IEEE HELWAN
Student Branch

IEEE ComSoc HelwanSC
IEEE Communications Society

Remote Control Car (RC-CAR)

Team Members

SN	Student Name
1	Ahmed Ali Fahmy
2	Alfred Nagy Alfred
3	Ahmed Mohamed Talaat

Submission Date: 16-7-2024

Table of Contents

Table of Figures

1. Abstract

The RC-Car Embedded System Project represents a culmination of efforts in designing and developing a remote-controlled car using the ATmega32 microcontroller. This project aims to combine fundamental principles of electronics, programming, and mechanical engineering to create a functional and educational prototype. The primary objective is to demonstrate the integration of various hardware and software components to achieve remote control functionality via a mobile app. Key features include Bluetooth communication for wireless control, motor control using PWM techniques, and a robust power management system. An additional feature of the project is the integration of an ultrasonic sensor for obstacle detection, enabling the car to autonomously detect and avoid obstacles in its path. Through iterative design, programming, and testing phases, the project emphasizes hands-on learning in embedded systems. The documentation outlines the system architecture, design methodology, implementation details, performance analysis, and potential future enhancements, providing insights into the practical applications of embedded systems technology.

2. Introduction

2. 1 Project Overview

The RC-Car Embedded System Project focuses on the development of a small-scale remote-controlled car using the ATmega32 microcontroller. This project integrates principles of electronics, programming, and mechanical design to create a functional prototype capable of wireless operation via a mobile app. The car incorporates essential components such as DC motors, a motor driver, a Bluetooth module (HC-06), an ultrasonic sensor for obstacle detection, and a power supply to facilitate controlled movement and communication.

2. 2 Objective

The primary objective of this project is to provide a practical learning experience in embedded systems design and implementation. By constructing the RC-Car, participants gain hands-on exposure to:

- Microcontroller Programming: Utilizing Embedded C to control motor functions and manage Bluetooth communication.
- Hardware Integration: Connecting and configuring electronic components to ensure seamless operation.
- System Optimization: Implementing efficient power management and control algorithms to enhance performance.
- Educational Value: Demonstrating the application of embedded systems in real-world scenarios, emphasizing both technical skills and problem-solving abilities.

2. 3 Scope

The scope of this project encompasses:

- System Design: Detailed planning and schematic design of the electronic circuitry and mechanical structure.
- Software Development: Writing and debugging code for the ATmega32 microcontroller to interpret commands and drive motors.
- Testing and Validation: Conducting rigorous testing to verify functionality, reliability, and performance under various conditions.
- Documentation: Compiling comprehensive documentation to capture design decisions, development processes, and outcomes for future reference and educational purposes.

3. System Architecture

3. 1 Hardware Components

1. ATmega32 Microcontroller:
    - Acts as the central processing unit, controlling all operations of the RC-Car.
    - Utilizes GPIO pins for interfacing with peripheral components and motor control.
2. Motors and Car Body:
    - Motors: Two DC motors responsible for driving the car's movement.
    - Car Body: Provides structural support and houses all electronic components securely.
3. Motor Driver (L298N):
    - Facilitates motor control by interpreting signals from the microcontroller and adjusting motor speed and direction accordingly.
4. Bluetooth Module (HC-06):
    - Enables wireless communication between the RC-Car and a mobile app.
    - Utilizes UART communication protocol to receive control commands from the mobile app.
5. Ultrasonic Sensor:
    - Detects obstacles in the car's path and provides distance measurements to the microcontroller.
    - Allows the car to switch to obstacle detection mode, enabling autonomous navigation around obstacles.
6. Power Supply:
    - Supplies appropriate voltage and current to all components, ensuring stable operation.

- Typically powered by a rechargeable battery, ensuring mobility and flexibility in usage.

3. 2 Software Components

1. Embedded C Programming:
    - Developed for the ATmega32 microcontroller to handle tasks such as Bluetooth communication, motor control via PWM, and sensor interfacing.
2. Mobile App Interface:
    - Allows users to send control commands to the RC-Car, translating user inputs into readable signals for the microcontroller.
    - Sends control commands (forward, backward, left, right, stop, speed) to the RC-Car via the Bluetooth module.

3. 3 System Integration

1. Command Reception and Processing:
    - Mobile app sends commands (via Bluetooth) to the HC-06 module.
    - HC-06 module forwards these commands to the ATmega32 microcontroller for interpretation.
2. Motor Control:
    - ATmega32 interprets received commands to control motor speed and direction through the L298N motor driver.
    - PWM signals generated by the microcontroller regulate motor speed.
3. Obstacle Detection and Avoidance:
    - Ultrasonic sensor continuously scans for obstacles in the car’s path.
    - When an obstacle is detected, the microcontroller processes the distance data and adjusts the car’s movement to avoid collision.

4. Design and Development

4. 1 Circuit Diagram

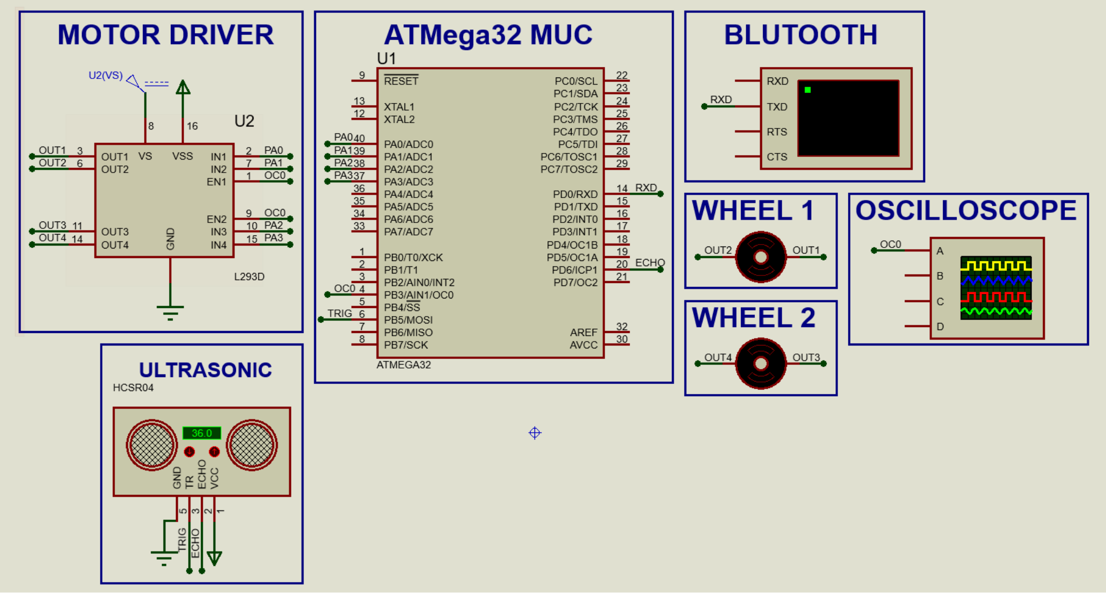

Figure 1: Circuit Diagram

5. Features and Functionality

5. 1 Remote Control Mechanism

Bluetooth Communication:

- Facilitates wireless control of the RC-Car via a mobile app.
- Uses the HC-06 Bluetooth module for reliable communication between the app and the ATmega32 microcontroller.

5. 2 Motor Control

Speed and Direction Control:

- ATmega32 microcontroller regulates motor speed and direction using PWM signals.
- Enables precise maneuvering of the RC-Car in various directions (forward, backward, left, right).

5. 3 Obstacle Detection Mode

Ultrasonic Sensor:

- Detects obstacles in the car's path using ultrasonic waves.
- Provides distance measurements to the microcontroller, enabling autonomous navigation around obstacles.
- When an obstacle is detected within a predefined range, the car can stop or change direction to avoid collision.

5. 4 Communication Protocols

UART Communication:

- Establishes communication between the ATmega32 microcontroller and the HC-06 Bluetooth module.
- Ensures reliable data transmission for remote control commands from the mobile app to the RC-Car.

6. Peripheral Configuration

6. 1 GPIO (General Purpose Input/Output)

1. Configures and controls GPIO pins for interfacing with external devices and sensors.
2. Implements functions for setting pin direction (input/output), reading pin states, and writing digital signals.

6. 2 UART (Universal Asynchronous Receiver-Transmitter)

1. Initializes and manages UART communication for serial data transmission.
2. Implements functions for baud rate configuration, data frame format, and data transmission/reception.

6. 3 PWM (Pulse-Width Modulation)

1. Generates PWM signals to control motor speed and direction through the L298N motor driver.
2. Configures PWM channels, duty cycles, and frequency settings based on motor control requirements.

6. 4 Timer1 as Input Capture Unit (ICU)

1. Utilizes Timer1 for the Input Capture Unit (ICU) feature to accurately measure the time intervals for ultrasonic sensor signals.
2. Provides precise timing measurements for calculating distances to obstacles based on ultrasonic echo signals.

