Contents

Introduction

The Token Ring Network is a network where all processors, in our case ECUs, are connected in a LAN, in our case using a CAN Bus. Its most distinctive feature is that each ECU takes its turn in broadcasting to the network. In this project the token is represented as a WHITE LED light and the broadcasted message is a press on one of two switches.

Another application in this project is an ADAS (Advanced Driver Assistance Systems) Simulator in which only an ultrasonic sensor operation is implemented.

The system is implemented as 2 subsystems: Master ECU (Gives token) and ADAS ECUs (Take token and are used to implement the ADAS System).

Create CAN network from 3 ECU (Node ID 1, 2 and 3). Each ECU Has the following functional requirements:

1- Token Ring.
2- ADAS simulation.

1- Token Ring

- A ring token should be rotated along the network nodes.
- Each token should hold the token 1 second before transmitting to the next ECU.
- The ECU which has the token should turn on the white LED as long as it holds the token.
- At initial state each ECU is associated with a specific color to be turned on as the following:
  Node 1 -> RED , Node 2 -> GREEN and Node 3 -> BLUE
- The token LED will overwrite the ECU specific color.
- If SW1 of the token owner ECU is pressed. Then the color sequence will be changed and started from this ECU. For instance, after the initial state declared above, if SW1 of the node 3 is pressed while it is holding the token (white LED is on), the color sequence will be
  Node 3 -> RED, Node 1 -> GREEN and Node 2 -> BLUE
- If SW2 of the token owner ECU is pressed, then the color sequence will be changed and started from this ECU with reversed order. For instance, after the initial state declared above if SW2 of the node 2 is pressed while it is holding the token (white LED is on), the color sequence will be
  Node 2-> BLUE, Node 3 -> RED and Node 1 -> GREEN

2- ADAS Simulator

- System has Node2 and Node3 which acting as car ADAS simulator (controlling car based on car and obstacle distance).
  Node2 role is:
  - Act as Sensor ECU.
  - Responsible for receiving the distance sensor read though UART on Duty cycle.
  - Send Distance over CAN every 100 ms. (The latest received).
  - Convert the Duty cycle to Distance based on the following table:

Duty Cycle	Distance
0% to 10%	Sensor Error
11% to 20%	1 meter
21% to 40%	2 meter
41% to 60%	3 meter
61% to 100%	No Obstacle

- Node3 role is:
  - Act as Car Control ECU.
  - Responsible of receiving the distance from CAN bus and generate the command.
  - CMD Shall display on the PC Terminal (UART Terminal)

