Técnicas Digitales II

Timer

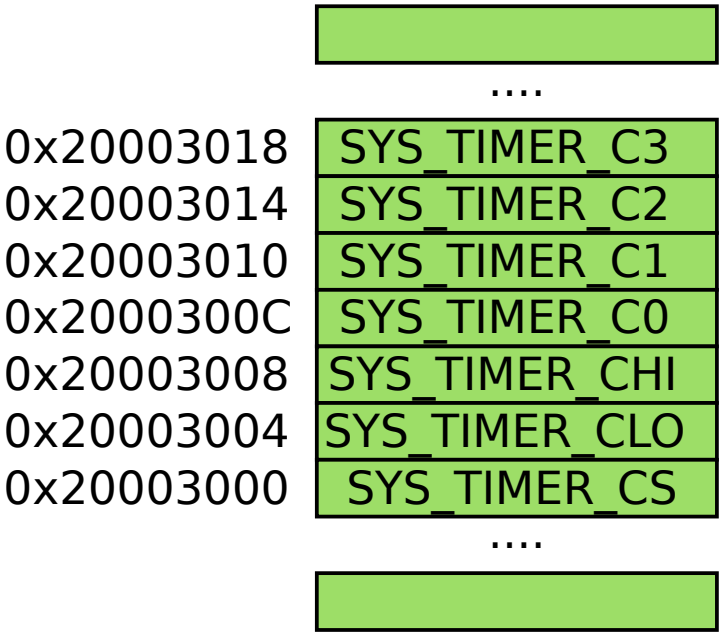
Dr.Ing.Steiner Guillermo

Timer

- La medición de el tiempo es importante en sistemas embebidos.
- Pueden ser mediciones obvias como la medición de tiempos de cocción en un microondas, horas de encendido de una lampara en un invernadero.
- U otras como controlar la velocidad en un motor, controlar la cantidad de tiempo de cocción en un microonda o cualquier sistema de control que necesite una discretización.

Timer en BCM2835

- Este periférico posee un system timer de 64 bits de corrida libre que incrementa cada 1uS (1 MHz).
- Luego 4 comparadores de 32 bits, permite comparar cada uno de estos valores con la parte baja del system timer.
- Cuando algún comparador coincide con el valor del timer principal, modifica una bandera.
- Los registros son:



Registros del Timer

- El system timer posee dos registros de solo lectura
  - **SYS_TIMER_CLO** con los 32 bits mas bajos.
  - **SYS_TIMER_CHI** con los 32 bits mas altos.
- Los comparadores están asignados a 4 registros
  - **SYS_TIMER_C0**
  - ..
  - **SYS_TIMER_C3**
- Cuando alguno de los comparadores coincide con el **SYS_TIMER_CLO**, el bit correspondiente se activa. Estos bit son los 4 bits menos significativos de **SYS_TIMER_CS**
  bit M0 .. M3

Multitarea y Timer

- Un problema cuando se usa algún proceso dependiente del tiempo es la latencia.
- Latencia es el tiempo desde que se produce la interrupción hasta que se ejecuta la primera instrucción de la rutina de tratamiento.
- Aquí la latencia se involucra de otra manera.

Problema

Se inicia un temporizado.
Durante la espera el SO cambia a otra tarea y luego a otra.
Durante la ejecución de esas tareas se produce el matching
No hay manera de medir o fijar el tiempo desde que se produce la comparación y cuando el SO ejecute la tarea que verifica la bandera.

Multitarea y Timer

- La solución es anular las interrupciones, de modo que el SO no pueda interrumpir la tarea.
- Esta solución impide que se restablezca el SO ante un error de la función.
- Solo se utiliza en situaciones criticas y se debe asegurar de restablecer las interrupciones una vez finalizada la espera critica.

Driver EasyPIO + Timer

- Se implementa una función que realice una demora (*delay*)
- Para esto se utiliza el canal 1, otros canales pueden ser utilizados por el procesador gráfico o el SO.
- Por no tener puertos de E/S no es necesario inicializar nada.
- Se presenta una función para demora en uS y otra en mS

```
void delayMicros(int micros) {
    SYS_TIMER_C1 = SYS_TIMER_CLO + micros;//se establece el reg de compara
    SYS_TIMER_CSbits.M1 = 1;              // Se borra las banderas
     while (SYS_TIMER_CSbits.M1 == 0);    // Esperar hasta el match
}

void delayMillis(int millis) {
    delayMicros(millis*1000);
}
```

Driver EasyPIO + Timer

- Un ejemplo sencillo para realizar un *blinking* en un led
- El led es el disponible GPIO47 de la Pi B+ y realiza 20 repeticiones a 5Hz.
- El programa anula las interrupciones y luego las habilita.

```
#include "EasyPIO.h"
void main(void) {
   int i;
   pioInit();
   pinMode(47, OUTPUT);  // Configura GPIO como salida
   NoInterrupts();       // deshabilita las interrupciones
   for (i = 0; i<20; i ++ ) {
      delayMillis(150);
      digitalWrite(47, 0); // Apaga el led
      delayMillis(50);
      digitalWrite(47, 1); // Enciende el led.
   }
   Interrupts();         // habilita las interrupciones
}
```

- Bibliografía

  Harris & Harris. Digital design and computer architecture: ARM edition. Elsevier, 2015. Capítulo 9.

  BCM2835-ARM-Peripherals.pdf

¿ Preguntas ?