7. Layered Architecture

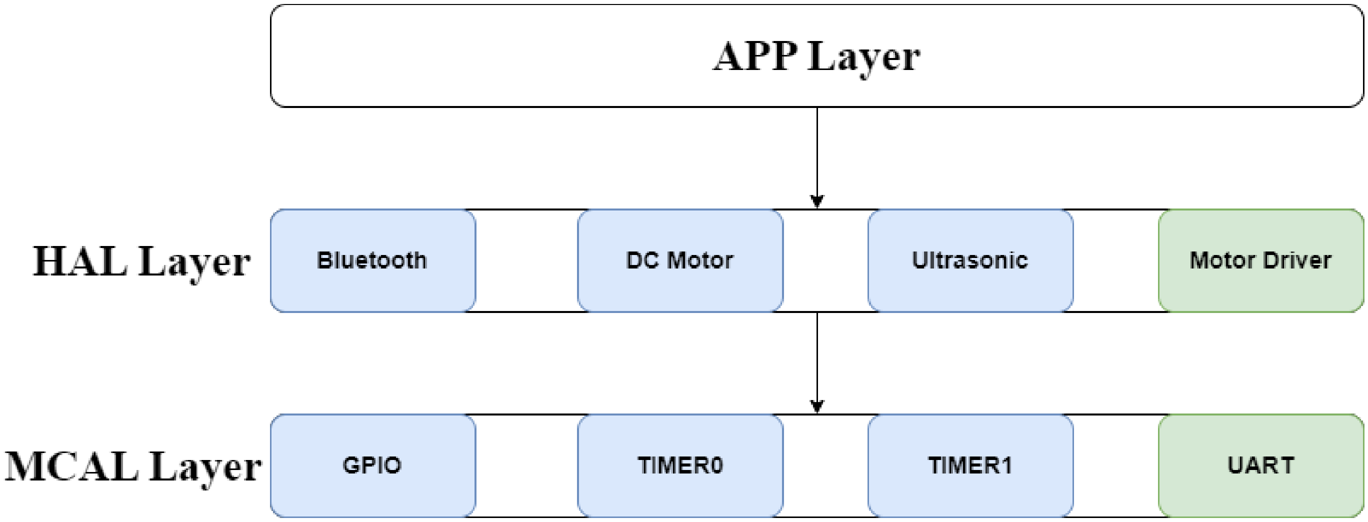

Figure 2: Layered Architecture

8. List Of Components

SN	Item Type	Item Code	Quantity	Price
1	Microcontroller	ATmega32	1	260 LE
2	Bluetooth Module	HC-6	1	190 LE
3	Motor Driver	L298N	1	100 LE
4	Car Body	-	1	200 LE
5	Car Wheel	-	2	50 LE
6	Chargeable Battery	-	3	60 LE
7	Battery Holder	-	1	10 LE
8	Ultrasonic	HC_SR04	1	35 LE
9	Miscellaneous Items	-	-	Around 50 LE

9. Hardware Photos

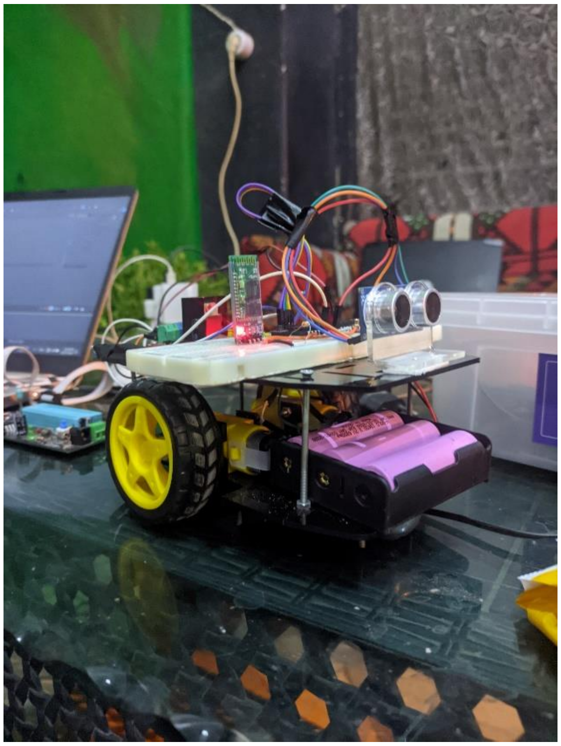

Figure 3: Hardware Photo1

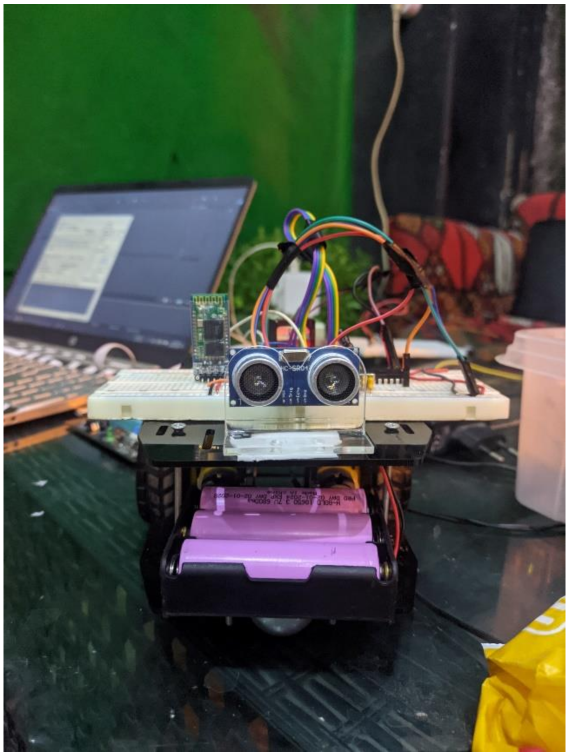

Figure 4: Hardware Photo2

10. Source Code

10. 1 Main Function

```
#include "HAL/BLUETOOTH/bluetooth.h"
#include "HAL/MOTOR/motor.h"
#include "HAL/MOTOR_DRIVER/motorDriver.h"
#include "HAL/ULTRASONIC_SENSOR/ultrasonic_sensor.h"
#include "MCAL/GPIO/gpio_private.h"

#include <util/delay.h>

uint8 HC5_input = 0;
uint8 Ultra_Distance = 0;

void RC_Desicion();
void ULTRA_Desicion();
void Desicion_Direction();

int main()
{
    /* Configuration and initialization functions */
    DcMotor_Init();
    bluetooth_init();
    Ultrasonic_init();

    SREG_REG.Bits.I_Bit = 1;

    while (1)
    {
        HC5_input = bluetooth_recieveByte();
        while (HC5_input == 'W') //RC mode
        {
            HC5_input = bluetooth_recieveByte();

            RC_Desicion();

            HC5_input = 'W';
        }
        while (HC5_input == 'U') //ultrasonic mode
        {
            Set_Speed(40);
            Ultra_Distance = Ultrasonic_readDistance();

            Desicion_Direction();
            ULTRA_Desicion();

            HC5_input = 'U';
        }
    }
}

void RC_Desicion()
{
    switch (HC5_input)
    {
    case 'F':
        Move_Forward();
        break;
    case 'B':
        Move_Backward();
        break;
    case 'L':
        Move_Left();
        break;
    case 'R':
        Move_Right();
        break;
    case 'G':
        Move_Left();
        _delay_ms(50);
        Move_Forward();
        _delay_ms(50);
        break;
    case 'I':
        Move_Right();
        _delay_ms(50);
        Move_Forward();
        _delay_ms(50);
        break;
    case 'H':
        Move_Left();
        _delay_ms(50);
        Move_Backward();
        _delay_ms(50);
        break;
    case 'J':
        Move_Right();
        _delay_ms(50);
        Move_Backward();
        _delay_ms(50);
        break;
    case 'S':
        stop();
        break;
    case '1':
        Set_Speed(10);
        break;
    case '2':
        Set_Speed(20);
        break;
    case '3':
        Set_Speed(30);
        break;
    case '4':
        Set_Speed(40);
        break;
    case '5':
        Set_Speed(50);
        break;
    case '6':
        Set_Speed(60);
        break;
    case '7':
        Set_Speed(70);
        break;
    case '8':
        Set_Speed(80);
        break;
    case '9':
        Set_Speed(90);
        break;
    case 'q':
        Set_Speed(100);
        break;
    default:
        stop();
        break;
    }
}

void Desicion_Direction()
{
    if (Ultra_Distance > 10)
    {
        HC5_input = 'F';
    }
    else
    {
        uint8 Right_val = 0, Left_val = 0;

        HC5_input = 'R';
        ULTRA_Desicion();

        Right_val = Ultrasonic_readDistance();

        HC5_input = 'L';
        ULTRA_Desicion();
        ULTRA_Desicion();

        Left_val = Ultrasonic_readDistance();

        HC5_input = 'R';
        ULTRA_Desicion();

        if (Right_val >= Left_val && Right_val > 10)
        {
            HC5_input = 'R';
        }
        else if (Left_val >= Right_val && Left_val > 10)
        {
            HC5_input = 'L';
        }
        else
        {
            HC5_input = 'R';
            ULTRA_Desicion();
            ULTRA_Desicion();

            HC5_input = 'F';
        }

    }
}

void ULTRA_Desicion()
{

    switch (HC5_input)
    {
    case 'F':
        Move_Forward();
        break;
    case 'B':
        Move_Backward();
        break;
    case 'L':
        Move_Left();
        _delay_ms(500);
        break;
    case 'R':
        Move_Right();
        _delay_ms(500);
        break;
    default:
        stop();
        break;
    }

}
```

