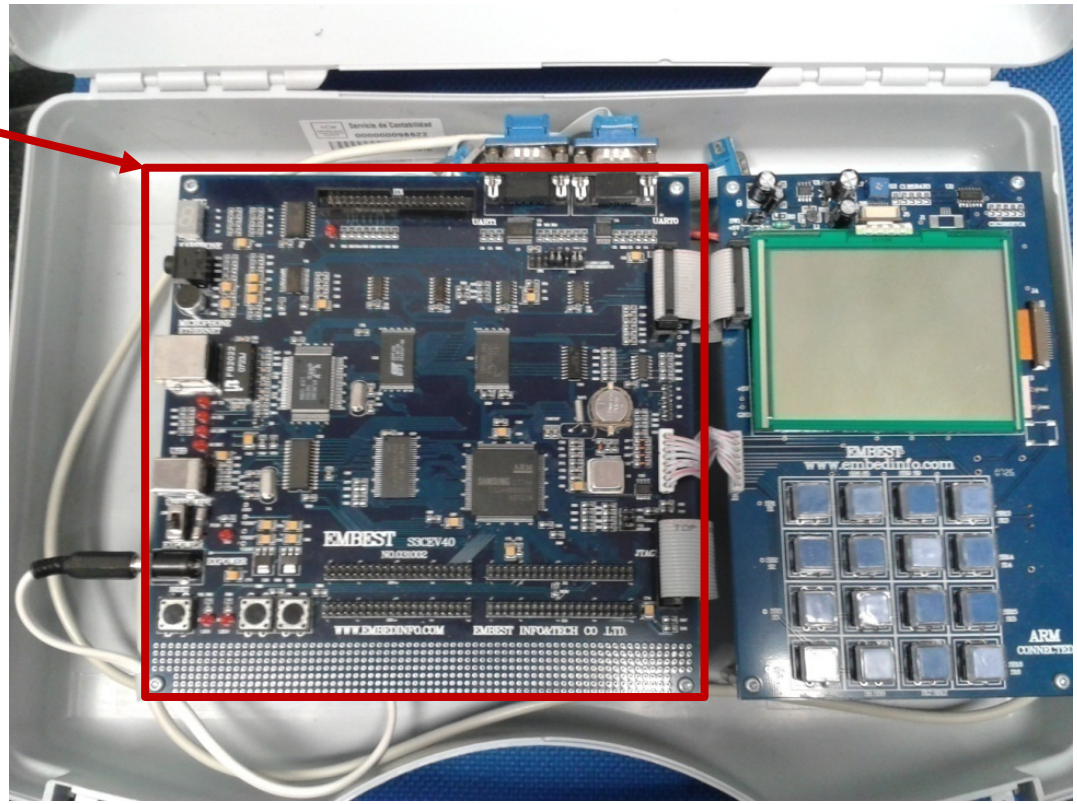
Módulo Entrada/salida

ARM - Placa Embest S·CEV40

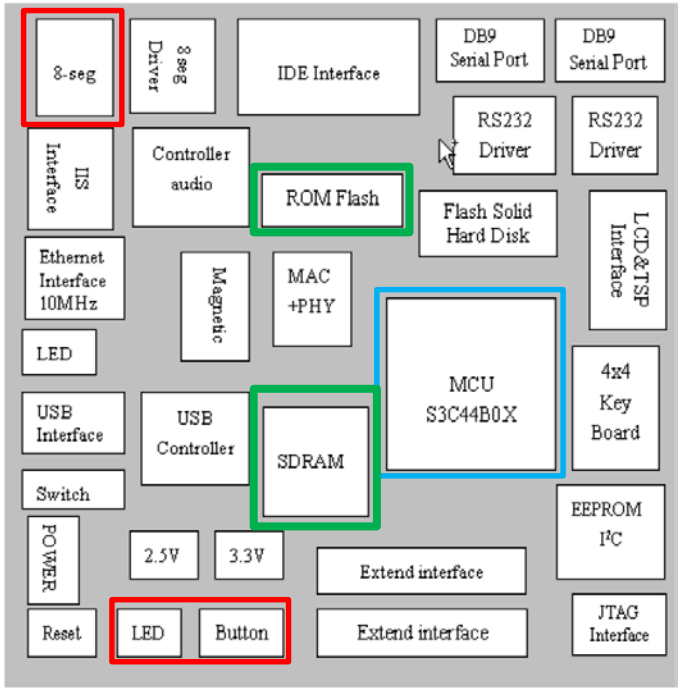
Prácticas: puesto de trabajo

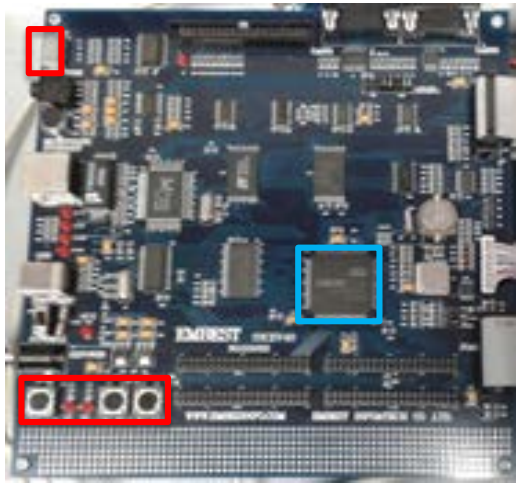
- Kit ARM (maletín)
  - Placa Embest S3CEV40
  - LCD + Touchpad + Teclado
  - Interfaz JTAG Olimex
  - Cables de conexión
- Software
  - Entorno basado en Eclipse + OpenOCD
  - Toolchain GNU



Placa Embest S3CEV40

- **La placa S3CEV40 tiene**
  - Controladores de E/S de:
    - **Display de 8-segmentos**
    - **LCD**
    - **Teclado**
    - …
  - **Los chis de memoria**
  - **System on Chip S3C44B0X** de Samsung:
    - El procesador ARM7TDMI
    - Controlador de memoria
    - Controladores de E/S de:
      - **2 led**
      - **2 pulsadores**
      - …

