UNIVERSIDAD DE ALCALÁ ESCUELA POLITÉCNICA SUPERIOR DEPARTAMENTO DE ELECTRÓNICA		GRADO EN INGENIERÍA EN ELECTRÓNICA DE COMUNICACIONES	
ASIGNATURA	**SISTEMAS ELECTRÓNICOS DIGITALES AVANZADOS**	**FECHA**	**3 –MAYO-2017**
APELLIDOS, NOMBRE		**GRUPO**	

PRUEBA DE EVALUACIÓN INTERMEDIA 2

CUESTION 1

Suponiendo un sistema con un LPC1768 con las tareas que se indican a continuación, identifique los parámetros **D** y **T** de cada una de ellas y **justifique el valor explicando el subsistema interno del microcontrolador que utilizaría** y en qué modo se configuraría. Tenga en cuenta que es un único microcontrolador con recursos limitados.

a) Recepción de datos a 38400 baudios por la UART 3
b) Generación de una señal cuadrada con un ciclo de trabajo de un 50% de frecuencia variable entre 1kHz y 100kHz.
c) Generación de una señal PWM de 10kHz de frecuencia con un ciclo de trabajo que oscila entre un 10% y un 90%.
d) Reproducción repetida de un mensaje de audio de 10 segundos de duración con una frecuencia de muestro de 8kHz y muestras de 10 bits utilizando DMA.
e) Grabación simultánea de dos canales de audio a una frecuencia de muestreo de 8kHz durante un segundo cuando lo solicite el usuario.

CUESTIÓN 2

Suponiendo que un sistema basado en el LPC1768 tiene tres tareas con los parámetros que se indican en la tabla y el programa principal tiene una región crítica de 1ms

Tarea	Prioridad	Subprioridad	C	T	D
Tarea 1	0	0	1	5	5
Tarea 2	0	1	2	10	5
Tarea 3	1	0	5	30	30
Tarea 4	1	1	10	30	30

Nota: Unidades en ms

Se pide:

a) Analice si el sistema es ejecutable.

b) Si el sistema es ejecutable:

- Calcule la duración máxima de una región crítica del programa principal que haría que el sistema dejara de ser ejecutable.

Si el sistema no es ejecutable:

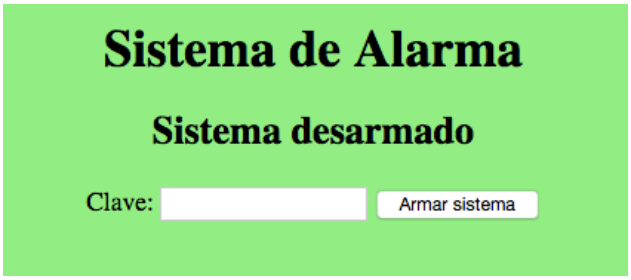
- Identifique los factores que hacen que el sistema no sea ejecutable y proponga una o varias soluciones que hagan el sistema ejecutable.

CUESTIÓN 3.

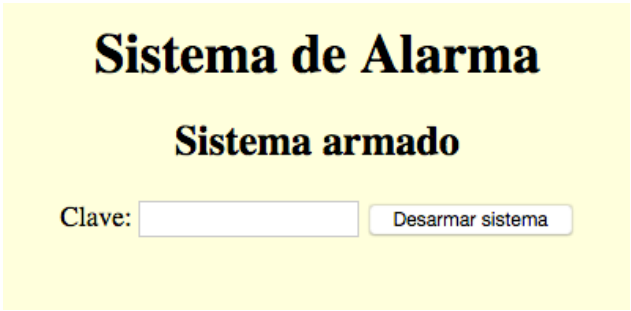
En una aplicación de monitorización remota de un sistema de alarma, se desea visualizar remotamente el estado del sistema y posibilitar su activación o desactivación al acceder a la página **alarma.cgi**. El estado del sistema se almacena en la variable global **uint8_t estado** que podrá tomar los valores 0 (sistema desarmado), 1 (sistema armado) y 2 (alarma activa). Además de indicar textualmente el estado del sistema, el fondo de la pantalla dependerá del estado: verde (sistema desarmado), amarillo (sistema armado), y rojo (alarma activada). Por seguridad, para poder armar o desarmar el sistema, deberá introducirse una clave que deberá coincidir con la cadena de texto almacenada en el array **char password[10]**. Si este código es correcto se cambiará el estado accediendo a la variable **estado**. En la tabla siguiente se muestra el código HTML y la salida visualizada en el navegador dependiendo del estado.