10. 2 GPIO Driver

➢ gpio_private.h

```
#ifndef GPIO_PRIVATE_H_
#define GPIO_PRIVATE_H_

#include "../../LIB/std_types.h"

/*******************************************************************************
 *                 GPIO Registers type structure declarations                  *
 *******************************************************************************/
/* Bitmap structure for PORTA register */
typedef union
{
    uint8 Byte; /* All register bits */
    struct
    {
        uint8 PA0_Bit :1; /* Register bit 0 */
        uint8 PA1_Bit :1; /* Register bit 1 */
        uint8 PA2_Bit :1; /* Register bit 2 */
        uint8 PA3_Bit :1; /* Register bit 3 */
        uint8 PA4_Bit :1; /* Register bit 4 */
        uint8 PA5_Bit :1; /* Register bit 5 */
        uint8 PA6_Bit :1; /* Register bit 6 */
        uint8 PA7_Bit :1; /* Register bit 7 */
    } Bits;
} GPIO_PORTA_Type;
```

```c
/* Bitmap structure for PORTB register */
typedef union
{
    uint8 Byte; /* All register bits */
    struct
    {
        uint8 PB0_Bit :1; /* Register bit 0 */
        uint8 PB1_Bit :1; /* Register bit 1 */
        uint8 PB2_Bit :1; /* Register bit 2 */
        uint8 PB3_Bit :1; /* Register bit 3 */
        uint8 PB4_Bit :1; /* Register bit 4 */
        uint8 PB5_Bit :1; /* Register bit 5 */
        uint8 PB6_Bit :1; /* Register bit 6 */
        uint8 PB7_Bit :1; /* Register bit 7 */
    } Bits;
} GPIO_PORTB_Type;

/* Bitmap structure for PORTC register */
typedef union
{
    uint8 Byte; /* All register bits */
    struct
    {
        uint8 PC0_Bit :1; /* Register bit 0 */
        uint8 PC1_Bit :1; /* Register bit 1 */
        uint8 PC2_Bit :1; /* Register bit 2 */
        uint8 PC3_Bit :1; /* Register bit 3 */
        uint8 PC4_Bit :1; /* Register bit 4 */
        uint8 PC5_Bit :1; /* Register bit 5 */
        uint8 PC6_Bit :1; /* Register bit 6 */
        uint8 PC7_Bit :1; /* Register bit 7 */
    } Bits;
} GPIO_PORTC_Type;

/* Bitmap structure for PORTD register */
typedef union
{
    uint8 Byte; /* All register bits */
    struct
    {
        uint8 PD0_Bit :1; /* Register bit 0 */
        uint8 PD1_Bit :1; /* Register bit 1 */
        uint8 PD2_Bit :1; /* Register bit 2 */
        uint8 PD3_Bit :1; /* Register bit 3 */
        uint8 PD4_Bit :1; /* Register bit 4 */
        uint8 PD5_Bit :1; /* Register bit 5 */
        uint8 PD6_Bit :1; /* Register bit 6 */
        uint8 PD7_Bit :1; /* Register bit 7 */
    } Bits;
} GPIO_PORTD_Type;

/*******************************************************************************
 *                  SREG Register type structure declarations                  *
 *******************************************************************************/

/* Bitmap structure for SREG register */
typedef union
{
    uint8 Byte; /* All register bits */
    struct
    {
        uint8 C_Bit :1; /* C bit 0 */
        uint8 Z_Bit :1; /* Z bit 1 */
        uint8 N_Bit :1; /* N bit 2 */
        uint8 V_Bit :1; /* V bit 3 */
        uint8 S_Bit :1; /* S bit 4 */
        uint8 H_Bit :1; /* H bit 5 */
        uint8 T_Bit :1; /* T bit 6 */
        uint8 I_Bit :1; /* I bit 7 */
    } Bits;
} SREG_Type;

/*******************************************************************************
 *                         GPIO Registers Definitions                          *
 *******************************************************************************/
/* definition for PORTA registers */
#define PORTA_REG (*(volatile GPIO_PORTA_Type *const)     0x3B)
#define DDRA_REG (*(volatile GPIO_PORTA_Type *const)      0x3A)
#define PINA_REG (*(volatile const GPIO_PORTA_Type *const)0x39)

/* definition for PORTB registers */
#define PORTB_REG (*(volatile GPIO_PORTB_Type *const)     0x38)
#define DDRB_REG (*(volatile GPIO_PORTB_Type *const)      0x37)
#define PINB_REG (*(volatile const GPIO_PORTB_Type *const)0x36)

/* definition for PORTC registers */
#define PORTC_REG (*(volatile GPIO_PORTC_Type *const)     0x35)
#define DDRC_REG (*(volatile GPIO_PORTC_Type *const)      0x34)
#define PINC_REG (*(volatile const GPIO_PORTC_Type *const)0x33)

/* definition for PORTD registers */
```

```
#define PORTD_REG (*(volatile GPIO_PORTD_Type *const)     0x32)
#define DDRD_REG (*(volatile GPIO_PORTD_Type *const)      0x31)
#define PIND_REG (*(volatile const GPIO_PORTD_Type *const)0x30)

/********************************************************************************
 *                          SREG Register Definitions                           *
 ********************************************************************************/

/* definition for SREG registers */
#define SREG_REG (*(volatile SREG_Type *const) 0x5F)
#endif /* GPIO_PRIVATE_H_ */
```

- gpio.h

```
#ifndef GPIO_H_
#define GPIO_H_

#include "../../LIB/std_types.h"

/********************************************************************************
 *                                 Definitions                                  *
 ********************************************************************************/
#define NUM_OF_PORTS           4
#define NUM_OF_PINS_PER_PORT   8

#define PORTA_ID               0
#define PORTB_ID               1
#define PORTC_ID               2
#define PORTD_ID               3

#define PIN0_ID                0
#define PIN1_ID                1
#define PIN2_ID                2
#define PIN3_ID                3
#define PIN4_ID                4
#define PIN5_ID                5
#define PIN6_ID                6
#define PIN7_ID                7

/********************************************************************************
 *                              Types Declaration                               *
 ********************************************************************************/
typedef enum
{
    PIN_INPUT, PIN_OUTPUT
} GPIO_PinDirectionType;
```

```
typedef enum
{
    PORT_INPUT, PORT_OUTPUT = 0xFF
} GPIO_PortDirectionType;

/********************************************************************************
 *                              Functions Prototypes                            *
 ********************************************************************************/

/*
 * Description :
 * Setup the direction of the required pin input/output.
 * If the input port number or pin number are not correct, The function will not handle
the request.
 */
void GPIO_setupPinDirection(uint8 port_num, uint8 pin_num,
        GPIO_PinDirectionType direction);

/*
 * Description :
 * Write the value Logic High or Logic Low on the required pin.
 * If the input port number or pin number are not correct, The function will not handle
the request.
 * If the pin is input, this function will enable/disable the internal pull-up resistor.
 */
void GPIO_writePin(uint8 port_num, uint8 pin_num, uint8 value);

/*
 * Description :
 * Read and return the value for the required pin, it should be Logic High or Logic Low.
 * If the input port number or pin number are not correct, The function will return
Logic Low.
 */
uint8 GPIO_readPin(uint8 port_num, uint8 pin_num);

/*
 * Description :
 * Setup the direction of the required port all pins input/output.
 * If the direction value is PORT_INPUT all pins in this port should be input pins.
 * If the direction value is PORT_OUTPUT all pins in this port should be output pins.
 * If the input port number is not correct, The function will not handle the request.
 */
void GPIO_setupPortDirection(uint8 port_num, GPIO_PortDirectionType direction);

/*
 * Description :
```

```
 * Write the value on the required port.
 * If any pin in the port is output pin the value will be written.
 * If any pin in the port is input pin this will activate/deactivate the internal pull-up resistor.
 * If the input port number is not correct, The function will not handle the request.
 */
void GPIO_writePort(uint8 port_num, uint8 value);

/*
 * Description :
 * Read and return the value of the required port.
 * If the input port number is not correct, The function will return ZERO value.
 */
uint8 GPIO_readPort(uint8 port_num);

#endif /* GPIO_H_ */
```

