```
#include <DS3231.h>
#include <DFRobot_sim808.h>
#include <SoftwareSerial.h>
#include <SD.h>
#include "ELMduino.h"
#include <MPU6050.h>
```

- These lines include the necessary libraries for the DS3231 real-time clock, SIM808 module, SoftwareSerial communication, SD card module, ELMduino library for ELM327 module, and MPU6050 module.

```
DS3231 rtc(2, 3);
```

- This line creates an instance of the DS3231 class called `rtc` and initializes it with the I2C interface pins 2 and 3.

```
SoftwareSerial mySerial(10, 11); //sim808 serial
```

- This line creates a SoftwareSerial object called `mySerial` to establish communication with the SIM808 module using pins 10 (RX) and 11 (TX).

```
DFRobot_SIM808 sim808(&mySerial);  //Connect RX,TX,PWR,
```

- This line creates an instance of the DFRobot_SIM808 class called `sim808` and initializes it with the `mySerial` object to communicate with the SIM808 module.

```
SoftwareSerial obdSerial(0, 1); // RX, TX obd
```

- This line creates a SoftwareSerial object called `obdSerial` to establish communication with the OBD-II module using pins 0 (RX) and 1 (TX).

```
const int chipSelect = 53;
```

- This line defines the chip select pin for the SD card module as pin 53.

```
const int sensorPin = A0;  // analog input pin for the sensor
```

- This line defines the analog input pin A0 as `sensorPin` for reading the sensor value.

```
#define PHONE_NUMBER1 "94769439154"
#define PHONE_NUMBER2 "94713651522"
```

- These lines define the phone numbers `PHONE_NUMBER1` and `PHONE_NUMBER2` as strings.

```
#define MESSAGE_LENGTH 5
```

- This line defines the length of the message as `MESSAGE_LENGTH` with a value of 5.

```
#define ELM_PORT obdSerial
```

- This line defines the ELM327 module's communication port as `ELM_PORT`, which is the `obdSerial` object.

```
int sensorValue;
```

- This line declares an integer variable `sensorValue` to store the value read from the sensor.

```
char latStr[15];
char lonStr[15];
char message[5];
```

- These lines declare character arrays `latStr`, `lonStr`, and `message` to store latitude, longitude, and the SOS message.

```
int messageIndex = 0;
```

- This line initializes an integer variable `messageIndex` to 0, which will be used to keep track of the index while constructing the SOS message.

```
char phone[16];
```

- This line declares a character array `phone` to store the phone number for sending messages.

```
char datetime[24];
```

- This line declares a character array `datetime` to store the date and time.

```
String date;
String time;
```

- These lines declare String objects `date` and `time` to store the date and time retrieved from the real-time clock.

```
char MESSAGE[300];
```

- This line declares a character array `MESSAGE` to store the final message to be sent.

```
float

 lat, lon, speed;
```

- These lines declare float variables `lat`, `lon`, and `speed` to store latitude, longitude, and speed values.

```
ELM327 myELM327;
```

- This line creates an instance of the ELM327 class called `myELM327` to communicate with the ELM327 module.

```
MPU6050 mpu;
```

- This line creates an instance of the MPU6050 class called `mpu` to communicate with the MPU6050 module.

```
int rpm = 0;
float fuel = 0;
```

- These lines declare integer variable `rpm` and float variable `fuel` to store RPM and fuel values.

```
int ax, ay, az;
int gx, gy, gz;
```

- These lines declare integer variables `ax`, `ay`, `az`, `gx`, `gy`, and `gz` to store acceleration and gyroscope values.

Here's an explanation of each line in the `setup()` function:

```
rtc.begin();
```

- This line initializes the DS3231 real-time clock by calling the `begin()` function.

```
mySerial.begin(9600);
```

- This line initializes the communication with the SIM808 module through the `mySerial` object using a baud rate of 9600.

```
Serial.begin(9600);
```

- This line initializes the serial communication with the computer through the USB cable using a baud rate of 9600. It allows printing debug messages to the serial monitor.

```
if (!SD.begin(chipSelect)) {
  Serial.println("SD card initialization failed");
  return;
}
```

- This block of code initializes the SD card module by calling the `begin()` function with the chip select pin defined as `chipSelect`. If the initialization fails, it prints an error message and returns from the function.

```
Serial.println("SD card initialized");
```

