UNIVERSIDAD DE ALCALÁ ESCUELA POLITÉCNICA SUPERIOR DEPARTAMENTO DE ELECTRÓNICA		GRADO EN INGENIERÍA EN ELECTRÓNICA DE COMUNICACIONES	
ASIGNATURA	SISTEMAS ELECTRÓNICOS DIGITALES AVANZADOS	FECHA	MARZO 2016
APELLIDOS, NOMBRE	SOLUCIÓN	GRUPO	

PRUEBA DE EVALUACIÓN INTERMEDIA 1

Ejercicio 1

Se propone diseñar un sistema empotrado basado en microcontrolador LPC1768 (Fcpu=100MHz) capaz de realizar medidas de distancia en un plano y sobre un entorno de 180º, con posibilidad de ser controlado de forma manual mediante un potenciómetro o desde un ordenador mediante una interfaz serie asíncrona. El sistema está basado en el sensor de distancia por ultrasonidos SRF04 (ver Anexo 1) y en un servomotor. El diagrama de bloques del sistema se representa en la figura 1.

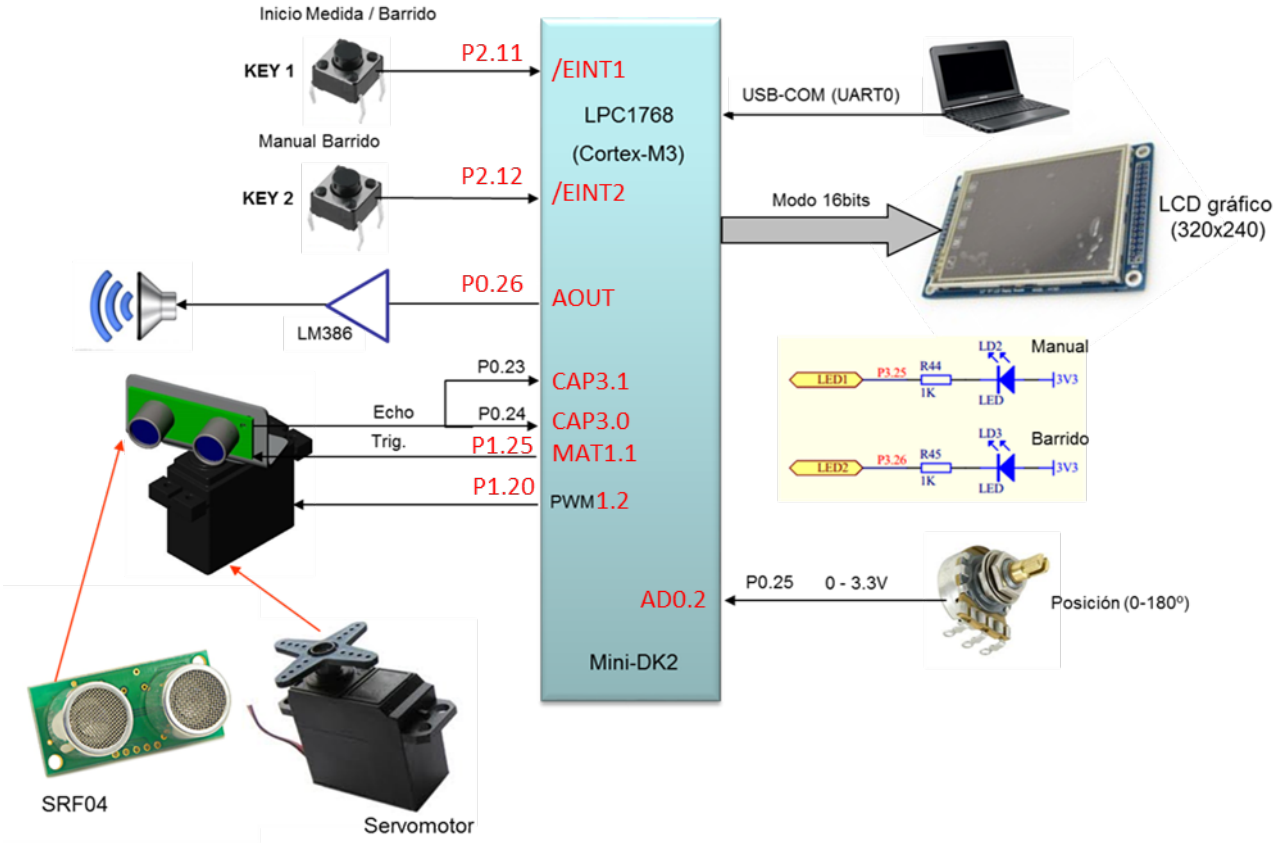


Figura 1. Diagrama de bloques del escáner ultrasónico

El sistema tiene dos modos de funcionamiento. Mediante KEY2 seleccionamos entre el modo **manual** y de **barrido**. Los LEDs 2 y 3 señalizan el modo configurado.

