Electrónica Digital II

Santiago Rúa Pérez, PhD.

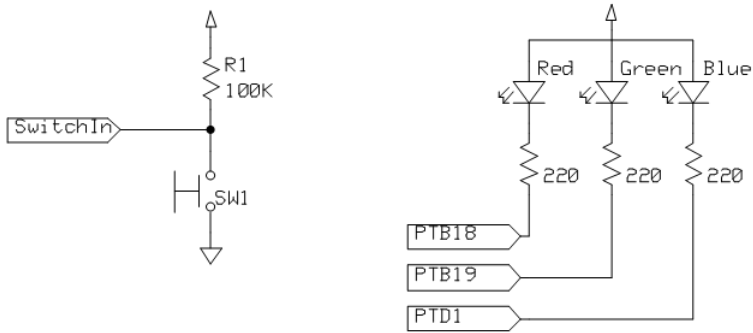
18 de septiembre de 2022

INTERRUPCIONES

Objetivos

- Entender los conceptos de excepciones e interrupciones
  - Entrando en el manejo de una excepción.
  - Saliendo del manejo de una excepción.
- Cortex-M4 interrupciones.
- Análisis de tiempos
- Diseño de programas con interrupciones.
- Fuentes
  - Cortex M4 Device Generic User Guide - DUI0553
  - Cortex M4 Technical Reference Manual - DDI0439B

Ejemplo de sistema con interrupción



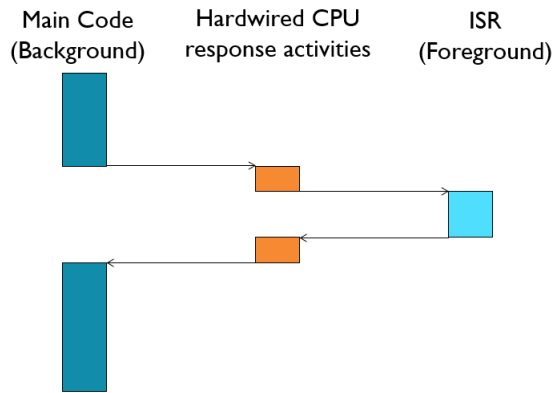
- Meta: cambiar el color del led RGB cuando se presiona el suiche.
- Explicar en detalle la interfaz entre el suiche y el módulo LED.
- Necesidad de adicionar suiche externo.

Cómo detectar cuando se presiona el suiche?

- Haciendo polling - a través de software.
  - Lento - se necesita explícitamente estar monitoreando el suiche y preguntar por el mismo.
  - Desperdicio de tiempo de la CPU - entre más rápido necesitemos una respuesta, mas seguido necesitamos estar chequeando.
  - Escalabilidad mala - difícil construir sistema con muchas actividades que puedan responder de forma ágil. Dependerá de otros procesos.
- Interrupciones - Uso especial de hardware dentro del MCU para detectar eventos, corre un código específico (interrupt service routine - ISR)
  - Eficiente - el código solo se ejecuta cuando es necesario.
  - Rápido - mecanismo de hardware.
  - Escalabilidad buena
    - El tiempo de respuesta del ISR no depende de otros procesos.
    - Los módulos pueden desarrollarse de forma independiente.

Secuencia de la interrupción

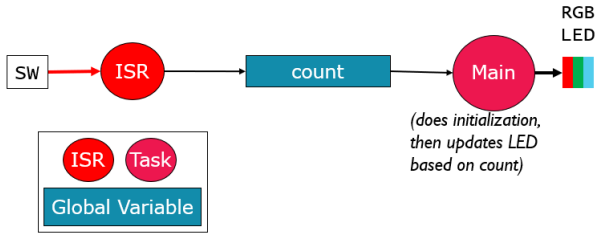
- El código principal esta corriendo.
- Ocurre algo que dispara la interrupción.
- El procesador hace un procesamiento de manejo del hardware.
- El procesador ejecuta ISR.
- El procesador hace manejo de hardware.
- El procesador continua con la ejecución del programa.