➢ gpio.c

```
#include "gpio.h"
#include "../../LIB/common_macros.h" /* To use the macros like SET_BIT */
#include "gpio_private.h" /* To use the IO Ports Registers */
#include "../../LIB/std_types.h"

/*
 * Description :
 * Setup the direction of the required pin input/output.
 * If the input port number or pin number are not correct, The function will not handle the request.
 */
void GPIO_setupPinDirection(uint8 port_num, uint8 pin_num,
        GPIO_PinDirectionType direction)
{
    /*
     * Check if the input port number is greater than NUM_OF_PINS_PER_PORT value.
     * Or if the input pin number is greater than NUM_OF_PINS_PER_PORT value.
     * In this case the input is not valid port/pin number
     */
    if ((pin_num >= NUM_OF_PINS_PER_PORT) || (port_num >= NUM_OF_PORTS))
    {
        /* Do Nothing */
    }
    else
    {
        /* Setup the pin direction as required */
        switch (port_num)
        {
        case PORTA_ID:
            if (direction == PIN_OUTPUT)
            {
                SET_BIT(DDRA_REG.Byte, pin_num);
            }
            else
            {
                CLEAR_BIT(DDRA_REG.Byte, pin_num);
            }
            break;
        case PORTB_ID:
            if (direction == PIN_OUTPUT)
            {
                SET_BIT(DDRB_REG.Byte, pin_num);
            }
            else
            {
                CLEAR_BIT(DDRB_REG.Byte, pin_num);
            }
            break;
        case PORTC_ID:
            if (direction == PIN_OUTPUT)
            {
                SET_BIT(DDRC_REG.Byte, pin_num);
            }
            else
            {
                CLEAR_BIT(DDRC_REG.Byte, pin_num);
            }
            break;
        case PORTD_ID:
            if (direction == PIN_OUTPUT)
            {
                SET_BIT(DDRD_REG.Byte, pin_num);
            }
            else
            {
                CLEAR_BIT(DDRD_REG.Byte, pin_num);
            }
            break;
        }
    }
}

/*
```

```
 * Description :
 * Write the value Logic High or Logic Low on the required pin.
 * If the input port number or pin number are not correct, The function will not handle the request.
 * If the pin is input, this function will enable/disable the internal pull-up resistor.
 */
void GPIO_writePin(uint8 port_num, uint8 pin_num, uint8 value)
{
    /*
     * Check if the input port number is greater than NUM_OF_PINS_PER_PORT value.
     * Or if the input pin number is greater than NUM_OF_PINS_PER_PORT value.
     * In this case the input is not valid port/pin number
     */
    if ((pin_num >= NUM_OF_PINS_PER_PORT) || (port_num >= NUM_OF_PORTS))
    {
        /* Do Nothing */
    }
    else
    {
        /* Setup the pin value as required */
        switch (port_num)
        {
        case PORTA_ID:
            if (value == LOGIC_LOW)
            {
                CLEAR_BIT(PORTA_REG.Byte, pin_num);
            }
            else
            {
                SET_BIT(PORTA_REG.Byte, pin_num);
            }
            break;
        case PORTB_ID:
            if (value == LOGIC_LOW)
            {
                CLEAR_BIT(PORTB_REG.Byte, pin_num);
            }
            else
            {
                SET_BIT(PORTB_REG.Byte, pin_num);
            }
            break;
        case PORTC_ID:
            if (value == LOGIC_LOW)
            {
                CLEAR_BIT(PORTC_REG.Byte, pin_num);
            }
            else
            {
                SET_BIT(PORTC_REG.Byte, pin_num);
            }
            break;
        case PORTD_ID:
            if (value == LOGIC_LOW)
            {
                CLEAR_BIT(PORTD_REG.Byte, pin_num);
            }
            else
            {
                SET_BIT(PORTD_REG.Byte, pin_num);
            }
            break;
        }
    }
}

/*
 * Description :
 * Read and return the value for the required pin, it should be Logic High or Logic Low.
 * If the input port number or pin number are not correct, The function will return
Logic Low.
 */
uint8 GPIO_readPin(uint8 port_num, uint8 pin_num)
{
    uint8 value = LOGIC_LOW;
    /*
     * Check if the input port number is greater than NUM_OF_PINS_PER_PORT value.
     * Or if the input pin number is greater than NUM_OF_PINS_PER_PORT value.
     * In this case the input is not valid port/pin number
     */
    if ((pin_num >= NUM_OF_PINS_PER_PORT) || (port_num >= NUM_OF_PORTS))
    {
        /* Do Nothing */
    }
    else
    {
        /* Return the pin value */
        switch (port_num)
        {
        case PORTA_ID:
            if (BIT_IS_SET(PINA_REG.Byte, pin_num))
            {
                value = LOGIC_HIGH;
            }
            else
            {
                value = LOGIC_LOW;
            }
            break;
        case PORTB_ID:
            if (BIT_IS_SET(PINB_REG.Byte, pin_num))
            {
                value = LOGIC_HIGH;
            }
            else
            {
                value = LOGIC_LOW;
            }
            break;
        case PORTC_ID:
            if (BIT_IS_SET(PINC_REG.Byte, pin_num))
            {
                value = LOGIC_HIGH;
            }
            else
            {
                value = LOGIC_LOW;
            }
            break;
        case PORTD_ID:
            if (BIT_IS_SET(PIND_REG.Byte, pin_num))
            {
                value = LOGIC_HIGH;
            }
            else
            {
                value = LOGIC_LOW;
            }
            break;
        }
    }
    return value;
}

/*
 * Description :
 * Setup the direction of the required port all pins input/output.
 * If the direction value is PORT_INPUT all pins in this port should be input pins.
 * If the direction value is PORT_OUTPUT all pins in this port should be output pins.
 * If the input port number is not correct, The function will not handle the request.
 */
void GPIO_setupPortDirection(uint8 port_num, GPIO_PortDirectionType direction)
{
    /*
     * Check if the input number is greater than NUM_OF_PORTS value.
     * In this case the input is not valid port number
     */
    if (port_num >= NUM_OF_PORTS)
    {
        /* Do Nothing */
    }
    else
    {
        /* Setup the port direction as required */
        switch (port_num)
        {
        case PORTA_ID:
            DDRA_REG.Byte = direction;
            break;
        case PORTB_ID:
            DDRB_REG.Byte = direction;
            break;
        case PORTC_ID:
            DDRC_REG.Byte = direction;
            break;
        case PORTD_ID:
            DDRD_REG.Byte = direction;
            break;
        }
    }
}

/*
 * Description :
 * Write the value on the required port.
 * If any pin in the port is output pin the value will be written.
 * If any pin in the port is input pin this will activate/deactivate the internal pull-
up resistor.
 * If the input port number is not correct, The function will not handle the request.
 */
void GPIO_writePort(uint8 port_num, uint8 value)
{
    /*
     * Check if the input number is greater than NUM_OF_PORTS value.
     * In this case the input is not valid port number
     */
    if (port_num >= NUM_OF_PORTS)
    {
        /* Do Nothing */
    }
    else
    {
        /* Setup the port value as required */
        switch (port_num)
        {
        case PORTA_ID:
            PORTA_REG.Byte = value;
            break;
        case PORTB_ID:
            PORTB_REG.Byte = value;
            break;
        case PORTC_ID:
            PORTC_REG.Byte = value;
            break;
        case PORTD_ID:
            PORTD_REG.Byte = value;
            break;
        }
    }
}

/*
 * Description :
 * Read and return the value of the required port.
 * If the input port number is not correct, The function will return ZERO value.
 */
uint8 GPIO_readPort(uint8 port_num)
{
    uint8 value = LOGIC_LOW;
    /*
     * Check if the input number is greater than NUM_OF_PORTS value.
     * In this case the input is not valid port number
     */
    if (port_num >= NUM_OF_PORTS)
    {
        return 0;
    }
    else
    {
        /* Return the pin value */
        switch (port_num)
        {
        case PORTA_ID:
            value = PINA_REG.Byte;
            break;
        case PORTB_ID:
            value = PINB_REG.Byte;
            break;
        case PORTC_ID:
            value = PINC_REG.Byte;
            break;
        case PORTD_ID:
            value = PIND_REG.Byte;
            break;
        }
    }
    return value;
}
```

10. 3 TIMER0 Driver

- timer0_private.h

```
#ifndef TIMER0_PRIVATE_H_
#define TIMER0_PRIVATE_H_

#include "../../LIB/std_types.h"

/******************************************************************************
 *                Timer0 Registers type structure declarations                 *
 *******************************************************************************/

/* Bitmap structure for TCCR0 register */
typedef union
{
    uint8 Byte; /* All register bits */
    struct
    {
        uint8 CS00_Bit :1; /* CS00 bit 0 */
        uint8 CS01_Bit :1; /* CS01 bit 1 */
        uint8 CS02_Bit :1; /* CS02 bit 2 */
        uint8 WGM01_Bit :1; /* WGM01 bit 3 */
        uint8 COM00_Bit :1; /* COM00 bit 4 */
        uint8 COM01_Bit :1; /* COM01 bit 5 */
        uint8 WGM00_Bit :1; /* WGM00 bit 6 */
        uint8 FOC0_Bit :1; /* FOC0 bit 7 */
    } Bits;
} Timer0_TCCR0_Type;

/*******************************************************************************
 *                        Timer0 Registers Definitions                         *
 *******************************************************************************/

/* definition for TCCR0 register */
#define TCCR0_REG (*(volatile Timer0_TCCR0_Type *const) 0x53)
/* definition for TCNT0 registers */
#define TCNT0_REG (*(volatile uint8 *const) 0x52)

/* definition for OCR0 registers */
#define OCR0_REG  (*(volatile uint8 *const) 0x5C)

/*******************************************************************************
 *            Timers Interrupt Register type structure declarations            *
 *******************************************************************************/

/* Bitmap structure for TIMSK register */
typedef union
{
    uint8 Byte; /* All register bits */
    struct
    {
        uint8 TOIE0_Bit :1; /* TOIE0 bit 0 */
        uint8 OCIE0_Bit :1; /* OCIE0 bit 1 */
        uint8 TOIE1_Bit :1; /* TOIE1 bit 2 */
        uint8 OCIE1B_Bit :1; /* OCIE1B bit 3 */
        uint8 OCIE1A_Bit :1; /* OCIE1A bit 4 */
        uint8 TICIE1_Bit :1; /* TICIE1 bit 5 */
        uint8 TOIE2_Bit :1; /* TOIE2 bit 6 */
        uint8 OCIE2_Bit :1; /* OCIE2 bit 7 */
    } Bits;
} Timers_TIMSK_Type;

/* Bitmap structure for TIFR register */
typedef union
{
    uint8 Byte; /* All register bits */
    struct
    {
        uint8 TOV0_Bit :1; /* TOV0 bit 0 */
        uint8 OCF0_Bit :1; /* OCF0 bit 1 */
        uint8 TOV1_Bit :1; /* TOV1 bit 2 */
        uint8 OCF1B_Bit :1; /* OCF1B bit 3 */
        uint8 OCF1A_Bit :1; /* OCF1A bit 4 */
        uint8 ICF1_Bit :1; /* ICF1 bit 5 */
        uint8 TOV2_Bit :1; /* TOV2 bit 6 */
        uint8 OCF2_Bit :1; /* OCF2 bit 7 */
    } Bits;
} Timers_TIFR_Type;

/*******************************************************************************
 *                     Timers Interrupt Register Definitions                    *
 *******************************************************************************/

/* definition for TIMSK register */
#ifndef TIMSK_REG
#define TIMSK_REG (*(volatile Timers_TIMSK_Type *const) 0x59)
#endif

/* definition for TIFR register */
#ifndef TIFR_REG
#define TIFR_REG  (*(volatile Timers_TIFR_Type *const)  0x58)
#endif

#endif /* TIMER0_PRIVATE_H_ */
```

➢ timer0.h

```
#ifndef TIMER0_H_
#define TIMER0_H_

#include "../../LIB/std_types.h"

/*******************************************************************************
 *                                Definitions                                   *
 *******************************************************************************/

#define OC0_PORTID      PORTB_ID
#define OC0_PINID       PIN3_ID

#define OC0_MAX_VALUE   255

/*******************************************************************************
 *                             Types Declaration                                *
 *******************************************************************************/

typedef enum
{
    NORMAL_MODE, PHASE_CORRECT_MODE, CTC_MODE, FAST_PWM_MODE
```

```c
} TIMER0_ModeType;

typedef enum
{
    NORMAL_MODE_OC0_DISCONNECTED,
    CTC_TOGGLE__PWM_RESERVED,
    CTC_CLEAR__PWM_NON_INVERTING,
    CTC_SET__PWM_INVERTING
} TIMER0_CompareOutputType;
typedef enum
{
    NO_CLOCK_SOURCE,
    PRESCALER_1,
    PRESCALER_8,
    PRESCALER_64,
    PRESCALER_256,
    PRESCALER_1024,
    EXTERNAL_SOURCE_FALLING_EDGE,
    EXTERNAL_SOURCE_RISING_EDGE
} TIMER0_PrescalerType;

typedef struct
{
    uint8 initial_value;
    uint8 compare_value; /* for CTC mode only */
    TIMER0_ModeType mode;
    TIMER0_PrescalerType prescaler;
    TIMER0_CompareOutputType outputMode;
} TIMER0_ConfigType;

/*******************************************************************************
 *                       Functions Prototypes                                  *
 *******************************************************************************/

/*
 * Description :
 * Function to initialize the Timer driver.
 */
void Timer0_init(const TIMER0_ConfigType *Config_Ptr);

/*
 * Description :
 * Function to disable the Timer0.
 */
void Timer0_deInit(void);
```

```
/*
 * Description :
 * Function to set the Call Back function address.
 */
void Timer0_setCallBack(void (*a_ptr)(void));

/*
 * Description:
 * Setup the compare value based on the required input duty cycle.
 */
void Timer0_PWM_Start(uint8 a_dutyCycle);

#endif /* TIMER0_H_ */
```