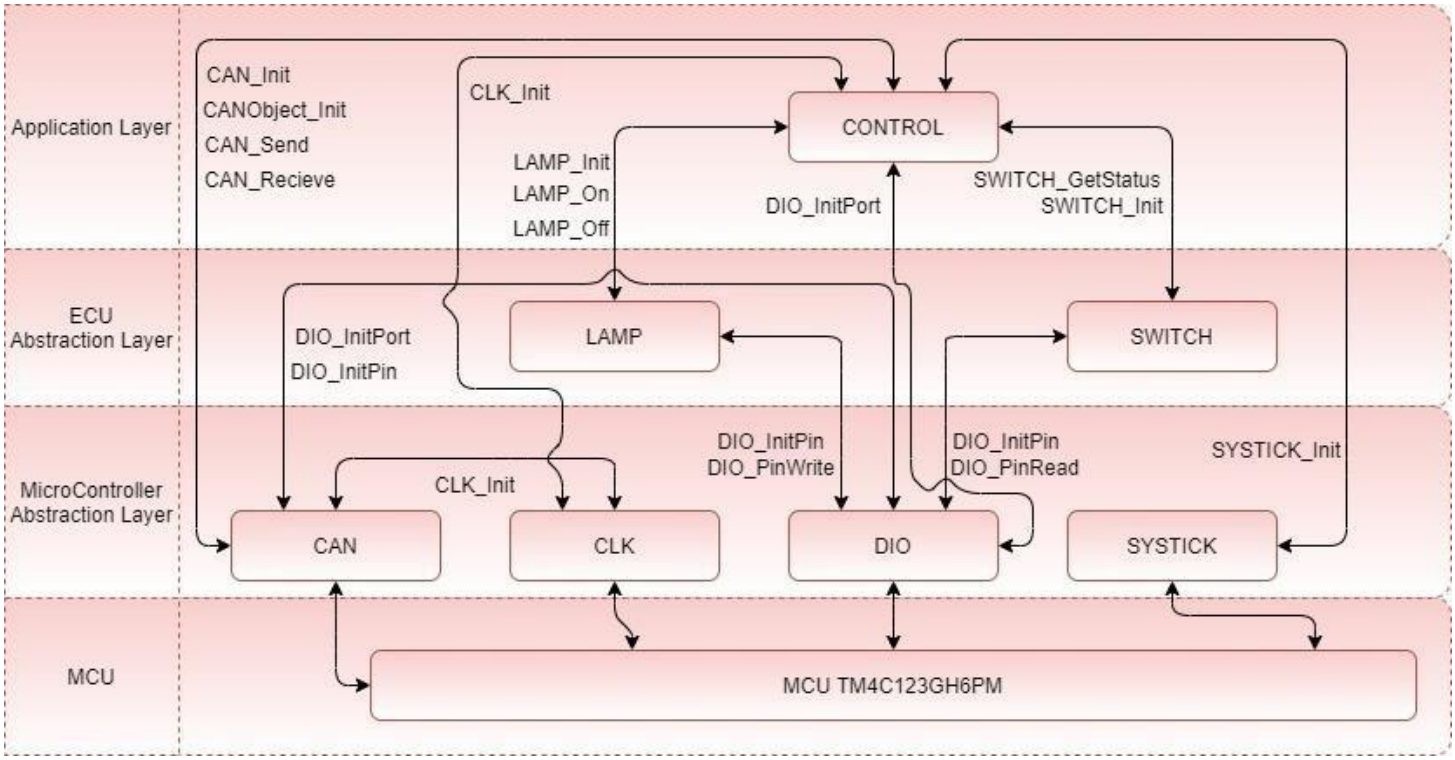
• CMD shall be based on the following table.

Distance	CMD
Sensor Error	CAR STOP
1 meter	CAR TURN
2 meter	Move fwd and reduce the speed
3 meter	Move fwd
No Obstacle	Move fwd and increase the speed

