# Simulation of communication in industrial networks – CAN network

Student: Andreea-Ioana Avram

---

Structure of Computer Systems Project

---

Technical University of Cluj-Napoca

January 8, 2024

# Contents

# Introduction

## 1.1 Context

The aim of this project is to simulate communication in a **C**ontroller **A**rea **N**etwork (CAN) by incorporating key features such as message arbitration, error detection, and priority-based transmission. CAN networks are widely used in industries where real-time reliable transmission is essential, and simulating these communication processes helps to provide a better understanding of how such networks manage data transmission.

This simulation will serve as a platform for analyzing CAN communication performance, providing insight into different strategies and how they can affect the network efficiency and reliability. By creating an interactive simulation, this project provides users with an educational tool for exploring how a CAN network behaves under various conditions, such as node failures, traffic loads, and error-prone environments.

## 1.2 Objectives

The aim of this project is to create a simulation of a Controller Area Network, focusing on elements such as message arbitration, error detection, and priority-based message transmission. The simulation is going to provide users with understanding about the way CAN communication happens under various conditions, making it an interactive educational tool. The user's capabilities are described in List 1.2.1.

**List 1.2.1: Users capabilities in the CAN Simulation**

- Manage network nodes: users can add or remove nodes to customize the network.
- Custom Messages: users can define message with desired properties.
- Inject Errors and Test Network Robustness: users can introduce errors and observe how the system detects and handles them.
- Run Predefine Scenarios: users will have access to predefined scenarios that simulate common CAN communication conditions.

By simulating various real-world scenarios, such as node failures, high traffic, and error-prone environments, offering visualization of the processes that can occur during CAN communication.

# Bibliographic Research

## 2.1 CAN communication fundamentals

The Controller Area Network (CAN) protocol operates as a highly reliable, multi-master, broadcast communication system that enables data exchange between microcontroller units (MCUs) without a host device. Each connection point in a CAN network is known as a "**node**", critical components that independently manage message transmission, reception, and error handling. This node structure supports a decentralized approach, where multiple nodes can attempt to send messages simultaneously and resolve conflicts through an efficient priority-based arbitration process.

Each node in a CAN network is configured with a **unique identifier** that plays a key role in the arbitration process. This identifier, often referred to as the **message ID**, determines the priority of the messages that each node attempts to send. Identifiers are set according to the message priority and the specific application requirements. IDs are typically determined during system design based on the functions of the nodes. In custom systems or simulations, IDs can be assigned manually or programmatically, following these priority rules to replicate real-world message handling and arbitration.

Each CAN node can act as either a **transmitter** or a **receiver**, depending on the network's requirements. For instance, nodes transmitting diagnostic or control data, such as engine speed or temperature, may do so continuously, while others might only transmit during specific conditions, such as a system fault. Nodes are also designed to respond to specific message requests from other nodes in the network through a process known as **Remote Transmission Request** (RTR). This functionality enables dynamic data exchange and facilitates real-time monitoring of critical system parameters.

Each node's role and functionality are defined through the types of messages it can send and receive. This message-centric approach differentiates CAN from traditional address-based networks, as CAN does not rely on node addresses. Instead, each message carries an **identifier** that not only defines the **message's priority** but also acts as an **indicator of its content**. This setup allows nodes to communicate flexibly and efficiently in a broadcast model, where all nodes on the bus receive and interpret every message but only respond to the ones relevant to them.

The fundamental structure of CAN messages includes fields such as the start of frame (SOF), arbitration field, data length code (DLC), data field, CRC field, acknowledgment slot, and end of frame (EOF) (see figure 1).

There are four main types of CAN messages or "frames" that will be defined in the project with their respective fields presented below.

- **Data Frame** represents the most common frame, used to send actual data from one node to the entire network. It contains an identifier that establishes the message priority and up to 8 bytes of data.
  The following fields are standardized as follows for the Data Frame messages[4]:
  1. **R**emote **T**ransmission **R**equest bit must be dominant (0);
  2. **Id**entifier **E**xtension bit must be dominant (0) for the base frame format with 11-

bit identifiers;
3. **C**yclic **R**edundancy **C**heck delimiter bit must be recessive (1);
4. In **Ack**nowledgement Slot, transmitter sends recessive (1), and any receiver can assert a dominant (0);
5. **Ack**nowledgement delimiter must be recessive (1);
6. End-of-frame must be recessive (1);

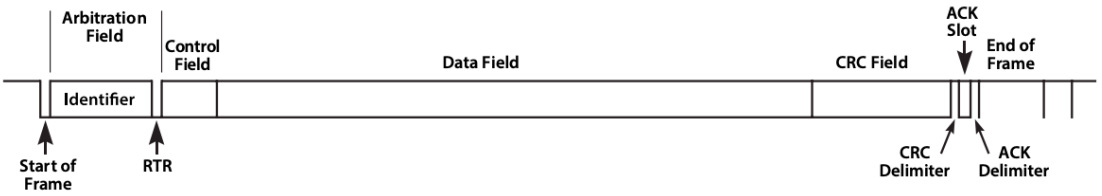


*Figure 1: CAN Standard Data Frame [1]*

- **Remote Frame** allows a node to request data from another node by transmitting only the identifier without any data. The node holding the data corresponding to the identifier will then respond with a Data Frame.
  There are some differences between a data frame and a remote frame:
  1. **R**emote **T**ransmission **R**equest bit is recessive (1) in the Remote Frame
  2. No Data Field in the Remote Frame; is transmitted as a dominant bit in the data frame and secondly in the remote frame there is no data field.
  3. **D**ata **L**ength **C**ode field indicates the data length of the requested message, not the transmitted one.

  In the event of a data frame and a remote frame with the same identifier being transmitted at the same time, the data frame wins arbitration due to the dominant RTR bit following the identifier.

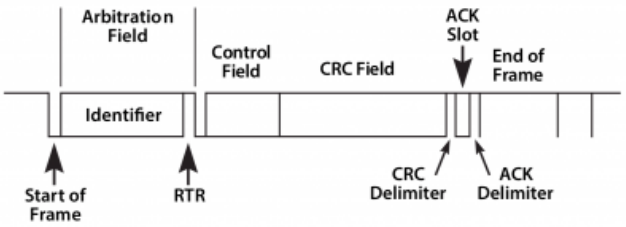


*Figure 2: CAN Standard Remote Frame [1]*

- **Error Frame** is broadcasted automatically by any node that detects an error in a message transmission.
  It consists of an **Error Flag**, which is 6 bits of the same value (thus violating the bit-stuffing rule) and an **Error Delimiter**, which is 8 recessive bits. The Error Delimiter provides some space in which the other nodes on the bus can send their Error Flags when they detect the first Error Flag. The Error Frame prompts other nodes to disregard faulty messages and retransmit them.

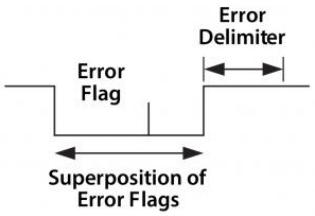

*Figure 3: CAN Error Frame [1]*

- **Overload Frame** is used to introduce a delay between message transmissions, allowing nodes time to process incoming data. This frame is less common but is beneficial in systems with high traffic, where nodes may need extra time before handling the next message. It consists of two bit fields: **Overload Flag** and **Overload Delimiter**. The Overload Flag has six dominant bits, and the Overload Delimiter consists of eight recessive bits.

