



ARMv7 - Generic Interrupt Controller

Alejandro Furfaro

28 de abril de 2022

Contenido

- 1 Controlador de Interrupciones
- 2 Interrupciones de Hardware
 - Características y Modos de trabajo
 - Manejo de Múltiples Interrupciones
- 3 GIC Architecture
 - Generic Interrupt Controller
- 4 Implementación: SITARA 3358. Cortex-A8
 - Descripción funcional
 - Procesamiento de Interrupciones
 - Modelo de Programación Básico
- 5 ANEXO
 - Anticipos de System Programming

Asignación de Interrupciones

- Cualquier sistema de cómputo debe necesariamente disponer de controlador de interrupciones.
- De otro modo no puede gestionar interrupciones provenientes de múltiples fuentes de hardware (situación habitual).
- Normalmente presentan una interfaz para el programador del Core, compuesta por una serie de registros de estados y control.
- El software que se ejecuta en el core utiliza éstos registros para realizar las siguientes actividades mínimamente esenciales:
 - Establecer una máscara para fuentes de interrupción individuales,
 - reconocer (acknowledge) interrupciones provenientes de dispositivos externos,
 - asignar prioridades a cada fuente de interrupción,
 - y determinar cual/es fuente/s de interrupción requieren atención en cada momento.



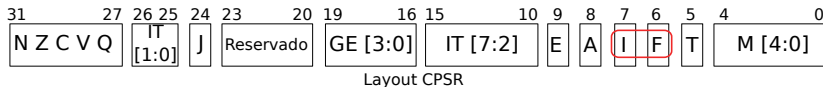
Contenido

- 1 Controlador de Interrupciones
- 2 Interrupciones de Hardware
 - Características y Modos de trabajo
 - Manejo de Multiples Interrupciones
- 3 GIC Architecture
 - Generic Interrupt Controller
- 4 Implementación: SITARA 3358. Cortex-A8
 - Descripción funcional
 - Procesamiento de Interrupciones
 - Modelo de Programación Básico
- 5 ANEXO
 - Anticipos de System Programming



ARMv7 Interrupciones Externas (hardware)

- El Controlador de Interrupciones de un core ARMv7 puede diseñarse en forma específica para el SoC, o ser una implementación de la arquitectura ARM Generic Interrupt Controller (GIC).
- Cualquier procesador de ARM tiene dos líneas de entrada para recibir requerimientos de interrupción desde dispositivos externos: **IRQ** y **FIQ**. Ambas activas bajas.
- Los requerimientos de interrupción que puede enviar la diversidad de dispositivos de E/S conectados al Core, finalmente deben ser mapeados en **IRQ** o **FIQ**.
- Antes del Controlador de Interrupciones...
- Cada entrada responderá al requerimiento siempre que los bits de deshabilitación **CPSR.I** y **CPSR.F** estén en '0'.



ARMv7 Interrupciones Externas (hardware)

La instrucción **CPS** permite manejar de manera simple estos dos bits, además del bit *Abort Enable* **CPSR.A**, y ya que está establecer el Modo del procesador.

Sintaxis:



```
CPSeffect iflags{, #mode}
CPS #mode
```

effect: Define la operación. **IE**: Interrupt/Abort Enable, o **ID**: Interrupt/Abort Disable

iflags: Define el objeto de la operación **a**: aborts imprecisos, **i**: interrupciones **IRQ**, o **f**: Interrupciones **FIQ**

Ejemplos:

```
CPSIE if      /* Habilita IRQ y FIQ. */
CPSID A       /* Deshabilita aborts imprecisos. */
CPSID ai, #17 /* Deshabilita aborts imprecisos e interrupciones, */
              /* e ingresa a modo FIQ */
CPS #16       /* Entra a modo User */
```



Contenido

- 1 Controlador de Interrupciones
- 2 Interrupciones de Hardware
 - Características y Modos de trabajo
 - Manejo de Múltiples Interrupciones
- 3 GIC Architecture
 - Generic Interrupt Controller
- 4 Implementación: SITARA 3358. Cortex-A8
 - Descripción funcional
 - Procesamiento de Interrupciones
 - Modelo de Programación Básico
- 5 ANEXO
 - Anticipos de System Programming



Manejo de Interrupciones Simple

Funcionamiento

Una vez que se produce un tipo de interrupción determinado, las demás interrupciones del mismo tipo se deshabilitan automáticamente, hasta que finalice la atención de la que está en curso. No hay priorización alguna.

Características

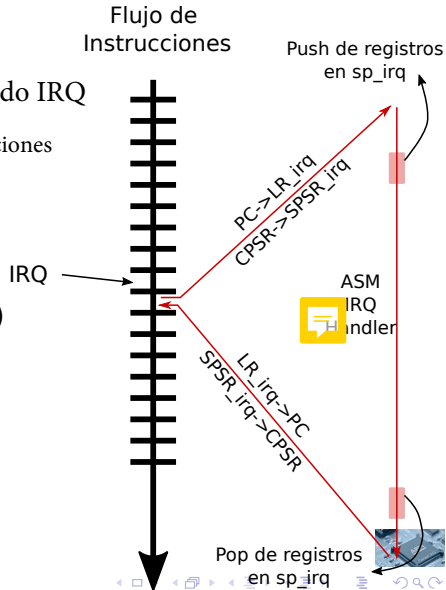
No es apto para sistemas complejos. Apto para probar preliminarmente algún handler no reentrante crítico

el procesador deshabilita las interrupciones, por ende hasta que no se termina de atender a la primer interrupción no se reciben otras interrupciones. Cuando se termina el handler se vuelven a habilitar solas.



Manejo de Interrupciones Simple (IRQ)

- 1 **PC** \rightarrow **LR_IRQ**
- 2 **SPSR_IRQ** \leftarrow **CPSR**
- 3 **CPSR.Mode** [4:0] \leftarrow 10010 modo IRQ
- 4 **CPSR.I** \leftarrow 1 se deshabilitan las interrupciones
- 5 **PC** \leftarrow Vector Table IRQ Entry
- 6 Ejecuta la instrucción de la IRQ Entry (típicamente un salto)
- 7 El handler salva en el stack (push) el/los registros/s que puedan ser alterados. Los restaura (pop) antes de retornar.
- 8 Se ejecuta el core del handler
- 9 Regresa: **PC** \leftarrow **LR_IRQ**,
CPSR \leftarrow **SPSR_IRQ**

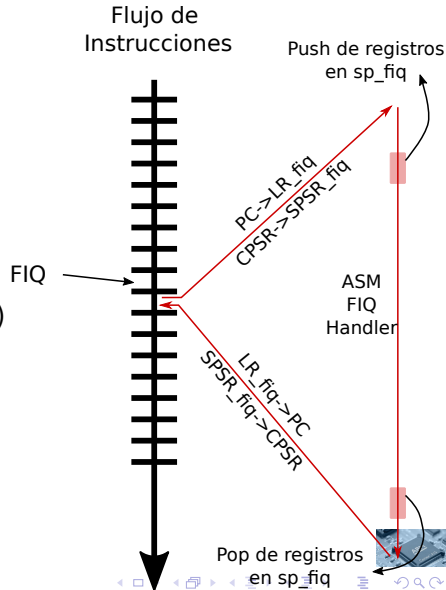


Manejo de Interrupciones Simple (FIQ)



Manejo de Interrupciones Simple (FIQ)

- 1 $PC \rightarrow LR_FIQ$
- 2 $SPSR_FIQ \leftarrow CPSR$
- 3 $CPSR.Mode[4:0] \leftarrow 10001$
- 4 $CPSR.F \leftarrow 1$
- 5 $PC \leftarrow \text{Vector Table IRQ Entry}$
- 6 Ejecuta la instrucción de la IRQ Entry (típicamente un salto)
- 7 El handler salva en el stack (push) el/los registros/s que puedan ser alterados. Los restaura (pop) antes de retornar.
- 8 Se ejecuta el core del handler
- 9 Regresa: $PC \leftarrow LR_FIQ$,
 $CPSR \leftarrow SPSR_FIQ$



Ejemplos

Por el tema de la ABI no pusheamos de r4 a r11

IRQ-Handler :

```
PUSH {r0-r3, r12, lr} /*Almacena registros en la pila */  
BL identify_source /*Devuelve dirección del handler en R0*/  
BLX R0 /*Salta al handler (escrito en C o ASM)*/  
POP {r0-r3, r12, lr} /*Recupera registros de la pila*/  
SUBS pc, lr, #4 /*Retorna utilizando el LR modificado*/
```

- *identify_source*, interactúa con el controlador específico para determinar cual de todas las posibles fuentes de interrupción es la responsable del requerimiento.
- Regresa en **R0** la dirección de la función que implementa el handler correspondiente. Elegimos **R0**, para que *identify_source* sea compatible con el ABI ARM, y pueda invocarse desde C también, si se lo requiere.



Manejo de Interrupciones Anidado

- Anidamiento de interrupciones significa poder aceptar una nueva interrupción antes de finalizar el procesamiento de la interrupción corriente.
- Permite la priorización de interrupciones urgentes, permitiendo alcanzar los requerimientos temporales. Por ejemplo, demora mínima en atención (latency).
- No está impuesto por el hardware.
- El costo del anidamiento es, por lo tanto, mayor complejidad en el software.
- Hay un único handler de atención para las diferentes fuentes de interrupción que implementa condiciones adecuadas de priorización.
- Y debe aceptar re-entrancia.

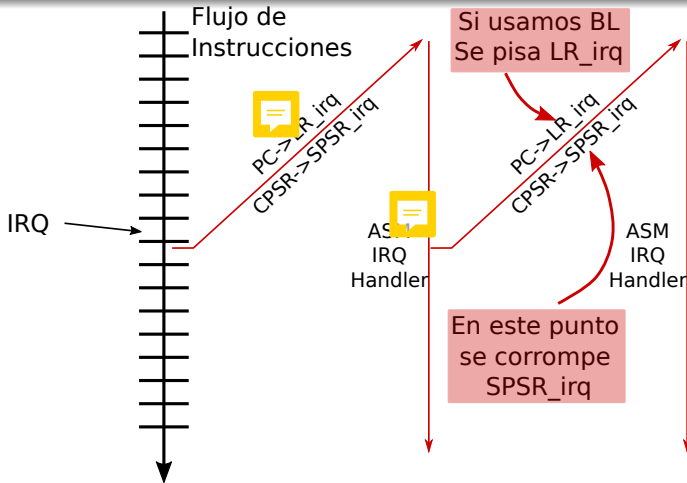


Manejo de Interrupciones Anidado

- Un handler de interrupción re-entrante debe salvar el estado IRQ o FIQ antes de conmutar el core al nuevo estado.
- Además debe salvar el estado del nuevo modo antes de invocar a la función de anidamiento de interrupciones con las interrupciones habilitadas. en esta parte se debe hacer $I=1$
- De otro modo, como las interrupciones de hardware pueden ocurrir en cualquier momento (no son determinísticas), puede pisarse el registro de estados original con el de la nueva interrupción y cuando la interrupción original deba retornar al programa interrumpido originalmente, el registro **SPSR** del modo IRQ o FIQ contiene otro estado, y el programa entra inexorablemente en falla.
- Para evitar esto, el handler anidado debe cambiar a un modo privilegiado en el que deshabilite las interrupciones hasta completar esta tarea.



