Solución del Examen final

Problema 1. Acondicionamiento de sensores (4 puntos)

1.1. Sensor de temperatura RTD (2 puntos)

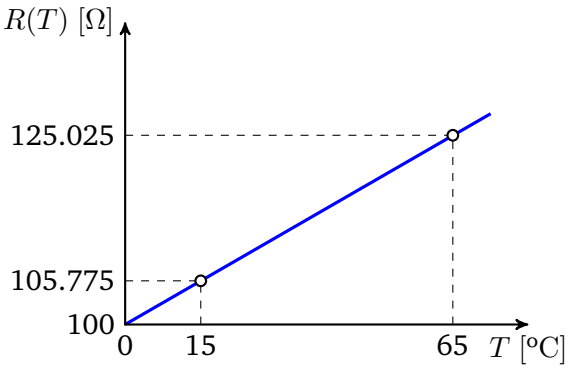
a) **(1,5 puntos)** Al tratarse de un apartado de diseño libre existen muchas soluciones diferentes. A continuación se presentan varias posibilidades.

Se trata de un sensor lineal, cuya resistencia en los extremos del rango de medida (15ºC y 65ºC) es:

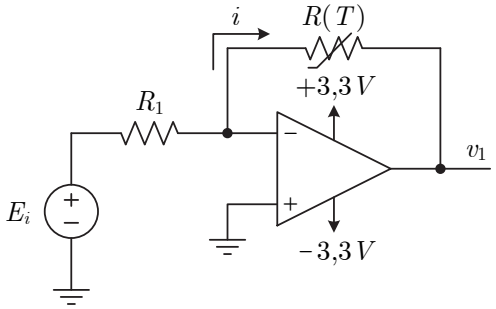
$$R(T) = R_0 \cdot (1 + \alpha \cdot \Delta T)$$

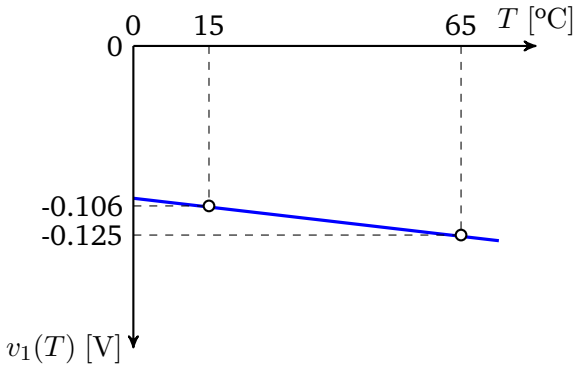
$$R(T = 15\,^{\circ}\mathrm{C}) = 105,775\,\Omega$$

$$R(T = 65\,^{\circ}\mathrm{C}) = 125,025\,\Omega$$



Se puede acondicionar el sensor como si fuera de grandes o pequeñas variaciones (véase la respuesta del apartado b) para una justificación detallada). Si se considera que se trata de un sensor de **grandes variaciones**, el primer paso es convertir la resistencia en tensión utilizando una **fuente de corriente**. Generalmente se hace circular una intensidad lo más grande posible por el sensor con el objetivo de amplificar menos en las etapas siguientes, lo cual reduce los errores a la salida debidos a los operacionales y a las tolerancias de las resistencias. Dado que la corriente máxima para poder despreciar el efecto del autocalentamiento es de 1 mA:

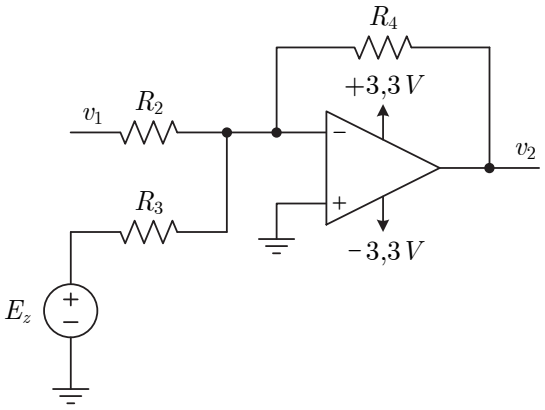


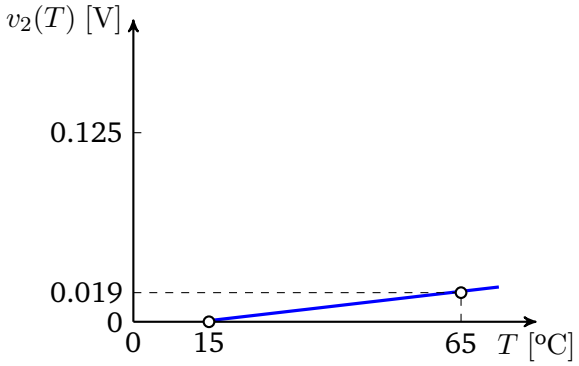


$$i = \frac{E_i}{R_1} = 1\,\mathrm{mA} \Rightarrow \boxed{E_i = 3,3\,\mathrm{V}} \quad \boxed{R_1 = 3,3k}$$