The connection between nodes in a CAN network is defined by the messages themselves. Each node is designed to listen to the CAN bus and interpret messages based on their identifiers. This means that nodes don't send messages directly to one another; instead, they broadcast messages to the network, where every node receives them simultaneously. Each node then decides whether to accept or ignore a message based on its identifier, ensuring flexibility in data transmission.

By organizing communication around message types and identifiers, CAN networks achieve a scalable, decentralized setup that supports real-time, reliable data exchange across various applications. This broadcast-based model makes it easy to add new nodes without complex reconfiguration, as each node can immediately start receiving and processing network messages based on the identifiers they recognize.

## 2.2 Message Transmission

In a Controller Area Network (CAN), arbitration and priority control are essential processes that manage the simultaneous communication attempts of multiple nodes. As a multi-master system, CAN allows any node to initiate transmission when the bus is idle. However, in cases where multiple nodes attempt to send messages simultaneously, an efficient mechanism is required to prevent data collisions and ensure that high-priority messages are transmitted first. CAN achieves this through a unique process called **bitwise arbitration** based on each message's identifier, with lower values indicating higher priority.

Arbitration operates by transmitting each identifier bit-by-bit while all transmitting nodes monitor the bus. Bits in CAN are classified as either dominant (logic 0) or recessive (logic 1). During the arbitration phase, if a node transmits a recessive bit but reads a dominant bit on the bus, it understands that a higher-priority message is competing for transmission. The node then immediately stops its transmission attempt, deferring to the message with the lower identifier (i.e., higher priority). This process is referred to as **non-destructive arbitration** because it ensures that no data is lost or corrupted during conflict resolution. The highest-priority message completes transmission without interference, while the lower-priority nodes wait for the next available transmission opportunity.

| | Start bit | ID bits | | | | | | | | | | The rest of the frame |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| **Node 15** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | |
| **Node 16** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | Stopped Transmitting | | | | |
| **CAN data** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | |

*Figure 4: Example of Arbitration process [4]*

This arbitration and priority control system is invaluable in real-time applications. For example, in automotive networks, messages related to crucial functions like braking or airbag deployment are given high-priority identifiers. This prioritization guarantees that critical data reaches the relevant components promptly, even in high-traffic conditions. Similarly, in industrial automation systems, emergency signals are prioritized to ensure rapid response times, enhancing system reliability and safety.

Beyond enabling efficient, high-priority message transmission, CAN's arbitration system supports flexible, decentralized network architecture. Nodes can be added or removed from the network without reconfiguring the system, as each message's identifier and priority are intrinsic to the message itself rather than controlled by a central entity. This design allows CAN networks to be easily scalable and adaptable to various applications. The benefits of CAN's arbitration and priority control include efficiency, as high-priority messages are sent without delay; flexibility, allowing seamless integration of new nodes; and data integrity, with collisions avoided and accurate message transmission maintained. Altogether, these features make CAN a reliable choice for real-time, high-stakes environments, such as automotive, medical, and industrial automation applications.

## 2.3 Error Handling

In CAN networks, error detection and recovery mechanisms play a crucial role in ensuring reliable communication by identifying and handling transmission issues promptly. The CAN protocol employs several detection methods, including Cyclic Redundancy Checks (CRC), bit monitoring, and acknowledgment verification. These mechanisms allow a node to recognize errors within a transmitted frame. When an error is detected, the node raises an Active Error Flag and broadcasts an Error Frame to notify all other nodes. This flag instructs the network to discard the corrupted message, reset, and prepare for retransmission.

Each node tracks errors through two counters, the **Transmit Error Counter** (TEC) and **Receive Error Counter** (REC), which increase as errors are detected. When a transmit error occurs, the TEC is incremented by 8 points, while a receive error adds 1 point to the REC. Conversely, correctly received or transmitted messages reduce the counters, as each success gradually decreases the respective error points [1]. Nodes transition through three states based on these error counts:

**Error Active**: In this initial state, a node communicates normally but monitors errors closely. Each detected error increases the TEC or REC count.

**Error Passive**: If errors persist and a node's counter exceeds a set threshold, the node enters the Error Passive state, reducing its network impact. In this state, it can still receive messages but sends **Passive Error Flags** instead of Active ones, which minimizes interference with other network traffic.

**Bus Off**: Persistent errors cause the node to reach the Bus Off state, effectively isolating

it from the network to prevent further disruptions. This isolation protects other nodes from potential data corruption or ongoing errors.

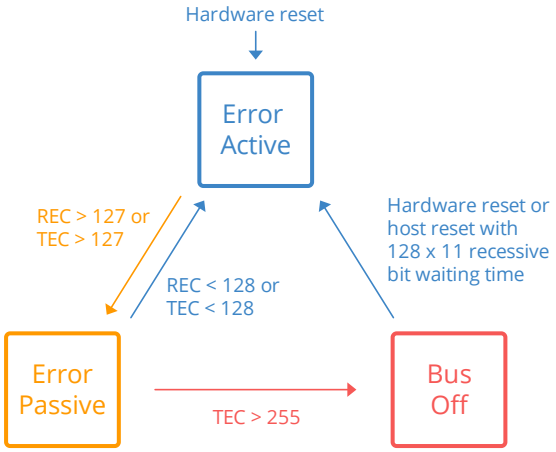


*Figure 5: CAN node error tracking through state changes [5]*

When a node recovers from errors, CAN's automatic retransmission mechanism allows it to resend messages without manual intervention, maintaining network flow. If a Bus Off node corrects its issues, it re-enters the network gradually after a reset to prevent reintroduction of errors. This process ensures stability and resilience, which are especially important for high-stakes, real-time applications in fields like automotive and industrial control.

# Analysis

## 3.1 Project Proposal

The final simulation of a Controller Area Network communication will encompass the features described below.

**List 3.1.1: Features of the simulation of CAN communication**

1. Dynamic Node Management
2. Message Creation and Transmission
3. Error Injection and Handling
4. Real-time Visualization
5. Predefined Scenarios

## 3.2 Project Analysis

### 3.2.1 Simulating CAN network communication

The specific goals of the CAN network simulation center on accurately modeling message transmission, error detection, and priority-based arbitration within industrial network scenarios. The primary objective is to create a simulation that replicates key CAN communication mechanisms to illustrate how these networks handle simultaneous transmission attempts, detect and manage errors, and maintain reliable communication between nodes with varying message priorities. By modeling these processes, the simulator serves as an educational tool to demonstrate the robustness of CAN protocols in environments requiring high reliability and efficiency, such as industrial automation and automotive systems.

The CAN network simulator is equipped with a range of features that enhance both functionality and interactivity:

- **Predefined and Interactive Scenario Execution**: Predefined and custom scenarios allow users to simulate conditions such as node failure, high network load, and error-prone environments, giving insight into how the CAN protocol manages these situations and maintains reliable communication.
- **Dynamic Node Management**: Users can add, remove, and configure nodes within the network, allowing for customized network setups that reflect real-world scenarios.
- **Error Injection**: The simulator enables users to introduce errors into the network, such as CRC mismatches or acknowledgment failures, to test the robustness and recovery mechanisms of the CAN protocol.
- **Real-Time Visualization**: The simulator includes a graphical interface that visualizes message flow, node status, and error occurrences in real-time, providing an intuitive and engaging learning experience.