Secuencia de la interrupción

- Rutina asincronica disparada por hardware.
  - Disparado por una señal de hardware.
  - Asincrónico - puede pasar en cualquier momento del programa.
  - Rutina por software - Servicio de rutina corre como respuesta a la interrupción.
- Mecanismo fundamental de los microcontroladores.
  - Genera un procedimiento eficiente basado en evento en vez de polling
  - Genera una respuesta rápida sin importar el estado del programa.
  - Permite muchos hilos en sistemas embebidos sin la necesidad de sistemas operativos.

Ejemplo de uso



- Req1: cuando el suiche 1 es presionado, ISR incrementará un contador.
- Req2: el código principal prende los leds de acuerdo al valor del contador (Azul: 4, Verde:2, Rojo:1).
- Req3: ISR se ejecuta de acuerdo a la interrupción.

Entrando al administrador de la excepción

- Terminar la instrucción actual (excepto las largas).
- Guarda todo el contexto de los registros en el stack actual.
- Cambiar el modo de operación.
- Carga en el PC el administrador de la excepción.
- Carga registros LR y IPSR.
- Inicia ejecución de la interrupción.

Usualmente le toma al procesador 16 ciclos.

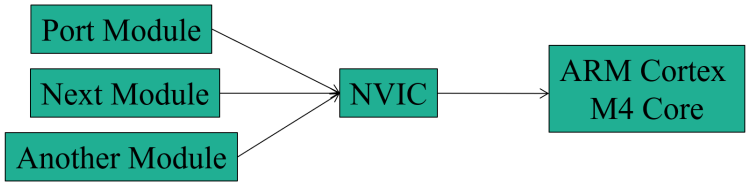
Saliendo de la interrupción

- Ejecuta el proceso de la interrupción.
- Selecciona el stack de retorno, restaura contexto.
- Vuelve a la ejecución del programa normal.

Interrupciones del microcontrolador Cortex M4

- Tipos de interrupciones.
  - Interrupciones por hardware.
    - **Asíncronas**: no estan relacionadas con lo que el procesador está ejecutando.
    - Ejemplo: se termina la interrupción, un caracter es recibido en el puerto serial, el ADC termina la conversión.
  - Excepciones, fallas o interrupciones por software.
    - **Síncronas**: son el resultado de la ejecución específica de un código.
    - Ejemplos: instrucciones sin definir, overflow ocurre en la ejecución de una instrucción.
  - La mayoría de interrupciones se pueden habilitar o deshabilitar cuando se necesite.
- Rutina de servicio de interrupcción.
  - Subrutina que el procesador se ve obligado a realizar para responder a un evento específico dentro del procesador.
  - Despues de que se complete el ISR, MCU vuelve a donde se estaba ejecutando.

Ejemplo de uso



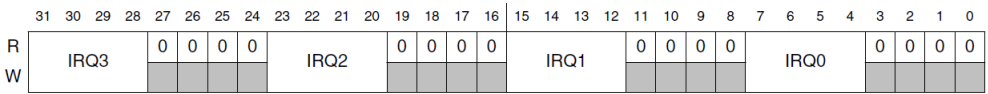
- NVIC (*Nested Vector Interrupt Controller*) maneja y prioriza las interrupciones externas del MCU.
- Las interrupciones son un tipo de excepciones.
- Modos
  - Thread Mode: se entra cuando se genera un reset.
  - Handler mode: entra para ejecutar una excepción.
- Nivel de privilegios.
- Stack pointers.
  - Main Stack Pointer, MSP
  - Process Stack Pointer, PSP
- Estados de excepción: inactivo, pendiente, activo.

Algunas fuente de interrupciones

Address	Vector	IRQ	Source	Description
0x0000_0004	1		ARM Core	Initial program counter
0x0000_0008	2		ARM Core	Non-maskable interrupt
0x0000_0040-80	16-32	0-16	DMA controller	Transfer complete or error
0x0000_00A0-A4	40-41	24-25	I2C	Status and error
0x0000_00A8-AC	42-43	26-27	SPI	Status and error
0x0000_00BC-D8	47-54	31-38	UART	Status and error
0x0000_012C	75	59	Port Control Module	Port A
0x0000_0130	76	60	Port Control Module	Port B
0x0000_0134	77	61	Port Control Module	Port C
0x0000_0138	78	62	Port Control Module	Port D
0x0000_013C	79	63	Port Control Module	Port E

Hasta 85 vectores diferentes del procesador, 16 del procesador principal. Table 3-5 reference manual.