$$v_1 = -\frac{E_i}{R_1} \cdot R(T) \Rightarrow \begin{cases} v_1(T = 15\,^{\circ}\mathrm{C}) = -0,106\,\mathrm{V} \\ v_1(T = 65\,^{\circ}\mathrm{C}) = -0,125\,\mathrm{V} \end{cases}$$

A continuación hay que realizar el **ajuste de cero**, para que a 15ºC haya 0 V. Lo más sencillo es emplear un sumador con ganancia 1 V/V para todas las entradas. En este caso se ha optado por una configuración inversora.





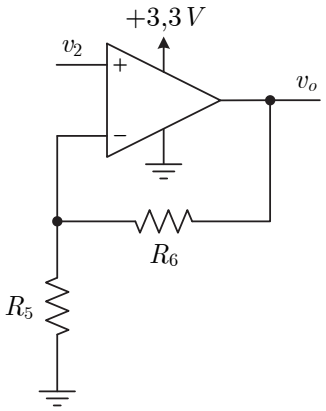
$$R_2 = R_3 = R_4 = \boxed{10k}$$

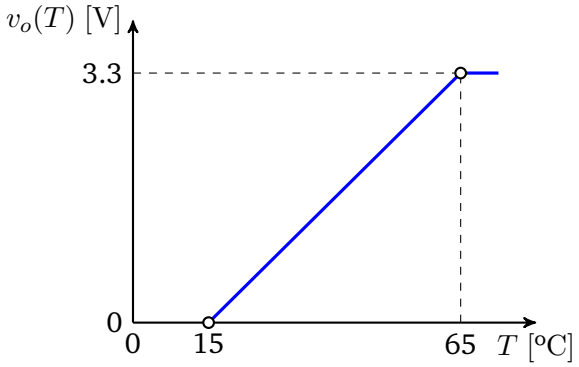
$$v_2 = -(v_1 + E_z)$$

$$v_2(T = 15\,^{\circ}\text{C}) = -(-0,106 + E_z) = 0 \Rightarrow \boxed{E_z = 0,106\,\text{V}}$$

Finalmente, es necesario amplificar para conseguir la **máxima sensibilidad** a la salida. Por tanto, a 65ºC debe haber 3,3 V, que es la tensión de saturación del operacional y el valor máximo que puede leer el conversor A/D del microcontrolador.

Observación: Es importante que la última etapa esté alimentada entre 0 y 3,3 V para que no exista riesgo de dañar el conversor A/D si la temperatura sale fuera del rango de diseño.





$$v_o = v_2 \cdot \left(1 + \frac{R_6}{R_5}\right)$$

$$v_o(T = 65\,^{\circ}\text{C}) = 0,019 \cdot \left(1 + \frac{R_6}{R_5}\right) = 3,3 \Rightarrow \frac{R_6}{R_5} = 170,43 \Rightarrow \boxed{R_5 = 3,3k} \quad \boxed{R_6 = 560k}$$

El inconveniente de este diseño es que al ser la ganancia de la última etapa tan elevada, cualquier error en el ajuste de cero se refleja a la salida multiplicado por 171.

El circuito final, resulta:

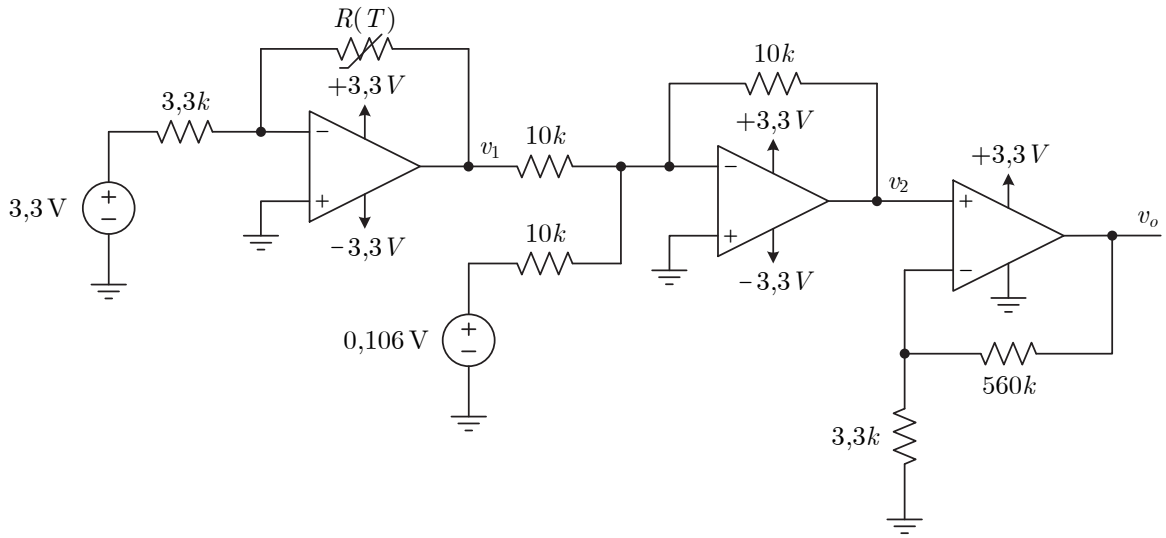


Figura 1. Circuito de acondicionamiento para el sensor RTD si se considera de grandes variaciones.

Se puede eliminar el último operacional si se utiliza un sumador con ganancias distintas para cada entrada:

$$v_o = v_2 = -R_4 \cdot \left(\frac{v_1}{R_2} + \frac{E_z}{R_3} \right) \Rightarrow \begin{cases} v_2(T = 15\,^{\circ}\text{C}) = 0\,\text{V} \\ v_2(T = 65\,^{\circ}\text{C}) = 3,3\,\text{V} \end{cases}$$

Hay dos grados de libertad. Una posible solucion es:

$$\boxed{R_2 = 1,5k \quad R_3 = 47k \quad R_4 \approx 250k \quad E_z = 3,3\,\text{V}}$$

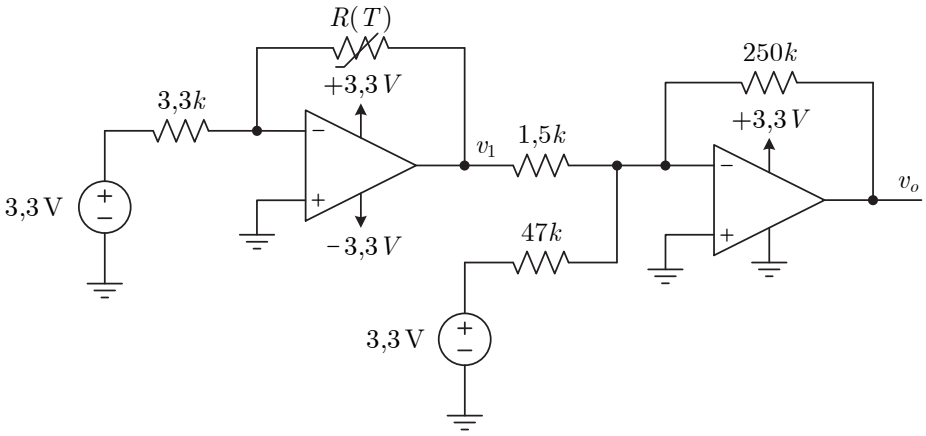
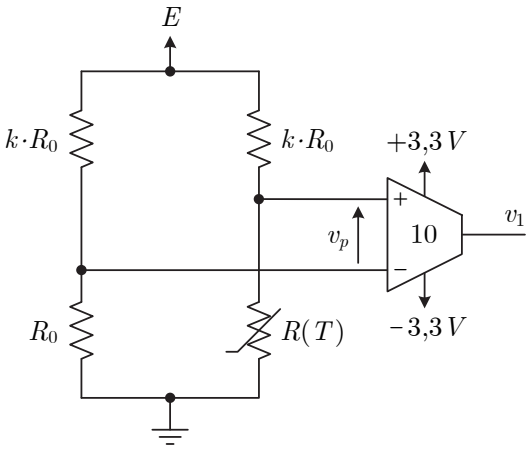


Figura 2. Circuito de acondicionamiento para el sensor RTD con sólo dos amplificadores operacionales.

Observación: También es posible alimentar el sumador con tensión simétrica y colocar un *buffer* alimentado a tensión simple a la salida para proteger el conversor A/D.

Como se ha mencionado anteriormente, otra alternativa es acondicionar el sensor como uno de **pequeñas variaciones**. Para mejorar la aproximación lineal del puente se va a tomar un valor de k grande, por ejemplo $\boxed{k = 10}$. También se fijará $\boxed{E = 3,3\,\text{V}}$.



