



Laboratorio 3:

Drivers de dispositivos elementales

salida por leds y displays 7-segmentos

Programación de sistemas y dispositivos

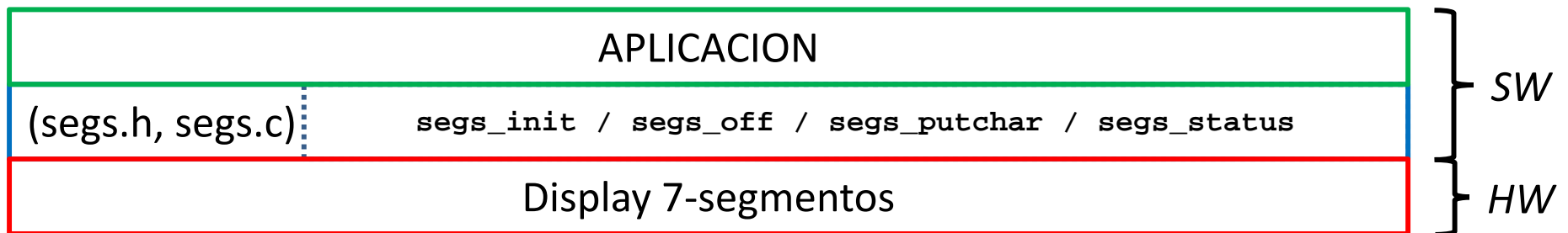
José Manuel Mendías Cuadros

*Dpto. Arquitectura de Computadores y Automática
Universidad Complutense de Madrid*

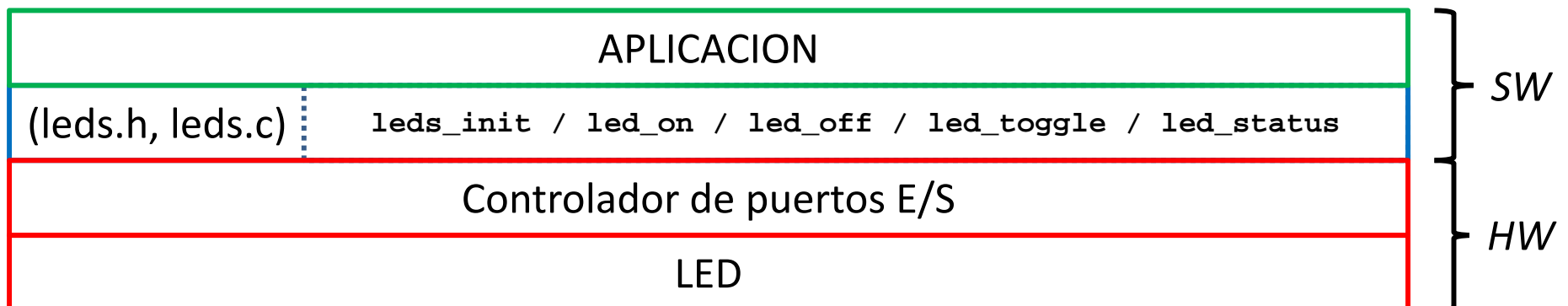


Presentación

- Desarrollar una capa de firmware para **escribir en un display 7-segs**
 - Permitirá visualizar dígitos, apagar el display y conocer el dígito que se visualiza
 - El **dígito lo almacenará el driver**, dado que el display es un dispositivo de solo escritura.



- Desarrollar una capa de firmware para **apagar/encender leds**
 - Permitirá identificar cada led, apagarlo/encenderlo y conocer su estado
 - El **estado lo almacenará el driver**, dado que un led es un dispositivo de solo escritura.





