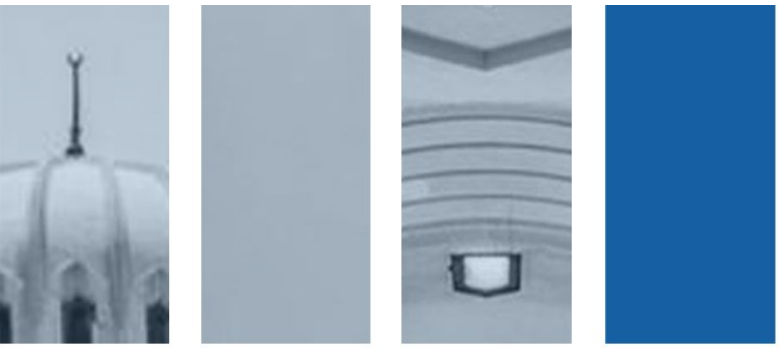
MCT441,
Embedded Automotive Networking,
SPRING 2020.



RESEARCH & PROJECT SUBMISSIONS

Program: MCTA

Course Code: MCT411

Course Name: Automotive Embedded Networking

Examination Committee

Dr. Shereif Hammad

Eng. Mohamed AbdelHay

Ain Shams University
Faculty of Engineering
Credit Hours Engineering Programs (I-CHEP)
Spring Semester – 2020

Student Personal Information for Group Work (TEAM #2)

Student Names:	Student Codes:
Assem Ibrahim Sehsah	17P8130
Ahmed Magdy Ibrahim	14P8208
Ahmed Mostafa Anwar	17P8123
Muhamed Essam Mahmoud	15P8186
Muhannad Hossam Rayan	15P8198
Yousef Mohsen Ibrahim	15P8030

Plagiarism Statement

I certify that this assignment / report is my own work, based on my personal study and/or research and that I have acknowledged all material and sources used in its preparation, whether they are books, articles, reports, lecture notes, and any other kind of document, electronic or personal communication. I also certify that this assignment / report has not been previously been submitted for assessment for another course. I certify that I have not copied in part or whole or otherwise plagiarized the work of other students and / or persons.

Signature/Student

Date: Aug 14, 2020

Table of Contents

Table of Figures

1.0 INTRODUCTION

During CSE347 course, we studied analysis and design of real-time embedded systems. We have addressed scheduler and priority based pre-emptive Free RTOS ported on Cortex M3/M4 platform. And during MCT441 course we learned about the communication protocols. So this project is to make sure that we understood the real time operating systems and communication protocols right, and can make applications depending on that.

1.1 Specific Purpose

We have addressed scheduler and priority based pre-emptive Free RTOS on Cortex M4 platform. Queue, Semaphores, critical sections, and interrupts management are covered in previous courses, in addition to the networking in this course such as CAN and UART, and the goal of this project is to make a good experience by implementing all that we have learned during this course and previous courses in one project.

1.2 Intended Audience

Dr. Shereif Hammad (doctor of the course), Eng. Mohamed AbdelHay, and of course anyone who is interested in the content of this project or interested in embedded system and communication protocols in general.

1.3 Motivation

Learning and experience new things about embedded systems, experience how to build our own different scheduler, learn more about communication protocols and understand how the layers system works, and it will be a good training for us to practice on coding in teams.

1.4 Background

We already studied about Microcontrollers in a previous course, and studied how to make a suitable C code to run on TIVA-C, how interrupts work, how registers work, and how to use it right. And in Embedded System Design course we studied analysis and design of real time embedded systems. We have addressed scheduler and priority based pre-emptive Free RTOS ported on Cortex M4 platform and learned about Queue, Semaphores, critical sections, and interrupts management online. And in this course we learned about communication protocols such as CAN Bus and UART and how these protocols work.

2.0 SYSTEM DESIGN PART

2.1 Project Specifications and Description

The logical ring uses token ring message. This message is transmitted from low address (low ID) to high address (high ID) and finally from the highest address back to the lowest address, which forms a ring. In below Figure, the ring message is transmitted from 1 to 2, then from 2 to 3, and so on. The ring message is finally transmitted from 3 to 1 to form a ring, and then the ring transmission starts over again.