- timer0.c

```
#include "timer0.h"
#include "timer0_private.h"
#include <avr/interrupt.h>

/*******************************************************************************
 *                           Global Variables                                  *
 *******************************************************************************/

/* Global variables to hold the address of the call back function in the application */
static volatile void (*g_callBackPtr)(void) = NULL_PTR;

/*******************************************************************************
 *                       Interrupt Service Routines                            *
 *******************************************************************************/

ISR(TIMER0_OVF_vect)
{
    if (g_callBackPtr != NULL_PTR)
    {
        /* Call the Call Back function in the application after the edge is detected */
        (*g_callBackPtr)();
    }
}

ISR(TIMER0_COMP_vect)
{
    if (g_callBackPtr != NULL_PTR)
    {
        /* Call the Call Back function in the application after the edge is detected */
        (*g_callBackPtr)();
    }
}

/*******************************************************************************
 *                      Functions Definitions                                  *
 *******************************************************************************/

/*
 * Description :
 * Function to initialize the Timer driver.
 */
void Timer0_init(const TIMER0_ConfigType *Config_Ptr)
{
    /************************** TCCR0 Description **************************
     * FOC0:      Force Output Compare for Compare unit (non-PWM mode)
     * WGM01:0    Waveform Generation Mode, selected in configuration
     * COM01:0    Compare Match Output Mode, selected in configuration
     * CS02:0     Clock Select, selected in configuration
     ***********************************************************************/

    /*
     * insert the required mode in WGM bits (WGM00 and WGM01) of TCCR0 Register
     */
    if (Config_Ptr->mode == NORMAL_MODE)
    {
        TCCR0_REG.Bits.FOC0_Bit = 1;
        TCCR0_REG.Bits.WGM00_Bit = 0;
        TCCR0_REG.Bits.WGM01_Bit = 0;
    }
    if (Config_Ptr->mode == CTC_MODE)
    {
        TCCR0_REG.Bits.FOC0_Bit = 1;
        TCCR0_REG.Bits.WGM00_Bit = 0;
        TCCR0_REG.Bits.WGM01_Bit = 1;
    }
    if (Config_Ptr->mode == FAST_PWM_MODE)
    {
        TCCR0_REG.Bits.FOC0_Bit = 0;
        TCCR0_REG.Bits.WGM00_Bit = 1;
        TCCR0_REG.Bits.WGM01_Bit = 1;
    }

    /*
     * insert the required compare output mode in COM bits (COM00 and COM01) of TCCR0
Register
     */
    if (Config_Ptr->outputMode == NORMAL_MODE_OC0_DISCONNECTED)
    {
        TCCR0_REG.Bits.COM00_Bit = 0;
        TCCR0_REG.Bits.COM01_Bit = 0;
    }
    if (Config_Ptr->outputMode == CTC_TOGGLE__PWM_RESERVED)
    {
        TCCR0_REG.Bits.COM00_Bit = 1;
        TCCR0_REG.Bits.COM01_Bit = 0;
    }
    if (Config_Ptr->outputMode == CTC_CLEAR__PWM_NON_INVERTING)
    {
        TCCR0_REG.Bits.COM00_Bit = 0;
        TCCR0_REG.Bits.COM01_Bit = 1;
    }
    if (Config_Ptr->outputMode == CTC_SET__PWM_INVERTING)
    {
        TCCR0_REG.Bits.COM00_Bit = 1;
        TCCR0_REG.Bits.COM01_Bit = 1;
    }

    /*
     * insert the required prescaler in CS bits (CS00, CS01 and CS02) of TCCR0 Register
     */
    TCCR0_REG.Byte = (TCCR0_REG.Byte & 0xF8) | (Config_Ptr->prescaler);

    TCNT0_REG = Config_Ptr->initial_value;
    OCR0_REG = Config_Ptr->compare_value;

    /* Enable Timer/Counter0 Output Compare Match interrupt */
    TIMSK_REG.Bits.OCIE0_Bit = 1;

    /* Enable Timer/Counter0 Overflow Interrupt */
    TIMSK_REG.Bits.TOIE0_Bit = 1;
}

/*
 * Description :
 * Function to disable the Timer0.
 */
void Timer0_deInit(void)
{
    /* Clear All Timer1 Registers */
    TCCR0_REG.Byte = 0;
    TCNT0_REG = 0;
    OCR0_REG = 0;
```

```c
    /* Disable the Output Compare A match and Overflow interrupt */
    TIMSK_REG.Bits.OCIE0_Bit = 0;
    TIMSK_REG.Bits.TOIE0_Bit = 0;

    /* Reset the global pointer value */
    g_callBackPtr = NULL_PTR;
}

/*
 * Description :
 * Function to set the Call Back function address.
 */
void Timer0_setCallBack(void (*a_ptr)(void))
{
    /* Save the address of the Call back function in a global variable */
    g_callBackPtr = a_ptr;
}

/*
 * Description:
 * Setup the compare value based on the required input duty cycle.
 */
void Timer0_PWM_Start(uint8 a_dutyCycle)
{
    /* If the input duty cycle greater than 100, then set it to 100 */
    if (a_dutyCycle > 100)
    {
        a_dutyCycle = 100;
    }
    /* If the input duty cycle less than 0, then set it to 0 */
    if (a_dutyCycle < 0)
    {
        a_dutyCycle = 0;
    }
    OCR0_REG = (uint8) ((uint16) a_dutyCycle * OC0_MAX_VALUE / 100);
}
```

10. 4 UART Driver

➢ uart.h

```c
#include "../../LIB/std_types.h"

/*******************************************************************************
 *                        Functions Prototypes                                 *
 *******************************************************************************/

/*
 * Description :
 * Functional responsible for Initialize the UART device by:
 * 1. Setup the Frame format like number of data bits, parity bit type and number of
stop bits.
 * 2. Enable the UART.
 * 3. Setup the UART baud rate.
 */
void UART_init(uint32 baud_rate);

/*
 * Description :
 * Functional responsible for send byte to another UART device.
 */
void UART_sendByte(const uint8 data);

/*
 * Description :
 * Functional responsible for receive byte from another UART device.
 */
uint8 UART_recieveByte(void);

/*
 * Description :
 * Send the required string through UART to the other UART device.
 */
void UART_sendString(const uint8 *Str);

/*
 * Description :
 * Receive the required string until the '#' symbol through UART from the other UART
device.
 */
void UART_receiveString(uint8 *Str); // Receive until #

#endif /* UART_H_ */
```

- ➢ uart.c

```
#include "uart.h"
#include "avr/io.h" /* To use the UART Registers */
#include "../../LIB/common_macros.h" /* To use the macros like SET_BIT */

/*******************************************************************************
```

```
 *                       Functions Definitions                                  *
 *******************************************************************************/

/*
 * Description :
 * Functional responsible for Initialize the UART device by:
 * 1. Setup the Frame format like number of data bits, parity bit type and number of stop bits.
 * 2. Enable the UART.
 * 3. Setup the UART baud rate.
 */
void UART_init(uint32 baud_rate)
{
    uint16 ubrr_value = 0;

    /* U2X = 1 for double transmission speed */
    UCSRA = (1 << U2X);

    /************************** UCSRB Description **************************
     * RXCIE = 0 Disable USART RX Complete Interrupt Enable
     * TXCIE = 0 Disable USART Tx Complete Interrupt Enable
     * UDRIE = 0 Disable USART Data Register Empty Interrupt Enable
     * RXEN  = 1 Receiver Enable
     * RXEN  = 1 Transmitter Enable
     * UCSZ2 = 0 For 8-bit data mode
     * RXB8 & TXB8 not used for 8-bit data mode
     ***********************************************************************/
    UCSRB = (1 << RXEN) | (1 << TXEN);

    /************************** UCSRC Description **************************
     * URSEL   = 1 The URSEL must be one when writing the UCSRC
     * UMSEL   = 0 Asynchronous Operation
     * UPM1:0  = 00 Disable parity bit
     * USBS    = 0 One stop bit
     * UCSZ1:0 = 11 For 8-bit data mode
     * UCPOL   = 0 Used with the Synchronous operation only
     ***********************************************************************/
    UCSRC = (1 << URSEL) | (1 << UCSZ0) | (1 << UCSZ1);

    /* Calculate the UBRR register value */
    ubrr_value = (uint16) (((F_CPU / (baud_rate * 8UL))) - 1);

    /* First 8 bits from the BAUD_PRESCALE inside UBRRL and last 4 bits in UBRRH*/
    UBRRH = ubrr_value >> 8;
    UBRRL = ubrr_value;
}
```

```
/*
 * Description :
 * Functional responsible for send byte to another UART device.
 */
void UART_sendByte(const uint8 data)
{
    /*
     * UDRE flag is set when the Tx buffer (UDR) is empty and ready for
     * transmitting a new byte so wait until this flag is set to one
     */
    while (BIT_IS_CLEAR(UCSRA, UDRE))
    {
    }

    /*
     * Put the required data in the UDR register and it also clear the UDRE flag as
     * the UDR register is not empty now
     */
    UDR = data;

    /************************* Another Method *************************
     UDR = data;
     while(BIT_IS_CLEAR(UCSRA,TXC)){} // Wait until the transmission is complete TXC = 1
     SET_BIT(UCSRA,TXC); // Clear the TXC flag
     *******************************************************************/
}

/*
 * Description :
 * Functional responsible for receive byte from another UART device.
 */
uint8 UART_recieveByte(void)
{
    /* RXC flag is set when the UART receive data so wait until this flag is set to one
*/
    while (BIT_IS_CLEAR(UCSRA, RXC))
    {
    }

    /*
     * Read the received data from the Rx buffer (UDR)
     * The RXC flag will be cleared after read the data
     */
    return UDR;
}
```

```
/*
 * Description :
 * Send the required string through UART to the other UART device.
 */
void UART_sendString(const uint8 *Str)
{
    uint8 i = 0;

    /* Send the whole string */
    while (Str[i] != '\0')
    {
        UART_sendByte(Str[i]);
        i++;
    }
    /************************* Another Method *************************
     while(*Str != '\0')
     {
     UART_sendByte(*Str);
     Str++;
     }
     *******************************************************************/
}

/*
 * Description :
 * Receive the required string until the '#' symbol through UART from the other UART
device.
 */
void UART_receiveString(uint8 *Str)
{
    uint8 i = 0;

    /* Receive the first byte */
    Str[i] = UART_recieveByte();

    /* Receive the whole string until the '#' */
    while (Str[i] != '#')
    {
        i++;
        Str[i] = UART_recieveByte();
    }

    /* After receiving the whole string plus the '#', replace the '#' with '\0' */
    Str[i] = '\0';
}
```

