CODE RACE CHALLENGE 2025

– TEAM WECAN–

PROJECT NAME

"Support System For Monitoring Startup And Parking Procedures For New Drivers Based On CAN Signals"

Members

Ngô Minh Huấn	Ho Chi Minh City University of Technology
Phạm Trần Đức Hạnh	Ho Chi Minh City University of Technology
Lê Văn Đình Huy	Ho Chi Minh City University of Technology

16/7/2025

Table of Contents

1 Software Design

1.1 Idea Specification

1.1.1 Overview

Driving at the **beginning and end of a trip**—when starting the engine or parking the vehicle—is often challenging for **inexperienced drivers**. Misjudgments during these moments can lead to:

- Gear misuse
- Engine strain
- Risk of vehicle roll-away

This project presents a **simple yet effective embedded support system** that detects and alerts unsafe driving conditions during *start-up* and *post-parking* phases, using only standard **CAN bus signals**.

1.1.2 Key Features

- **Gear Detection**: via `CAN ID 0x401`, identifies gear state: `P (Parking)`, `R (Reverse)`, `N (Neutral)`, `D (Drive)`.
- **Engine Status Monitoring**: via `CAN ID 0x380`, distinguishes between *Pre-Ignition*, *Ignition*, and *OFF*.
- **Real-Time Feedback**: Intuitive warnings displayed on a compact OLED screen.
- **Minimal Hardware Setup**: Based on `Arduino-compatible MCU`, `MCP2515 CAN module`, and `SH1106 OLED`—ensures low cost and simple integration.

1.1.3 Use Case Scenarios

Case 1 – Start-Up Reminder for Safe Driving: When the driver turns on the ignition and prepares to shift to Drive (`D`), the system displays a instruction: *"Press brake fully"* — guiding new drivers to follow safe procedures during vehicle start-up.

Case 2 – Forgetting to Shift to Parking After Stopping: Engine is turned OFF but gear is not in `P`. The system displays: *"Warning: Roll | Instruction: Shift to P or Hold Brake"* – preventing potential rollback incidents.

1.1.4 Why It Matters

- **Beginner-Friendly**: Tailored for learner drivers or those with limited experience.
- **Safety-Oriented**: Prevents critical low-speed errors with real-time assistance.
- **Simple, Reliable, Embedded**: No need for complex AI, vision systems, or extra sensors.
- **Bosch-Relevant**: Aligns with Bosch's vision for *affordable vehicle intelligence* based on ECU data.

1.1.5 Impact

This project is a practical demonstration of how **low-cost embedded systems** can be used to enhance driver safety by utilizing existing vehicle infrastructure. It enables:

- Early driver confidence
- Safer behavior during parking and start-up
- A scalable, maintainable support solution for smart mobility

1.2 Signal Definition and CAN Data Interpretation

The system relies on real-time CAN Bus signals to detect unsafe vehicle states during engine startup and vehicle shutdown. Specifically, it tracks gear position, engine status and vehicle speed. These inputs allow the system to provide real-time driver guidance and trigger safety alerts.

1.2.1 Overview of Hardware Modules for CAN-Based Input and Output Integration

To build the Safe Start and Parking Detection System, the following hardware modules are integrated:

- **Arduino Mega 2560**: Acts as the central node responsible for receiving CAN messages from the vehicle network, interpreting gear and engine signals, and managing display and alert logic. Its multiple serial and digital I/O ports make it ideal for handling concurrent communication and output.

- **Arduino Nano**: Functions as a **dedicated CAN message simulator**, emulating gear-related CAN signals in the absence of a real vehicle ECU. It interfaces with a **4x4 matrix keypad**, where each key represents a specific driving gear (e.g., `P`, `R`, `N`, `D`) or other input conditions. When the user presses a key, the Nano interprets the input and constructs a corresponding CAN message that mimics how an actual transmission control unit would encode the gear state.
- **MCP2515 CAN Bus Module (x2)**: Two modules are used:
  - One connected to the Arduino Mega for receiving gear and engine state messages from the CAN network.
  - One connected to the Arduino Nano for injecting user-defined gear messages via the keypad (used in simulation or testing scenarios).

  The MCP2515 handles low-level CAN protocol and interfaces via SPI with the microcontrollers.
- **4x4 Matrix Keypad Module**: Provides user input for selecting gear states in simulation mode. Each button can be mapped to a specific gear (e.g., P, R, N, D) or command, allowing manual triggering of CAN messages for development or demo purposes.
- **SH1106 OLED Display (128x64)**: Displays real-time driving status including current gear, engine state, and safety reminders. The high-contrast OLED display ensures good visibility and intuitive feedback for the driver in all lighting conditions.
- **SPI and I^{2}C Wiring**: The system uses SPI communication for MCP2515 modules and I^{2}C for OLED display, enabling efficient parallel integration of input/output devices with minimal pin usage.

1.2.2 CAN Messages Used in the Project

This project utilizes three specific CAN messages sourced from the official DBC file of the Honda City 2018, provided by the competition organizers. These messages include:

- **CVT_191 (CAN ID 0x401)**: Contains information about the current gear state of the vehicle. In particular, bits 0–3 of byte 0 represent the selected gear as follows:
  - Bit 0 – Parking (P)

  - Bit 1 – Reverse (R)
  - Bit 2 – Neutral (N)
  - Bit 3 – Drive (D)

- **ENG_17C (CAN ID 0x380)**: Provides engine-related status, including:
  - Bit 5 of byte 5 (bit index 45) – Pre-Ignition (Key ON)
  - Bit 7 of byte 5 (bit index 47) – Engine Running (Ignition ON)
- **Custom Simulation Message**: While not originating from the DBC directly, a custom-formatted message is generated on the Arduino Nano to simulate gear and engine conditions based on user input. This message format strictly follows the field layout of the real CVT_191 and ENG_17C messages to ensure compatibility with the receiver system.

Why Use the Original Layout? To achieve realistic system behavior and facilitate accurate testing, we adhere to the exact bit layout defined in the Honda DBC file. By doing so:

- The receiver (e.g., Arduino Mega) can parse simulated messages identically to real in-vehicle messages.
- Debugging and future integration with actual car systems becomes seamless.
- This approach ensures that only a change in message source (real ECU $\rightarrow$ Nano) is needed, with no modifications in receiver-side code logic.

Simulation via Arduino Nano The **Arduino Nano** reads user input from the keypad and generates corresponding CAN messages using the `MCP2515` module. These messages mimic the structure of `CVT_191` and `ENG_17C` exactly—bit-for-bit—ensuring that the receiving system cannot distinguish between simulated and real data. This design enables the entire system to be tested in lab environments without the need for a real vehicle or live CAN traffic.