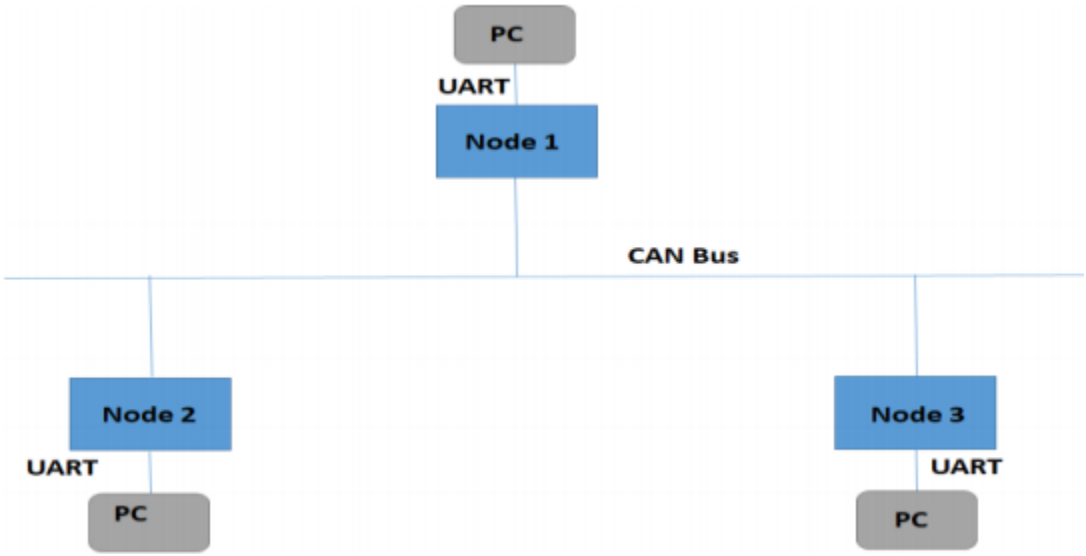


Figure 1: System Diagram

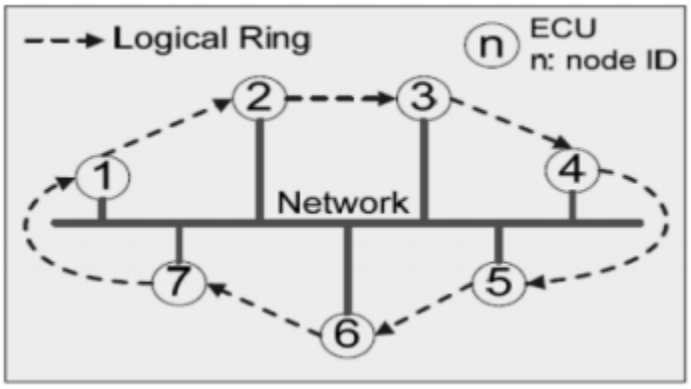


Figure 2: CAN Token Ring

2.2 Project Management

We separated the project into small parts. Each part we discussed thoroughly. Each one in the team has a critical rule. If a part fail, the project fails. We set our timeline perfectly to make sure if any delay happens, we can still complete our project.

The best part is the layered architecture. Each one of us makes and tests small number of files and functions. By doing so, the project management was much easier than before where the code in one file and debugging was a nightmare.

We created 32 files to contain all our project needs. Each member had participated in coding. We double-checked naming convention used in our project. Consistent naming scheme is much better than nothing.

Before each member submit his file to us, he makes sure it is fully working by making deep testing to make sure of the functionality.

Having one team leader (Assem Ibrahim) helped a lot solving many arguments. It made us take quick and critical decisions. Before picking our team leader, everyone was making what it fits. But with strict orders for the leader each of us was not thinking but to implement what he needed in order to complete the big picture and have a working project.

2.3 Functional Requirements

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

2.4 Token Ring

2.4.1 Desired features

1) A ring token should be rotated along the network nodes.
2) Each token should hold the token 1 second before transmitting to the next ECU.
3) The ECU which has the token should turn on the white LED as long as it holds the token.
4) At initial state, each ECU is associated with a specific color to be turned on as the following:

Node 1	RED
Node 2	GREEN
Node 3	BLUE

5) The token LED will overwrite the ECU specific color.
6) 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
Node 2	BLUE

1) 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	GREEN
Node 1	RED

2.4.2 Feature obtained

1) Feature 1: *A ring token should be rotated along the network nodes*, is achieved as long as the scheduler is running.
2) Feature 2: *Each token should hold the token 1 second before transmitting to the next ECU*, is obtained by "IndLEDCtrlSys_TokenTimeout" function.

```
/*******************************************************************************
 * Name         : IndLEDCtrlSys_TokenTimeout                                   *
 * Inputs       : None                                                         *
 * Outputs      : None                                                         *
 * Return Value : None                                                         *
 * Description  : Clears the token if and only if we have a valid token.       *
 *******************************************************************************/
void IndLEDCtrlSys_TokenTimeout(void)
{
/* Static counter for timeout */
static volatile uint16_t Counter = INITIALIZED_TO_ZERO;

    /* Check if ECU has token. */
    if (true == g_HasToken)
    {
        /* Increment counter by 1. */
        ++Counter;

        /* Check timeout. */
        if (TOKEN_TIMEOUT == Counter)
        {
            /* Reset counter. */
            Counter = INITIALIZED_TO_ZERO;

            /* Remove token from ECU as it is passed to other. */
            g_HasToken = false;

            /* Enable sending token. */
            g_SendToken = true;
        }
    }
    else
    {
        /* MISRA */
    }
}
```