10. 5 BLUETOOTH Driver

➢ bluetooth.h

```
#include "../../LIB/std_types.h"

#ifndef BLUETOOTH_H_
#define BLUETOOTH_H_

/*******************************************************************************
 *                                Definitions                                  *
 *******************************************************************************/

/*
 * Description :
 * The Function responsible for setup the bluetooth module with suitable baud rate.
 */
void bluetooth_init();

/*
 * Description :
 * The Function responsible for receive data from bluetooth module.
 */
uint8 bluetooth_recieveByte();

/*
 * Description :
 * The Function responsible for send data through bluetooth module.
 */
void bluetooth_sendByte(uint8 data);

#endif /* BLUETOOTH_H_ */
```

➢ bluetooth.c

```
#include "../../MCAL/UART/uart.h"
#include "bluetooth.h"

/*
 * Description :
 * The Function responsible for setup the bluetooth module with suitable baud rate.
 */
void bluetooth_init()
{
    UART_init(9600);
}
```

```
/*
 * Description :
 * The Function responsible for receive data from bluetooth module.
 */
uint8 bluetooth_recieveByte()
{
    return UART_recieveByte();
}

/*
 * Description :
 * The Function responsible for send data through bluetooth module.
 */
void bluetooth_sendByte(uint8 data)
{
    UART_sendByte(data);
}
```

10. 6 DC MOTOR Driver

- motor.h

```
#ifndef MOTOR_H_
#define MOTOR_H_

#include "../../LIB/std_types.h"

/*******************************************************************************
 *                         Types Declaration                                   *
 *******************************************************************************/

typedef enum
{
    STOP, ANTI_CLOCK_WISE, CLOCK_Wise
} DcMotor_State;

/*******************************************************************************
 *                                Definitions                                  *
 *******************************************************************************/

#define MOTOR1_PORTID               PORTA_ID
#define MOTOR1_IN1_PINID            PIN0_ID
#define MOTOR1_IN2_PINID            PIN1_ID

#define MOTOR2_PORTID               PORTA_ID
```

```
#define MOTOR2_IN1_PINID            PIN2_ID
#define MOTOR2_IN2_PINID            PIN3_ID

#define MOTOR_MAX_SPEED         100

/*******************************************************************************
 *                              Functions Prototypes                           *
 *******************************************************************************/

/*
 * Description :
 * 1. The Function responsible for setup the direction for the two motor pins through
the GPIO driver.
 * 2. Stop at the DC-Motor at the beginning through the GPIO driver.
 */
void DcMotor_Init(void);

/*
 * Description :
 * 1. The function responsible for rotate the DC Motor 1 CW/
 *    or A-CW or stop the motor based on the state input state value.
 * 2. Send the required duty cycle to the PWM driver based on the required speed value.
 */
void DcMotor1_Rotate(DcMotor_State a_state);

/*
 * Description :
 * 1. The function responsible for rotate the DC Motor 2 CW/
 *    or A-CW or stop the motor based on the state input state value.
 * 2. Send the required duty cycle to the PWM driver based on the required speed value.
 */
void DcMotor2_Rotate(DcMotor_State a_state);

#endif /* MOTOR_H_ */
```

motor.c

```
#include "../../MCAL/GPIO/gpio.h"
#include "../../MCAL/TIMER0/timer0.h"
#include "motor.h"
#include "../../LIB/common_macros.h"

/*
 * Description :
 * 1. The Function responsible for setup the direction for the two motor pins through
the GPIO driver.
```

```
 * 2. Stop at the DC-Motor at the beginning through the GPIO driver.
 */
void DcMotor_Init(void)
{
    /* Make The Configuration Of Timer0 To Be in FAST PWM Mode */
    TIMER0_ConfigType TIMER0_CONFIG =
    { 0, 0, FAST_PWM_MODE, PRESCALER_64, CTC_CLEAR__PWM_NON_INVERTING };

    /* Init Timer0 in FAST PWM Mode */
    Timer0_init(&TIMER0_CONFIG);
    /* Set OC0 pin direction as output */
    GPIO_setupPinDirection(OC0_PORTID, OC0_PINID, PIN_OUTPUT);

    /* Set IN1 and IN2 motor 1 pins direction as output */
    GPIO_setupPinDirection(MOTOR1_PORTID, MOTOR1_IN1_PINID, PIN_OUTPUT);
    GPIO_setupPinDirection(MOTOR1_PORTID, MOTOR1_IN2_PINID, PIN_OUTPUT);

    /* Set IN1 and IN2 motor 2 pins direction as output */
    GPIO_setupPinDirection(MOTOR2_PORTID, MOTOR2_IN1_PINID, PIN_OUTPUT);
    GPIO_setupPinDirection(MOTOR2_PORTID, MOTOR2_IN2_PINID, PIN_OUTPUT);

    /* Stop motor 1 as initial state */
    GPIO_writePin(MOTOR1_PORTID, MOTOR1_IN1_PINID, LOGIC_LOW);
    GPIO_writePin(MOTOR1_PORTID, MOTOR1_IN2_PINID, LOGIC_LOW);

    /* Stop motor 2 as initial state */
    GPIO_writePin(MOTOR2_PORTID, MOTOR2_IN1_PINID, LOGIC_LOW);
    GPIO_writePin(MOTOR2_PORTID, MOTOR2_IN2_PINID, LOGIC_LOW);

    /* Inital the two motors works at full speed */
    Timer0_PWM_Start(100);
}

/*
 * Description :
 * 1. The function responsible for rotate the DC Motor 1 CW/
 *    or A-CW or stop the motor based on the state input state value.
 * 2. Send the required duty cycle to the PWM driver based on the required speed value.
 */
void DcMotor1_Rotate(DcMotor_State a_state)
{
    switch (a_state)
    {
    case CLOCK_Wise:
        /* Clock wise mode => (IN1 = 1 and INT2 = 0) */
        GPIO_writePin(MOTOR1_PORTID, MOTOR1_IN1_PINID, LOGIC_HIGH);
        GPIO_writePin(MOTOR1_PORTID, MOTOR1_IN2_PINID, LOGIC_LOW);
        break;
    case ANTI_CLOCK_WISE:
        /* Anti clock wise mode => (IN1 = 0 and INT2 = 1) */
        GPIO_writePin(MOTOR1_PORTID, MOTOR1_IN1_PINID, LOGIC_LOW);
        GPIO_writePin(MOTOR1_PORTID, MOTOR1_IN2_PINID, LOGIC_HIGH);
        break;
    default:
        /* Any case else, the motor be in stop mode */
        GPIO_writePin(MOTOR1_PORTID, MOTOR1_IN1_PINID, LOGIC_LOW);
        GPIO_writePin(MOTOR1_PORTID, MOTOR1_IN2_PINID, LOGIC_LOW);
    }
}

/*
 * Description :
 * 1. The function responsible for rotate the DC Motor 2 CW/
 *    or A-CW or stop the motor based on the state input state value.
 * 2. Send the required duty cycle to the PWM driver based on the required speed value.
 */
void DcMotor2_Rotate(DcMotor_State a_state)
{
    switch (a_state)
    {
    case CLOCK_Wise:
        /* Clock wise mode => (IN1 = 1 and INT2 = 0) */
        GPIO_writePin(MOTOR2_PORTID, MOTOR2_IN1_PINID, LOGIC_HIGH);
        GPIO_writePin(MOTOR2_PORTID, MOTOR2_IN2_PINID, LOGIC_LOW);
        break;
    case ANTI_CLOCK_WISE:
        /* Anti clock wise mode => (IN1 = 0 and INT2 = 1) */
        GPIO_writePin(MOTOR2_PORTID, MOTOR2_IN1_PINID, LOGIC_LOW);
        GPIO_writePin(MOTOR2_PORTID, MOTOR2_IN2_PINID, LOGIC_HIGH);
        break;
    default:
        /* Any case else, the motor be in stop mode */
        GPIO_writePin(MOTOR2_PORTID, MOTOR2_IN1_PINID, LOGIC_LOW);
        GPIO_writePin(MOTOR2_PORTID, MOTOR2_IN2_PINID, LOGIC_LOW);
    }
}
```

