**ASSIGNMENT 1**

## Design requirements

Hãy thiết kế 1 chức năng trong xe (ví dụ hệ thống đèn, sưởi ghế, chỉnh ghế điện, gạt mưa…):

* Break down xuống thành các SWC
* Các internal behavior:
  + Runnable và mô tả chức năng của từng Runnable
  + Events để trigger các Runnable, mô tả.
* Design các port để thực hiện communication, allocate vào runnable cụ thể
* Allocate vào các ECU (nhiều hơn 1) để thể hiện inter-communication và intra-communication trong RTE

## Design Concept

* System Design: In-vehicle air conditioning system.
* Function:
* Temperature Monitoring: Continuously reads engine temperature from sensors.
* Engine Speed Monitoring: Continuously reads engine speed to control the activation of the air conditioning system.
* Compressor Activation: Sends signals to activate the compressor.
* Cooling Fan Activation: Sends signals to turn on the cooling fan when needed.
* Fault Diagnosis: Detects sensor and actuator faults and logs error codes.

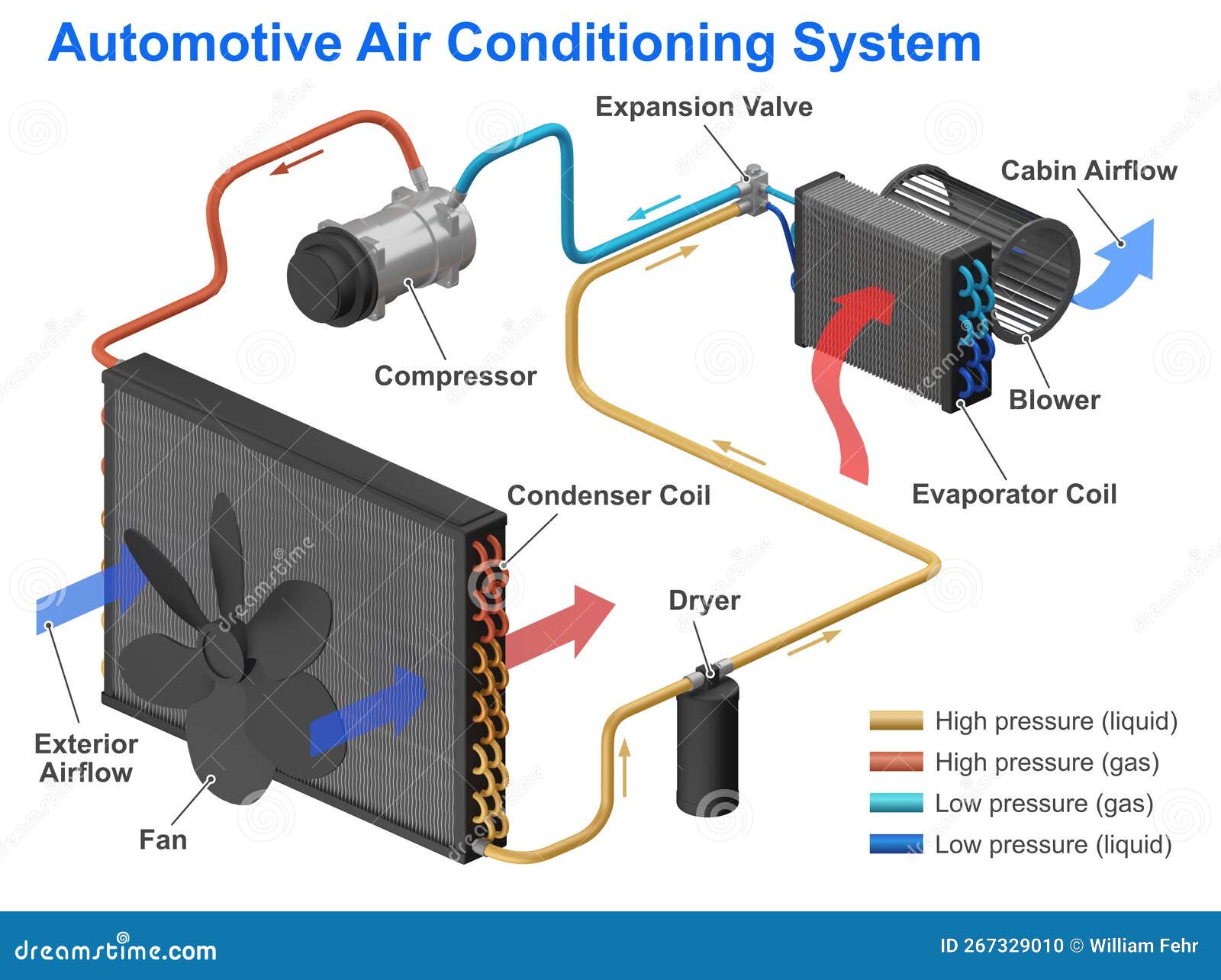
****

Figure 1. Air Conditioning System

**1.3 SWCs diagram**

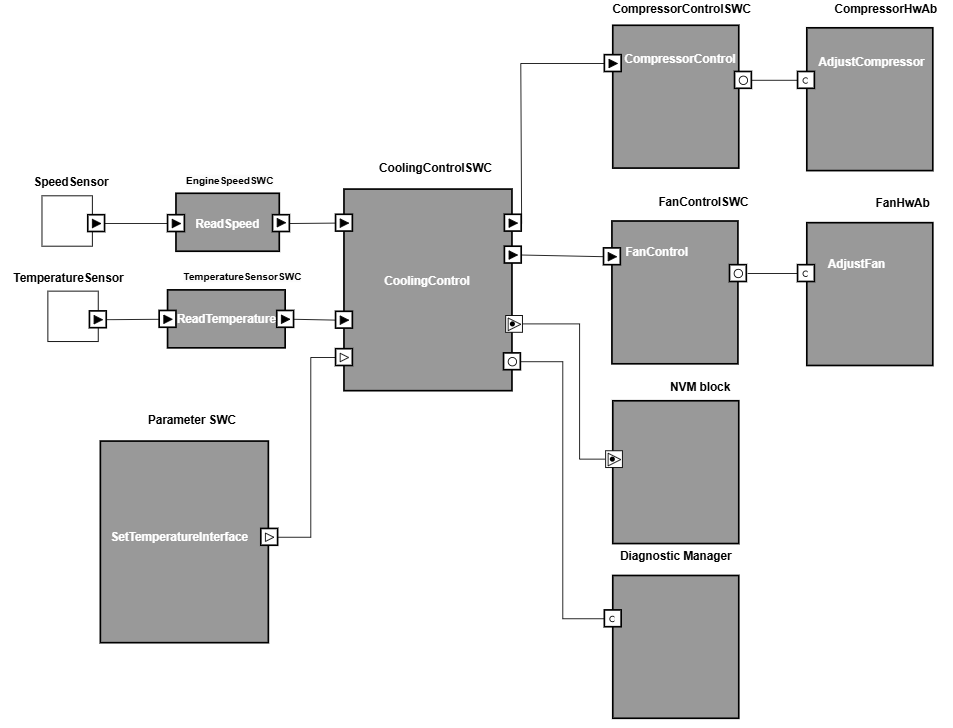
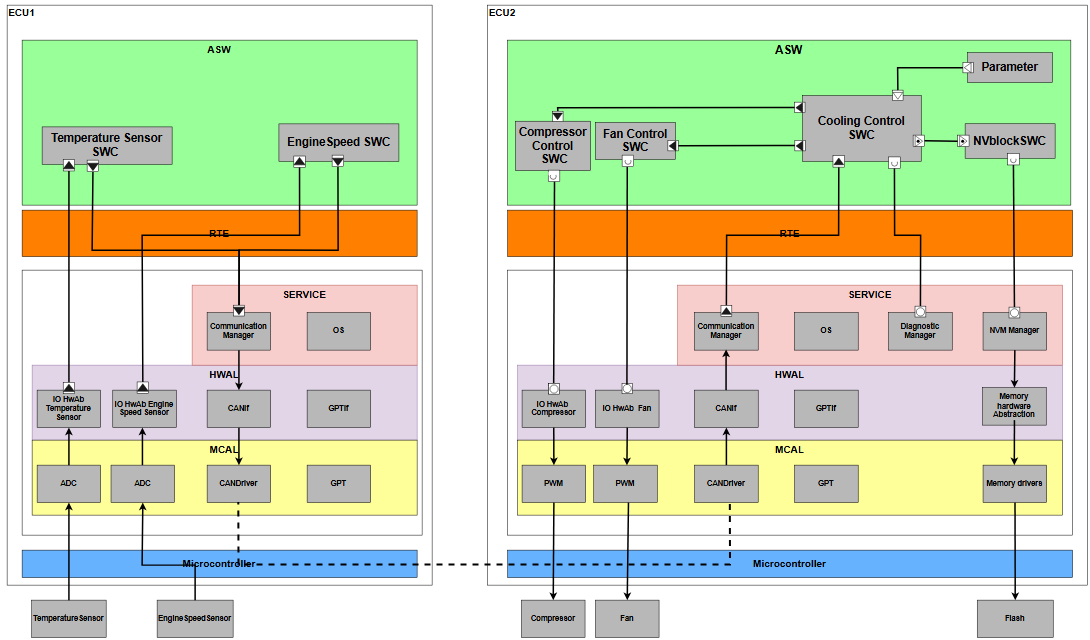
****

Figure 2. SWCs diagram

|  |  |
| --- | --- |
| **Component in SWCs diagram** | **Function** |