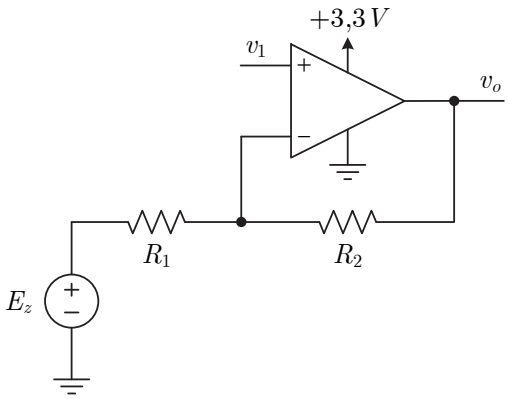
$$v_p = E \cdot \frac{k}{(k+1)^2} \cdot \alpha \cdot \Delta T$$

$$v_p(T = 15\,^{\circ}\text{C}) = 15,75\,\text{mV}$$

$$v_p(T = 65\,^{\circ}\text{C}) = 68,25\,\text{mV}$$

$$v_1 = 10 \cdot v_p \Rightarrow \begin{cases} v_p(T = 15\,^{\circ}\text{C}) = 157,5\,\text{mV} \\ v_p(T = 65\,^{\circ}\text{C}) = 682,5\,\text{mV} \end{cases}$$

Finalmente, hay que conseguir que a 15ºC haya 0 V y a 65ºC, 3,3 V. Se pueden cumplir ambas restricciones con un sumador como el siguiente:



$$v_o = v_1 \cdot \left(1 + \frac{R_2}{R_1}\right) - E_z \cdot \frac{R_2}{R_1}$$

$$\begin{cases} v_o(T = 15\,^{\circ}\text{C}) = 0\,\text{V} \\ v_o(T = 65\,^{\circ}\text{C}) = 3,3\,\text{V} \end{cases}$$

$$\boxed{R_1 = 10k \quad R_2 \approx 53k \quad E_z = 0,187\,\text{V}}$$

El circuito definitivo quedaría como se muestra en la Figura 3.

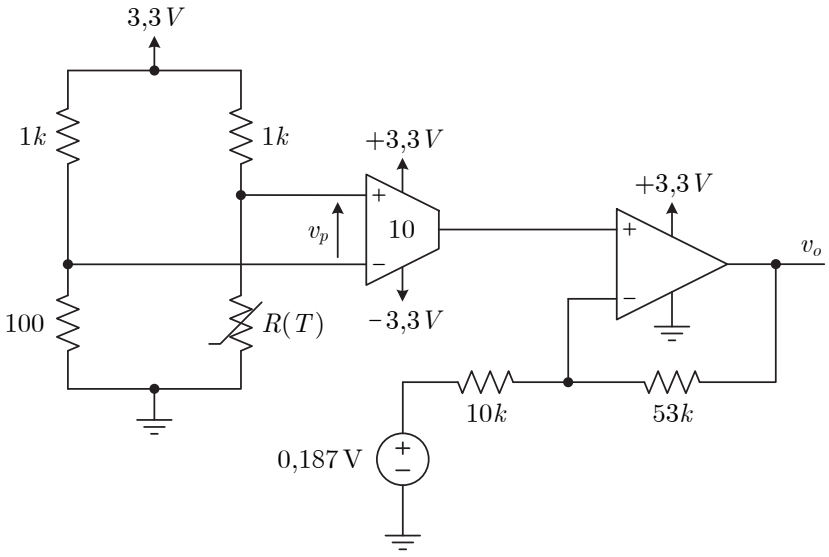


Figura 3. Circuito de acondicionamiento para el sensor RTD si se considera de pequeñas variaciones.

b) **(0,5 puntos)** Si se calcula la relación entre la variación de la resistencia del sensor en el rango de medida deseado y el valor medio de la resistencia en ese rango:

$$\frac{R_{\text{max}} - R_{\text{min}}}{R_{\text{media}}} = \frac{125,025 - 105,775}{115,400} = 0,167$$

Para que un sensor resistivo se considere de pequeñas variaciones la regla práctica es que el resultado de la ecuación anterior sea mucho menor que 1. Por norma general, eso significa que debe ser al menos diez veces inferior. Como no es el caso, se puede acondionar como si se tratara de un sensor de grandes variaciones. No obstante, dado que el valor es bastante próximo a 0,1 podría ser aconsejable utilizar un puente de Wheatstone para mejorar la exactitud de la medida.