Estando en el **modo barrido**, al pulsar KEY1, el sonar comienza a realizar un barrido (0-180º) con una determinada resolución (en grados y configurable) deteniéndose en cada posición durante **medio segundo** para realizar la medida de distancia correspondiente. Al final del barrido se detendrá automáticamente esperando actuar de nuevo sobre el pulsador para iniciar de nuevo el barrido.

En el **modo manual**, el potenciómetro permite llevar al servo a una determinada posición (0-180º) de manera proporcional a su recorrido. Una vez situado en la posición deseada, mediante **KEY1** realizamos la medida de distancia al provocar el disparo del sensor SRF04 como consecuencia de aplicar un pulso a nivel alto en la entrada **TRIG**. Dicha medida se obtiene a partir de la duración del pulso en la salida **Echo**, cuyo valor es el tiempo que tardan los ultrasonidos en ir y volver tras rebotar en el obstáculo.
Tanto en el modo manual como de barrido, cada vez que se obtiene una medida, se ha de mostrar en centímetros sobre el **LCD** (en Anexo 2 está la librería con sus funciones).

El valor de la tensión analógica que proporciona el potenciómetro se ha de tomar **periódicamente cada 50ms**.

Mientras la distancia medida sea menor que la de un **umbral** (programable) se ha de activar una señal de alarma consistente en un tono de **1 kHz** (señal senoidal con **20 muestras/ciclo**).

Considere que el SysTick está habilitado y que interrumpe periódicamente cada 50ms y utilice su función de interrupción para hacer medidas de tiempos grandes.

a) Complete sobre el diagrama de la figura1, el nombre del pin (**Pn.x**) sobre la línea de conexión y el nombre del recurso utilizado dentro del bloque que representa la Mini-DK2 (ej. **MAT1.0**) excepto el LCD.

b) Complete la función de configuración del ADC e indique la frecuencia de muestreo del canal seleccionado considerando que la Fclk del ADC sea la **mínima posible**.

```
void init_ADC(void)
{
LPC_SC->PCONP|= (1<<12);              // Power ON
LPC_PINCON->PINSEL1|= (0x02<<18);     // P0.25 es AD0.2
LPC_PINCON->PINMODE1|=(0x02<<18);     // Deshabilita pullup/pulldown
LPC_ADC->ADCR= (    1<<2)|           // Canal 2
                (0xFF<<8)|           // CLKDIV=255
FclkADC=25MHz/256=97,656kHz
                  (1<<16)|           // Modo Burst
                  (1<<21);           // PDN=1
NVIC_DisableIRQ(ADC_IRQn);           // ADC no interrumpe!!!
}
```

$$F_{muestreo} = \frac{97{,}656\ \text{kHz}}{65} = 1502{,}4\ \text{Hz}$$

c) Complete la función de configuración de la señal PWM, y de actualización de la posición del servo. Considere que el periodo sea de **15ms**, y el tiempo a nivel alto varíe entre **0.6-2.4ms**, para un movimiento de su posición entre 0º y 180º.

```
void config_pwm(void)
{
LPC_PINCON->PINSEL3|=(2<<8);
LPC_SC->PCONP|=(1 <<6);  //Power PMW module
LPC_PWM1->MR0=F_pclk*15e-3 -1; // frecuencia PWM
LPC_PWM1->PCR|=(1<<10); //ENA2=1
LPC_PWM1->MCR|=(1<<1);  //Reset on Match
LPC_PWM1->TCR|=(1<<0)|(1<<3); //Start
}

void set_servo(uint8_t grados)
{
LPC_PWM1->MR2 = F_pclk*0.6e-3 + F_pclk*1.8e-3*grados/180;
LPC_PWM1->LER|= (1<<2)|(1<<0);
}
```

d) Escriba la función de interrupción que permite realizar la medida de la distancia en centímetros, a partir del pulso que entrega el sensor SRF04. Considere que la velocidad de los ultrasonidos es **342m/s** y que la variable global ***distancia***, se modifica con la distancia al obstáculo en centímetros. Explique la configuración del recurso utilizado (TimerN) sin escribir el código (puede utilizar pseudocódigo).
NOTA1: Considere que el timer cuenta con una resolución de **1 microsegundo**.