| Speed Sensor | Air Conditioning Control Based on Engine Speed   * If the engine is off → The air conditioning system turns off. * If the engine is running → The air conditioning system turns on. |
| Temperature Sensor | Measures the air temperature inside the vehicle and provides data for processing. |
| Engine Speed SWC | Read and process data from the engine speed sensor. |
| Temperature Sensor SWC | Read and process data from temperature sensors. |
| CoolingControlSWC | Manage the entire temperature control system. Decide to turn on/off the air conditioning system. |
| CompressorControlSWC | Control the air conditioner's refrigeration compressor. |
| CompressorHwAb | Execute refrigeration compressor control commands. |
| FanControlSWC | Control the air conditioner fan. |
| FanHwAb | Execute fan speed control commands. |
| NVM block | Manage data storage and retrieval in lossless memory. |
| Diagnostic Manager | Manage and process diagnostic requests from Tester or components in the ECU. |
| Parameter SWC | Simulate data from the user or from the external environment. |

Table 1. Components in SWC diagram

****Figure 3. Inter-communication diagram

**1.4 Internal Behavior**

**a. Runnables & Events:**

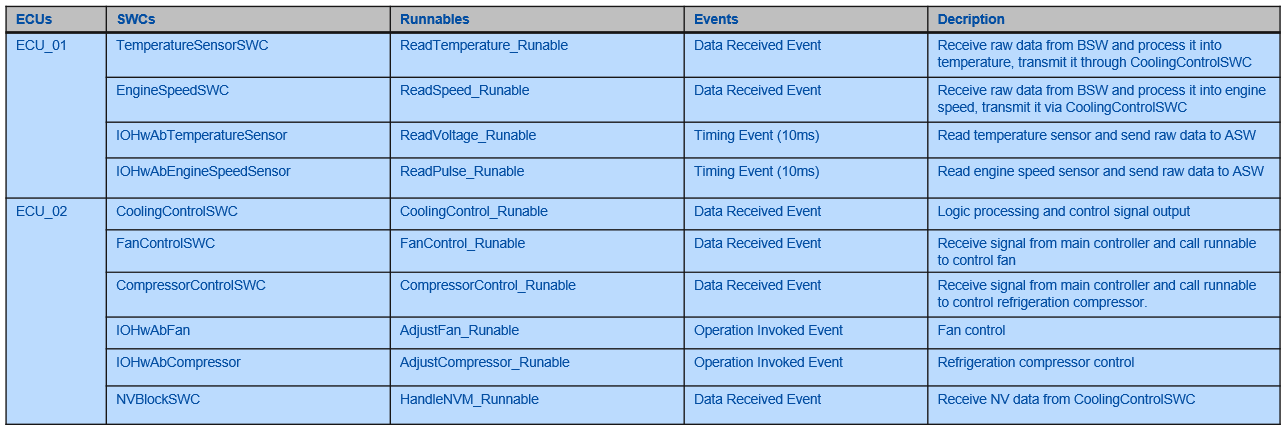
List of Runnables and Events: 

