# SAFEDRIVER

## Sistema de Evaluación Semiautomática del Estado del Conductor

Hugo Collantes
Gabriel Cuesta

# INDEX

## 1. SUMMARY:

A semi-automated "SAFEDRIVER" system has been created using high-level C programming to ensure a reduction in accidents through the implementation of this system. Drivers will be subjected to tests that cause the highest accident rates, such as alcohol, smoke, and driving inattentiveness. The project consists of these tests, and various results will be obtained adjusted to safety parameters within the legal range.

It should be noted that this does not involve strict driver control; it is merely informative, with the aim of raising awareness among all drivers about preventing a potential accident.

## 2. OBJECTIVES:

The main objective is to correctly perform the three aforementioned tests and display the results. To achieve this, both the LCD screen and the mobile phone will be monitored, showing the sensor response for each test.

The sensor calibration will be as accurate as possible, ensuring reliability for the user.

The user will be able to repeat the entire test cycle by using a reset button after each test.

Finally, the user will be shown whether all tests were passed or, on the contrary, whether any test was failed.

## 3. MODULES:

In this project, we use several modules for the different actions of our program. Since there are several distinct modules, we've chosen to discuss each one separately, explaining how it works and how to configure it. We don't think it's necessary to explain the code line by line, but we will highlight some of the most important or unusual functions and configurations..

### 3.1 TIMER0:

The purpose of TIMER0 is to signal the various analog measurements; each time an analog measurement is taken, an analog measurement will be taken. This way, we can obtain multiple measurements in a given time.

```
//Timer 0:
OPTION_REG= 0x07; //Timer 0 como temporizador, flanco subida pre-divisor 256 al Timer0
TMR0=0X79;
```

For its configuration we modify the OPTION_REG and the TMR0, as indicated in the image.

## 3.2 TIMER2:

The purpose of TIMER2 is to use it to implement two PWM-controlled alarms, using the CCP1 (pulse width modulation) module. This PWM signal controls a piezoelectric buzzer. In this project, it works in conjunction with the CCP1 module to control the duty cycle and tone frequency of our alarms.

```
//Timer 2:
T2CON=0x05;      //Solo TMR2 on y factor de división 4
PR2=249;
TMR2=0;
```

TIMER2 defines the alarm frequency by adjusting the value stored in the PR2 register which as seen in the code starts at 249.

## 3.3 CCP1:

Our CCP1 module works as a PWM module, its purpose is to implement two alarms. It works in conjunction with TIMER2.

```
//Modulo CCP:
CCPR1L=0;        //REGISTRO CON TODO CEROS
CCP1CON=0X0C;    //MODO PWM
```

Initial setup is simple, however, this module is completed with the alarm functions.

```
//FUNCIONES DE PMW, PARA LAS ALARMAS
//Configuracion inicial del PMW (hecho asi por claridad)
void init_PWM(unsigned int frecuencia) {
    unsigned int pr2_val = (_XTAL_FREQ / (frecuencia * 16)) - 1;
    PR2 = (unsigned char)pr2_val;
    CCP1CON = 0x0C;    // Modo PWM
    T2CON = 0x05;      //Prescaler 1:4
    TRISC2=0;
    CCPR1L = (pr2_val + 1) / 2;
}
//Seleccion del ciclo de trabajo
void set_PWM_duty(unsigned int duty) {
    unsigned int dc = duty * 4; // Convertimos duty (0-1023) a 10 bits
    CCPR1L = duty >> 2;
    CCP1CONbits.CCP1X = duty & 0x01;
    CCP1CONbits.CCP1Y =(duty >> 1) & 0x01;
}
```

Our init_PWM and set_PWM_duty functions do exactly what they say, they allow us to modify both the PWM frequency, by modifying PR2, and the duty cycle, in the 2 bits of CCPR1CON (bits 4 and 5) and 8 of CCPR1. This is useful since we are using 2 different alarms.

```
//Alarma 1 (media)
void alarma1() {
    init_PWM(2000);
    set_PWM_duty(250);
    unsigned int i = 0;

    while (i < 10) {
        __delay_ms(500);
        set_PWM_duty(0);        // Silencio
        __delay_ms(500);
        set_PWM_duty(250);      // Reactivar
        i++;
    }
    __delay_ms(250);
    set_PWM_duty(0);
}
```

Using alarm 1 as an example, we can see how the two previous functions are used to facilitate the process. Alarm 2 will follow the same process, but with a different frequency and duty cycle.

## 3.4 ADCON:

The analog-to-digital module will be used to obtain information from the various sensors and transform it into digital data for processing by the microcontroller.

The A/D converter is divided into two control registers: ADCON0, which is used to change channels and initiate conversions; and ADCON1, which selects the result format and configures the A/D module inputs.

```
//Modulo analogico digital:
ADCON0=0X89;  //canal 1
ADCON1=0X00;
```

In the ADCON0 register, we start at channel 1. All inputs are defined as analog in ADCON1, and the result is stored on the left. The clock selection is determined by ADCS0, ADCS1, and ADCS2, programmed as 0, 1, and 0 respectively, so our A/D module frequency will be Fosc/8.

## 3.5 USART:

The USART module allows us to view the results of our tests on a Bluetooth-connected device. In our case, it will only send data, so its use is somewhat simpler.

```
//Función de configuracion de USART (hecho asi por claridad)
void uart_init() {
    TXEN = 1;
    BRGH = 1;
    SPBRG = 25;
    SYNC = 0;
    SPEN = 1;
}
```

Its configuration is provided by a function, this is done this way for code clarity. It involves activating the bits required for its operation and selecting 9600 baud, which is provided by the SPBRG.