1.2. Detector de humo (2 puntos)

a) **(1 punto)** Para anular la corriente de oscuridad hay que conseguir que v_x sea insensible a I_0.

$$v_1 = -R_g \cdot I_0$$

$$v_2 = R_g \cdot (2I_0 + k \cdot P_{L_2} + k \cdot P_{L_3})$$

$$v_x = v_1 \cdot \frac{R_2}{R_1 + R_2} + v_2 \cdot \frac{R_1}{R_1 + R_2}$$

$$\frac{\partial v_x}{\partial I_0} = \frac{R_g}{R_1 + R_2} \cdot (2R_1 - R_2) = 0 \Rightarrow R_2 = 2R_1 \Rightarrow \boxed{R_1 = 10k} \quad \boxed{R_2 = 20k}$$

b) **(1 punto)** Si P_2 y P_3 miden una obscuración del 0 % (0 lux), ya hay 0 V a la salida, por lo que sólo es necesario resolver el caso del otro extremo, que cuando la obscuración media sea del 2 %, v_o valga 3,3 V.

$$v_o = v_x \cdot \left(1 + \frac{R_4}{R_3}\right) = 3,3\,\text{V}$$

Eso significa que **ambos fotodiodos** tienen que medir una obscuración del 2 % (50 lux) o cualquier combinación que en media dé un 2 %, por ejemplo, que P_2 mida un 1 % de obscuración y P_3, un 3 %. Como ya se ha eliminado la corriente de oscuridad I_0, la tensión a la salida es:

$$v_o = R_g \cdot k \cdot (P_{L_2} + P_{L_3}) \cdot \frac{R_1}{R_1 + R_2} \cdot \left(1 + \frac{R_4}{R_3}\right) = 3,3$$

$$v_o = 10^6 \cdot 3 \cdot 10^{-9} \cdot (50 + 50) \cdot \frac{1}{3} \cdot \left(1 + \frac{R_4}{R_3}\right) = 3,3 \Rightarrow \frac{R_4}{R_3} = 32 \Rightarrow \boxed{R_3 = 1k} \quad \boxed{R_4 \approx 33k}$$

Problema 2. Sistema digital (6 puntos)

a) **(0,5 puntos)** Existen varias formas de conectar las entradas y salidas digitales, todas ellas igualmente válidas. En este caso, se ha optado por una configuración *pull-up* para el pulsador y el LED. En cuanto al motor que mueve el ventilador, sólo necesita girar en un sentido, por lo que no es necesario utilizar un puente en H, si bien, también sería posible.

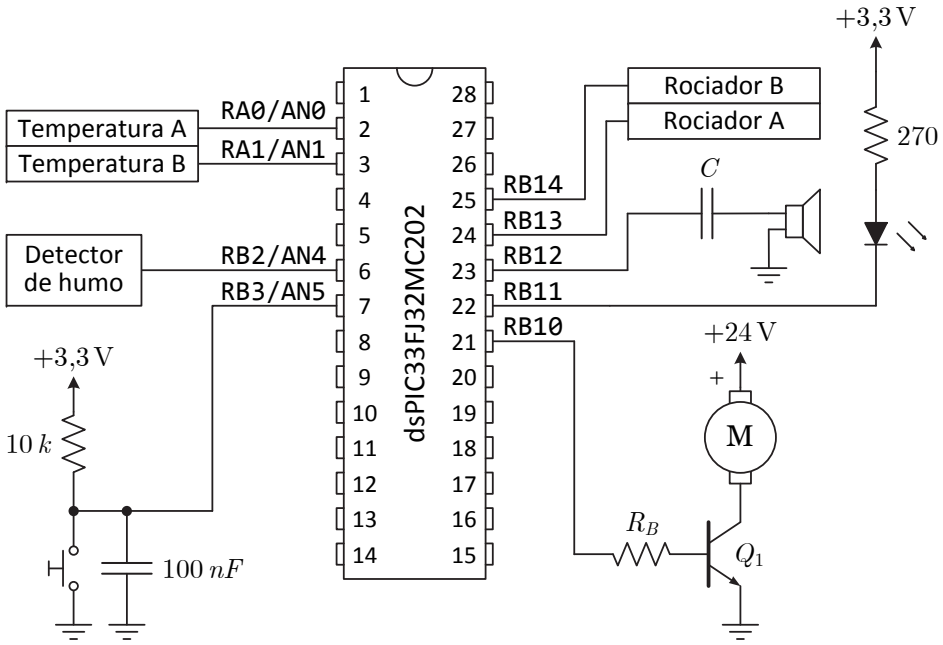


Figura 4. Esquema electrónico del sistema de extracción de humos y extinción de incendios.