Table 1. Runables and event

**1.5 Design and description of Ports**

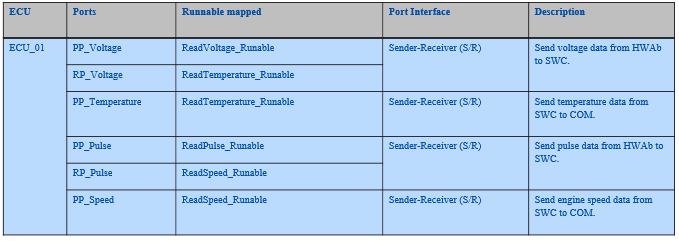
List of used Ports in ECU\_01:  


Table 2. Ports in ECU\_01

List of used Ports in ECU\_02:

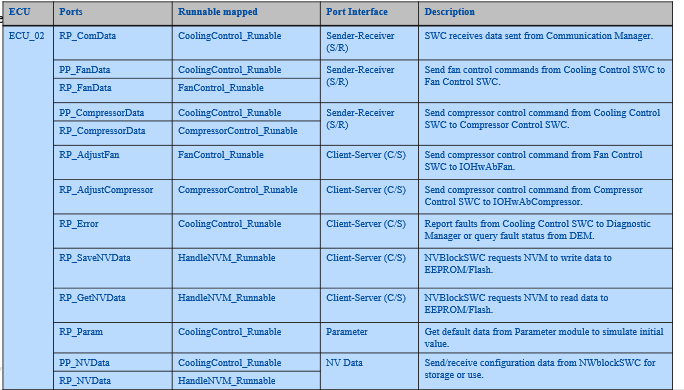


Table 3. Ports in ECU\_02

**ASSIGNMENT 2: OS**

**1. Design requirements**

Create task and define attribute (task priority, task type…)

Mapping Runnable with defined tasks.

Mapping Rte event and OS Event.

**2. Implementation**

**2.1 Create task, define attribute and mapping Runnable with defined tasks**

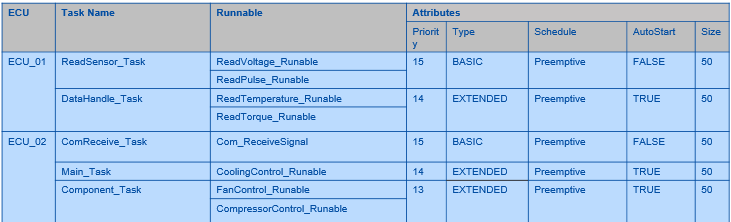


Table 4. Ports in ECU\_02

**2.2 Mapping Runnables to RTE & OS Events**

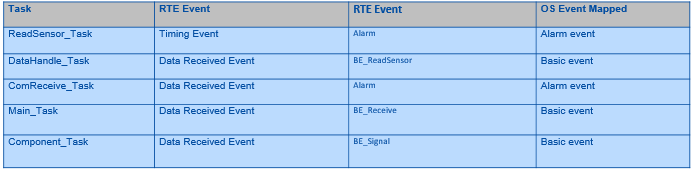
****

Table 5. Mapping Runnables to RTE & OS Events

**ASSIGNMENT 3: COMMUNICATION (CAN)**

**1. Design requirements**

From the sample DBC file, define the messages used in the system.

Describe the layout of the message, signal attributes: bit position, length, byte order.

Describe the characteristics of the message: cyclic, number of repetitions…

**2. Implementation**

The **Automotive Air Conditioning System** uses two types of signals transmitted over the **CAN (Controller Area Network)**, including:

* **Sensor signal** from the **Temperature Sensor(ECU1)** sent to **Cooling Control SWC (ECU2).**
* **Sensor signal** from the **Engine Speed Sensor(ECU1)** sent to **Cooling Control SWC (ECU2).**

With this requirement, we design a network with **one message** transmitted from **ECU1 to ECU2**, containing **two signals**.

## CAN Network

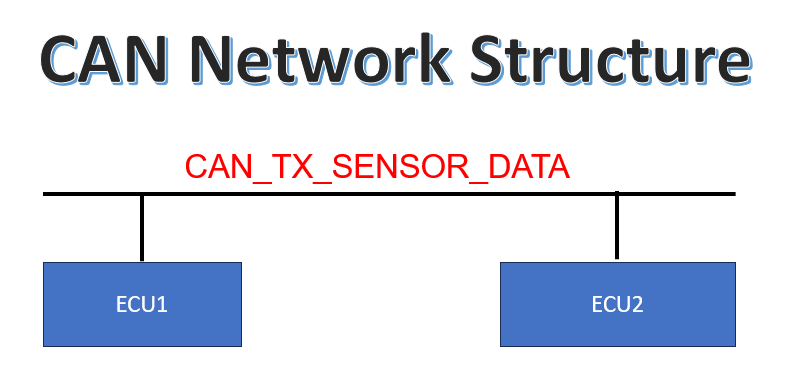
A **CAN (Controller Area Network) Network** is a communication system that connects multiple Electronic Control Units (**ECUs**) within a vehicle or industrial system. It enables **real-time, reliable, and efficient** data exchange between various components without the need for a central computer.