Indique el contenifo del fichero **temperatura.cgi** a la que accedería el navegador y el contenido de las funciones ***cgi_func(…)***, ***cgi_process_data(...)***, y/o ***cgi_process_var(...)*** necesarias.

```
<!DOCTYPE html>
<html>
<head>
  <meta http-equiv='refresh' content='15'>
</head>
<body style='background-color:lightgreen;'>
 <center>
   <H1> Sistema de Alarma</H1>
   <H2> Sistema desarmado</H2>
 </center>
 <form action='/alarma.cgi' method='get'>
   <center> Clave: <input type='text' name='clave1' value=''>
   <input type='submit' value='Armar sistema'>
   </center>
 </form>
</body>
</html>
```




```
<!DOCTYPE html>
<html>
<head>
  <meta http-equiv='refresh' content='15'>
</head>
<body style='background-color:lightyellow;'>
 <center>
   <H1> Sistema de Alarma</H1>
   <H2> Sistema armado</H2>
 </center>
 <form action='/alarma.cgi' method='get'>
   <center> Clave: <input type='text' name='clave2' value=''>
   <input type='submit' value='Desarma sistema'>
   </center>
 </form>
</body>
</html>
```




```
<!DOCTYPE html>
<html>
<head>
  <meta http-equiv='refresh' content='15'>
</head>
<body style='background-color:red;'>
 <center>
   <H1> Sistema de Alarma</H1>
   <H2> Alarma activa</H2>
 </center>
 <form action='/alarma.cgi' method='get'>
   <center> Clave: <input type='text' name='clave3' value=''>
   <input type='submit' value='Desarmar sistema'>
   </center>
 </form>
</body>
</html>
```

CUESTIÓN 4.
Explique qué es un DSP y sus características fundamentales.

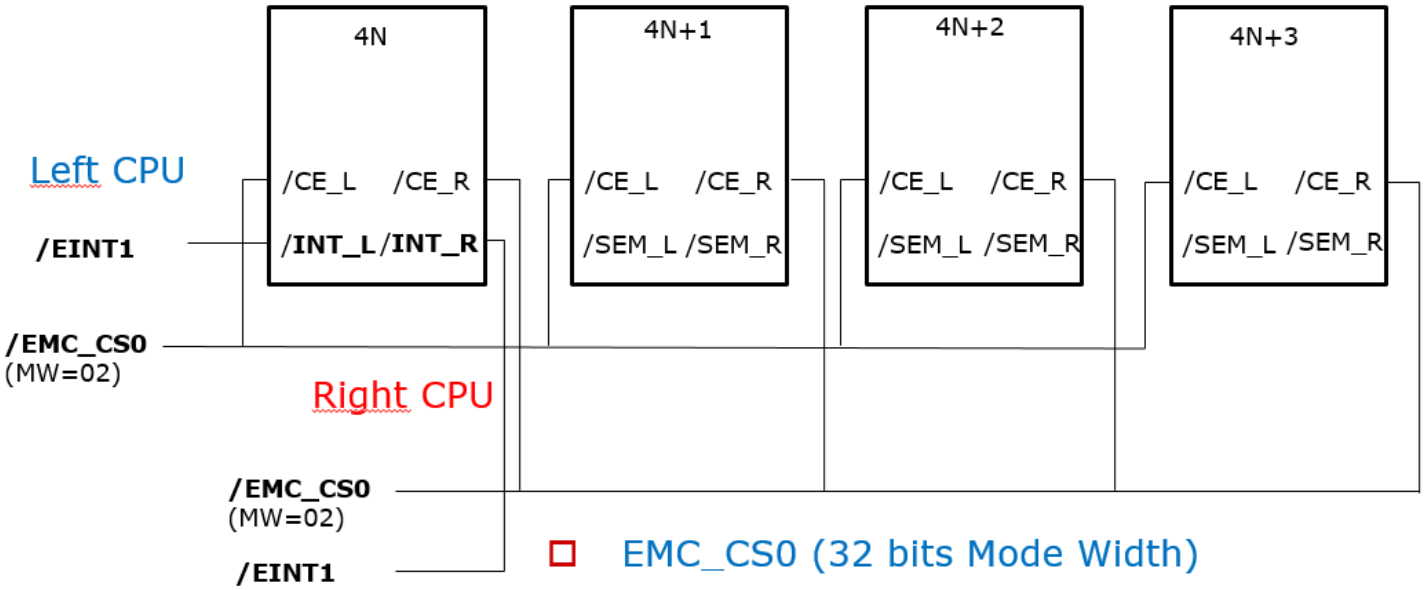
CUESTIÓN 5.
Explique qué se entiene por System on Chip.