Figure 3: IndLEDCtrlSys_TokenTimeout function

3) Feature 4: *At initial state, each ECU is associated with a specific color to be turned on*. Feature 6: *If SW1 of the token owner ECU is pressed. Then the color sequence will be changed and started from this ECU*, and Feature 7: *If SW2 of the token owner ECU is pressed, then the color sequence will be changed and started from this ECU with reversed order*. All of this mentioned in the Designed features section, is achieved by "IndLEDCtrlSys_UpdateDefaultColor" function.

```
/*******************************************************************************
 * Name         : IndLEDCtrlSys_UpdateDefaultColor                             *
 * Inputs       : None                                                         *
 * Outputs      : None                                                         *
 * Return Value : None                                                         *
 * Description  : Updates default ECU indicator LED color.                    *
 *******************************************************************************/
void IndLEDCtrlSys_UpdateDefaultColor(void)
{
    /* Check if ECU has token. */
    if (true == g_HasToken)
    {
        /* Check if SW1 is pressed. */
        if (IS_PRESSED == g_SW1_Status)
        {
            /* Set default color to start. */
            g_Default_Color = color[START];
        }
        else if (IS_PRESSED == g_SW2_Status)
        {
            /* Set default color to end (reversed). */
            g_Default_Color = color[END];
        }
        else
        {
            /* MISRA */
        }
    }
    else /* NO TOKEN */
    {
        if ((true == g_SW1_Received) && (true == g_SW2_Received))
        {
            /* DO NOTHING */
        }
        else /* AT LEAST ONE SWITCH IS PRESSED */
        {
            /* Check if switch 1 is pressed. */
            if (true == g_SW1_Received)
            {
                /* Set new default color. */
                g_Default_Color = state[NODE_ID][g_PrevNodeID];
            }
            /* Check if switch 2 is pressed. */
            else if (true == g_SW2_Received)
            {
                /* Set new default color (reversed). */
                g_Default_Color = state[NODE_ID][(g_PrevNodeID + MAP) % COUNT];
            }
            else /* NOTHING IS PRESSED */
            {
                /* MISRA */
            }
        }
    }
}
```

Figure 4: IndLEDCtrlSys_UpdateDefaultColor function

4) Feature 3: *The ECU which has the token should turn on the white LED as long as it holds the token*, and Feature 5: *The token LED will overwrite the ECU specific color*, are achieved by "IndLEDCtrlSys_UpdateIndLED" function.

```
/*****************************************************************************
 * Name         : IndLEDCtrlSys_UpdateIndLED                                 *
 * Inputs       : None                                                       *
 * Outputs      : None                                                       *
 * Return Value : None                                                       *
 * Description  : Update indicator LED color depending on token.            *
 *****************************************************************************/
void IndLEDCtrlSys_UpdateIndLED(void)
{
    /* Check if ECU has token. */
    if (true == g_HasToken)
    {
        /* Set indicator LED color (white). */
        IndicatorLED_SetColor(COLOR_WHITE);
    }
    else
    {
        /* Set indicator LED color. */
        IndicatorLED_SetColor(g_Default_Color);
    }
}
```

Figure 5: IndLEDCtrlSys_UpdateIndLED function

2.5 ADAS Simulation

2.5.1 Desired features

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 forward & reduce speed
3 meter	Move forward
No Obstacle	Move forward & increase speed

2.5.2 Feature obtained

1) Node 2 role can be achieved by 3 functions:
   - ADASSensorSimulator_UpdateDutyCycle
   - ADASSensorSimulator_SendDistance
   - ADASSensorSimulator_UpdateDistance

```
void ADASSensorSimulator_UpdateDutyCycle(void)
{
uint8_t ReceivedByte;   /* received duty cycle */

    /* Get received byte. */
    ReceivedByte = UART_ReceiveByte();

    /* Check if UART has sent. */
    if (true == g_UART_available)
    {
        /* Store received byte. */
        g_SensorDutyCycle = ReceivedByte;
    }
    else /* UART NOT AVAILABLE */
    {
        /* MISRA */
    }
}
```

Figure 6: ADASSensorSimulator_UpdateDutyCycle function

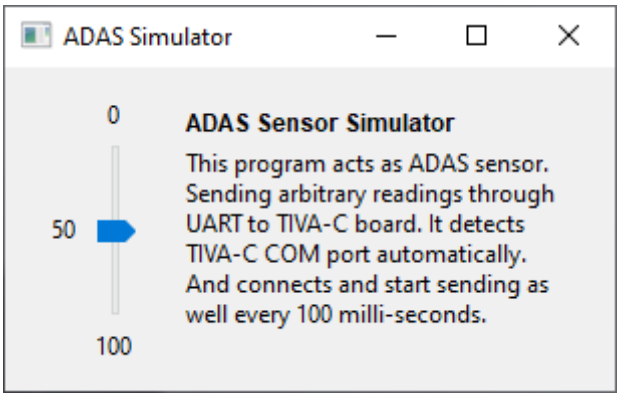
```
void ADASSensorSimulator_SendDistance(void)
{
    /* Update duty cycle readings. */
    ADASSensorSimulator_UpdateDutyCycle();

    /* Update distance to be sent. */
    ADASSensorSimulator_UpdateDistance();

    /* Compress message to be sent. */
    MessageServices_Compress(&g_txCAN, DISTANCE_MSG_ID);

    /* Transmit over CAN. */
    CAN_TransmitByte();
}
```