10. 7 L298N Driver

➢ motorDriver.h

```
#ifndef MOTORDRIVER_H_
```

```
#define MOTORDRIVER_H_

#include "../../LIB/std_types.h"

/*******************************************************************************
 *                              Functions Prototypes                           *
 *******************************************************************************/

/*
 * Description :
 * The Function responsible for moving the car forward.
 */
void Move_Forward();

/*
 * Description :
 * The Function responsible for moving the car backward.
 */
void Move_Backward();

/*
 * Description :
 * The Function responsible for moving the car right.
 */
void Move_Right();

/*
 * Description :
 * The Function responsible for moving the car left.
 */
void Move_Left();

/*
 * Description :
 * The Function responsible for stop the car.
 */
void stop();

/*
 * Description :
 * The Function responsible for setting the car speed.
 */
void Set_Speed(uint8 speed);

#endif /* MOTORDRIVER_H_ */
```

motorDriver.c

```c
#include "motorDriver.h"
#include "../MOTOR/motor.h"
#include "../../MCAL/TIMER0/timer0.h"

/*
 * Description :
 * The Function responsible for moving the car forward.
 */
void Move_Forward()
{
    DcMotor1_Rotate(ANTI_CLOCK_WISE);
    DcMotor2_Rotate(ANTI_CLOCK_WISE);
}

/*
 * Description :
 * The Function responsible for moving the car backward.
 */
void Move_Backward()
{
    DcMotor1_Rotate(CLOCK_Wise);
    DcMotor2_Rotate(CLOCK_Wise);
}

/*
 * Description :
 * The Function responsible for moving the car right.
 */
void Move_Right()
{
    DcMotor1_Rotate(CLOCK_Wise);
    DcMotor2_Rotate(ANTI_CLOCK_WISE);
}

/*
 * Description :
 * The Function responsible for moving the car left.
 */
void Move_Left()
{
    DcMotor1_Rotate(ANTI_CLOCK_WISE);
    DcMotor2_Rotate(CLOCK_Wise);
}
```

```
/*
 * Description :
 * The Function responsible for stop the car.
 */
void stop()
{
    DcMotor1_Rotate(STOP);
    DcMotor2_Rotate(STOP);
}

/*
 * Description :
 * The Function responsible for setting the car speed.
 */
void Set_Speed(uint8 speed)
{
    Timer0_PWM_Start(speed);
}
```

10. 8 ICU Driver

➢ timer1_private.h

```
#ifndef TIMER_PRIVATE_H_
#define TIMER_PRIVATE_H_

#include "../../LIB/std_types.h"

/*******************************************************************************
 *                Timer1 Registers type structure declarations                 *
 *******************************************************************************/

/* Bitmap structure for TCCR1A register */
typedef union
{
    uint8 Byte; /* All register bits */
    struct
    {
        uint8 WGM10_Bit  : 1;  /* WGM10 bit 0 */
        uint8 WGM11_Bit  : 1;  /* WGM11 bit 1 */
        uint8 FOC1B_Bit  : 1;  /* FOC1B bit 2 */
        uint8 FOC1A_Bit  : 1;  /* FOC1A bit 3 */
        uint8 COM1B0_Bit : 1;  /* COM1B0 bit 4 */
        uint8 COM1B1_Bit : 1;  /* COM1B1 bit 5 */
        uint8 COM1A0_Bit : 1;  /* COM1A0 bit 6 */
        uint8 COM1A1_Bit : 1;  /* COM1A1 bit 7 */
    } Bits;
} Timer1_TCCR1A_Type;

/* Bitmap structure for TCCR1B register */
typedef union
{
    uint8 Byte; /* All register bits */
    struct
    {
        uint8 CS10_Bit  : 1;  /* CS10 bit 0 */
        uint8 CS11_Bit  : 1;  /* CS11 bit 1 */
        uint8 CS12_Bit  : 1;  /* CS12 bit 2 */
        uint8 WGM12_Bit : 1;  /* WGM12 bit 3 */
        uint8 WGM13_Bit : 1;  /* WGM13 bit 4 */
        uint8 : 1;            /* Reserved bit 5 */
        uint8 ICES1_Bit : 1;  /* ICES1 bit 6 */
        uint8 ICNC1_Bit : 1;  /* ICNC1 bit 7 */
    } Bits;
} Timer1_TCCR1B_Type;

/*******************************************************************************
 *                         Timer1 Registers Definitions                           *
 *******************************************************************************/
/* definition for TCCR1A register */
#define TCCR1A_REG (*(volatile Timer1_TCCR1A_Type *const) 0x4F)

/* definition for TCCR1B register */
#define TCCR1B_REG (*(volatile Timer1_TCCR1B_Type *const) 0x4E)

/* definition for TCNT1 registers */
#define TCNT1H_REG (*(volatile uint8 *const)  0x4D)

#define TCNT1L_REG (*(volatile uint8 *const)  0x4C)

#define TCNT1_REG  (*(volatile uint16 *const) 0x4C)

/* definition for OCR1A registers */
#define OCR1AH_REG (*(volatile uint8 *const)  0x4B)

#define OCR1AL_REG (*(volatile uint8 *const)  0x4A)

#define OCR1A_REG  (*(volatile uint16 *const) 0x4A)

/* definition for OCR1B registers */
#define OCR1BH_REG (*(volatile uint8 *const)  0x49)
```

```
#define OCR1BL_REG (*(volatile uint8 *const)  0x48)

#define OCR1B_REG  (*(volatile uint16 *const) 0x48)

/* definition for ICR1 registers */
#define ICR1H_REG (*(volatile uint8 *const)  0x47)

#define ICR1L_REG (*(volatile uint8 *const)  0x46)

#define ICR1_REG  (*(volatile uint16 *const) 0x46)

/********************************************************************************
 *            Timers Interrupt Register type structure declarations             *
 ********************************************************************************/

/* Bitmap structure for TIMSK register */
typedef union
{
    uint8 Byte; /* All register bits */
    struct
    {
        uint8 TOIE0_Bit  : 1;  /* TOIE0 bit 0 */
        uint8 OCIE0_Bit  : 1;  /* OCIE0 bit 1 */
        uint8 TOIE1_Bit  : 1;  /* TOIE1 bit 2 */
        uint8 OCIE1B_Bit : 1;  /* OCIE1B bit 3 */
        uint8 OCIE1A_Bit : 1;  /* OCIE1A bit 4 */
        uint8 TICIE1_Bit : 1;  /* TICIE1 bit 5 */
        uint8 TOIE2_Bit  : 1;  /* TOIE2 bit 6 */
        uint8 OCIE2_Bit  : 1;  /* OCIE2 bit 7 */
    } Bits;
} Timers_TIMSK_Type;

/* Bitmap structure for TIFR register */
typedef union
{
    uint8 Byte; /* All register bits */
    struct
    {
        uint8 TOV0_Bit  : 1;  /* TOV0 bit 0 */
        uint8 OCF0_Bit  : 1;  /* OCF0 bit 1 */
        uint8 TOV1_Bit  : 1;  /* TOV1 bit 2 */
        uint8 OCF1B_Bit : 1;  /* OCF1B bit 3 */
        uint8 OCF1A_Bit : 1;  /* OCF1A bit 4 */
        uint8 ICF1_Bit  : 1;  /* ICF1 bit 5 */
        uint8 TOV2_Bit  : 1;  /* TOV2 bit 6 */
```

```c
        uint8 OCF2_Bit  : 1;  /* OCF2 bit 7 */
    } Bits;
} Timers_TIFR_Type;

/*******************************************************************************
 *                    Timers Interrupt Register Definitions                    *
 *******************************************************************************/

/* definition for TIMSK register */
#define TIMSK_REG (*(volatile Timers_TIMSK_Type *const) 0x59)

/* definition for TIFR register */
#define TIFR_REG  (*(volatile Timers_TIFR_Type *const)  0x58)

#endif /* TIMER_PRIVATE_H_ */
```

- icu.h

```c
#ifndef ICU_H_
#define ICU_H_

#include "../../LIB/std_types.h"

/*******************************************************************************
 *                         Types Declaration                                   *
 *******************************************************************************/
typedef enum
{
    NO_CLOCK, F_CPU_CLOCK, F_CPU_8, F_CPU_64, F_CPU_256, F_CPU_1024
} ICU_ClockType;

typedef enum
{
    FALLING, RAISING
} ICU_EdgeType;

typedef struct
{
    ICU_ClockType clock;
    ICU_EdgeType edge;
} ICU_ConfigType;

/*******************************************************************************
 *                      Functions Prototypes                                   *
 *******************************************************************************/
```

```c
/*
 * Description : Function to initialize the ICU driver
 *  1. Set the required clock.
 *  2. Set the required edge detection.
 *  3. Enable the Input Capture Interrupt.
 *  4. Initialize Timer1 Registers
 */
void ICU_init(const ICU_ConfigType *Config_Ptr);

/*
 * Description: Function to set the Call Back function address.
 */
void ICU_setCallBack(void (*a_ptr)(void));

/*
 * Description: Function to set the required edge detection.
 */
void ICU_setEdgeDetectionType(const ICU_EdgeType edgeType);

/*
 * Description: Function to get the Timer1 Value when the input is captured
 *              The value stored at Input Capture Register ICR1
 */
uint16 ICU_getInputCaptureValue(void);

/*
 * Description: Function to clear the Timer1 Value to start count from ZERO
 */
void ICU_clearTimerValue(void);

/*
 * Description: Function to disable the Timer1 to stop the ICU Driver
 */
void ICU_deInit(void);

#endif /* ICU_H_ */
```