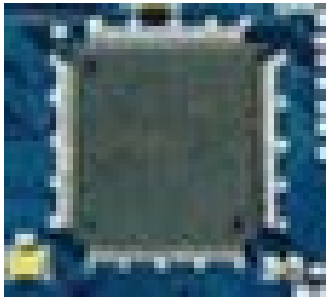


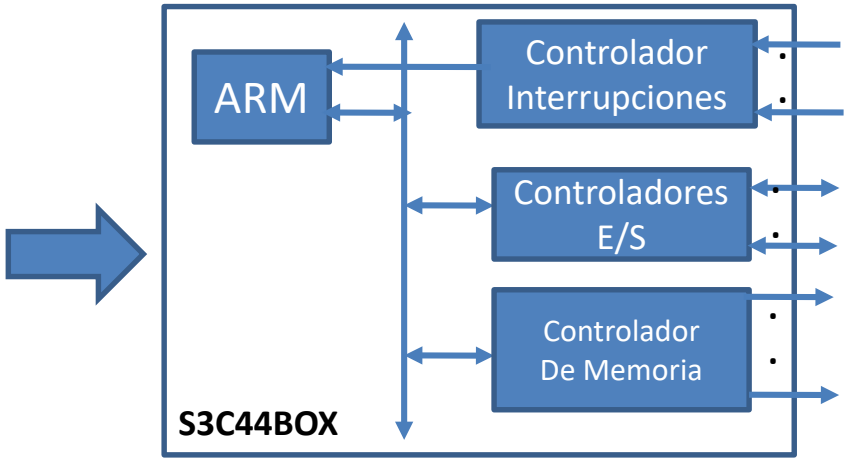


System on Chip S3C44B0X

- **El procesador: ARM7TDMI**
  - Es un procesador de la familia ARM
  - Tiene un bus de direcciones **de 32 bits**
    - Es capaz de direccionar potencialmente un espacio total de 2^{32} **= 4GB** de memoria
- **La E/S está localizada en memoria**
  - La memoria y los dispositivos de E/S **comparten el espacio de direcciones**
  - Para los accesos a E/S se usan instrucciones tipo load/store
    - Es responsabilidad del programador saber a que dispositivo está asociada cada dirección
  - La asignación de rangos de direcciones a dispositivos suele recibir el nombre de **mapa de memoria del sistema**