Manejo de Interrupciones Anidado

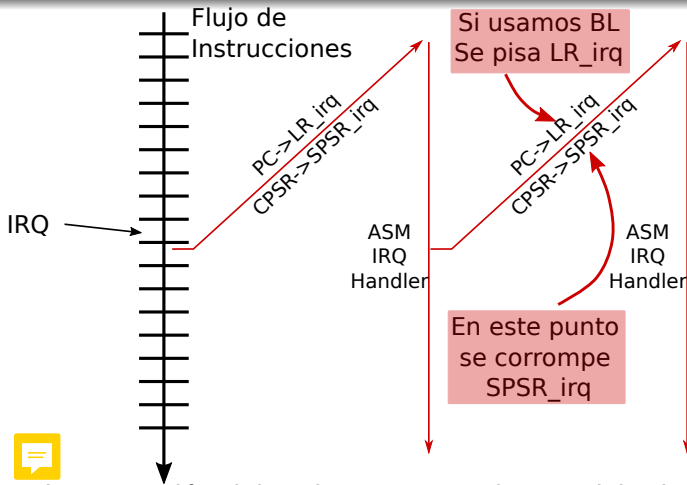


La solución es apilar el **SPSR** antes de habilitar las interrupciones.

```
SRSFD sp!,#0x12/*SRS Store Return Status (LR y CPSR).*/
/*FD: Full Descending, sp: stack, 0x12: modo IRQ.*/
/*!: luego de la operación actualiza SP_irq.*/
```



Manejo de Interrupciones Anidado



Para evitar la corrupción del registro **LR_irq**, lo que debe hacerse es **switchear a modo Supervisor** antes de ejecutar la instrucción de Salto, **BL**. Se retornará a este estado antes de regresar al handler previo.



Secuencia de Handler de Interrupciones Anidado

- 1 El handler de interrupción debe resguardar todos aquellos registros que puedan ser modificados, incluido **SPSR_irq**.
- 2 Identificar la fuente de la interrupción y eventualmente limpiar el bit en el hardware externo, para evitar un re disparo eventual de la misma interrupción
- 3 Cambiar el core a Modo **SVC**, manteniendo el bit **CPSR.I** seteado (interrupciones deshabilitadas).
- 4 Salvar la dirección de retorno en el stack de Modo **SVC** (que estará ubicado en la memoria de modo kernel) y habilitar las interrupciones.



Secuencia de Handler de Interrupciones Anidado

- 5 Invocar al código del handler específico de la fuente de interrupción.
- 6 Al retorno desde éste handler específico, deshabilitar las interrupciones y recuperar la dirección de retorno de excepción desde el stack de Modo **SVC**.
- 7 Recuperar los registros guardados en el stack de Modo **SVC**, incluido el **PC**, y el **CPSR**, el cual devolverá el core a su Modo original. Si el bit **SPRS.I** no estaba seteado, las interrupciones quedarán nuevamente habilitadas.



Ejemplo de Handler de Anidado para IRQ

IRQ_Handler :

```

SUB      lr, lr, #4
SRSFDB  sp!, #0x1f /* LR_irq y SPSR_irq al stack modo Sys... */
CPS      #0x1f      /* CPS switchea a Modo System. */

PUSH     {r0-r3, r12} /* Salva contexto en stack Modo System. */
AND      r1, sp, #4    /* Asegura stack alineado a 8-byte. */
SUB      sp, sp, r1
PUSH     {r1, lr}      /* Ajuste a 8-byte y LR_sys al stack. */

BL       id_source     /* Devuelve dirección del handler en R0 */

CPSIE    i             /* Habilita IRQ con CPS */
BL       R0            /* Invoca al handler reentrante */
CPSID    i             /* Deshabilita IRQ con CPS */

POP      {r1, lr}      /* Recupera LR_sys */
ADD      sp, sp, r1     /* Desajusta stack */
POP      {r0-r3, r12}  /* Recupera los registros */
RFE      sp!           /* Retorna desde el stack de Modo System. */

```

se puede
usar BIC

{

PUSH
AND
SUB
PUSH

{r0-r3, r12} /* Salva contexto en stack Modo System. */
r1, sp, #4 /* Asegura stack alineado a 8-byte. */
sp, sp, r1
{r1, lr} /* Ajuste a 8-byte y LR_sys al stack. */

BL id_source /* Devuelve dirección del handler en R0 */

CPSIE i /* Habilita IRQ con CPS */
BL R0 /* Invoca al handler reentrante */
CPSID i /* Deshabilita IRQ con CPS */

POP {r1, lr} /* Recupera LR_sys */
ADD sp, sp, r1 /* Desajusta stack */
POP {r0-r3, r12} /* Recupera los registros */
RFE sp! /* Retorna desde el stack de Modo System. */



Contenido

- 1 Controlador de Interrupciones
- 2 Interrupciones de Hardware
 - Características y Modos de trabajo
 - Manejo de Múltiples Interrupciones
- 3 **GIC Architecture**
 - **Generic Interrupt Controller**
- 4 Implementación: SITARA 3358. Cortex-A8
 - Descripción funcional
 - Procesamiento de Interrupciones
 - Modelo de Programación Básico
- 5 ANEXO
 - Anticipos de System Programming



Parte de una Arquitectura


Generic Interrupt Controller

- Comprende un set de recursos de hardware para manejar interrupciones en sistema con un solo core o múltiples cores.
- Define un conjunto de registros mapeados en memoria, que permiten gestionar las diferentes fuentes de interrupción y controlar su comportamiento, y en el caso de sistemas multicore, enrutar interrupciones a un core individual determinado.
- Permite habilitar y deshabilitar individualmente las diferentes fuentes de interrupción de hardware, priorizarlas por hardware, y generar interrupciones de software.
- Provee soporte para las Extensiones de Seguridad.
- Acepta interrupciones a nivel del SoC y las mapea en un core individual como IRQ o FIQ.



Dos bloques funcionales (perspectiva del software)

Distribuidor (todo esto es parte del GIC)


- Tiene cableadas todas las fuentes de interrupción del sistema.
- Se compone de un conjunto de registros que controlan las propiedades de cada fuente de interrupción de manera individual, tales como prioridad, estado, seguridad, información de vectorización, y habilitación. 
- Determina cuál de las interrupciones será enviada a un core a través de la **Interfaz con la CPU**.

Interfaz con la CPU

- Es la entrada por donde el core recibe una interrupción.
- Contiene una serie de registros para enmascarar, identificar y controlar el estado de las interrupciones enviadas al core.
- Hay una **interfaz con la CPU** por cada core del sistema.

Identificación de interrupciones

Interrupt ID

Las fuentes de interrupciones se identifican unívocamente en el software mediante un número denominado ***Interrupt ID***. 

En base a este número el software puede invocar al handler de interrupción correspondiente.

Los valores específicos de ***Interrupt ID***, se determinan en el diseño del sistema.

Hay tres tipos de Interrupciones:

- Software Generated Interrupts (SGI)
- Private Peripheral Interrupts (PPI)
- Shared Peripheral Interrupts (SPI)



Software Generated Interrupts (SGI)

- Se generan escribiendo desde el software en el registro **ICDSGIR**, *Software Generated Interrupt Register*, específico del **Distribuidor**.
- Su principal aplicación es la comunicación inter core.
- Pueden destinarse a todos los cores o a un grupo seleccionable de cores.
- Para esto último se reservan los números de interrupción 0 a 15.
- El número exacto que se utiliza para este fin, queda a criterio del diseñador del software de manejo de interrupciones.



Private Peripheral Interrupts

- Se trata de una interrupción generada por una fuente de interrupción privada o exclusiva de un core determinado.
- Se reservan para este tipo los números 16 a 31.
- El número asignado es identificador de una fuente privada del core.
- Puede coincidir con la misma fuente de otro core.
- Un claro ejemplo es el temporizador de cada core



Shared Peripheral Interrupts

- Se trata de una interrupción generada por una fuente de interrupción que puede ser enrutada a mas de un core.
- Se reservan para este tipo los números 32 a 1020.
- Un dispositivo SPI es el caso mas claro, entre otros.
- Su interrupción puede ser derivada a cualquiera de los cores del sistema.



Señalización de una interrupción

- **Edge-Triggered**: Disparo por flanco. Requiere que la señal en la entrada de la fuente de interrupción pase del estado bajo al estado alto (Flanco ascendente de disparo). Permanece vigente hasta que se limpie.
- **Level-Triggered**: Disparo por nivel. Se considera que una fuente de interrupción está activa si el nivel de tensión aplicado en su estado es alto.



Estados de una interrupción

- **Inactive**: Significa que la interrupción aun no ha sido enviada.
- **Pending**: Significa que la interrupción ha sido enviada pero está esperando ser atendida por el core. Cuando están en este estado son candidatas a ser enviadas a la **Interfaz con la CPU**, y mas tarde nuevamente al core.
- **Active**: Indica que la interrupción ha sido atendida por el core y está siendo procesada
- **Active and Pending**: Indica que el core está procesando esta fuente de interrupción pero el GIC tiene un nuevo requerimiento pendiente para esa misma fuente.



Procesamiento de una interrupción en el GIC

- 1 La prioridad de las diferentes fuentes de interrupción y la lista de cores a los que cada una puede ser derivada se maneja en el ***Distribuidor***.
- 2 Cuando se recibe una interrupción de una fuente determinada, ésta se establece en el ***Distribuidor*** en estado ***Pending*** (o ***Active and Pending*** en el caso en que esa fuente de interrupción haya generado un requerimiento previo que sigue estando en proceso)
- 3 El ***Distribuidor***, determina entre las pendientes aquella de mayor prioridad y la deriva a la ***Interfaz con la CPU*** del core correspondiente.
- 4 La ***Interfaz con la CPU*** envía una señal de interrupción por la entrada ***FIQ*** o la entrada ***IRQ*** de la CPU.