Figure 3.1 - CAN Network Structure

- Message Definition

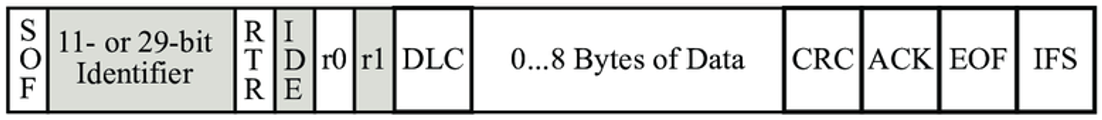
A **Message** in a **CAN network** represents a structured **data frame** transmitted over the CAN bus. Each **CAN frame** contains a **unique identifier (CAN ID)**, a **data length code (DLC)** specifying the number of data bytes, and multiple **signals** that represent different types of information. A message can be of different **frame types**, such as **Data Frame, Remote Frame, Error Frame, or Overload Frame**, depending on its function within the network.

Figure 3.2 - Message in CAN

Message Properties:

|  |  |
| --- | --- |
| **Property** | **Description** |
| Name | Its how the message is displayed in Tools in human readable format |
| Type | Says if its standard or extended frame. Relates to **IDE** bit |
| ID | The CAN Identifier. |
| Transmitter | Which ECU or NODE is transmitter of this message |
| Tx Method | How the message is transmitted . Based on attribute **GenMsgSendType** |
| DLC | Data length code. Says how many bytes of data is in data field of the frame |
| Cycle Time | If the Tx method is cyclic that what is the periodicity of frame. **GenMsgCycleTime** |
| Signals/Layout | What signals are present and what are their positions in the data field of the CAN frame. For this additional property of **startbit** is mentioned in message. |

Table 3.1 - Message Properties

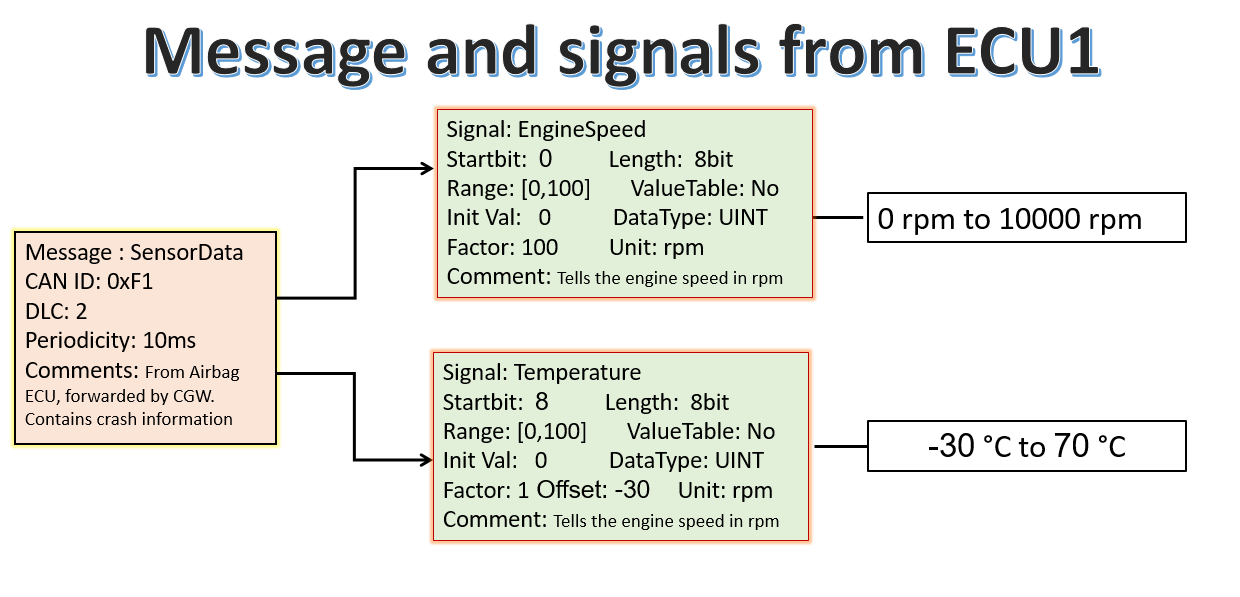


Figure 3.3 - Message and signals

Message Configuration:

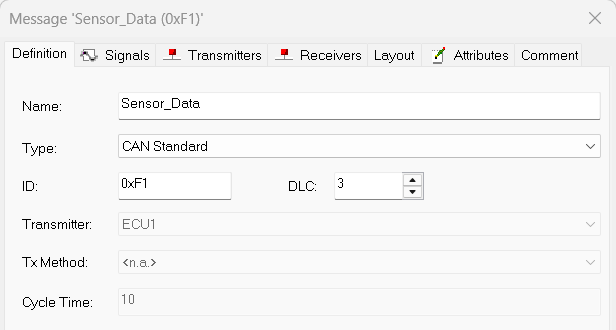


Figure 3.4 - Message Configuration

## Layout message, signal attributes

A **Signal** represents a specific piece of data within a **Message**. Each **Signal** occupies a defined number of bits in the **Data Frame** and follows a specific format.

|  |  |
| --- | --- |
| **Property** | **Description** |
| Name | Signal identified by this in all tools |
| Length | The size of the signal. How many bits in CAN data field will this signal occupy |
| Byte Order | Tells about endianness [More details in next slides] |
| Unit | Make sense of physical value of signal |
| Value Type | datatype related detail for the signal |
| Init Value | The value of signal in can frame when not even a single valid value has been received by application. |
| Factor and Offset | Used to compute internal value from physical value |
| Min/Max | Provides the range of the signal value |
| StartBit | Tells the position of lsb of signal in data field in CAN frame |