NOTA2: Tenga en cuenta que se ha de sacar por el **LCD** la medida obtenida (un * si la medida no es válida). Considere si fuese el caso la activación de la señal de alarma.

```
void TIMER3_IRQHandler(void)
{
   LPC_TIM3->IR|=(1<<5);                                  // borrar flag CR1
    pulso_duracion= LPC_TIM3->CR0-LPC_TIM3->CR1;     // en microseg.
   distancia= pulso_duracion*342e-6*100/2;           // en centímetros
   if(distancia > umbral)LPC_TIM1->TCR=0x02;          // stop timer
   else LPC_TIM1->TCR=0x01;                           // start timer

   if(pulso_duracion>24000)   // 24ms
   {
   medida_OK=0;          // medida erronea
    GUI_Text(10, 10, "Distancia=    *", White, Black); // 10 caracteres

   }
   else
   {
   medida_OK=1;
    sprintf(cadena,"Distancia= %3.2f \n\r",distancia); // 16 caracteres
   GUI_Text(10, 10, cadena, White, Black);
   }

   if(modo&&Start) set_servo(grados_barrido);  // garantizamos que el servo
                                               // alcanza su posición antes del
                                              // disparo en el modo barrido
}
```

Configuración Timer:

Timer 3 modo captura:

- CAP3.0 → flanco subida (NO interrumpe)
- CAP3.1 → flanco bajada (SI interrumpe)

Preescaler = 24 (Ftick = Fpclk/25 = 1Mhz)
Modo running (cuenta continuamente)
Hab. Interrupciones (sólo flanco de bajada)
Asig. Prioridad
Start Timer

e) Escriba las funciones de interrupción de los pulsadores KEY1 y KEY2 de la tarjeta Mini-DK2. Considere que existe la variable global ***modo***, que define el modo de funcionamiento (*modo*=0, manual; *modo*=1, barrido) así como la(s) variable(s) que considere necesarias. Active el LED correspondiente en función del modo seleccionado e inicie la medida o de la orden de inicio del barrido (**Start=1**) según corresponda.

```
void EINT2_IRQHandler(void) // Key 2
{
LPC_SC->EXTINT=(1<<1); //borrar flag
modo^=1;
if(modo)LPC_GPIO3->FIOPIN=(1<<25); //LED3 activo
else LPC_GPIO3->FIOPIN=(2<<25);     //LED2 activo
}
void EINT1_IRQHandler(void) // Key 1
{
LPC_SC->EXTINT=(1<<2); //borrar flag
if(modo==0)  disparo_SRF04();
else{
      Start=1;
      set_servo(0); // posición 0°
      }
}
```

f) Escriba la función de interrupción del Timer 1 que saca las muestras hacia el DAC para generar la señal de alarma, y el valor del registro **MRx** correspondiente para obtener la frecuencia deseada. Considere ya inicializado el *array* ***muestras[20]*** con los valores discretos de un ciclo. Indique la frecuencia de interrupción.

```
void TIMER1_IRQHandler(void)
{
LPC_TIM1->IR|=(1<<0);   // borrar flag
LPC_DAC->DACR= muestras[indice++]<<6;
if (indice==20)indice=0;
}
```

El periodo de interrupción queda determinado por tiempo hasta alcanzar el valor del registro Match.
Considerando la frecuencia de salida de 1kHz y 20 muestras/ciclo

MR0=F_pclk/1000/20 – 1 = 1249
F_int=F_out*N_muestras_ciclo= 20kHz

g) Explique qué recurso utilizaría y como lo configuraría para reducir la carga de CPU durante la generación de la señal de alarma.

Evitaríamos la ejecución de la interrupción del Timer3 empleando el DMA en modo Linked con el propio canal (ej. Canal 0), para que al finalizar la transferencia se inicie de nuevo sin que sea necesaria la interrupción del DMA.