1.2.3 Bit-Level Signals Used in the System

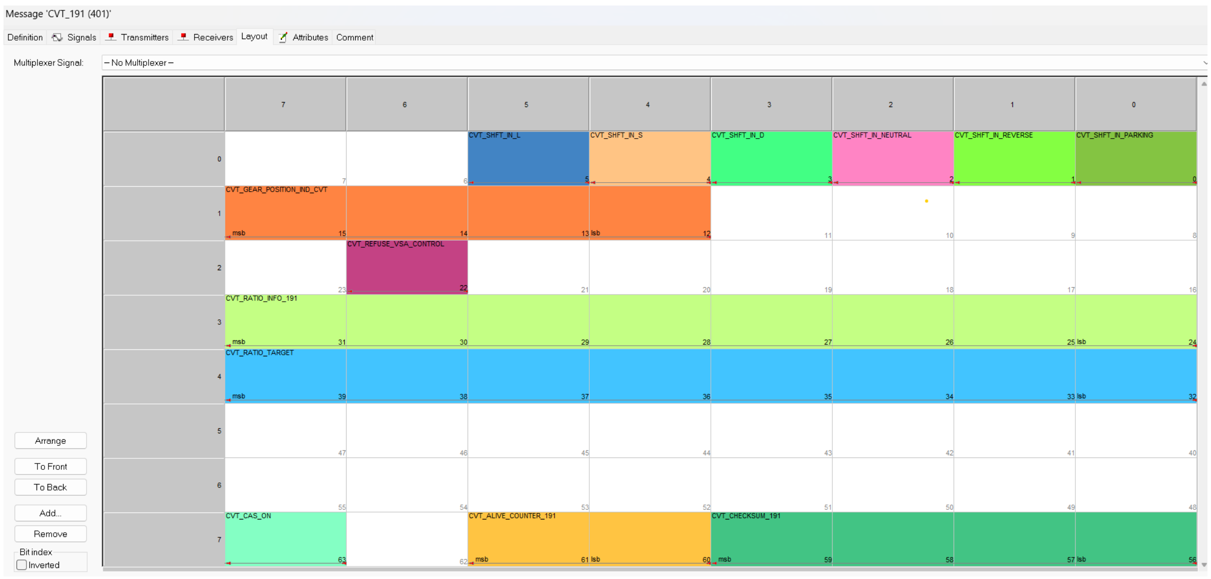
From the parsed CAN messages `CVT_191 (ID 0x401)` and `ENG_17C (ID 0x380)`, the project extracts a number of individual bit signals that are crucial for monitoring gear status and engine progress. These bit-level signals are described below:

1. Gear Shift Signals from CVT_191 (CAN ID 0x401) The gear status is determined by reading the lower 4 bits of `byte 0` from the CVT_191 message. Each bit represents a unique gear state.

- **CVT_SHFT_IN_P** (Bit 0, Byte 0): Indicates that the gear lever is in Parking (P) mode.
- **CVT_SHFT_IN_R** (Bit 1, Byte 0): Indicates Reverse (R) gear is selected.
- **CVT_SHFT_IN_N** (Bit 2, Byte 0): Indicates Neutral (N) gear is selected.
- **CVT_SHFT_IN_D** (Bit 3, Byte 0): Indicates Drive (D) gear is selected.

These signals are mutually exclusive—only one bit should be HIGH at any given time, and they are used by the system to decide the current transmission state of the vehicle.

Figure 1 below illustrates the CVT gear signal layout (byte 0 of CVT_191):



Multiplexer Signal: – No Multiplexer –

	7	6	5	4	3	2	1	0
0	7	6	CVT_SHFT_IN_L 5	CVT_SHFT_IN_S 4	CVT_SHFT_IN_D 3	CVT_SHFT_IN_NEUTRAL 2	CVT_SHFT_IN_REVERSE 1	CVT_SHFT_IN_PARKING 0
1	CVT_GEAR_POSITION_IND_CVT msb 15	14	13	lsb 12	11	10	9	8
2	23	CVT_REFUSE_VSA_CONTROL 22	21	20	19	18	17	16
3	CVT_RATIO_INFO_191 msb 31	30	29	28	27	26	25	lsb 24
4	CVT_RATIO_TARGET msb 39	38	37	36	35	34	33	lsb 32
5	47	46	45	44	43	42	41	40
6	55	54	53	52	51	50	49	48
7	CVT_CAS_ON 63	62	CVT_ALIVE_COUNTER_191 msb 61	lsb 60	CVT_CHECKSUM_191 msb 59	58	57	lsb 56

Hình 1: Bit layout of gear status signals in CVT_191 (byte 0)

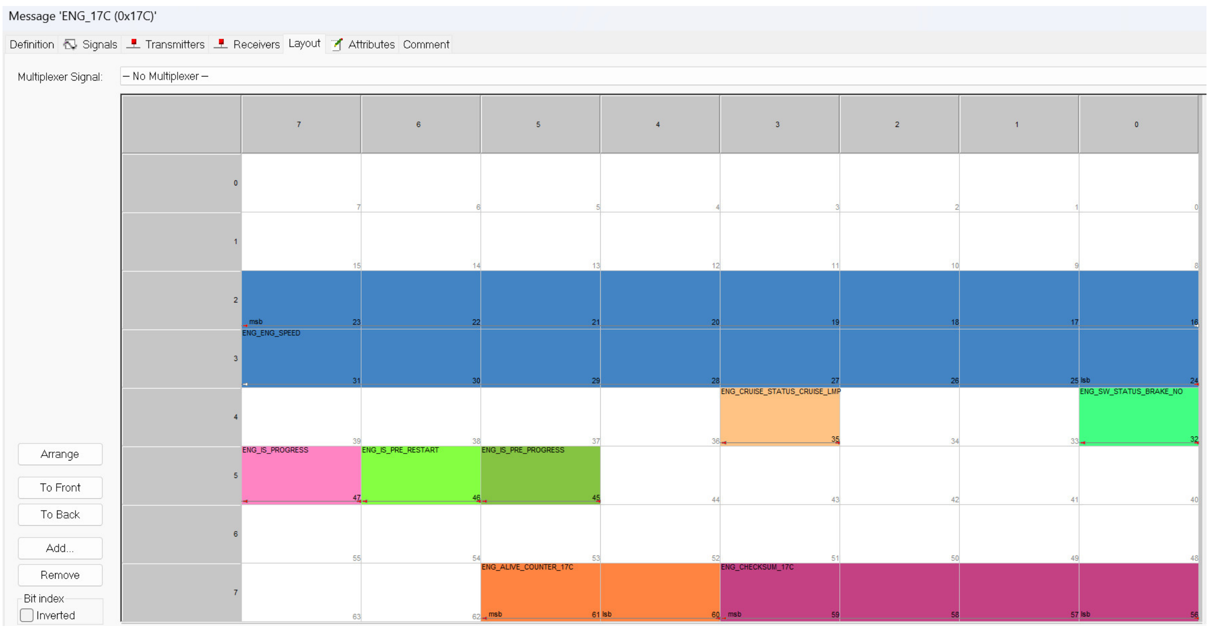
2. Engine Status Signals from ENG_17C (CAN ID 0x380) The engine's ignition state is detected from two specific bits in `byte 5` of the ENG_17C message:

- **ENG_IS_PRE_PROGRESS** (Bit 5, Byte 5): Indicates that the key is in ON position (Pre-Ignition), but engine has not started yet. This corresponds to the driver's intent to start the car.