Chip S3C44B0X



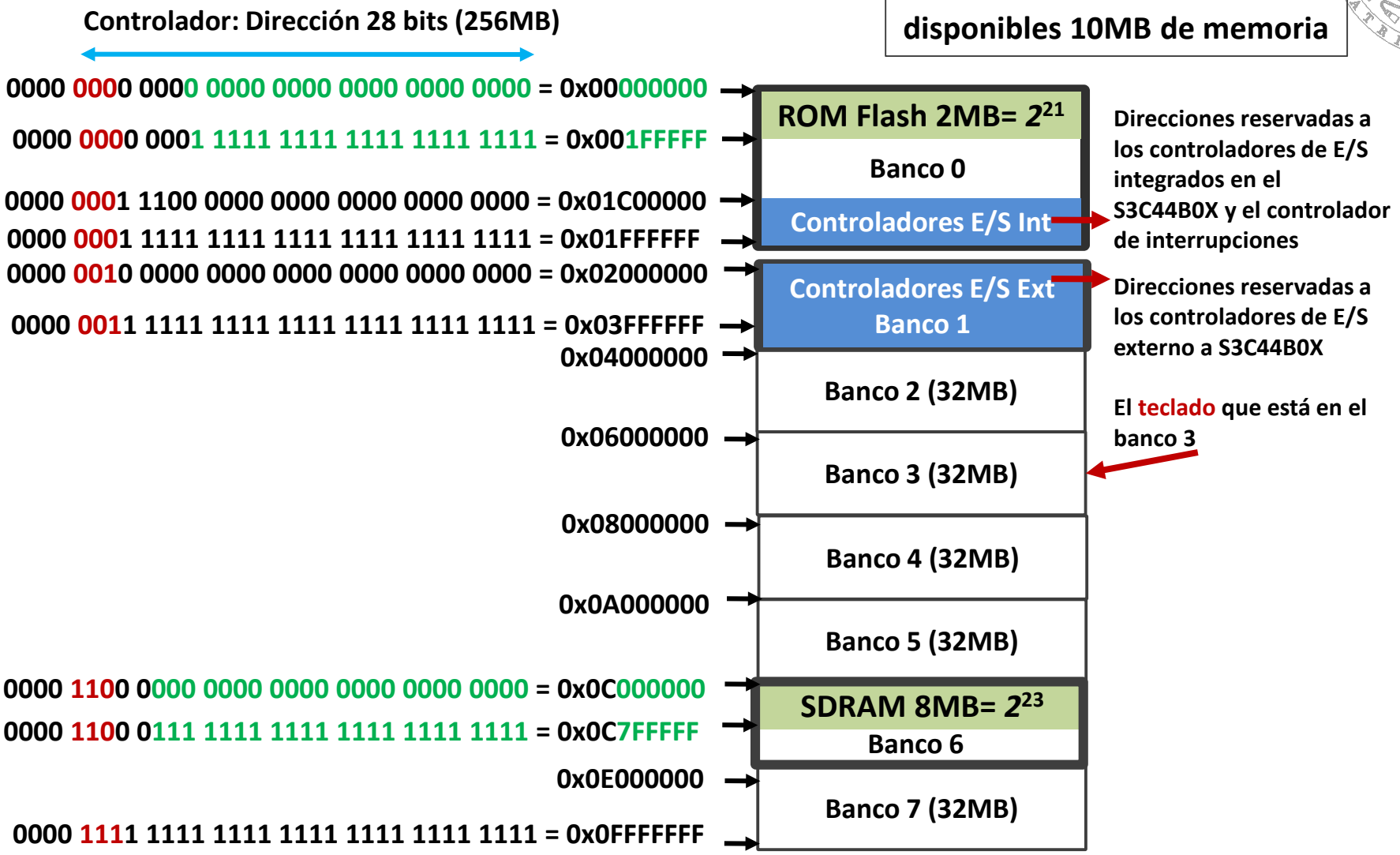


El controlador de memoria

- **Es el responsable de actuar de interfaz entre los módulos de memoria externos (ROM o RAM) y el bus del sistema**
- Reduce el espacio de direcciones efectivo a 2^{28} **=256MB**
- Divide el espacio de direcciones en **8** regiones o rangos independientes: **bancos**
  - Cada banco es de 32MB
  - Cada banco puede ser asignado a un chip de memoria externo distinto (módulo)
  - **En el laboratorio sólo hay conectados chips de memoria al banco 0 y 6**
- **Funcionamiento:**
  - Toma la dirección que hay en el bus y comprueba a que rango de memoria pertenece:
    - Activa la señal **GCS (activa en baja)** para activar el chip correspondiente a ese banco
    - Genera las señales necesarias para realizar el acceso
  - Si el acceso a memoria falla (p.ej. se realiza el acceso a un banco para el que no hay asignado módulo de memoria)
    - Es responsabilidad del controlador detectar el error y generar una excepción de Abort

Mapa de memoria del sistema

En el laboratorio sólo tenemos disponibles 10MB de memoria

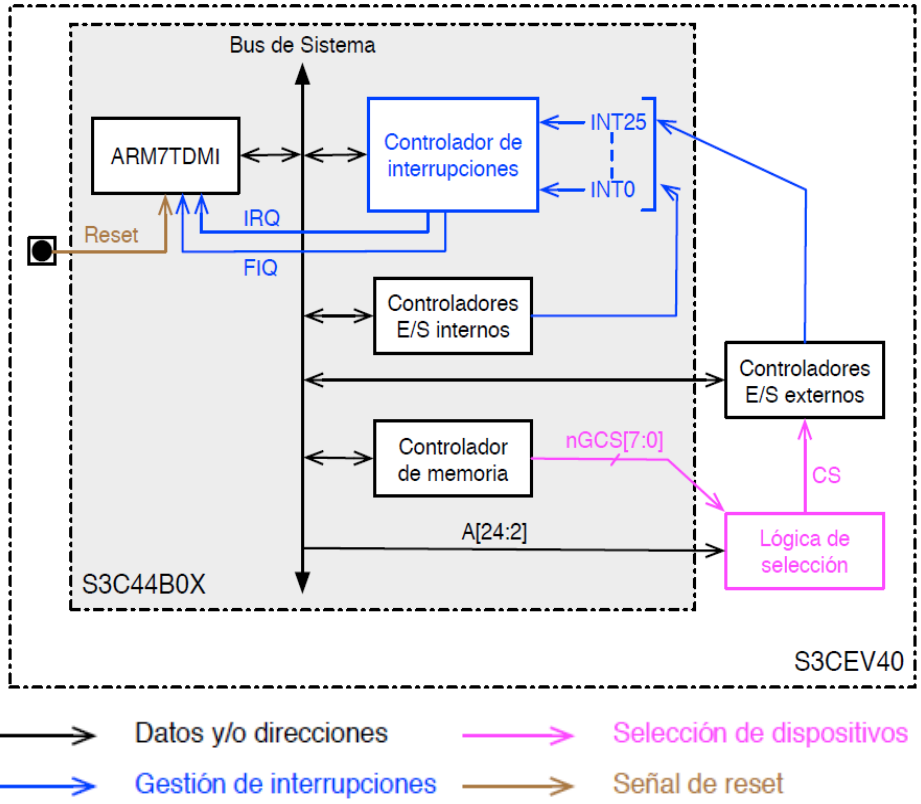


Cada banco se activa con la señal nGCSi (i=0-7) que genera el controlador de memoria

El banco lo indican los bits 25,26 y 27

Sistema de E/S

- El controlador de interrupciones
- Los controladores de E/S internos y externos
  - Dos tipo de dispositivos:
    - Accedidos mediante pines de E/S del S3C44B0X
      - Controlador GPIO : permite gestionar la funcionalidad de los pines multifunción
    - Accedidos mediante direcciones de memoria
      - Lógica de selección para habilitar su señal Chip Select