LPC_GPDMACH0->DMACCSrcAddr = (uint32_t) &muestras[0]; // Fuente Memoria
LPC_GPDMACH0->DMACCDestAddr = (uint32_t) &(LPC_DAC->DACR); //Destino Periférico
Transfer Size=20; Tamaño transferencia= 16 bits; Incrementa Fuente; No incremento Destino
Sin interrupción (ojo, modo Linked)
LPC_DAC->DACCNTVAL = (F_pclk/20/1000) -1; // Frecuencia transf. DMA

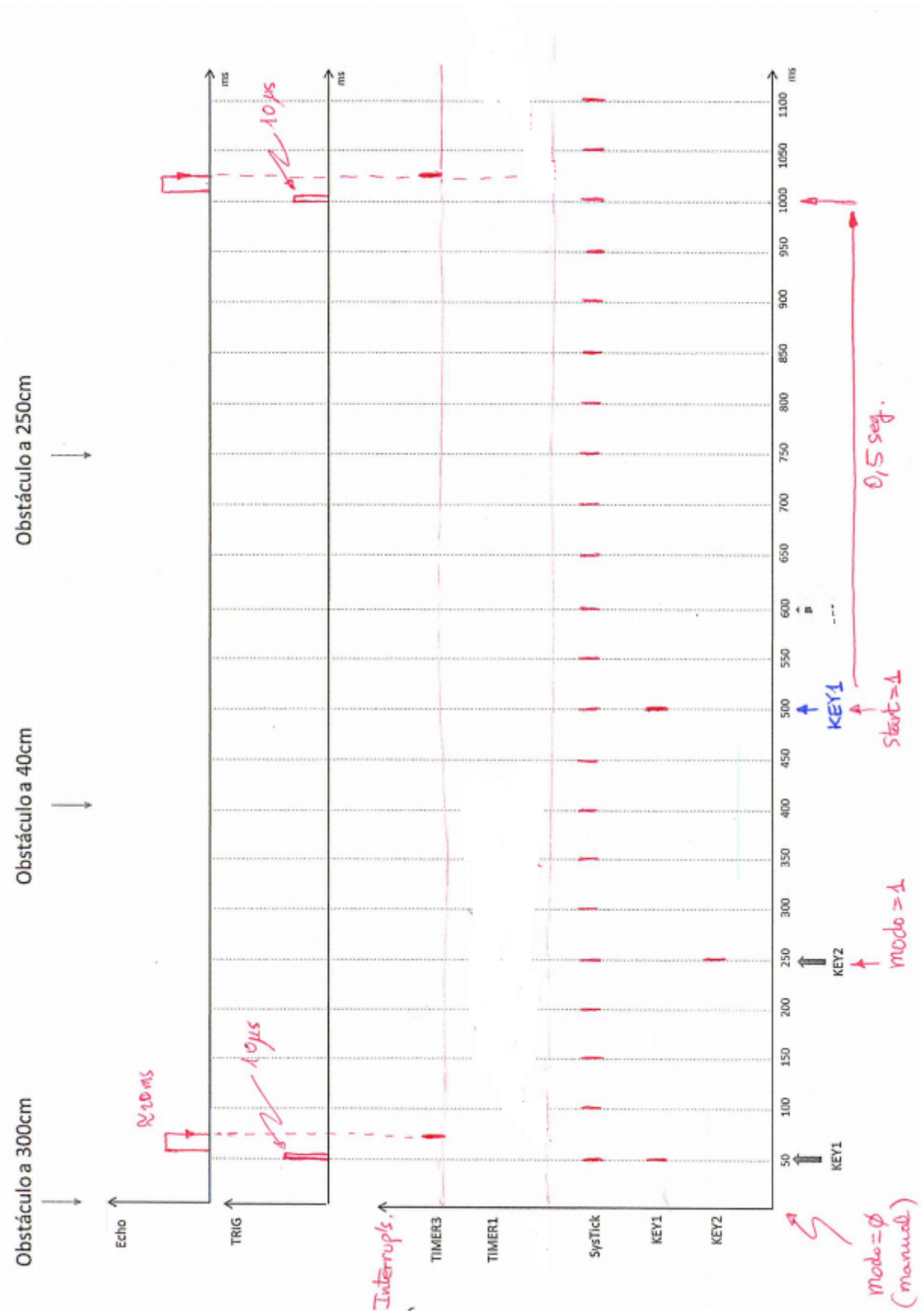
h) Escriba la parte del código encargado de realizar el barrido dentro de la función de interrupción del SysTick. Considere que existe la variable ***N_pasos*** que configura el número de pasos o medidas a realizar dentro de cada barrido (*N_pasos*=180º/resolucion), la variable ***Start*** que al ponerla a uno inicia el barrido, y cuantas variables auxiliares precise.
NOTA 1: Tenga en cuenta que el servo avanza su posición cada **0,5 segundos.**
NOTA 2: Considere ya escrita la función ***disparo_SRF04()*** que provoca el pulso de 10us de duración por la entrada TRIG para iniciar una medida.

```
   void SysTick_IRQHandler(void)
{
// se ejecuta cada 50mseg
   if (modo&&Start) {
   contador++;
                  if(contador==10){
                                 contador=0;
                                 disparo_SRF04();
                                 pasos++;
                                 grados_barrido+=resolucion;
                                 // ver NOTA
                                 if(pasos==N_pasos){
                                                Start=0;
                                                pasos=0;
                                                fin_barrido=1;
                                              }

   else if(modo==0){
   grados=((LPC_ADC_ADDR2>>8)&0xFF)*180/255; // valor del potenciómetro
   set_servo(grados);
   }
}
```