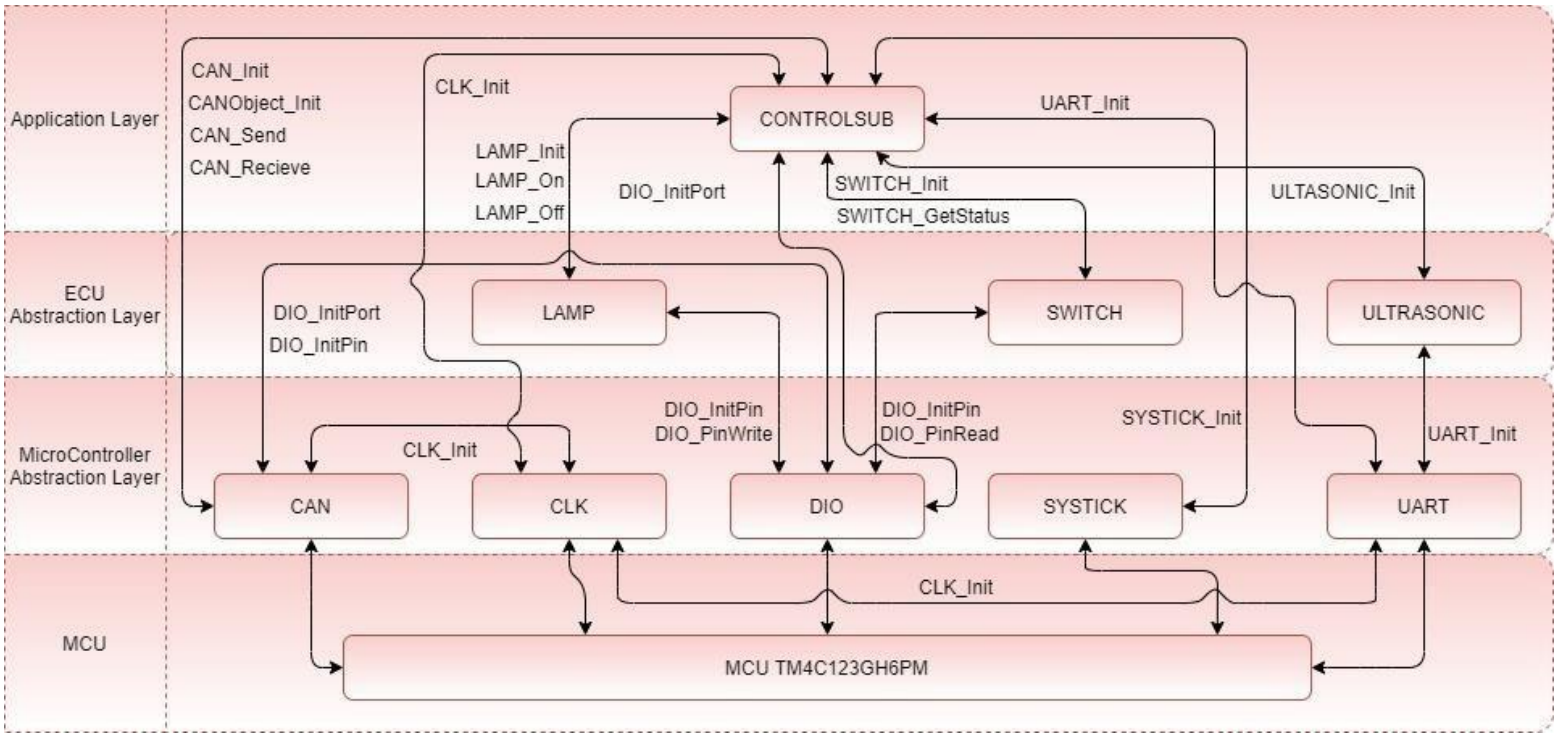
For the upcoming components to be implemented a header file, **COMMON.h** must be included.

Static System Architecture

General Architecture



Subordinate ECU Architecture



MCU Abstraction Layer Components

CLK

This is module is concerned with operating with the clocking system of the MCU, in this case a TM4C123GH6PM.

APIs

- E_Status CLK_Init (uint8_t *port*)

E_Status CLK_Init (uint8_t *port*)

Description: This function is used to initialise and start all clock functions for a certain port.

Input: The variable *port* is the port number to open the clock for and it ranges from the values PORTA to PORTF, case sensitive.

Return: **E_OK** if initialisation is successful. **E_NOK** if unsuccessful or wrongful inputs.

DIO

This is module is concerned with the functionalities of all Digital Input & Output (DIO) peripheral of the MCU, in this case a TM4C123GH6PM.

APIs

- E_Status DIO_InitPort (uint8_t *port*)
- E_Status DIO_InitPin (uint8_t *port*, uint8_t *pin*, uint8_t *mode*)

- E_Status DIO_PinRead (uint8_t *port*, uint8_t *pin*, uint8_t* *data*)
- E_Status DIO_PinWrite (uint8_t *port*, uint8_t *pin*, uint8_t *data*)

E_Status DIO_InitPort (uint8_t port)

Description: This function is used to initialise and unlock the specified port.

Input: The variable *port* is the port number to open the clock for and it ranges from the values **PORTA** to **PORTF**, case sensitive.

Return: **E_OK** if operation is successful. **E_NOK** if unsuccessful or wrongful inputs.