### 3.2.2 Simulation Requirements and Capabilities

This section outlines the requirements and capabilities needed for an effective CAN network simulation, specifying the simulation's intended functionalities and performance expectations.

- Functional Requirements:
  - The simulator must allow for the creation and deletion of nodes, each configurable with unique message IDs and transmission intervals.
  - The simulator should support various CAN message types (Data, Remote, Error, Overload) and apply priority-based arbitration for message transmission.
  - Error detection and handling features must be included, such as CRC checks, acknowledgment verification, and handling of errors through retransmission and state changes (e.g., Error Active, Error Passive, Bus Off).
  - The simulation must support both predefined and interactive modes, allowing users to execute and modify scenarios in real-time.
- Non-Functional Requirements:
  - The simulator should provide a graphical interface that intuitively displays message flow, node activity, and error occurrences in real-time.
  - The system must be responsive, with minimal delay in updating the visualization as user interactions and node changes occur.
  - The simulation should be modular, allowing for easy addition of new message types, error types, or scenario options in the future.

### 3.2.3 Use Case Scenarios

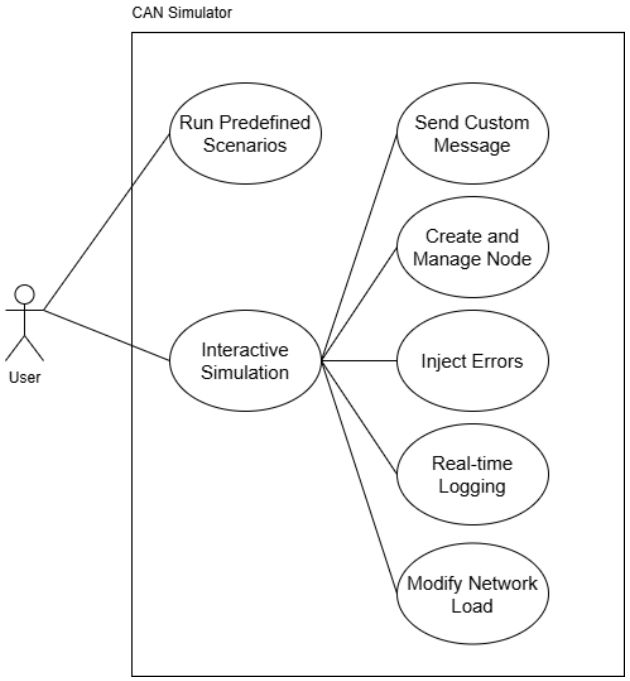


*Figure 6: Use Case Diagram - Main Functionalities of the CAN Network in the Simulation*

#### 3.2.3.1 Run Predefined Scenarios

- Basic Message Transmission
    - Data Frame
    - Remote Frame
    - Error Frame
- Arbitration Test
- Error Detection and Handling
    - Bit Monitoring
    - Bit Stuffing
    - Frame Check
    - Acknowledgement Check
    - Cyclic Redundancy Check
- Node Failure and Recovery

These scenarios comprehensively test the CAN protocol's core principles, including priority handling, error detection, fault recovery, and high-load efficiency, ensuring the simulation accurately models CAN's robust communication framework.

#### 3.2.3.2 Interactive Simulation

In the Interactive Simulation, the user will be able to perform a variety of actions to test and explore the CAN network's capabilities. The following list represents the features that are to be included in the simulation:

- Send Custom Messages: Users can create and send custom messages across the network, selecting message types (e.g., Data, Remote, Error), setting priorities, and observing how they interact within the simulation.
- Create and Manage Nodes: Users can add, configure, or remove nodes from the network, simulating dynamic changes in network topology. This allows testing of node behaviour in both normal and error conditions.
- Inject Errors: Users can simulate errors such as bit errors, CRC mismatches, or acknowledgment failures. This feature helps to observe how error handling and recovery mechanisms operate under faulty conditions.
- Real-Time Logging: The simulation will provide real-time logging of network activity, displaying information about messages sent, errors encountered, and state transitions. This helps users analyse network behaviour closely.
- Modify Network Load: Users can adjust the network load, increasing or decreasing message traffic to see how the CAN network handles high and low traffic volumes, which is useful for testing priority and arbitration under stress.

These features make the interactive simulation a comprehensive tool for learning and testing CAN protocol behaviour in various situations, giving users hands-on control over the network environment.

# Design

## 4.1 CAN Network Simulation Overview

The CAN Network Simulation is structured around three primary classes: **CAN Bus Class**, **CAN Node Class**, and **CAN Message Class**. The CAN Bus serves as the central hub, managing message flow, prioritizing transmissions, and handling errors. CAN Node instances represent individual devices on this network, each capable of sending and receiving messages but relying on the bus for arbitration and communication. CAN Message represents various message types (Data, Remote, Error, Overload) exchanged between nodes. Each class interacts through dependencies and associations: the CAN Bus acts as the mediator, CAN Node creates and processes messages, and CAN Message encapsulates data and control information, creating a cohesive structure that visualizes how classes connect and manage data flow.

Together, they form an interconnected system that accurately simulates CAN network operations. The bus coordinates all interactions, nodes represent devices that send and receive messages, and messages carry the data and signals that enable network communication, error handling, and prioritization.

