Laboratorio 5:

Gestión básica de interrupciones

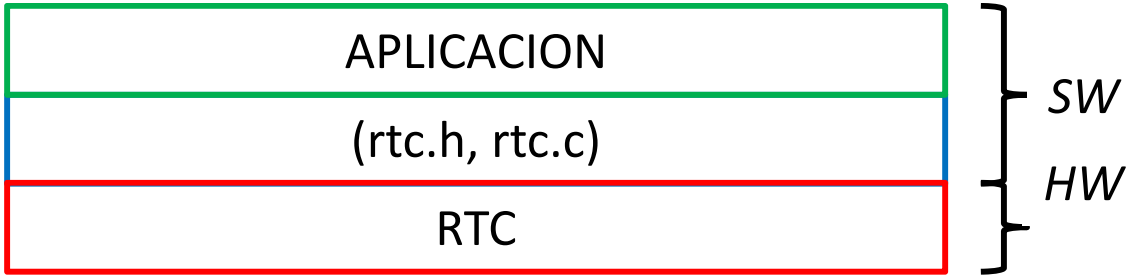
programación de un reloj de tiempo real

Programación de sistemas y dispositivos

José Manuel Mendías Cuadros
Dpto. Arquitectura de Computadores y Automática
Universidad Complutense de Madrid

Presentación

- Desarrollar una capa de firmware para la gestión de un RTC
  - Implementaremos 5 funciones.
    - Inicialización: **`rtc_init`**
    - Actualización/recuperación de hora y fecha: **`rtc_puttime / rtc_gettime`**
    - Activación/desactivación de interrupciones periódicas por ticks del RTC así como instalación de la RTI que la atenderá: **`rtc_open / rtc_close`**



- Dado que trabajaremos con interrupciones es necesario configurar previamente el controlador de interrupciones
  - Tendrá una configuración fija:
    - Todas las fuentes interrumpirán por la línea IRQ del procesador (no usaremos la línea FIQ)
    - Todas las IRQ serán vectorizadas
    - Las interrupciones tendrán un esquema de priorización fijo equivalente al de por defecto fijo
  - No desarrollaremos un driver, lo haremos como parte de la inicialización del sistema

Controlador de interrupciones

configuración

- Prioridad entre líneas del maestro: fija
  - I_PMST[12] = 1
- Prioridad relativa líneas del maestro: mGA > mGB > mGC > mGD
  - I_PMST[7:6] = 0 — mGA primero
  - I_PMST[5:4] = 1 — mGB segundo
  - I_PMST[3:2] = 2 — mGC tercero
  - I_PMST[1:0] = 3 — mGD cuarto
- Prioridad entre líneas de los esclavos: fija
  - I_PMST[11:8] = 0xF
- Prioridad entre líneas del esclavo A: EINT0 > EINT1 > EINT2 > EINT3
  - I_PSLV[31:30] = 0 — EINT0 primero
  - I_PSLV[29:28] = 1 — EINT1 primero
  - I_PSLV[27:26] = 2 — EINT2 primero
  - I_PSLV[25:24] = 3 — EINT3 primero

Controlador de interrupciones

configuración

- Prioridad entre líneas del esclavo B: ZDMA0 > ZDMA1 > BDMA0 > BDMA1
  - I_PSLV[23:22] = 0 — ZDMA0 primero
  - I_PSLV[21:20] = 1 — ZDMA1 segundo
  - I_PSLV[19:18] = 2 — BDMA0 tercero
  - I_PSLV[17:16] = 3 — BDMA1 cuarto
- Prioridad entre líneas del esclavo C: TIMER0> TIMER1 > TIMER2 > TIMER3
  - I_PSLV[15:14] = 0 — TIMER0 primero
  - I_PSLV[13:12] = 1 — TIMER1 segundo
  - I_PSLV[11:10] = 2 — TIMER2 tercero
  - I_PSLV[9:8] = 3 — TIMER3 cuarto
- Prioridad entre líneas del esclavo D: URxD0 > URxD1 > IIC > SIO
  - I_PSLV[7:6] = 0 — URxD0 primero
  - I_PSLV[5:4] = 1 — URxD1 segundo
  - I_PSLV[3:2] = 2 — IIC tercero
  - I_PSLV[1:0] = 3 — SIO cuarto
- Esta es la configuración inicial tras reset.