Figure 7: ADASSensorSimulator_UpdateDistance function

```
void ADASSensorSimulator_UpdateDistance(void)
{
    /* Update distance based on duty cycle. */
    if (g_SensorDutyCycle <= ADAS_LIMIT_0)
    {
        /* Update stored distance. */
        g_SensorDistance = ADAS_SENSOR_ERROR;
    }
    else if (g_SensorDutyCycle <= ADAS_LIMIT_1)
    {
        /* Update stored distance. */
        g_SensorDistance = ADAS_SENSOR_METER_1;
    }
    else if (g_SensorDutyCycle <= ADAS_LIMIT_2)
    {
        /* Update stored distance. */
        g_SensorDistance = ADAS_SENSOR_METER_2;
    }
    else if (g_SensorDutyCycle <= ADAS_LIMIT_3)
    {
        /* Update stored distance. */
        g_SensorDistance = ADAS_SENSOR_METER_3;
    }
    else if (g_SensorDutyCycle <= ADAS_LIMIT_4)
    {
        /* Update stored distance. */
        g_SensorDistance = ADAS_SENSOR_NO_OBSTACLE
    }
    else
    {
        /* More than 100% (IMPOSSIBLE VALUE) */
        /* DO NOTHING */
    }
}
```

Figure 8: ADASSensorSimulator_SendDistance function

Note: *the values that node 2 receives is produced and sent by a python code that sends a number from 0 to 100 by* ***a slider***.



2) Node 3 role can be achieved by ADASSimulator_Control function.

```
void ADASSimulator_Control(void)
{

    if (true == g_ADASReceived)
    {
        g_ADASReceived = false;

        switch (g_ReceivedDistance)
        {
        case ADAS_SENSOR_ERROR:
            UART_printf(ADAS_T0);
        break;
        case ADAS_SENSOR_METER_1:
            UART_printf(ADAS_T1);
        break;
        case ADAS_SENSOR_METER_2:
            UART_printf(ADAS_T2);
        break;
        case ADAS_SENSOR_METER_3:
            UART_printf(ADAS_T3);
        break;
        default: /* NO OBSTACLE */
            UART_printf(ADAS_T4);
        }
    }
    else
    {
        /* MISRA */
    }
}
```

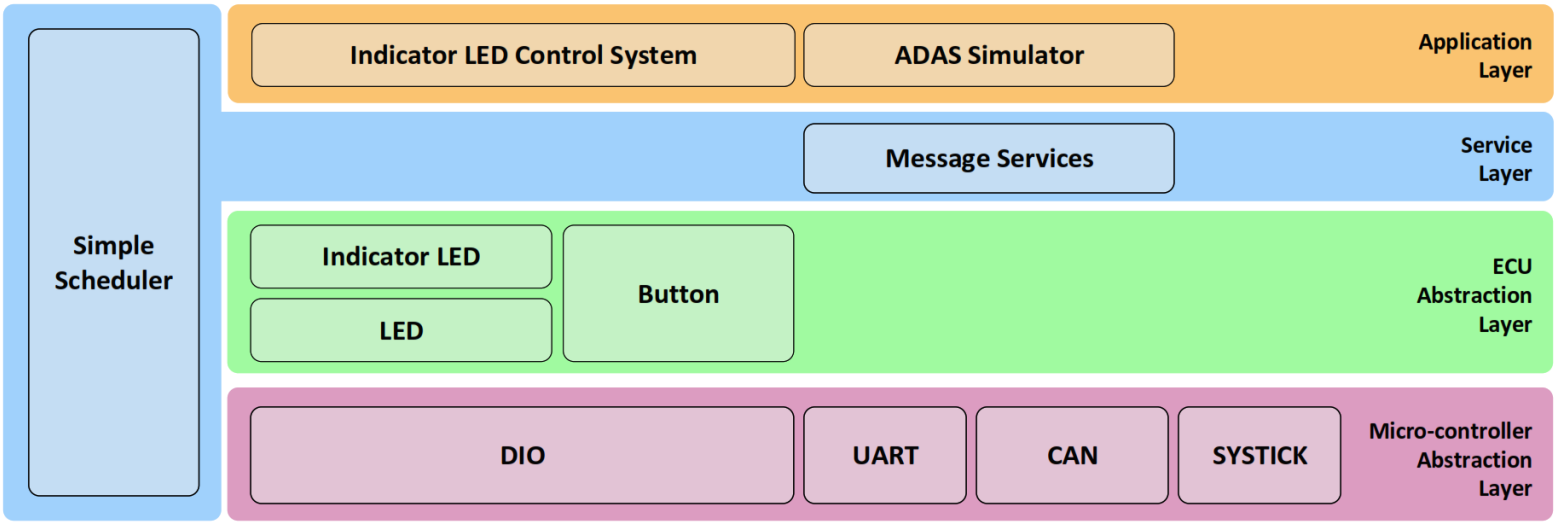
Figure 9: ADASSimulator_Control function