CUESTIÓN 6.

Explique qué función tiene el "Scheduler" y el "Dispatcher" en un Sistema Operativo en Tiempo Real.

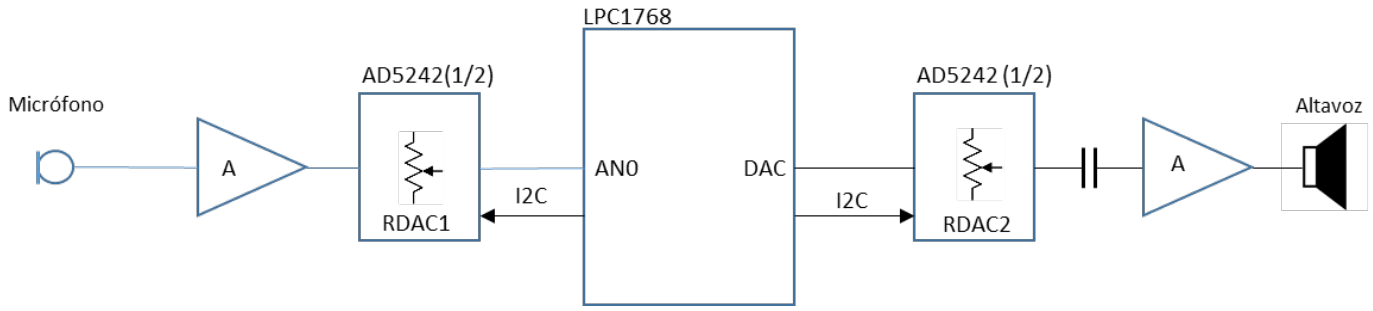
CUESTIÓN 7.

Explique en detalle el método de arbitración por interrupción de las memorias DUAL-PORT y justifícalo sobre el esquema de la figura. Considere que el bloque de memoria está mapeado en la dirección 0x8000.0000 y que cada chip tiene una capacidad de 16Kbytes.