Controlador de interrupciones

configuración

- Todas las fuentes de interrupción conectadas al controlador interrumpirán a través de la línea IRQ del procesador
  - INTMOD = 0 — todas en IRQ mode
  - INTCON[1] = 0 — interrupciones IRQ habilitadas
  - INTCON[0] = 1 — interrupciones FIQ deshabilitadas
- Todas las interrupciones IRQ serán vectorizadas
  - INTCON[2] = 0 — modo vectorizado para las IRQ

- Resumen:
  - I_PMST = 0x1F1B (0001.1111.0001.1011)
  - I_PSLV = 0x1B1B1B1B (0001.1011.0001.1011.0001.1011.0001.1011)
  - INTMOD = 0x0
  - INTCON = 0x1 (0001)

Controlador de interrupciones

gestión de interrupciones

- Para evitar que un dispositivo (o todos) interrumpa
  - INTMSK[i] = 1 enmascara la del dispositivo i
  - INTMSK[26] = 1 enmascara globalmente a todos los dispositivos
- Para permitir que un dispositivo interrumpa
  - INTMSK[26] = 0 desenmascara globalmente a todos los dispositivos
  - INTMSK[i] = 0 desenmascara la del dispositivo i
- Para indicar la finalización de la ISR al controlador de interrupciones:
  - I_ISPC[i] = 1 borra el flag de interrupción pendiente del dispositivo i
  - Si la RTI no borra el flag, el controlador de interrupciones indefinidamente lanzará de nuevo la RTI cada vez que ésta finalice
- Adicionalmente se puede ignorar el contenido de:
  - I_CSLV, I_CMST, porque las prioridades de interrupción son fijas
  - F_ISPC, porque todas las interrupciones son en modo IRQ
  - INTPND, I_ISPR, porque todas las interrupciones están vectorizadas y no es necesario hacer pooling de interrupciones pendientes ni priorizarlas por SW

Reloj de tiempo real

configuración

- Fuente de reloj primaria: única opción posible
  - RTCCON[1] = 0 XTAL
- Registros BCD: única opción posible
  - RTCCON[2] = 0 merged
- Lectura/escritura de registros: deshabilitada
  - RTCCON[0] = 0
  - Con eso se reduce el consumo del dispositivo (que va alimentado por pilas)
- Alarma: deshabilitada
  - RTCALM = 0
- Función Round Reset: deshabilitada
  - RTCRST[3] = 0

- Resumen:
  - RTCCON = 0x0 (0000)
  - RTCALM = 0x0
  - RTCRST = 0x0 (0XXX)

Reloj de tiempo real

gestión

- Para leer/modificar la hora y fecha del RTC
  - RTCCON[0] = 1 habilita el acceso a los registros de hora/fecha del RTC
  - Leer/escribir los registros BCDxxx
  - RTCCON[0] = 0 deshabilita el acceso a los registros de hora/fecha del RTC
  - Todos los datos están codificados con 1 ó 2 dígitos BCD (4 bits por dígito)

- Para deshabilitar la generación de ticks del RTC
  - TICNT[7] = 0 deshabilita la interrupción periódica por tick del RTC

- Para habilitar la generación de ticks del RTC
  - TICNT[6:0] = n periodo entre interrupción = (n+1)/128 segundos
  - TICNT[7] = 1 habilita la interrupción periódica por tick del RTC
  - Aunque estén habilitadas en el RTC, para que sean efectivas deben estar también desenmascaradas en el controlador de interrupciones.

Inicialización del sistema

system.c

- En este lab configuraremos el controlador de interrupciones e instalaremos ISR "por defecto" en la tabla virtual de vectores de IRQ:

```
#include <s3c44b0x.h>
#include <s3cev40.h>
#include <system.h>
#include <uart.h>
#include <common_types.h>

static void port_init( void );
static void install_dummy_isr( void );
static void show_sys_info( void );
```

Declara 32 ISR que se ejecutarán en caso de que la aplicación no instale otras

```
void isr_SWI_dummy( void )    __attribute__ ((interrupt ("SWI")));
void isr_UNDEF_dummy( void )  __attribute__ ((interrupt ("UNDEF")));
void isr_IRQ_dummy( void )    __attribute__ ((interrupt ("IRQ")));
void isr_FIQ_dummy( void )    __attribute__ ((interrupt ("FIQ")));
void isr_PABORT_dummy( void ) __attribute__ ((interrupt ("ABORT")));
void isr_DABORT_dummy( void ) __attribute__ ((interrupt ("ABORT")));
void isr_ADC_dummy( void )    __attribute__ ((interrupt ("IRQ")));
void isr_RTC_dummy( void )    __attribute__ ((interrupt ("IRQ")));
...
void isr_USB_dummy( void )    __attribute__ ((interrupt ("IRQ")));
```

