi nội bộ (internal behavior), và các sự kiện (events) cho thành phần phần mềm của chúng ta.

Chúng ta sẽ bắt đầu với `ECUAbstraction.arxml`.

```
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns="http://autosar.org/schema/r4.4.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://autosar.org/schema/r4.4.0 AUTOSAR_4-4-0.xsd">
    <AR-PACKAGES>
        <AR-PACKAGE>
            <SHORT-NAME>SWComponent</SHORT-NAME>
            <ELEMENTS>
                <ECU-ABSTRACTION-SW-COMPONENT-TYPE>
                    <SHORT-NAME>TemperatureHBridgeSoftwareComponent</SHORT-
NAME>
                    <PORTS>
                        <!-- Provider Port for Temperature -->
                        <P-PORT-PROTOTYPE>
                            <SHORT-NAME>PP_Temperature</SHORT-NAME>
                            <PROVIDED-INTERFACE-TREF DEST="SENDER-RECEIVER-
INTERFACE">/Interfaces/I_Temperature</PROVIDED-INTERFACE-TREF>
                        </P-PORT-PROTOTYPE>
                        <!-- Provider Port for Start Motor -->
                        <P-PORT-PROTOTYPE>
                            <SHORT-NAME>PP_StartMotor</SHORT-NAME>
                            <PROVIDED-INTERFACE-TREF DEST="CLIENT-SERVER-
INTERFACE">/Interfaces/I_StartMotor</PROVIDED-INTERFACE-TREF>
                        </P-PORT-PROTOTYPE>
                        <!-- Provider Port for Stop Motor -->
                        <P-PORT-PROTOTYPE>
                            <SHORT-NAME>PP_StopMotor</SHORT-NAME>
                            <PROVIDED-INTERFACE-TREF DEST="CLIENT-SERVER-
INTERFACE">/Interfaces/I_StopMotor</PROVIDED-INTERFACE-TREF>
                        </P-PORT-PROTOTYPE>
                    </PORTS>
                    <INTERNAL-BEHAVIORS>
                        <INTERNAL-BEHAVIOR>
                            <SHORT-NAME>IB_SWC</SHORT-NAME>
                            <RUNNABLES>
                                <!-- Runnable for Reading Temperature -->
                                <RUNNABLE-ENTITY>
                                    <SHORT-NAME>ReadTemperature</SHORT-NAME>
                                    <SYMBOL>ReadTemperature_function</SYMBOL>
                                    <MINIMUM-START-INTERVAL>0.0</MINIMUM-
START-INTERVAL>
                                    <CAN-BE-INVOKED-CONCURRENTLY>false</CAN-
BE-INVOKED-CONCURRENTLY>
                                    <DATA-SEND-POINTS>
```

```
                                        <VARIABLE-ACCESS>
                                            <TARGET-DATA-PROTOTYPE-TREF
DEST="VARIABLE-DATA-PROTOTYPE">/Interfaces/I_Temperature/Temperature</TARGET-
DATA-PROTOTYPE-TREF>
                                            <PORT-PROTOTYPE-REF DEST="P-PORT-
PROTOTYPE">/SWComponent/TemperatureHBridgeSoftwareComponent/PP_Temperature</P
ORT-PROTOTYPE-REF>
                                        </VARIABLE-ACCESS>
                                    </DATA-SEND-POINTS>
                                </RUNNABLE-ENTITY>
                                <!-- Runnable for Starting Motor -->
                                <RUNNABLE-ENTITY>
                                    <SHORT-NAME>StartMotor</SHORT-NAME>
                                    <SYMBOL>StartMotor_function</SYMBOL>
                                    <MINIMUM-START-INTERVAL>0.0</MINIMUM-
START-INTERVAL>
                                    <CAN-BE-INVOKED-CONCURRENTLY>false</CAN-
BE-INVOKED-CONCURRENTLY>
                                </RUNNABLE-ENTITY>
                                <!-- Runnable for Stopping Motor -->
                                <RUNNABLE-ENTITY>
                                    <SHORT-NAME>StopMotor</SHORT-NAME>
                                    <SYMBOL>StopMotor_function</SYMBOL>
                                    <MINIMUM-START-INTERVAL>0.0</MINIMUM-
START-INTERVAL>
                                    <CAN-BE-INVOKED-CONCURRENTLY>false</CAN-
BE-INVOKED-CONCURRENTLY>
                                </RUNNABLE-ENTITY>
                            </RUNNABLES>
                            <EVENTS>
                                <!-- Timing Event for Reading Temperature
every 10ms -->
                                <TIMING-EVENT>
                                    <SHORT-NAME>TE_10ms</SHORT-NAME>
                                    <START-ON-EVENT>
                                        <RUNNABLE-ENTITY-REF DEST="RUNNABLE-
ENTITY">/SWComponent/TemperatureHBridgeSoftwareComponent/InternalBehavior/Rea
dTemperature</RUNNABLE-ENTITY-REF>
                                    </START-ON-EVENT>
                                    <PERIOD>0.01</PERIOD>
                                </TIMING-EVENT>
                                <!-- Operation Invoked Event for Starting
Motor -->
                                <OPERATION-INVOKED-EVENT>
                                    <SHORT-NAME>OIE_StartMotor</SHORT-NAME>
                                    <START-ON-EVENT>
                                        <RUNNABLE-ENTITY-REF DEST="RUNNABLE-
ENTITY">/SWComponent/TemperatureHBridgeSoftwareComponent/InternalBehavior/Sta
rtMotor</RUNNABLE-ENTITY-REF>
                                    </START-ON-EVENT>
                                    <OPERATION-REF DEST="CLIENT-SERVER-
OPERATION">/Interfaces/I_StartMotor/StartMotor</OPERATION-REF>
                                    <CONTEXT-P-PORT-REF DEST="P-PORT-
PROTOTYPE">/SWComponent/TemperatureHBridgeSoftwareComponent/PP_StartMotor</CO
NTEXT-P-PORT-REF>
                                </OPERATION-INVOKED-EVENT>
```

```
                                <!-- Operation Invoked Event for Stopping
Motor -->
                                <OPERATION-INVOKED-EVENT>
                                    <SHORT-NAME>OIE_StopMotor</SHORT-NAME>
                                    <START-ON-EVENT>
                                        <RUNNABLE-ENTITY-REF DEST="RUNNABLE-
ENTITY">/SWComponent/TemperatureHBridgeSoftwareComponent/InternalBehavior/Sto
pMotor</RUNNABLE-ENTITY-REF>
                                    </START-ON-EVENT>
                                    <OPERATION-REF DEST="CLIENT-SERVER-
OPERATION">/Interfaces/I_StopMotor/StopMotor</OPERATION-REF>
                                    <CONTEXT-P-PORT-REF DEST="P-PORT-
PROTOTYPE">/SWComponent/TemperatureHBridgeSoftwareComponent/PP_StopMotor</CON
TEXT-P-PORT-REF>
                                </OPERATION-INVOKED-EVENT>
                            </EVENTS>
                        </INTERNAL-BEHAVIOR>
                    </INTERNAL-BEHAVIORS>
                </ECU-ABSTRACTION-SW-COMPONENT-TYPE>
            </ELEMENTS>
        </AR-PACKAGE>
    </AR-PACKAGES>
</AUTOSAR>
```