NOTA: La orden de mover el servo se da dentro de la interrup. del Timer 3 para garantizar que alcanza su posición antes de realizar la medida. Esto es una de las posibles alternativas.

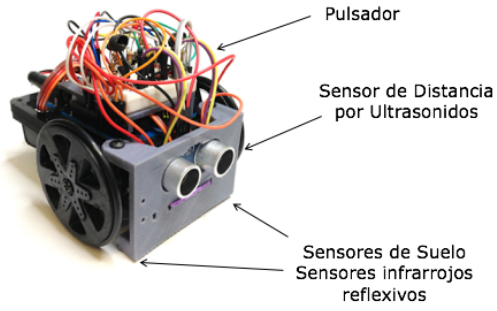
i) Complete el diagrama temporal de manera aproximada de las tareas asociadas a las distintas fuentes de interrupción y señales de control, en función de la aparición de los eventos de entrada señalados. Tenga en cuenta que después del reset el escáner está en **modo manual**. El umbral de detección se fija en 50cm.

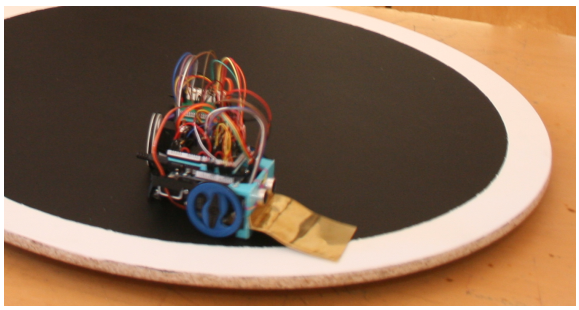


Ejercicio 2

Se desea desarrollar el software de un robot de sumo para que participe en una competición. El robot dispone de los siguientes elementos:

- Un **sensor frontal** de distancia que mide en centímetros la distancia a un objeto que tiene delante. Si no detecta nada en su rango de medida, entrega el valor **-1**. El resultado de la medida se encuentra en una variable global ***int distancia***.
- **Dos sensores** de infrarrojos reflexivos situados en los dos extremos de la parte delantera del robot apuntando al suelo. Detectan si debajo del robot hay blanco o negro guardando la información en los dos bits de menor peso de la variable global char suelo de forma que si toma el valor 0x00 detectan los dos sensores negro, 0x01 detecta blanco sólo el de la derecha, 0x02 detecta blanco sólo el de la izquierda y 0x03 detectan blanco los dos sensores.
- Una interrupción periódica que se ejecuta cada **10ms**.
- Existen cuatro funciones que controlan el movimiento del robot:
    - Avanza() – hace que el robot avance hacia delante.
    - Rerocede() – hace que el robot vaya hacia atrás
    - Derecha() – hace que el robot gire en el sitio hacia la derecha
    - Izquierda() – hace que el robot gire en el sitio hacia la izquierda
    - Detenido() – El robot permanece parado
- El robot dispone de un **pulsado**r que refleja su valor en la variable ***char pulsador*** que vale cero cuando no está pulsado y uno cuando está pulsado.