This is where a problem arises: LCD compatibility. Since we want to use the printf() function, this creates problems since both use the basic configuration function putch(). To do this, we must use a variable that allows the printf() function to understand which of the two modules, LCD or USART, it should send the message to.

```
void putch(unsigned char c){
    if (cambio==0){
        lcd_putch(c);
    }
    else{
        while (!TXIF)
        continue;
        TXREG = c;
    }
}
```

With this solution we must remember that every time we want to send a message via Bluetooth the variable "change" must be equal to 1, and to send it via LCD it must be 0. With this we can now write the 3 functions that will send data to the mobile.

```
//Funcion de envio de resultados por Bluetooth
//IMPORTANTE, el cambio de la variable "cambio" para habilitar el USART
//y desactivarlos para continuar con el LCD activo
void enviar_resultado_alcohol(int mgml) {
    cambio=1;
    printf("Alc: %d mg/L\r\n", mgml);
    cambio=0;
}
void enviar_resultado_humo(int gas) {
    cambio=1;
    printf("Humo: %d ppm\r\n", gas);
    cambio=0;
}
void enviar_resultado_atencion(int ok) {
    cambio=1;
    if (ok) {
        printf("Superado!");
    } else {
        printf("No esta atento");
    }
    cambio=0;
}
```

These are very simple functions that allow us to improve the clarity of the main code by making the LCD and USART changes separately. Simply enter the value you want to send in each of the functions.

## 3.6 LCD:

The LCD screen will display the measurement results, as well as various user reference messages necessary to complete all tests. Compatibility with the USART module has already been discussed previously..

## 3.7 INTERRUPTIONS:

In order to use interrupts we must enable the different interrupts.

```
//Interrupciones:
    GIE=1;      //Generales
    PEIE=1;

    T0IE=0;     //Timer 0
    T0IF=0;

    TMR1IE=1;   //Timer 1
    TMR1IF=0;

    INTE=1;     //Interrupciones del puerto B y RB0
    INTF=0;
    RBIE=1;
    RBIF=0;

    CCP1IE=1;   //Interrupciones de módulo CCP
    CCP1IF=0;


    ADIE=1;     //Interrupciones módulo analógico digital.
    ADIF=0;
```

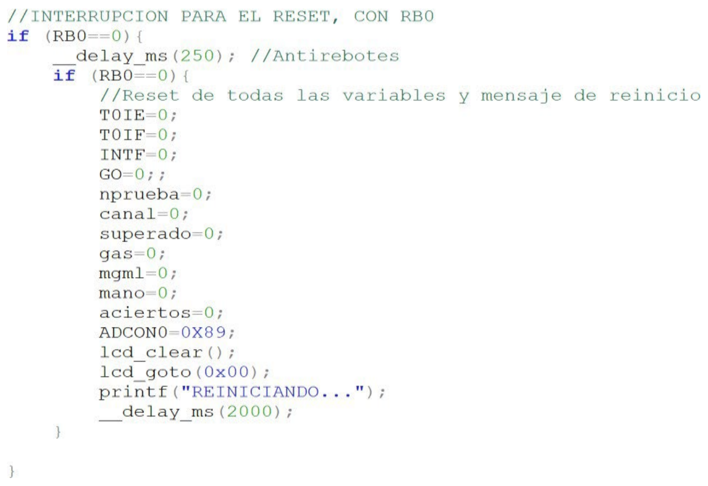
Two interrupts were primarily used, Timer0 and RB0, for different functions. The first was used to perform measurements with the A/D module. We distinguish between two different types of measurements: analog and digital, both of which use the Timer 0 overflow.

```
//Mediciones digitales, pulsadores KY
if (T0IF==1 && canal==2){
    if (RB1==1 && RB2==1){
        superado=1;
    }else{
        superado=0;
    }
    lcd_clear();
    lcd_goto(0x00);
    printf("Continue...");
    T0IF=0;
}
```

In this short code you can see that when TIMER0 overflows and the channel (variable) is two, our system will perform a digital measurement where depending on whether RB1 and RB2 are pressed (“hands on the wheel”) one response or another will be obtained.

```
//Mediciones analogicas, Alcohol y Humo
if (T0IF==1 && canal!=2){
    GO=1;
    //Obtener el dato de nuestra mediciónes analogicas
    if (ADIF==1){
        GO=0;
        ADIF=0;
        adc[canal]=(ADRESH << 2)|(ADRESL >> 6);
    }
    //Calculo del porcentaje de alcohol en el aliento, Asumiendo linealidad de 0 a 0.8mg/L
    //Mostrar los valores en LCD
    lcd_clear();
    lcd_goto(0x00);
    if (canal==0){
        mgml=((adc[canal])*1.6);
        printf("Alc. %d mg/ml",mgml);
    }
    if (canal==1){
        gas=(adc[canal]);
        printf("Gas: %d ppm",gas);
    }
    T0IF=0;
}
```

The second function of the overflow is to launch the analog measurements. Two tests are performed: one for breathalyzer and the other for gas. Each test is saved in adc[channel (variable)] as previously configured, set to the left. Depending on our channel variable, one or the other will be executed within the interrupt. Remember to set the flag for a subsequent interrupt.



```
//INTERRUPCION PARA EL RESET, CON RB0
if (RB0==0){
    __delay_ms(250); //Antirebotes
    if (RB0==0){
        //Reset de todas las variables y mensaje de reinicio
        T0IE=0;
        T0IF=0;
        INTF=0;
        GO=0;;
        nprueba=0;
        canal=0;
        superado=0;
        gas=0;
        mgml=0;
        mano=0;
        aciertos=0;
        ADCON0=0X89;
        lcd_clear();
        lcd_goto(0x00);
        printf("REINICIANDO...");
        __delay_ms(2000);
    }

}
```

In this “RESET” of our system, a RB0 button is used as an interruption. When this is pressed, all variables are reset, all flags are set to 0, measurements are stopped, and channel 1 is selected. In this way, a complete reset of our configuration occurs..

## 3.8 DIGITAL TERMINALS:

There are three inputs, RA4, RB1, and RB2, for different purposes. RA4 acts as a button that, when pressed, acts as a jumper to skip the test. Otherwise, we'll be left without doing anything, and the display will tell us to press it if we want to continue. The final test checks RB1 and RB2 to complete it optimally.

It's also important to mention the use of RC6 and RC7 for the USART module and PORTD for the LCD.

# 4. IMPLEMENTED TOOLS:

As explained above, each test required a different measurement, so several sensors were needed. The first of these, which allows us to detect alcohol in the driver's breath, is an MQ-3 sensor. A can containing ethanol was used as a simulation to verify the sensor's reliability.