2. Mã C cho lớp trừu tượng ECU

2.1 Tệp `ECUAbstraction.c`

Tạo tệp `ECUAbstraction.c` để chứa mã C cho lớp trừu tượng ECU.

```
#include "Rte_TemperatureHBridgeSoftwareComponent.h"

void ReadTemperature_function(void) {
    Std_ReturnType result;
    uint16 adcResult;

    result = Adc_StartGroupConversion(ADC_GROUP_TEMPERATURE);
    if (result == E_OK) {
        result = Adc_ReadGroup(ADC_GROUP_TEMPERATURE, &adcResult);
        if (result == E_OK) {
            float temperature = (float)adcResult * 0.1; // Example conversion
            Rte_Write_PP_Temperature_Temperature(temperature);
        }
    }
}

void StartMotor_function(uint8 Fan_Speed) {
    if (Fan_Speed > 0) {
        Dio_WriteChannel(DIO_CHANNEL_HBRIDGE, STD_HIGH);
        // Additional logic to set fan speed
    } else {
        Dio_WriteChannel(DIO_CHANNEL_HBRIDGE, STD_LOW);
    }
}
```

```
void StopMotor_function(void) {
    Dio_WriteChannel(DIO_CHANNEL_HBRIDGE, STD_LOW);
}
```