Note: *the message received and printed on the CMD by python code*.

```
/* Conditions */
#define ADAS_SENSOR_ERROR       (0)
#define ADAS_SENSOR_METER_1     (1)
#define ADAS_SENSOR_METER_2     (2)
#define ADAS_SENSOR_METER_3     (3)
#define ADAS_SENSOR_NO_OBSTACLE (4)

/* CMD: UART output simulation */
#define ADAS_T0 "\n(0) CAR STOP!"
#define ADAS_T1 "\n(1) CAR TURN!"
#define ADAS_T2 "\n(2) MOVE FWD, REDUCE SPEED!"
#define ADAS_T3 "\n(3) MOVE FWD!"
#define ADAS_T4 "\n(4) MOVE FWD, INCREASE SPEED!"
```

Figure 10: Output Simulation

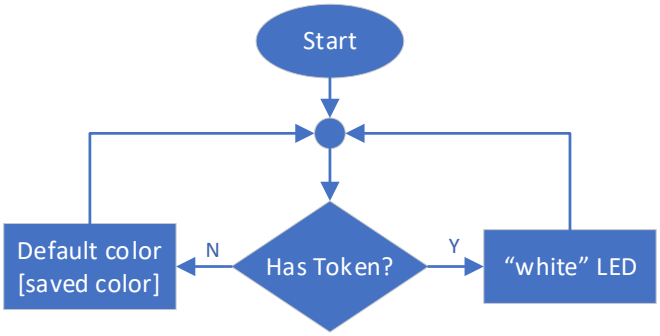
2.5 Architecture Layer



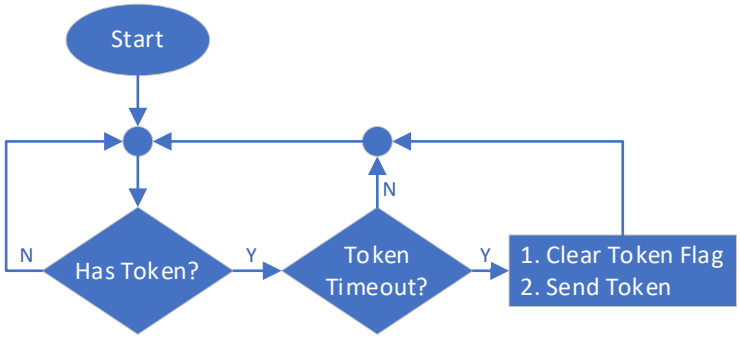
MODULE NAME	HEADER FILES	SOURCE FILES
DIO	diodrv_regs.h diodrv.h	diodrv.c
UART	uart_stdio.h	uart_stdio.c
CAN	can.h	can.c
SYSTICK	systick.h systick_regs.h	systick.c
LED	led.h	led.c
BUTTON	button.h	button.c
INDICATOR LED	indicator_led.h	indicator_led.c
SIMPLE SCHEDULER	scheduler.h	scheduler.c
MESSAGE SERVICES	message_services.h	message_services.c
INDICATOR LED CONTROL SYSTEM	indicator_led_control_system.h	indicator_led_control_system.c
ADAS SENSOR SIMULATOR	adas_sensor_simulator.h	adas_sensor_simulator.c
ADAS SIMULATOR	adas_simulator.h	adas_simulator.c
APPLICATION	app.h	app.c
OTHER FILES	common.h configuration.h types.h	