NVIC registros y estados



- Prioridad - permite al programa priorizar la respuesta si ocurren dos interrupciones.
- IPR0-59 registros: 4 bits por interrupcion, cuantro fuentes de interrupción por registro.
- La prioridad 0 es la de mayor atención.
- CMSIS: NVIC_SetPriority(IRQnum,priority)
- Enable: permite habilitar las interrupciones.
  - Se accesan con dos registros: Set enable con NVIC_ISER, clear enable con NVIC_ICER
  - CMSIS: NVIC_EnableIRQ(IRQnum), NVIC_DisableIRQ(IRQnum)
- Pending: la interrupción se dio, pero no ha sido atendida.
  - CMSIS: NVIC_SetPendingIRQ(IRQnum), NVIC_ClearPendingIRQ(IRQnum)

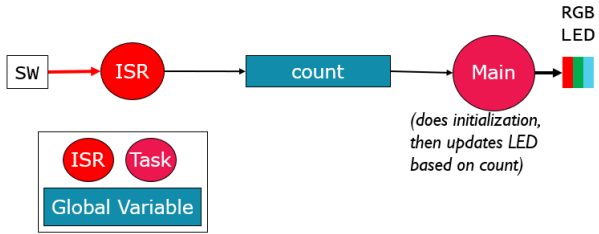
Excepciones del procesador

- Similar a deshabilitar interrupciones globales en la MCU.
- PRIMASK - registro de mascara de excepción
  - Solo utiliza el bit cero.
  - Ponerlo en 1 evita la activación de cualquier interrupción.
  - Ponerlo en 0 permite la activación de todas las excepciones.
  - Ayuda a prevenir borrado de datos o condiciones extrañas de comportamiento.
- CMSIS-CORE API
  - void __enable_irq() - clears PM flag
  - void __disable_irq() - sets PM flag
  - uint32_t __get_PRIMASK() - returns value of PRIMASK
  - void __set_PRIMASK(uint32_t x) - sets PRIMASK to x

Priorización

- Las excepciones son priorizadas para responder en orden de acuerdo a la solicitud dada (número pequeño = mayor la prioridad)
- Algunas prioridades están fijas.
  - Reset: -3, mayor prioridad.
  - NMI: -2
  - Hard fault: -1
- Las prioridades de los periféricos son ajustables.
  - Los valores son almacenados en el registro de prioridad de la interrupción IPR
  - Un valor por encima de cero.
- Solicitud simultanea de excepciones? Se atiende la de menor número.
- Una nueva excepción se requiere mientras se está atendiendo otra interrupción?
  - La nueva tiene mayor prioridad? Se atiende la nueva excepción.
  - Es menor? entonces se pone en estado pendiente. Termina la ejecución de la actual y se atiende la nueva.

Ejemplo



- Req1: cuando el suiche 1 es presionado, ISR incrementará un contador.
- Req2: el código principal prende los leds de acuerdo al valor del contador (Azul: 3, Verde:2, Rojo:1).
- Req3: ISR se ejecuta su linea de acuerdo a la interrupción.

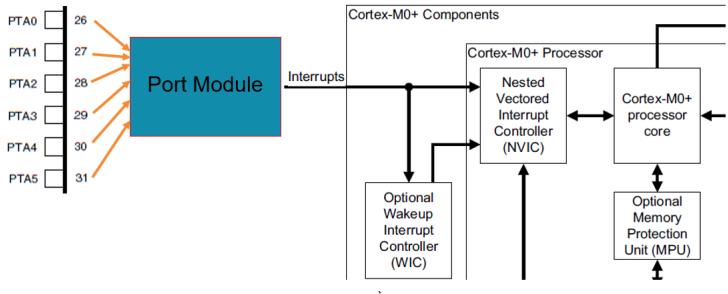
Construyendo el programa

- Primero haga el paso al paso del programa en pequeños pasos.
  - Programa principal:
    - Inicializar el sistema.
    - Inicializar suiche
    - Inicializar el led.
    - Inicializar las interrupciones.
    - Luego repita
    - Actualizar el led basado en el contador.
  - Atención a la interrupción
    - Actualizar el contador.
- Determinar que variable se va a compartir entre el programa principal y la atención a la interrupción.
  - A través de variables globales se comunican el programa principal y la interrupción.
  - Siempre declare las variables compartidas como volatile.
  - Asegure que el acceso de la variable es atómico.