Controlador de interrupciones

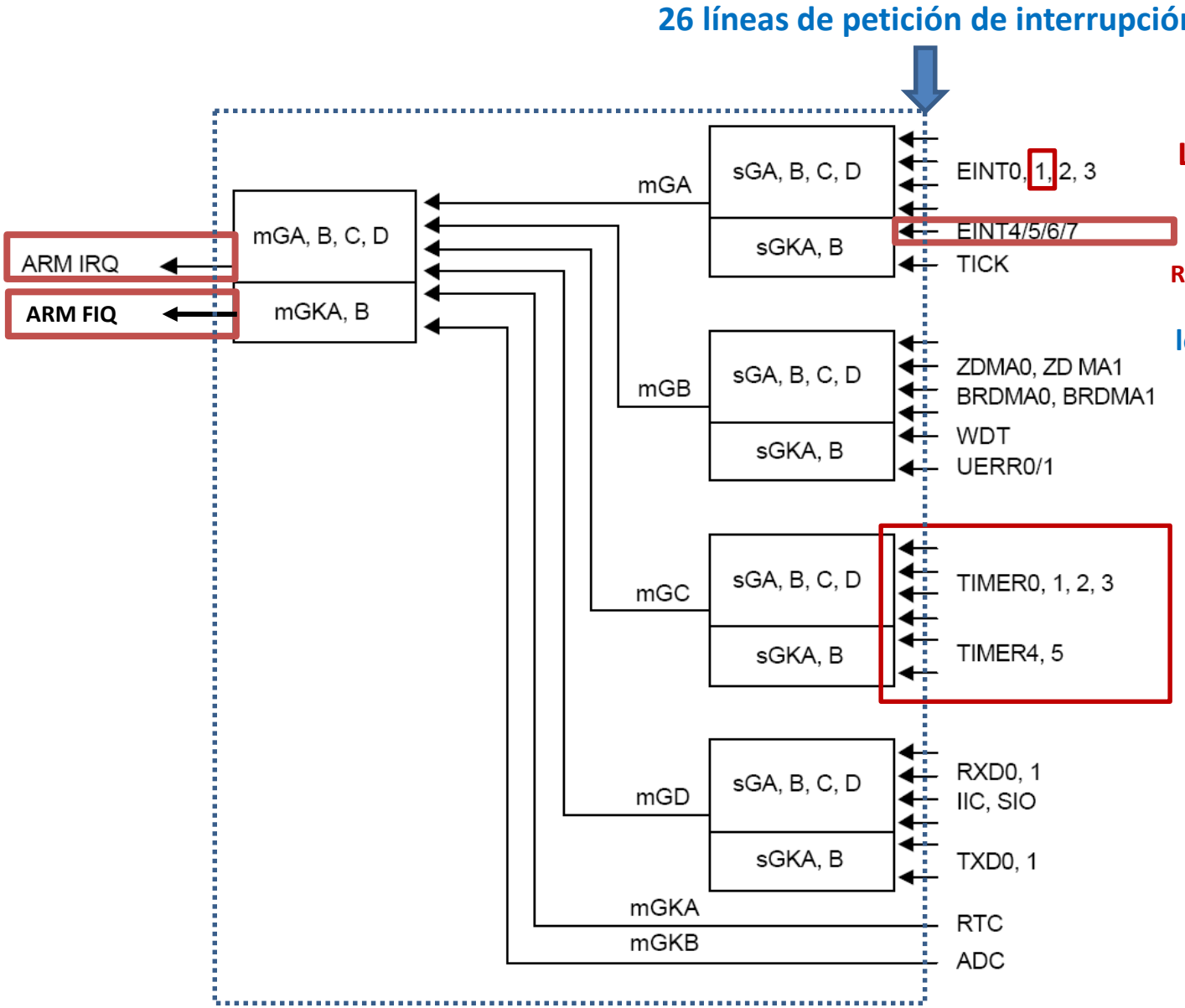
- Sirve para mejorar/ampliar la gestión de interrupciones del procesador

  - Permite **desdoblar las dos líneas** de interrupción, IRQ y FIQ, **en 26 líneas**
    - 30 posibles fuentes de interrupción (la línea 21 y la 14 admiten varias fuentes)

  - Hace que sea **más rápido el proceso de identificación** de la fuente de interrupción
    - Consulta un registro del propio controlador, en lugar de consultar el registro de estado de cada dispositivo
    - Sólo si alguna de las líneas del controlador es compartida por varios dispositivos, es necesario consultar los registros de cada dispositivo

  - Permite definir **dos modos** para las interrupciones:
    - **Autovectorizadas** (No vectoriazadas)
    - **Vectorizadas**

Controlador de interrupciones



Línea 24=INT1
El teclado

Línea 21
Recibe petición de Interrup de 4 dispositivos,
los botones: EINT 6 y 7

Líneas 8 - 13
En las prácticas usaremos un Timer

Controlador de interrupciones

- Para **gestionar las interrupciones** **el controlador tiene los siguientes registros**
  - **INTCON** (Interrupt Control Register), Dir: **0x01E00000**, 1 bit por línea

V = 0, habilita las interrupciones vectorizadas
1, habilita las interrupciones NO vectorizadas
I = 0, habilita la línea IRQ
1, deshabilita la línea IRQ
F = 0, habilita la línea FIQ
1, deshabilita la línea FIQ

  - **INTMOD** (Interrupt Mode Register), Dir: **0x01E00008**, 1 bit por línea

0 = si la línea activa el modo IRQ
1 = si la línea activa el modo FIQ

  - **INTMSK** (Interrupt Mask Register), Dir: **0x01E0000C**, 1 bit por línea

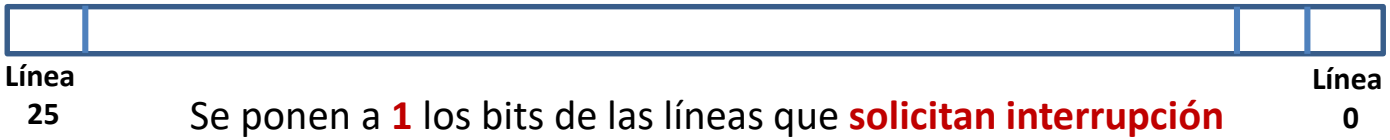
0 = interrupción habilitada ; 1 = interrupción enmascarada

1= enmascara todas las interrupciones

Controlador de interrupciones

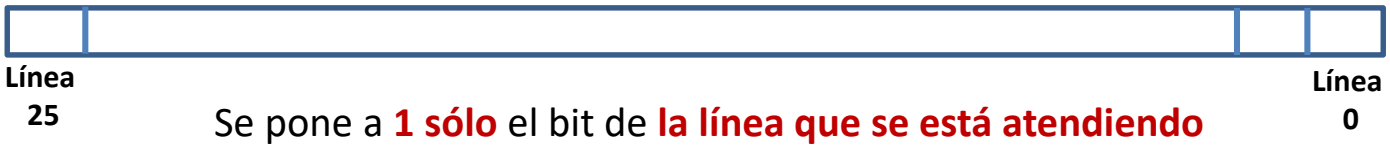
- Para **gestionar las interrupciones** **el controlador tiene los siguientes registros**

  - **INTPND** (Interrupt Pending Register) Dir: **0x01E00004**, 1 bit por línea



Se ponen a **1** los bits de las líneas que **solicitan interrupción**

  - **I_ISPR** (IRQ Int. Service Pending Register), Dir: **0x01E00020,** 1 bit por línea