- This line prints a message to the serial monitor indicating that the SD card has been successfully initialized.

```
while (!sim808.init()) {
  delay(2000);
  Serial.print("Sim808 init error\r\n");
}
```

- This block of code initializes the SIM808 module by calling the `init()` function in a loop until it successfully initializes. If initialization fails, it prints an error message to the serial monitor and waits for 2 seconds before trying again.

```
Serial.println("Sim808 init success");
```

- This line prints a message to the serial monitor indicating that the SIM808 module has been successfully initialized.

```
rtc.setDate(16, 6, 2023);
rtc.setTime(12, 19, 30);
```

- These lines set the date and time of the DS3231 real-time clock. In this case, it sets the date to June 16, 2023, and the time to 12:19:30.

```
if (sim808.attachGPS()){
  Serial.println("Open the GPS power success");
}
else
  Serial.println("Open the GPS power failure");
```

- This block of code checks if the GPS power is successfully turned on by calling the `attachGPS()` function of the SIM808 module. If successful, it prints a message indicating success. Otherwise, it prints a message indicating failure.

```
if (!myELM327.begin(ELM_PORT, true, 2000)) {
  Serial.println("Couldn't connect to OBD scanner");
}
```

- This block of code initializes the ELM327 module by calling the `begin()` function of the `myELM327` object, specifying the communication port, enabling automatic protocol detection (`true`), and setting a timeout of 2000 milliseconds. If the connection to the OBD scanner fails, it prints an error message.

```
Serial.println("Connected to ELM327");
```

- This line prints a message to the serial monitor indicating that the connection to the ELM327 module has been successfully established.

```
mpu.initialize();
```

- This line initializes the MPU6050 module by calling the `initialize()` function of the `mpu` object.

```
Serial.println("MPU-6050 Test");
```

- This line prints a message to the serial monitor indicating that the MPU-6050 module is being tested.

```
Serial.println("Initializing MPU6050...");
mpu.initialize();
```

- These lines initialize and test the connection with the MPU6050 module. It prints messages to the serial monitor to indicate the initialization and connection status.

```
delay(1000);
```

- This

line introduces a delay of 1000 milliseconds (1 second) to allow for stabilization and settle the system before moving on to the next part of the code.

Here's an explanation of each line in the `loop()` function:

```
if (sim808.getGPS()) {
```

- This line checks if GPS data is available by calling the `getGPS()` function of the SIM808 module. If GPS data is available, the code inside the if statement will be executed.

```
getGPSData();
```

- This line calls a function named `getGPSData()` to retrieve GPS data from the SIM808 module.

```
getECUData();
```

- This line calls a function named `getECUData()` to retrieve data from the ECU (engine control unit).

```
delay(2000);
```

- This line introduces a delay of 2000 milliseconds (2 seconds) before proceeding to the next line of code.

```
sensorValue = analogRead(sensorPin);
```

- This line reads the analog value from the `sensorPin` analog input pin and assigns it to the `sensorValue` variable. It is used to read a sensor connected to the pin.

```
gyro();
```

- This line calls a function named `gyro()` to handle gyroscope-related operations.

```
date = rtc.getDateStr();
time = rtc.getTimeStr();
```

- These lines retrieve the current date and time from the DS3231 real-time clock and assign them to the `date` and `time` variables respectively. The `getDateStr()` and `getTimeStr()` functions are used to convert the date and time values into string representations.

```
Serial.print("date: ");
Serial.println(date);
Serial.print("time: ");
Serial.println(time);
```

- These lines print the current date and time to the serial monitor for debugging purposes.

```
messageIndex = sim808.isSMSunread();
Serial.print("messageIndex: ");
Serial.println(messageIndex);
```

- These lines check if there are any unread SMS messages by calling the `isSMSunread()` function of the SIM808 module. The value is stored in the `messageIndex` variable, and its value is printed to the serial monitor.

```
dtostrf(lat, 10, 7, latStr);
dtostrf(lon, 10, 7, lonStr);
```

- These lines convert the latitude and longitude values (stored in the variables `lat` and `lon`) from floating-point format to strings with 7 decimal places using the `dtostrf()` function. The converted values are stored in the `latStr` and `lonStr` variables respectively.

```
if (sensorValue > threshold) {
  sosTrigger();
}
```

- This line checks if the `sensorValue` is greater than the defined `threshold`. If it is, the `sosTrigger()` function is called to handle the SOS trigger.

```
upload();
```

- This line calls a function named `upload()` to handle the uploading of data to a server or storage.