Tóm tắt

Chúng ta đã cấu hình lớp trừu tượng ECU (ECU Abstraction Layer) với các cổng (ports), hành vi nội bộ (internal behavior), và các sự kiện (events). Chúng ta cũng đã tạo mã C cho lớp trừu tượng ECU. Trong phần tiếp theo, chúng ta sẽ cấu hình lớp ứng dụng và hoàn thành yêu cầu điều khiển quạt làm mát.

33. DEMO - ECU ABSTRACTION DEVELOPMENT (C FILE)

1. Thiết lập tệp `ECUAbstraction.c`

Tạo tệp `ECUAbstraction.c` để chứa mã C cho lớp trừu tượng ECU. Tất cả các thành phần phần mềm đều bắt đầu với tiêu đề ứng dụng RTE để bao gồm các API RTE được tạo cho các cấu hình chúng ta đã thực hiện trong ARXML.

2. Cài đặt mã nguồn cho các runnable

Mã nguồn của chúng ta sẽ được viết theo các runnable mà chúng ta đã định nghĩa trong ARXML. Chúng ta sẽ cần tham chiếu đến cấu hình ARXML để đảm bảo rằng mã nguồn của chúng ta tuân theo các định danh đã định nghĩa.

2.1 Tiêu đề RTE

Bắt đầu bằng cách bao gồm tiêu đề RTE với định dạng `RTE_` + Tên thành phần + `.h`.

```
#include "Rte_TemperatureHBridgeSoftwareComponent.h"
```

2.2 Hàm ReadTemperature

Định nghĩa hàm `ReadTemperature` để kích hoạt chuyển đổi ADC, đọc kết quả từ lớp MCAL và truyền giá trị chuyển đổi lên lớp ứng dụng thông qua RTE.

```
void ReadTemperature_function(void) {
    Std_ReturnType result;
    uint16 adcResult;

    result = Adc_StartGroupConversion(1);
    if (result == E_OK) {
        result = Adc_ReadGroup(1, &adcResult);
        if (result == E_OK) {
            float pinVoltage = (2.5 / 1024) * adcResult;
            float temperature = pinVoltage * 100; // Assuming sensor's output
is 0.35V at 35 degrees Celsius

            Rte_Write_PP_Temperature_Temperature(temperature);
        }
    }
}
```

2.3 Hàm StartMotor

Định nghĩa hàm `StartMotor` để bật quạt làm mát dựa trên tham số tốc độ quạt.

```
void StartMotor_function(uint8 Fan_Speed) {
    if (Fan_Speed < 50) {
        // Set IO pins for low speed
        Dio_WriteChannel(10, STD_HIGH);
```

```
        Dio_WriteChannel(11, STD_LOW);
    } else {
        // Set IO pins for high speed
        Dio_WriteChannel(10, STD_HIGH);
        Dio_WriteChannel(11, STD_HIGH);
    }
}
```

2.4 Hàm StopMotor

Định nghĩa hàm `StopMotor` để dừng quạt làm mát.

```
void StopMotor_function(void) {
    Dio_WriteChannel(10, STD_LOW);
    Dio_WriteChannel(11, STD_LOW);
}
```

Tóm tắt

Chúng ta đã hoàn thành cấu hình ARXML và phát triển mã nguồn C cho lớp trừu tượng ECU (ECU Abstraction Layer). Các bước chính bao gồm định nghĩa các runnable, ánh xạ các sự kiện, và viết mã nguồn cho các hàm tương ứng trong tệp `ECUAbstraction.c`.