Se desea **realizar el satechart que modele el comportamiento del robot** de la siguiente manera:

- El robot debe permanecer parado en un principio .
- En el momento de iniciar la competición, el propietario del robot **presiona el pulsador** y se separa del área de juego.
- El robot debe permanecer **quieto durante 5 segundos**. Pasado este tiempo comenzará a moverse.
- El robot tendrá los siguientes comportamientos relacionados con el movimiento:
    - **Navegación** – El robot avanza hasta que detecta blanco en alguno de los dos sensores de infrarrojos, momento en que realiza una secuencia de "escape" retrocediendo durante 2 segundos, girando hacia el lado que desee el programador durante 2 segundos y pasando al comportamiento de "búsqueda".
    - **Ataque** – En este modo el robot avanza aunque detecte blanco en los sensores.
    - **Búsqueda** – El robot gira en el sitio hacia derecha o izquierda según prefiera el programador. Si en el giro el robot detecta blanco, iniciará una secuencia de "escape".
- **El robot estará en "Búsqueda" durante un máximo de 5 segundos, momento en que entrará en modo "Navegación" durante 10 segundos y de nuevo a "Búsqueda".**
- Si en cualquier momento el robot detecta un objeto delante suyo a **menos de 30cm** iniciará una secuencia se "ataque" pasando a modo "navegación" en caso de dejar de detectar al oponente.
- **Pasados 30 segundos, desde el inicio de la prueba, el robot debe detenerse**.

Se pide:

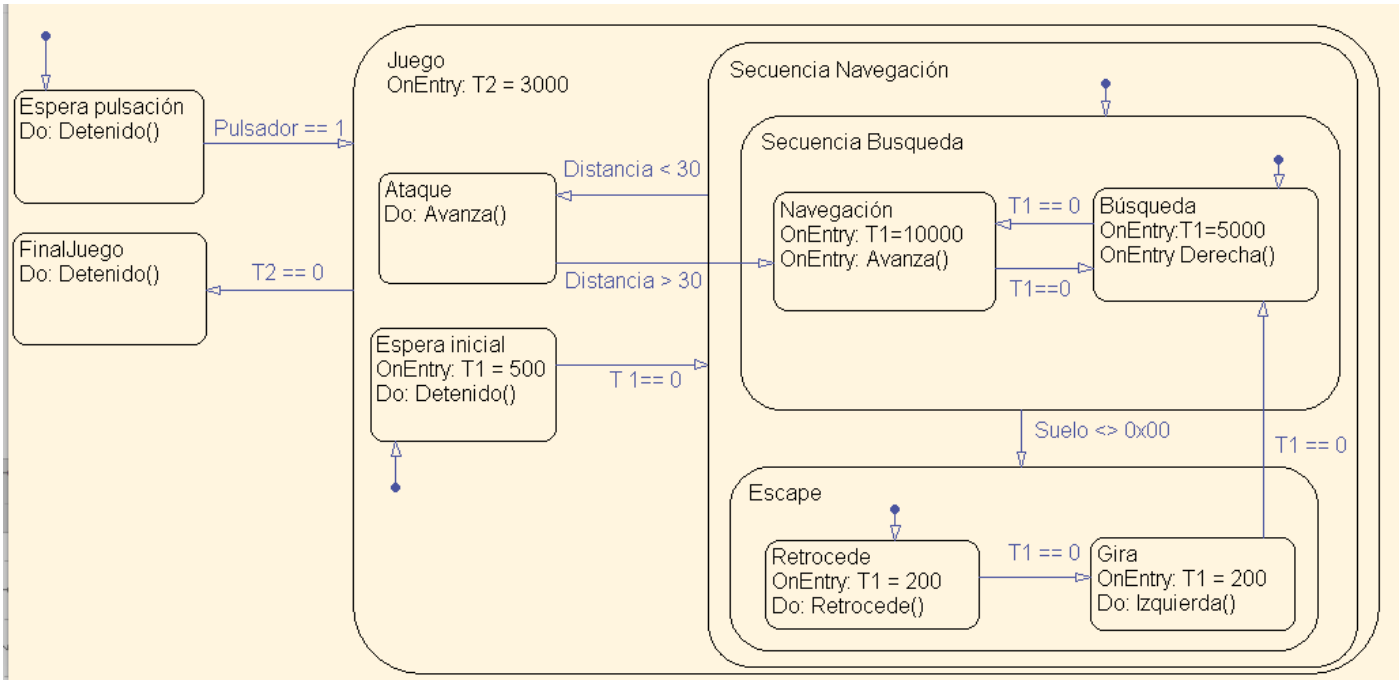
a) Realice el StateChart que modele el comportamiento del robot.
b) Realice el código que iría en el programa principal implementado y en la interrupción periódica que implemente el nivel superior del StateChart.