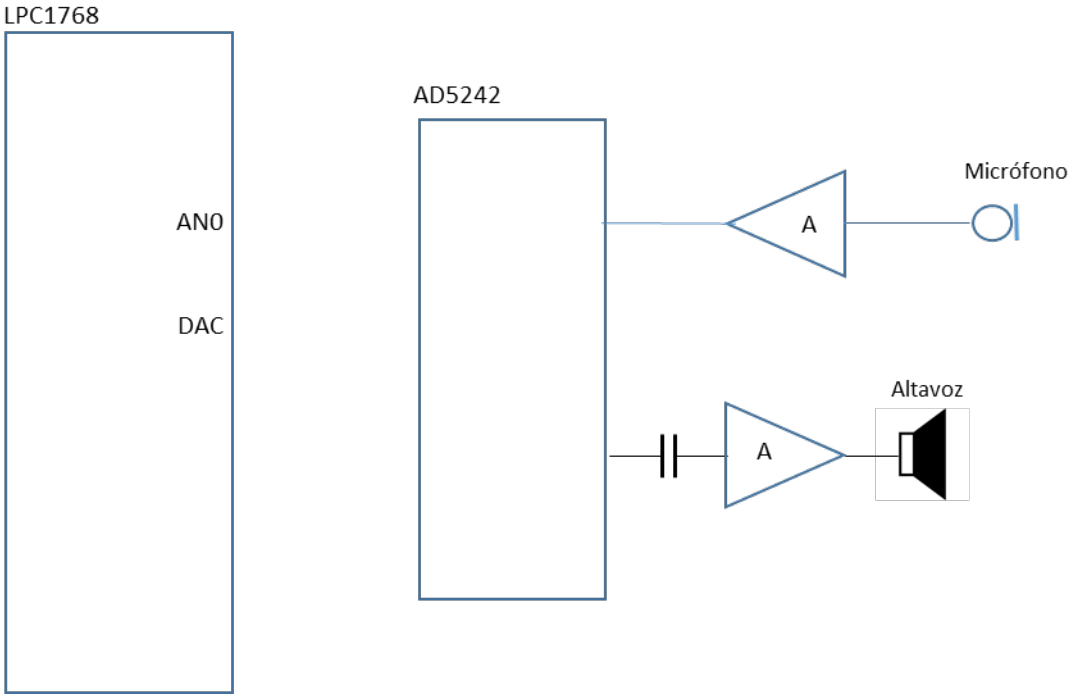


CUESTIÓN 8.

El diagrama de bloques del circuito de la figura representa un sistema de grabación/reproducción de audio en el que se utilizan dos potenciómetros digitales para controlar la ganancia del micrófono y el volumen del altavoz.



Dibuja detalladamente el diagrama de conexión del AD5242 con el LPC1768 teniendo en cuenta la librería de funciones del bus I2C del Anexo 1 y escriba la función que permite variar el volumen de salida entre el 0 y 100% *i2C_volumen(char porcentaje)*



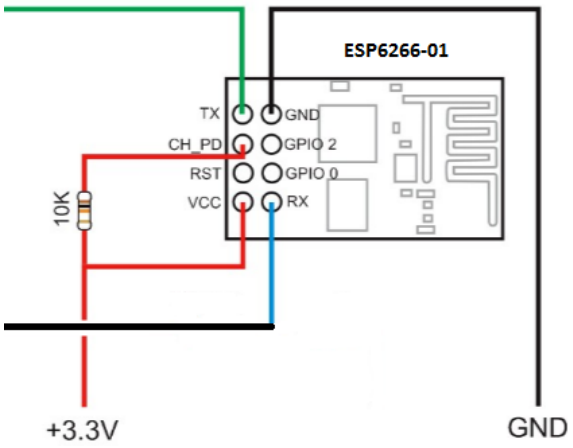
CUESTIÓN 9.

Se desea controlar un sistema empotrado basado en el LPC1768 mediante una conexión WIFI, para lo cual se conecta con la UART3 el módulo ESP8266.

- Complete el diagrama de conexión del LPC1768 con el módulo WIFI.
- Complete la función de configuración del puerto serie y **realice los cálculos** para configurar la velocidad a 115200 baudios, 8 bits por dato y sin paridad.
- Calcule la velocidad real obtenida y el tiempo que tarda en transmitirse el comando enviado al ejecutarse la sentencia, *tx_cadena_UART3("Ejecutando...\n\r")*. Ver Anexo 2.

```
void uart3_init(void)
{
      LPC_PINCON->PINSEL  |=        ;
      LPC_PINCON->PINSEL  |=        ;
      LPC_UART3->LCR=      ;
      LPC_UART3->DLL=      ;
      LPC_UART3->DLM=      ;
      LPC_UART3->LCR=       ;
}
```

.



Anexo 1. AD5242 y librería de funciones del bus I2C

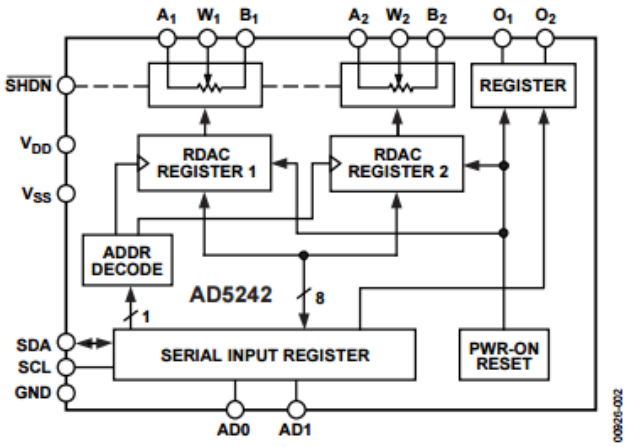


Figure 2. AD5242 Functional Block Diagram

Table 2.

