

Interrupciones y microcontroladores

Sistemas embebidos para tiempo real

Objetivos

- Describir el flujo de ejecución de un microcontrolador: contexto principal (main, también llamado “tarea”) y de interrupciones.
- Describir la secuencia de ejecución de interrupciones.
- Reconocer el problema de datos compartidos.
- Identificar los tiempos involucrados en las interrupciones.

Agenda próximas dos clases

- Primera clase
 - Fundamentos de las Interrupciones
 - Problema de datos compartidos
 - Latencia en las interrupciones
- Segunda clase
 - Soluciones al problema de los datos compartidos

Actividad colectiva

- Objetivos:
 - Evaluación de conocimientos previos sobre interrupciones, puesta a punto y repaso.
- Responder las siguientes preguntas por escrito
 1. ¿Qué es una ISR? ¿Qué es una IRQ?
 2. ¿Cómo se encuentra la ISR para una IRQ?
 3. ¿Se puede interrumpir en el medio de una instrucción?
 4. Si dos interrupciones ocurren al mismo tiempo: ¿qué pasa?
 5. ¿Se pueden anidar interrupciones? (nested)
 6. ¿Qué pasa con las interrupciones cuando están deshabilitadas?
 7. ¿Qué pasa si nos olvidamos de rehabilitarlas?
 8. ¿Se pueden escribir ISR en C?
 9. En los microcontroladores: ¿Cómo interrumpen los periféricos integrados?

Actividad en grupo

- Interrupciones en MSP430
 - Actividad:
 - responder las preguntas anteriores para micros de la familia MSP430xF2xxx
 - Grupos:
 - 3 a 4 participantes
 - Tiempo:
 - 5-10 minutos
 - Material: Manual de la familia MSP430xF2xxx, páginas 33-40
 - Puesta en común.