Panorama general

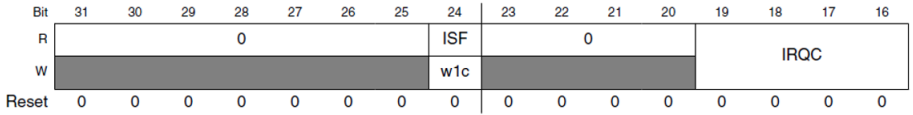
- main
  - Nivel más arriba de ejecución.
- suiches
  - #defines para las conexiones del suiche
  - declaración de la variable contador.
  - código para inicializar suiche e interrupción.
  - ISR para el suiche.
- Leds
  - #define para las conexiones del led.
  - Código para inicializar el Led.
- debug_signals
  - #define para la señal de debug.
  - Código para inicializar el debug.

Port module



- Los puertos estan conectado al NVIC.
- Registros relevantes
  - PCR
    - Cada registro corresponde a la entrada de un pin.
  - ISFR
    - Cada bit corresponde a una entrada.
    - El bit es uno si ha sido detectada una interrupción.

Pin Control Register



- ISF indica si se ha detectado una interrupción. Es lo mismo que acceder al ISFR.
- IRQC define el comportamiento externo
- También puede trabajar con acceso directo a memoria.

IRQC	Configuración
0000	Interrupción deshabilitada
...	DMA
1000	Interrupción cuando es cero
1001	Interrupción en flanco de subida
1010	Interrupción en flanco de bajada
1011	Interrupción en los dos casos
1100	Interrupción cuando es uno

Inicialización de la interrupción

```
1  void init_switch(void){
2    //Clock activation PTA
3    SIM->SCGC5 |= SIM_SCGC5_PORTA_MASK;
4
5    // Port as GPIO and interrupt in falling edge
6    PORTA->PCR[SW1] = PORT_PCR_MUX(1) | PORT_PCR_IRQC(0x0A);
7
8    // GPIO as input (PTA4)
9    PTA->PDDR &= ~MASK(SW1);
10
11   /* Enable Interrupts */
12   NVIC_SetPriority(PORTA_IRQn, 2);
13   NVIC_ClearPendingIRQ(PORTA_IRQn);
14   NVIC_EnableIRQ(PORTA_IRQn);
15
16 }
17
```

Atención a interrupción

Rutina de servicio de atención a interrupción.

```
1  void PORTA_IRQHandler(void) {
2    if ((PORTA->ISFR & MASK(SW1))) {
3      band=1;
4    }
5    // clear status flags
6    PORTA->ISFR = 0xffffffff;
7  }
8
```

Programa principal

```
1  int main(void) {
2    __disable_irq();              /* disable all IRQs */
3    init_rgb();
4    init_switch();
5    __enable_irq();               /* global enable IRQs */
6    while(1) {
7      RGB_task();
8      __wfi();
9    }
10   return 0 ;
11 }
```

Tarea RGB

Máquina de estados para la tarea RGB

```
1  void RGB_task(void){
2    static enum {L_RED,L_GREEN,L_BLUE} next_state=L_RED;
3    if(band==1){
4      switch(next_state){
5        case L_RED:
6          Control_RGB_LEDs(1,0,0);
7          next_state=L_GREEN;
8          break;
9        case L_GREEN:
10         Control_RGB_LEDs(0,1,0);
11         next_state=L_BLUE;
12         break;
13       case L_BLUE:
14         Control_RGB_LEDs(0,0,1);
15         next_state=L_RED;
16         break;
17     }
18     band = 0;
19   }
20 }
21
```

Características del ISR

- No hay argumentos o valores de retorno en la función - void es el unico tipo de datos válido.
- Mantenerlo corto y simple
  - Más fácil de hacer debug.
  - Mejora el tiempo de respuesta del sistema.
- Nombre la función del ISR de acuerdo al sistema de excepcion del CMSIS-CORE
- Lea el registro con la bandera de status para determinar quien genero la interrupción.
- Limpie la bandera escribiendo unos en el ISFR.