Table 3.2 - Signal Description

EngineSpeed Signal Configuration:

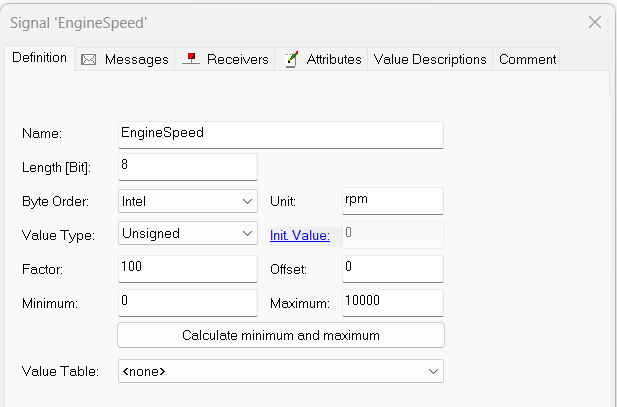


Figure 3.5 - EngineSpeed Signal Configuration

Temperature Signal Configuration:

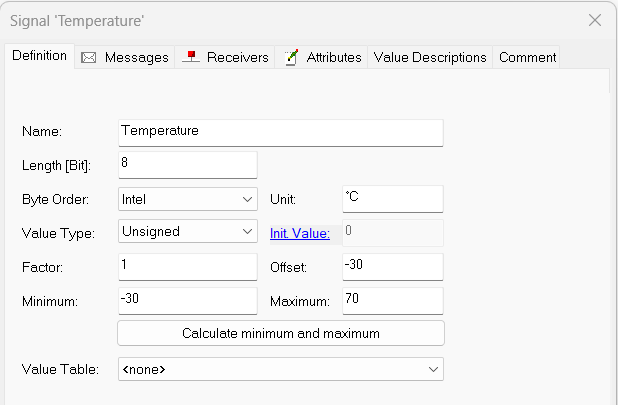


Figure 3.6 - Temperature Signal Configuration

Message Layout:

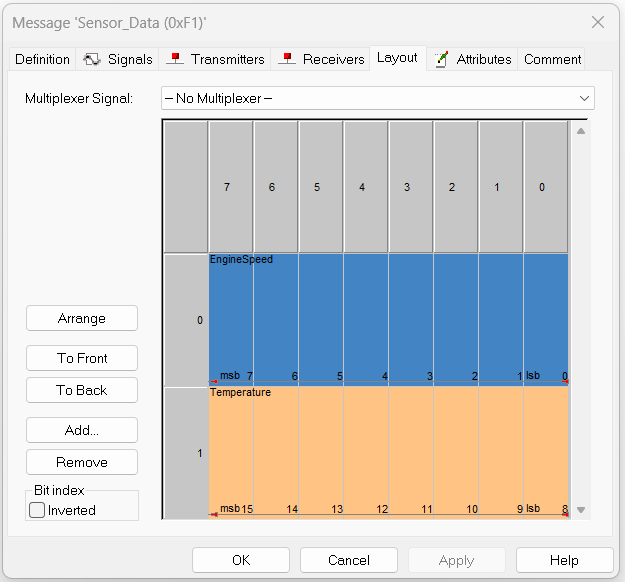


Figure 3.7 - Message Layout

## Message Attributes

A **message attribute** is used to define **additional properties** of a CAN message, such as its **cycle time, send type, or delay**. These attributes help configure how messages behave in the **CAN network**.

|  |  |  |
| --- | --- | --- |
| Attribute Name | Description | Example value |
| GenMsgCycleTime | Defines the periodic transmission time of the message in milliseconds (**e.g., 10ms for cyclic transmission**). | GenMsgCycleTime = 10; |
| GenMsgSendType | Defines the message transmission type (**Cyclic, OnChange, or NoMsgSendType**). | GenMsgSendType = "Cyclic"; |
| GenMsgDelayTime | Defines the minimum delay before re-transmitting the message (in ms). | GenMsgDelayTime = 5; |

Figure 3.8 - Message Attributes

**GenMsgCycleTime Configuration:**

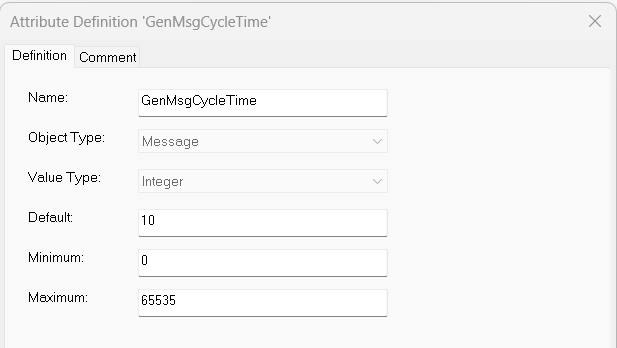


Figure 3.9 - GenMsgCycleTime Configuration

GenMsgSendType Configuration:

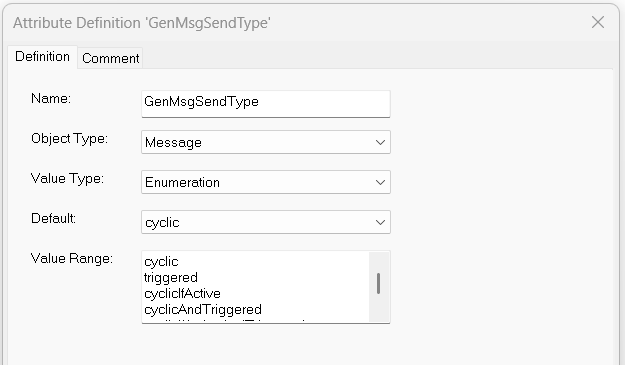


Figure 3.10 - GenMsgSendType Configuration

# ASSIGNMENT 4: DIAGNOSTIC

## 4.1 Requirement:

* Define DTC có khả năng xảy ra với chức năng đó
* Các step để report lỗi cho DEM và step để đọc lỗi từ Tester

**4.2 Design implementation:**

**4.2.1 Definition:**

* DTC (Diagnostic Trouble Code) is a code used in diagnostic systems to identify issues or faults in system functions. In AUTOSAR, DTCs are defined for each function or module and help monitor problems when they occur. These issues can be hardware failures, software errors, or data-related problems.