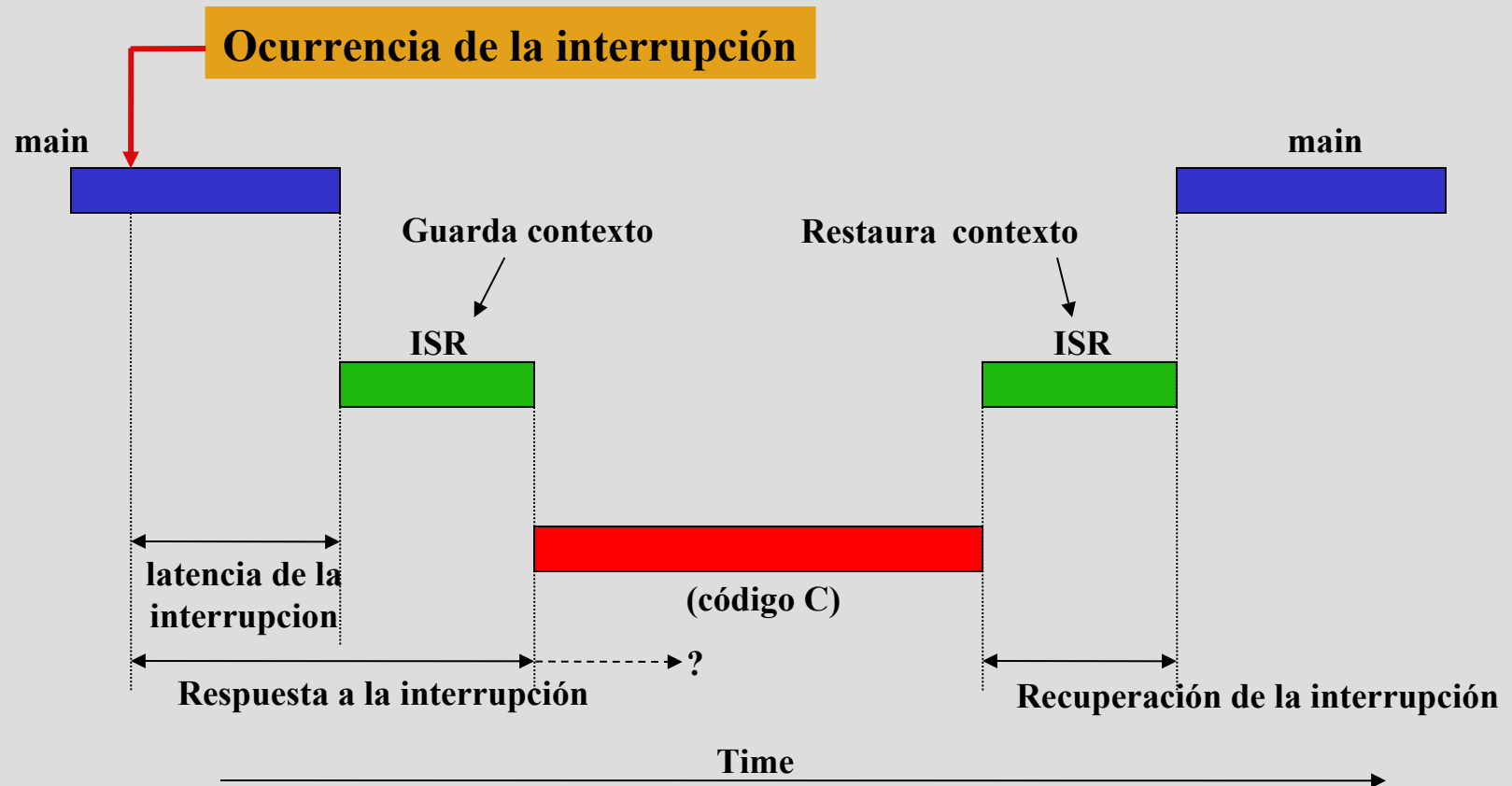
Interrupciones

- Escenario en sistemas de tiempo real
 - Procesador ejecutando un programa
 - Ocurre algo que necesita ser atendido
 - Procesador pone en “espera” el hilo principal y lo atiende
- Secuencia de eventos típica
 - Dispositivo que necesita atención se lo indica al μP a través de una señal de hardware.
 - μP guarda información de la tarea en ejecución (PC) y carga en PC la dirección de inicio de la ISR.
 - Se ejecuta la ISR y al final se retorna con RETI
 - Se carga en valor guardado del PC

Interrupciones

- Atención:
 - ISR comienza a ejecutarse en el “medio” del programa
 - CPU está en uso (registros con valores, etc.)
 - Al retornar de la ISR debe seguir como si nada hubiera pasado
- Entonces la ISR tiene que:
 - guardar el contexto (estado del procesador: SR, etc)
 - hace lo que tiene que hacer (dar respuesta a la interrupción)
 - restaura el contexto
 - retorna de la subrutina

Interrupciones: diagrama de tiempos



Basado en: J. Archibald, "Real-Time and Embedded Systems", slides: set2.ppt

Alternativas a las interrupciones

- Consulta (Polling):
 - Se toma la iniciativa de chequear (proactivo) en intervalos regulares (idealmente).
 - En aplicaciones simples puede manejar los eventos sin retardos significativos.
 - Usualmente funciona bien para aplicaciones con una sola tarea.
- Desventajas de polling
 - Cierta dificultad en aplicaciones con múltiples tareas.
 - Usa la CPU ineficientemente
 - Tiempo insumido en el chequeo podría ser usado en procesado
 - No es un problema en cuando el procesado es liviano ya que la CPU no tiene nada que hacer (salvo consumir)

Ventajas de las interrupciones

- Retardos uniformes en respuesta a eventos
- Independiente de la complejidad de la tarea que se está ejecutando
- Las “tareas” pueden dormir hasta que ocurra un evento
 - Esencial en sistemas multitarea basados en prioridades
- Estas ventajas tienen un costo
 - La complejidad del software de sistemas basados en interrupciones es más alta que los sistemas basados en polling

