Modern day vehicles come with various safety and entertainment features to make our lives better. Their safety features mean that driving has become safer than ever. Driving technology has changed dramatically over the past decade. Engines are more efficient, in-car entertainment has become as good as your living room. However, safety features are considered more important than entertainment features, consequently they have higher priorities when it comes to execution. In this project, you implemented some of these features using an Arduino Mega board, giving each feature a suitable priority. We used Arduino C alongside FreeRTOS to implement this project. We assembled this project by selecting three features which are Automatic Emergency Braking, G eneral Purpose Display and Automatic Headlights, and Sound System. The car moves using the motors and Pulse Width Modulation and stops when it detects an obstacle producing audible warning. Date, Time, Temperature and Current gear are displayed on the LCD Display. Also, the car's LED is turned on when it is dark. On the car, we can play, pause, play next or play previous MP3 audio files via the speaker using buttons.

The components used are:

- Car with 4 motors
- H bridge (the motor driver) is the circuit which allows the movement of the wheels forward and backward.
- Buzzer is used to produce audible warning when the car detects an obstacle.
- Ultra-sonic sensor is used to determine the distance between the car and an obstacle.
- Joystick is used to move the car forward and backward.
- 9 volts battery is used to supply the power needed for the car.
- 16x2 LCD is used to display the date, time, temperature and current gear.
- Potentiometer is used for the contrast of the LCD.
- Temperature sensor is used to measure the temperature to display it on the LCD.
- Light sensor is used to detect when it is dark.
- LEDs are used as the car's headlights that light up when it is dark.
- RTC (real time external clock) is used to know the time.
- Mini MP3 module is used to connect the Arduino and the speaker together.
- SD card is used to save three playable songs and placed in the DFPlayer mini MP3 module.
- Speaker is used to output the sound.
- Three push buttons used as play/pause button, play next button and play previous button.
- Jumper wires (male to male)
- Jumper wires (male to female)

The project full circuit:

The names of the libraries used are:

- "Arduino.h" includes all the Arduino specific stuff and the AVR stuff.
- "SoftwareSerial.h" allows serial communication on other digital pins of the Arduino.
- "DFRobotDFPlayerMini.h" used to be able to play the songs on the speakers.
- "RTClib.h" allows keep tracking of date and time
- <LiquidCrystal.h> allows an Arduino board to control LiquidCrystal displays (LCDs), works with in either 4- or 8-bit mode.
- <Arduino_FreeRTOS.h> includes the generic headers required for the FreeRTOS port being used.
- <event_groups.h> collection of bits to which an application can assign a meaning.
- <FreeRTOSConfig.h> contains a multitude of API and environment configurations.
- <semphr.h> allows the use of semaphores that are used for pure synchronization between tasks or between an interrupt and a task
- <queue.h> allows the use of queues in the code.
- <list.h> allows the use of lists in the code.
- <timers.h> can make callbacks or toggle pin states.
- <portable.h> portable layer API. Each function must be defined for each port.
- <projdefs.h> defines the prototype to which task functions must conform.
- <task.h> is used to be able to create and handle a task for each feature.
- <croutine.h> allows the use of co-routines, creating and handling them.
- <portmacro.h> Each FreeRTOS port has a unique portmacro.h header file.
- <FreeRTOSVariant.h> Contains the AVR specific configurations for this port of freeRTOS.
- <dht11.h> allows reading the humidity and temperature from the sensors and store it.
- <Wire.h> allows you to communicate with I2C / TWI devices.

How do you take and handle the inputs?

We handle an input by declaring it as an int and assiging a variable, this integer represents the number of the digital pin connected to it. Then we use pinMode(x, INPUT) in the setup function to set it as an input. However, Arduino pins default to inputs, so they don't need to be explicitly declared as inputs with pinMode() when you're using them as inputs. Pins configured this way are said to be in a high-impedance state. There are also some inputs that are declared as INPUT_PULLUP. The INPUT_PULLUP accesses the 20K built-in pullup resistors by setting the pinMode() as INPUT_PULLUP.

This effectively inverts the behavior of the INPUT mode, where HIGH means the sensor is off, and LOW means the sensor is on.

An example:

```
int inPin = 7;    // pushbutton connected to digital pin 7

void setup() {

  pinMode(inPin, INPUT);   // sets the digital pin 7 as input

}

void loop() {

  digitalRead(inPin);   // read the input pin

}
```

How do you configure and handle the outputs?