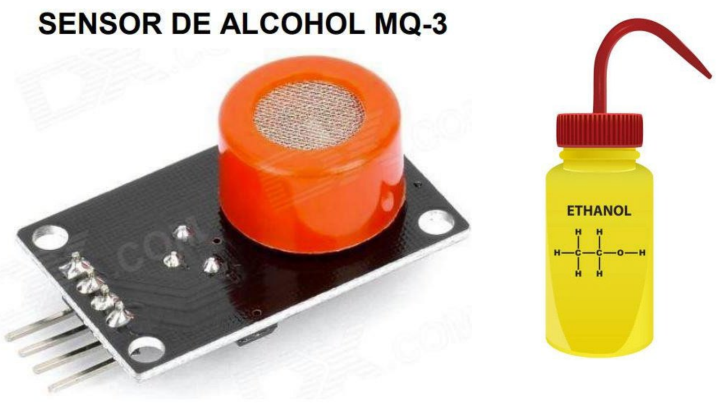


Figure 1: First test tool.

Once this test was completed, we moved on to the second test, which consisted of a smoke/gas test. Thanks to the MQ-2 sensor, we can detect if there is any toxic smoke in the vehicle, if the driver has been smoking, etc., in order to prevent driving risks. A lighter and a piece of paper were enough to activate the sensor and simulate smoke under any circumstances.

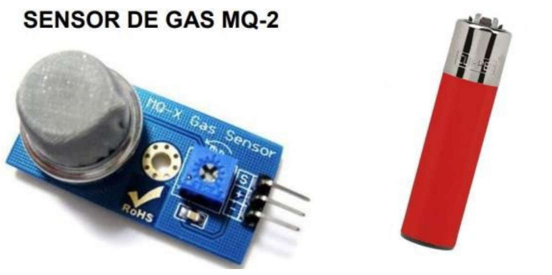


Figure 2: Tools used for the second test.

For the last test, which consisted of checking whether the driver kept both hands on the steering wheel, two sensors were used that give a digital signal when pressed.

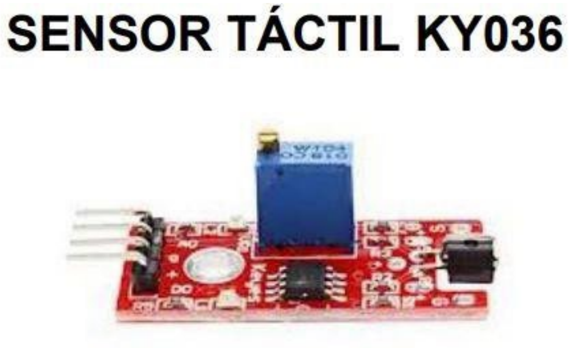


Figure 3: Tools used for the third test.

As for our RGB LED actuator, it will show us a light depending on the result obtained as a response from the system. It has 3 different lights that we use completely.

Figura 4: RGB led Actuator

Finally, the HC-06 module allows us to display the results after each test on the phone, so the user can view it from somewhere other than the LCD, saving only the test result.

Figura 5: Tool for showing the result.

# 5. TERMINAL CONNECTIONS:

| | |
|---|---|
| MQ-3 Sensor | Three of the sensor's terminals are connected to the board:<br>• Vcc pin to +5V<br>• GND pin to 0V<br>• Aout pin to RA1 |
| MQ-2 Sensor | Three of the sensor's terminals are connected to the board:<br>• Vcc pin to +5V<br>• GND pin to 0V<br>• A0 pin to RA2 |
| RGB LED | Four of the sensor's terminals are connected to the board:<br>• GND pin to 0V<br>• R pin to RB4<br>• G pin to RC5<br>• B pin to RC4 |
| KY306 (2 will be used) | Three of the sensor's terminals are connected to the board:<br>• Vcc pin to +5V<br>• GND pin to 0V<br>• D0 pin to RB1/RB2 (depending on which one is used, it will be connected to either RB1 or RB2) |
| HC-06 | Three of the module's terminals are connected to the board:<br>• Vcc pin to +5V<br>• GND pin to 0V<br>• TXD pin to RC7<br>• RXD pin to RC6 |
| RA4 | Manual push button |
| RB0 | Manual push button |

Table 1: Representation of integrated components, pushbuttons and their connections.

# 6. OPERATING DIAGRAM:

## 6.1 MAIN DIAGRAM

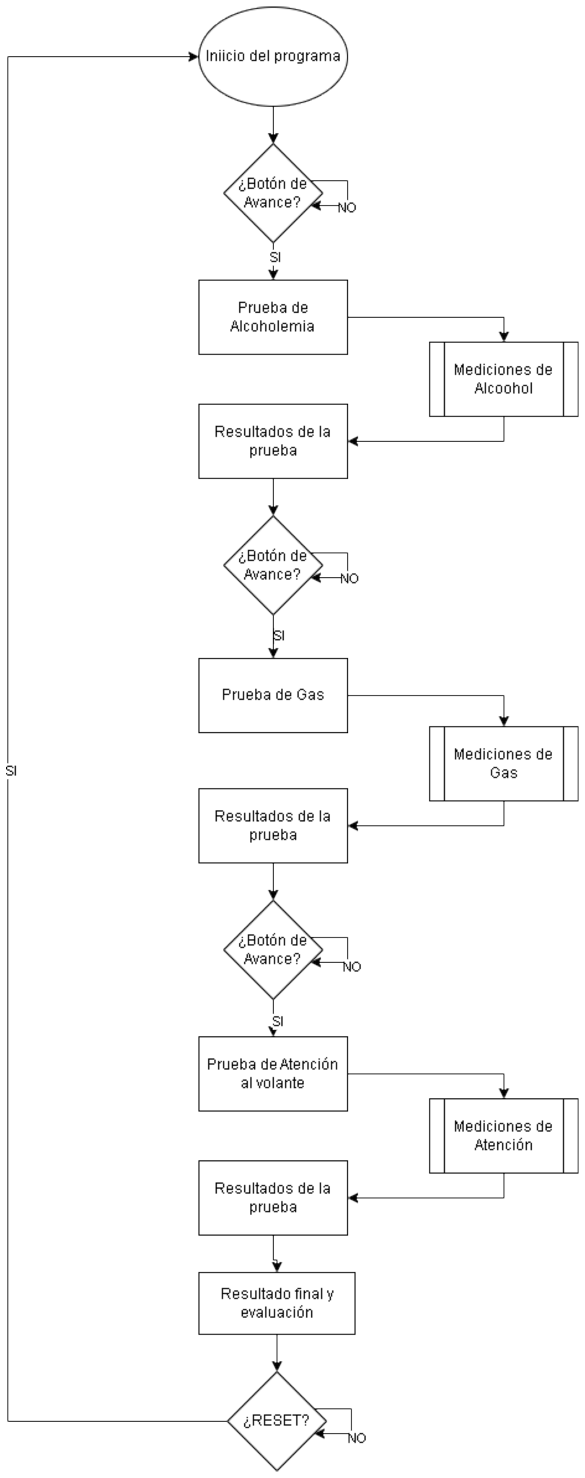
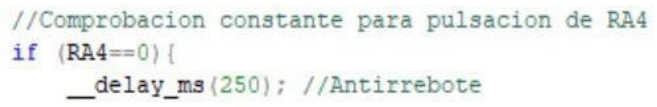