Here's an explanation of each line in the `getECUData()` function:

```
float tempRPM = myELM327.rpm();
float tempFuel = myELM327.fuelLevel();
```

- These lines call the `rpm()` and `fuelLevel()` functions of the `myELM327` object to retrieve the RPM (revolutions per minute) and fuel level data

from the ECU (engine control unit) respectively. The values are stored in temporary variables `tempRPM` and `tempFuel`.

```
if (myELM327.nb_rx_state == ELM_SUCCESS)
{
  rpm = (int)tempRPM;
  fuel = (float)tempFuel;
  Serial.print("RPM: "); Serial.println(rpm);
  Serial.print("Fuel Level: "); Serial.println(fuel);
}
```

- This condition checks if the `nb_rx_state` member variable of the `myELM327` object is equal to `ELM_SUCCESS`, indicating that the ELM327 module successfully received the requested data from the ECU. If the condition is true, the following code is executed:
  - The `tempRPM` value is cast to an integer and assigned to the `rpm` variable.
  - The `tempFuel` value is assigned to the `fuel` variable.
  - The RPM and fuel level values are printed to the serial monitor for debugging purposes.

```
else if (myELM327.nb_rx_state != ELM_GETTING_MSG)
  myELM327.printError();
```

- If the previous condition is not true, this `else if` block is executed. It checks if the `nb_rx_state` member variable is not equal to `ELM_GETTING_MSG`, indicating an error occurred while retrieving data from the ECU. If the condition is true, the `printError()` function of the `myELM327` object is called to print the error message to the serial monitor.

Here's an explanation of each line in the `sosTriger()` function:

```
String MESSAGE = "Impact Detected! https://www.google.com/maps/search/?api=1&query=" + Strin
```

- This line creates a `String` variable named `MESSAGE` and assigns it a message string. The message contains the text "Impact Detected!" followed by a Google Maps URL that includes the latitude (`latStr`) and longitude (`lonStr`) values.

```
MESSAGE.replace("query= ", "query=");
```

- This line replaces any occurrence of the substring "query=" (with a trailing space) in the `MESSAGE` string with "query=". It essentially removes any unwanted space after the "query=" part of the URL.

```
sim808.sendSMS(PHONE_NUMBER1, MESSAGE.c_str());
sim808.sendSMS(PHONE_NUMBER2, MESSAGE.c_str());
```

- These lines send an SMS message using the `sim808` object to two phone numbers: `PHONE_NUMBER1` and `PHONE_NUMBER2`. The content of the SMS message is the `MESSAGE` string. The `c_str()` function is used to convert the `MESSAGE` string to a C-style string, which is the expected format for the `sendSMS()` function.

```
Serial.print(MESSAGE);
```

- This line prints the `MESSAGE` string to the serial monitor for debugging purposes.

The `sosTriger()` function is responsible for constructing an SOS message with the impact location using latitude and longitude values, sending the SOS message to two phone numbers, and printing the message to the serial monitor.

Here's an explanation of each line in the `fileWrite()` function:

```
File dataFile = SD.open("data.csv", FILE_WRITE);
```

- This line declares a `File` object named `dataFile` and opens the file named "data.csv" in write mode using the `SD.open()` function from the SD library. The `FILE_WRITE` parameter indicates that the file should be opened for writing.

```
if (dataFile) {
```

- This line checks if the file was successfully opened. It evaluates whether the `dataFile` object is valid, which means the file opening was successful.

```
dataFile.print(date);
dataFile.print(",");
dataFile.print(time);
dataFile.print(",");
dataFile.print(latStr);
dataFile.print(",");
dataFile.print(lonStr);
dataFile.print(",");
dataFile.print(speed);
dataFile.print(",");
dataFile.print(rpm);
dataFile.print(",");
dataFile.print(fuel);
dataFile.print(",");
dataFile.print(ax);
dataFile.print(",");
dataFile.print(ay);
dataFile.print(",");
```

```
dataFile.print(az);
dataFile.print(",");
dataFile.print(gx);
dataFile.print(",");
dataFile.print(gy);
dataFile.print(",");
dataFile.print(gz);
dataFile.print(",");
dataFile.print(sensorValue);
dataFile.print(",");
```

