Práctica 2: Sistema de E/S y dispositivos básicos

Proyecto Hardware

recordatorio práctica 1

- 21/10 corrección
- 28/10 entrega
- Extensión memoria:
  - aprox. 10 hojas
  - 4000 palabras
    - Sin incluir tabla contenidos, portada y anexos necesarios

Índice

- **Objetivos y descripción de la práctica 2**
- Sistema de E/S de la placa S3CEV40

Objetivo: Desarrollar código para un sistema empotrado real

- En esta placa el procesador está acompañado de muchos dispositivos, principalmente de entrada/salida
- Vamos a aprender a interaccionar con ellos trabajando en C, pero empleando ensamblador cuando sea necesario

- Depurar un código con varias fuentes de interrupción activas
- Depurar un código en ejecución (no sólo paso a paso)

¿Qué tenemos que aprender?

- Entender la configuración de la placa:
  - Utilización de un script de configuración para inicializar el espacio de memoria
  - Registros de configuración de los elementos utilizados
- Gestión del hardware del sistema utilizando C:
  - Utilización de las bibliotecas de la placa
  - Gestión de las interrupciones en C
  - Entender las estructuras que genera el compilador a partir del código fuente (especialmente la pila de programa)

Descripción de la práctica

- Estudiar el proyecto que os damos y aprender a usar:
  - botones
  - 8led (7-segmentos)
  - temporizadores (*timers*)
- Abstracción del Hardware
  - encapsular todas las funciones de entrada/salida
  - emulador trabajar sin la placa
    - `Compilación condicional mediante #ifdef/#else/#endif`

Descripción de la práctica

- Gestión de las excepciones
  - realizar una función de tratamiento de excepciones
  - Al producirse una excepción (data abort, undefined, software interrupt) se invocará a esta función
    - Identifica qué excepción se ha producido
    - Detiene la ejecución
    - Avisa al usuario (ejemplo led parpadeando)

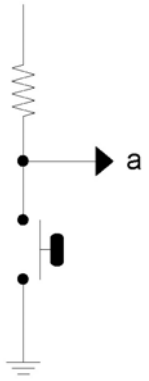
Descripción de la práctica

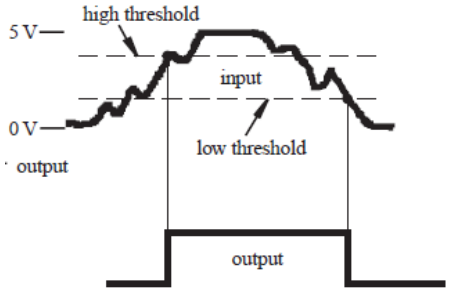
Eventos asíncronos

- Desarrollar una pila de depuración
  - ***push_debug(uint32_t ID_evento, uint32_t auxData)*** : introduce en la pila dos enteros:
    - `Registro con:`
      - `ID_evento`: se puede usar para identificar los eventos
      - `auxData`: datos auxiliares aclaratorios
    - time-stamp (del timer2)
  - Nos dice cuándo se han producido los eventos asíncronos que nos interesan
    (por ejemplo, los rebotes de los pulsadores)
  - Lista circular: no puede salirse de sus límites

Descripción de la práctica

- Eliminar rebotes del teclado y pulsadores
  - iniciar y programar IRQ botón
  - Al detectar pulsación





Descripción de la práctica

- **Jugar**
  - Elegir la implementación más eficiente de las funciones de la **P1**
  - Incluir una máquina de estados
    - Incluyendo la iteración con el usuario y la E/S
      - botones
      - 8-leds
      - timers
      - posteriormente LCD

Máquina de estados

- **El juego comienza al pulsarse un botón**
- **A continuación**
  - El botón izquierdo se usa para introducir Fila y Columna
  - El derecho para confirmar
  - El 8led permite visualizar el valor en cada instante