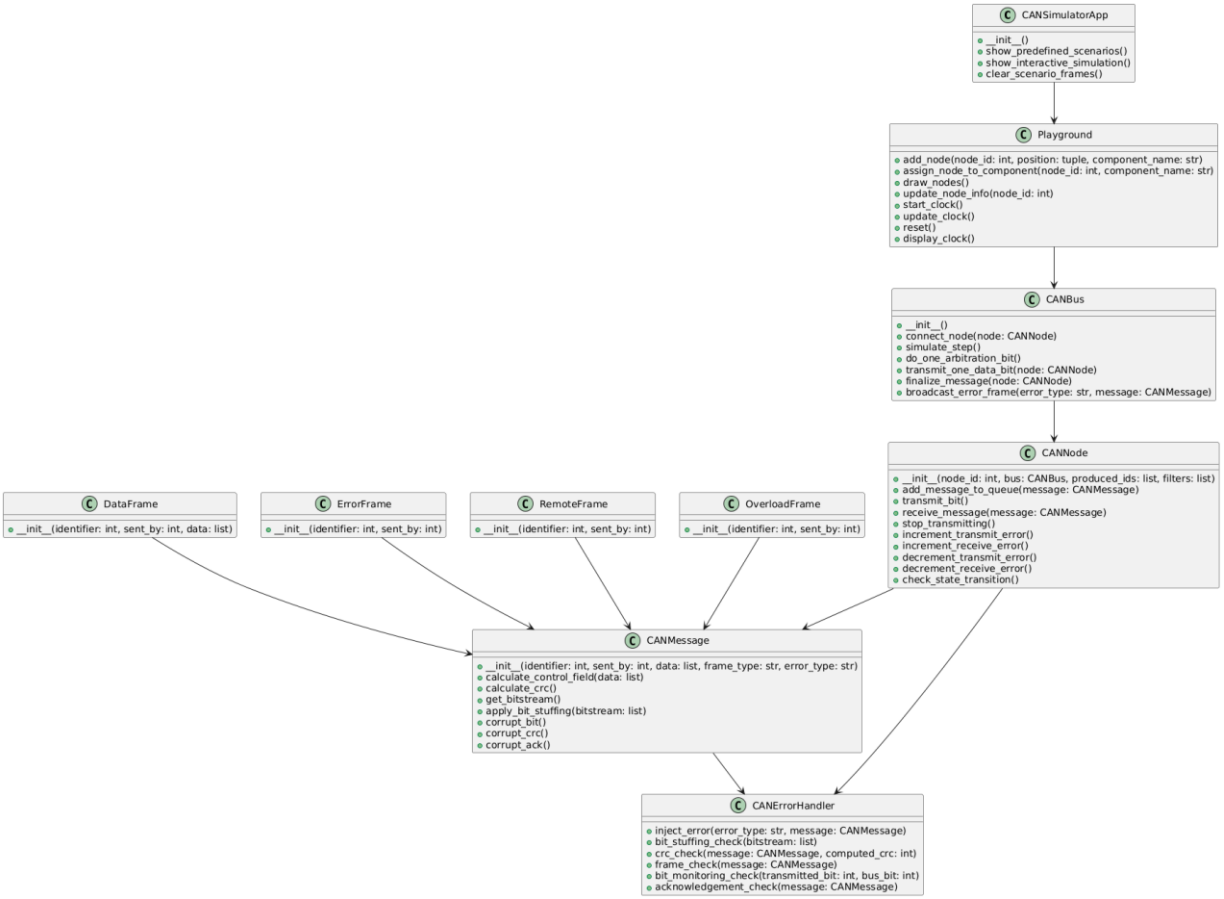


*Figure 7: Class Diagram for CAN Simulator*

## 4.2 Components Design

### 4.2.1 CAN Bus Class

The CAN Bus serves as the **central communication channel** in a Controller Area Network, responsible for managing message flow among nodes and prioritizing transmissions. It uses bitwise arbitration to determine message priority, where lower message IDs (higher priority) dominate during simultaneous transmissions. This non-destructive arbitration ensures efficient and collision-free message handling, making CAN suitable for real-time applications.

In terms of error management, the CAN Bus focuses on detecting basic transmission issues and relies on a streamlined approach to notify other components when errors are encountered, allowing the network to maintain reliability without interruptions. This combination of arbitration and basic error detection ensures high-priority, fault-tolerant communication across the network.

### 4.2.2 CAN Node Class

A CAN Node represents an **individual device** within the CAN network, capable of sending and receiving messages. Nodes use the CAN Bus to manage message transmission, relying on the bus for arbitration to ensure that higher-priority messages transmit first. Each node tracks its status and adjusts its behaviour accordingly to maintain stable communication within the network.

Nodes handle message creation, assigning unique IDs and adding necessary fields (such as CRC for error checking) to ensure CAN compliance. Messages are queued and transmitted through the CAN Bus, where arbitration determines which messages are prioritized. This design allows each CAN Node to operate independently while contributing to coordinated, prioritized communication in the network.

### 4.2.3 CAN Message Class

A CAN Message represents a **data packet transmitted** across the CAN network, with each message having a unique identifier that determines its priority during transmission. It includes fields like the control field, CRC for error checking, and payload data. CRC and other integrity fields are essential for ensuring message accuracy and reliability in the network.

Each CAN Message is structured with start/stop bits, control fields, and an algorithm for CRC calculation. Subclasses such as **Data Frame**, **Remote Frame**, **Error Frame**, and **Overload Frame** extend CAN Message for specific message types, each handling data payloads and signalling differently. This structure allows the network to accommodate a variety of message functions and requirements seamlessly.

### 4.2.4 CAN Error Handler Class

The CAN Error Handler class is responsible for **managing error detection and handling** within the CAN network simulation. It centralizes error-handling functions,

detecting issues such as CRC mismatches, acknowledgment errors, and bit-stuffing violations during message transmission. Each node relies on CAN Error Handler to monitor these issues and to initiate error responses, such as broadcasting error frames and updating error counters. This functionality helps the network maintain stability by isolating nodes that exceed acceptable error thresholds, following CAN protocol standards for error handling.

### 4.2.5 CAN Simulator Class

The CAN Simulator class serves as the **main control hub** for the entire CAN network simulation. This class initializes and manages all core components, including the CAN Bus, multiple CAN Node instances, and predefined scenarios for testing purposes. Acting as the orchestrator of the simulation, CAN Simulator allows users to interact with the CAN network by starting or stopping simulations, injecting custom messages, and manipulating node settings. It also provides an interface for real-time feedback and visualization, giving users insights into network activity and error handling as it occurs.

In the CAN Simulator class, there is a **set of predefined conditions** that simulate various network behaviours in the CAN simulation, such as high traffic loads, error-prone environments, and node failures. This feature allows users to observe how the network responds to different scenarios, offering insight into how CAN protocols manage challenging conditions. Each scenario modifies aspects of the simulation, such as adjusting node properties or generating specific types of message traffic, to replicate real-world conditions in a controlled environment.

This way, CAN Simulator controls when and how scenarios are executed.

# Implementation

The simulation operates in a continuous loop where nodes send messages to the CAN bus, which manages arbitration based on message ID priority to determine the order of transmission. Throughout this process, the messages are monitored closely to detect and respond to issues in real-time. The user interface (UI) provides a visual representation of network activity, including message flow, priority handling, and error states.

## 5.1 Overview of Core Components

Each component plays a unique role in simulating CAN network functionality, from managing message flow and priority-based arbitration to detecting errors and providing real-time feedback.

- **CAN Bus**: The CAN Bus class acts as the central communication hub, managing message flow and arbitration across nodes. It contains methods to *add nodes*, *check message queues*, and *determine message priority* using **bitwise arbitration**. By interacting with the error handler, it ensures *error detection and response*, *broadcasting error frames* and introducing *overload frames* to regulate traffic and maintain network stability.
- **CAN Node**: The CAN Node class represents individual devices on the network. Each node can *generate and queue messages*, *set error states*, and *handle various error types*. With attributes such as message queue, error counters, and state information, nodes adapt dynamically to network conditions. They rely on CAN Error Handler for *error detection* and *send messages* via the CAN Bus for prioritized transmission based on their identifiers.
- **CAN Message**: The CAN Message class defines the message structure for network communication. Messages include fields like *identifier*, *data field*, and *cyclic redundancy check*, which are essential for message prioritization and integrity. The class has subclasses, including **Data Frame**, **Remote Frame**, **Error Frame**, and **Overload Frame**, each tailored to specific CAN functions. CAN Message also supports *bit-stuffing* and *bitstream generation* for secure transmission across the bus.
- **CAN Error Handler**: The CAN Error Handler class centralizes error management, providing methods to *detect bit errors*, *CRC mismatches*, *acknowledgment failures*, and *bit-stuffing violations*. It maintains error counters and flags nodes when errors are identified. Error types trigger specific responses, such as broadcasting error frames, updating node states, or injecting simulated errors for testing scenarios.

## 5.2 Message Transmission Process

The message transmission process in the CAN network simulation models the creation, arbitration, and delivery of messages, maintaining priority-based, collision-free communication.

Each node can generate messages through the *send_message()* method, selecting a frame type and adding a unique identifier, data payload, and an error type for injecting

errors. Created messages are placed in the node's message queue for transmission.

When multiple nodes have messages ready, bitwise arbitration is performed, selecting the message with the lowest identifier (highest priority). The *simulate_step()* method retrieves and ranks messages, transmitting the highest-priority message while queuing the rest for future cycles. This process ensures that lower-priority messages wait, maintaining order and preventing data collisions.