Procesamiento de una interrupción en el GIC

- 5 El core ejecuta el handler de la interrupción **IRQ** o **FIQ**, de acuerdo con la entrada por la que ha ingresado el requerimiento de interrupción.
- 6 El software lee el ***Interrupt ID*** de los registros del GIC .
- 7 Con éste número, invoca al handler específico de interrupción del periférico correspondiente a la fuente de interrupción recibida, para finalmente atender el requerimiento concreto.
- 8 Una vez finalizado éste handler, el software debe acceder al registro de la ***Interfaz con la CPU***, para **señalar el fin de la atención de la interrupción.**
- 9 El GIC actualiza el estado de la interrupción en sus registros. Oscilará entre **Active** y **Pending**, mientras la interrupción esté siendo atendida, y quedará en **Inactive** una vez atendida.



Configuración

- El GIC se presenta al software como un conjunto de registros memory-mapped. Se accede como si fuera un periférico.
- El bloque **Distribuidor** es común a todos los cores, pero la **Interfaz con la CPU** bancaea sus registros de manera individual para cada core. 
- Cada core utiliza el mismo rango de direcciones para acceder a su propia **Interfaz con la CPU**.
- Ningún core tiene acceso a los registros de la **Interfaz con la CPU** de otro core. **Nunca**.
- El espacio de registros memory-mapped correspondiente a los registros del **Distribuidor** permite a todos los cores configurar diferentes propiedades.



Configuración - Propiedades

- Prioridad. Permite al **Distribuidor** determinar cual será la siguiente fuente de interrupción a derivar hacia la **Interfaz con la CPU**.
- Es posible configurar para cada core un nivel mínimo de prioridad para que una fuente le sea derivada.
- Configuración de la Interrupción. Determina si la línea de hardware se activa por flanco o por nivel.
- Target de la interrupción. Lista de cores hacia los que se puede derivar el requerimiento de interrupción.
- Habilitación/Deshabilitación. Solo las interrupciones de fuentes habilitadas serán derivadas por el **Distribuidor** a un core cuando pasan a estado **Pending**.
- La seguridad de la interrupción. Esto es, si el handler va a ejecutar en modo Seguro o Normal. ????????
-



Inicialización

- El GIC está deshabilitado luego del reset. Debe ser inicializado antes de comenzar a aceptar requerimientos de interrupción.
- El **Distribuidor** y la **Interfaz con la CPU** se habilitan por separado.
- Se inicializan todas las características recién descriptas para el **Distribuidor** y se lo habilita a través de su Registro de control.
- Para cada **Interfaz con la CPU** se programan la máscara de prioridades y los ajustes de preemption.
- Cada **Interfaz con la CPU** debe ser habilitada por su core.
- Esto deja al GIC listo para enviar requerimientos de interrupción.
- Antes de aceptar interrupciones el core debe inicializar el vector de interrupciones apuntando a los handlers previstos para cada interrupción ARM.
- Finalmente en el **CPSR** de cada core, se debe habilitar las interrupciones.



Inicialización

Opciones de Habilitación/Deshabilitación

Deshabilitar el ***Distribuidor***, deshabilita el sistema de interrupciones completo para la totalidad de los cores.

Deshabilitar el bloque de ***Interfaz con la CPU*** equivale a deshabilitar las interrupciones de un core individual seteando el bit **CPSR.I**.

También pueden habilitarse o deshabilitarse, en forma individual las interrupciones en el ***Distribuidor***.



Procesamiento de una interrupción en el Software

- 1 El core que recibe la interrupción vectoriza al handler de alto nivel (salta a la dirección de la tabla de vectores de Interrupción).
- 2 Lee el *Interrupt Acknowledge Register* de la **Interfaz con la CPU**, y obtiene de allí el **Interrupt ID**.
- 3 Al detectar esta lectura el **Distribuidor** setea la fuente de interrupción en estado **Active**.
- 4 Con el **Interrupt ID** el handler de alto nivel determina cual es el handler específico de la interrupción, y lo puede invocar.
- 5 Cuando el handler específico termina su ejecución retorna al handler de alto nivel que escribe en el **End Of Interrupt Register** de la **Interfaz con la CPU**.
- 6 Si el estado estaba **Active**, lo cambia a **Inactive**, y si estaba **Active and Pending** lo cambia a **Pending**

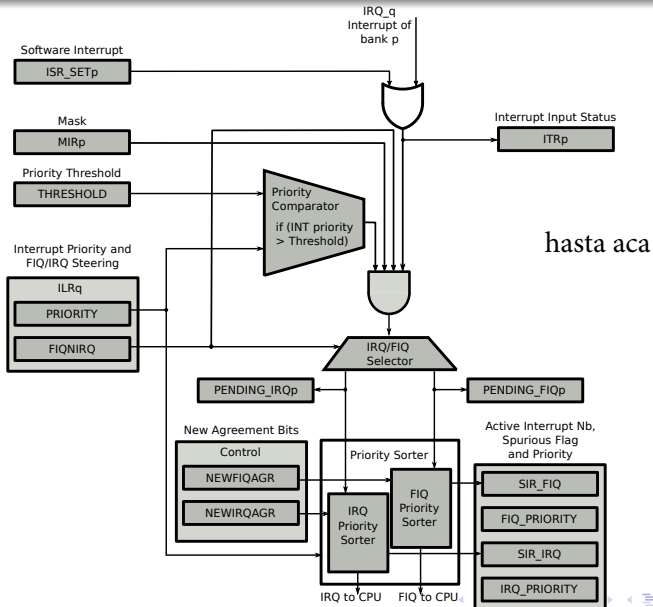


Contenido

- 1 Controlador de Interrupciones
- 2 Interrupciones de Hardware
 - Características y Modos de trabajo
 - Manejo de Múltiples Interrupciones
- 3 GIC Architecture
 - Generic Interrupt Controller
- 4 Implementación: SITARA 3358. Cortex-A8
 - Descripción funcional
 - Procesamiento de Interrupciones
 - Modelo de Programación Básico
- 5 ANEXO
 - Anticipos de System Programming



Sitara 335x Interrupt Controller - Diagrama Funcional



Contenido

- 1 Controlador de Interrupciones
- 2 Interrupciones de Hardware
 - Características y Modos de trabajo
 - Manejo de Múltiples Interrupciones
- 3 GIC Architecture
 - Generic Interrupt Controller
- 4 Implementación: SITARA 3358. Cortex-A8
 - Descripción funcional
 - **Procesamiento de Interrupciones**
 - Modelo de Programación Básico
- 5 ANEXO
 - Anticipos de System Programming



Generalidades

- El controlador interrupciones del 335x acepta hasta 128 entradas de interrupciones de hardware. Solo activas por nivel. Cada dispositivo debe activar la entrada y sostenerla hasta que el handler la desactive por software.
- Los registros de control y estado que dispone el controlador se mapean en memoria ocupando un espacio de 4 KiB, a partir de la Dirección **Base 0x48200000**, hasta **0x48200FFF**.
- En general los registros que llevan en su nombre la **n** como sufijo, cumplen las siguientes cuestiones de forma generales:
 - **n** es un número de 0 a 3. Es decir se trata siempre de 4 registros con sufijo 0, 1, 2, o 3.
 - Son un Bitmap de las 128 fuentes de interrupción que maneja el controlador. Bit 0 del registro terminado en 0, corresponde al **Interrupt ID 0**. Bit 31 del registro finalizado en 3, al **Interrupt ID 127**.



Selección de entradas

- Para evaluar la/s fuente/s de interrupción activas y activarla/desactivarla/s se dispone de dos juegos de 4 registros en el controlador:

INTC_ISR_SETn: Mapeados en **Base** + [0x90, 0xB0, 0xD0, 0xF0]. Lectura: indica el estado de cada fuente de interrupción: '1': Activa, '0': Inactiva. Escritura: aquellos bits que reciben '1' activan la generación de una interrupción asociada a ese **Interrupt ID** (Software Interrupt).

INTC_ISR_CLEARn: Mapeados en **Base** + [0x94, 0xB4, 0xD4, 0xF4]. Su lectura es siempre 0. El handler desactiva la interrupción correspondiente escribiendo '0' en el bit correspondiente.



Máscaras individuales

- **INTC_MIRn** mapeados en **Base+ [0x84, 0xA4, 0xC4, 0xE4]**. Muestran la máscara e interrupciones y se puede cambiar seteando o limpiando bit a bit.
- **INTC_MIR_CLEARn** mapeados en **Base+ [0x88, 0xA8, 0xC8, 0xE8]**. Setear un bit particular deshabilita la interrupción correspondiente.
- **INTC_MIR_SETn** mapeados en **Base+ [0x8C, 0xAC, 0xCC, 0xEC]**. Setear un bit particular habilita la interrupción correspondiente.
- Los últimos dos registros parecen redundantes con el **INTC_MIRn**. Se introducen para poder setear o limpiar máscaras de manera mas directa, aunque se mantenga el **INTC_MIRn** como legacy. Nada mas.



Generación hacia la CPU

- La respuesta del Controlador de interrupciones a una fuente de interrupción no enmascarable deriva en dos tipos de requerimiento de interrupciones al procesador:

IRQ Interrupciones de Baja prioridad

FIQ Interrupciones rápidas (No disponibles en dispositivos de propósito general)

- Hay 128 Registros de 32 bits (**INTC_ILR_0** a **INTC_ILR_127**), uno por cada fuente de interrupción.
- En el bit 0 se establece el tipo de interrupción para la fuente individual: 0 (default) establece que se envía por **IRQ**, y 1 por **FIQ**.
- En el caso de los dispositivos de Propósito general se envía por **IRQ** siempre, y este bit es **Reserved**.
- El campo de bits [7-2] de cada registro se denomina **Priority**, y establece la prioridad de cada dispositivo. Este campo se matchea con el **PRIORITYTHRESHOLD** del registro **INTC_THRESHOLD**.

Chequeo de activaciones en estado intermedios

- **INTC_ITRn** muestran el estado actual (o raw) de las fuentes de interrupción antes de ser enmascaradas.
Mapeados en **Base+ [0x80, 0xA0, 0xC0, 0xE0]**.
- **INTC_PENDING_IRQn** e **INTC_PENDING_FIQn** muestran el estado actual (o raw) de las fuentes de interrupción luego ser definido el tipo de interrupción (de acuerdo con esto depende a que registro consultar), y de la gestión de máscaras, pero antes de establecerse las prioridades.
Mapeados respectivamente en **Base+ [0x98, 0xB8, 0xD8, 0xF8]** y **Base+ [0x9C, 0xBC, 0xDC, 0xFC]**.