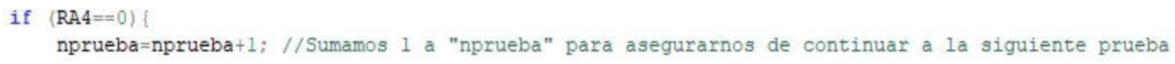
Figure 6: Main flux diagram.

This flowchart is quite straightforward. It involves three tests performed in sequence, with RA4 serving as the button that allows progression. Although the main scheme is simple, there are several functions and variables that change as the program advances.



```
//Comprobacion constante para pulsacion de RA4
if (RA4==0){
    __delay_ms(250); //Antirrebote
```

Debounce handling for the button is also taken into account.

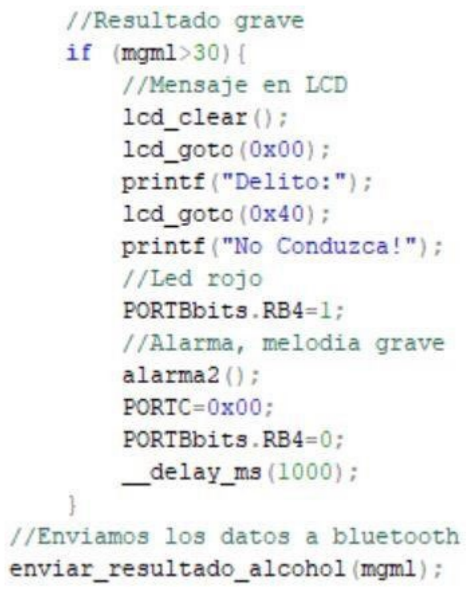


```
if (RA4==0){
    nprueba=nprueba+1; //Sumamos 1 a "nprueba" para asegurarnos de continuar a la siguiente prueba
```

As can be seen in the flowchart, by pressing a button—in this case RA4—we use the variable "nprueba" to determine which test we are currently performing. This variable starts at '0' and increments by 1 with each test.

The measurement subroutines will be explained in the interrupt section, in the following flowchart.

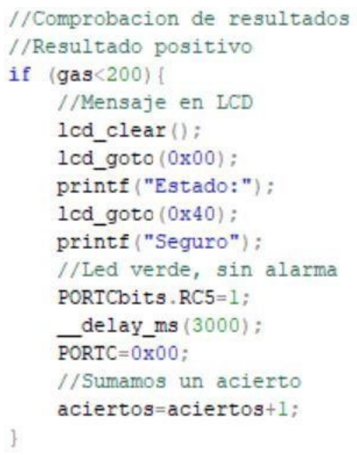
After the measurement, we must evaluate the test results. For analog measurements—alcohol and gas—we have selected three ranges intended to resemble real-world criteria. As an example, for alcohol measurement, we use a simple conditional structure to evaluate the "mgml" value.



```
    //Resultado grave
    if (mgml>30){
        //Mensaje en LCD
        lcd_clear();
        lcd_goto(0x00);
        printf("Delito:");
        lcd_goto(0x40);
        printf("No Conduzca!");
        //Led rojo
        PORTBbits.RB4=1;
        //Alarma, melodia grave
        alarma2();
        PORTC=0x00;
        PORTBbits.RB4=0;
        __delay_ms(1000);
    }
//Enviamos los datos a bluetooth
enviar_resultado_alcohol(mgml);
```

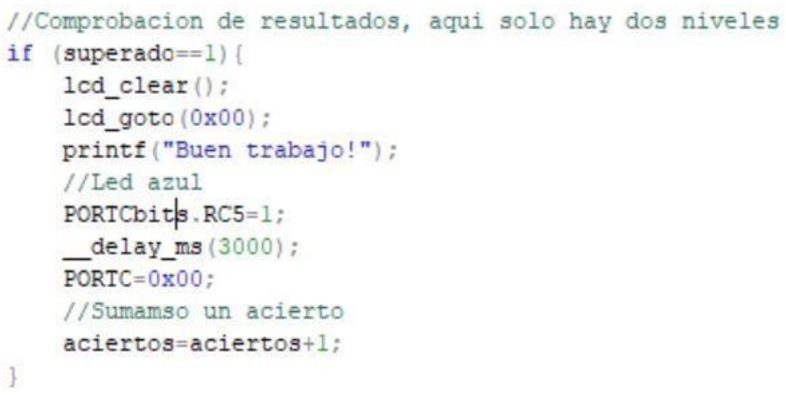
In this case, we evaluate what happens if the measured value exceeds 30mg/L. This range indicates that the driver would exceed the legal alcohol limit. A warning message is displayed to the driver via the LCD, and both an audible and visual alert are activated—a buzzer and a red light on the RGB LED, respectively.