Với cấu hình và mã nguồn này, chúng ta đã hoàn thành phần mềm trừu tượng ECU, cho phép giao tiếp giữa các lớp phần mềm và phần cứng thông qua RTE. Tiếp theo, chúng ta có thể tiến hành cấu hình và phát triển lớp ứng dụng để hoàn thiện yêu cầu điều khiển quạt làm mát.

34. DEMO - ASW DEVELOPMENT (ARXML)

1. Thiết lập tệp `ApplicationSWC.c`

Tạo tệp `ApplicationSWC.c` để chứa mã C cho lớp ứng dụng. Bắt đầu bằng cách bao gồm tiêu đề RTE cho thành phần phần mềm của chúng ta.

```
#include "Rte_TemperatureControlSWC.h"
```

2. Cài đặt mã nguồn cho các runnable

Mã nguồn của chúng ta sẽ được viết theo các runnable mà chúng ta đã định nghĩa trong ARXML. Chúng ta sẽ cần tham chiếu đến cấu hình ARXML để đảm bảo rằng mã nguồn của chúng ta tuân theo các định danh đã định nghĩa.

2.1 Hàm `ApplicationTemperatureControl_function`

Định nghĩa hàm `ApplicationTemperatureControl_function` để đọc nhiệt độ từ cổng nhận và quyết định bật hoặc tắt động cơ dựa trên nhiệt độ đọc được.

```
void ApplicationTemperatureControl_function(void) {
    Std_ReturnType result;
    float temperature;

    // Đọc nhiệt độ từ cổng nhận
    result = Rte_Read_RP_Temperature_Temperature(&temperature);
    if (result == E_OK) {
        // Quyết định bật hoặc tắt động cơ dựa trên nhiệt độ
        if (temperature > 50.0) {
            // Bật động cơ
            result = Rte_Call_RP_StartMotor_StartMotor(100); // 100 là tốc độ
động cơ giả định
            if (result != E_OK) {
                // Xử lý lỗi nếu có
            }
        } else {
            // Tắt động cơ
            result = Rte_Call_RP_StopMotor_StopMotor();
            if (result != E_OK) {
                // Xử lý lỗi nếu có
            }
        }
    } else {
        // Xử lý lỗi nếu có
    }
}
```

Tóm tắt

Chúng ta đã hoàn thành cấu hình ARXML và phát triển mã nguồn C cho lớp ứng dụng (Application Layer). Các bước chính bao gồm định nghĩa các runnable, ánh xạ các sự kiện, và viết mã nguồn cho các hàm tương ứng trong tệp `ApplicationSWC.c`.

Với cấu hình và mã nguồn này, chúng ta đã hoàn thành lớp ứng dụng, cho phép giao tiếp giữa các thành phần phần mềm và phần cứng thông qua RTE. Tiếp theo, chúng ta có thể tiếp tục cấu hình và phát triển các lớp khác nếu cần để hoàn thiện yêu cầu điều khiển quạt làm mát.

35. DEMO - ASW DEVELOPMENT (C FILE)

Bây giờ chúng ta đã hoàn thành cấu hình ARXML, chúng ta sẽ bắt đầu viết mã C cho lớp ứng dụng bằng Notepad++.

1. Thiết lập tệp `ApplicationSWC.c`

Tạo và mở tệp `ApplicationSWC.c` để chứa mã C cho lớp ứng dụng.

```
#include "Rte_TemperatureControlSWC.h"
```

2. Cài đặt mã nguồn cho các runnable

Chúng ta sẽ viết mã nguồn cho runnable mà chúng ta đã định nghĩa trong ARXML. Chúng ta sẽ cần tham chiếu đến cấu hình ARXML để đảm bảo rằng mã nguồn của chúng ta tuân theo các định danh đã định nghĩa.

