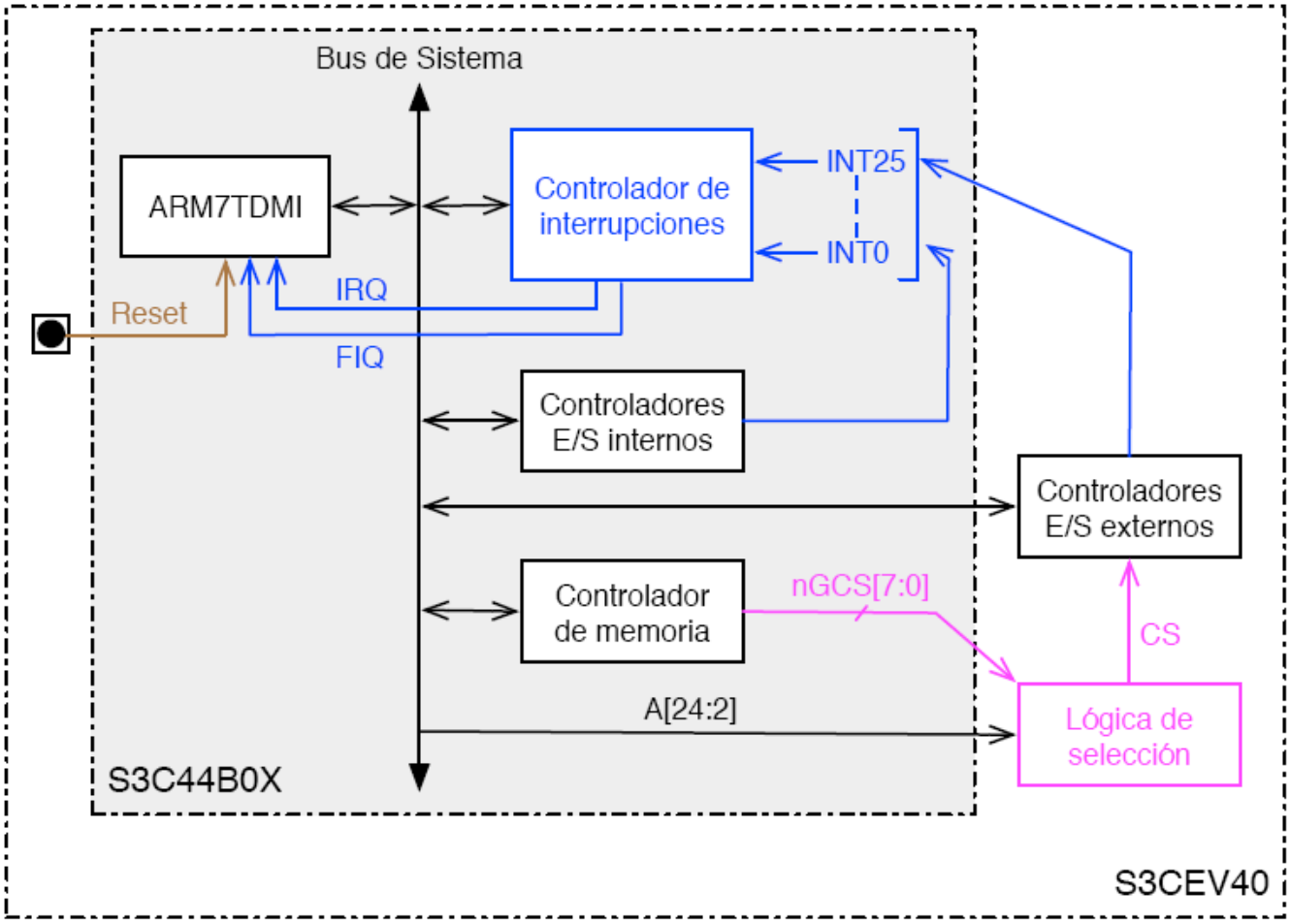
Práctica 2

E/S con interrupciones

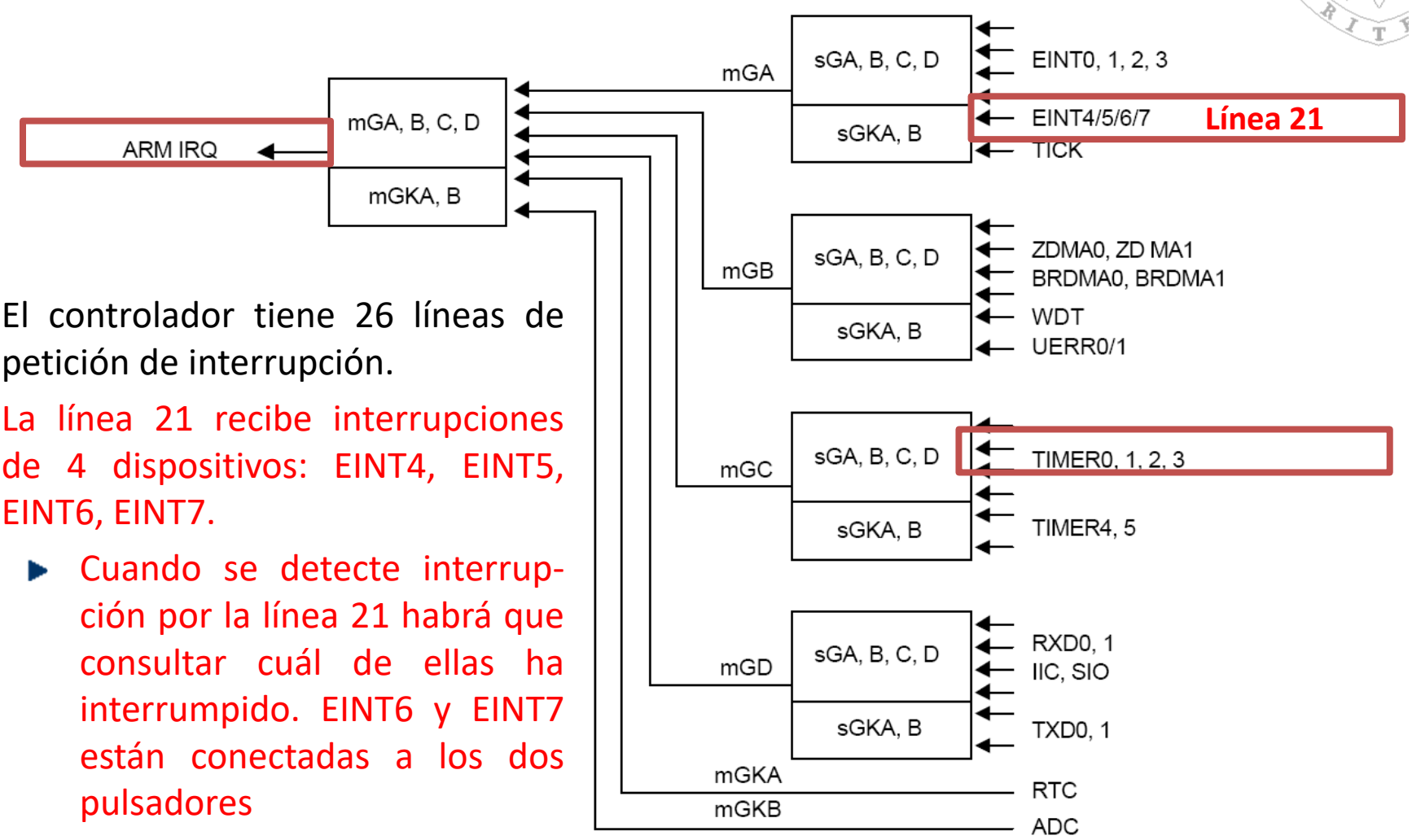
ec

Control de interrupciones en ARM



Datos y/o direcciones
Gestión de interrupciones
Selección de dispositivos
Señal de reset

Controlador de interrupciones



- El controlador tiene 26 líneas de petición de interrupción.
- La línea 21 recibe interrupciones de 4 dispositivos: EINT4, EINT5, EINT6, EINT7.
  - Cuando se detecte interrupción por la línea 21 habrá que consultar cuál de ellas ha interrumpido. EINT6 y EINT7 están conectadas a los dos pulsadores

¿Cómo se configuran los registros desde C?