Se pone a **1 sólo** el bit de **la línea que se está atendiendo**

Este registro es esencial cuando se están en Modo Vectorizado ya que el arbitraje es HW

Estos son Registros sólo de lectura

Controlador de interrupciones

- Para **gestionar las interrupciones** el controlador tiene los siguientes registros

  - **I_ISPC** (IRQ Int. Service Pending Clear register), Dir: **0x01E00024**, 1 bit por línea

Cuando acaba la RTI hay que poner a 1 el bit de la línea de la interrupción que ha acabado de atenderse → el controlador borra el mismo bit del registro de interrupciones pendientes (INTPND)

  - **F_ISPC** (FIQ Int. Service pending Clear register), 1 bit por línea
    - Lo mismo que I_ISPC pero para FIQ

Estos son Registros sólo de escritura

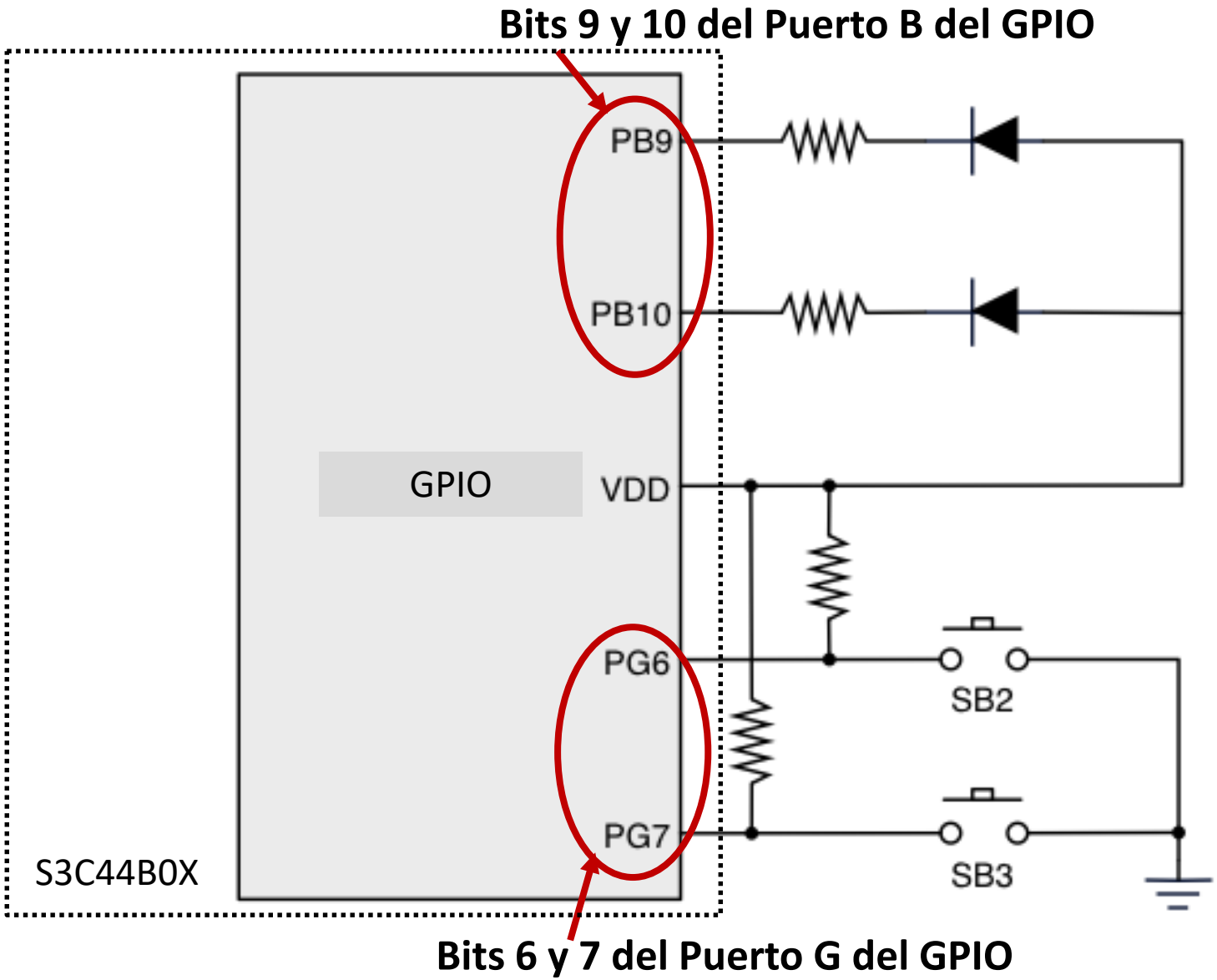
El controlador de E/S interno (GPIO)

- **Mantiene el control de 71 pines** multifuncionales
  - Permite configurar la funcionalidad escogida para cada uno de estos pines

- Los 71 pines, divididos en 7 grupos, a los que denominamos **puertos**:
  - 2 grupos de 9 bits de E/S (Puertos E y F)
  - 2 grupos de 8 bits de E/S (**Puerto**s D y **G**) → Aquí se conectan **los pulsadores**
  - 1 grupo de 16 bits de E/S (Puerto C)
  - 1 grupo de 11 bits de E/S (**Puerto B**) → Aquí se conectan **los LEDS**
  - 1 grupo de 10 bits de E/S (Puerto A)

- Cada puerto
  - Es gestionado mediante 2-4 registros, depende del puerto
  - Por defecto está configurado a un valor seguro teniendo en cuenta lo que hay conectado a dichos pines en la placa
    - **En las prácticas tendréis que configurarlo con el valor adecuado para que haga lo que se os pide**