Definición de tipos

`common_types.h`

```
#ifndef __COMMON_TYPES_H__
#define __COMMON_TYPES_H__

typedef unsigned char      boolean;
typedef signed char        int8;
typedef signed short int   int16;
typedef signed int         int32;
typedef signed long long int int64;
typedef unsigned char      uint8;
typedef unsigned short int  uint16;
typedef unsigned int        uint32;
typedef unsigned long long int uint64;

#define NULL ((void *) 0)

#define TRUE (1)
#define FALSE (0)

#define ON (1)
#define OFF (0)

#endif
```

evita referencias circulares y redeclaraciones

definición de tipos atómicos con indicación de su representación hardware (signo y tamaño)

definición de macros de utilidad

Definición de nemotécnicos

s3c44box.h



```
#ifndef __S3C44BOX_H__
#define __S3C44BOX_H__
```

```
#include <common_types.h> ..... utiliza los tipos de datos definidos para la plataforma
```

```
#define PCONA      (*(volatile uint32 *)0x1d20000)
#define PDATA      (*(volatile uint32 *)0x1d20004)
#define PCONB      (*(volatile uint32 *)0x1d20008)
#define PDATB      (*(volatile uint32 *)0x1d2000c)
#define PCONC      (*(volatile uint32 *)0x1d20010)
#define PDATC      (*(volatile uint32 *)0x1d20014)
#define PUPC       (*(volatile uint32 *)0x1d20018)
...
```

definición de nemotécnicos para las direcciones en donde están mapeados **todos los registros de todos los dispositivos internos** del s3c44box

```
#define BIT_ADC      (1<<0)
#define BIT_RTC      (1<<1)
#define BIT_UTXD1    (1<<2)
...
```

definición de nemotécnicos para los bits de los registros de máscara, interrupción pendiente, etc. del controlador de interrupciones del s3c44box

```
#endif
```

Definición de nemotécnicos

s3cev40.h



```
#ifndef __S3CEV40_H__
#define __S3CEV40_H__

#include <common_types.h> ..... utiliza los tipos de datos definidos para la plataforma

#define CPU      ("S3C44B0X")
#define CLKIN    (8000000U)

#define ROM_START_ADDRESS (0x00000000)
...
#define SEGS      (*(volatile uint8 *)0x02140000)
#define KEYPAD_ADDR ((volatile uint8 *)0x06000000)

#define KEYPAD_KEYDOWN_DELAY (30) ..... características físicas de los dispositivos
...

#define pISR_RESET (*(volatile uint32 *)0xc7fff00)
#define pISR_UNDEF (*(volatile uint32 *)0xc7fff04)
...

#define BIT_PB      (1<<21) // EINT4567
#define BIT_ETHERNET (1<<22) // EINT3
...

#endif
```

definición de nemotécnicos de para los límites de las regiones de memoria así como las direcciones de mapeo de los dispositivos externos

definición de nemotécnicos para las direcciones en donde ubicar las direcciones de RTI

Driver del display 7-segmentos

segs.h



- Declaración de **rutinas públicas** para:
 - inicialización del driver y del dispositivo
 - entrada/salida con el dispositivo

```
#ifndef __SEGS_H__
#define __SEGS_H__

#include <common_types.h> ..... utiliza los tipos de datos definidos para la plataforma

#define SEGS_OFF (0xff) ..... Declara una macro para indicar display apagado

void segs_init( void ); ..... Inicializa el driver y apaga el display 7-segmentos

void segs_off( void ); ..... Apaga el display 7-segmentos

void segs_putchar( uint8 n ); ..... Visualiza el número indicado en el display 7-segmentos

uint8 segs_status( void ); ..... Devuelve el número que se está visualizando en el display
7-segmentos o SEGS_OFF si está apagado

#endif
```

Driver del display 7-segmentos

segs.c



- Implementa las rutinas públicas haciendo uso de datos privados

```
#include <s3c44b0x.h>
#include <s3cev40.h>
#include <segs.h>

static const uint8 hex2segs[16] = {0x12, ...};
static uint8 state;
```

} declaración de variables globales (conservan su valor durante toda la ejecución del programa) con visibilidad local a este módulo (static)

```
void segs_init( void )
{
    segs_off();
}

void segs_off( void )
{
    state = SEGS_OFF; ..... actualiza estado
    SEGS = state; ..... actualiza el display
}

void segs_putchar( uint8 n )
{
    state = n & 0x0f; ..... por seguridad, se queda los 4 LSB del argumento
    SEGS = hex2segs[state];
};

uint8 segs_status( void )
{
    return status;
};
```

almacena localmente el número que visualiza del display ya que no es posible leer datos de un display 7-segmentos