Máquina de estados

- Al comenzar aparece una F en el 8led
- Al pulsar el botón izquierdo se visualizará un 1 en el 8led
  - Cuando levante el dedo, el número que hay en el 8led se mantendrá fijo.
  - Si de nuevo vuelve a pulsar el botón izquierdo, se irá incrementado como antes. Si llega al 9 se volverá al 1.
- Al pulsar el botón derecho se confirma la fila y se muestra una C en el 8led
- Se repite el proceso para la columna
- Tras confirmar fila y columna el movimiento se procesará

Apartados opcionales

- Auto-incremento:
  - una pulsación de más de 1/3 segundo se ira incrementando el número cada aprox. 180ms
- Permitir interrupciones anidadas
- Estudiar el código de la función `init()` y la estructura del linker script. Detectar un problema que puede aparecer con alguna de las direcciones de los segmentos al declarar datos de tamaños inferiores al tamaño de palabra y plantear una solución.

Fechas de entrega

- Primera parte: trabajar con el proyecto que os damos hasta paso 8
  - Inicio tercera sesión
  - Aproximadamente del 8 al 12 de Noviembre
- Segunda parte:
  - Aproximadamente el 27 de Noviembre
  - Los turnos de corrección aparecerán en Moodle

Uso extra de placa

- TRABAJAR SIN PLACA

- Fuera de las horas asignadas
- Si quedan placas tras el reparto a los del grupo (puntualidad)
  - se realizará un sorteo entre los interesados
  - el uso de horas extra de placa penalizará la nota final de la práctica

Material disponible

- En Moodle disponéis de:
  - Un proyecto que utiliza los pulsadores, leds, 8led y el timer0 de la placa:
    - **debéis seguir el mismo esquema que este proyecto**
    - **OjO contiene bugs y errores conceptuales**
  - Cuadernos de prácticas escritos por compañeros de la Universidad Complutense:
    - **EntradaSalida.pdf**: describe los principales elementos de entrada / salida
    - **P2-ec.pdf**: describe la gestión de excepciones y el mapa de memoria que vamos a usar
  - Documentación original de la placa

Índice

- Objetivos y descripción de la práctica
- **Sistema de E/S de la placa S3CEV40**

Sistema de E/S del S3C44B0X

- Puertos de E/S
  - E/S mapeada en memoria (ARM7TDMI):
    - Los registros que controlan a los distintos elementos de la placa tienen asignados una dirección de memoria
    - La entrada/salida se gestiona escribiendo/leyendo en esas direcciones
  - Registros especiales (internos) tienen reservado el último tramo del banco-0 [0x01C0_0000 - 0x01F_FFFF]
  - El resto de dispositivos (externos) suelen estar ubicados en el banco-1 [0x0200_0000 - 0x03FF_FFFF]

S3CEV40: Sistema de E/S

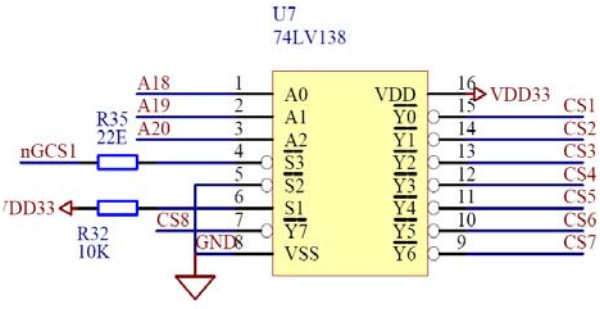
- Dos tipos de dispositivos:
  - Accedidos mediante pines de E/S del S3C44B0X
  - Accedidos mediante direcciones de memoria

Dispositivo	CS	Dirección
USB	CS1	0x0200_0000 – 0x0203_FFFF
Nand Flash	CS2	0x0204_0000 – 0x0207_FFFF
IDE (IOR/W)	CS3	0x0208_0000 – 0x020B_FFFF
IDE (KEY)	CS4	0x020C_0000 – 0x020F_FFFF
IDE (PDIAG)	CS5	0x0210_0000 – 0x0213_FFFF
8-SEG	CS6	0x0214_0000 – 0x0217_FFFF
ETHERNET	CS7	0x0218_0000 – 0x021B_FFFF
LCD	CS8	0x021C_0000 – 0x021F_FFFF
Teclado	nGCS3	0x0600_0000 – 0X07FF_FFFF