S	0	1	0	1	1	AD1	AD0	R/$\overline{W}$	A	$\overline{A}$/B	RS	SD	O_1	O_2	X	X	X	A	D7	D6	D5	D4	D3	D2	D1	D0	A	P
	Slave Address Byte									Instruction Byte									Data Byte									

where:
S = start condition
P = stop condition
A = acknowledge
X = don't care
AD1, AD0 = Package pin programmable address bits. Must be matched with the logic states at Pin AD1 and Pin AD0.
R/$\overline{W}$ = Read enable at high and output to SDA. Write enable at low.
$\overline{A}$/B = RDAC subaddress select; 0 for RDAC1 and 1 for RDAC2.
RS = Midscale reset, active high.
SD = Shutdown in active high. Same as $\overline{SHDN}$ except inverse logic.
O_1, O_2 = Output logic pin latched values
D7, D6, D5, D4, D3, D2, D1, D0 = data bits.

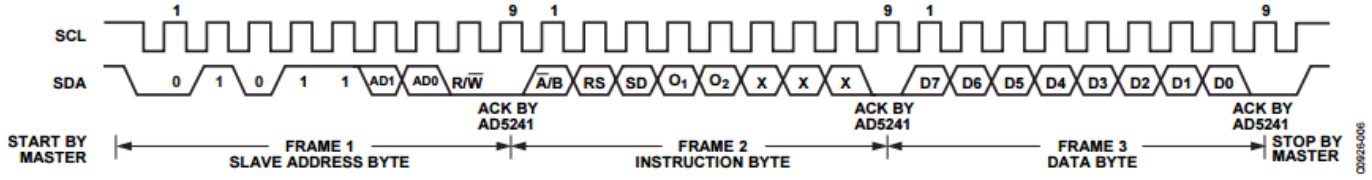


Figure 4. Writing to the RDAC Serial Register

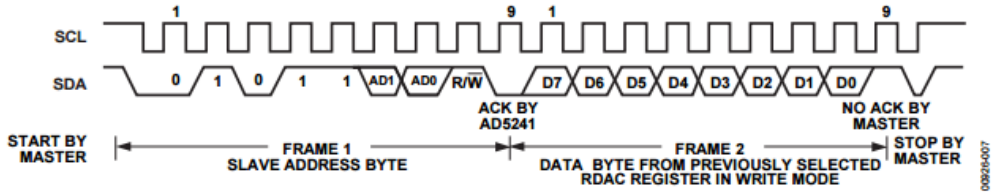


Figure 5. Reading Data from a Previously Selected RDAC Register in Write Mode

FUNCIONES DEL BUS i2c:

```
#include <LPC17xx.h>
#define SDA 0
#define SCL 1

void I2Cdelay(void)//retardo minimo de 4.7 us
{
 unsigned char i;
 for(i=0;i<100;i++); //Modificar límite para garantizar los tiempos (Bus standar -->F_max=100kHz)
}

//Genera un pulso de reloj (1 ciclo)
void pulso_SCL(void)
{
LPC_GPIO0->FIOSET=(1<<SCL);   // Genera pulso de reloj (nivel alto)
I2Cdelay();
LPC_GPIO0->FIOCLR=(1<<SCL);   // Nivel bajo
I2Cdelay();
}


void I2CSendByte(unsigned char byte)
{
 unsigned char i;
 for(i=0;i<8;i++){

        if (byte &0x80)  LPC_GPIO0->FIOSET=(1<<SDA); // envia cada bit, comenzando por el MSB
        else  LPC_GPIO0->FIOCLR=(1<<SDA);
        byte = byte <<1;             // siguiente bit
        pulso_SCL();
       }

//Leer ACK que envía el Slave (el Master ha de enviar un pulso de reloj)
// CONFIGURAR PIN SDA COMO ENTRADA;        //espera ACK(config. pin como entrada)
 LPC_GPIO0->FIODIR&=~(1<<SDA);
 pulso_SCL();

// CONFIGURA PIN SDA COMO SALIDA;
 LPC_GPIO0->FIODIR|=(1<<SDA);               // Dejamos SDA de nuevo como salida
}


//Función que envía START + Byte de dirección del Slave (con bit LSB inicando R/W)
void I2CSendAddr(unsigned char addr, unsigned char rw)
{
 //CONFIGURAR PINs SDA, SCL COMO SALIDAS;  // Por si se nos olvidada en la conf. general.
 LPC_GPIO0->FIODIR|=(1<<SDA)|(1<<SCL);

 LPC_GPIO0->FIOSET|=(1<<SDA)|(1<<SCL);    // SDA y SCL a nivel alto para garantizar el
                                          // nivel de reposo del bus + tiempo.
 I2Cdelay();
 SDA=0;                                 //condicion de START: Bajar SDA y luego SCL
 I2Cdelay();
 SCL=0;
 I2Cdelay();

 I2CSendByte((addr=addr<<1) + rw);   //envia byte de direccion
                                     //addr, direccion (7bits)