**4.2.2 DTC structure:**

**4.2.2.1 DTC Structure Overview:**

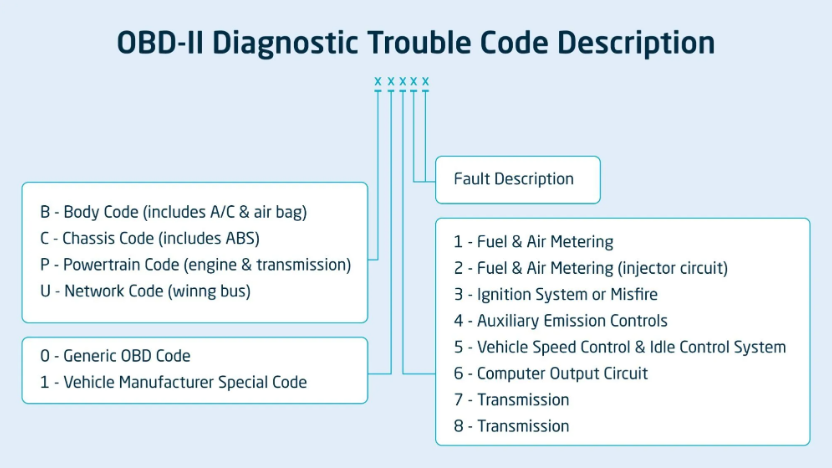
* ****Each DTC consists of 5 characters, including letters and digits. The structure of a DTC can be described as the figure below:

Figure 4.1 DTC Structure

|  |  |  |
| --- | --- | --- |
| **Character** | **Value** | **Explanation** |
| First Character (Letter) | P, C, B, U | Represents the type of system or component with the issue |
| Second Character (Digit 1) | 0 or 1 | Represents the type of code (standard or enhanced) |
| Third Character (Digit 2) | 0 to 9 | Defines the subsystem or area of the issue |
| Fourth and Fifth Characters (Digit 4 & 5) | 0 to 9 | Provide further details about the specific issue in the system or component |

Figure 4.2 Characters in the DTC code

**4.2.3 DTC structure:**

|  |  |  |
| --- | --- | --- |
| **DTC No.** | **Detection conditions or fail** | **Scope of influence** |
| **B1 470**  Temperature Sensor Malfunction | - The interior temperature sensor fails to provide proper data to the ECU.  - The cabin temperature is not accurately regulated. | - Electrical connections (wiring, connectors).  - ECU\_01  - Interior temperature sensor. |
| **P0 500**  Speed Sensor Malfunction | - The vehicle speed sensor fails to send correct speed data to the ECU.  - The air conditioning system may fail to activate properly, as it relies on speed data to adjust the system. | - Vehicle Speed Sensor  - ECU\_01  - Electrical connections (wiring, connectors). |
| **U1 100**  ECU\_01 Lost Communication | - ECU\_01 lost connection to the Can bus | - Electrical connections (wiring, connectors).  - ECU\_01 |
| **U1 100**  ECU\_02 Lost Communication | - ECU\_02 lost connection to the Can bus | - Electrical connections (wiring, connectors).  - ECU\_02 |

Figure 4.3 Characters in the DTC code

**4.2.4 Error reporting and error reading process:**

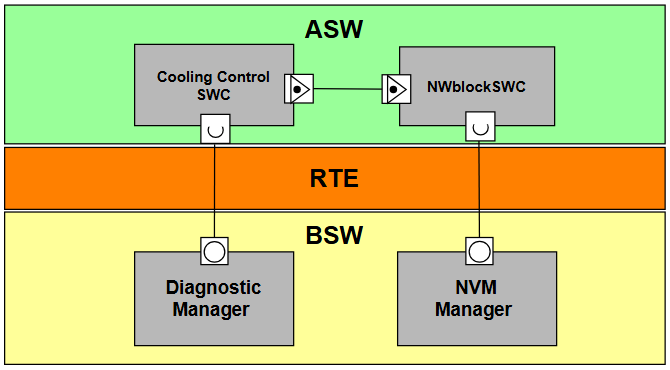
****

Figure 4.3 The communication between DEM module with SWC and BSW

**Steps to report bugs to DEM:**

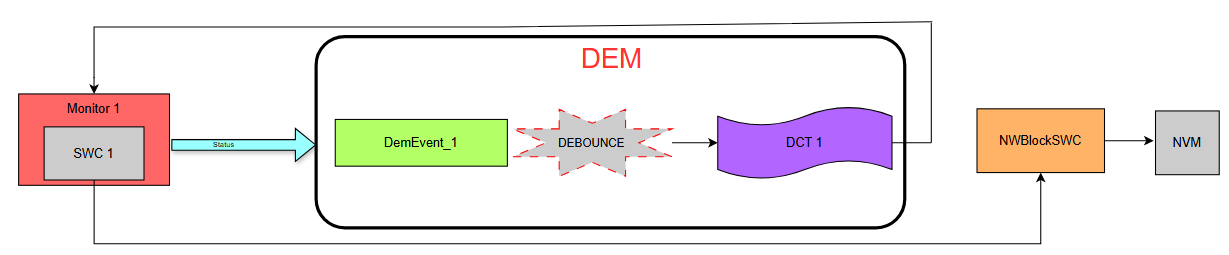
****

Table 4.4 process of reporting bugs to DEM

* Component Monitoring is a mechanism for monitoring software components (SWC, BSW) in an AUTOSAR system to ensure they function correctly.
* **Step 1: Monitoring Function supervises the SWC**

Each SWC has a Monitoring Function that checks the system’s operation.