b) **(0,5 puntos)** Aunque se podría utilizar un driver genérico que permitiera manejar todos los pines analógicos, se ha adoptado la solución más sencilla de controlar únicamente AN0, AN1 y AN4.

adc.h

```
#ifndef _ADC_H
#define _ADC_H

/**
 * Función de inicialización del conversor A/D.
 *
 * Utiliza los pines AN0, AN1 y AN4.
 */
void inicializarADC(void);

/**
 * Devuelve la última medida de un pin analógico.
 *
 * Si aún no se ha realizado ninguna conversión A/D devuelve 0.
 *
 * @param  pin_analogico  Índice del pin {0, 1, 4}.
 *
 * @return Valor medido.
 */
unsigned int getMedidaADC(unsigned char pin_analogico);

#endif
```

adc.c

```c
#include <p33FJ32MC202.h>
#include "adc.h"

#define NUM_PINES_AD 6

static unsigned int medida_adc[] = {0, 0, 0, 0, 0, 0};

void inicializarADC(void) {
   // Configurar AN0, AN1 y AN4 como entradas analógicas
    AD1PCFGL &= ~0x0013;
    TRISA |= 0x0003;
    TRISB |= 1 << 2;

    AD1CON3 = 0x0105;     // Muestreo: 1 ciclo, ADCS = 5
    IFS0bits.AD1IF = 0;   // Borrar la bandera
    IEC0bits.AD1IE = 1;   // Habilitar interrupciones
    IPC3bits.AD1IP = 4;   // Prioridad interrupciones
    AD1CON1 = 0x80E0;     // ON, conversión automática

    AD1CHS0 = 0;          // Seleccionar AN0 para empezar
    AD1CON1 |= 1 << 1;    // Empezar a muestrear
}

unsigned int getMedidaADC(unsigned char pin_analogico) {
    if (pin_analogico >= NUM_PINES_AD)
        return 0;

    return medida_adc[pin_analogico];
}

void __attribute__((interrupt, no_auto_psv)) _ADC1Interrupt(void) {
    static unsigned char pin_ad = 0;

    IFS0bits.AD1IF = 0;              // Borrar la bandera
    medida_adc[pin_ad] = ADC1BUF0;   // Guardar la medida

    // Buscar el siguiente pin que hay que convertir
    if (pin_ad == 0)
        pin_ad = 1;
    else if (pin_ad == 1)
        pin_ad = 4;
    else
        pin_ad = 0;

    AD1CHS0 = pin_ad;    // Seleccionar el nuevo pin A/D
    AD1CON1 |= 1 << 1;   // Empezar a muestrear
}
```

c) **(1 punto)** El sistema tiene dos procesos completamente independientes:

- Como se deriva de los dos primeros requisitos de las especificaciones de *software*, el **extractor** cuenta con un único actuador, el ventilador, cuyo estado depende sólo de la medida del detector de humos.
- El **sistema de extinción** de incendios, que utiliza la medida de los dos sensores de temperatura como entrada para controlar los rociadores, la luz de emergencia y el altavoz, así como el pulsador de desactivación.

Por tanto, lo más sencillo es dividir el problema en dos módulos: *extractor.h* y *extintor.h*. El primero de ellos no necesita máquina de estados porque la salida (la velocidad del ventilador), depende directamente de la entrada (la obscuracion medida) y no hay que tener en cuenta estados anteriores, es decir, no hace falta que el sistema tenga memoria.

En cuanto al sistema de extinción, la Figura 5 muestra el **diagrama de estados** para una zona, por ejemplo, la A. Como las zonas son independientes, la B tiene su propia máquina de estados análoga. La única transición que no aparece reflejada es que si cualquiera de los dos sectores llega al estado 3 (emergencia), que es común a ambas zonas, la otra zona pasa automáticamente al estado 3.