- **ENG_IS_PROGRESS** (Bit 7, Byte 5): Indicates that the engine is running (Ignition ON).

These signals are used to determine if the driver has begun starting the vehicle, and to check if the engine is currently on or off.

Figure 2 illustrates the position of engine-related status bits in byte 5 of ENG_17C:



Hình 2: Bit layout of engine status in ENG_17C (byte 5)

Usage in Logic These signals are directly mapped in software using either 'bitRead(data[byte], bit)' or 'getBit(data, bitIndex)' (where `bitIndex = byte * 8 + bit`). The logic uses these bits to detect risky states such as:

- Entering Drive gear too early while engine is still in pre-start phase.
- Turning off the engine while not in Parking gear.

This low-level bit interpretation ensures the system behaves identically whether the data is real-time from a car ECU or simulated from a keypad controller.

1.2.4 Summary of Signal Mapping and Usage

This section has detailed the specific CAN signals and their individual bits that the system relies on to determine driver intent and vehicle status. By parsing only a few select bits from two key CAN messages (`CVT_191` and `ENG_17C`), the system can reliably detect the following:

- **Gear State (P, R, N, D)** — extracted from bits 0–3 of byte 0 in `CVT_191`.
- **Engine Status** — determined by bits 5 and 7 of byte 5 in `ENG_17C`, which indicate pre-ignition (key ON) and full ignition (engine ON) states respectively.

These minimal but meaningful signals enable the system to:

- Issue early warnings when unsafe gear shifting occurs during engine start-up.
- Alert the user when the vehicle is turned off without shifting to Parking.

Importantly, this bit-level signal approach ensures compatibility with both real vehicle ECUs and simulated message injection (e.g., from Arduino-based test modules), maintaining system accuracy in both testing and real-world deployment scenarios.

1.3 Algorithm Development

The safety monitoring system is designed to assist the new driver in two critical situations:

- When starting the vehicle, the driver shifts into Drive (D) too early without pressing the brake — this can be dangerous.
- When parking and turning off the engine, the driver forgets to shift into Park (P), increasing the risk of the vehicle rolling away.

Based on signals collected via the CAN bus (speed, engine status, gear lever position), the system analyzes the car's state, provides intuitive warnings and instructions to ensure user safety.

1.3.1 Flowchart Design

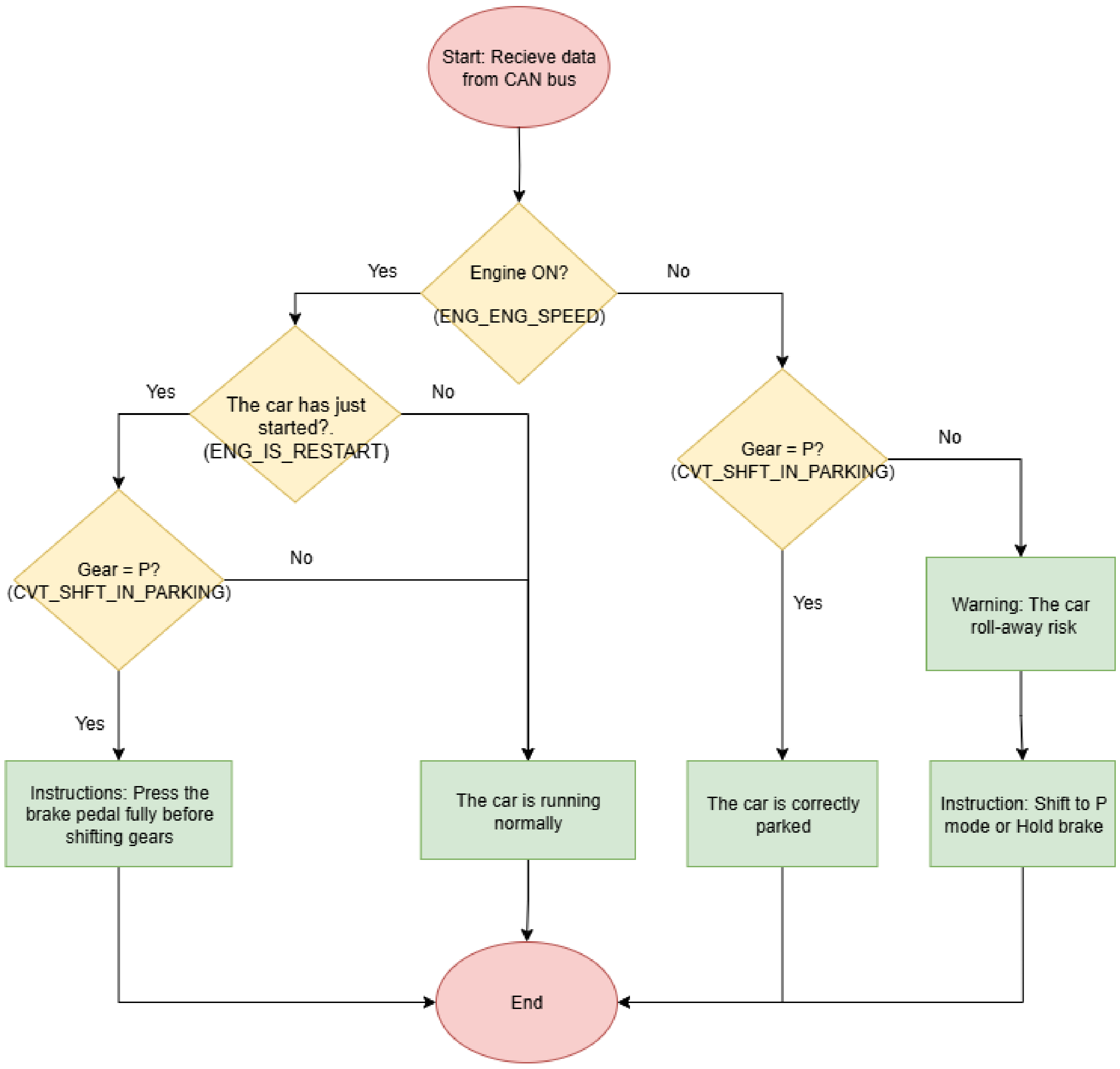

Start: Recieve data from CAN bus
Engine ON? (ENG_ENG_SPEED)
Yes
No
The car has just started?. (ENG_IS_RESTART)
Yes
No
Gear = P? (CVT_SHFT_IN_PARKING)
No
Yes
Gear = P? (CVT_SHFT_IN_PARKING)
No
Yes
Warning: The car roll-away risk
Instructions: Press the brake pedal fully before shifting gears
The car is running normally
The car is correctly parked
Instruction: Shift to P mode or Hold brake
End

Figure 3: Flowchart of System monitoring startup and parking for new drivers

1.3.2 Logic Design

1. **System Start**

   The system receives real-time CAN bus data every **0.5 seconds**, including:

   - Engine Status
   - Gear Lever Position
   - etc.