Existen múltiples posibles soluciones. A continuación se muestra una de ellas.

La interrupción periódica se utiliza para generar las temporizaciones no bloqueantes. En la interrupción periódica que se ejecuta cada 10ms debería de llamarse a la siguiente función:

```
void int_10ms(void) {
   if (T1 > 0 ) T1--;
   if (T2 > 0) T2 --;
}
```

Cuando el programa principal hace T2 = 3000, tras 3000 ejecuciones de la interrupción T2 valdrá cero, es decir, tras 3000 * 10ms = 30 seg. Las variables T1 y T2 serían globales.



Las variables EstadoNivel_1, EstadoNivel_1, EstadoNivel2_juego y Entry serían variables globales.

```
void Ejecuta_StateChart_Nivel1(void) {
   if (EstadoNivel_1 == DEFAULT) EstadoNivel_1 = N1_ESPERA_PULSACION;

   switch (EstadoNivel_1) {
      case N1_ESPERA_PULSACION:
         Detenido();
         if (Pulsador == 1) {
           EstadoNivel_1 = N1_JUEGO;
           EstadoNivel_2_juego = DEFAULT;
           Entry = 1;
         }
         break;
      case N1_FINAL_JUEGO:
         Detenido();
         break;
      case N1_JUEGO:
         if Entry {
             T2 = 3000;
             Entry = 0;
         }
         if (T2 == 0) {
            EstadoNivel_1 = N1_JUEGO;
            EstadoNivel_2_juego = DEFAULT;
            Entry = 1;
         }
         Ejecuta_StateChart_Nivel2_Juego();
         break;
   }
}
```

ANEXO 1. Descripción del sensor SRF04

El módulo SRF04 (Figura 2) es un sensor de distancias por ultrasonidos capaz de detectar objetos y calcular la distancia a la que se encuentra en un rango de 3 a 300 cm. El sensor funciona por ultrasonidos y contiene toda la electrónica encargada de hacer la medición. Su uso es tan sencillo como enviar el pulso de disparo para iniciar una medida y medir la anchura del pulso recibido a su salida.

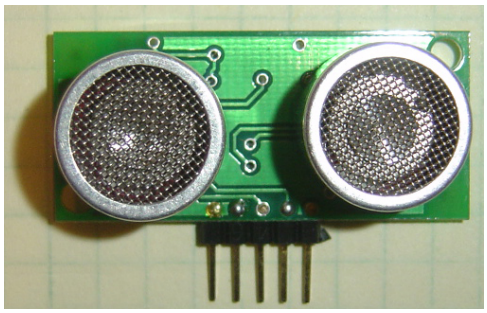


Figura 2. Aspecto del sensor SRF04

Funcionamiento

El sensor SRF04 funciona emitiendo impulsos de ultrasonidos inaudibles para el oído humano. Los impulsos emitidos viajan a la velocidad del sonido hasta alcanzar un objeto, entonces el sonido es reflejado y captado de nuevo por el receptor de ultrasonidos. Lo que hace el controlador incorporado al recibir una señal de disparo, es emitir una ráfaga de impulsos (8 ciclos de 40kHz) y a continuación empieza a contar el tiempo que tarda en llegar el eco. Este tiempo se traduce en un pulso de eco de anchura proporcional a la distancia a la que se encuentra el objeto (Figura 3).