Inicialización del sistema

system.c

```
void sys_init( void )
{
  WTCON = 0;
  INTMASK = ~0;  ····· Enmascara todas las interrupciones
  ...
  I_PMST = ...;
  I_PSLV = ...;
  INTMOD = ...;
  install_dummy_isr(); ····· Instala RTI por defecto a todas los tipos de interrupción
  EXTINTPND = ...;····· Borra interrupciones externas pendientes por  la línea EINT[7:4]
  I_ISPC = ...; ····· Borra todas las interrupciones pendientes
  INTCON = ...;

  SET_OPMODE( SVCMODE );····· Pone el procesador en modo SVC
  SET_IRQFLAG( 0 ); ····· Habilita en el procesador las interrupciones IRQ
  SET_FIQFLAG( 1 ); ····· Deshabilita en el procesador las interrupciones FIQ

  port_init();
  uart0_init();
  show_sys_info();····· Envía por la UART0 información de autoría y de sistema

}
```

Inicialización del sistema

system.c

```
static void install_dummy_isr( void )
{
  ...
  pISR_TICK = (uint32) isr_TICK_dummy;
  ...
}
```

Instala cada RTI en su lugar de la tabla virtual de vectores de IRQ (véase s3cev40.h)

```
void isr_TICK_dummy( void )
{
  uart0_puts( "\n\n*** ERROR FATAL: ejecutando isr_TICK_dummy" );
  SEGS = 0x75;
  while( 1 );
}
```

Cada RTI muestra algún mensaje de error y bloquea el sistema

Driver del RTC

rtc.h

```
#ifndef __RTC_H__
#define __RTC_H__

#include <common_types.h>

typedef struct
{
  uint8 sec;
  uint8 min;
  uint8 hour;
  uint8 mday;
  uint8 wday;
  uint8 mon;
  uint8 year;
} rtc_time_t;

void rtc_init( void );
void rtc_puttime( rtc_time_t *rtc_time );
void rtc_gettime( rtc_time_t *rtc_time );
void rtc_open( void (*isr)(void), uint8 tick_count );
void rtc_close( void );

#endif //__RTC_H__
```

Estructura para almacenar en binario la hora/fecha del sistema:
Segundo (0-59), minuto (0-59), hora (0-23)
- día del mes (1-31), día de la semana (1-7) comenzando por el domingo
- mes (1-12)
- año (0-99)

Driver del RTC

rtc.c

```
extern void isr_TICK_dummy( void );

void rtc_init( void )
{
  TICNT   = ...;
  RTCALM  = ...;
  RTCRST  = ...;
  RTCCON  = ...;
  BCDYEAR = ...;
  BCDMON  = ...;
  BCDDAY  = ...;
  BCDDATE = ...;
  BCDHOUR = ...;
  BCDMIN  = ...;
  BCDSEC  = ...;
  ALMYEAR = ...;
  ALMMON  = ...;
  ALMDAY  = ...;
  ALMHOUR = ...;
  ALMMIN  = ...;
  ALMSEC  = ...;
  RTCCON &= ...;
}
```

Debe habilitar la posibilidad de leer/escribir los registros de hora/fecha del RTC

Inicializa la hora/fecha a las 00:00:00 del martes 1 de enero de 2013

Inicializa a 0 los registros de alarma

Deshabilita la posibilidad de leer/escribir los registros de hora/fecha del RTC

Driver del RTC

rtc.c

```
void rtc_puttime( rtc_time_t *rtc_time )
{
  RTCCON |= ...;
```

Habilita la posibilidad de leer/escribir los registros de hora/fecha del RTC

```
  BCDYEAR = ...;
  BCDMON  = ...;
  BCDDAY  = ...;
  BCDDATE = ...;
  BCDHOUR = ...;
  BCDMIN  = ...;
  BCDSEC  = ...;
```

Actualiza la hora y fecha del RTC a la indicada por el argumento.
Debe hacer una **conversión binario -> BCD** de los datos-

```
  RTCCON &= ...;
```

Deshabilita la posibilidad de leer/escribir los registros de hora/fecha del RTC

```
}
```