                                     //rw=1, lectura
                                     //rw=0, escritura
}


// Función para leer un Byte del Slave. El Master envía al final de la lectura
// el bit ACK o NACK (si es último byte leído) que se pasa como argumento de la función.
unsigned char I2CGetByte(unsigned char ACK)
{
// ACK = 0, para cualquier byte que no sea el ultimo.
// ACK = 1 (NACK), despues de leer el ultimo byte
 unsigned char i, byte;
 //CONFIGURAR PIN SDA COMO ENTRADA;  //configura pin SDA como entrada
 LPC_GPIO0->FIODIR&=~(1<<SDA);
 for(i=0;i<8;i++){                                  //lee un bit comenzando por el MSB
```

```
                LPC_GPIO0->FIOSET=(1<<SCL);//mientras SCL=1
                I2Cdelay();
                byte=byte<<1;
                if(LPC_GPIO0->FIOPIN&(1<<SDA)) byte++;//Si leemos "1" sumamos para introducir el "1"
                LPC_GPIO0->FIOCLR=(1<<SCL);         //Si leemos "0" solo desplazamos (se introduce un "0")
                I2Cdelay();
               }

 //CONFIGURAR PIN SDA COMO SALIDA;            // Master envía un ACK por cada byte leído.
 LPC_GPIO0->FIODIR|=(1<<SDA);

 if(ACK)LPC_GPIO0->FIOSET=(1<<SDA);          // ACK o (NACK) es funcion del último byte leído
       else LPC_GPIO0->FIOCLR=(1<<SDA);

 pulso_SCL();                                // Pulso de reloj para su envío
 return (byte);
}



void I2CSendStop(void)
{
 LPC_GPIO0->FIOCLR=(1<<SDA);
 I2Cdelay();
 LPC_GPIO0->FIOSET=(1<<SCL);                 // Subir SCL, y después SDA!! para dejar el bus en reposo
 I2Cdelay();
 LPC_GPIO0->FIOSET=(1<<SDA);
 I2Cdelay();
}
```

Anexo 2. Funciones del Puerto Serie (UART0)

```
void UART0_IRQHandler(void) {

   switch(LPC_UART0->IIR&0x0E) {

   case 0x04:                  /* RBR, Receiver Buffer Ready */
     *ptr_rx=LPC_UART0->RBR;   /* lee el dato recibido y lo almacena */
     if(*ptr_rx++ ==13){       /* Caracter return --> Cadena completa */
       *ptr_rx=0;              /* Añadimos el caracter null para tratar los datos recibidos como una cadena*/
       rx_completa = 1;        /* rx completa */
       ptr_rx=buffer;          /* puntero al inicio del buffer para nueva recepción */
     }
     break;

   case 0x02:                      /* THRE, Transmit Holding Register empty */
     if(*ptr_tx!=0)
       LPC_UART0->THR = *ptr_tx++; /* carga un nuevo dato para ser transmitido */
     else
       tx_completa=1;
     break;
   }
}
```

```
void tx_cadena_UART0(char *cadena)
{
   ptr_tx=cadena;
   tx_completa=0;
   LPC_UART0->THR = *ptr_tx;   // IMPORTANTE: Introducir un carácter al comienzo para iniciar TX o
}                              // activar flag interrupción por registro transmisor vacio
```