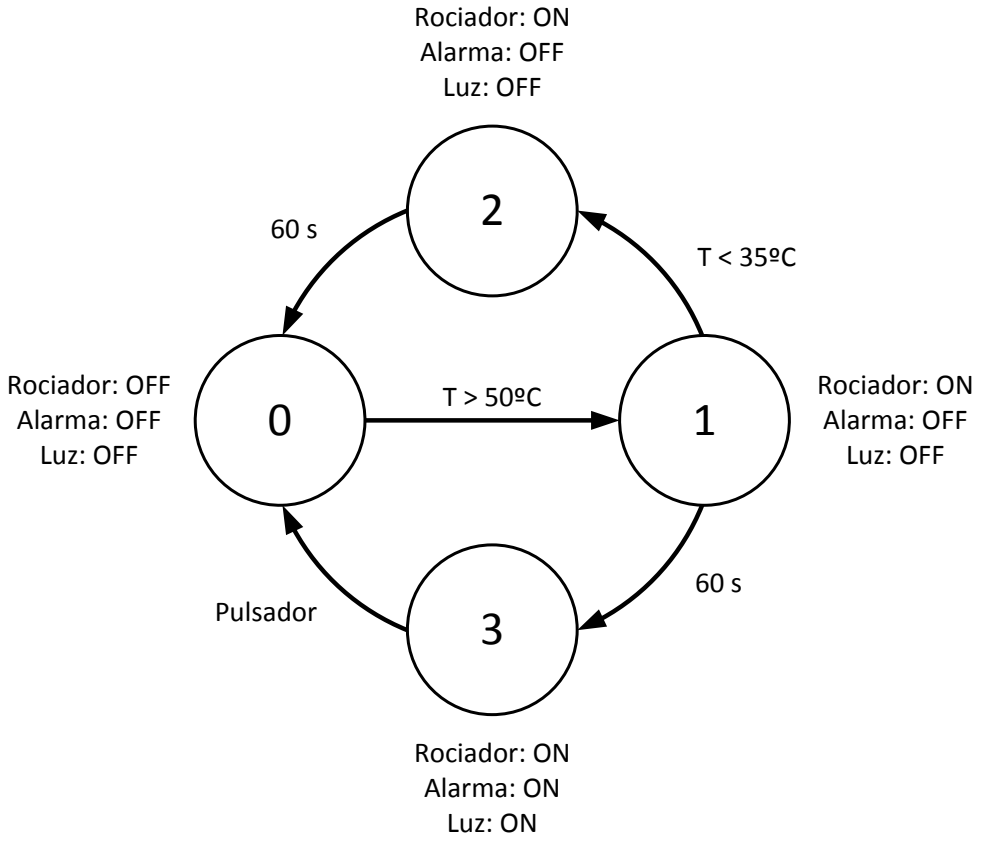


Figura 5. Diagrama de estados de una zona del sistema de extinción.

Para el extractor no es necesario ningún **timer** porque se utilzará el driver del periférico generador de PWM de *motor.h*. En cambio, para el sistema de extinción se usará un único timer para generar la señal de PWM que hace sonar la alarma bitonal y para contar los tiempos de las transiciones. El máximo común divisor de dichas señales determina el periodo del timer, que será de **0,5 ms** porque hay que generar señales de 1 kHz (T = 1 ms) con un factor de servicio del 50 %.

d) **(4 puntos)**

extractor.h

```
#ifndef _EXTRACTOR_H
#define _EXTRACTOR_H

/**
 * Inicializa el sistema de extracción de humos.
 *
 * Utiliza los pines RB2 y RB10 y el generador de PWM.
 */
void inicializarExtractor(void);

/**
 * Controla la velocidad del ventilador en función del humo.
 */
void tareaExtractor(void);

#endif
```

extractor.c

```
#include <p33FJ32MC202.h>
#include "extractor.h"
#include "adc.h"
#include "motor.h"

#define PIN_DETECTOR_HUMO 4  // AN4

void inicializarExtractor(void) {
    configurarPinMotor(PWM_H3);
    inicializarMotor();
    dcMotor(3, 0);
}

void tareaExtractor(void) {
    unsigned int obscuracion;  // 1023 equivale al 2%
    unsigned int dc = 0;

    obscuracion = getMedidaADC(PIN_DETECTOR_HUMO);

    if (obscuracion >= 103) { // 103 equivale al 0,2%
      // Factor de servicio = (100 / 920) * (obscuracion - 103)
      // Se simplifica la expresión para que asegurarse de que
      // todos los resultados intermedios caben en 16 bits.
        dc = (5 * (obscuracion - 103)) / 46;
   }

   dcMotor(3, dc);
}
```

extintor.h

```
#ifndef _EXTINTOR_H
#define _EXTINTOR_H

/**
 * Inicializa el sistema de extinción de incendios.
 *
 * Utiliza el Timer 1 y los pines RA0, RA1, RB3 y RB11 a RB14.
 */
void inicializarExtintor(void);

/**
 * Máquina de estados del sistema de extinción.
 */
void tareaExtintor(void);

#endif
```