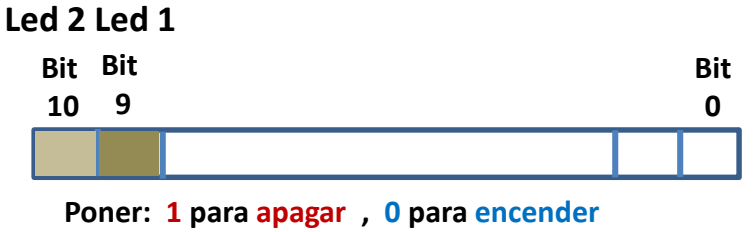
Leds y Pulsadores



Conexión a los leds

- **LEDs**
  - Existen 2 leds
  - Accesibles a través del puerto B

- **Puerto B** (11 bits)
  - Registro de datos: **PDATB**



  - Registro de control: **PCONB**

0 para configurar el pin de salida

Registro	Dirección	R/W	Descripción	Valor por defecto
PCONB	0x01D20008	R/W	Configuración pines	0x7ff
PDATB	0x01D2000C	R/W	Datos	Undef

- **Acciones a realizar**
  - Configurar los bits 9 y 10 del puerto B como pines de salida
  - Escribir en los bits 9 y 10 del registro de datos un valor según se quiera encender o apagar el led

Ejemplo: enceder y apagar los leds

En ensamblador

```
.equ rPDATB, 0x01D2000C
.equ rPCONB, 0x01D20008
```

En C

```
#define rPCONB   (*(volatile unsigned *)0x1d20008)
#define rPDATB   (*(volatile unsigned *)0x1d2000c)
```

Setup: configurar los pines para salida

```
ldr r0,=rPCONB
ldr r1, [r0]
movn r2, #(0x3 << 9)
and r1, r1, r2
str r1, [r0]
```

} bic r1, r1, #(0x3 << 9)

```
rPCONB &= ~(0x3 << 9);
```

Para encender los dos leds

```
ldr r0, =rPDATB
ldr r1, [r0]
movn r2, #(0x3 << 9)
and r1, r1, r2
str r1, [r0]
```

} bic r1, r1, #(0x3 << 9)

```
rPDATB &= ~(0x3 << 9);
```

Para apagar los dos leds

```
ldr r0, =rPDATB
ldr r1, [r0]
or r1, r1, #(0x3 << 9)
str r1, [r0]
```

```
rPDATB |= (0x3 << 9);
```

Conexión de los pulsadores

- **Pulsadores**
  - Existen 2 botones (pulsadores)
  - Accesibles a través del puerto G
- **Puerto G**
  - Registro de datos: **PDATG**
  - Registro de configuración: **PUPG**
    - Permite activar o no una resistencia de pull-up1 por cada pin
  - Registro de control: **PCONG**
    - Permite configurar los pines para
      - Ser de **entrada**
      - Ser de **salida**
      - **Activar una petición de interrupción** por las líneas **EINT*** del controlador de interrupciones

Bot2 Bot1

1 indica NO pulsado 0 indica pulsado

poner 0 para activar la resistencia

00 para configurar el pin como entrada
11 para activar interrupción EINT 6 o 7 (línea 21) **si se pulsa alguno de los botones**

Registro	Dirección	R/W	Descripción	Valor por defecto
PCONG	0x01D20040	R/W	Configuración pines	0x0
PDATG	0x01D20044	R/W	Datos	Undef
PUPG	0x01D20048	R/W	Deshabilitar pull-up	0x0

Ejemplo: leer el estado del pulsador 1

- Si los pines 6 y 7 se **configuran como entrada**, para saber si se ha pulsado un botón hay que leer el bit correspondiente del registro PDATG

En ensamblador

```
.equ  rEXTINT,     0x1d20050
.equ rPDATG,       0x1d20044
.equ  MaskPulSI,   0x00
```

En C

```
#define rEXTINT    (*(volatile unsigned *) 0x1d20050)
#define rPDATG     (*(volatile unsigned *)0x1d20044)
```

Leer del registro de datos del puerto G el estado del botón 1

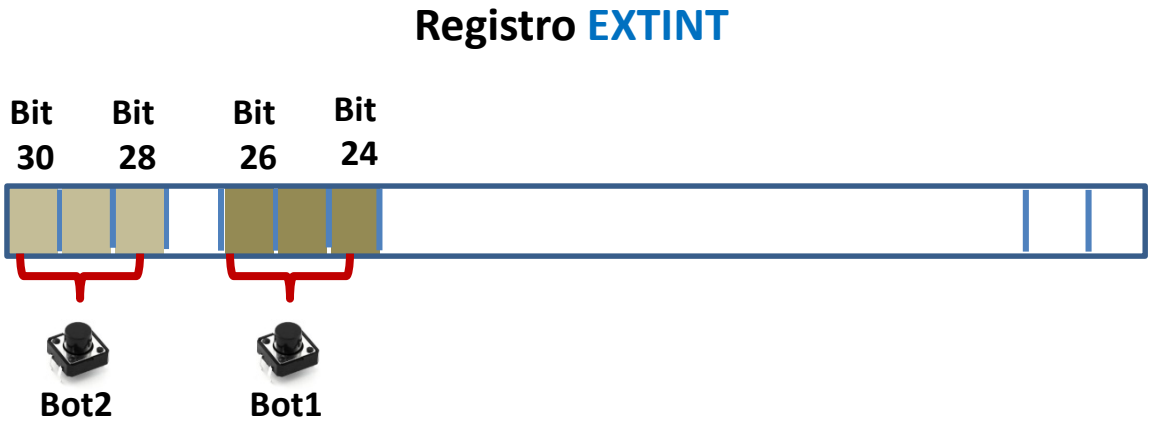
```
ldr r0,=PDATG
ldr r1,[r0]
and r1,r1, #(0x1 <<6)
cmp r1,#MaskPulSI
beq PuladoSI
…
```

```
int  reg

reg = rPDATG
If ( reg & (0x1 << 6) == 0)
            pulsado=SI
else
            pulsado=NO
```