Manejo de Prioridades

- El campo de bits [7-0] del registro **INTC_THRESHOLD**, mapeado en **Base+0x68**, se denomina **PRIORITYTHRESHOLD**, y permite definir el umbral de prioridades.
- Acepta valores entre **0x00** (máxima) y **0x7F** (mínima). El valor **0xFF** (valor luego del reset) deshabilita el uso umbral. El umbral de prioridad **0x00**, se trata como **0x01**.
- Los bits **INTC_THRESHOLD[31-8]** está **Reserved**.
- El valor programado en el intervalo útil, establece un criterio para preemption por parte de interrupciones de alta prioridad.
- Al recibir un requerimiento de interrupción de una fuente, el controlador evalúa:

$$\text{INTC_ILRn.Priority} > \text{INTC_THRESHOLD.PRIORITYTHRESHOLD}$$
 Si es **TRUE** todas las fuentes de interrupción de menor o igual prioridad a la del umbral quedan enmascaradas.



Ordenamiento de Prioridades

- Si dos fuentes de interrupción de la misma prioridad y tipo envían requerimiento en forma simultánea, el controlador atenderá primero a la de mayor **Interrupt ID**.
- Pueden procesarse y enviarse **IRQ** y **FIQ** en forma simultánea.
- Se setean los bits correspondientes a todas las interrupciones recibidas en los registros **INTC_PENDING_IRQn** e **INTC_PENDING_FIQn**.
- En caso de simultaneidad se resuelve con el criterio ya explicado y una vez determinada la interrupción a enviar se setea el campo **ACTIVEIRQ** del registro **INTC_SIR_IRQ[6:0]**, o **ACTIVEFIQ** del registro **INTC_SIR_FIQ[6:0]**, o ambos simultáneamente en caso que se hayan priorizado requerimientos mapeados con ambos tipos.



Ordenamiento de Prioridades

- Estos bits permanecen activos hasta que se setea alguno de los bits `INTC_CONTROL.NEWIRQAGR` o `INTC_CONTROL.NEWFIQAGR` desde el handler indicando que la interrupción fue atendida.
- Cuando se setea cualquiera de los bits mencionados anteriormente, si permanecen requerimientos pendientes en cualquiera de los registros `INTC_PENDING_IRQn` y/o `INTC_PENDING_FIQn`, se dispara el ordenamiento de prioridades y se envía la señal **FIQ**, o **IRQ** a la CPU. Si en cambio no hay bits de requerimientos pendientes activos se desactiva la línea **IRQ** o **FIQ** según corresponda.



Protección del espacio de memoria de registros

- Cuando los requerimientos del sistema definen implementar multitasking en dos niveles de privilegio, es conveniente proteger el área de memoria donde están mapeados los registros, asegurando que las tareas no puedan acceder allí.
- Es de practica que en un sistema multitasking la gestión de interrupciones desde el punto de vista del software la maneje el sistema operativo.
- Para ello podemos setear el bit 0 del registro **INTC_PROTECTION** denominado **PROTECTION**, y el acceso a los registros del controlador de interrupciones queda restringido al modo supervisor.



Power saving

- El SOC SITARA 3358 tiene tres dominios de clock.

Interface clock

Functional clock

Synchronizer clock

- El Controlador de Interrupciones proporciona modos de trabajo Idle en cada dominio con el objetivo de no consumir energía cuando no se está procesando ninguna interrupción
- Hay dos registros que contienen los tres bits (uno por dominio) que permiten setear el estado Idle en cada caso.

INTC_SYSCONFIG.AUTOIDLE Es el bit 0.

INTC_IDLE.FUNCIDLE Es el bit 0.

INTC_IDLE.TURBO Es el bit 1.



Power saving



Power saving

INTC_SYSCONFIG.AUTOIDLE Si no hay actividad en el bus, un '1' en este bit pone el módulo en Idle desconectando el *interface clock*. Se reasume por interrupción sin delay. Luego de un reset este modo está deshabilitado.



Power saving

INTC_SYSCONFIG.AUTOIDLE Si no hay actividad en el bus, un '1' en este bit pone el módulo en Idle desconectando el *interface clock*. Se reasume por interrupción sin delay. Luego de un reset este modo está deshabilitado.

INTC_IDLE.FUNCIDLE Se activa con un '0'. Cuando esto ocurre en caso que no haya **IRQ**, o **FIQ** en curso, y no haya tampoco interrupciones pendientes, se deshabilita el *functional clock* del módulo poniéndolo en bajo consumo. Se reasume cuando ingresa un interrupción con un ciclo de latency. Luego del reset está activo.



Power saving

INTC_SYSCONFIG.AUTOIDLE Si no hay actividad en el bus, un '1' en este bit pone el módulo en Idle desconectando el *interface clock*. Se reasume por interrupción sin delay. Luego de un reset este modo está deshabilitado.

INTC_IDLE.FUNCIDLE Se activa con un '0'. Cuando esto ocurre en caso que no haya **IRQ**, o **FIQ** en curso, y no haya tampoco interrupciones pendientes, se deshabilita el *functional clock* del módulo poniéndolo en bajo consumo. Se reasume cuando ingresa un interrupción con un ciclo de latency. Luego del reset está activo.

INTC_IDLE.TURBO El *sincronizer clock* permite a las interrupciones externas resincronizarse antes de ser enmascaradas. El modo de bajo consumo se activa con '1' en este bit. Si bien baja el consumo cuando se recupera agrega entre 4 y 6 ciclos de clock a la atención de **IRQ** y **FIQ** para procesarlas.



Power saving



Power saving

- Una interrupción de tipo **IRQ** o **FIQ** demanda para su atención 4 ciclos de clock ± 1 ciclo.



Power saving

- Una interrupción de tipo **IRQ** o **FIQ** demanda para su atención 4 ciclos de clock ± 1 ciclo.
- En caso de estar activo el bit **INTC_IDLE.TURBO**, se extiende a 6 ciclos de clock, pero permite ahorrar energía.



Power saving

- Una interrupción de tipo **IRQ** o **FIQ** demanda para su atención 4 ciclos de clock ± 1 ciclo.
- En caso de estar activo el bit **INTC_IDLE.TURBO**, se extiende a 6 ciclos de clock, pero permite ahorrar energía.
- Se podría desactivar el bit **INTC_SYSCONFIG.AUTOIDLE**, y ahorrar un ciclo de clock, pero el costo es energético y no compensa dicha ganancia.



Power saving

- Una interrupción de tipo **IRQ** o **FIQ** demanda para su atención 4 ciclos de clock ± 1 ciclo.
- En caso de estar activo el bit **INTC_IDLE.TURBO**, se extiende a 6 ciclos de clock, pero permite ahorrar energía.
- Se podría desactivar el bit **INTC_SYSCONFIG.AUTOIDLE**, y ahorrar un ciclo de clock, pero el costo es energético y no compensa dicha ganancia.
- Si durante el proceso de ordenamiento de prioridades el controlador lee **INTC_SIR_IRQ** o **INTC_SIR_FIQ**, la lectura sufre un “stall” hasta que el proceso de ordenamiento de prioridades finalice y se actualicen los registros. En general, sin embargo, el delay entre la generación de la interrupción y la resolución de la prioridad nunca es tan alto como para que estas lecturas se requieran antes de su finalización.



Contenido

- 1 Controlador de Interrupciones
- 2 Interrupciones de Hardware
 - Características y Modos de trabajo
 - Manejo de Múltiples Interrupciones
- 3 GIC Architecture
 - Generic Interrupt Controller
- 4 Implementación: SITARA 3358. Cortex-A8
 - Descripción funcional
 - Procesamiento de Interrupciones
 - Modelo de Programación Básico
- 5 ANEXO
 - Anticipos de System Programming



Secuencia de Inicialización



Secuencia de Inicialización

- Si es necesario, activar el bit **INTC_SYSCONFIG.AUTOIDLE** para habilitar el autogating del interface clock.



Secuencia de Inicialización

- Si es necesario, activar el bit **INTC_SYSCONFIG.AUTOIDLE** para habilitar el autogating del interface clock.
- Si es necesario, activar el bit **INTC_IDLE.TURBO** para habilitar el autogating del sincronizer clock.



Secuencia de Inicialización

- Si es necesario, activar el bit **INTC_SYSCONFIG.AUTOIDLE** para habilitar el autogating del interface clock.
- Si es necesario, activar el bit **INTC_IDLE.TURBO** para habilitar el autogating del sincronizer clock.
- Si es necesario, limpiar el bit **INTC_IDLE.FUNCIDLE** para deshabilitar el autogating del functional clock.



Secuencia de Inicialización

- Si es necesario, activar el bit **INTC_SYSCONFIG.AUTOIDLE** para habilitar el autogating del interface clock.
- Si es necesario, activar el bit **INTC_IDLE.TURBO** para habilitar el autogating del sincronizer clock.
- Si es necesario, limpiar el bit **INTC_IDLE.FUNCIDLE** para deshabilitar el autogating del functional clock.
- Programar en cada registro **INTC_ILRn** la prioridad de cada interrupción y su tipo. Por default todas la interrupciones se mapean **IRQ** y tienen prioridad 00H, es decir máxima.



Secuencia de Inicialización

- Si es necesario, activar el bit **INTC_SYSCONFIG.AUTOIDLE** para habilitar el autogating del interface clock.
- Si es necesario, activar el bit **INTC_IDLE.TURBO** para habilitar el autogating del sincronizer clock.
- Si es necesario, limpiar el bit **INTC_IDLE.FUNCIDLE** para deshabilitar el autogating del functional clock.
- Programar en cada registro **INTC_ILRn** la prioridad de cada interrupción y su tipo. Por default todas las interrupciones se mapean **IRQ** y tienen prioridad 00H, es decir máxima.
- Habilitar las interrupciones programando los bits en cada registro **INTC_MIRn** (por default todas las fuentes de interrupción están enmascaradas). Si no hay restricciones de compatibilidad se pueden utilizar los registros **INTC_MIR_SETn** o **INTC_MIR_CLEARn**.



Procesamiento de una Interrupción



Procesamiento de una Interrupción

- Luego de inicializar los registros **INTC_ILRn** y **INTC_MIRn**, con los datos de operación se comienza a recibir interrupciones.



Procesamiento de una Interrupción

- Luego de inicializar los registros **INTC_ILRn** y **INTC_MIRn**, con los datos de operación se comienza a recibir interrupciones.
- El procedimiento es el mismo para **IRQ** como para **FIQ**.



Procesamiento de una Interrupción

- Luego de inicializar los registros **INTC_ILRn** y **INTC_MIRn**, con los datos de operación se comienza a recibir interrupciones.
- El procedimiento es el mismo para **IRQ** como para **FIQ**.
- 1 Ingresan una o más interrupciones (señales **M_IRQ_n**) y tanto **IRQ** como **FIQ** están inactivas.