S3CEV40: Sistema de E/S

- Selección de chips

A20	A19	A18	CS	Modulo
0	0	0	CS1	USB
0	0	1	CS2	Nand Flash
0	1	0	CS3	IDE
0	1	1	CS4	IDE
1	0	0	CS5	IDE
1	0	1	CS6	8-SEG
1	1	0	CS7	ETHERNET
1	1	1	CS8	LCD



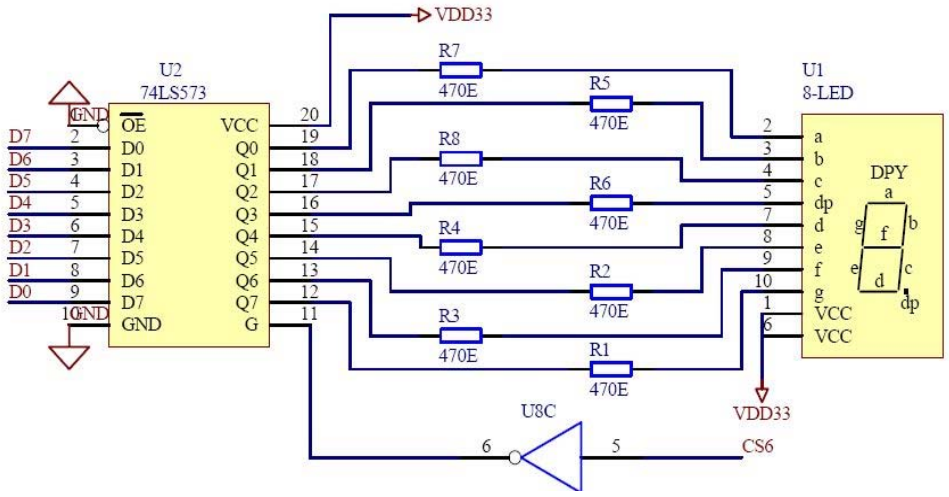
74LV138 is a 3-to-8 line decoder/demultiplexer; inverting

Controlador de pines de E/S (GPIO)

- 71 pines multifuncionales agrupados en 7 puertos (A, B, C, D, E, F y G)
- De momento sólo nos interesan los puertos B y G
- Se manejan mediante 2/4 registros dependiendo del puerto (habitualmente 3)
- Configurados por defecto a un valor seguro

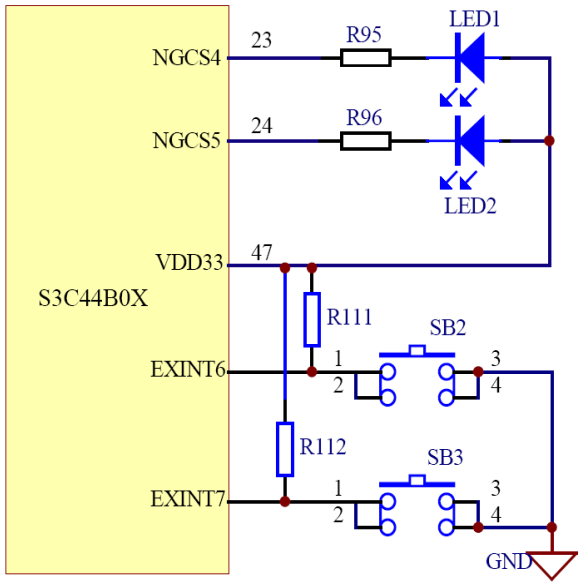
Display 8 segmentos

- Compuesto por 8 diodos
- Se conecta al byte menos significativo del bus de datos mediante latches