2. **Engine State Check**
   The system determines whether the engine is currently ON or OFF:
   - **If the engine is ON:**
     - The system checks if the car is in the startup phase (just turned on).
   - **If the engine is OFF:**
     - The system checks the gear position to verify whether the driver has properly shifted to Park (`P`).
3. **Startup Instructions**
   When the engine is just turned on, the system will check the Gear:
   - **If the gear is in Park (`P`):**
     - A **startup instruction** is displayed to notify the driver.
   - **If the gear is not in Park (`P`):**
     - No further action is required. the driver is actively in control of the situation
4. **Stop & Gear Position Check**
   When the engine is OFF, the system checks gear position:
   - **If the gear is in Park (`P`):**
     - No further action is required. The vehicle is safely parked.
   - **If the gear is not in Park (`P`):**
     - A **parking warning and instruction** is displayed to notify the driver.
5. **Algorithm Termination**
   After all conditions have been evaluated and appropriate actions have been executed, the algorithm terminates..

Displayed Warnings:

- **Startup Instruction:** `"Press the brake pedal fully before shifting gears."`
- **Parking Warning:** `"The car roll-away risk."`
- **Parking Instruction:** `"Shift to P mode or Hold Brake"`

2 Implementation and Testing

2.1 Simulation Node (Arduino Nano)

This part of the system emulates CAN messages for gear and engine states using an Arduino Nano. Acting as a virtual ECU, it sends gear signals based on keypad input and toggles engine status flags according to predefined logic. The simulation replicates the layout of actual vehicle messages defined in the provided DBC file. The simulation node emulates real ECU messages on the CAN bus using an **Arduino Nano**, functioning as a virtual transmitter of gear and engine states. It allows controlled testing without the need for a real vehicle ECU.

2.1.1 Functionality Overview

- **Input Interface:** A 4x4 matrix keypad is used to trigger specific signal combinations. Keys `'0'`–`'3'` simulate gear inputs (P, R, N, D), while `'4'` and `'5'` toggle engine status signals.
- **CAN Transmission:** The Arduino Nano sends CAN messages conforming to the layout of `CVT_191` and `ENG_17C` messages, extracted from the DBC file provided by the organizers.
- **Hardware Setup:** The Nano communicates with the CAN bus via an **MCP2515 module** over SPI.

2.1.2 Message Simulation Logic

1. Gear Status Message – `CVT_191 (ID: 0x401)`:
Gear position is encoded in bits 0 to 3 of `data[0]`:

- Bit 0: `CVT_SHFT_IN_P`
- Bit 1: `CVT_SHFT_IN_R`
- Bit 2: `CVT_SHFT_IN_N`
- Bit 3: `CVT_SHFT_IN_D`

```
cvtMsg.data[0] |= (bitState[0] << 0);  // P
cvtMsg.data[0] |= (bitState[1] << 1);  // R
cvtMsg.data[0] |= (bitState[2] << 2);  // N
cvtMsg.data[0] |= (bitState[3] << 3);  // D
```

Listing 1: Setting Gear Bits in CVT_191 Message

2. Engine Status Message – `ENG_17C (ID: 0x380)`:
Two boolean engine status signals are encoded within `data[5]`:

- Bit 5 (bit 45): `ENG_IS_PRE_PROGRESS`
- Bit 7 (bit 47): `ENG_IS_PROGRESS`

```
if (bitState[4]) engMsg.data[5] |= (1 << 5); // Pre-progress
if (bitState[5]) engMsg.data[5] |= (1 << 7); // Is-progress
```

Listing 2: Setting Engine Status Bits in ENG_17C Message

Summary of Roles

This simulation node plays a vital role in system validation by:

- Emulating valid vehicle state transitions (e.g., shift to `D`, ignition on).
- Testing how the main receiver node responds to improper or unsafe sequences.
- Allowing controlled demonstration and debugging without needing access to a physical vehicle.

2.2 Main MCU Logic (Arduino Mega Receiver)

The central controller of the system is an **Arduino Mega**, which acts as the receiver and decision-making unit. It listens to two specific CAN messages representing gear status and engine status, processes them, and gives real-time warnings or confirmations on an OLED screen to support safe driving behavior.

2.2.1 Key Functions

- **CAN Bus Monitoring**: Receives messages `CVT_191` (ID: `0x401`) and `ENG_17C` (ID: `0x380`) using an MCP2515 CAN module.
- **Signal Extraction**: Decodes relevant bits representing the gear position and engine states.
- **User Feedback**: Displays helpful and intuitive text warnings on a SH1106 OLED based on real-time conditions.

2.2.2 Signal Decoding

The following bits are extracted from incoming CAN frames:

- **CVT_191 (0x401)**:
  - Bit 0: Gear in `P` (Parking)
  - Bit 1: Gear in `R` (Reverse)
  - Bit 2: Gear in `N` (Neutral)
  - Bit 3: Gear in `D` (Drive)
- **ENG_17C (0x380)**:
  - Bit 45: `ENG_IS_PRE_PROGRESS` (Key ON, before crank)
  - Bit 47: `ENG_IS_PROGRESS` (Engine running)

These bits are accessed through a utility function:

```
bool getBit(uint8_t* data, int bitIndex) {
  int byteIndex = bitIndex / 8;
  int bitOffset = bitIndex % 8;
  return (data[byteIndex] >> bitOffset) & 0x01;
}
```

Listing 3: Read Specific Bit From CAN Frame

2.2.3 OLED Display Logic

The OLED screen always shows current gear and engine status, and dynamically updates warning messages based on unsafe conditions.