Conexión de los pulsadores

- Si los botones **se configuran para generar interrupción** es necesario configurar **cómo se quiere detectar la interrupción**



- 000 = Interrupción por nivel bajo
- 001 = Interrupción por nivel alto
- 01x = Disparado por flanco de bajada
- 10x = Disparado por flanco de subida
- 11x = Disparado por ambos flancos

Ejemplo: Configurar modo de interrupción del pulsador 1

Pulsador 1 es el **bit 6** del **registro de datos** del puerto G
Para el registro **EXINT** son los **bit 24, 25 y 26**

En ensamblador

```
.equ  rEXTINT,  0x1d20050
.equ rPDATG,   0x1d20044
```

En C

```
#define rEXTINT    (*(volatile unsigned *) 0x1d20050)
#define rPDATG     (*(volatile unsigned *)0x1d20044)
```

Escribir el modo LLOW (Interrupción por nivel bajo, código 000) para el pulsador 1

```
ldr r0,= rEXTINT
ldr r1, [r0]
bic r1, r1, #(0x7 << 24)
str r1, [r0]
```

```
rEXTINT &= ~(0x7 << 24);
```

Escribir el modo LHIGH (Interrupción por nivel alto, código 001) para el pulsador 1

```
ldr r0, = rEXTINT
ldr r1, [r0]
bic r1, r1, #(0x7 << 24)
orr r1, r1, #(0x1 << 24)
str r1, [r0]
```

```
rEXTINT & = ~(0x7 << 24)
rEXTINT | =  (0x1 << 24);
```

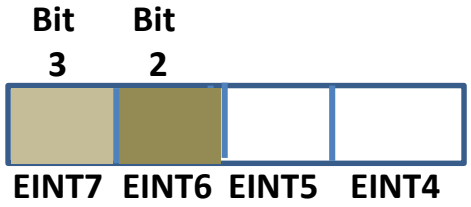
También se puede poner así:

```
rEXTINT = (rEXTINT & ~(0x7 << 24)) | (0x1 << 24);
```

Conexión de los pulsadores

- Una vez configurado cómo se quiere detectar la interrupción, cómo saber **que botón** es el que **ha generado la interrupción**
  - Si se pulsa el **botón 1** se activa **EINT6**
  - Si se pulsa el **botón 2** se activa **EINT7**
  - Ambas señales (EINT6 y EINT7) entran por la **misma línea** (21) del controlador de interrupciones cómo saber cual ha sido

Registro EXTINTPND



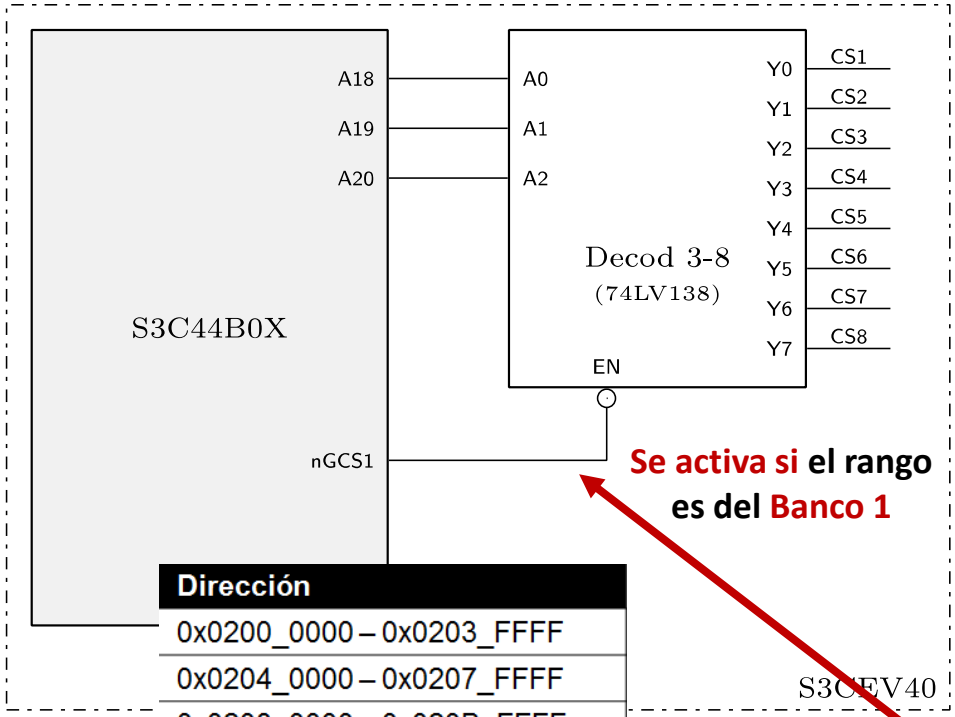
Al **pulsar botón 1** → **EINT6 se activa y el bit 2** de **EXINTPND y el bit 21** de **INTPND** se ponen a 1

Al **pulsar botón 2** → **EINT7 se activa y el bit 3** de **EXINTPND y el bit 21** de **INTPND** se ponen a 1

NOTA: la **RTI que atiende estas interrupciones** tiene que borrar el bit correspondiente de este registro (**escribiendo un '1', no un '0'**) **antes de borrar el bit de ISPC del controlador de interrupciones**

Dispositivos externos

- **Su rango de direcciones es del Banco 1 del mapa de memoria**
  - **Nosotros vamos a usar 8-segmentos**



A20	A19	A18	CS	Dispositivo
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

Se activa si el rango es del Banco 1

los bits 18, 19 y 20 activan la señal CS del dispositivo

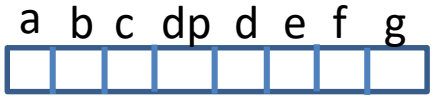
Dirección
0x0200_0000 – 0x0203_FFFF
0x0204_0000 – 0x0207_FFFF
0x0208_0000 – 0x020B_FFFF
0x020C_0000 – 0x020F_FFFF
0x0210_0000 – 0x0213_FFFF
0x0214_0000 – 0x0217_FFFF
0x0218_0000 – 0x021B_FFFF
0x021C_0000 – 0x021F_FFFF