Procesamiento de una Interrupción

- Luego de inicializar los registros **INTC_ILRn** y **INTC_MIRn**, con los datos de operación se comienza a recibir interrupciones.
 - El procedimiento es el mismo para **IRQ** como para **FIQ**.
- 1 Ingresa una o mas interrupciones (señales **M_IRQ_n**) y tanto **IRQ** como **FIQ** están inactivas.
 - 2 Se evalúa el bit **INTC_ILRm.FIQNIRO**. Si es '0' se activa la señal **INTC_IRQ**. Si está en '1', se activa la señal **INTC_FIQ**.



Procesamiento de una Interrupción

- Luego de inicializar los registros **INTC_ILRn** y **INTC_MIRn**, con los datos de operación se comienza a recibir interrupciones.
 - El procedimiento es el mismo para **IRQ** como para **FIQ**.
- 1 Ingresa una o mas interrupciones (señales **M_IRQ_n**) y tanto **IRQ** como **FIQ** están inactivas.
 - 2 Se evalúa el bit **INTC_ILRm.FIQNIRO**. Si es '0' se activa la señal **INTC_IRQ**. Si está en '1', se activa la señal **INTC_FIQ**.
 - 3 El controlador de Interrupciones activa el mecanismo de ordenamiento de prioridades, y de acuerdo con la línea de interrupción que fue activada escribe el **Interrupt ID** en el campo de bits **INTC_SIR_IRQ[6:0]** o **INTC_SIR_FIQ[6:0]**, para indicar la interrupción que está activa.



Procesamiento de una Interrupción

- 4 El core salva el contexto básico y llama al ISR de acuerdo con:



Procesamiento de una Interrupción

- 4 El core salva el contexto básico y llama al ISR de acuerdo con:

```
LR = PC + 4 /*return link */
SPSR = CPSR /*Save CPSR before execution */
CPSR[5] = 0 /*Execute in ARM state */
CPSR[7] = 1 /*Disable IRQ */
CPSR[8] = 1 /*Disable Imprecise Data Aborts */
CPSR[9] = CP15_reg1_EEbit /*Endianness on exception entry */
if interrupt == IRQ then
    CPSR[4:0] = 0b10010 /*Enter IRQ mode */
if high vectors configured then
    PC = 0xFFFF0018
else
    PC = 0x00000018 /*execute interrupt vector */
else if interrupt == FIQ then
    CPSR[4:0] = 0b10001 /*Enter FIQ mode */
    CPSR[6] = 1 /*Disable FIQ */
if high vectors configured then
    PC = 0xFFFF001C
else
    PC = 0x0000001C /*execute interrupt vector */
endif
```


Procesamiento de una Interrupción

- 5 El ISR salva el contexto restante, identifica el **Interrpt ID** en `INTC_SIR_IRQ[6:0]` o `INTC_SIR_FIQ[6:0]`, y llama al handler.



Procesamiento de una Interrupción

- 5 El ISR salva el contexto restante, identifica el **Interrupt ID** en **INTC_SIR_IRQ[6:0]** o **INTC_SIR_FIQ[6:0]**, y llama al handler.

```

/* INTC_SIR_IRQ/INTC_SIR_FIQ register address */
INTC_SIR_IRQ_ADDR .word 0x48200040
/* Si es FIQ: INTC_SIR_FIQ_ADDR 0x48200044.*/
/* ACTIVEIRQ Máscara para obtener Interrupt ID.*/
ACTIVEIRQ_MASK .equ 0x7F
_IRQ_ISR: /* o bien _FIQ_ISR, si es FIQ.*/
    STMFDP SP!, {R0-R12, LR} /* Salva GP Registers y Link register.*/
    MRS R11, SPSR /* Resguarda SPSR en R11.*/
    /* Obtiene el nivel de prioridad mas alto activo IRQ/FIQ.*/
    LDR R10, INTC_SIR_IRQ_ADDR /* o INTC_SIR_FIQ_ADDR si es FIQ.*/
    LDR R10, [R10] /* Lee registro INTC_SIR_IRQ/INTC_SIR_FIQ.*/
    AND R10, R10, #ACTIVEIRQ_MASK /* Máscara para R10=Interrupt ID.*/
    /* Salta al handler de interrupción específico.*/
    LDR PC, [PC, R10, LSL #2] /* PC apunta a instrucción + 8.*/
    NOP /* Indexa la tabla con el PC.*/
    /* Tabla de direcciones de los handlers.*/
    .word IRQ0handler /* For IRQ0 of BANK0.*/
    .word IRQ1handler
    .word IRQ2handler

```

Procesamiento de una Interrupción

- 6 La subrutina del handler ejecuta el código específico del periférico que está interrumpiendo, maneja el evento, y desactiva la condición de interrupción del lado del dispositivo.



Procesamiento de una Interrupción

- 6 La subrutina del handler ejecuta el código específico del periférico que está interrumpiendo, maneja el evento, y desactiva la condición de interrupción del lado del dispositivo.

```
/* Subrutina para IRQ0 */
IRQ0handler:
    /* Salva registros de trabajo. */
    STMFD SP!, {R0-R1}
    /* Accede al registro de estados del módulo periférico. */
    /* para inactivar la señal de interrupción M_IRQ_0. */
    MOV R0, #0x7 /* Máscara para los 3 flags. */
    /* R1 apunta al Status Register del Módulo. */
    LDR R1, MODULE0.STATUS_REG_ADDR
    /* Inactiva la interrupción del periférico. */
    STR R0, [R1] /* Limpia los 3 flags. */
    /* Recupera los registros de trabajo. */
    LDMFD SP!, {R0-R1}
    /* Salta a la parte final del ISR. */
    B IRQ_ISR_end
```



Procesamiento de una Interrupción

- 7 El ISR setea el bit correspondiente (`INTC_CONTROL.NEWIRQAGR` o `INTC_CONTROL.NEWFIQAGR`) al retornar de la subrutina, rehabilitando el procesamiento de requerimientos `IRQ` o `FIQ` pendientes, y recupera el contexto ARM en el código subsiguientes. Debido a que las escrituras se efectúan en un bus Interconectado, es necesario asegurar que las escrituras precedentes se realicen antes de habilitar `IRQ` o `FIQ`, se utilizan Data Synchronization Barrier¹. Esta operación asegura que la línea `IRQ` o `FIQ` se inactive antes de habilitar las interrupciones `IRQ` o `FIQ`. Luego de ésto el Controlador procesa cualquier otra interrupción pendiente o si no las hay simplemente desactiva la línea `IRQ` o `FIQ`.

¹Ver explicación en el Apéndice “Anticipos de System Programming”



Procesamiento de una Interrupción

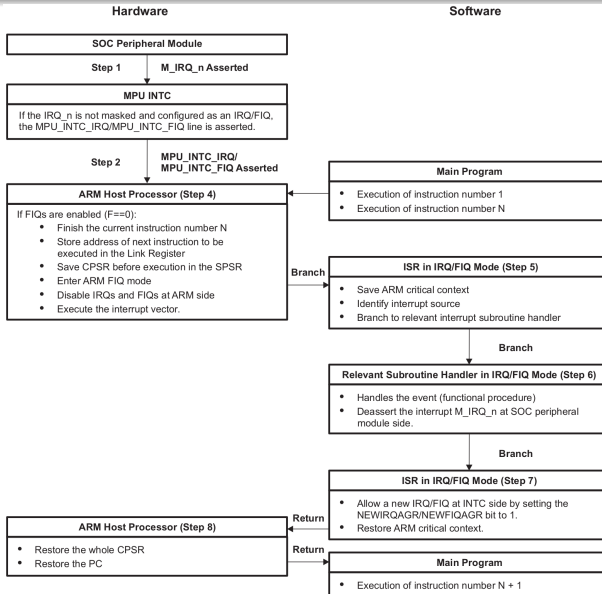
```

/* Dirección del registro INTC_CONTROL.*/
INTC_CONTROL_ADDR .word 0x48200048
/*Máscara para setear solo el bit NEWIRQAGR o NEWFIQAGR.*/
NEWIRQAGR_MASK .equ 0x01 /*Para FIQ: NEWFIQAGR_MASK .equ 0x02*/
IRQ_ISR_end: /*Para FIQ_ISR_end: */
/*Habilita nuevas IRQs/FIQs en el controlador.*/
/*No hay write back de bits en INTC_CONTROL (es write only)*/
MOV R0, #NEWIRQAGR_MASK /*R0 tiene la posición del bit NEWIRQAGR*/
/*Para FIQ, MOV R0, #NEWFIQAGR_MASK.*/
LDR R1, INTC_CONTROL_ADDR /* R1 Apunta al INTC_CONTROL*/
STR R0, [R1] /*NEWIRQAGR=1 o NEWFIQAGR=1. Habilita IRQs/FIQ.*/
/*Data Synchronization Barrier. */
MOV R0, #0
MCR P15, #0, R0, C7, C10, #4 /*Ver ''Anticipos de System
                                Programming''*/
/*Recupera el contexto critico.*/
MSR SPSR, R11 /*Recupera SPSR desde R11.*/
LDMFD SP!, {R0-R12, LR} /*Recupera Registros y Link Register.*/
/*Retorna luego de manejar la interrupción.*/
SUBS PC, LR, #4
/*Al retornar del ISR, automáticamente recupera el contexto:*/
CPSR = SPSR
PC = LR

```



Procesamiento de una Interrupción



Procesamiento Preemptive de una Interrupción



Procesamiento Preemptive de una Interrupción

- El objetivo es obtener un sistema de interrupciones anidado que permita disminuir la demora en atención (latency) para las interrupciones más prioritarias.



Procesamiento Preemptive de una Interrupción

- El objetivo es obtener un sistema de interrupciones anidado que permita disminuir la demora en atención (latency) para las interrupciones más prioritarias.
- Concepto: Un **ISR Preemptive** puede ser interrumpido por otro de mayor prioridad.



Procesamiento Preemptive de una Interrupción

- El objetivo es obtener un sistema de interrupciones anidado que permita disminuir la demora en atención (latency) para las interrupciones más prioritarias.
- Concepto: Un **ISR Preemptive** puede ser interrumpido por otro de mayor prioridad.
- Riesgos: Corrupción de datos. Queda el control a cargo del programador de Sistemas.



Procesamiento Preemptive de una Interrupción

- El objetivo es obtener un sistema de interrupciones anidado que permita disminuir la demora en atención (latency) para las interrupciones más prioritarias.
- Concepto: Un **ISR Preemptive** puede ser interrumpido por otro de mayor prioridad.
- Riesgos: Corrupción de datos. Queda el control a cargo del programador de Sistemas.
- Queda a cargo del programador resguardar todos los registros que puedan corromperse al procesar la interrupción anidada, lo mismo que habilitar **IRQ** o **FIQ**.