The *deliver_message()* method broadcasts the prioritized message bit by bit to all nodes. Each node receives and validates the message based on CRC and acknowledgment checks. Nodes respond to valid messages, and if no acknowledgment is detected, the error handler is triggered to handle the transmission error. This bit-by-bit delivery, coupled with acknowledgment and error handling, ensures reliable and organized communication on the network.

## 5.3 Error Detection and Handling Techniques

The CAN Node class includes several methods to detect and respond to specific types of errors during message transmission and reception. Each node maintains an internal state and error counters (**Transmit Error Counter** and **Receive Error Counter**) that adjust dynamically based on detected errors. Errors are identified by comparing expected and actual values for various message elements or through checks implemented by the CAN Error Handler. Inside the error handler, there are various methods for detecting specific errors:

- **Bit Monitoring**: The *bit_monitoring()* method compares transmitted and received bits, detecting inconsistencies that indicate transmission faults.
- **Bit Stuffing Check**: The *bit_stuffing_check()* method scans the bitstream for more than five consecutive identical bits, flagging any violations as stuffing errors.
- **Frame Check**: The *frame_check()* method validates that the message structure adheres to CAN protocol specifications, ensuring correct delimiters and frame lengths.
- **Acknowledgment Check**: The *acknowledgement_check()* method verifies if the message's acknowledgment slot is set correctly, flagging any unacknowledged messages.
- **CRC Check**: The *crc_check()* method compares the message's CRC value with a recalculated CRC, identifying data integrity issues.

Additionally, there is a method for *injecting errors* that allows for simulating errors by modifying specific message fields based on the error type, supporting testing and scenario-based error handling.

## 5.4 User Interface and Predefined Scenarios Implementation

The user interface is intended to provide an interactive and visual platform for users to observe and control the CAN network. The implementation of the interactive simulation and predefined scenarios in the simulation closely follows the design philosophy of

professional CAN network tools[6]. Each node is represented visually, showing its state, such as Error Active, Error Passive, or Bus Off, transmit error counters (TEC) and receive error counters (REC).

The predefined scenarios provide structured, automated tests that simulate various network conditions, allowing users to observe how the CAN protocol handles priority management, error detection, and recovery in dynamic environments. Each scenario will be implemented as a function, controlling the behavior of nodes, message types, with specific conditions and actions tailored to simulate network behaviors. The **Message Transmission** scenario demonstrates how nodes send and receive Data Frames under normal operating conditions, with the CAN Bus managing the flow of information and ensuring proper acknowledgment. The **Arbitration Test** simulates simultaneous message transmissions by multiple nodes, showcasing the deterministic resolution of conflicts through bitwise arbitration. The scenario emphasizes how message identifiers dictate transmission priority and ensures the lowest identifier (highest priority) wins arbitration. Another critical scenario, **Error Injection**, introduces controlled faults like bit-stuffing violations, CRC errors, or acknowledgment failures. This test validates the system's error detection and correction mechanisms, observing how nodes respond to errors and transition through states such as Error Passive or Bus Off. Additionally, the **Node Failure Test** pushes individual nodes to their fault thresholds, highlighting how the network isolates malfunctioning nodes to maintain overall communication stability.

In addition to predefined scenarios, the simulation will include interactive features that allow users to manually adjust network conditions and observe CAN behavior. Each *interactive feature* will be implemented as a *control* within the CAN Simulator class and will be linked to *specific functions* within the CAN component classes to ensure that user inputs dynamically affect network conditions. These interactive controls provide a hands-on environment, allowing users to experiment with CAN protocol features and observe network behavior under different configurations, traffic loads, and error conditions. Users can dynamically add or remove nodes, configure message payloads, and inject specific errors into the network. For example, a user might introduce a CRC error during an ongoing transmission to observe how the network identifies and handles the issue. Another interactive control allows for adjusting the transmission interval of nodes, enabling experimentation with network traffic and load conditions. The simulation also supports custom message creation, where users can specify frame types, identifiers, and data payloads to test unique scenarios.

### 5.4.1 Concepts in CAN

In the *"Concepts in CAN"*, users can test basic functionalities of a CAN network. This feature is designed to help users understand core principles and operations of CAN communication. The options available are:

- *Transmit any type of frame*: Users can send Data Frames, Overload Frames, Error Frames, or Remote Frames from any node to observe how the network handles each type of message.

- *Transmit data frames with errors injected*: Users can introduce specific errors, such as:
    - Acknowledgment Error
    - Cyclic Redundancy Check (CRC) Error
    - Bit Monitor Error
    - Bit Stuffing Error
    - Form Error
- *Test arbitration*: Users can simulate a situation where multiple nodes attempt to transmit simultaneously and observe how the arbitration mechanism resolves conflicts based on message priority

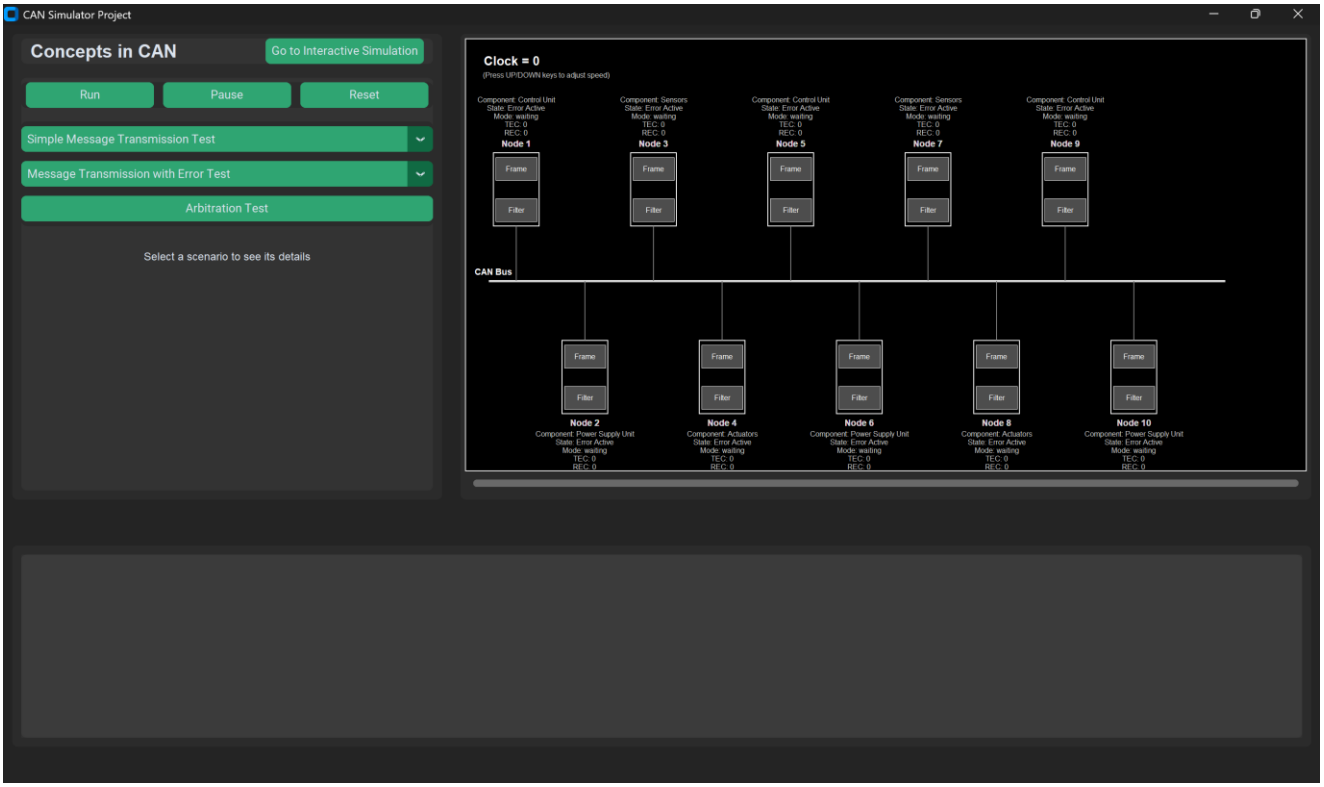


*Figure 8: "Concepts in CAN" Page*

### 5.4.2 Interactive Simulation

The *"Interactive Simulation"* page offers a dynamic environment for users to manipulate and experiment with the CAN network. Key functionalities include:

- *Select the message load*: Users can adjust the volume of message traffic in the network, simulating scenarios of low or high load conditions.
- *Edit node configuration*: Users can add new nodes, remove existing nodes, or change the configuration of any node to test different setups.
- *Send custom messages*: Users can create and send custom messages with specific parameters, such as frame type, message ID, and payload. Errors can also be injected into these custom messages.

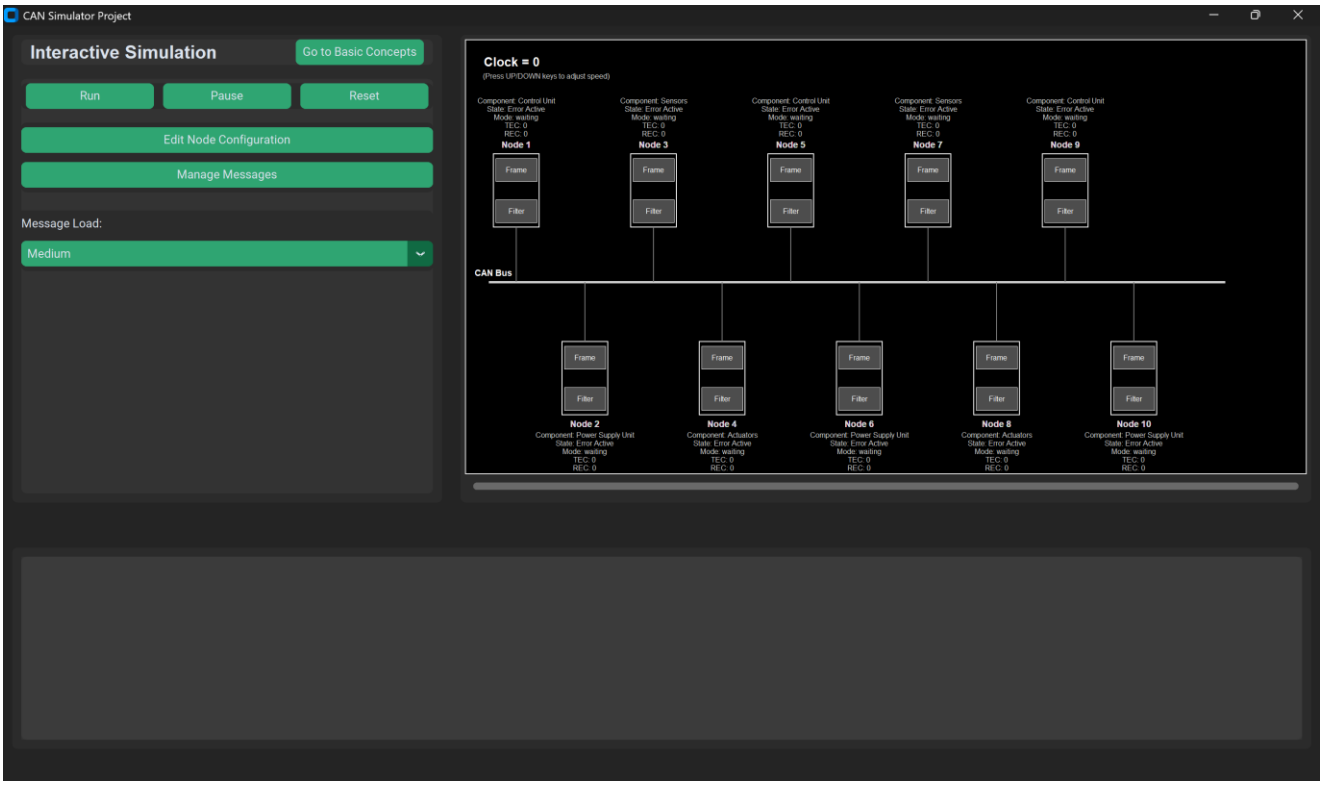


*Figure 9: "Interactive Simulation" Page*

# Testing and Validation

In this chapter are presented the functionalities of the CAN Simulator Application.

## 6.1 Testing Message Transmission and Arbitration

### 6.1.1 Frame Transmission Testing

This test validates that messages are transmitted correctly across the network, with nodes adhering to the CAN protocol.