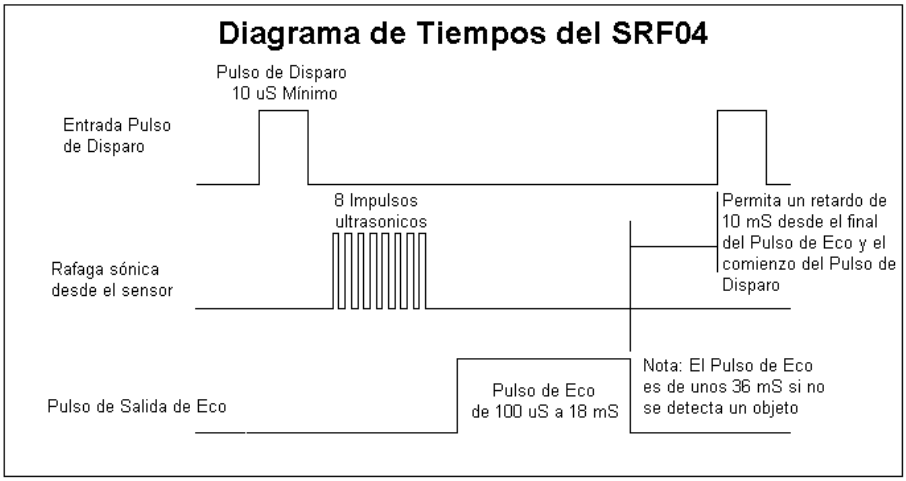


Figura 3. Diagrama de tiempos del sensor ultrasónico SRF04

Desde un punto de vista práctico, lo que hay que hacer es mandar una señal de disparo en el pin 3 del SRF04 y después leer la anchura del impulso que nos proporciona en el pin 2. El pulso de disparo tiene que tener una anchura mínima de **10 µS**. Después leemos el pulso de salida de Eco y medimos su longitud que es proporcional al eco recibido. En caso de que no se produzca ningún eco, porque no se encuentra un objeto, el pulso de eco tiene una longitud aproximada de **36 ms**. Hay que dejar un retardo de 10 ms desde que se hace una lectura hasta que se realiza la siguiente, con el fin de que el circuito se estabilice. En la Figura 4 se muestran las características del sensor y los pines de conexión.

Tensión	5V
Consumo	30 mA Tip. 50mA Max.
Frecuencia:	40 Khz.
Distancia Mínima:	3 cm.
Distancia Máxima:	300 cm.
Sensibilidad:	Detecta un palo de escoba a 3 m.
Pulso de Disparo	10 uS min. TTL
Pulso de Eco:	100 uS - 18 mS
Retardo entre pulsos:	10 mS Mínimo
Pulso de Eco:	100 uS - 18 mS
Tamaño:	43 x 20 x 17 mm
Peso:	10 gr.

Figura 4. Características y conexionado del SRF04

ANEXO 2. Funciones del LCD

```
/* LCD color */
#define White          0xFFFF
#define Black          0x0000
#define Grey           0xF7DE
#define Blue           0x001F
#define Blue2          0x051F
#define Red            0xF800
#define Magenta        0xF81F
#define Green          0x07E0
#define Cyan           0x7FFF
#define Yellow         0xFFE0

/*********************************************************************************
* Function Name  : RGB565CONVERT
* Description    : 24Î»×ª»»16Î»
* Input          : - red: R
*                  - green: G
*          - blue: B
* Output         : None
* Return         : RGB ÑÕÉ«Öµ
* Attention      : None
*********************************************************************************/
#define RGB565CONVERT(red, green, blue)\
(uint16_t)( (( red   >> 3 ) << 11 ) | \
(( green >> 2 ) << 5  ) | \
( blue  >> 3 ))

/* Private function prototypes -----------------------------------------------*/
void LCD_Initializtion(void);
void LCD_Clear(uint16_t Color);
uint16_t LCD_GetPoint(uint16_t Xpos,uint16_t Ypos);
void LCD_SetPoint(uint16_t Xpos,uint16_t Ypos,uint16_t point);
void LCD_DrawLine( uint16_t x0, uint16_t y0, uint16_t x1, uint16_t y1 , uint16_t color );
void PutChar( uint16_t Xpos, uint16_t Ypos, uint8_t ASCI, uint16_t charColor, uint16_t bkColor );
void GUI_Text(uint16_t Xpos, uint16_t Ypos, uint8_t *str,uint16_t Color, uint16_t bkColor);
#endif
```