Procesamiento Preemptive de una Interrupción

- El objetivo es obtener un sistema de interrupciones anidado que permita disminuir la demora en atención (latency) para las interrupciones más prioritarias.
- Concepto: Un **ISR Preemptive** puede ser interrumpido por otro de mayor prioridad.
- Riesgos: Corrupción de datos. Queda el control a cargo del programador de Sistemas.
- Queda a cargo del programador resguardar todos los registros que puedan corromperse al procesar la interrupción anidada, lo mismo que habilitar **IRQ** o **FIQ**.
- No hay diferencias de fondo en el procesamiento de **IRQ** o **FIQ**, excepto en el nombre de los flags o registros de cada tipo. Pero son los mismos pasos.



Procesamiento Preemptive de una Interrupción

- El objetivo es obtener un sistema de interrupciones anidado que permita disminuir la demora en atención (latency) para las interrupciones más prioritarias.
- Concepto: Un **ISR Preemptive** puede ser interrumpido por otro de mayor prioridad.
- Riesgos: Corrupción de datos. Queda el control a cargo del programador de Sistemas.
- Queda a cargo del programador resguardar todos los registros que puedan corromperse al procesar la interrupción anidada, lo mismo que habilitar **IRQ** o **FIQ**.
- No hay diferencias de fondo en el procesamiento de **IRQ** o **FIQ**, excepto en el nombre de los flags o registros de cada tipo. Pero son los mismos pasos.
- A continuación la secuencia básica de un **ISR Preemptive**.



Procesamiento Preemptive de una Interrupción



Procesamiento Preemptive de una Interrupción

- 1 Se salvan los registros críticos del procesador.



Procesamiento Preemptive de una Interrupción

- 1 Se salvan los registros críticos del procesador.
- 2 Resguardar el campo de bits **INTC_THRESHOLD** [7 : 0], denominado **PRIORITYTHRESHOLD** antes de modificarlo.



Procesamiento Preemptive de una Interrupción

- 1 Se salvan los registros críticos del procesador.
- 2 Resguardar el campo de bits **INTC_THRESHOLD** [7 : 0], denominado **PRIORITYTHRESHOLD** antes de modificarlo.
- 3 Leer del campo de bits **INTC_IRQ_PRIORITY.IRQPRIORITY** o **INTC_FIQ_PRIORITY.FIQPRIORITY**, según corresponda, la prioridad de la interrupción en curso, y escribirlo en el campo de bits **INTC_THRESHOLD** [7 : 0].



Procesamiento Preemptive de una Interrupción

- 1 Se salvan los registros críticos del procesador.
- 2 Resguardar el campo de bits **INTC_THRESHOLD** [7 : 0], denominado **PRIORITYTHRESHOLD** antes de modificarlo.
- 3 Leer del campo de bits **INTC_IRQ_PRIORITY**.**IRQPRIORITY** o **INTC_FIQ_PRIORITY**.**FIQPRIORITY**, según corresponda, la prioridad de la interrupción en curso, y escribirlo en el campo de bits **INTC_THRESHOLD** [7 : 0].
- 4 Lee, según corresponda, el campo de bits **INTC_SIR_IRQ** [6 : 0] denominado como vimos **ACTIVEIRQ**, o bien **INTC_SIR_FIQ** [6 : 0] denominado **ACTIVEFIQ**, para identificar la interrupción en curso.



Procesamiento Preemptive de una Interrupción

- 5 Escribe '1', según corresponda, en `INTC_CONTROL.NEWIRQAGR` o en `INTC_CONTROL.NEWFIQAGR`, para asegurar que el controlador solo envíe a la CPU interrupciones de dispositivos de mayor prioridad.



Procesamiento Preemptive de una Interrupción

- 5 Escribe '1', según corresponda, en `INTC_CONTROL.NEWIRQAGR` o en `INTC_CONTROL.NEWFIQAGR`, para asegurar que el controlador solo envíe a la CPU interrupciones de dispositivos de mayor prioridad.
- 6 Como las escrituras se cursan por un bus interconectado, para asegurar que se completen las escrituras precedentes antes de habilitar `IRQ` o `FIQ`, se debe emplear un Data Synchronization Barrier¹. Esta operación asegura que se desactive cualquier interrupción antes de habilitar nuevamente `IRQ` a `FIQ`.

¹Ver explicación en el Apéndice “Anticipos de System Programming”



Procesamiento Preemptive de una Interrupción

- 5 Escribe '1', según corresponda, en **INTC_CONTROL.NEWIRQAGR** o en **INTC_CONTROL.NEWFIQAGR**, para asegurar que el controlador solo envíe a la CPU interrupciones de dispositivos de mayor prioridad.
- 6 Como las escrituras se cursan por un bus interconectado, para asegurar que se completen las escrituras precedentes antes de habilitar **IRQ** o **FIQ**, se debe emplear un Data Synchronization Barrier¹. Esta operación asegura que se desactive cualquier interrupción antes de habilitar nuevamente **IRQ** a **FIQ**.
- 7 Habilitar **IRQ** o **FIQ**.

¹Ver explicación en el Apéndice “Anticipos de System Programming”



Procesamiento Preemptive de una Interrupción

- 5 Escribe '1', según corresponda, en `INTC_CONTROL.NEWIRQAGR` o en `INTC_CONTROL.NEWFIQAGR`, para asegurar que el controlador solo envíe a la CPU interrupciones de dispositivos de mayor prioridad.
- 6 Como las escrituras se cursan por un bus interconectado, para asegurar que se completen las escrituras precedentes antes de habilitar `IRQ` o `FIQ`, se debe emplear un Data Synchronization Barrier¹. Esta operación asegura que se desactive cualquier interrupción antes de habilitar nuevamente `IRQ` a `FIQ`.
- 7 Habilitar `IRQ` o `FIQ`.
- 8 Saltar a la subrutina relevante del handler.

¹Ver explicación en el Apéndice “Anticipos de System Programming”



Procesamiento Preemptive de una Interrupción

```
/*Máscara para obtener solo el bit field.*/  
ACTIVEPRIO_MASK .equ 0x7F  
_IRQ_ISR:  
/*Paso 1 : Salvar el contexto crítico.*/  
STMFD SP!, {R0-R12, LR} /*Resguarda los registros de trabajo.*/  
MRS R11, SPSR /*Resguarda SPSR en R11.*/  
/*Paso 2 : Salvar el registro INTC_THRESHOLD en R12*/  
LDR R0, INTC_THRESHOLD_ADDR  
LDR R12, [R0]  
/*(1) El mecanismo de umbral de prioridad se habilita de manera  
automática cuando se escribe un valor de prioridad en el rango de  
0x00 a 0x7F. Si se escribe el valor 0xFF (predeterminado luego del  
reset) se desactiva el mecanismo de umbral de prioridad. No se  
deben utilizar valores entre 0x3F y 0xFF. Cuando se utiliza umbral  
de prioridad por hardware, las prioridades de las interrupciones  
seleccionadas como FIQ o IRQ se entrelazan; caso contrario, son  
independientes. Cuando se entrelazan, todas las prioridades de  
FIQ deben establecerse más altas que todas las prioridades de IRQ  
para mantener la prioridad relativa de FIQ sobre IRQ.*/
```



Procesamiento Preemptive de una Interrupción

/*(2) Cuando se manejan FIQ utilizando el mecanismo de umbral de prioridad, los bits NEWFIQAGR y NEWIRQAGR deben escribirse al mismo tiempo para garantizar que se aplique el nuevo umbral de prioridad mientras se realiza una clasificación de IRQ. El core ARM no vió esta IRQ, ya que habrá sido enmascarada al ingresar al ISR de FIQ. Sin embargo, la fuente de la IRQ permanece activa y finalmente se procesa cuando el umbral de prioridad cae a una prioridad lo suficientemente baja como para permitir su procesamiento. Durante el ISR de IRQ no se requiere escribir un Reconocimiento de Nueva FIQ, ya que la clasificación de FIQ no se ve afectada (siempre que todas las prioridades de FIQ sean más altas que todas las prioridades de IRQ).*/

/*Paso 3: Obtener prioridad más alta de la IRQ activa.*/

LDR R1, INTC_IRQ_PRIORITY_ADDR /* o INTC_FIQ_PRIORITY_ADDR.*/

LDR R1, [R1] /* R1 = INTC_IRQ_PRIORITY o R1 = INTC_FIQ_PRIORITY.*/

AND R1, R1, #ACTIVEPRIO_MASK /* Aplica máscara y obtiene la prioridad de la interrupción.*/

STR R1, [R0] /* Escribe prioridad en el registro INTC_THRESHOLD*/

/*Paso 4: Obtener el número más alto de prioridad de IRQ activa.*/

LDR R10, INTC_SIR_IRQ_ADDR /* o INTC_SIR_FIQ_ADDR. */

LDR R10, [R10] /* Lee registro INTC_SIR_IRQ o INTC_SIR_FIQ.*/

AND R10, R10, #ACTIVEIRQ_MASK /* Aplica máscara y obtiene el número de la interrupción activa.*/



Procesamiento Preemptive de una Interrupción

```

/*Paso 5: Habilita nuevas IRQs y FIQs en el controlador.*/
MOV R0, #0x1 /* Obtiene posición del bit NEWIRQAGR.
               Para FIQ se usa MOV R0,#0x3 en busca de NEWFIQAGR .
               */
LDR R1, INTC_CONTROL_ADDR
STR R0, [R1] /* Escribe los bits NEWIRQAGR y NEWFIQAGR.*/
/*Paso 6: Data Synchronization Barrier.*/
MOV R0, #0
MCR P15, #0, R0, C7, C10, #4 /* Ver ‘‘Anticipos de System
                               Programming’’ */
/*Paso 7 : Lee-modifica-escribe el registro CPSR para habilitar
IRQs/FIQs en el core.*/
MRS R0, CPSR /* Lee status register.*/
BIC R0, R0, #0x80 /* Limpia bit I. Para bit F usamos /0x40.*/
MSR CPSR, R0 /* Escribe status register para habilitar IRQs.*/
/*Paso 8: Salta al handler de subrutina relevante...*/
LDR PC, [PC, R10, #8] /* Dirección base del PC apunta a ésta */
NOP /* instrucción + 8 para indexar la tabla
    de saltos con el PC.*/

/*Table of handler start addresses.*/
.word IRQ0handler /*IRQ0 BANK0*/
.word IRQ1handler
.word IRQ2handler

```



Procesamiento Preemptive de una Interrupción

Al retornar del handler relevante de **IRQ**, o **FIQ**



Procesamiento Preemptive de una Interrupción

Al retornar del handler relevante de **IRQ**, o **FIQ**

- 9 Deshabilita las interrupciones den el Core ARM.