- These lines write the values to the `dataFile` using the `print()` function. Each value is separated by a comma to conform to the CSV file format. The values being written include `date`, `time`, `latStr`, `lonStr`, `speed`, `rpm`, `fuel`, `ax`, `ay`, `az`, `gx`, `gy`, `gz`, and `sensorValue`.

```
dataFile.println();  // end the line
```

- This line writes a newline character to the `dataFile`, which effectively ends the current line of data.

```
dataFile.close();
```

- This line closes the `dataFile` to ensure that the changes are saved and the file resources are released.

```
Serial.println("Data written to file");
```

- This line prints a message to the serial monitor indicating that the data has been successfully written to the file.

```
Serial.println("Error opening file");
```

- This line prints a message to the serial monitor if there was an error opening the file.

The `fileWrite()` function is responsible for opening a file named "data.csv" in write mode, writing various sensor data values to the file in CSV format, and closing the file. It also provides feedback through the serial monitor.

Here's an explanation of each line in the `upload()` function:

```
fileWrite();
```

- This line calls the `fileWrite()` function to write data to the SD card.

```
if (mySerial.available())
  Serial.write(mySerial.read());
```

- This code checks if there is any data available on the `mySerial` serial connection and if so, it reads and writes that data to the Serial monitor.

```
mySerial.println("AT");
delay(1000);
```

- These lines send the "AT" command to the SIM808 module and wait for 1 second before executing the next command.

```
// Several other AT commands are sent to the SIM808 module with delay intervals between ther

mySerial.println("AT+CPIN?");
delay(1000);

mySerial.println("AT+CREG?");
delay(1000);

mySerial.println("AT+CGATT?");
delay(1000);

mySerial.println("AT+CIPSHUT");
delay(1000);

mySerial.println("AT+CIPSTATUS");
delay(2000);

mySerial.println("AT+CIPMUX=0");
delay(2000);
```

- These lines send various AT commands to the SIM808 module to perform actions such as checking the SIM card status, network registration status, attachment status, shutting down the IP connection, getting the IP connection status, and configuring the IP connection multiplexer. Delay intervals are added after each command to allow time for the SIM808 module to respond.

```
ShowSerialData();
```

- This line calls the `ShowSerialData()` function to display any data received from the SIM808 module on the Serial monitor.

```
mySerial.println("AT+CSTT=\"dialogbb\"");  //start task and setting the APN,
delay(1000);
ShowSerialData();
```

- These lines set the Access Point Name (APN) for the GPRS connection. The specific APN used here is "dialogbb" (which may vary depending on the network provider). Delay intervals are added, and the received data is displayed on the Serial monitor.

```
mySerial.println("AT+CIICR");  //bring up wireless connection
delay(3000);
ShowSerialData();
```

- These lines bring up the wireless GPRS connection. Delay intervals are added, and the received data is displayed on the Serial monitor.

```
mySerial.println("AT+CIFSR");  //get local IP address
delay(2000);
ShowSerialData();
```

- These lines retrieve the local IP address of the SIM808 module. Delay intervals are added, and the received data is displayed on the Serial monitor.

```
mySerial.println("AT+CIPSPRT=0");
delay(3000);
ShowSerialData();
```

- These lines configure the IP connection to not print the received data to the Serial monitor. Delay intervals are added, and the received data is displayed on the Serial monitor.

```
mySerial.println("AT+CIPSTART=\"TCP\",\"api.thingspeak.com\",\"80\"");  //start up the conne
delay(2000);
ShowSerialData();
```

- These lines initiate a TCP connection with the "api.thingspeak.com" server on port 80. Delay intervals are added, and the received data is displayed on the Serial monitor.

```
mySerial.println("AT+CIPSEND");  //begin send data to remote server
delay(4000);
ShowSerialData();
```

- These lines signal the SIM808 module to start sending data to the remote server. Delay intervals are added, and the received data is displayed on the Serial monitor.

```
String str

 = "GET https://api.thingspeak.com/update?api_key=DTUJH2YW8QKCOG4F&field1=" + String(latStr)
str.replace("field1= ", "field1=");
Serial.println(str);
mySerial.println(str);
```

- These lines construct a URL string that includes the data to be sent to the remote server (in this case, to the ThingSpeak API). The values of `latStr`, `lonStr`, `time`, `date`, `speed`, `sensorValue`, `rpm`, and `fuel` are concatenated to the URL. The `str.replace()` function removes any spaces between the latitude/longitude values and the "&field3" parameter. The constructed