In the intermediate case, the LCD message and alerts are slightly different, indicating a lower level of risk. These alarms and their functions have already been explained in the modules section. Additionally, the measurement result is sent via Bluetooth so the user can access it from their mobile device.



```
//Comprobacion de resultados
//Resultado positivo
if (gas<200){
    //Mensaje en LCD
    lcd_clear();
    lcd_goto(0x00);
    printf("Estado:");
    lcd_goto(0x40);
    printf("Seguro");
    //Led verde, sin alarma
    PORTCbits.RC5=1;
    __delay_ms(3000);
    PORTC=0x00;
    //Sumamos un acierto
    aciertos=aciertos+1;
}
```

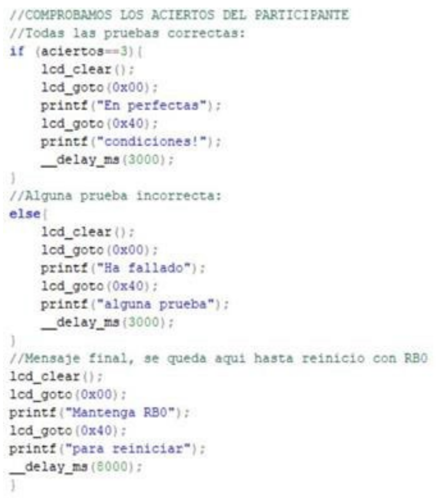
In the case of passing the test, the process is very similar: a message will be displayed on the LCD, and a green LED in the RGB LED will be activated. No alarm will be triggered. Additionally, '1' will be added to the "aciertos" variable, which, as its name suggests, allows us to track the number of tests successfully passed by the driver.



```
//Comprobacion de resultados, aqui solo hay dos niveles
if (superado==1){
    lcd_clear();
    lcd_goto(0x00);
    printf("Buen trabajo!");
    //Led azul
    PORTCbits.RC5=1;
    __delay_ms(3000);
    PORTC=0x00;
    //Sumamso un acierto
    aciertos=aciertos+1;
}
```

While our result evaluation for Alcohol and Gas is practically identical, this evaluation is somewhat different. Since it is a digital measurement, we will obtain either a positive or negative result, represented by the value of the "superado" variable, where '1' means the test was passed and '0' means it was failed.

This means there are only two possible outcomes in the evaluation: correct or incorrect. The internal process is similar to the previous ones, triggering a visual and sound alert in case of failure, and incrementing the "aciertos" (successes) variable in case of success.



```
//COMPROBAMOS LOS ACIERTOS DEL PARTICIPANTE
//Todas las pruebas correctas:
if (aciertos==3){
    lcd_clear();
    lcd_goto(0x00);
    printf("En perfectas");
    lcd_goto(0x40);
    printf("condiciones!");
    __delay_ms(3000);
}
//Alguna prueba incorrecta:
else{
    lcd_clear();
    lcd_goto(0x00);
    printf("Ha fallado");
    lcd_goto(0x40);
    printf("alguna prueba");
    __delay_ms(3000);
}
//Mensaje final, se queda aqui hasta reinicio con RB0
lcd_clear();
lcd_goto(0x00);
printf("Mantenga RB0");
lcd_goto(0x40);
printf("para reiniciar");
__delay_ms(8000);
}
```

In this final part of the main function, the number of successfully completed tests is evaluated using the variable "aciertos". Regardless of whether the user has passed all tests correctly or not, the driver will be given the option to repeat the entire process by pressing a button that acts as an interrupt. This will be discussed in more detail later.

## 6.2 INTERRUPTION DIAGRAM:

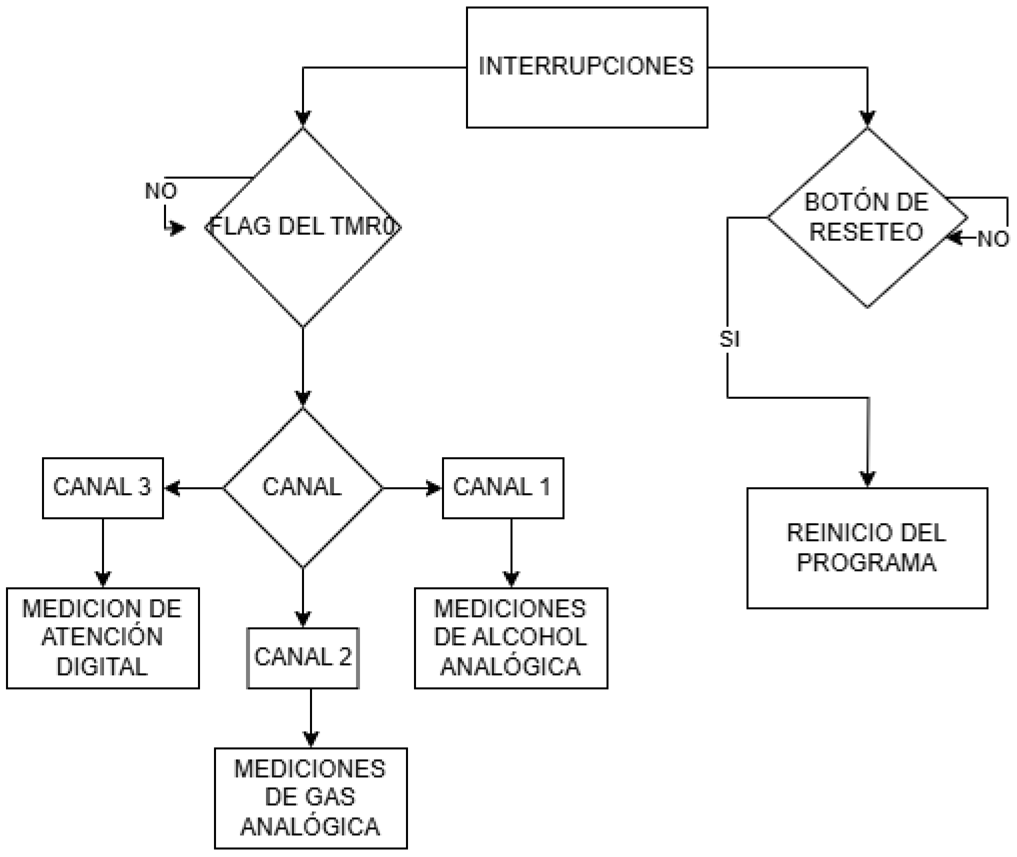


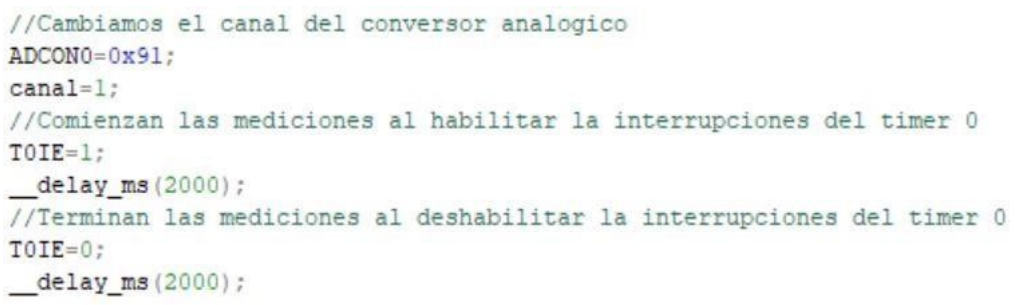
Figure 7: Interruption flux diagram

As indicated in our flowchart, our measurements are divided into two main groups:

- TIMER0 interrupts , which we use for measurements, and
- RB0 interrupts , which we use for system reset.