Pulsadores y LEDs

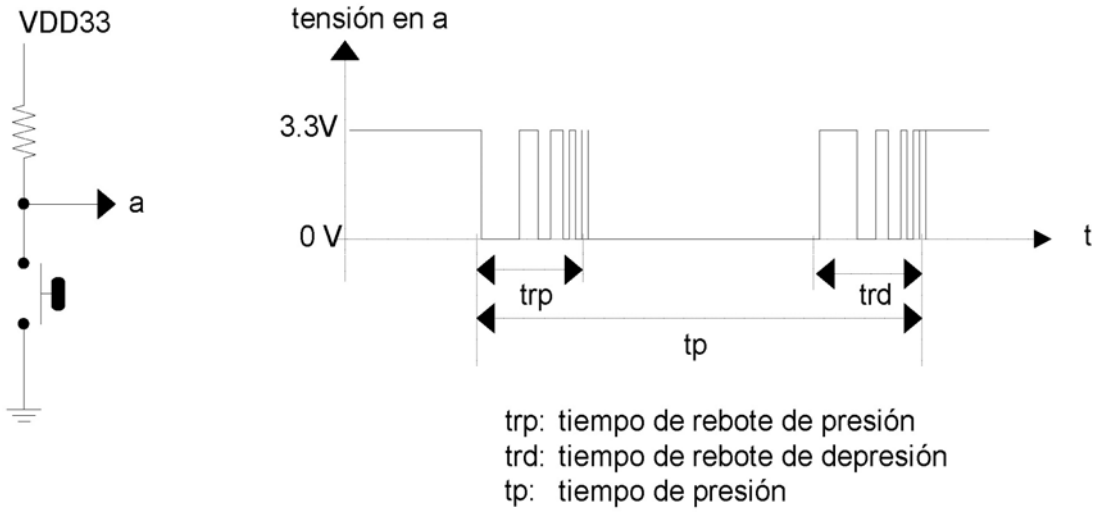
- Comportamiento:
  - LEDs:
    - Si nGS4=0, LED1 on
    - Si nGS5=0, LED2 on
    - PDATB bits 9 y 10
  - Pulsadores (button):
    - SB2 pulsado, EXINT6=0
    - SB3 pulsado, EXINT7=0
    - PDATG bits 6 y 7
    - EXTINTPND bit 2 y 3 en RSI



interface button

- `enum estado_button {button_none, button_iz, button_dr}`
- `button_iniciar()`
  - `inicializa dispositivo`
- `button_empezar()`
  - `re-activa interrupciones`
- `estado_button button_estado()`
  - `lee el estado actual del botón`

Nuestras pulsaciones generan rebotes



trp: tiempo de rebote de presión
trd: tiempo de rebote de depresión
tp: tiempo de presión

El sistema funcionará mal si no se gestionan los rebotes

- Una pulsación puede generar muchas interrupciones:
  - Gestión ineficiente: haces varias veces lo mismo
  - Malfuncionamiento: haces cosas que no deberías

Se pueden solucionar introduciendo retardos

- Rebotes de presión
  - Esperar un tiempo antes de efectuar la identificación de tecla
- Rebotes de depresión
  - Opción 1: Poner un “wait” tras identificar la tecla
    - **¡No es robusta y no nos sirve!**: no sabemos cuándo levantará el dedo el usuario.
  - Opción 2: detectar flanco de subida y de bajada
    - Tendremos una interrupción al presionar y al levantar
  - Opción 3: comprobar periódicamente **si la tecla está pulsada o no**
    - Podemos hacerlo mirando el bit correspondiente del registro **PDATG**
    - Al pulsar generamos un retardo antes de identificar la tecla
    - Al levantar generamos un retardo antes de habilitar de nuevo la interrupción correspondiente
    - **Vamos a seguir este esquema**