E_Status DIO_InitPin (uint8_t *port*, uint8_t pin, uint8_t *mode*)

Description: This function is used to assign specific settings to pins used for a certain port.

Input: The variable *port* is the port number and it ranges from the values PORTA to PORTF, case sensitive. The variable *pin* is the pin number the user wants to assign certain settings to, and it ranges from **PIN0** to **PIN7**, case sensitive. The variable mode is the logical OR of any of the following modes that are also case sensitive: **OUTPUT**, **INPUT**, **INPUT_PULLUP**, **INPUT_PULLDOWN**, **INPUT_CAN**, **OUTPUT_CAN**.

Return: **E_OK** if the operation is successful. **E_NOK** if unsuccessful or wrongful inputs.

E_Status DIO_PinRead (uint8_t *port*, uint8_t *pin*, uint8_t* *data*)

Description: This function is used to read a value off a pin on a certain port into a variable of the user's choice.

Input: The variable *port* is the port number and it ranges from the values PORTA to PORTF, case sensitive. The variable *pin* is the pin number the user wants to assign certain settings to, and it ranges from **PIN0** to **PIN7**, case sensitive. The variable *data* is a pointer to a variable of type **uint8_t** for the data to be stored and it either has the value **HIGH** or **LOW**.

Return: **E_OK** if the operation is successful. **E_NOK** if unsuccessful or wrongful inputs.

E_Status DIO_PinWrite (uint8_t port, uint8_t pin, uint8_t data)

Description: This function is used to write a value onto a pin on a certain port from a variable of the user's choice.

Input: The variable *port* is the port number and it ranges from the values PORTA to PORTF, case sensitive. The variable *pin* is the pin number the user wants to assign certain settings to, and it ranges from **PIN0** to **PIN7**, case sensitive. The variable *data* is a variable of type **uint8_t** for the data to be written and it either has the value **HIGH** or **LOW**.

Return: **E_OK** if initialisation is successful. **E_NOK** if unsuccessful or wrongful inputs.

CAN

This is module is concerned with operating the CAN functionalities of the MCU, in this case a TM4C123GH6PM.

APIs

- E_Status CAN_Init (uint8_t *channel*)
- E_Status CANObject_Init (tCANMsgObject * *CANObj*, uint8_t * *msgData*, uint8_t *MODE*, uint32_t *MSG_ID*, uint32_t *OBJ_ID*)
- E_Status CAN_Send (tCANMsgObject *CANObj*, uint32_t *OBJ_ID*)
- E_Status CAN_Recieve (tCANMsgObject *CANObj*, uint32_t *OBJ_ID*)

E_Status CAN_Init (uint8_t *channel*)

Description: This function is used to initialise the CAN module bound to a certain port.

Input: The variable *channel* is used to determine which hardware port to initialise the CAN protocol on and it ranges from **CAN_CH0** to **CAN_CH4.**

But in our case we are only using CAN_CH0 as our Secondary Hardware (Can Transceiver Permits).

Return: **E_OK** if initialisation is successful. **E_NOK** if unsuccessful or wrongful inputs.

E_Status CANObject_Init (tCANMsgObject * *CANObj*, uint8_t * *msgData*, uint8_t *MODE*, uint32_t *MSG_ID*, uint32_t *OBJ_ID*)

Description: This function is used to initialise a CAN object to be used by any application.
Input: The variable *CANObj* is the user defined pointer to CAN object of type **tCANMsgObject**. The variable *msgData* is a user defined pointer of type **uint8_t** to be used to carry messages in the CAN operations. *MODE* determines whether the object will be used to receive data into or transmit date from and it takes one of two values, **CAN_TX** or **CAN_RX**. *MSG_ID* is the (11-29 bit) unique message identifier.*OBJ_ID is the ID for a specific Can Object used for Interrupt Handling & Message Transmission/Reception and ranges from 0-32.*

Return: **E_OK** if the operation is successful. **E_NOK** if unsuccessful or wrongful inputs.