## 6.2.1 TIMER0 INTERRUPTION:

TIMER0 is responsible for the measurement process. Within this process, there are two types of measurements: analog and digital . Although both follow a very similar scheme, it is important to highlight the role of the ADCON0 register in our measurements. Since we have two different analog measurements — alcohol and gas — connected to RA1 and RA2 respectively, we must configure this register accordingly.

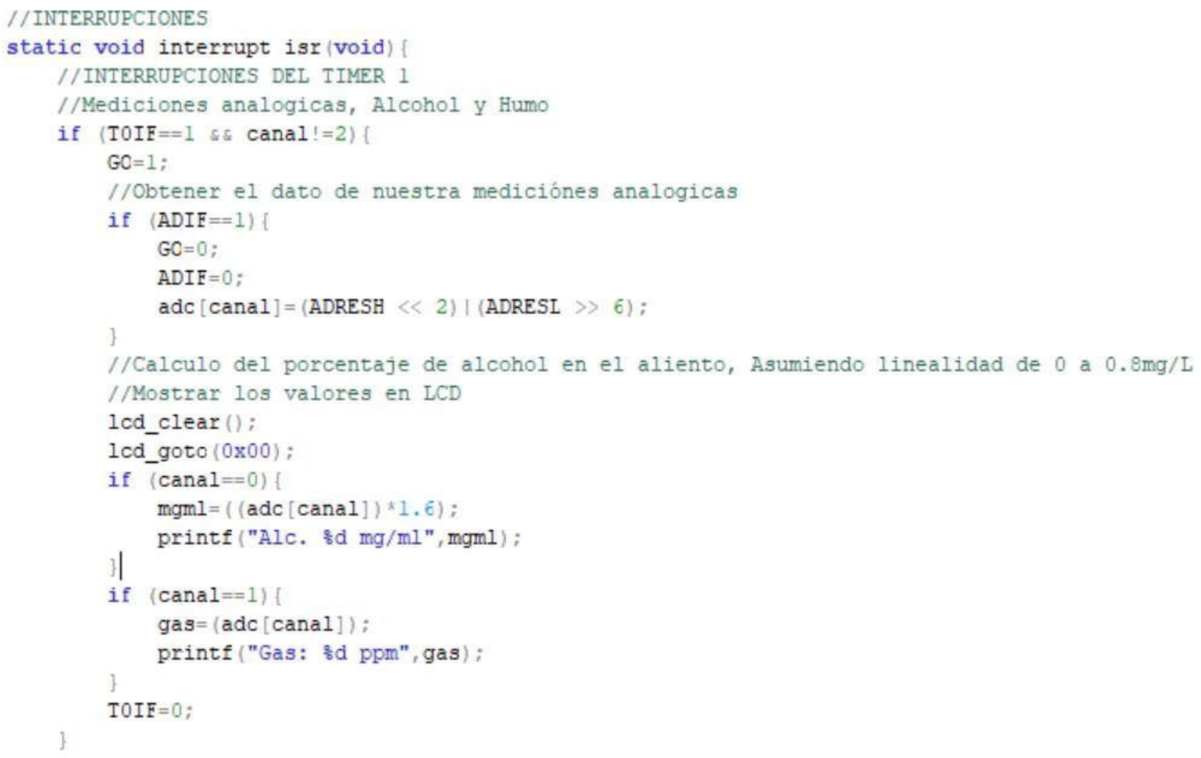


```
//Cambiamos el canal del conversor analogico
ADCON0=0x91;
canal=1;
//Comienzan las mediciones al habilitar la interrupciones del timer 0
T0IE=1;
__delay_ms(2000);
//Terminan las mediciones al deshabilitar la interrupciones del timer 0
T0IE=0;
__delay_ms(2000);
```

If we look at the code above, which corresponds to the alcohol measurement, we can see that the first step is configuring ADCON0, specifically bits 3, 4, and 5, which indicate the selected channel. In this case, the input is RA1, so the channel is '1', corresponding to the hexadecimal value '0x91'. For gas measurement, the ADCON0 channel must be '2', which corresponds to '0xA1'.

It's also important to manage the variable 'canal' (channel), as it tells us where to store the measurement values. Thanks to this variable, we can use a single function for both measurements.

Now that the channel is set and we know where to store the data, we need to start the measurement process. This process is the same for both analog and digital measurements. We begin by enabling Timer0 interrupts through the T0IE bit.

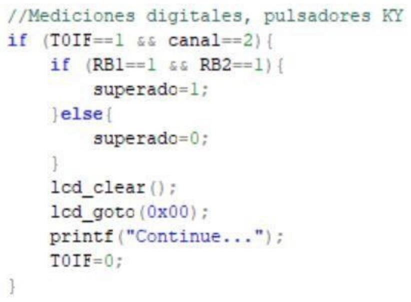


```
//INTERRUPCIONES
static void interrupt isr(void){
    //INTERRUPCIONES DEL TIMER 1
    //Mediciones analogicas, Alcohol y Humo
    if (T0IF==1 && canal!=2){
        GO=1;
        //Obtener el dato de nuestra mediciónes analogicas
        if (ADIF==1){
            GO=0;
            ADIF=0;
            adc[canal]=(ADRESH << 2)|(ADRESL >> 6);
        }
        //Calculo del porcentaje de alcohol en el aliento, Asumiendo linealidad de 0 a 0.8mg/L
        //Mostrar los valores en LCD
        lcd_clear();
        lcd_goto(0x00);
        if (canal==0){
            mgml=((adc[canal])*1.6);
            printf("Alc. %d mg/ml",mgml);
        }
        if (canal==1){
            gas=(adc[canal]);
            printf("Gas: %d ppm",gas);
        }
        T0IF=0;
    }
```

Once the interrupt is enabled, our system will take several measurements within a given time frame. The timing is determined by the "__delay_ms()" function, along with the time required for each individual measurement. The number of samples is related to the TIMER0 overflow period.