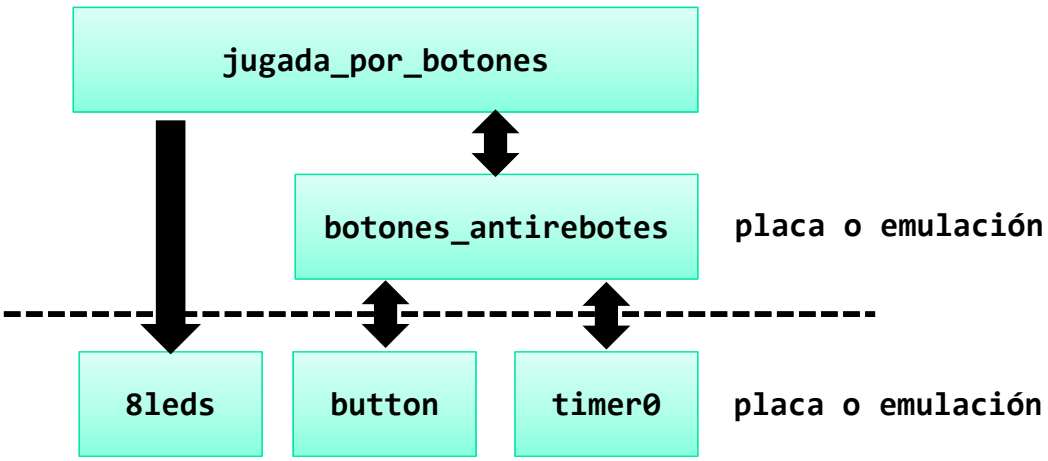
Los retardos se pueden gestionar con timers

- Hay dos formas de gestionar los retardos:
  - Espera activa: el procesador entra en un bucle en el que ejecuta NOPs el tiempo que sea necesario
    - Es muy **ineficiente** porque no puede hacerse trabajo útil durante la espera, y **peligrosa** porque si no se tiene cuidado puede generar bloqueos: **¡no nos sirve!**
  - Programar un temporizador:
    - Le pedimos a un temporizador que nos avise
    - Mientras tanto el procesador puede hacer otras tareas

Pasos a seguir en la gestión de los rebotes:

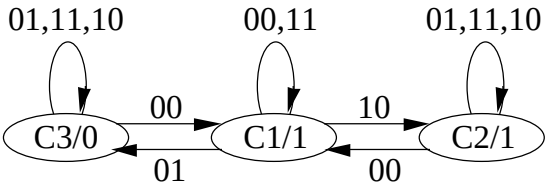
- Al detectar la pulsación:
  - deshabilitamos las interrupciones de los botones
  - identificamos el botón pulsado
  - llamar a la función de callback si está definida.
- Esperar un retardo inicial
- Tras el retardo inicial:
  - comprobar si sigue pulsado (muestreo)
- Cuando se levante el dedo:
  - introducir un retardo final
- Reiniciar en funcionamiento
  - habilitar la IRQ del botón.

Arquitectura sistema botones



Autómatas. Implementación

- Ej.: Detección sentido contrario
  - **MOORE**
  - Entradas nivel muestreadas (síncronas)
  - Salidas asíncronas

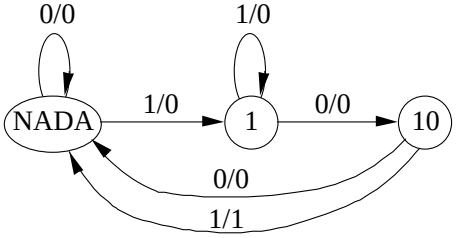



```
// Estado pertenece al conjunto de estados
Entrada  = Leer_Entrada () ;
switch (Estado)
  {
    case C1 : Salida(NO_ALARMA) ;
              switch (Entrada) {
                case I01 : Estado = C3 ; break ;
                case I10 : Estado = C2 ; break ;
                default : }
              break ;
    case C2 : Salida (NO_ALARMA) ;
              if (Entrada  == I00) Estado = C1 ;
              break ;
    case C3 : Salida (ALARMA) ;
              if (Entrada  == I00) Estado = C1 ;
              break ;
  }
```