Datos volatiles

- Los compiladores asumen que las variables en memoria no cambian espontáneamente y optimizan código con base en esto.
  - No vuelven a cargar desde memoria si la función actual no la ha cambiado.
  - Carga la variable desde memoria a los registros (acceso más rápido).
  - Escribe de nuevo en memoria cuando el procedimiento a terminado o cuando se queda sin registros que compilar.
- La optimización puede fallar.
- Variables que pueden fallar:
  - Registros de periféricos mapeados en memoria - el registro cambia por si mismo.
  - Variables globales modificas por ISR - ISR cambia la variable.
  - Variables globales en una aplicación multihilos.

Datos compartidos no anatómicos

Algunos datos o objetos le toman múltiples operaciones para ser modificados.

- Se desea mantener un manejo del tiempo.
- Se utiliza una interrupción de un segundo. 1Hz.
- Sistema
  - La estructura TimerVal lleva el conteo.
  - La estructura se actual dentro de la rutina ISR.

```
void GetDateTime(DateTimeType * DT){
 DT->day = TimerVal.day;
 DT->hour = TimerVal.hour;
 DT->minute = TimerVal.minute;
 DT->second = TimerVal.second;
}
```

```
void DateTimeISR(void){
 TimerVal.second++;
 if (TimerVal.second > 59){
   TimerVal.second = 0;
   TimerVal.minute++;
   if (TimerVal.minute > 59) {
     TimerVal.minute = 0;
     TimerVal.hour++;
     if (TimerVal.hour > 23) {
        TimerVal.hour = 0;
       TimerVal.day++;
       … etc.
     }
```

Datos compartidos no anatómicos - Problema?

- Problema
  - Una interrupción en el momento inadecuado puede ocasionar una falla en la actualización de DT
- Caso de falla
  - TimerVal es {10, 23, 59, 59} (10th day, 23:59:59)
  - El código principal llama a la tarea GetDateTime(), el cual comienza la actualización de los datos. DT: day = 10, hour = 23
  - Una interrupción por timer ocurre, se actualiza TimerVal a {11, 0, 0, 0}
  - GetDateTime() retoma la ejecución y copia el restos de los campos de DT: minute = 0, second = 0
  - DT ahora vale 10, 23, 0, 0
  - El sistema piensa que salto una hora antes.
- Problema fundamental - **Condición de carrera**
  - Preemption le posibilita al ISR tomar el control y posiblemente reescribir data.
  - Se debe asegurar acceso atómico (indivisible) al objeto. Por ejemplo si el procesador es de 32 bits, tengo variables mas grandes?

Definiciones

- **Condición de carrera**: Comportamiento anómalo debido a una dependencia crítica inesperada del momento relativo de los eventos. El resultado del código de ejemplo depende del tiempo relativo de las operaciones de lectura y escritura.
- **Sección crítica**: Una sección de código que crea una posible condición de carrera. La sección de código solo puede ser ejecutada por un proceso a la vez. Se requiere algún mecanismo de sincronización en la entrada y salida de la sección crítica para asegurar el uso exclusivo.

Solución - Deshabilitar momentáneamente Preemption

- Prevenir preemption dentro de una sección crítica del código.
- Si ISR puede escribir a una variable compartida, entonces se debe deshabilitar esa sección.
  - Almacenar el estado actual de la interrupción
  - deshabilite interrupciones hasta que actualice las variables compartidas.
- Restaure el estado anterior.
- Use el CMSIS-CORE para guardar, almacenar, y restaurar las interrupciones.
- Evite si es posible la desactivación de las interrupciones si retrasa la respuesta a todas las demás solicitudes de procesamiento
- Que sea el tiempo mas corto posible.

```
void GetDateTime(DateTimeType *
DT){
 uint32_t m;

 m = __get_PRIMASK();
 __disable_irq();

 DT->day = TimerVal.day;
 DT->hour = TimerVal.hour;
 DT->minute = TimerVal.minute;
 DT->second = TimerVal.second;
 __set_PRIMASK(m);
}
```

Laboratorio 3 (10 %)

- Realice un juego de simon dice con cuatro pulsadores y cuatro leds externos. El objetivo es que el sistema genere una secuencia de pulsaciones de los led para que el usuario siga en el mismo orden. Utilice interrupciones para los cuatro pulsadores externos. A continuación les dejo un enlace
  Video de simon dice

INTERRUPCIONES

GRACIAS