Startup Warning (Case 1):

When engine is running *(ENG_IS_PROGRESS = 1)* and ignition is still in pre-crank mode *(ENG_IS_PRE_PROGRESS = 1)*, and gear is in `P`, the system alerts the user to press the brake fully before shifting:

```
if (engineProgress && enginePreProgress && gearState == 0) {
  displayMessage1 = ">>> Press brake fully";
}
```

Parking Reminder (Case 2):

If the engine is OFF *(ENG_IS_PROGRESS = 0)* but gear is not in `P`, a rollback warning is shown:

```
if (!engineProgress && gearState != 0) {
  displayMessage1 = "[!] Warning : Roll";
  displayMessage2 = ">>> Shift to P or";
  displayMessage3 = ">>> Hold Brake";
}
```

Warnings are automatically cleared after 3 seconds of stable state:

```
if (now - lastDisplayTime > 3000 && displayMessage1 != "") {
  displayMessage1 = ""; displayMessage2 = ""; displayMessage3 = "";
}
```

2.2.4 Startup OLED Greeting

At boot time, the OLED shows a welcoming screen using a large font to indicate the system is live:

```
u8g.setFont(u8g_font_fub30r);
u8g.drawStr(16, 48, "Hello");
```

2.2.5 Summary

This main controller continuously monitors real-time driving context and gear operations. With only two CAN messages and simple embedded logic, it effectively:

- Detects incorrect actions during engine start-up or parking.
- Prevents accidental roll-away or gear misuse.
- Improves driver awareness with contextual visual feedback.

Its minimal hardware and clearly defined CAN signal mapping offer a practical, replicable solution for driver assistance systems targeting entry-level or infrequent drivers.

2.3 System Testing and Validation

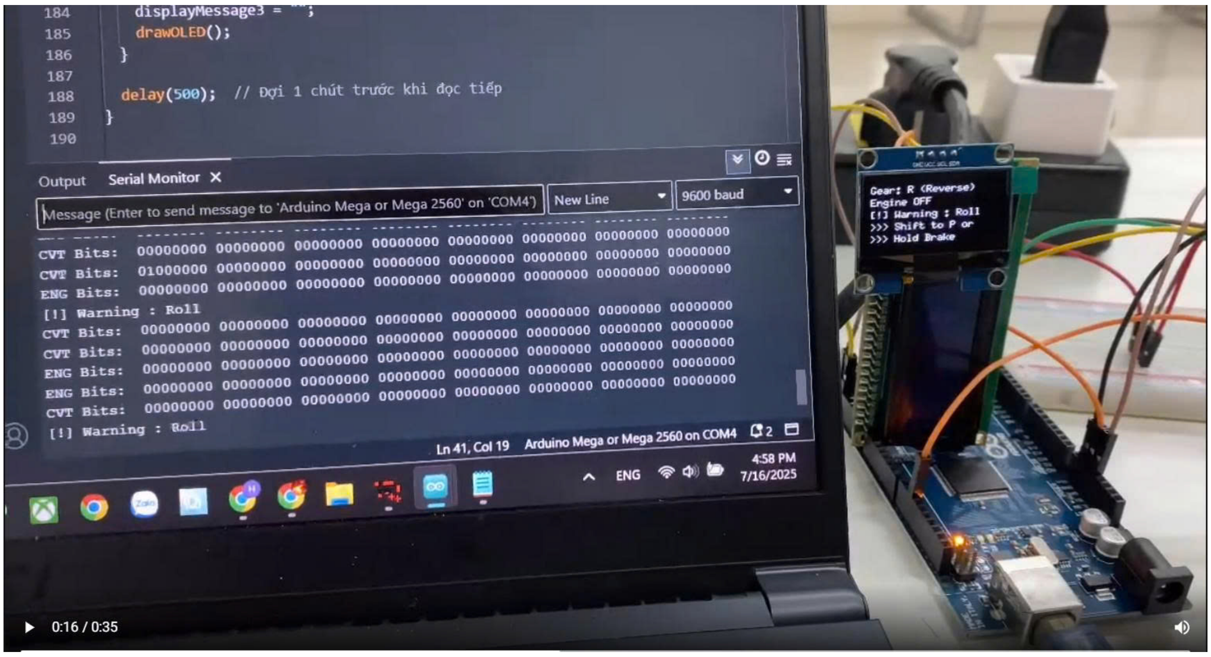
To ensure the system behaves as intended across various driving scenarios, a series of targeted test cases were conducted. These tests validate the receiver's logic, confirming whether appropriate visual warnings are displayed on the OLED screen and ensuring safe driving prompts are triggered only when necessary. Each test simulates a realistic driving context using the simulation node and observes the system's real-time response.

2.3.1 Test Case 1: Parking Without Shifting to `P`

- **Description:**
  After parking the vehicle, the driver turns off the engine but forgets to shift the gear back to `P`. The gear remains in `R`, which poses a rollback risk.
- **Condition:**
  - `engineProgress = 0`
  - `gearState = 1` (Reverse)
- **Expected Result:**
  The OLED displays a rollback warning, accompanied by the message:

```
Warning: Roll
Shift to P or
 Hold Brake
```

- **Validation Result:**
  - System correctly identifies the unsafe gear state post-engine-off and provides appropriate visual cues to prevent unintended vehicle movement.
  - A demonstration of the system behavior for this test case can be viewed here: `Test Case 1 - Parking Without Shifting to P`



```
    displayMessage3 = "";
    drawOLED();
  }

  delay(500);  // Đợi 1 chút trước khi đọc tiếp
}
```

Output Serial Monitor

Message (Enter to send message to 'Arduino Mega or Mega 2560' on 'COM4') New Line 9600 baud

```
CVT Bits:  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
CVT Bits:  01000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
ENG Bits:  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[!] Warning : Roll
CVT Bits:  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
CVT Bits:  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
ENG Bits:  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
ENG Bits:  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
CVT Bits:  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[!] Warning : Roll
```

Ln 41, Col 19 Arduino Mega or Mega 2560 on COM4

```
Gear: R (Reverse)
Engine OFF
[!] Warning : Roll
>>> Shift to P or
>>> Hold Brake
```