2.6 Block Diagram

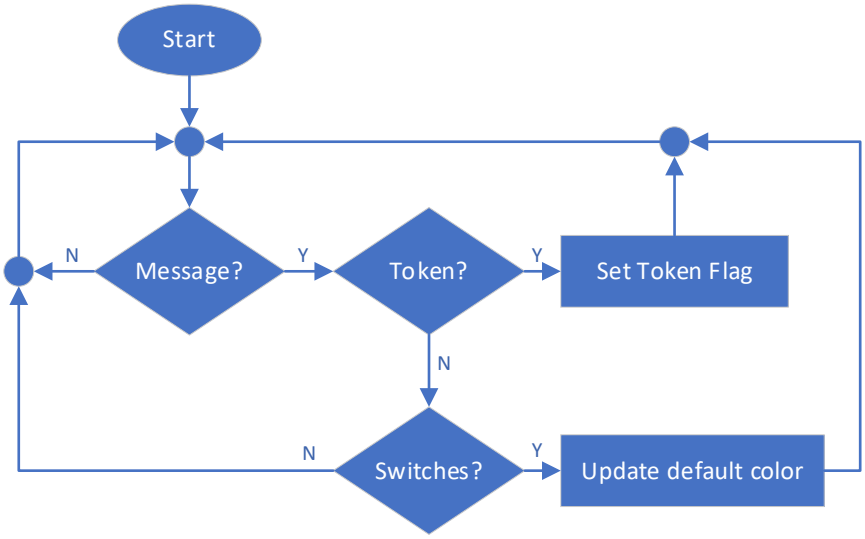
2.6.1 LED Color Display logic



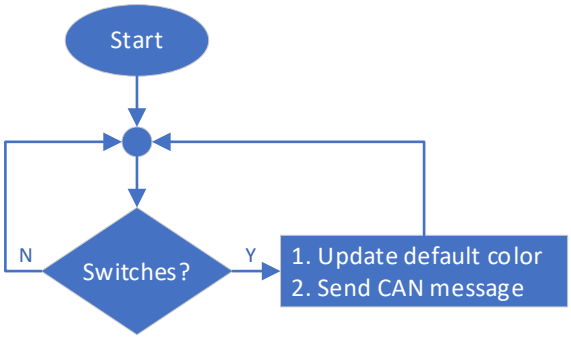
2.6.2 Check Token Timeout logic



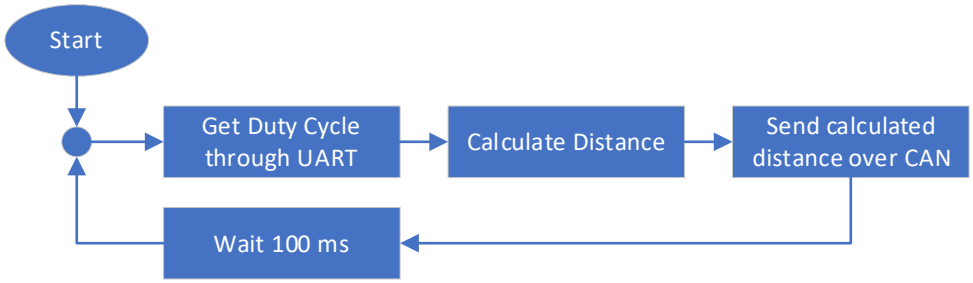
2.6.3 Indicator LED Message Received logic



2.6.4 If **Switches pressed** logic



2.6.5 **ADAS Sensor Simulator** logic



2.6.6 **ADAS Simulator** logic

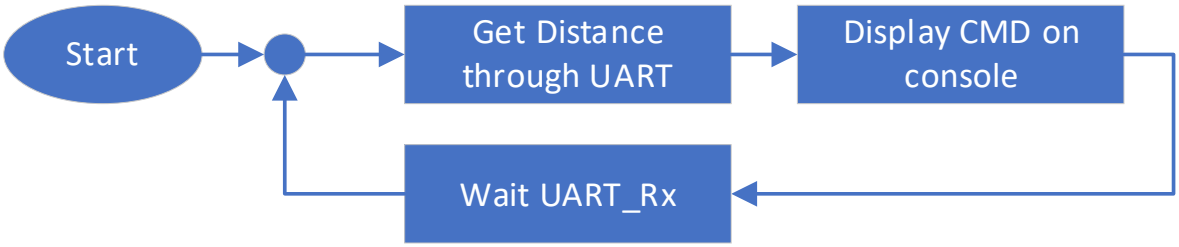
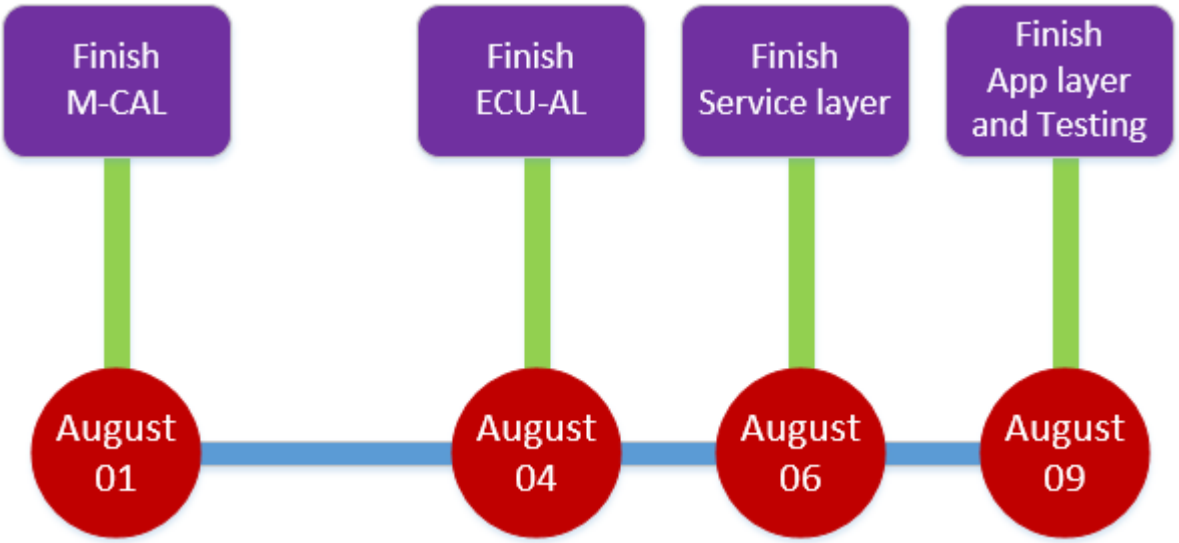


Figure 11: Node Flow Charts

2.7 Plan of Timeline



3.0 TECHNICAL DETAILS PART

3.1 Design Details

We needed a the simulation file first to be ready so we can test on it whenever we want and that file will be sharing among all the team members. After we build the simulation file and test with a normal code, we then decided the ports and pins that we will use. Then every member took his tasks and started to think how to do it.