In our project, the SWC monitors the temperature in the car, sensor signals, and engine speed to detect faults.

* **Step 2: SWC detects a fault and reports it to DEM**

If the Monitoring Function detects an anomaly, the SWC will call the DEM API to report the fault to DEM. The SWC calls Rte\_Call\_ReportEventStatus(EVENT\_ID, Status) to update the fault status.

The possible statuses include:

|  |  |
| --- | --- |
| **Status sent to DEM** | **Description** |
| **Pre-Pass** (DEM\_EVENT\_STATUS\_PREPASSED) | The data may be valid, but it is not certain. |
| **Pre-Failed** (DEM\_EVENT\_STATUS\_PREFAILED) | |  | | --- | | An error may have occurred, further confirmation is required. |  |  | | --- | |  | |
| **Failed** (DEM\_EVENT\_STATUS\_FAILED) | |  | | --- | | Confirm the error occurred. |  |  | | --- | |  | |
| **Passed** (DEM\_EVENT\_STATUS\_PASSED) | The error has been fixed, the system operates normally. |

Table 4.5 The Status of DEM Events

* **Step 3: DEM Uses Debounce Algorithm to Confirm the Fault**

DEM uses a debounce algorithm to track the number of times a fault occurs. If the fault persists for a predefined threshold (Fail Counter), it is then confirmed as a real fault (FAILED). DEM validates the fault and updates the DTC (Diagnostic Trouble Code).

* **Step 4: SWC Requests Fault Status from DEM**

The SWC can request DEM to check the current status of a fault.

* **Step 5: Store the Fault in NVM if DEM Confirms It**

If DEM confirms the fault, the SWC does not write directly to NVM. Instead, the SWC must store the fault in NVM via NWBlockSWC using the function Rte\_PP\_NVData().

* **Step 6: SWC Reports That the Fault Has Been Resolved by Sending "Passed" to DEM**

If the SWC detects that the system has returned to normal, it does not automatically clear the fault status but must report it back to DEM.

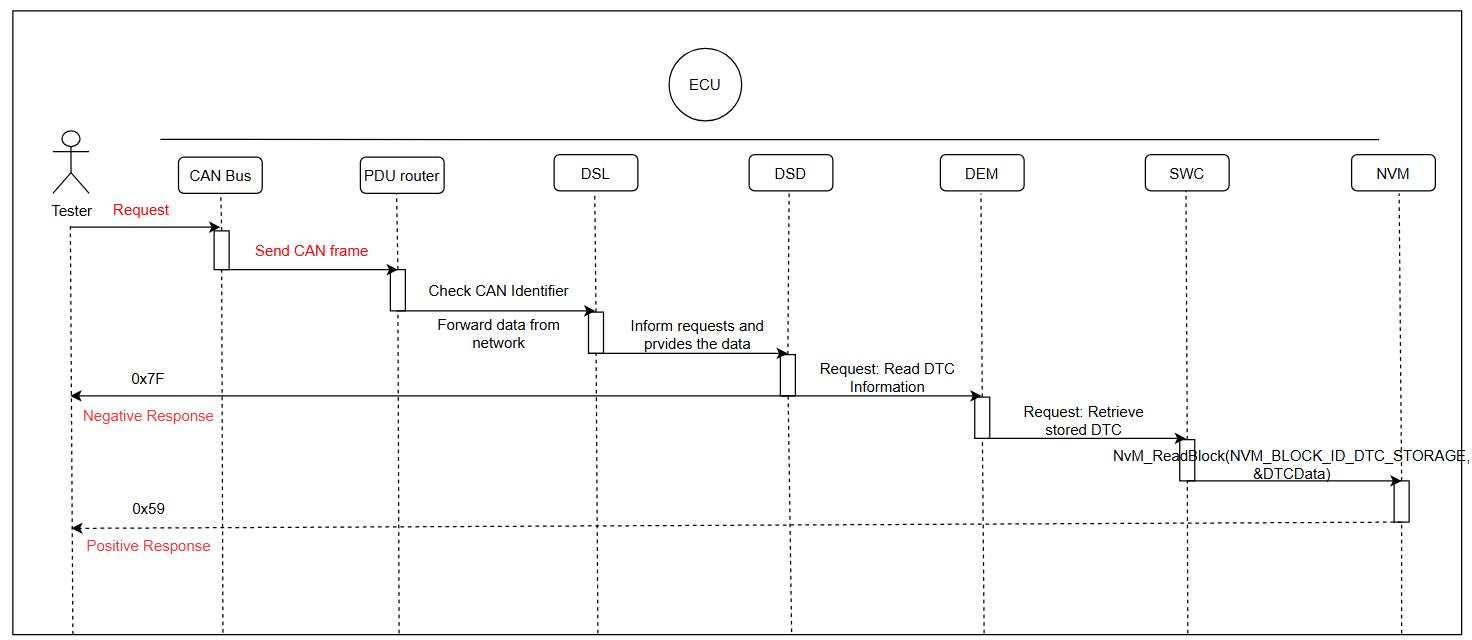
The SWC calls Rte\_Call\_ReportEventStatus(EVENT\_ID, DEM\_EVENT\_STATUS\_PASSED) to clear the fault status.

* **Step 7: DEM Activates the Aging Counter to Ensure the Fault Has Disappeared**

When a fault transitions from FAILED → PASSED, DEM does not immediately erase the DTC. Instead, it uses an Aging Counter. If the fault does not reappear for a certain number of driving cycles, DEM will erase the DTC from memory.