URL string is printed to the Serial monitor, and then it is sent to the remote server via the SIM808 module.

```
delay(4000);
ShowSerialData();
```

- These lines introduce a delay and display any received data on the Serial monitor.

```
mySerial.println((char)26);  //sending
delay(2000);
mySerial.println();
ShowSerialData();
```

- These lines send the ASCII character 26 (representing the Ctrl+Z command) to indicate the end of the data being sent to the remote server. Delay intervals are added, and the received data is displayed on the Serial monitor.

```
mySerial.println("AT+CIPSHUT");  //close the connection
delay(100);
ShowSerialData();
delay(2000);
```

- These lines close the TCP/IP connection. Delay intervals are added, and the received data is displayed on the Serial monitor.

```
if (messageIndex > 0) {
  // Code for reading and processing SMS messages
}
```

- This code block checks if there is an unread SMS message (`messageIndex` variable stores the index of the message). If there is a message, it reads the SMS, deletes it, and performs further processing.

The `upload()` function handles various operations related to establishing a GPRS connection, configuring the SIM808 module, sending data to a remote server, and handling SMS messages. It interacts with the SIM808 module using AT commands and displays relevant information on the Serial monitor.

Here's an explanation of each line in the `getGPSData()` function:

```
lat = (sim808.GPSdata.lat);
lon = (sim808.GPSdata.lon);
speed = sim808.GPSdata.speed_kph;
```

- These lines extract the latitude, longitude, and speed data from the `GPSdata` structure of the `sim808` object. The values are assigned to the

respective variables `lat`, `lon`, and `speed`.

```
Serial.print("latitude :");
Serial.println(lat, 5);
Serial.print("longitude :");
Serial.println(lon, 5);
Serial.print("speed_kph :");
Serial.println(speed, 5);
```

- These lines display the latitude, longitude, and speed data on the Serial monitor. The values are printed with 5 decimal places for accuracy.

Overall, the `getGPSData()` function retrieves the GPS data (latitude, longitude, and speed) from the `sim808` object and displays it on the Serial monitor.

Here's an explanation of each line in the `msg()` function:

```
String MESSAGE = "https://www.google.com/maps/search/?api=1&query=" + String(latStr) + "," +
```

- This line creates a string variable `MESSAGE` that represents a Google Maps URL. It concatenates the base URL "https://www.google.com/maps/search/?api=1&query=" with the latitude (`latStr`) and longitude (`lonStr`) converted to strings.

```
MESSAGE.replace("query= ", "query=");
```

- This line replaces any occurrence of "query=" (with a space after the equal sign) in the `MESSAGE` string with "query=" (without the space). It ensures that the URL is correctly formatted.

```
sim808.sendSMS(PHONE_NUMBER1, MESSAGE.c_str());
sim808.sendSMS(PHONE_NUMBER2, MESSAGE.c_str());
```

- These lines send an SMS message containing the `MESSAGE` string to two phone numbers specified by `PHONE_NUMBER1` and `PHONE_NUMBER2`. The `MESSAGE.c_str()` function is used to convert the `MESSAGE` string to a C-style string required by the `sendSMS()` function.

```
Serial.print(MESSAGE);
```

- This line prints the `MESSAGE` string on the Serial monitor. It displays the Google Maps URL containing the latitude and longitude.

Overall, the `msg()` function creates a Google Maps URL based on the latitude and longitude, sends SMS messages with the URL to specified phone numbers, and prints the URL on the Serial monitor.

Here's an explanation of each line in the `ShowSerialData()` function:

```
while (mySerial.available() != 0)
```

- This line creates a loop that continues as long as there is data available to read from the `mySerial` object (which represents a SoftwareSerial connection).

```
Serial.write(mySerial.read());
```

- This line reads a single byte of data from the `mySerial` object and writes it to the Serial monitor. It essentially forwards the data received on `mySerial` to the Serial monitor.

```
delay(1000);
```

- This line introduces a delay of 1000 milliseconds (1 second) between consecutive iterations of the loop. It allows time for the remaining data in the `mySerial` buffer to be read and displayed before the next iteration.

The purpose of the `ShowSerialData()` function is to read data from the `mySerial` object and display it on the Serial monitor. It continuously reads and displays data as long as there is data available to read. The delay between iterations ensures that all the available data is processed before moving on.