Pins configured as OUTPUT with pinMode() are said to be in a low-impedance state. Also, we handle an output by declaring it as an int and assigning a variable, this integer represents the number of the digital pin connected to it. Then we use pinMode(x, OUTPUT) in the setup function to set it as an output.

An example:

```
int ledPin = 13;                // LED connected to digital pin 13

void setup() {

  pinMode(ledPin, OUTPUT);   // sets the digital pin 7 as input

}

void loop()

{

  digitalWrite(ledPin, HIGH);  // sets the LED on

  delay(1000);                // waits for a second

  digitalWrite(ledPin, LOW);   // sets the LED off

  delay(1000);                // waits for a second

}
```

The features and the tasks:

```
xTaskCreate(TaskMove, "Move", 1000, NULL, 2, NULL);

xTaskCreate(TaskModule2, "Module2", 1000, NULL, 1, NULL);

xTaskCreate(TaskModule3, "Module3", 1000, NULL, 1, NULL);
```

The three features were divided into three tasks. The task concerned with the movement "TaskMove" has the highest priority while the other two have the same priority. And we used vTaskDelayUntil to delay the tasks until a specified time to ensure a constant execution frequency. The first task "TaskMove" contains the movement and braking of the car, reading the movement of the joystick to detect the direction the car should move in and setting the buzzer to HIGH if the car faces an obstacle. The second task "Module2" gets date and time using rtc, reads the temperature, gets the gear and starts printing these values on the LCD. It also reads the value of the light sensor and if it exceeds a certain value "600" which means it is dark, it turns the LEDs on. Finally, the last task "Module3" is concerned with checking which push buttons is pressed and starts taking action according to the push button pressed, it can either play, pause, play next or play previous. It also sets the volume of the mp3 player. In this module, we used binary semaphores to make sure the next and previous buttons don't work if no sound is being played.

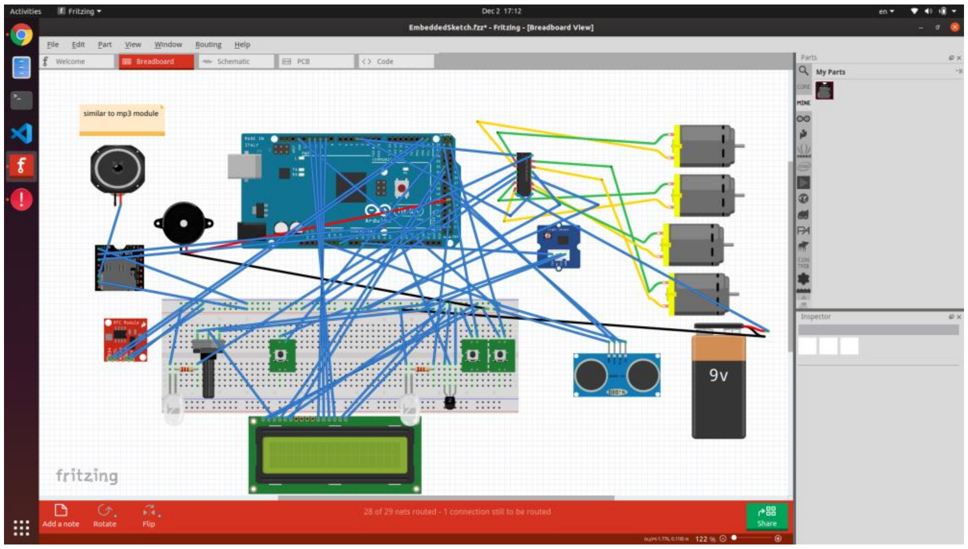
The problems or limitations faced during the implementation of your project:

We had a problem while printing on the LCD as we were connecting some of the jumper wires to analog pins while it was supposed to be connected to the digital pins. We didn't notice this easily; we didn't why wasn't it working although the code is correctly implemented. Moreover, we needed to weld the jumper wires to the speakers as it has no place for the pins, which delayed the last module implementation. Also, when we uploaded the three songs on the SD card and placed it in the mp3 module, nothing was working. Then we noticed that the songs need to be converted to mono using audacity to remove the noise and it worked. Besides, we had a problem with the push buttons, it wasn't functioning properly until we changed it to INPUT_PULLUP.

The work among the team members:

We didn't divide the project into parts among us. We worked on each module all together; we were all trying together using the same laptop. However, Mohamed Hossam was the one responsible for searching for the components and getting them.

The schematic diagram:



similar to mp3 module

28 of 29 nets routed - 1 connection still to be routed