- icu.c

```c
#include "../../MCAL/GPIO/gpio_private.h"   /* To use the IO Ports Registers */
#include "../../LIB/common_macros.h"    /* To use the macros like SET_BIT */

#include "icu.h"
#include "timer1_private.h" /* To use ICU/Timer1 Registers */
#include <avr/interrupt.h>  /* For ICU ISR */
```

```
/*******************************************************************************
 *                             Global Variables                                *
 *******************************************************************************/

/* Global variables to hold the address of the call back function in the application */
static volatile void (*g_callBackPtr)(void) = NULL_PTR;

/*******************************************************************************
 *                         Interrupt Service Routines                          *
 *******************************************************************************/

ISR(TIMER1_CAPT_vect)
{
    if (g_callBackPtr != NULL_PTR)
    {
        /* Call the Call Back function in the application after the edge is detected */
        (*g_callBackPtr)(); /* another method to call the function using pointer to
function g_callBackPtr(); */
    }
}

/*******************************************************************************
 *                         Functions Definitions                               *
 *******************************************************************************/
/*
 * Description : Function to initialize the ICU driver
 *  1. Set the required clock.
 *  2. Set the required edge detection.
 *  3. Enable the Input Capture Interrupt.
 *  4. Initialize Timer1 Registers
 */
void ICU_init(const ICU_ConfigType *Config_Ptr)
{
    /* Configure ICP1/PD6 as i/p pin */
    DDRD_REG.Bits.PD6_Bit = 0;

    /* Timer1 always operates in Normal Mode */
    TCCR1A_REG.Bits.FOC1A_Bit = 1;
    TCCR1A_REG.Bits.FOC1B_Bit = 1;

    /*
     * insert the required clock value in the first three bits (CS10, CS11 and CS12)
     * of TCCR1B Register
     */
    TCCR1B_REG.Byte = (TCCR1B_REG.Byte & 0xF8) | (Config_Ptr->clock);
    /*
```

```
	 * insert the required edge type in ICES1 bit in TCCR1B Register
	 */
	TCCR1B_REG.Byte = (TCCR1B_REG.Byte & 0xBF) | ((Config_Ptr->edge) << 6);

	/* Initial Value for Timer1 */
	TCNT1_REG = 0;

	/* Initial Value for the input capture register */
	ICR1_REG = 0;

	/* Enable the Input Capture interrupt to generate an interrupt when edge is detected
on ICP1/PD6 pin */
	TIMSK_REG.Bits.TICIE1_Bit = 1;
}

/*
 * Description: Function to set the Call Back function address.
 */
void ICU_setCallBack(void (*a_ptr)(void))
{
	/* Save the address of the Call back function in a global variable */
	g_callBackPtr = a_ptr;
}

/*
 * Description: Function to set the required edge detection.
 */
void ICU_setEdgeDetectionType(const ICU_EdgeType a_edgeType)
{
	/*
	 * insert the required edge type in ICES1 bit in TCCR1B Register
	 */
	TCCR1B_REG.Byte = (TCCR1B_REG.Byte & 0xBF) | (a_edgeType << 6);
}

/*
 * Description: Function to get the Timer1 Value when the input is captured
 *              The value stored at Input Capture Register ICR1
 */
uint16 ICU_getInputCaptureValue(void)
{
	return ICR1_REG;
}

/*
 * Description: Function to clear the Timer1 Value to start count from ZERO
 */
void ICU_clearTimerValue(void)
{
    TCNT1_REG = 0;
}

/*
 * Description: Function to disable the Timer1 to stop the ICU Driver
 */
void ICU_deInit(void)
{
    /* Clear All Timer1/ICU Registers */
    TCCR1A_REG.Byte = 0;
    TCCR1B_REG.Byte = 0;
    TCNT1_REG = 0;
    ICR1_REG = 0;

    /* Disable the Input Capture interrupt */
    TIMSK_REG.Bits.TICIE1_Bit = 0;

    /* Reset the global pointer value */
    g_callBackPtr = NULL_PTR;
}
```

10. 9 Ultrasonic Driver

- ultrasonic_sensor.h

```
#ifndef ULTRASONIC_SENSOR_H_
#define ULTRASONIC_SENSOR_H_

#include "../../LIB/std_types.h"

/*******************************************************************************
 *                                Definitions                                  *
 *******************************************************************************/

#define TRIGGER_PORT_ID         PORTB_ID
#define TRIGGER_PIN_ID          PIN5_ID
#define TRIGGER_DELAY_VALUE     20

/*******************************************************************************
 *                              Functions Prototypes                           *
 *******************************************************************************/

/*
```

```
 * Description :
 * 1. Initialize the ICU driver as required.
 * 2. Setup the ICU call back function.
 * 3. Setup the direction for the trigger pin as output pin through the GPIO driver.
 */
void Ultrasonic_init(void);

/*
 * Description :
 * Send the Trigger pulse to the ultrasonic.
 */
void Ultrasonic_Trigger(void);

/*
 * Description :
 * 1. Send the trigger pulse by using Ultrasonic_Trigger function.
 * 2. Start the measurements by the ICU from this moment.
 */
uint16 Ultrasonic_readDistance(void);

/*
 * Description :
 * 1. This is the call back function called by the ICU driver.
 * 2. This is used to calculate the high time (pulse time) generated by the ultrasonic sensor.
 */
void Ultrasonic_edgeProcessing(void);

#endif /* ULTRASONIC_SENSOR_H_ */
```

➢ ultrasonic_sensor.c

```
#include "../../MCAL/GPIO/gpio.h" /* To use setup direction function */
#include "../../MCAL/ICU/icu.h"  /* To use ICU driver function */

#include "ultrasonic_sensor.h"
#include <util/delay.h> /* To use _delay_us in Ultrasonic_Trigger function */

static uint8 g_edgeCount = 0;
static uint16 g_echoTime = 0;

/*
 * Description :
 * 1. Initialize the ICU driver as required.
 * 2. Setup the ICU call back function.
 * 3. Setup the direction for the trigger pin as output pin through the GPIO driver.
 */
void Ultrasonic_init(void)
{
    /* Setup the trigger pin direction as output */
    GPIO_setupPinDirection(TRIGGER_PORT_ID, TRIGGER_PIN_ID, PIN_OUTPUT);

    /* Configuration type of ICU with F_CPU/8 and raising edge  */
    ICU_ConfigType ICU_config =
        {F_CPU_8, RAISING};

    /* Call the initialization function */
    ICU_init(&ICU_config);

    /* Setup the call back function which be handled each interrupt */
    ICU_setCallBack(Ultrasonic_edgeProcessing);
}

/*
 * Description :
 * Send the Trigger pulse to the ultrasonic.
 */
void Ultrasonic_Trigger(void)
{
    /*  Transmit trigger pulse of at least 10 us to the HC-SR04 Trig Pin */
    GPIO_writePin(TRIGGER_PORT_ID, TRIGGER_PIN_ID, LOGIC_HIGH);
    /* 20 us to ensure that the trigger pulse has been sent successfully */
    _delay_us(TRIGGER_DELAY_VALUE);
    GPIO_writePin(TRIGGER_PORT_ID, TRIGGER_PIN_ID, LOGIC_LOW);
}

/*
 * Description :
 * 1. Send the trigger pulse by using Ultrasonic_Trigger function.
 * 2. Start the measurements by the ICU from this moment.
 */
uint16 Ultrasonic_readDistance(void)
{
    uint16 distance;
    /* Send the trigger pulse to HC-SR04 Trig Pin */
    Ultrasonic_Trigger();

    /*
     * Calculation details:
     * Sound velocity = 340.00 m/s = 34000 cm/s
     * The distance of Object (in cm) = (340000*echoTime)/2 = 17000 * echoTime
     * F_CPU/8 for timer frequency.
     * Then time to execute 1 instruction is 1 us.
     * Distance = 17000 x (echoTime) x 1 x 10^-6 cm = 0.017 x (echoTime) cm = (echoTime) / 58.8 cm
     */
    distance = g_echoTime / 58.8;
    return distance;
}

/*
 * Description :
 * 1. This is the call back function called by the ICU driver.
 * 2. This is used to calculate the high time (pulse time) generated by the ultrasonic sensor.
 */
void Ultrasonic_edgeProcessing(void)
{
    g_edgeCount++;
    if (g_edgeCount == 1)
    {
        /*
         * Clear the timer counter register to start measurements from the
         * first detected rising edge
         */
        ICU_clearTimerValue();
        /* Detect falling edge */
        ICU_setEdgeDetectionType(FALLING);
    }
    else if (g_edgeCount == 2)
    {
        /* Store the High time value */
        g_echoTime = ICU_getInputCaptureValue();
        /* Detect rising edge */
        ICU_setEdgeDetectionType(RAISING);
        /* For the next distance measurements operation */
        g_edgeCount = 0;
    }
}
```

10. 10 std_types.h

```
#ifndef STD_TYPES_H_
#define STD_TYPES_H_

/* Boolean Data Type */
```

```
typedef unsigned char boolean;

/* Boolean Values */
#ifndef FALSE
#define FALSE       (0u)
#endif
#ifndef TRUE
#define TRUE        (1u)
#endif

#define LOGIC_HIGH        (1u)
#define LOGIC_LOW         (0u)

#define NULL_PTR    ((void*)0)

typedef unsigned char uint8;           /*           0 .. 255              */

typedef signed char sint8;             /*        -128 .. +127             */

typedef unsigned short uint16;         /*           0 .. 65535            */

typedef signed short sint16;           /*      -32768 .. +32767           */

typedef unsigned long uint32;          /*           0 .. 4294967295       */

typedef signed long sint32;            /* -2147483648 .. +2147483647      */

typedef unsigned long long uint64;     /*       0 .. 18446744073709551615  */

typedef signed long long sint64;       /* -9223372036854775808 .. 9223372036854775807 */

typedef float float32;
typedef double float64;

#endif /* STD_TYPE_H_ */
```

10. 11 comman_macros.h

```
#ifndef COMMON_MACROS
#define COMMON_MACROS

/* Set a certain bit in any register */
#define SET_BIT(REG,BIT) (REG|=(1<<BIT))

/* Clear a certain bit in any register */
#define CLEAR_BIT(REG,BIT) (REG&=(~(1<<BIT)))
```

```
/* Toggle a certain bit in any register */
#define TOGGLE_BIT(REG,BIT) (REG^=(1<<BIT))

/* Rotate right the register value with specific number of rotates */
#define ROR(REG,num) ( REG= (REG>>num) | (REG<<(8-num)) )

/* Rotate left the register value with specific number of rotates */
#define ROL(REG,num) ( REG= (REG<<num) | (REG>>(8-num)) )

/* Check if a specific bit is set in any register and return true if yes */
#define BIT_IS_SET(REG,BIT) ( REG & (1<<BIT) )

/* Check if a specific bit is cleared in any register and return true if yes */
#define BIT_IS_CLEAR(REG,BIT) ( !(REG & (1<<BIT)) )

/* Get a specified bit in register */
#define GET_BIT(REG, BIT) ((REG & (1 << BIT)) >> BIT)

#endif
```