- Se han definido en 44b.h todos los registros del sistema de entrada salida con el que vamos a trabajar en el laboratorio:

  rNOMBREDELREGISTRO

  – Se les ha asociado un tamaño y una posición fija de memoria

  – Se puede leer o escribir en cualquiera de esos registros utilizando el operador =

Anatomía de una RTI

- Prólogo
  - deberá guardar TODOS los registros que vaya a modificar
- Epílogo
  - restaurar todos los registros modificados
  - retorno DEPENDIENTE de la excepción

Excepción	Inst. Retorno
Reset	NA
Data Abort	`SUBS PC, R14_abt, #8`
FIQ	`SUBS PC, R14_fiq, #4`
IRQ	`SUBS PC, R14_irq, #4`
Prefetch Abort	`SUBS PC, R14_abt, #4`
Undef	`MOVS PC, R14_und`
SWI	`MOVS PC, R14_svc`

RTI en C

- Las RTI NO tienen por qué hacerse en ensamblador
  - Pero hay que indicar al compilador que la función no es *normal* y que debe generar un prólogo/epílogo especiales

```
void trata_SWI (void)  __attribute__ ((interrupt ("SWI")));
void trata_Undef(void)  __attribute__ ((interrupt ("UNDEF")));
void trata_FIQ(void)  __attribute__ ((interrupt ("FIQ")));
void trata_Pabort(void)  __attribute__ ((interrupt ("ABORT")));
void trata_Dabort(void) __attribute__ ((interrupt ("ABORT")));
void trata_IRQ(void)  __attribute__ ((interrupt ("IRQ")));
```

Controlador de interrupciones

- Funcionamiento interrupciones vectorizadas:
  - Cuando el ARM intenta leer la instrucción en la dir. 0x18 (vector IRQ)
  - El controlador actúa sobre el bus de datos e inserta una instrucción de salto al vector correspondiente a la línea más prioritaria activa
    - EINT0 (0x20)
    - EINT1 (0x24)
    - …
    - EINT4/5/6/7 (0x30)
    - …
    - INT_ADC (0xc0)
  - En las correspondientes direcciones de memoria es necesario almacenar instrucciones de salto a las correspondientes rutinas de servicio
  - Afortunadamente el compilador hace el trabajo por nosotros

Controlador de interrupciones

- Desde lenguaje C:

#include "44b.h“

Cabecera de macros que permite manipular las direcciones usuales de manera simbólica

Declaración de que mi_rutina_RTA es una rutina de tratamiento de interrupción:

```
void mi_rutina_RTI(void) __attribute__ ((interrupt ("IRQ")));
```

Registrar mi_rutina_RTI como rutina de tratamiento de interrupción de la línea 21 (EINT4567):

```
pISR_EINT4567   = (unsigned)mi_rutina_RTI;
```

Conexión de los pulsadores

- Puerto G (8 pines)

Registro	Dirección	R/W	Descripción
PCONG	0x01D20040	R/W	Configuración pines
PDATG	0x01D20044	R/W	Datos
PUPG	0x01D20048	R/W	Deshabilitar pull-up
EXTINT	0x01D20050	R/W	Tipo de interrupción: por nivel o por flanco
EXTINTPND	0x01D20054	R/W	Consultar y borrar interrupción

  - Con la configuración por PCONG=0xf000 se activan las líneas de interrupción EINT*
    - **PG7 = EINT7**
    - **PG6 = EINT6**
  - Para que funcionen es necesario PUPG=0x0

Configuración de botones por interrupciones

- Siguen estando en el puerto G (pines 6 y 7)
  - Línea de interrupción: *EINT4567*
    - Los dos botones van por la misma línea
  - Ya no son entradas: queremos que sean fuente de interrupción
    - PCONG: 6 y 7. ¡¡Modificar sólo esos!!
  - Necesitamos registros para configurar las interrupciones
    - Y demultiplexar los dos botones

Puerto G: configuración de interrupciones

- EXTINT (External Interruption Control Register): 3 bits por línea externa de interrupción
  - 000: interrumpe cuando el nivel es bajo
  - 001: nivel alto
  - **01x: flanco bajada** (Usaremos este)
  - 10x: flanco de subida
  - 11x: ambos flancos
- Recuerda: habilitamos la resistencia de *pull-up*