Fuentes de interrupción

- Depende de lo que se conecte a los pines de interrupción
- En microcontroladores: periféricos integrados + pin/es input
- Ejemplos:
 - Llegada de datos de un dispositivo E/S.
 - Terminación de una petición previa a un dispositivo E/S.
 - Cambio en la lectura de un sensor.
 - Acción de un usuario (presionado de un botón).
 - Detección de una falla
 - externa al CPU
 - interna al CPU (exception)
 - Expiración de un temporizador
 - Falla en la alimentación

Ejemplo de uso de interrupciones

- Descripción
 - Control de temperaturas en dos tanques.
 - Las temperaturas deben ser iguales.
 - Si son diferentes disparar alarma.
- Implementación
 - Interrupción periódica para lectura de temperaturas.
 - Lectura de temperatura es “instantánea”.
 - Verificación de temperaturas se hace en loop principal.

control-obvio.c

```
static int iTemperatures[2];

#pragma vector=TIMERX
__interrupt void vReadTemperatures(void)
{
    iTemperatures[0] = !! lee valor desde hardware
    iTemperatures[1] = !! lee valor desde hardware
}

int iTemp0;
int iTemp1;

void main (void)
{
    while(TRUE)
    {
        iTemp0 = iTemperatures[0];
        iTemp1 = iTemperatures[1];
        if (iTemp0 != iTemp1)
            !! Se activa una alarma muy molesta;
    }
}
```

control-oculto.c

```
static int iTemperatures[2];

#pragma vector=TIMERX
__interrupt void vReadTemperatures(void)
{
    iTemperatures[0] = !! lee valor desde hardware
    iTemperatures[1] = !! lee valor desde hardware
}

int iTemp0;
int iTemp1;

void main (void)
{
    while(TRUE)
    {
        if (iTemperatures[0] != iTemperatures[1])
            !! Se activa una alarma muy molesta;;
    }
}
```

control-oculto.c

```
static int iTemperatures[2];

#pragma vector=TIMERX
__interrupt void vReadTemperatures(void)
{
    iTemperatures[0] = !! lee valor desde hardware
    iTemperatures[1] = !! lee valor desde hardware
}

int iTemp0;
int iTemp1;

void main (void)
{
    while(TRUE)
    {
        if (iTemperatures[0] != iTemperatures[1])
            !! Se activa una alarma muy molesta;
    }
}
```

```
MOV     R1, (iTemperature[0])
MOV     R2, (iTemperature[1])
SUB     R1, R2
JCOND   ZERO, TEMP_OK
;
;Codigo para disparar alarma
;
TEMP_OK:
```

Problema de los datos compartidos

- ¿Cuándo está potencialmente presente?
 - Cuando se comparten datos entre ISR y el resto del código.
- ¿Por qué se comparten datos?
 - Las ISR deben hacer el trabajo estrictamente necesario para atender el hardware (lo veremos más tarde).
- ¿Cuándo surge y cuál es el problema ?
 - Cuando la ISR se ejecuta en el instante “inesperado”
 - Se produce inconsistencia de datos entre la ISR y el resto del código

Bug de los datos compartidos

- Características
 - No ocurre siempre
 - cuando se produce una interrupción entre dos instrucciones críticas
 - Baja probabilidad de ocurrencia, sin embargo ocurre:
 - cuando no se presta mucha atención
 - cuando no se está conectado para debugging
 - después que el *rover* “aterrizó” en Marte
 - en la demo con los clientes
 - Difícil de encontrar (típicamente se “arregla” solo)
 - En consecuencia: evitar este bug

Terminología

- Atómico:
 - una parte de un programa se dice atómico si no puede ser interrumpido, es decir se puede garantizar que será ejecutado como una unidad inseparable (atómica)
- Sección crítica:
 - conjunto de instrucciones que deben ser atómicas para asegurar el funcionamiento correcto.

Resumen hasta ahora...