E_Status CAN_Send (tCANMsgObject *CANObj*, uint32_t *OBJ_ID*)

Description: This function is used to send the data segment of a CAN object into the network.

Input: The variables are predefined**.**

Return: **E_OK** if the operation is successful. **E_NOK** if unsuccessful or wrongful inputs.

E_Status CAN_Recieve (tCANMsgObject *CANObj*, uint32_t *OBJ_ID*))

Description: This function is used to receive into a data segment of a CAN object from the network.

Input: The variables are predefined**.**

Return: **E_OK** if the operation is successful. **E_NOK** if unsuccessful or wrongful inputs.

SYSTICK

This is module is concerned with operating the SysTick timer functionalities of the MCU & acts as a Task Scheduler, in this case a TM4C123GH6PM.

APIs

- E_Status SYSTICK_Init (uint32_t *value*)

E_Status SYSTICK_Init (uint32_t *value*)

Description: This function is used to initialise SysTick functionalities of the MCU.

Input: The variable *value* could be dividends of the value **TOKENPERIOD**.

Return: **E_OK** if the initialisation is successful. **E_NOK** if unsuccessful or wrongful inputs.

UART

This is module is concerned with operating with the UART module of the MCU, in this case a TM4C123GH6PM. It is mainly used in the subordinate level of the Token Ring Network however it was used in the master level for debugging purposes.

APIs

- E_Status UART_Init (uint8_t *port*, uint32_t *baud*, uint32_t *clksrc*, bool *xInt*)

E_Status UART_Init (uint8_t *port*, uint32_t *baud*, uint32_t *clksrc*, bool *xInt*)

Description: This function is used to initialise the UART functionalities of the MCU.

Input: The variable *port* is the port number and it ranges from the values UART0 to UART7, case sensitive. The variable *baud* is the baud rate of the user's choice. The variable *clksrc* is used to determine the clock source to be used by the application and it could be one of two values either **PIOSC** or **SYSTEM**. The variable *xInt* is used to initialise a UART interrupt function.

Return: **E_OK** if initialisation is successful. **E_NOK** if unsuccessful or wrongful inputs.

Interrupt Handling

This section discusses the default interrupt handlers of each of the functionalities available in the application. They are all declared in the start-up file and could be modified within their definitions in their own c source files. They are as follows:

- **void CANIntHandler (void)** [CAN.h]
- **void MASTER_SYSTICKIntHandler (void)** [SYSTICK.h] (Master Ecu)
- **void SYSTICKIntHandler (void)** [SYSTICK.h] (Slave Ecu)
- **void UARTIntHandler (void)** [UART.h]

ECU Abstraction Layer Components

For a more normalised use of the lower level components above, they can be used efficiently as the following components.

SWITCH

This module is concerned with any switch ported to the MCU. It is attached to the DIO component of the preceding layer.

APIs

- E_Status SWITCH_Init (uint8_t *switch_num*)
- E_Status SWITCH_GetStatus (uint8_t *switch_num*, uint8_t * *status*)

E_Status SWITCH_Init (uint8_t *switch_num*)

Description: This function is used to initialise the hardware to incorporate the onboard switch.

Input: The variable *switch_num* is defined as one of two, **SWITCH_CH1** or **SWITCH_CH2.**

Return: **E_OK** if the operation is successful. **E_NOK** if unsuccessful or wrongful inputs.

E_Status SWITCH_GetStatus (uint8_t *switch_num*, uint8_t * *status*)

Description: This function is used to retrieve the status of the given switch and store it in a user defined variable.

Input: The pointer to **uint8_t,** *status* is user defined.
Return: **E_OK** if the operation is successful. **E_NOK** if unsuccessful or wrongful inputs.

LAMP

This module is concerned with the onboard lamps. It is attached to the DIO component of the preceding layer.