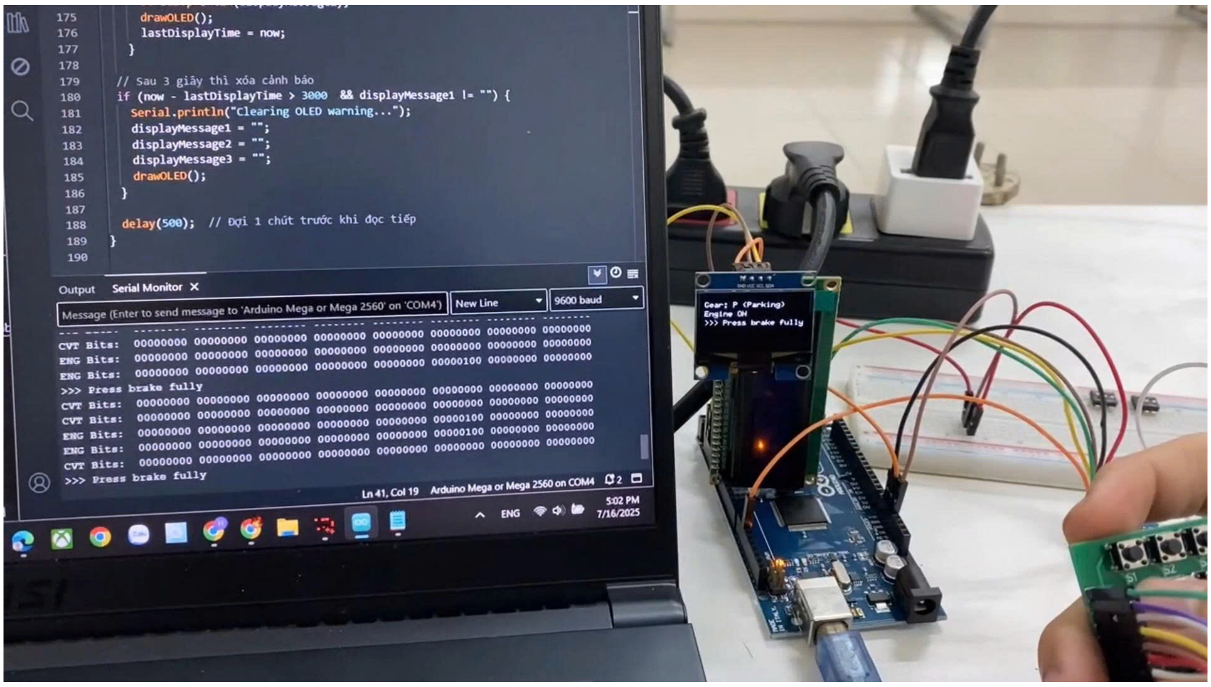
Figure 4: Validation Result – Test case 1

2.3.2 Test Case 2: Engine Start in Park

- **Description:**
  The driver starts the car with the gear in `P`. The system detects that the engine is on and still in the pre-progress phase.
- **Condition:**
  - `engineProgress = 1` (Engine Running)
  - `enginePreProgress = 1` (Ignition just turned on)
  - `gearState = 0` (Park)
- **Expected Result:**
  A reminder is shown on the OLED prompting the user to fully press the brake before attempting to shift out of `P`:

  ```
  Press brake fully
  ```
- **Validation Result:**

- System correctly reminds the user to apply the brake before gear transition, supporting safe startup behavior.
- A demonstration of the system behavior for this test case can be viewed here: `Test Case 2 - Engine Start in Park`



```
    drawOLED();
    lastDisplayTime = now;
  }

 // Sau 3 giây thì xóa cảnh báo
 if (now - lastDisplayTime > 3000  && displayMessage1 != "") {
   Serial.println("Clearing OLED warning...");
   displayMessage1 = "";
   displayMessage2 = "";
   displayMessage3 = "";
   drawOLED();
 }

 delay(500);  // Đợi 1 chút trước khi đọc tiếp
}
```

```
CVT Bits:  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
ENG Bits:  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
ENG Bits:  00000000 00000000 00000000 00000000 00000000 00000100 00000000 00000000
>>> Press brake fully
CVT Bits:  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
CVT Bits:  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
ENG Bits:  00000000 00000000 00000000 00000000 00000000 00000100 00000000 00000000
ENG Bits:  00000000 00000000 00000000 00000000 00000000 00000100 00000000 00000000
CVT Bits:  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
>>> Press brake fully
```

```
Gear: P (Parking)
Engine ON
>>> Press brake fully
```

Output
Serial Monitor
Message (Enter to send message to 'Arduino Mega or Mega 2560' on 'COM4')
New Line
9600 baud
Ln 41, Col 19
Arduino Mega or Mega 2560 on COM4
ENG
5:02 PM
7/16/2025

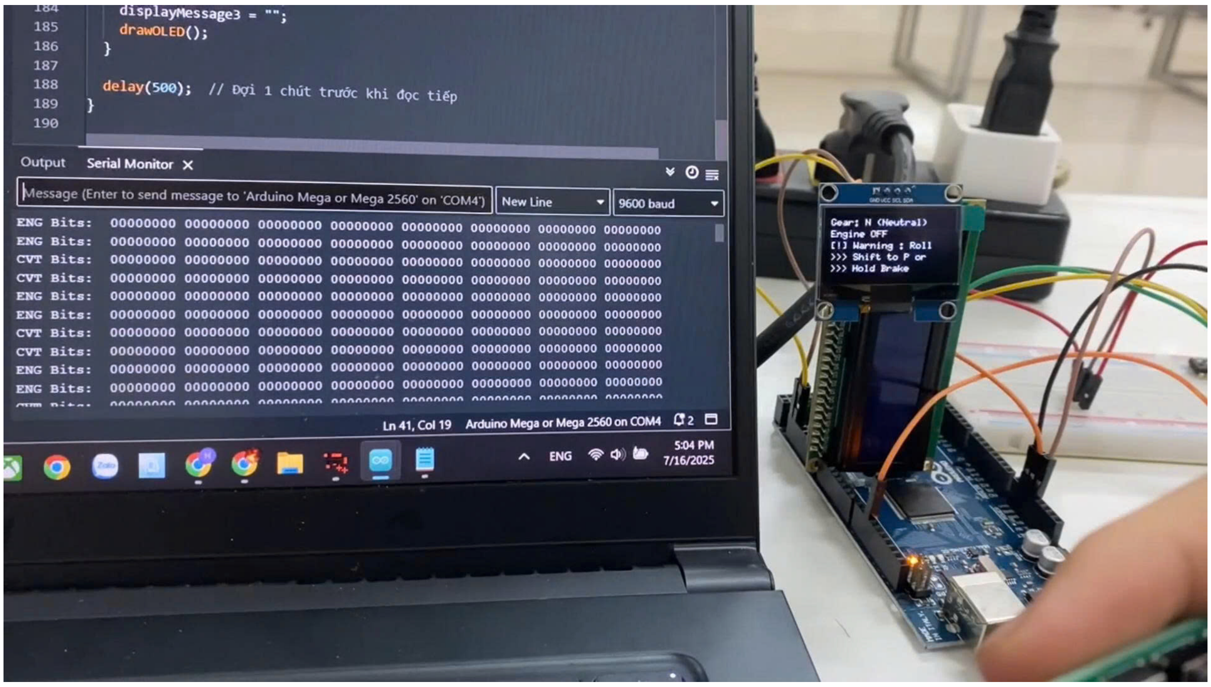
Figure 5: Validation Result – Test case 2