- Problema de datos compartidos:
 - surge cuando código de “contexto main” accede datos compartidos con ISR de manera no atómica.
- Instrucciones de máquina
 - es la unidad atómica natural de ejecución
- Líneas de código C
 - muchas veces se mapea en varias líneas en assembler, entonces no son atómicas
 - si una línea de C necesita que sea atómica pasa a ser una sección crítica
- ¿Cómo podemos hacer atómica una sección crítica?
 - Enfoque preferido: deshabilitar y rehabilitar interrupciones
 - Veremos otros...

solucion.c

```
static int iTemperatures[2];

#pragma vector=TIMERX
__interrupt void vReadTemperatures(void) {
    iTemperatures[0] = !! lee valor desde hardware
    iTemperatures[1] = !! lee valor desde hardware
}

int iTemp0;
int iTemp1;

void main (void) {
    while(TRUE) {
        __disable_interrupt();
        iTemp0 = iTemperatures[0];
        iTemp1 = iTemperatures[1];
        __enable_interrupt();

        if (iTemp0 != iTemp1)
            !! Se activa una alarma muy molesta;
    }
}
```

Más ejemplos

- Temporizador

timer.c

```
static int iSeconds, iMinutes, iHours;

#pragma vector=TIMERX
__interrupt void vUpdateTime(void) {
    ++iSeconds;
    if (iSeconds >= 60){
        iSeconds = 0;
        ++iMinutes;
        if (iMinutes >= 60){
            iMinutes = 0;
            ++iHours;
            if (iHours >= 24)
                iHours = 0;
        }
    }
    // Hacer lo que haya que hace con el hardware
}


long iSecondsSinceMidnight(void) {
    return ( ((iHours*60) + iMinutes) *60) + iSeconds );
}
```

timer.c

```
static int iSeconds, iMinutes, iHours;
```

```
#pragma vector=TIMERX
```

```
__interrupt void vUpdateTime(void) {  
    // ...  
}
```



```
long iSecondsSinceMidnight(void) {  
    __disable_interrupt();  
    return ( ((iHours*60) + iMinutes) *60) + iSeconds );  
    __enable_interrupt();  
}
```

```
long iSecondsSinceMidnight(void) {  
    long lRetrunVal;  
    __disable_interrupt();  
    lRetrunVal = (((iHours*60) + iMinutes) *60) + iSeconds;  
    __enable_interrupt();  
    return lReturnVal;  
}
```

Tiempo de respuesta

- Interrupciones:
 - herramienta para obtener buenos tiempos de respuesta
- Concepto:
 - cantidad de tiempo que le lleva al sistema responder a una interrupción.
- ¿Cuán rápido se responde a las interrupciones?