Driver del RTC

rtc.c

```
void rtc_gettime( rtc_time_t *rtc_time )
{
  RTCCON |= ...;
  rtc_time->year = ...;
  rtc_time->mon  = ...;
  rtc_time->mday = ...;
  rtc_time->wday = ...;
  rtc_time->hour = ...;
  rtc_time->min  = ...;
  rtc_time->sec  = ...;
  if( ! rtc_time->sec ){
    rtc_time->year = ...;
    rtc_time->mon  = ...;
    rtc_time->mday = ...;
    rtc_time->wday = ...;
    rtc_time->hour = ...;
    rtc_time->min  = ...;
    rtc_time->sec  = ...;
  };
  RTCCON &= ...;
}
```

Habilita la posibilidad de leer/escribir los registros de hora/fecha del RTC

Almacena en el argumento la hora y fecha del RTC.
Debe hacer una **conversión BCD -> binario** de los datos.

La lectura de los segundos debe ser la última

Si los segundos leídos son 0, repite la lectura para evitar la inconsistencia derivada de la lectura no atómica de la hora/fecha del RTC (está almacenada en varios registros que se leen secuencialmente)

Deshabilita la posibilidad de leer/escribir los registros de hora/fecha del RTC

Driver del RTC

rtc.c

```
void rtc_open( void (*isr)(void), uint8 tick_count )
{
  pISR_TICK = ...; ····· instala la ISR argumento en la tabla virtual de vectores de IRQ
  I_ISPC    = ...; ····· borra flag de interrupción pendiente por ticks de RTC
  INTMSK   &= ...; ····· desenmascara globalmente interrupciones e interrupciones por tick de RTC
  TICNT     = ...; ····· habilita en el RTC la generación de ticks y fija el valor del contador que los genera
}

void rtc_close( void )
{
  TICNT     = ...; ····· deshabilita en el RTC la generación de ticks
  INTMSK   |= ...; ····· enmascara interrupciones por tick de RTC
  pISR_TICK = ...; ····· instala isr_TICK_dummy en la tabla virtual de vectores de interrupción
}
```

Aplicación

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

void main( void )
{
  rtc_time_t rtc_time;

  sys_init();
  uart0_init();
  rtc_init();

  rtc_gettime( &rtc_time );
  uart0_puts( "\n\nFecha y hora iniciales: " );
  uart0_putint( rtc_time.mday );
  ...
  uart0_puts( "\nIntroduzca nueva fecha\n" );
  uart0_puts( "  - Dia: " );
  rtc_time.mday = (uint8) uart0_getint();
  ...
  rtc_puttime( &rtc_time );

  rtc_open( isr_tick, 127 );
  while( 1 );
}
```

Declara esta función (de aplicación) como RTI por IRQ

Inicializa el sistema

Lee y visualiza fecha/hora del RTC

Pide nueva fecha/hora y actualiza el RTC

Instala una RTI (de aplicación) para que se dispare cada segundo

Indefinidamente la tarea main queda en espera

Aplicación

```
void isr_tick( void )
{
  rtc_time_t rtc_time;

  rtc_gettime( &rtc_time );

  uart0_puts( "\nFecha y hora: " );
  uart0_putint( rtc_time.mday );
  uart0_putchar( '/' );
  uart0_putint( rtc_time.mon );
  uart0_putchar( '/' );
  uart0_putint( rtc_time.year );
  uart0_putchar( ' ' );
  uart0_putint( rtc_time.hour );
  uart0_putchar( ':' );
  uart0_putint( rtc_time.min );
  uart0_putchar( ':' );
  uart0_putint( rtc_time.sec );

  I_ISPC = BIT_TICK;
}
```

Lee y visualiza fecha/hora del RTC

Borra el flag de interrupción pendiente
(si no, indefinidamente al terminar de ejecutar la función volvería a ejecutarse)

Tareas

1. Crear el proyecto **`lab5`** a partir de una copia de uno anterior.
2. Descargar de la Web en el directorio **`lab5`** el fichero **`lab5.c`**
3. Refrescar el proyecto **`lab5`**.
4. Descargar de la Web en el directorio **`BSP/include`** el fichero **`rtc.h`**
5. Codificar en **`BSP/source`** los ficheros:
   - **`system.c`** y **`rtc.c`**
6. Refrescar el proyecto **`BSP`**
7. Compilar primero el proyecto **`BSP`** y después el proyecto **`lab5`**.
8. Crear una configuración de depuración **`lab5`** a partir de una anterior.
9. Arrancar Termite.
10. Conectar la placa y encenderla.
11. Arrancar OpenOCD.
12. Arrancar la configuración de depuración **`lab5`**