2.3.3 Test Case 3: Sudden Engine Shutdown While in `N`

- **Description:**
  While stopped at a red light, the driver switches to neutral, and the engine unexpectedly shuts off (e.g., due to stalling).
- **Condition:**
  - `engineProgress = 0` (Engine Off)
  - `gearState = 2` (Neutral)
- **Expected Result:**
  OLED shows the rollback warning due to unsafe gear position:

```
Warning: Roll
Shift to P or
 Hold Brake
```

- **Validation Result:**
  - The system interprets the condition as unsafe and alerts the driver, helping prevent uncontrolled rolling.
  - A demonstration of the system behavior for this test case can be viewed here: `Test Case 3 - Sudden Engine Shutdown While in N`



```
      displayMessage3 = "";
      drawOLED();
    }

  delay(500);  // Đợi 1 chút trước khi đọc tiếp
}
```

```
ENG Bits:  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
ENG Bits:  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
CVT Bits:  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
CVT Bits:  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
ENG Bits:  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
ENG Bits:  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
CVT Bits:  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
CVT Bits:  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
ENG Bits:  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
ENG Bits:  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

```
Gear: N (Neutral)
Engine OFF
[!] Warning : Roll
>>> Shift to P or
>>> Hold Brake
```

Output
Serial Monitor
Message (Enter to send message to 'Arduino Mega or Mega 2560' on 'COM4')
New Line
9600 baud
Ln 41, Col 19
Arduino Mega or Mega 2560 on COM4
ENG
5:04 PM
7/16/2025
GND VCC SCL SDA

Figure 6: Validation Result – Test case 3

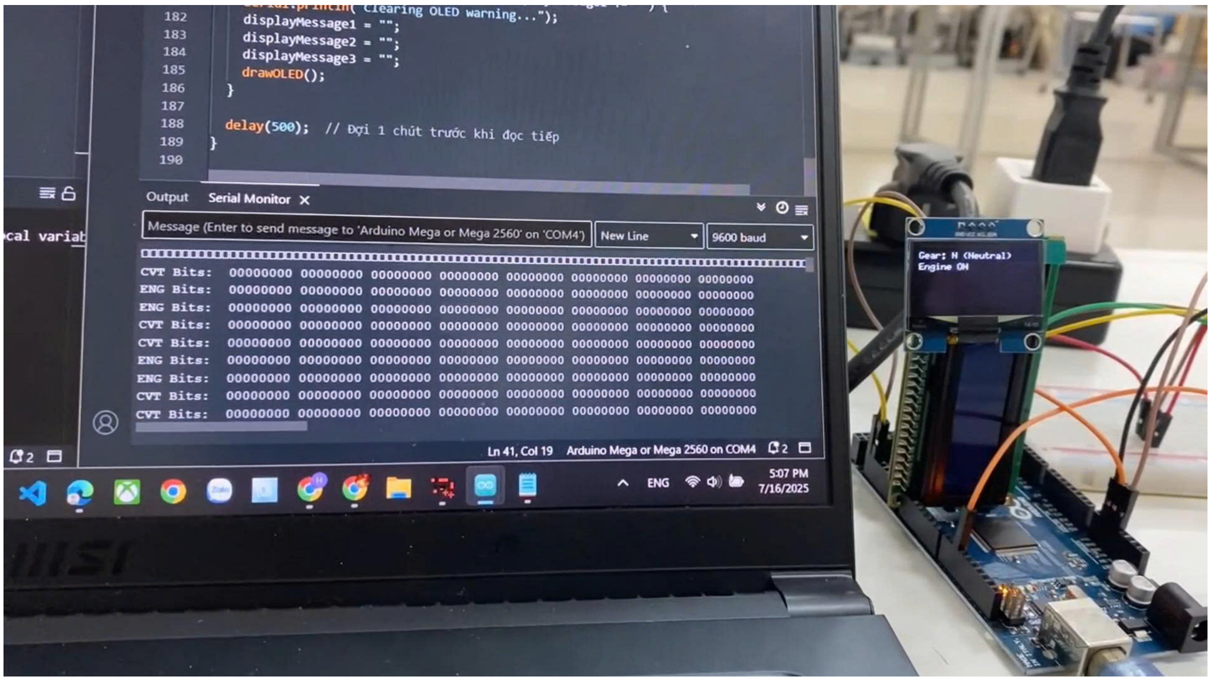
2.3.4 Test Case 4: Re-Start in Neutral After Stall

- **Description:**
  Following an unexpected stall, the driver restarts the vehicle with the gear still in neutral or reverse.

- **Condition:**
  - `engineProgress = 1` (Engine Running)

  - `enginePreProgress = 1` (Just restarted)
  - `gearState = 2` (Neutral)

- **Expected Result:**
  No warning needed, as the driver is actively handling the restart process.
  - The system appropriately suppresses warnings, allowing for uninterrupted operation during intentional restarts.
  - A demonstration of the system behavior for this test case can be viewed here: `Test Case 4 - Re-Start in Neutral After Stal`



```
        displayMessage1 = "";
        displayMessage2 = "";
        displayMessage3 = "";
        drawOLED();
    }

    delay(500);  // Đợi 1 chút trước khi đọc tiếp
}
```

Message (Enter to send message to 'Arduino Mega or Mega 2560' on 'COM4') · New Line · 9600 baud

```
CVT Bits:  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
ENG Bits:  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
ENG Bits:  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
CVT Bits:  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
CVT Bits:  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
ENG Bits:  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
ENG Bits:  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
CVT Bits:  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
CVT Bits:  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

```
Gear: N (Neutral)
Engine ON
```

Figure 7: Validation Result – Test case 4

2.3.5 Test Case 5: Normal Driving Scenario

- **Description:**
  Vehicle is operating normally with the engine on and gear in `D`.