Procesamiento Preemptive de una Interrupción

Al retornar del handler relevante de **IRQ**, o **FIQ**

- 9 Deshabilita las interrupciones den el Core ARM.
- 10 Recupera campo de bits **INTC_THRESHOLD . PRIORITYTHRESHOLD**.



Procesamiento Preemptive de una Interrupción

Al retornar del handler relevante de **IRQ**, o **FIQ**

- 9 Deshabilita las interrupciones den el Core ARM.
- 10 Recupera campo de bits **INTC_THRESHOLD.PRIORITYTHRESHOLD**.
- 11 Recupera contexto de registros críticos del core.



Procesamiento Preemptive de una Interrupción

Al retornar del handler relevante de **IRQ**, o **FIQ**

- 9 Deshabilita las interrupciones den el Core ARM.
- 10 Recupera campo de bits **INTC_THRESHOLD.PRIORITYTHRESHOLD**.
- 11 Recupera contexto de registros críticos del core.

```
IRQ_ISR_end:
```

```
/* Paso 1: Lee-modifica-escribe CPSR. Deshabilita IRQs/FIQs en el  
core ARM.*/
```

```
MRS R0, CPSR          /* Lee CPSR*/
```

```
ORR R0, R0, #0x80     /* Setea el bit I. Si es FIQ ORR R0, R0, #0x40 y  
setea bit F.*/
```

```
MSR CPSR, R0          /* Escribe el CPSR para deshabilitar las IRQs*/
```

```
/* Paso 2: Recupera el registro INTC_THRESHOLD desde R12*/
```

```
LDR R0, INTC_THRESHOLD_ADDR
```

```
STR R12, [R0]
```

```
/* Paso 3: Recupera el contexto de ejecución crítico.*/
```

```
MSR SPSR, R11          /* Recupera registro SPSR desde R11.*/
```

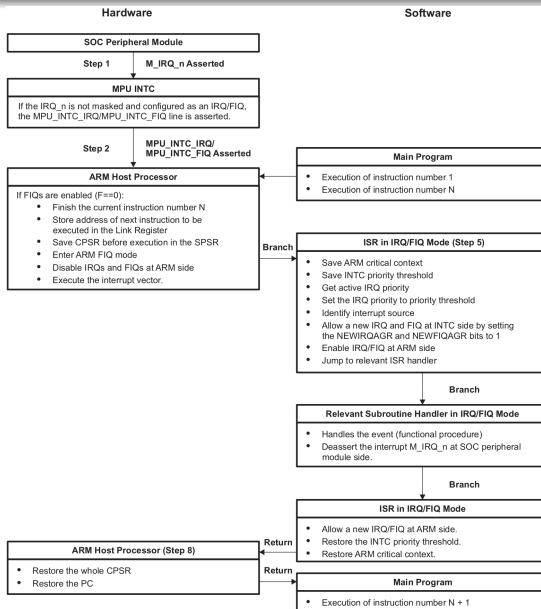
```
LDMFD SP!, {R0-R12, LR} /* Recupera registros de trabajo y Link  
register.*/
```

```
/* Retorna luego de manejar la interrupción.*/
```

```
SUBS PC, LR, #4
```



Procesamiento de una Interrupción Preemptive



Preemption de Interrupciones



Preemption de Interrupciones

- Si se activó preemption de interrupciones de alta prioridad sobre las de menor prioridad, cada ISR debe leer el valor de prioridad de la interrupción activa y escribirlo en el registro de Umbral de Prioridad.



Preemption de Interrupciones

- Si se activó preemption de interrupciones de alta prioridad sobre las de menor prioridad, cada ISR debe leer el valor de prioridad de la interrupción activa y escribirlo en el registro de Umbral de Prioridad.
- En cada nivel de preemption el programador debe salvar el valor del registro de Umbral de prioridades antes de pisarlo con el valor leído de la interrupción en curso, y restaurarlo antes de abandonar el ISR de ese nivel.



Preemption de Interrupciones

- Si se activó preemption de interrupciones de alta prioridad sobre las de menor prioridad, cada ISR debe leer el valor de prioridad de la interrupción activa y escribirlo en el registro de Umbral de Prioridad.
- En cada nivel de preemption el programador debe salvar el valor del registro de Umbral de prioridades antes de pisarlo con el valor leído de la interrupción en curso, y restaurarlo antes de abandonar el ISR de ese nivel.
- Para permitir a las interrupciones de mayor prioridad interrumpir el procesamiento de las de menor prioridad a partir de un punto del código de cada ISR, se debe escribir un '1' en los bits **NEW_IRQ_AGR**, o **NEW_FIQ_AGR** del registro **INTC_CONTROL**.



Preemption de Interrupciones

- El mecanismo de Umbral de Prioridad se habilita automáticamente al escribir en los registros **IRQ_PRIORITY** o **FIQ_PRIORITY**, un valor entre **0x00** y **0x7F**.



Preemption de Interrupciones

- El mecanismo de Umbral de Prioridad se habilita automáticamente al escribir en los registros **IRQ_PRIORITY** o **FIQ_PRIORITY**, un valor entre **0x00** y **0x7F**.
- Escribir en alguno de esos registros el valor **0xFF** (que por otra parte es el valor default en ambos registros luego de un reset), deshabilita el mecanismo de Umbral de Prioridad.



Preemption de Interrupciones

- El mecanismo de Umbral de Prioridad se habilita automáticamente al escribir en los registros **IRQ_PRIORITY** o **FIQ_PRIORITY**, un valor entre **0x00** y **0x7F**.
- Escribir en alguno de esos registros el valor **0xFF** (que por otra parte es el valor default en ambos registros luego de un reset), deshabilita el mecanismo de Umbral de Prioridad.
- Al establecer el uso del Umbral de Prioridad, se entrelazan las prioridades de interrupciones **IRQ** y **FIQ**. En caso contrario ambos tipos de interrupciones se mantienen independientes entre sí.



Preemption de Interrupciones

- El mecanismo de Umbral de Prioridad se habilita automáticamente al escribir en los registros **IRQ_PRIORITY** o **FIQ_PRIORITY**, un valor entre **0x00** y **0x7F**.
- Escribir en alguno de esos registros el valor **0xFF** (que por otra parte es el valor default en ambos registros luego de un reset), deshabilita el mecanismo de Umbral de Prioridad.
- Al establecer el uso del Umbral de Prioridad, se entrelazan las prioridades de interrupciones **IRQ** y **FIQ**. En caso contrario ambos tipos de interrupciones se mantienen independientes entre sí.
- En caso de usar Umbral de Prioridad, entonces, se requiere que los valores de prioridad de todas las **FIQ** sean mayores que los de todas las **IRQ**, de modo de mantener la prioridad relativa de **FIQ** sobre **IRQ**.



Preemption de Interrupciones

- Cuando se procesa una **FIQ** con manejo de Umbral de Prioridad, se requiere setear los bits **NEW_FIQ_AGR** y **NEW_IRQ_AGR** al mismo tiempo, para cubrir el riesgo que implicaría cambiar el Umbral de Prioridad mientras se está resolviendo el orden de prioridades de a **IRQ**.



Preemption de Interrupciones

- Cuando se procesa una **FIQ** con manejo de Umbral de Prioridad, se requiere setear los bits **NEW_FIQ_AGR** y **NEW_IRQ_AGR** al mismo tiempo, para cubrir el riesgo que implicaría cambiar el Umbral de Prioridad mientras se está resolviendo el orden de prioridades de a **IRQ**.
- Esta **IRQ** no habrá sido vista por el core ARM ya que habrá sido enmascarada al ingresar al ISR de la **FIQ**.



Preemption de Interrupciones

- Cuando se procesa una **FIQ** con manejo de Umbral de Prioridad, se requiere setear los bits **NEW_FIQ_AGR** y **NEW_IRQ_AGR** al mismo tiempo, para cubrir el riesgo que implicaría cambiar el Umbral de Prioridad mientras se está resolviendo el orden de prioridades de a **IRQ**.
- Esta **IRQ** no habrá sido vista por el core ARM ya que habrá sido enmascarada al ingresar al ISR de la **FIQ**.
- Sin embargo, la fuente de la **IRQ** permanecerá activa y finalmente se procesará cuando el Umbral de Prioridad caiga a una prioridad lo suficientemente baja.



Preemption de Interrupciones

- Cuando se procesa una **FIQ** con manejo de Umbral de Prioridad, se requiere setear los bits **NEW_FIQ_AGR** y **NEW_IRQ_AGR** al mismo tiempo, para cubrir el riesgo que implicaría cambiar el Umbral de Prioridad mientras se está resolviendo el orden de prioridades de a **IRQ**.
- Esta **IRQ** no habrá sido vista por el core ARM ya que habrá sido enmascarada al ingresar al ISR de la **FIQ**.
- Sin embargo, la fuente de la **IRQ** permanecerá activa y finalmente se procesará cuando el Umbral de Prioridad caiga a una prioridad lo suficientemente baja.
- Esta precaución es innecesaria en un ISR de una **IRQ**, ya que la clasificación de prioridades de una **FIQ** no se verá afectada (***siempre que se haya observado el requerimiento que todas las FIQ tengan prioridad mayor que todas las IRQ.***)



Manejo de Interrupciones Espurias



Manejo de Interrupciones Espurias

- Una interrupción espuria indica que el resultado de un ordenamiento de prioridades (tomado en una ventana de 10 ciclos de *functional clock*) es inválido.



Manejo de Interrupciones Espurias

- Una interrupción espuria indica que el resultado de un ordenamiento de prioridades (tomado en una ventana de 10 ciclos de *functional clock*) es inválido.
- Las posibles razones de un ordenamiento inválido pueden ser:



Manejo de Interrupciones Espurias

- Una interrupción espuria indica que el resultado de un ordenamiento de prioridades (tomado en una ventana de 10 ciclos de *functional clock*) es inválido.
- Las posibles razones de un ordenamiento inválido pueden ser:
 - La interrupción que disparó el ordenamiento dejó de estar válida durante el mismo.



Manejo de Interrupciones Espurias

- Una interrupción espuria indica que el resultado de un ordenamiento de prioridades (tomado en una ventana de 10 ciclos de *functional clock*) es inválido.
- Las posibles razones de un ordenamiento inválido pueden ser:
 - La interrupción que disparó el ordenamiento dejó de estar válida durante el mismo.
 - Durante el tiempo de ordenamiento de la prioridad se modificó una máscara que afectó su resultado.



Manejo de Interrupciones Espurias

- Una interrupción espuria indica que el resultado de un ordenamiento de prioridades (tomado en una ventana de 10 ciclos de *functional clock*) es inválido.
- Las posibles razones de un ordenamiento inválido pueden ser:
 - La interrupción que disparó el ordenamiento dejó de estar válida durante el mismo.
 - Durante el tiempo de ordenamiento de la prioridad se modificó una máscara que afectó su resultado.
- Por lo tanto, mientras esté activa una interrupción no deben modificarse los registros **INTC_MIRn**, **INTC_ILRn**, y **INTC_MIR_SETn**.



