Facultad de Informática
Universidad Complutense de Madrid

EXAMEN DE PROGRAMACIÓN DE SISTEMAS Y DISPOSITIVOS

CURSO 2024-25, PRIMERA CONVOCATORIA, 20 DE DICIEMBRE DE 2024

En la realización de los ejercicios se considerarán implementadas y, por tanto, podrán usarse todas las funciones y macros públicas incluidas en el BSP desarrollado en los laboratorios. No podrá usarse ninguna función perteneciente a bibliotecas estándar de C. Si fuera necesaria alguna función auxiliar adicional, deberá ser codificada explícitamente. Se usará únicamente aritmética entera.

1. **(0.25 puntos)** En el fichero **`s3cev40.h`** del BSP se define la macro SEGS como nemotécnico de la dirección de memoria **`0x02140000`** en la que se encuentra mapeado el *display* 7 segmentos. Sin embargo, esta dirección no es la única que puede usarse. Indique si cada una de las macros siguientes sería o no válida como a alternativa a la definida en **`s3cev40.h`**.
   a) `#define SEGS (*(volatile uint8 *)0x03f40000)`
   b) `#define SEGS (*(volatile uint8 *)0x02a40000)`
   c) `#define SEGS (*(volatile uint8 *)0x02180000)`
   d) `#define SEGS (*(volatile uint8 *)0x0395a56b)`

2. **(0.25 puntos)** Se desea añadir un nuevo pulsador a la placa de prototipado similar a los ya existentes, es decir, con una configuración tal que en reposo genera un 1 y un 0 mientras esté pulsado. Indique a qué pin y de qué puerto lo conectaría para que su presión pueda disparar interrupciones en el sistema, así como los cambios necesarios a realizar en la función **`port_init()`** para que esto sea posible.

3. **(0.25 puntos)** Indique los cambios necesarios a realizar en la función **`uart0_init()`**, para que la velocidad transmisión del interfaz sea 9600 baudios y los datos se transmitan con paridad par (*even parity*). Asuma que la frecuencia del reloj del sistema es 64 MHz.

4. **(0.25 puntos)** Para evitar que el *timer* 0 dispare interrupciones por fin de cuenta existen 2 alternativas: deshabilitar en el controlador de interrupciones las interrupciones de este dispositivo o parar la cuenta. Indique en cada caso la sentencia/sentencias C necesarias para hacerlo.

5. **(0.25 puntos)** Indique el retardo en ms que hace la llamada a esta función. Asuma que la frecuencia del reloj del sistema es 64 MHz.

```
void timer3_unknown_delay( void )
{
  TCFG0  = (TCFG0 & ~(0xff << 8)) | (127 << 8);
  TCFG1  = (TCFG1 & ~(0xf << 12)) | (3 << 12);
  TCNTB3 = 9375;
  TCON   = (TCON & ~(0xf << 16)) | (1 << 17);
  TCON   = (TCON & ~(0xf << 16)) | (1 << 16);
  while( !TCNTO3 );
  while( TCNTO3 );

}
```

6. **(0.25 puntos)** Indique el número real que representa de la siguiente constante suponiendo codificada en punto fijo Q24.8.

```
const int32 foo = 43688;
```

7. **(0.25 puntos)** Se desea que una tarea periódica cuyo prototipo es **`void Task( void )`** se ejecute cada décima de segundo bajo un kernel de planificación no expropiativo como el utilizado en los

laboratorios. El *tick* de reloj deberá generarse a una frecuencia de 1000 Hz. Complete los puntos suspensivos del siguiente código para que esto sea posible.

```
create_task( ..., ... );
timer0_open_ms( scheduler, ..., ... );
```

8. (0.25 puntos) En una aplicación bajo µC/OS-II existe una variable global, **`counterGlobal`**, compartida entre 2 tareas: **`Task1`** y **`Task2`**. Para garantizar la corrección de la aplicación es necesario proteger el acceso a esta variable mediante un semáforo mutex. Se pide:

a) Indicar la sentencia/sentencias C necesarias para declarar el semáforo.

b) Indicar la sentencia/sentencias C necesarias para crear el semáforo.

c) Completar el código de las 2 tareas siguientes para que el acceso a la variable **`counterGlobal`** sea mutuamente exclusivo.

```
void Task1( void *id )
{
  counterGlobal++;
  OSTimeDly( 5 );
}

void Task2( void *id )
{
  counterGlobal++;
  OSTimeDly( 5 );
}
```

9. (2 puntos) Para regular digitalmente la intensidad luminosa, así como reducir el consumo de un *display led* se usa la técnica denominada *dimming* que consiste en tener tenerlo apagado durante una fracción del tiempo. Para evitar que el usuario distinga el parpadeo, debe encenderse y apagarse a una frecuencia igual o superior a 50 Hz (frecuencia de refresco) y mantenerse encendido durante un periodo (tiempo de persistencia) proporcional a la intensidad deseada.

Se pide codificar en C una función que visualice un número por el *display* 7-segmentos con el brillo indicado (de 0 a 100). Si el brillo es distinto de 0, esta función configurará el *timer* 0 para que interrumpa 5000 veces por segundo (definiendo así 5000 *slots* de tiempo en cada segundo) e instalará como rutina de tratamiento otra función (que también deberá codificarse) encargada de encender el *display* visualizando el número indicado cada 100 slots y apagándolo transcurridos un número de *slots*, desde el momento del encendido, igual al brillo indicado. El efecto es que se aparará y encenderá a una frecuencia de 50 Hz manteniéndose encendido un intervalo de tiempo proporcional al nivel de brillo deseado. Si el brillo es 0, el *display* permanecerá apagado. El prototipo de la función será:

```
void segs_putchar_dimming( uint8 n, uint8 brightness );
```