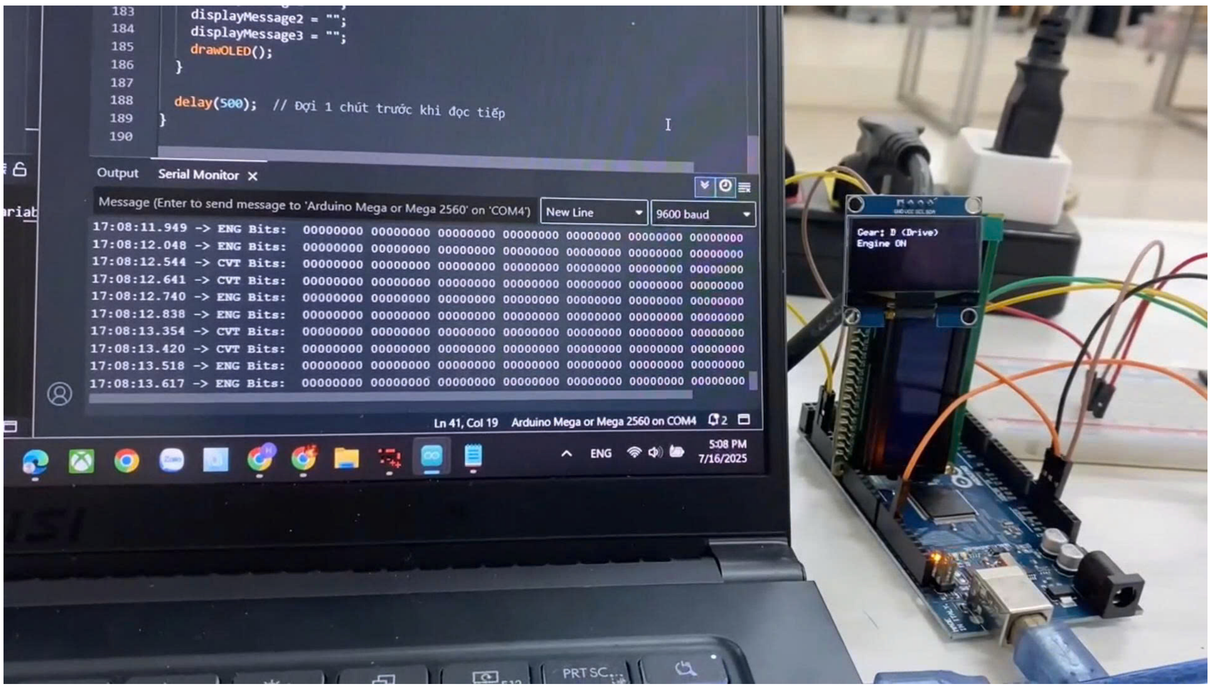
- **Condition:**
  - `engineProgress = 1` (Engine Running)
  - `gearState = 3` (Drive)

- `enginePreProgress = 0`

- **Expected Result:**

  No warnings, only regular gear and engine status shown on OLED.

  - The system correctly identifies a normal state and refrains from unnecessary alerts.
  - A demonstration of the system behavior for this test case can be viewed here: `Test Case 5 - Normal Driving Scenario`



```
        displayMessage2 = "";
        displayMessage3 = "";
        drawOLED();
    }

    delay(500);  // Đợi 1 chút trước khi đọc tiếp
}
```

```
17:08:11.949 -> ENG Bits:  00000000 00000000 00000000 00000000 00000000 00000000 00000000
17:08:12.048 -> ENG Bits:  00000000 00000000 00000000 00000000 00000000 00000000 00000000
17:08:12.544 -> CVT Bits:  00000000 00000000 00000000 00000000 00000000 00000000 00000000
17:08:12.641 -> CVT Bits:  00000000 00000000 00000000 00000000 00000000 00000000 00000000
17:08:12.740 -> ENG Bits:  00000000 00000000 00000000 00000000 00000000 00000000 00000000
17:08:12.838 -> ENG Bits:  00000000 00000000 00000000 00000000 00000000 00000000 00000000
17:08:13.354 -> CVT Bits:  00000000 00000000 00000000 00000000 00000000 00000000 00000000
17:08:13.420 -> CVT Bits:  00000000 00000000 00000000 00000000 00000000 00000000 00000000
17:08:13.518 -> ENG Bits:  00000000 00000000 00000000 00000000 00000000 00000000 00000000
17:08:13.617 -> ENG Bits:  00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

Output
Serial Monitor
Message (Enter to send message to 'Arduino Mega or Mega 2560' on 'COM4')
New Line
9600 baud
Ln 41, Col 19
Arduino Mega or Mega 2560 on COM4
ENG
5:08 PM
7/16/2025
Gear: D (Drive)
Engine ON

Figure 8: Validation Result – Test case 5

Summary

Through systematic testing, the main MCU logic has been validated against both routine and edge-case driving behaviors. The warning logic accurately distinguishes between critical and non-critical states, ensuring driver feedback is contextually relevant. These results affirm the reliability of the system in assisting drivers with safe startup and parking operations .

3 Innovation

This project introduces a compact, low-cost driver assistance system designed specifically to support startup and parking procedures—critical phases where inexperienced drivers are most prone to errors.

3.1 Simple Yet Effective Signal Usage

The system relies solely on standard CAN signals:

- `gearState` (from CAN ID 0x401): Identifies current gear (P, R, N, D).
- `enginePreProcess` and `engineProcess` (from CAN ID 0x380): Monitor engine states OFF/ON.

3.2 Real-Time Behavior Monitoring

By analyzing the timing and sequence of gear and engine signals, the system can:

- Detect unsafe actions.
- Identify incorrect parking steps.
- Provide immediate feedback via an OLED display, helping new drivers form safe habits.

3.3 Low-Cost and Easy Integration

The system uses an Arduino-compatible microcontroller, MCP2515 CAN module, and SH1106 OLED screen, ensuring minimal cost and straightforward deployment in personal vehicles, training cars, or fleet applications.

3.4 Potential Future Enhancements

While the current system is simple and effective, several future improvements could enhance its capabilities:

- **Personalized Behavior Analysis**: Learn driver habits over time to provide tailored feedback and reduce false warnings.

- **Training Support**: Record incorrect startup/parking sequences for review by instructors or learners; useful for driving schools.
- **Optional Sensor Add-ons**: Integrate basic sensors like brake switch or door status for improved accuracy, while keeping the core system lightweight.

References

[1] Bosch Mobility Solutions, "CAN Specification 2.0," [Online]. Available: https://dewesoft.com/blog/what-is-can-bus. [Accessed: 29-Jun-2025].

[2] Dewesoft, "What is CAN bus?", 2024. [Online]. Available: https://www.dewesoft.com/daq/what-is-can-bus. [Accessed: 29-Jun-2025].

[3] Visual Paradigm, “Flowchart Tutorial (with Symbols, Guide and Examples),” [Online]. Available: https://www.visual-paradigm.com/tutorials/flowchart-tutorial/. [Accessed: 29-Jun-2025].

[4] Vector Informatik GmbH, “Products A–Z,” Vector, [Online]. Available: https://www.vector.com/int/en/products/products-a-z/software/. [Accessed: July 17, 2025].

[5] Arduino, “Arduino Documentation,” [Online]. Available: https://docs.arduino.cc/. [Accessed: July 17, 2025].

[6] Fowler, C. J. (n.d.). *MCP_CAN Arduino Library*. https://github.com/coryjfowler/MCP_CAN_lib

[7] Texas Instruments, “Introduction to the Controller Area Network (CAN),” 2015. [Online]. Available: https://www.ti.com/lit/an/sloa101a/sloa101a.pdf