Manejo de Interrupciones Espurias

- Una interrupción espuria indica que el resultado de un ordenamiento de prioridades (tomado en una ventana de 10 ciclos de *functional clock*) es inválido.
- Las posibles razones de un ordenamiento inválido pueden ser:
 - La interrupción que disparó el ordenamiento dejó de estar válida durante el mismo.
 - Durante el tiempo de ordenamiento de la prioridad se modificó una máscara que afectó su resultado.
- Por lo tanto, mientras esté activa una interrupción no deben modificarse los registros **INTC_MIRn**, **INTC_ILRn**, y **INTC_MIR_SETn**.
- Solo la fuente de interrupción que disparó el ordenamiento de prioridad puede ser enmascarada con este proceso en curso.



Manejo de Interrupciones Espurias

- Si los siguientes registros cambian durante los 10 ciclos de *functional clock* luego de iniciado el ordenamiento de prioridad sus valores resultan inválidos.



Manejo de Interrupciones Espurias

- Si los siguientes registros cambian durante los 10 ciclos de *functional clock* luego de iniciado el ordenamiento de prioridad sus valores resultan inválidos.

INTC_SIR_IRQ



Manejo de Interrupciones Espurias

- Si los siguientes registros cambian durante los 10 ciclos de *functional clock* luego de iniciado el ordenamiento de prioridad sus valores resultan inválidos.

`INTC_SIR_IRQ`

`INTC_SIR_FIQ`



Manejo de Interrupciones Espurias

- Si los siguientes registros cambian durante los 10 ciclos de *functional clock* luego de iniciado el ordenamiento de prioridad sus valores resultan inválidos.

`INTC_SIR_IRQ`

`INTC_SIR_FIQ`

`INTC_IRQ_PRIORITY`



Manejo de Interrupciones Espurias

- Si los siguientes registros cambian durante los 10 ciclos de *functional clock* luego de iniciado el ordenamiento de prioridad sus valores resultan inválidos.

`INTC_SIR_IRQ`

`INTC_SIR_FIQ`

`INTC_IRQ_PRIORITY`

`INTC_FIQ_PRIORITY`



Manejo de Interrupciones Espurias

- Si los siguientes registros cambian durante los 10 ciclos de *functional clock* luego de iniciado el ordenamiento de prioridad sus valores resultan inválidos.

`INTC_SIR_IRQ`

`INTC_SIR_FIQ`

`INTC_IRQ_PRIORITY`

`INTC_FIQ_PRIORITY`

- Esta condición se detecta tanto en `IRQ` como `FIQ`, y se indica en los campos de bits de los registros `INTC_SIR_IRQ[31:7]` y `INTC_SIR_FIQ[31:7]`.



Manejo de Interrupciones Espurias

- Si los siguientes registros cambian durante los 10 ciclos de *functional clock* luego de iniciado el ordenamiento de prioridad sus valores resultan inválidos.

`INTC_SIR_IRQ`

`INTC_SIR_FIQ`

`INTC_IRQ_PRIORITY`

`INTC_FIQ_PRIORITY`

- Esta condición se detecta tanto en `IRQ` como `FIQ`, y se indica en los campos de bits de los registros `INTC_SIR_IRQ[31:7]` y `INTC_SIR_FIQ[31:7]`.
- Son copias, respectivamente, de `INTC_IRQ_PRIORITY[31:7]` y `INTC_FIQ_PRIORITY[31:7]`.



Manejo de Interrupciones Espurias

- Si los siguientes registros cambian durante los 10 ciclos de *functional clock* luego de iniciado el ordenamiento de prioridad sus valores resultan inválidos.

INTC_SIR_IRQ

INTC_SIR_FIQ

INTC_IRQ_PRIORITY

INTC_FIQ_PRIORITY

- Esta condición se detecta tanto en **IRQ** como **FIQ**, y se indica en los campos de bits de los registros **INTC_SIR_IRQ[31:7]** y **INTC_SIR_FIQ[31:7]**.
- Son copias, respectivamente, de **INTC_IRQ_PRIORITY[31:7]** y **INTC_FIQ_PRIORITY[31:7]**.
- En ambos layouts el campo de bit se llaman **SPURIOUSIRQFLAG** y **SPURIOUSFIQFLAG**, respectivamente.



Manejo de Interrupciones Espurias

- Si los siguientes registros cambian durante los 10 ciclos de *functional clock* luego de iniciado el ordenamiento de prioridad sus valores resultan inválidos.

INTC_SIR_IRQ

INTC_SIR_FIQ

INTC_IRQ_PRIORITY

INTC_FIQ_PRIORITY

- Esta condición se detecta tanto en **IRQ** como **FIQ**, y se indica en los campos de bits de los registros **INTC_SIR_IRQ[31:7]** y **INTC_SIR_FIQ[31:7]**.
- Son copias, respectivamente, de **INTC_IRQ_PRIORITY[31:7]** y **INTC_FIQ_PRIORITY[31:7]**.
- En ambos layouts el campo de bit se llaman **SPURIOUSIRQFLAG** y **SPURIOUSFIQFLAG**, respectivamente.
- Un '0' indica interrupción y prioridad válida. '1' inválida.



Contenido

- 1 Controlador de Interrupciones
- 2 Interrupciones de Hardware
 - Características y Modos de trabajo
 - Manejo de Múltiples Interrupciones
- 3 GIC Architecture
 - Generic Interrupt Controller
- 4 Implementación: SITARA 3358. Cortex-A8
 - Descripción funcional
 - Procesamiento de Interrupciones
 - Modelo de Programación Básico
- 5 ANEXO
 - Anticipos de System Programming



Data Synchronization Barrier

Memory Barrier

Es el término general que define a este tipo de instrucciones.

Se trata de un tipo de instrucción que actúa como barrera que fuerza a la CPU o el compilador se vean forzados a cumplir una restricción de orden en la ejecución de las instrucciones previas a la barrera.

En general aseguran que no se ejecute ninguna instrucción posterior a la barrera hasta que la CPU no haya completado todas las instrucciones previas a la barrera.

Por lo general cuando una CPU las soporta, su implementación es a través de instrucciones específicas.



Data Synchronization Barrier en ARMv7



Data Synchronization Barrier en ARMv7

- El modelo de Programación de Sistemas (System Programming) de ARMv7, se sustenta en un complejo sistema de coprocesadores.



Data Synchronization Barrier en ARMv7

- El modelo de Programación de Sistemas (System Programming) de ARMv7, se sustenta en un complejo sistema de coprocesadores.
- Se accede a estos mediante un pequeño conjunto de instrucciones específicas.



Data Synchronization Barrier en ARMv7

- El modelo de Programación de Sistemas (System Programming) de ARMv7, se sustenta en un complejo sistema de coprocesadores.
- Se accede a estos mediante un pequeño conjunto de instrucciones específicas.
- Éstas instrucciones son bastante complejas y poseen una serie de operandos.



Data Synchronization Barrier en ARMv7

- El modelo de Programación de Sistemas (System Programming) de ARMv7, se sustenta en un complejo sistema de coprocesadores.
- Se accede a estos mediante un pequeño conjunto de instrucciones específicas.
- Éstas instrucciones son bastante complejas y poseen una serie de operandos.
- El primer operando define el coprocesador al que se envía el requerimiento.



Data Synchronization Barrier en ARMv7

- El modelo de Programación de Sistemas (System Programming) de ARMv7, se sustenta en un complejo sistema de coprocesadores.
- Se accede a estos mediante un pequeño conjunto de instrucciones específicas.
- Éstas instrucciones son bastante complejas y poseen una serie de operandos.
- El primer operando define el coprocesador al que se envía el requerimiento.
- Dos operandos definen registros especiales del coprocesador y otros dos definen el valor que se escribe en cada registro.



Data Synchronization Barrier en ARMv7

- El modelo de Programación de Sistemas (System Programming) de ARMv7, se sustenta en un complejo sistema de coprocesadores.
- Se accede a estos mediante un pequeño conjunto de instrucciones específicas.
- Éstas instrucciones son bastante complejas y poseen una serie de operandos.
- El primer operando define el coprocesador al que se envía el requerimiento.
- Dos operandos definen registros especiales del coprocesador y otros dos que definen el valor que se escribe en cada registro.
- El registro del Core no debe ser el **PC**. Su valor se transfiere al registro **Cn** del core (el primero de los dos especificados en la instrucción) cuando la instrucción es **MCR**.



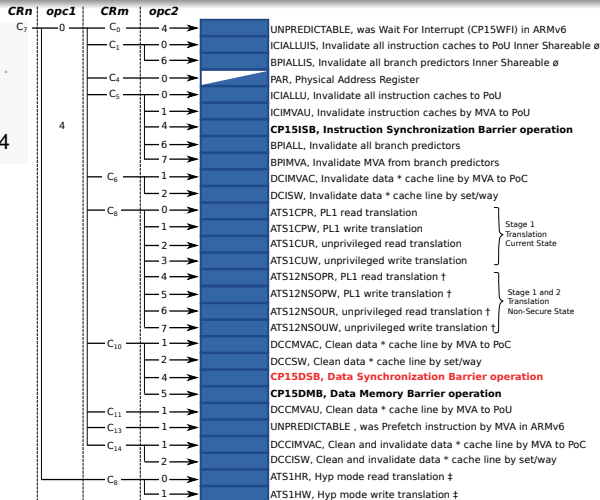
Registro c7 del Coprocesador 15

```
/* Paso 6: Data
   Synchronization Barrier.
   */
```

```
MOV R0, #0
```

```
MCR P15, #0, R0, C7, C10, #4
```

- CRn = C7
- opc1 = #0
- CRm = C10
- opc2 = #4



Tipo de Acceso

Read Only

Read Write

Write Only

‡ Introducido como parte de las Extensiones de Multiprocesamiento

† Implementado como parte de las Security Extensions solamente

‡ Implementado como parte de las Virtualization Extensions solamente



Referencias



Referencias

- Generic Interrupt Controller. ARM Ltd.

<https://documentation-service.arm.com/static/5f8ff21df86e16515cdbfafa?token=>



Referencias

- Generic Interrupt Controller. ARM Ltd.
<https://documentation-service.arm.com/static/5f8ff21df86e16515cdbfafa?token=>
- AM335x Sitara™Processors Technical Reference Manual.
<https://www.ti.com/lit/ug/spruh73q/spruh73q.pdf>

Capítulo 6. Descripción detallada de registros del controlador de interrupciones (layout), Mapa de direcciones de memoria, y tipo de interrupción.