2.1 Hàm `ApplicationTemperatureControl_function`

Định nghĩa hàm `ApplicationTemperatureControl_function` để đọc nhiệt độ từ cổng nhận và quyết định bật hoặc tắt động cơ dựa trên nhiệt độ đọc được.

```
void ApplicationTemperatureControl_function(void) {
    Std_ReturnType result;
    float temperature;

    // Đọc nhiệt độ từ cổng nhận
    result = Rte_Read_RP_Temperature_Temperature(&temperature);
    if (result == E_OK) {
        // Quyết định bật hoặc tắt động cơ dựa trên nhiệt độ
        if (temperature > 50.0 && temperature < 75.0) {
            // Bật động cơ ở tốc độ thấp
            result = Rte_Call_RP_StartMotor_StartMotor(25); // 25% tốc độ
quạt
            if (result != E_OK) {
                // Xử lý lỗi nếu có
            }
        } else if (temperature >= 75.0) {
            // Bật động cơ ở tốc độ cao
            result = Rte_Call_RP_StartMotor_StartMotor(100); // 100% tốc độ
quạt
            if (result != E_OK) {
                // Xử lý lỗi nếu có
            }
        } else {
            // Tắt động cơ nếu nhiệt độ dưới 50 độ
            result = Rte_Call_RP_StopMotor_StopMotor();
            if (result != E_OK) {
                // Xử lý lỗi nếu có
            }
        }
    } else {
```

```
        // Xử lý lỗi nếu có
    }
}
```

Tóm tắt

- Đã bao gồm tiêu đề RTE cho thành phần ứng dụng của chúng ta.
- Định nghĩa hàm `ApplicationTemperatureControl_function`.
- Đọc nhiệt độ từ cổng nhận.
- Quyết định bật hoặc tắt động cơ dựa trên nhiệt độ đọc được.

Với cấu hình và mã nguồn này, chúng ta đã hoàn thành lớp ứng dụng, cho phép giao tiếp giữa các thành phần phần mềm và phần cứng thông qua RTE. Tiếp theo, chúng ta có thể tiếp tục kiểm tra và xác minh mã của mình để đảm bảo rằng nó hoạt động đúng cách.

36. DEMO - COMPOSITIONS AND RTE

1. Cấu hình thành phần phần mềm (Software Component)

Chúng ta đã hoàn tất phát triển thành phần phần mềm ECU abstraction và thành phần phần mềm ứng dụng. Các thành phần này đã được cấu hình với các cổng và runnables thích hợp.

2. Tạo file thành phần liên kết (Composition)

Bây giờ, chúng ta sẽ tạo file `Composition.arxml` để cấu hình thành phần liên kết của chúng ta, kết nối các thành phần phần mềm đã phát triển.

Bước 1: Tạo file `Composition.arxml` và cấu hình ban đầu

- Tạo file `Composition.arxml` và sao chép các tag header tĩnh từ các file ARXML đã phát triển trước đó.
- Bắt đầu với cấu trúc gói AR-Package.

```
<AUTOSAR xmlns="http://autosar.org/schema/r4.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>Composition</SHORT-NAME>
      <ELEMENTS>
        <COMPOSITION-SW-COMPONENT-TYPE>
          <SHORT-NAME>TemperatureControlComposition</SHORT-NAME>
          <COMPONENTS>
            <!-- Thành phần phần mềm sẽ được liệt kê ở đây -->
          </COMPONENTS>
          <CONNECTORS>
            <!-- Kết nối sẽ được liệt kê ở đây -->
          </CONNECTORS>
        </COMPOSITION-SW-COMPONENT-TYPE>
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AUTOSAR>
```

Bước 2: Liệt kê các thành phần phần mềm đã phát triển

Chúng ta sẽ thêm các thành phần phần mềm đã phát triển (ECU abstraction và ứng dụng) vào cấu trúc thành phần.

```
<COMPONENTS>
  <SW-COMPONENT-PROTOTYPE>
    <SHORT-NAME>TemperatureHBridgeSWC</SHORT-NAME>
    <TYPE-TREF DEST="ECU-ABSTRACTION-SW-COMPONENT-
TYPE">/SWComponent/TemperatureHBridgeSWC</TYPE-TREF>
  </SW-COMPONENT-PROTOTYPE>
  <SW-COMPONENT-PROTOTYPE>
    <SHORT-NAME>ApplicationSWC</SHORT-NAME>
```

```
    <TYPE-TREF DEST="APPLICATION-SW-COMPONENT-
TYPE">/SWComponent/TemperatureControlSWC</TYPE-TREF>
  </SW-COMPONENT-PROTOTYPE>
</COMPONENTS>
```