APIs

- E_Status LAMP_Init (uint8_t *lamp_ch_num*)
- E_Status LAMP_On (uint8_t *lamp_ch_num*)
- E_Status LAMP_Off (uint8_t *lamp_ch_num*)

E_Status LAMP_Init (uint8_t *lamp_ch_num*)

Description: This function is used to initialise the hardware to incorporate the onboard lamps.

Input: The variable *lamp_ch_num* is defined as the logical OR of any of the following **LAMP_RED**, **LAMP_GREEN**, **LAMP_BLUE**, **LAMP_WHITE**.

Return: **E_OK** if the operation is successful. **E_NOK** if unsuccessful or wrongful inputs.

E_Status LAMP_On (uint8_t *lamp_ch_num*)

Description: This function is used to turn on one of the lamp channels if they are already initialised.

Input: The input is predefined.

Return: **E_OK** if the operation is successful. **E_NOK** if unsuccessful or wrongful inputs.

E_Status LAMP_Off (uint8_t *lamp_ch_num*)

Description: This function is used to turn off one of the lamp channels if they are already initialised.

Input: The input predefined.

Return: **E_OK** if the operation is successful. **E_NOK** if unsuccessful or wrongful inputs.

ULTRASONIC

This module is concerned with the simulation of an ultrasonic sensor for the ADAS Simulator and it is attached to the UART component of the preceding layer.

APIs

- E_Status ULTRASONIC_Init (void)
- E_Status CALCULATE_Distance (void)

E_Status ULTRASONIC_Init (void)

Description: This function is used to initialize the ultrasonic sensor function simulated by the MCU.

Input: None.
Return: **E_OK** if the operation is successful. **E_NOK** if unsuccessful or wrongful inputs.

E_Status CALCULATE_Distance (void)

Description: This function is used to calculate the actual distance from the duty cycle received from the UART module.

Input: None.

Return: **E_OK** if the operation is successful. **E_NOK** if unsuccessful or wrongful inputs.

Application Layer Components

CONTROL

This component of the application layer is implemented in the master portion of the project.

APIs

- E_Status CONTROL_Init (void)
- E_Status CONTROL_Token (void)
- E_Status CONTROL_Lamps (void)

E_Status CONTROL_Init (void)

Description: This function is used to initialise the control function & Initialize Necessary Peripherals.

Input: None.

Return: **E_OK** if the operation is successful. **E_NOK** if unsuccessful or wrongful inputs.

E_Status CONTROL_Token (void)

Description: This function is used to initiate the token rotation function of the master.

Input: None.

Return: **E_OK** if the operation is successful. **E_NOK** if unsuccessful or wrongful inputs.

E_Status CONTROL_Lamps (void)

Description: This function is used to turn on the specified lamps on the master according to the State of Each Node in the Network.

Input: None.

Return: **E_OK** if the operation is successful. **E_NOK** if unsuccessful or wrongful inputs.

CONTROLSUB

This component of the application layer is implemented in the subordinate portion of the project.

APIs

- E_Status CONTROLSUB_Init (uint8_t *ecu*)
- E_Status PROXIMITY_Report (void)
- E_Status ADAS_Control (void)
- E_Status CONTROLSUB (uint8_t *node*)

E_Status CONTROLSUB_Init (uint8_t *ecu*)

Description: This function is used to initialise the control function & Initialize Necessary Peripherals..

Input: The variable *ecu* is the variable used to determine the function of each subordinate ECU node. It can only take one of two variable **ECU_NODE2** or **ECU_NODE3**.

Return: **E_OK** if the operation is successful. **E_NOK** if unsuccessful or wrongful inputs.

E_Status PROXIMITY_Report (void)

Description: This function is used to calculate and send value of the distance from the UART Duty Cycle Input.

Input: None.

Return: **E_OK** if the operation is successful. **E_NOK** if unsuccessful or wrongful inputs.

E_Status ADAS_Control (void)

Description: This function is used to send the control action based on the proximity value.

Input: None.