extintor.c

```
#include <p33FJ32MC202.h>
#include "extintor.h"
#include "adc.h"

#define NUM_ZONAS       2
#define PIN_PULSADOR    3  // RB3
#define PIN_LUZ        11  // RB11
#define PIN_ALTAVOZ    12  // RB12
#define PIN_ROCIADOR_A 13  // RB13

// Contadores de tiempo: Rociador A = 0, Rociador B = 1
static unsigned int ticks[]    = {0, 0};     // Equivalen a 0,5 ms
static unsigned int segundos[] = {0, 0};
static unsigned char alarma_encendida = 0;  // ON = 1, OFF = 0

unsigned char comprobarPulsador(void);
void apagarRociador(unsigned char zona);
void encenderRociador(unsigned char zona);
void apagarLuz(void);
void encenderLuz(void);

void inicializarExtintor(void) {
   AD1PCFGL |= 1 << 5;
   TRISB &= ~0x7800;
   apagarRociador(0);
   apagarRociador(1);
   apagarLuz();

   T1CON = 0x0000;      // Preescalado: 1:1
   PR1   = 20000;       // PR1 = Tiempo * FCY * preescalado = 0,5 ms * 40 MHz * 1:1
   TMR1  = 0;
   IFS0bits.T1IF = 0;
   IEC0bits.T1IE = 1;
   IPC0bits.T1IP = 3;
   T1CON |= 1 << 15;    // Encender el timer
}
```

```
void tareaExtintor(void) {
   static unsigned char estado[] = {0, 0};
   unsigned char i;
   unsigned int temperatura;

   for (i = 0; i < NUM_ZONAS; i++) {
      temperatura = getMedidaADC(i);

        switch (estado[i]) {
            case 0:
                apagarRociador(i);

                if (temperatura > 716) {  // 50ºC
                    estado[i]   = 1;
                    ticks[i]    = 0;
                    segundos[i] = 0;
                }
                break;
            case 1:
                encenderRociador(i);

                if (temperatura < 409) {  // 35ºC
                    estado[i]   = 2;
                    ticks[i]    = 0;
                    segundos[i] = 0;
                }
                else if (segundos[i] >= 60) {
                    estado[0] = 3;
                    estado[1] = 3;
                    alarma_encendida = 1;
                }
                break;
            case 2:
                encenderRociador(i);

                if (segundos[i] >= 60) {
                    estado[i] = 0;
                }
                break;
            case 3:
                encenderRociador(i);
                encenderLuz();

                if (comprobarPulsador()) {
                    estado[0] = 0;
                    estado[1] = 0;
                    apagarLuz();
                    alarma_encendida = 0;
                }
                break;
        }
   }
}

void apagarRociador(unsigned char zona) {
   PORTB &= ~(1 << (PIN_ROCIADOR_A + zona));
}
```

```
void encenderRociador(unsigned char zona) {
   PORTB |= 1 << (PIN_ROCIADOR_A + zona);
}

void apagarLuz(void) {
   PORTB |= 1 << PIN_LUZ;
}

void encenderLuz(void) {
   PORTB &= ~(1 << PIN_LUZ);
}

unsigned char comprobarPulsador(void) {
    return !((PORTB >> PIN_PULSADOR) & 0x1);
}

void __attribute__((interrupt, no_auto_psv)) _T1Interrupt(void) {
    static unsigned int ticks_altavoz = 0, duracion_nota = 0, nota = 0;
    const unsigned int dc[] = {1, 5};
    unsigned char i;

    IFS0bits.T1IF = 0;  // Borrar la bandera

    for (i = 0; i < NUM_ZONAS; i++) {
        ticks[i]++;

        if (ticks[i] >= 2000) {
            ticks[i] = 0;
            segundos[i]++;
        }
    }

    if (alarma_encendida) {
        ticks_altavoz++;
        duracion_nota++;

        if (ticks_altavoz <= dc[nota])
            PORTB |= 1 << PIN_ALTAVOZ;
        else
            PORTB &= ~(1 << PIN_ALTAVOZ);

        if (ticks_altavoz >= 2 * dc[nota])
            ticks_altavoz = 0;

        if (duracion_nota >= 2000) {
            duracion_nota = 0;

            if (++nota >= 2)
                nota = 0;
        }
    }
    else {
        PORTB &= ~(1 << PIN_ALTAVOZ);
        ticks_altavoz = 0;
        duracion_nota = 0;
        nota = 0;
    }
}
```

main.c

```c
#include <p33FJ32MC202.h>
#include "config.h"
#include "adc.h"
#include "extractor.h"
#include "extintor.h"

/**
 * Sistema de extracción de humos y extinción de incendios.
 */
int main(void) {
    inicializarReloj();
    inicializarADC();
    inicializarExtractor();
    inicializarExtintor();

    while (1) {
        tareaExtractor();
        tareaExtintor();
    }

    return 0;
}
```