Bước 3: Tạo kết nối giữa các cổng

Chúng ta sẽ tạo các kết nối giữa các cổng của các thành phần phần mềm.

```
<CONNECTORS>
  <!-- Kết nối cho Temperature Interface -->
  <ASSEMBLY-SW-CONNECTOR>
    <SHORT-NAME>AssemblyConnector_Temperature</SHORT-NAME>
    <PROVIDER-IREFT>
      <CONTEXT-COMPONENT-REF>/Composition/TemperatureHBridgeSWC</CONTEXT-
COMPONENT-REF>
      <TARGET-P-PORT-REF>/TemperatureHBridgeSWC/PP_Temperature</TARGET-P-
PORT-REF>
    </PROVIDER-IREFT>
    <REQUESTER-IREFT>
      <CONTEXT-COMPONENT-REF>/Composition/ApplicationSWC</CONTEXT-COMPONENT-
REF>
      <TARGET-R-PORT-REF>/ApplicationSWC/RP_Temperature</TARGET-R-PORT-REF>
    </REQUESTER-IREFT>
  </ASSEMBLY-SW-CONNECTOR>

  <!-- Kết nối cho Start Motor Interface -->
  <ASSEMBLY-SW-CONNECTOR>
    <SHORT-NAME>AssemblyConnector_StartMotor</SHORT-NAME>
    <PROVIDER-IREFT>
      <CONTEXT-COMPONENT-REF>/Composition/TemperatureHBridgeSWC</CONTEXT-
COMPONENT-REF>
      <TARGET-P-PORT-REF>/TemperatureHBridgeSWC/PP_StartMotor</TARGET-P-PORT-
REF>
    </PROVIDER-IREFT>
    <REQUESTER-IREFT>
      <CONTEXT-COMPONENT-REF>/Composition/ApplicationSWC</CONTEXT-COMPONENT-
REF>
      <TARGET-R-PORT-REF>/ApplicationSWC/RP_StartMotor</TARGET-R-PORT-REF>
    </REQUESTER-IREFT>
  </ASSEMBLY-SW-CONNECTOR>

  <!-- Kết nối cho Stop Motor Interface -->
  <ASSEMBLY-SW-CONNECTOR>
    <SHORT-NAME>AssemblyConnector_StopMotor</SHORT-NAME>
    <PROVIDER-IREFT>
      <CONTEXT-COMPONENT-REF>/Composition/TemperatureHBridgeSWC</CONTEXT-
COMPONENT-REF>
      <TARGET-P-PORT-REF>/TemperatureHBridgeSWC/PP_StopMotor</TARGET-P-PORT-
REF>
    </PROVIDER-IREFT>
    <REQUESTER-IREFT>
      <CONTEXT-COMPONENT-REF>/Composition/ApplicationSWC</CONTEXT-COMPONENT-
REF>
      <TARGET-R-PORT-REF>/ApplicationSWC/RP_StopMotor</TARGET-R-PORT-REF>
    </REQUESTER-IREFT>
```

```
  </ASSEMBLY-SW-CONNECTOR>
</CONNECTORS>
```

Tổng kết

- Chúng ta đã hoàn tất phát triển các thành phần phần mềm cho lớp ECU abstraction và ứng dụng.
- Chúng ta đã tạo và cấu hình thành phần liên kết (Composition) để kết nối các cổng giữa các thành phần phần mềm.
- Chúng ta đã sử dụng các API RTE để thực hiện giao tiếp giữa các thành phần phần mềm.

Việc này hoàn thành quá trình phát triển phần mềm theo tiêu chuẩn Autosar, giúp dễ dàng quản lý, bảo trì và nâng cấp phần mềm trong các hệ thống phức tạp như ô tô.