Driver de los leds

leds.h / leds.c

- Declaración de **rutinas públicas** de inicialización y E/S
 - Su implementación deberá estar en el fichero **leds.c**

```
#ifndef __LEDS_H__
#define __LEDS_H__

#include <common_types.h>

#define LEFT_LED  (1)
#define RIGHT_LED (2) } Declara macros para identificar a cada led

void leds_init( void ); ..... Inicializa el driver y apaga ambos leds

void led_on( uint8 led ); ..... Enciende el led indicado

void led_off( uint8 led ); ..... Apaga el led indicado

void led_toggle( uint8 led ); ..... Conmuta el led indicado

uint8 led_status( uint8 led ); ..... Devuelve el estado (ON/OFF) del led indicado

#endif
```




Inicialización del sistema

system.h

- En prácticas anteriores hemos asumido que la inicialización del sistema estaba hecha antes de la ejecución del programa
 - De hecho la realiza el programa residente en ROM que se ejecuta tras reset.
- En prácticas sucesivas iremos **completando una rutina** para la inicialización del sistema completo.

```
#ifndef __SYSTEM_H__
#define __SYSTEM_H__

#define USRMODE (0x10)
...
#define SET_OPMODE( mode )
    asm volatile ( "mrs r0, cpsr" : : : "r0" );
    asm volatile ( "bic r0, r0, #0x1f" );
    asm volatile ( "orr r0, r0, %0" : : "i" (mode) );
    asm volatile ( "msr cpsr_c, r0" );
...

void sys_init( void ); ..... Inicializa el sistema

#endif
```

Colección de macros en ensamblador para manipulación del CPSR



Inicialización del sistema

system.c

- En este lab inicializaremos únicamente el controlador de puertos E/S

```
#include <s3c44b0x.h>
#include "system.h"

static void port_init( void );

void sys_init( void )
{
    port_init();
}
```

rutina de soporte privada al driver
(con visibilidad local al módulo)

Sin argumentos. El código no se va a reutilizar en otra plataforma y los dispositivos conectados al SoC son fijos

```
static void port_init( void )
{
    PDATA = ~0;
    PCONA = ...;
    PDATB = ~0;
    PCONB = ...;
    PDATC = ~0;
    PCONC = ...;
    PUPC = ...;
    PDATD = ~0;
    PCOND = ...;
    PUPD = ...;
    PDATF = ~0;
    PCONE = ...;
    PUPE = ...;
    PDATF = ~0;
    PCONF = ...;
    PUPF = ...;
    PDATG = ~0;
    PCONG = ...;
    PUPG = ~0;
    SPUCR = ...;
    EXTINT = ...;
}
```

Inicializa los 22 registros mapeados
en memoria del controlador interno
de puertos de E/S

La inicialización de registros
conviene **hacerla en orden**:
PDATx > PCONx > PUPx



Aplicación

```
#include <common_types.h>
#include "system.h"
#include <segs.h>
#include <leds.h>
```

} inclusión de declaraciones de macros y prototipos

```
void delay( void );
```

```
void main( void )
{
    uint8 i;
```

```
    sys_init();
    segs_init();
    leds_init();
```

} fase de inicialización
inicialización del sistema, drivers y dispositivos

```
    while( 1 )
    {
        ...
        for( i=0; i<16; i++ )
        {
            led_toggle( RIGHT_LED );
            led_toggle( LEFT_LED );
            segs_putchar( i );
            delay();
        }
        ...
    }
    ...
```

} fase de operación,
es un bucle infinito con llamadas a rutinas del driver

Tareas



1. Crear el proyecto **lab3** a partir de una copia de uno anterior.
2. Descargar de la Web en el directorio **lab3** el fichero **lab3.c**
3. Refrescar el proyecto **lab3**.
4. Descargar de la Web en el directorio **BSP/include** los ficheros:
 - **common_types.h**, **s3c44box.h**, **system.h**, **s3cev40.h**, **segs.h** y **leds.h**
5. Codificar en **BSP/source** los ficheros:
 - **system.c**, **segs.c** y **leds.c**
6. Refrescar el proyecto **BSP**.
7. Compilar primero el proyecto **BSP** y después el proyecto **lab3**.
8. Crear una configuración de depuración **lab3** a partir de una anterior.
9. Conectar la placa y encenderla.
10. Arrancar OpenOCD.
11. Arrancar la configuración de depuración **lab3**