Distinguir entre EINT 4 5 6 y 7

- Registro EXTINTPND (External Interruption Pending register): 4 bits
  - Sirve tanto para **consultar** como para **borrar** (escribiendo 1s en los bits a borrar) el bit de interrupción pendiente

Interrupción	[3]	[2]	[1]	[0]
EINT0	0	0	0	1
EINT1	0	0	1	0
EINT2	0	1	0	0
EINT3	1	0	0	0
EINT4*	0	0	0	1
EINT5*	0	0	1	0
EINT6*	**0**	**1**	**0**	**0**
EINT7*	**1**	**0**	**0**	**0**

* bit 21 de INTPND = 1

Controlador de interrupciones

Registro	Dirección	Descripción
INTCON	0x01E00000	Control de interrupciones
INTMOD	0x01E00008	Modo de interrupciones (IRQ o FIQ)
INTMSK	0x01E0000C	Máscara de interrupciones
INTPND	0x01E00004	Interrupciones pendientes
I_ISPC	0x01E00024	Borra interrupción pendiente por IRQ

Inicialización del controlador de interrupciones

- Registros de configuración

1. INTCON (Interrupt Control Register), 3 bits
   - V (bit [2]) = 1, no habilita las interrupciones vectorizadas
   - I (bit [1]) = 0, habilita la línea IRQ
   - F (bit [0]) = 1, no habilita la línea FIQ
2. INTMOD (Interrupt Mode Register), 1 bit por línea
   - 0 = modo IRQ;
3. INTMSK (Interrupt Mask Register), 1 bit por línea (0..25) + bit 26 para máscara global
   - 0 = int. disponible; 1 = int. enmascarada
   - Bit 26: bit GLOBAL para enmascarar todas las interrupciones

Controlador de interrupciones

- Registros de gestión
  - INTPND (Interrupt Pending Register), 1 bit por línea (0 – 25)
    - 0 = no hay solicitud; 1 = hay una solicitud
  - INTMSK (Interrupt Mask Register), 1 bit por línea (0 – 25)
    - 0 = int. disponible; 1 = int. enmascarada
    - Bit 26: bit GLOBAL para enmascarar todas las interrupciones
  - I_ISPR (IRQ Int. Service Pending Register), 1 bit por línea (0-25)
    - 1 en la línea cuya interrupción se está tratando en ese momento
    - Sólo habrá una línea a 1
  - I_ISPC (IRQ Int. Service Pending Clear register), 1 bit por línea
    - 1 = borra el bit correspondiente del INTPND e indica al controlador el final de la rutina de servicio **(!FIN RUTINA SERVICIO!)**
  - F_ISPC (FIQ Int. Service pending Clear register), 1 bit por línea
    - Idem

Subrutina de tratamiento de interrupciones

- La subrutina ISR_IRQ debe:
  - Ejecutar la rutina de detección para averiguar qué pulsadores se han pulsado
  - Cambiar el estado de los leds de acuerdo con la pulsación detectada
  - Borrar las interrupciones pendientes.

Borrar interrupciones pendientes

- En el dispositivo (puerto G)
  1. EXTINTPND (External Int. Pending register), 4 bits
     - Sirve tanto para consultar como para borrar el bit de interrupción pendiente por EINT4/5/6/7.
     - Se borra la interrupción **escribiendo 1** en el bit correspondiente.

     ⇒ STR ¿?, EXINTPND_ADDR
- En el controlador de interrupciones
  - **INTPND** (Interrupt Pending Register), 1 bit por línea: 0 = no hay solicitud; 1 = hay una solicitud
  - **I_ISPC** (IRQ Int. Service Pending Clear Register), 1 bit por línea.
    - Indica FIN de RUTINA de SERVICIO
  1. Para borrar el registro INTPND hay que escribir 1´s en I_ISPC.

     ⇒ STR ¿?, I_ISPC_ADDR

Int	[3]	[2]	[1]	[0]
EINT4*	1	1	1	0
EINT5*	1	1	0	1
EINT6*	1	0	1	1
EINT7*	0	1	1	1

* INTPND[21]=1