0x02140000= 000 0010 000X XX00 0000 0000 0000 0000

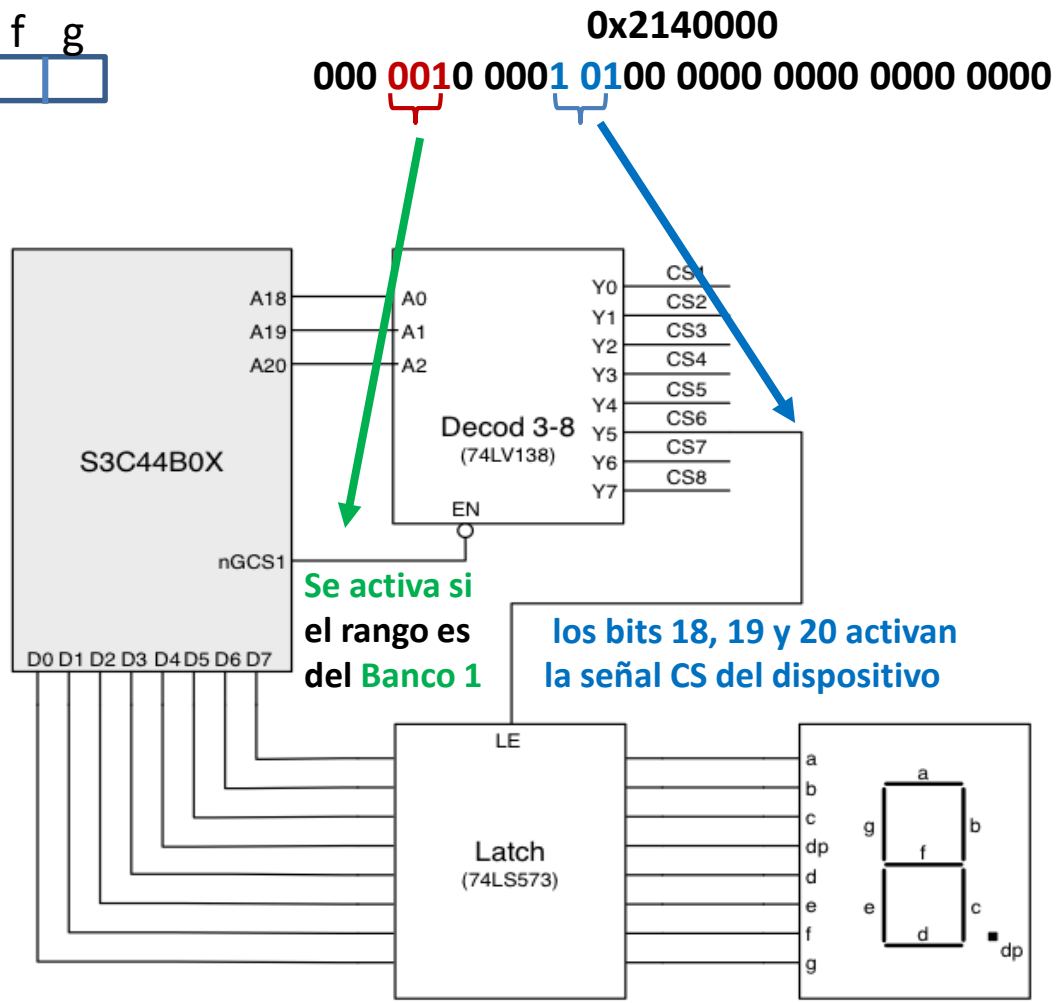
0x02170000= 000 0010 000X XX11 1111 1111 1111 1111

Display 8-segmentos

- Registro de datos (8 bits) en dirección **0x2140000** **(Banco 1 del controlador de memoria)**
  - **Cada segmento tiene asociado 1 bit del registro**
    - Si bit=0 → segmento encendido



- No existe registro de control
  - **Siempre configurados como salida**

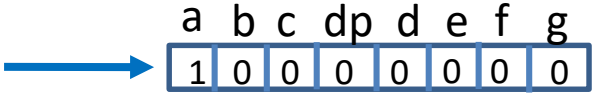


Mascaras útiles para el uso del display de 8 segmentos

- En la mascara ponemos un 1 en el bit del segmento que queremos que se encienda
  - Como el display funciona con lógica invertida, hay que invertir el valor al escribir en el puerto

En C

```
#define SEGMENT_A   0x80
#define SEGMENT_B   0x40
#define SEGMENT_C   0x20
#define SEGMENT_D   0x08
#define SEGMENT_E   0x04
#define SEGMENT_F   0x02
#define SEGMENT_G   0x01
#define SEGMENT_P   0x10
```

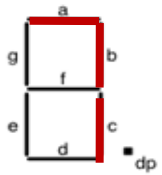


En Ensamblador seria con .equ

Mascaras útiles para el uso del display de 8 segmentos

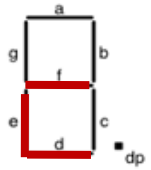
- Con las mascaras anteriores es interesante defirnir:
  - **Los números del 0 al 9.** Ejemplo del número 7

#define DIGIT_7 (SEGMENT_A | SEGMENT_B | SEGMENT_C)



  - **las letras de la A a la F.** Ejemplo de la letra C

#define DIGIT_C (SEGMENT_D | SEGMENT_E | SEGMENT_F)



- **Código para que el display muestre el número 7**

En Ensamblador

```
.equ LED8ADDR, 0x2140000
Ldr R0, =LED8ADDR
Movn R1, # DIGT_7
Str R1 [R0]
```

En C

```
#define  LED8ADDR (*(volatile unsigned char *) 0x2140000)

LED8ADDR = ~DIGT_7
```

Para negar el valor porque para encender hay que poner 0