Each time TIMER0 overflows, the T0IF flag triggers the measurement process. This process varies depending on whether the measurement is analog or digital, and it returns the latest value obtained in the ADRESH and ADRESL registers for further evaluation. It is a straightforward measurement method: the ADC is initiated by setting the GO bit, and the conversion result is retrieved once ADIF is triggered.

It is worth noting that these measurements have been linearized for simplicity; however, in reality, the values follow more complex logarithmic and exponential patterns.

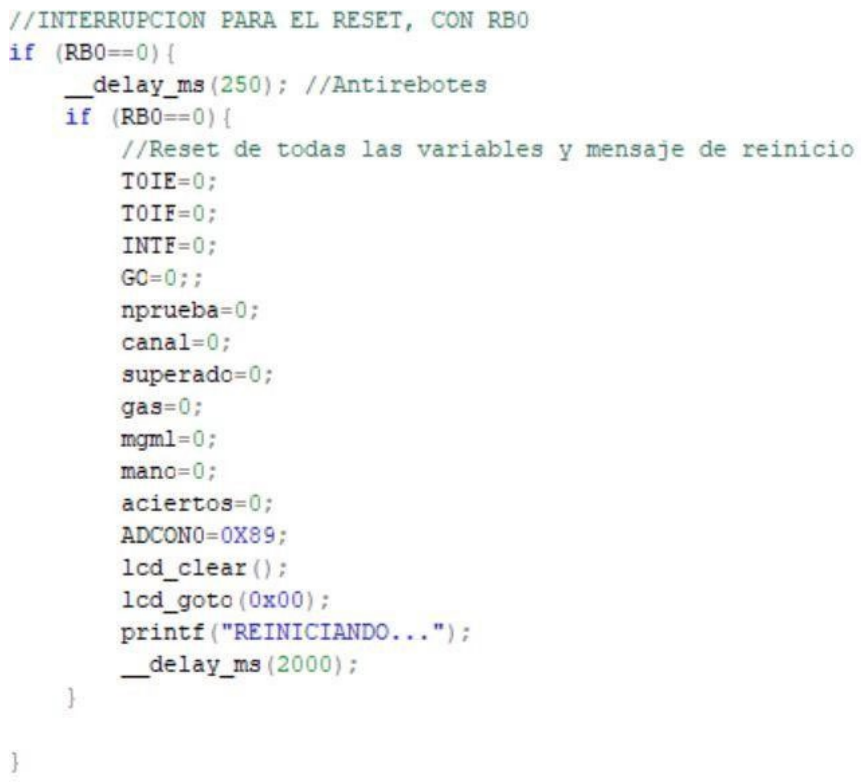


```
//Mediciones digitales, pulsadores KY
if (T0IF==1 && canal==2){
    if (RB1==1 && RB2==1){
        superado=1;
    }else{
        superado=0;
    }
    lcd_clear();
    lcd_goto(0x00);
    printf("Continue...");
    T0IF=0;
}
```

Digital measurement is much simpler, as it does not require any adjustments in ADCON0 (since it is digital), but it still follows the same sampling scheme using TIMER0.

### 6.2.2 RB0 INTERRUPTION:

The second type of interrupt is triggered by pressing the RB0 button, which will cause a COMPLETE RESET of our program, returning to its initial state.



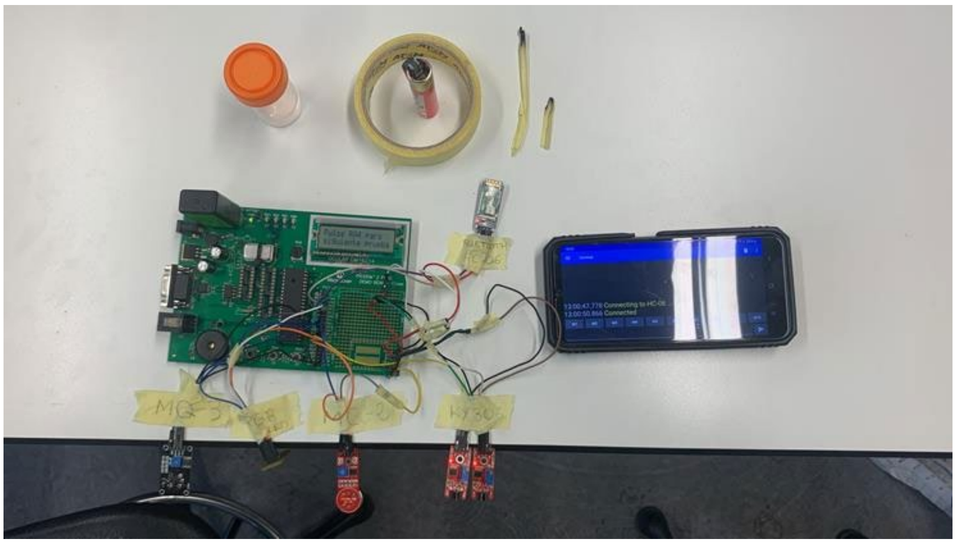
```
//INTERRUPCION PARA EL RESET, CON RB0
if (RB0==0){
    __delay_ms(250); //Antirebotes
    if (RB0==0){
        //Reset de todas las variables y mensaje de reinicio
        T0IE=0;
        T0IF=0;
        INTF=0;
        GO=0;;
        nprueba=0;
        canal=0;
        superado=0;
        gas=0;
        mgml=0;
        mano=0;
        aciertos=0;
        ADCON0=0X89;
        lcd_clear();
        lcd_goto(0x00);
        printf("REINICIANDO...");
        __delay_ms(2000);
    }

}
```

As can be seen, once it is confirmed that the button has been pressed, all variables are reset to zero to restart the program from the beginning, prior to the initial test. For this reason, the ADCON0 register is set to the corresponding channel. Regarding the TIMER0 interrupts, the T0IF flag is cleared to allow the next interrupt, and the external T0IE interrupts are disabled to prevent measurements until required. Finally, the INTF flag is cleared to properly handle future interrupts and prevent it from triggering continuously.