The ECU-AL (Hardware) cannot work without MCAL, and also the App layer cannot work without the Hardware layer, so we started with MCAL, then ECU-AL, then Service layer, and finally the App layer.

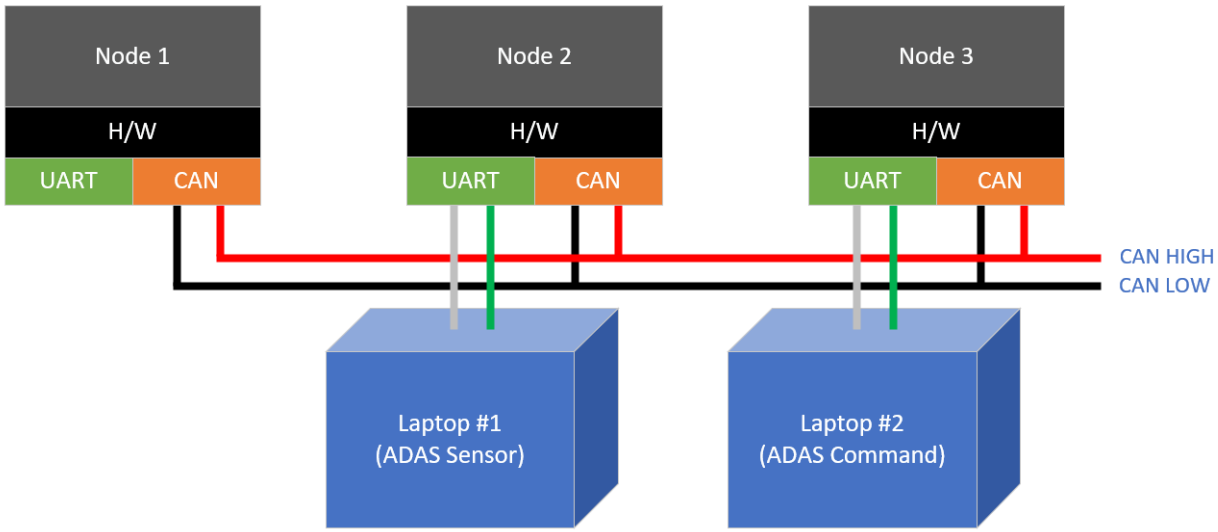
We made a role that every code line we write we must make a comment for it so all of us can understand what others wrote and any reader of the code can understand it. And we didn't use the registers (PORT, PIN, and DDR) with their names, instead we used the zeros & ones address to access them.

3.2 How to Test the Code

Each module that can be tested individually will be tested. Like DIO driver, Systick, Scheduler, etc. By doing so, each module will be working 100% with test cases that fits our purpose and need. If the module passed all tests, then it is finished and ready to be assembled with other modules in the application layer.

We tested all modules except application layer as it required additional hardware connections and cannot be tested individually.

3.3 Simulation



This diagram shows how we connected our project. The output is as required in requirements PDF provided.

Working Video Link: [Hyperlink] Video - All Project Features

3.4 Testing

After watching All features video, here are extreme cases for testing the requirements:

1. Color Sequence Test: [Hyperlink] Video - Color Sequence Test

2. Token Test: [Hyperlink] Video - Token Operation Test

3. ADAS Simulator: [Hyperlink] Video - ADAS Simulator

3.6 Lessons Learned

1. How to build a whole system from nothing.
2. The architecture layer is very easy to use.
3. How to communicate with team to build a big program.
4. How to build architecture layer system.
5. Understand more about the scheduler and how it works.
6. Understand more about the communication protocols and how they work.

3.7 Problems Faced

We had a problem with the application layer about the modules integration, that's why we returned to the Hardware Layer to make some changes and add some functions and variables to solve the problems in the App Layer specially in train system, so that we finished August 14 instead of August 9, and we learned that when we face problem in a high layer we can return to the a lower layer to solve the problem.

3.8 Function Prototypes

Now in this section we will provide **all extern functions** used in the project with **small comment** to indicate what they do.

diodrv.h

```
/* Initializes DIO pins. */
extern E_status DIO_InitPin(port_t, pin_t, diomode_t);

/* Reads DIO pins (i/p). */
extern E_status DIO_ReadPin(port_t, pin_t, diodata_t *);

/* Changes DIO pins (o/p). */
extern E_status DIO_WritePin(port_t, pin_t, diodata_t);
```

systick.h

```
/* Starts systick */
extern void Systick_Start(void);

/* Sets systick callback function */
extern void Systick_SetCallBack(volatile void(*fn_ptr)(void));
```

can.h

```
/* Initializes CAN bus */
extern void CAN_Init(void);

/* Initializes CAN bus for receiving */
extern void CAN_ReceiveConfig(void);

/* Initializes CAN bus for transmitting */
extern void CAN_TransmitConfig(void);

/* Transmits CAN byte as a message */
extern void CAN_TransmitByte(void);

/* Receives CAN byte as a message */
extern void CAN_ReceiveByte(void);
```