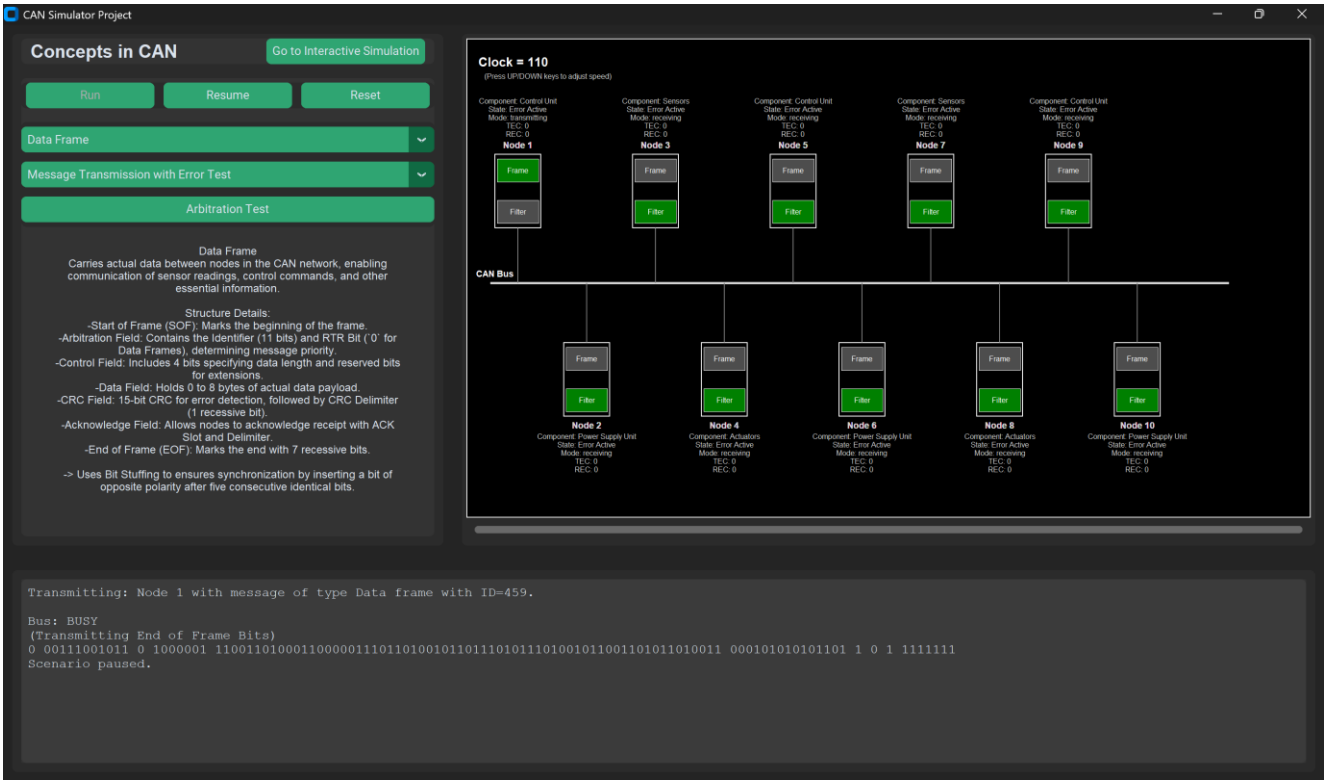


*Figure 10: Data Frame Transmission*

The process includes:

- Transmitting a Data Frame with a unique identifier and payload.
- Observing acknowledgment signals from receiving nodes.
- Confirming successful delivery through the real-time log.

The same process stands for all types of messages that are send through the CAN Bus.

### 6.1.2 Arbitration Testing

Arbitration testing ensures that when multiple nodes attempt to transmit simultaneously, the message with the highest priority (lowest identifier) is selected without data loss.


*Figure 11: Arbitration Example*

- Two or more nodes send messages with differing priorities.
- Observe the bitwise arbitration process in action, where lower-priority nodes defer to the higher-priority message.
- Verify that the winning node completes its transmission successfully.

# 6.2 Testing Error Handling and Node Failure

## 6.2.1 Error Testing

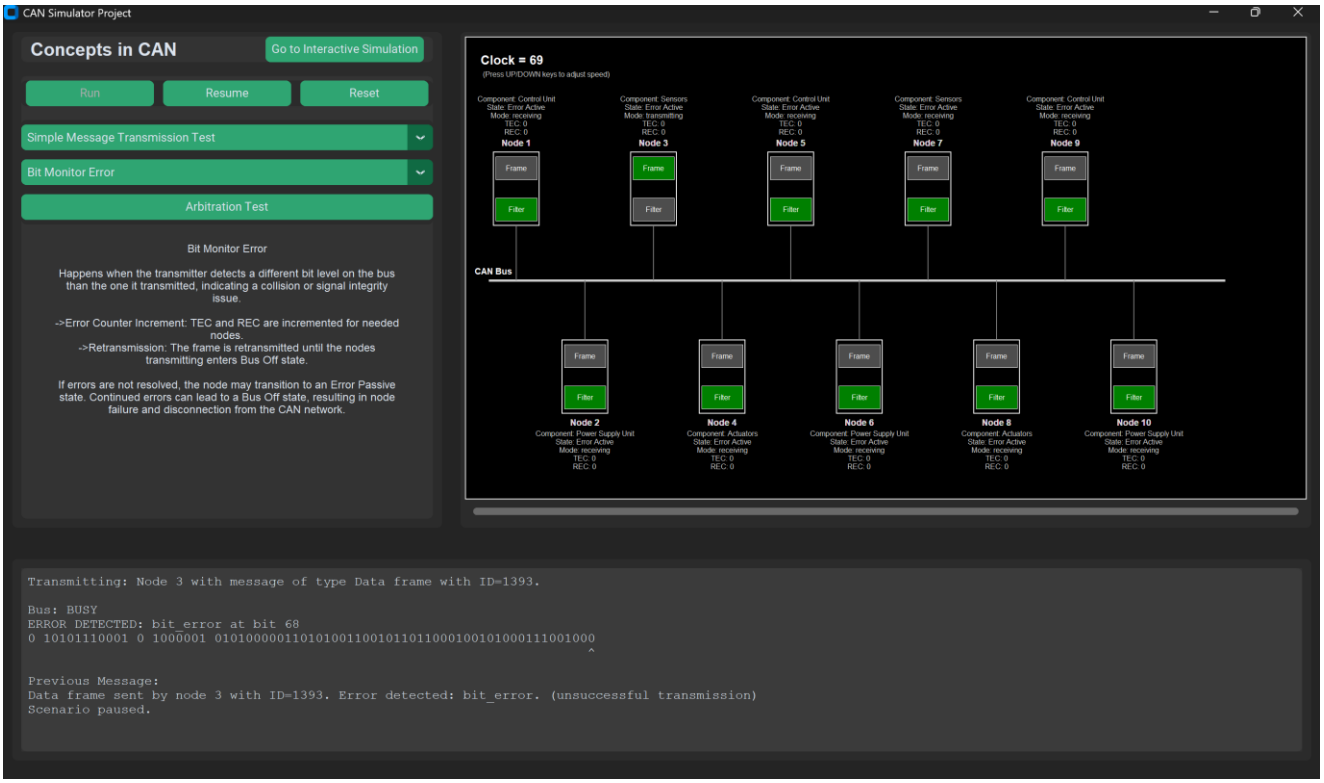


*Figure 12: Transmission of a Message with Bit Monitor Error Injected*

This test evaluates how the network responds to injected errors:

- Inject errors like CRC mismatch, bit-stuffing violation, or acknowledgment failure into a transmitted Data Frame.
- Observe the Error Frame broadcasted by the node detecting the issue.
- Verify how other nodes disregard the corrupted message and prepare for retransmission.