## 7. EXPERIMENTAL SETUP:

Finally, we include some images of our completed project.



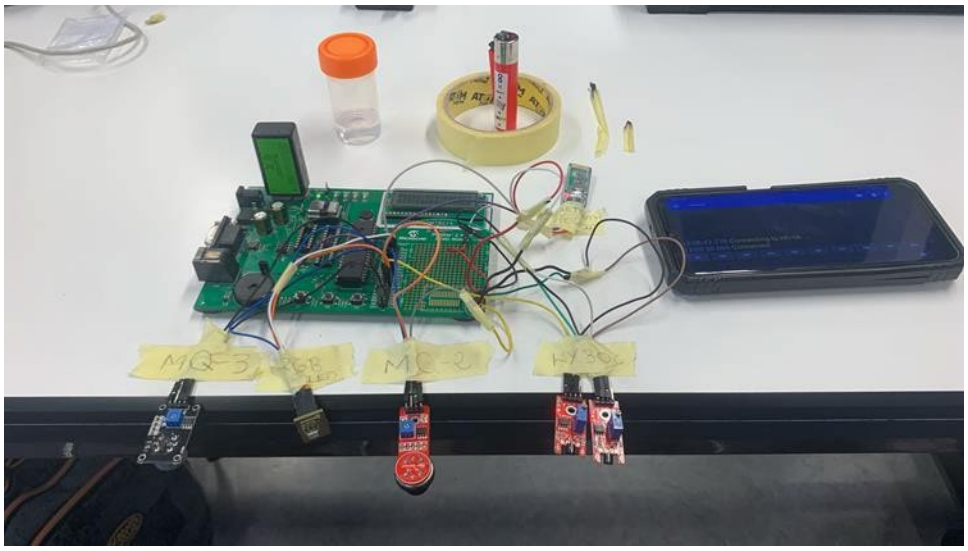
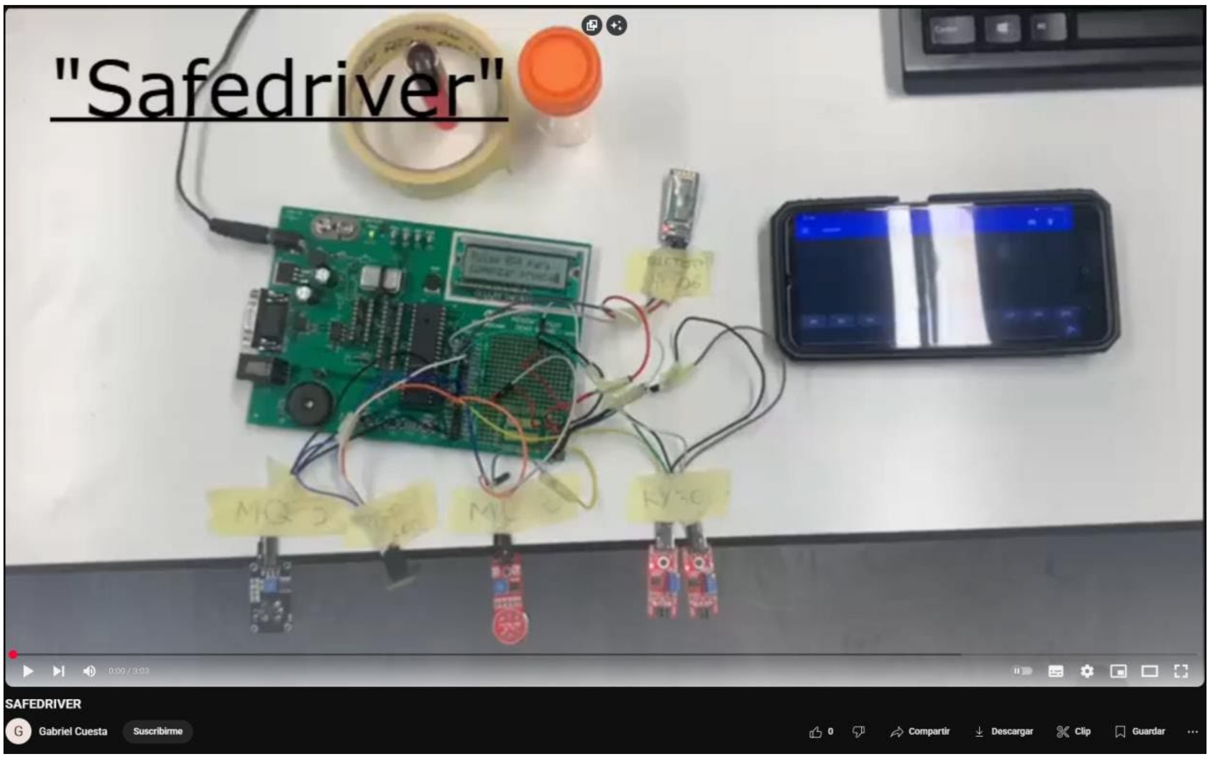


Figura 8: Experimental setup images

## 8.EXPLANATORY VIDEO:

Finally, we have a short 3-minute video that explains and demonstrates the system's operation. It is uploaded to YouTube and is publicly accessible.



https://youtu.be/7H1O3eaTX_g

# 9.CONCLUSIONS:

All proposed specifications have been fully met. Difficulties arose throughout the project, such as sensor calibration, for example.

Despite the non-linearity of the selected sensors, we were able to correctly simulate their operation by considering them as linear. However, this would be an important aspect to address in a more in-depth project. Another relevant point is that we initially intended to perform the attention test using the sensors in analog mode. This had to be adjusted due to encountered issues, leading us to explore the option of using them in digital mode, which successfully achieved the desired result.

Ultimately, this was a highly rewarding project from which we learned and gained a deeper understanding of both the PIC16F877A microcontroller and the programming language used.