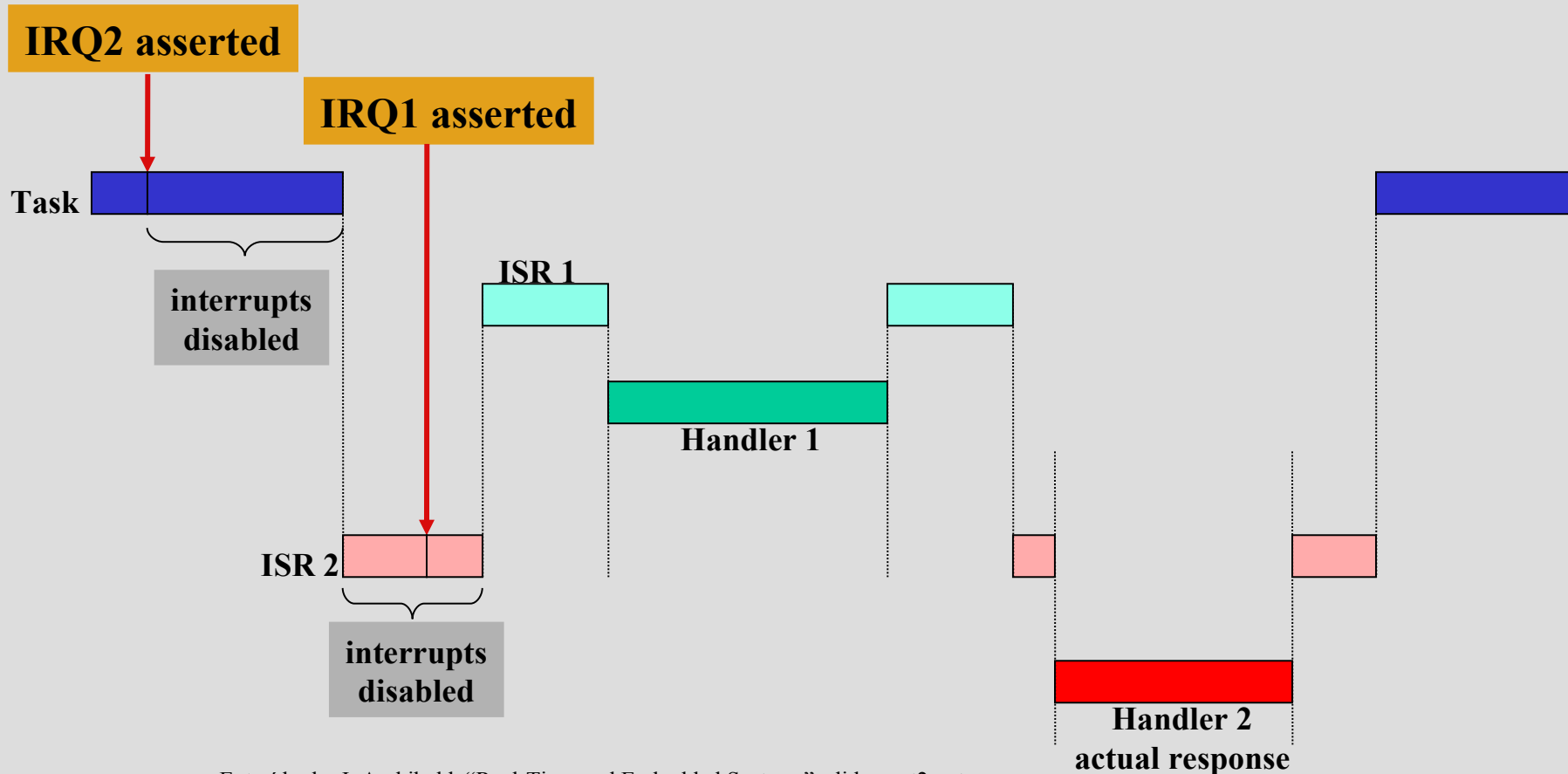
Tiempo de respuesta

- Depende de:
 1. El mayor período de tiempo en el cual las interrupciones están deshabilitadas.
 2. El tiempo que lleva ejecutar las ISR asociadas a las interrupciones de mayor prioridad.
 3. Cuánto tiempo le lleva al microprocesador finalizar lo que está haciendo y ejecutar la ISR.
 4. Cuánto tiempo le lleva a la ISR guardar el contexto y ejecutar instrucciones hasta considerarse "respuesta".

Análisis de los factores

- Tiempo máximo de las secciones críticas (factor 1)
 - mantenerlas cortas.
- Tiempo de ISR de mayor prioridad (factor 2)
 - asignar prioridades cuidadosamente
 - escribir subrutinas cortas
- Tiempo de respuesta del uP (factor 3)
 - fijo una vez seleccionado
- Tiempo para salvar contexto e ISR (factor 4)
 - salvar contexto: cantidad de registros
 - eficiencia de la subrutina: buena codificación

Tiempo de respuesta



Extraído de: J. Archibald, "Real-Time and Embedded Systems", slides: set2.ppt

Próxima clase

- Alternativas a deshabilitar interrupciones para el bug de datos compartidos

Bibliografía

- “An Embedded Software Primer” David E. Simon
 - Chapter 4: Interrupts
- "MSP430x1xx Family User's Guide"
- MSP430 Optimizing C/C++ Compiler v18.1.0 LTS. User's Guide.