Return: **E_OK** if the operation is successful. **E_NOK** if unsuccessful or wrongful inputs.

E_Status CONTROLSUB (uint8_t *node*)

Description: This function is used to turn on the specified lamps on the Sub Nodes according to the State of Each Node in the Network and it also Performs the ADAS Control Functionality.

Input: The variable *ecu* is the variable used to determine the function of each subordinate ECU node. It can only take one of two variable **ECU_NODE2** or **ECU_NODE3**.

Return: **E_OK** if the operation is successful. **E_NOK** if unsuccessful or wrongful inputs.

MISRA Report

Uncorrected Violations

The following part covers MISRA rules that were violated and not corrected due to certain IDE settings and third-party files ex. “Tivaware” driver library.

1. **Chapter 1.1**: All code shall conform to ISO 9899:1990 “Programming Languages – C”.
2. **Chapter 11.3**: A cast should not be performed between a pointer type and an integral type.
3. **Chapter 11.4**: A cast should not be performed between a pointer to object type & different pointer to object type.
4. **Chapter 19.4**: C macros shall only expand to a braced initializer, a constant, a string literal, a parenthesized expression, a type qualifier, a storage class specifier, or a do-while-zero construct.

Corrected Violations

The following part covers MISRA rules that were violated and corrected.

Note: The following part only covers project files created by our team and no other third-party files ex tivaware, start-up code.

1. **Chapter 5.2**: Identifiers in an inner scope shall not use the same name as an identifier in an outer scope, and therefore hide that identifier. **Modified in:**
   - CAN.c
   - CAN.h

2. **Chapter 5.7**: No Identifier name should be reused. **Modified in:**
   - CAN.c
   - CAN.h

3. **Chapter 6.1**: The plain character type shall only be used only for storage and use of character values. **Modified in:**
   - UART.c

4. **Chapter 8.1**: Functions shall have prototype declarations and the prototype shall be visible at both the function definition and call. **Modified in:**
   - MASTER.c (main.c)

5. **Chapter 8.5**: There shall be no definition of objects or functions in a header file. **Modified in:**
   - COMMON.h

6. **Chapter 8.7**: Objects shall be defined in a block scope if they are accessed only within a single function. **Modified in:**
   - DIO.c

7. **Chapter 9.1**: All automatic variables shall be assigned a value before being used. **Modified in:**
   - CAN.c
   - CLK.c

- CONTROL.c
- CONTROLSUB.c
- UART.c
- DIO.c
- LAMP.c
- SWITCH.c
- ULTRASONIC.c
- SYSTICK.c
- MASTER.c (main.c) in Master ECU
- ADAS.c (main.c) in Slave ECUs

8. **Chapter 10.1**: The value of an expression of integer type shall not be implicitly converted to different underlying type. **Modified in:**
   - CAN.c
   - CLK.c
   - CONTROL.c
   - CONTROLSUB.c
   - UART.c
   - DIO.c
9. **Chapter 10.6**: A "U" Suffix shall be applied to all constants of unsigned Type. **Modified in:**
   - CAN.c
   - CONTROL.c
   - CONTROLSUB.c
   - SYSTICK.c
10. **Chapter 15.3**: The final clause of the switch statement shall be the default clause. **Modified in:**
    - DIO.c
    - LAMP.c
    - CONTROL.c
    - CONTROLSUB.c
    - UART.c
11. **Chapter 16.4**: The identifiers used in declaration and definition of a function shall be identical. **Modified in:**
    - CAN.h
    - CLK.h
    - CONTROL.h
    - CONTROLSUB.h
    - UART.h
    - DIO.h
    - LAMP.h
    - SWITCH.h
    - ULTRASONIC.h
    - SYSTICK.h

References

Video link https://www.youtube.com/watch?v=VDqDBcFDyJc

Drive Link for Source code files
https://drive.google.com/drive/folders/1dE8FHSSgumxO3g7fwlfQiIvB9p8mx6zB