## 6.2.2 Node Failure Testing

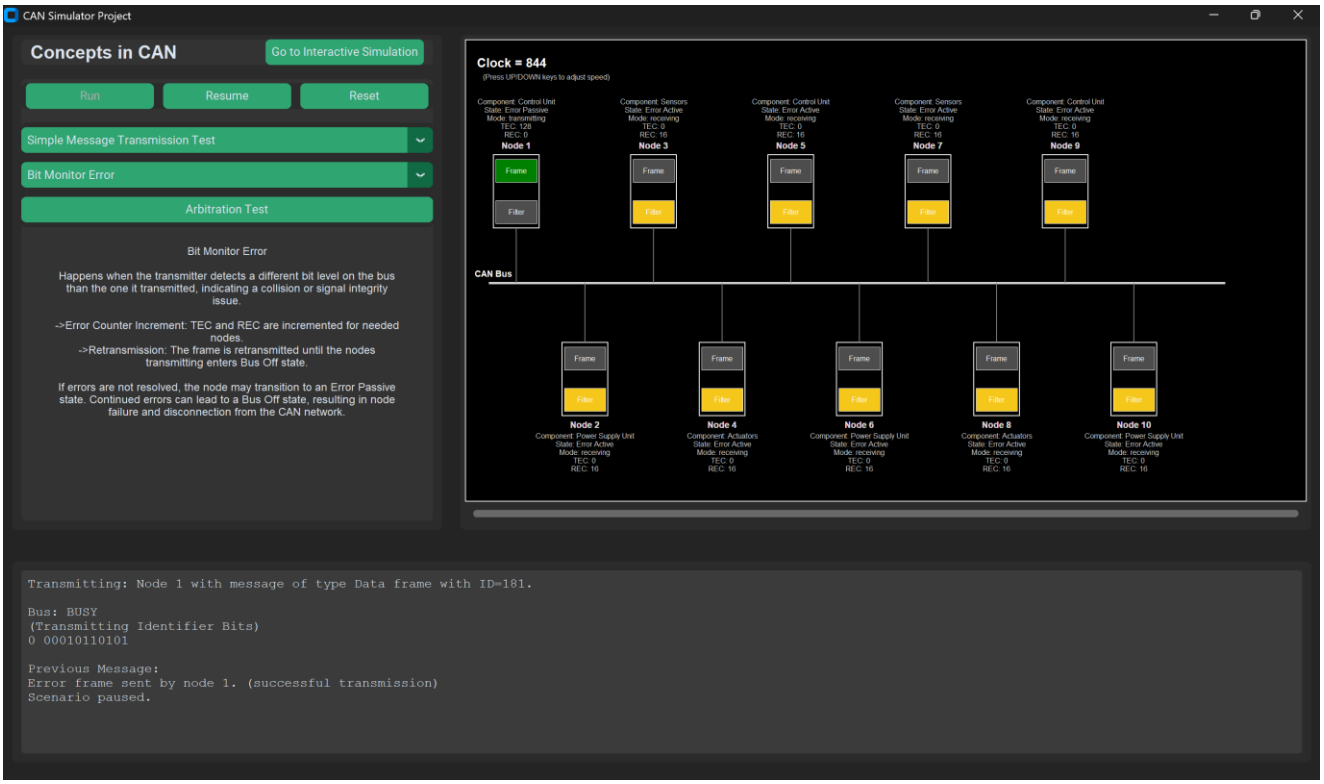


*Figure 13: Transition to Error Passive State*

- Simulate consecutive transmission errors to increase the Transmit Error Counter (TEC). (16 messages with error are transmitted for the transition to *Error Passive* state to take place, bringing the TEC to 128)
- Observe the node transitioning from *Error Active* to *Error Passive*.

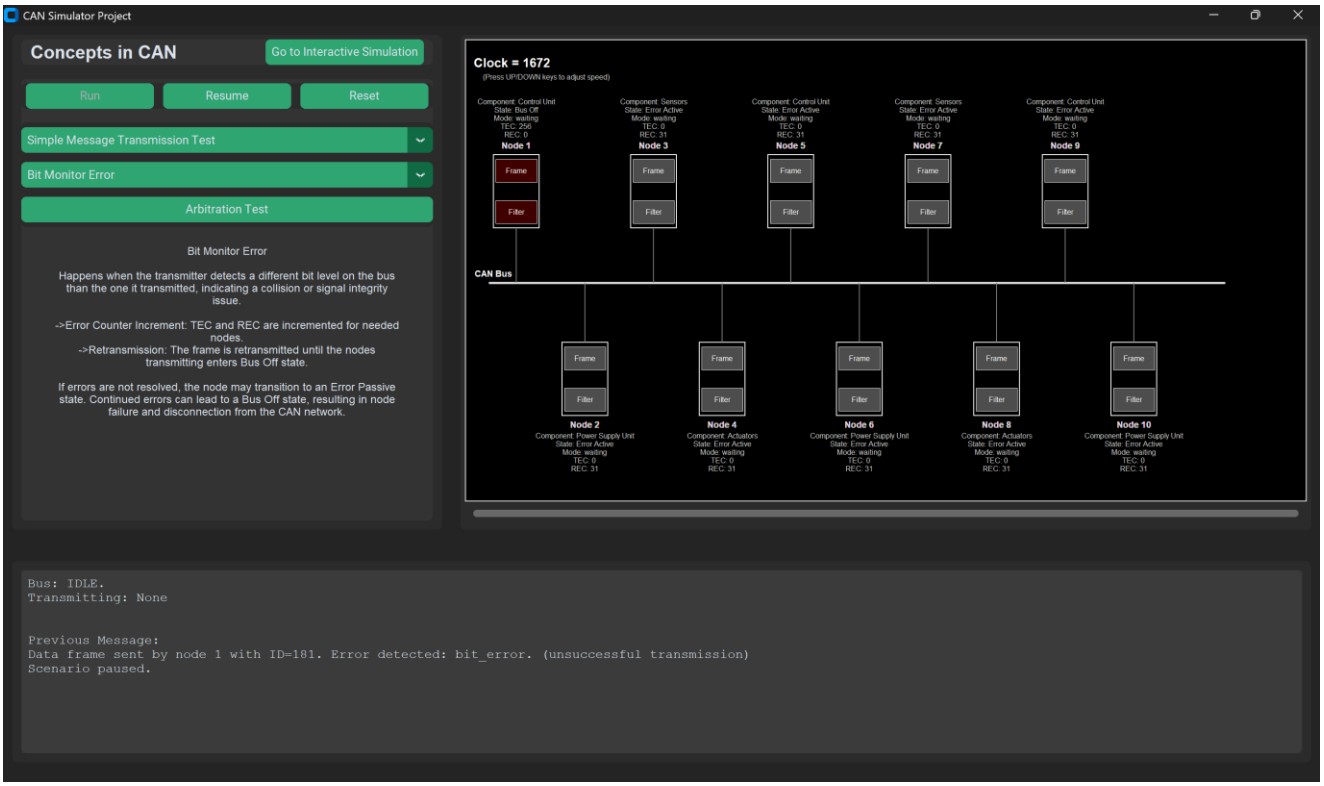


*Figure 14: Transition to Bus Off State*

- Continue to introduce errors until the TEC exceeds the Bus Off threshold. (32 messages with error must be transmitted for a node to reach the threshold, bringing the TEC to 256)
- Observe the node transitioning to the Bus Off state, isolating itself from the network.
- Verify that the node no longer participates in transmissions or receptions.

# Conclusion

This project successfully simulates communication in a Controller Area Network (CAN), highlighting essential features like message arbitration, error detection, and real-time visualization. By integrating predefined scenarios and interactive tools, the simulation provides a comprehensive understanding of CAN communication under various conditions, such as node failures, high traffic, and error-prone environments.

The simulation not only demonstrates the robustness of the CAN protocol but also serves as a valuable educational resource for exploring network behavior and testing system resilience. Future enhancements could include extending support for the extended frame format, implementing additional error handling mechanisms, or integrating the simulation with physical hardware to validate its real-world applicability.

# Bibliography


[1] Kvaser, “CAN protocol tutorial” [Online]. Available: https://www.kvaser.com/can-protocol-tutorial/

[2] Texas Instruments, “Introduction to the Controller Area Network (CAN)”, 2002. [Online]. Available: https://www.ti.com/lit/an/sloa101b/sloa101b.pdf

[3] Ashley Sharp, “Implementing Controller Area Network”, 2017. [Online]. Available: https:// www.theseus.fi/bitstream/handle/10024/140548/Thesis%20100.pdf

[4] Wikipedia, “CAN Bus” [Online]. Available: https://en.wikipedia.org/wiki/CAN_bus

[5] CSS Electronics, “CAN Bus Errors Explained – A Simple Intro”, 2023. [Online]. Available: https://www.csselectronics.com/pages/can-bus-errors-intro-tutorial

[6] Controller Area Network (CAN) in Automation (CiA), "CAN Knowledge." [Online]. Available: https://can-cia.org/can-knowledge