**4.2.5 Steps for Tester to Read Faults (DTCs)**

Table 4.6 Steps for Tester to Read Faults from NVM

****

* **Step 1: Tester Sends a Request to the ECU via CAN Bus**

The Tester sends a UDS (ISO 14229) request with Service ID 0x19 to read fault information (DTCs) from the ECU. Below is the specific Service ID 0x19 and its sub-functions:

|  |  |  |
| --- | --- | --- |
| **Service ID** | **Sub function** | **Description** |
| 0x19 | 0x01 | Report the Number of DTC by Status Mask |
| 0x02 | Report DTC by Status Mask |
| 0x03 | Report DTC Snapshot ID |
| 0x04 | Report DTC Snapshot Record by DTC |
| 0x05 | Report Stored Data by Record Number |
| 0x06 | Report DTC Extended Data Record by DTC Number |
| 0x07 | Report Number of DTC by Severity Mask Record |
| 0x08 | Report DTC by Severity Mask Record |
| 0x09 | Report Severity Information of DTC |
| 0x0A | Report Supported DTCs |
| 0x0B | Report First Test Failed DTC |
| 0x0C | Report First Confirmed DTC |
| 0x0D | Report Most Recent Test Failed DTC |
| 0x0F | Report Most Recent Confirmed DTC |

Table 4.7 Service ID 0x19

* **Step 2:** **CAN Bus Receives the UDS Request and Forwards It to the ECU**

The CAN Bus receives the UDS request from the Tester and forwards it to the appropriate ECU based on the CAN identifier.

* **Step 3:** **PDU Router Routes Data to the Diagnostic Layer**

The PDU Router (Protocol Data Unit Router) is responsible for routing data between different communication layers within the AUTOSAR communication system. In our project, the PDU Router directs diagnostic messages from the CAN Transport Protocol (CAN TP) to the Diagnostic Communication Manager (DCM).

* **Step 4: DSL Processes the Diagnostic Request**

When the ECU receives a diagnostic request from the Tester, the DSL (Diagnostic Session Layer) performs the following tasks:

* + Manages response time (timeout).
  + Controls the diagnostic session state (Session Control).
  + Forwards the request to DSD for processing.
* **Step 5:** **DSD Identifies the Service and Calls DEM to Read DTCs**

After successful validation, DSL forwards the request to DSD (Diagnostic Service Dispatcher), which is responsible for processing specific diagnostic services.

DSD determines that this is a DTC read request and sends the request to DEM (Diagnostic Event Manager).

* **Step 6: DEM Requests SWC to Read Data from NVM**

Since DEM does not directly access NVM, it sends a request to an intermediate SWC. When the SWC receives the request from DEM, it calls the NvM\_ReadBlock() API to read the DTC data stored in NVM (Non-Volatile Memory).

* **Step 7:** **SWC Sends DTC Data Back to DEM**

After retrieving the data from NVM, the SWC transfers the DTC information to DEM via the RTE (Runtime Environment).

DEM then processes the fault information and sends the final result to DSD.

* **Step 8:** **ECU Sends the Response Back to the Tester via CAN Bus**

DSD compiles the DTC data and sends the response to the Tester via CAN Bus.

If the requested DTC exists, the ECU sends a positive response (0x59) containing the DTC code and status back to the Tester.

If no DTCs are found, or if there is an issue, the ECU sends a negative response (0x7F)

# ASSIGNMENT 5: NvM, Wdg

## Design Requirement

* Define 1 hoặc nhiều supervised entity để thực hiện chức năng timing protection.
* Define NvM block (size, các attribute khác)

## Design implementation

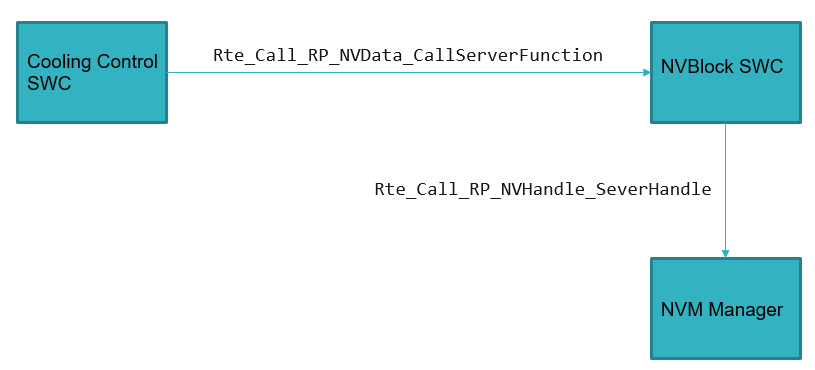
### 5.2.1 Watchdog Timer

* Watchdog Manager (WdgM) belongs to the Service Layer in AUTOSAR and is responsible for monitoring and responding when the software system encounters errors.
* Supervised Entity (SE) is a runnable or a task within the system.
* To implement Timing Protection, WdgM uses Supervised Entity (SE) to track the execution progress of each task and ensure that they are operating within the specified time limits.
* These Supervised Entities will use checkpoints; when they reach a checkpoint during the execution of tasks, WdgM will receive these updates and responses from the corresponding supervised entity.
* To monitor SE, WdgM applies various Supervision Mechanisms to detect and handle execution time errors:
  + **Alive Supervision**: Monitors if a task executes at the expected frequency.
  + **Deadline Supervision**: Ensures task completion within the maximum allowed time.
  + **Logical Supervision**: Verifies the correct execution sequence of checkpoints.
  + **Window Supervision**: Ensures checkpoints occur within a specified time window.

|  |  |  |
| --- | --- | --- |
| **Supervision Mechanism** | **Supervision Entity’s Name** | **Description** |
| Alive Supervision | SEID\_SENSOR\_ERROR | Supervisor in case of failing in reading sensor signal ->Recover (stop control actuator, delay 1s for next reading signal) |

**5.2.2 Define supervised entity:**

**5.3. Design NVM block**

****

In this project sever supply 4 service including write and read both parameters and error code. But I just use one service that is write error code to array for simulation with 2 error code:

- Temperature sensor fail if it out of range 0-100 C

- Speed sensor fail if it out of range 0-10000 rpm