Autómatas. Implementación

```
Espera_Sincronismo () ;
Entrada  = Leer_Bit () ;
switch (Estado)
  {
   case NADA : if (Entrada==0) {Salida=0; Estado=NADA;}
               else if (Entrada==1) {Salida=0; Estado=E1;}
               break ;
   case E1 :   if (Entrada==0) {Salida=0; Estado=E10;}
               else if (Entrada==1) {Salida=0; Estado=E1;}
               break ;
   case E10 :  if (Entrada==0) {Salida=0; Estado=NADA;}
               else if (Entrada==1) {Salida=1; Estado=E101;}
               break ;
   case E101 : if (Entrada==0) {Salida=0; Estado=NADA;}
               else if (Entrada==1) {Salida=0; Estado=E1;}
  }
Genera (Salida) ;
```

- Ej: reconocedor de cadenas: **MEALY**



Podéis encontrar una forma más eficiente de implementar MSF en: http://johnsantic.com/comp/state.html

Compilación condicional

- En C es posible utilizar directivas del preprocesador para compilar condicionalmente código
- Habréis visto que los ficheros de cabecera suelen incluir una guarda para no tener problemas de doble inclusión

```
#ifndef FICHERO1_H
#define FICHERO1_H
int var;
#endif
```


Compilación condicional

- Si fichero2.h incluye a fichero1.h y fichero_fuente.c incluye a ambos, sin la guarda tendremos dos definiciones de la variable var
- Otro posible uso es escribir múltiples implementaciones de una misma función.

Ejemplo compilación condicional

```
#define USAR_EMULACION

int escribir_pantalla(int n, int pos_x, pos_y) {
#ifdef USAR_EMULACION
  printf("x=%d, y=%d, n=%d\n", x, y, n);
#else
 // utilizar una zona de memoria como pantalla
  pantalla_mem[x * NUM_COL + y] =itoa(n);
#endif
}
```

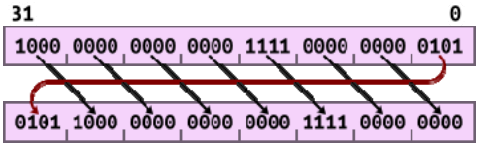
Si antes de compilar comentamos la definición, el código ejecutara la rama del `#else` y viceversa

Uso de ensamblador dentro de código de alto nivel

- A veces es útil añadir código ensamblador dentro de una función de C (u otro lenguaje) para no tener que invocar una función. Por ejemplo para rotar bits en ARM.
  - cuidado, se pierde portabilidad
- Gcc dispone de declaraciones `asm` para este fin. Formato:
  - `asm(código : operandos destino : operandos fuente : elementos contaminados);`

Ejemplo rotación 4 bits a la derecha

- El barrel shifter de ARM permite realizar está operación de manera muy sencilla



- En C es más difícil ...
  - uint32_t y = (x >> 4) | ((x & 0xF) << (32-4));

- Con asm extendido es posible:
  - `asm("mov %[resultado], %[valor], ror #4" : [resultado] "=r" (y) : [valor] "r" (x));`

Fuente: http://www.davespace.co.uk/arm/introduction-to-arm/img/dia/barrel-ror.png

Uso de ensamblador dentro de código de alto nivel 2

- El compilador leerá y escribirá automáticamente las variables además de preservar el valor de los registros
  - suele ser interesante dejar al compilador manejar los registros
- `=r` se utiliza para indicar que el operando es un registro de salida, `r` se utiliza para indicar que el operando es un registro de entrada.
  - Lista completa en http://www.ethernut.de/en/documents/arm-inline-asm.html
- Si dentro de la declaración escribimos en memoria o *flags* de estado es necesario añadir `memory` y/o `cc` en la lista de elementos contaminados
- También podemos escribir `asm volatile` al interactuar con periféricos