uart_stdio.h

```
/* Initializes UART */
extern void UART_Init(void);

/* Prints string to console */
extern void UART_printf(const char *);

/* Receives byte */
extern char UART_ReceiveByte(void);
```

button.h

```
/* Initializes selected button. */
extern E_status Button_Init(channel_t);

/* Reads selected button status. */
extern E_status Button_UpdateStatus(channel_t, diodata_t *);
```

led.h

```
/* Initializes selected LED. */
extern E_status LED_Init(channel_t);

/* Turns on selected LED. */
extern E_status LED_On(channel_t);

/* Turns off selected LED. */
extern E_status LED_Off(channel_t);
```

indicator_led.h

```
/* Initializes all LEDs required */
extern E_status IndicatorLED_Init(void);

/* Set color of indicator LED */
extern void IndicatorLED_SetColor(color_t);
```

app.h

```
/* Initializes tasks */
extern void Task_Init(void);
```

indicator_led_control_system.h

```
/* Checks for received messages */
extern void IndLEDCtrlSys_ReceiveMessage(void);

/* Transmits message */
extern void IndLEDCtrlSys_TransmitMessage(void);

/* Updates default ECU LED color variable (not white) */
extern void IndLEDCtrlSys_UpdateDefaultColor(void);

/* Checks for token Timeout */
extern void IndLEDCtrlSys_TokenTimeout(void);

/* Updates indicator LED color */
extern void IndLEDCtrlSys_UpdateIndLED(void);

/* Updates buttons status */
extern void IndLEDCtrlSys_UpdateButtonStatus(void);
```

scheduler.h

```
/* Initializes scheduler */
extern void Scheduler_Init(void);

/* Increments ticks for scheduler */
extern volatile void Scheduler_NewTimerTick(void);

/* Main scheduler component */
extern void Scheduler_Loop(void);
```

message_services.h

```
/* Compress message to be sent as a byte. */
extern void MessageServices_Compress(uint8_t *buffer, uint8_t msg_id);

/* Extract data from a message. */
extern void MessageServices_Extract(uint8_t buffer);
```

adas_sensor_simulator.h

```
/* Receive duty cycle by UART */
extern void ADASSensorSimulator_UpdateDutyCycle(void);

/* Calculates distance based on duty cycle */
extern void ADASSensorSimulator_UpdateDistance(void);

/* Sends distance over CANBUS */
extern void ADASSensorSimulator_SendDistance(void);
```

scheduler.h

```
/* Outputs command on UART console */
extern void ADASSimulator_Control(void);
```

“configuration.h” for node 1

```
/* NODE_IDs for indicator LED system */
#define NODE_ID                     ( NODE_1 )
#define NODE_ID_AFTER               ( NODE_2 )
#define NODE_ID_PREV                ( NODE_3 )

/* Initial conditions */
#define NODE_DEFAULT_COLOR          (COLOR_RED)
#define NODE_INITIAL_TOKEN_STATE    (true)
```

4.0 MISRA Report

4.1 Abstract

MISRA C IS A SET OF SOFTWARE DEVELOPMENT GUIDELINES FOR THE C PROGRAMMING LANGUAGE DEVELOPED BY MISRA (**MOTOR INDUSTRY SOFTWARE RELIABILITY ASSOCIATION**). ITS AIMS ARE TO FACILITATE CODE SAFETY, SECURITY, PORTABILITY AND RELIABILITY IN THE CONTEXT OF EMBEDDED SYSTEMS, SPECIFICALLY THOSE SYSTEMS PROGRAMMED IN ISO C / C90 / C99.

NOTE: All of MISRA warnings that appear belongs to TIVAWARE drivers.

4.2 MISRA Warnings

#1397-D (MISRA-C:2004 10.5/R) If the bitwise operators ~ and << are applied to an operand of underlying type unsigned char or unsigned short, the result shall be immediately cast to the underlying type of the operand

Justification: None

#1422-D (MISRA-C:2004 16.9/R) A function identifier shall only be used with either a preceding &, or with a parenthesized parameter list, which may be empty

Justification: when we tried to apply it , an error occurred “#60 function call is not allowed in a constant expression”

#1389-D (MISRA-C:2004 8.12/R) When an array is declared with external linkage, its size shall be stated explicitly or defined implicitly by initialization

Justification: we need to use the array size with variable that has constant value. However, when we tried to do so it builds with error.

#1393-D (MISRA-C:2004 10.1/R) The value of an expression of integer type shall not be implicitly converted to a different underlying type if the expression is complex

Justification: None

#1376-D (MISRA-C:2004 1.1/R) Ensure strict ANSI C mode (-PS) is enabled

Justification: None

5.0 DELIVERABLES

Main OneDrive Folder: [Hyperlink] MCT411 - Team #2 - Project Folder

Videos: [Hyperlink] MCT411 - Team #2 - Videos

Source Code (Uncompressed): [Hyperlink] MCT411 - Team #2 - Source Code (Uncompressed)

Source Code (Compressed): [Hyperlink] MCT411 - Team #2 - Source Code (Compressed)

Source Code (Python): [Hyperlink] MCT411 - Team #2 - ADAS Simulator (Python)