Apellidos	Nombre

Universidad de Alcalá

Dpto Automática. 13-05-2022

Examen de las Prácticas 4 y 5- Sistemas Empotrados

Duración: 60 minutos.

1. Completar la siguiente función *new_init_timing_service_40MHz*, análoga a la función *init_timing_service* de la práctica 4. La diferencia introducida en esta función es que trabaja con el timer 1 y no con el timer 0, teniendo en cuenta que el *underflow* del timer 1 genera la interrupción externa 9. La nueva función, además, va a utilizarse en un procesador que tiene **un reloj de 40MHz, y no de 20MHz**, como ocurría en la función original. **(2.0 puntos)**

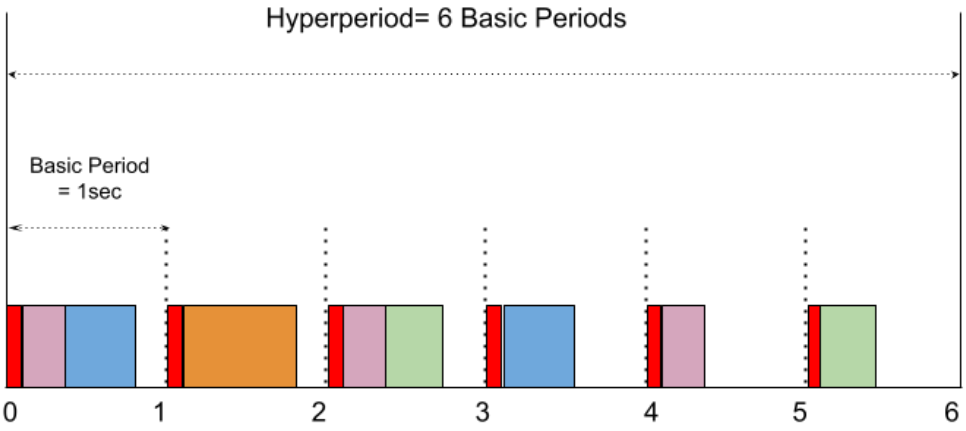
```
uint8_t new_init_timing_service_40MHz (uint8_t currentTime_in_Y2K)){
       //Deshabilitar interrupciones
       leon3_sys_call_disable_irqs();

       //enmascarar la irq del timer usando la macro TIMER_IRQ_LEVEL
       leon3_mask_irq(9);

       // Deshabilitar el timer usando la macro TIMER_ID
       leon3_timer_disable(1);

       //Deshabilitar la interrupción del Timer usando la macro TIMER_ID
       leon3_timer_disable_irq(1);

       //Configuración la TimerUnit:
         //-Salida del prescaler con frecuencia de 1 Hz
         //-Se habilita el freeze durante la depuración
         //-Se generan interrupciones separadas para los dos timers
       leon3_timerunit_set_configuration( 40-1, true , true );

       //Configuración del timer:
       leon3_timer_config(1, 1000000UL/TIMING_SERVICE_TICKS_PER_SECOND -1, false, true);

       //Instalar timertick_irq_handler como manejador de usuario de la interrupción del timer.
       leon3_install_user_hw_irq_handler(9, timertick_irq_handler);

       //Inicializar el reloj monotónico
       init_monotonic_clock(currentTime_in_Y2K);

       //Clear la interrupción externa del timer
       leon3_timer_clear_irq(1);

       //Desenmascarar la interrupción externa del timer.
       leon3_unmask_irq(9);

       //Habilitar la interrupción del timer
       leon3_timer_enable_irq(1);

       //Habilitar el timer
       leon3_timer_enable(1);

       //Habilitar interrupciones
       leon3_sys_call_enable_irqs();
       return error;
}
```

Otra posibilidad, aún más sencilla, era definir las macros y utilizar el mismo código que teníais en la práctica

```
#define LEON3_FREQ_MHZ                          40
#define TIMER_ID                                1
#define TIMER_IRQ_LEVEL                         9


uint8_t new_init_timing_service_40MHz (uint8_t currentTime_in_Y2K)){

        //Deshabilitar interrupciones
        leon3_sys_call_disable_irqs();


        //enmascarar la irq del timer usando la macro TIMER_IRQ_LEVEL
        leon3_mask_irq(TIMER_IRQ_LEVEL);

        // Deshabilitar el timer usando la macro TIMER_ID
        leon3_timer_disable(TIMER_ID);

        //Deshabilitar la interrupción del Timer usando la macro TIMER_ID
        leon3_timer_disable_irq(TIMER_ID);

        //Configuración la TimerUnit:
          //-Salida del prescaler con frecuencia de 1 Hz
          //-Se habilita el freeze durante la depuración
                  //-Se generan interrupciones separadas para los dos timers
        leon3_timerunit_set_configuration( LEON3_FREQ_MHZ -1, true , true );

        //Configuración del timer:
        leon3_timer_config(TIMER_ID, 1000000UL/TIMING_SERVICE_TICKS_PER_SECOND -1, false, true);

        //Instalar timertick_irq_handler como manejador de usuario de la interrupción del timer.
        leon3_install_user_hw_irq_handler(TIMER_IRQ_LEVEL, timertick_irq_handler);

        //Inicializar el reloj monotónico
        init_monotonic_clock(currentTime_in_Y2K);

        //Clear la interrupción externa del timer usando la macro TIMER_ID
        leon3_timer_clear_irq(TIMER_ID);

        //Desenmascarar la interrupción externa del timer. Usar la macro TIMER_IRQ_LEVEL
        leon3_unmask_irq(TIMER_IRQ_LEVEL);

        //Habilitar la interrupción del timer usando la macro TIMER_ID
        leon3_timer_enable_irq(TIMER_ID);


        //Habilitar el timer usando la macro TIMER_ID
        leon3_timer_enable(TIMER_ID);


        //Habilitar interrupciones
        leon3_sys_call_enable_irqs();

        return error;
}
```

2. Completar el siguiente código que implementa, mediante un ejecutivo cíclico, la ejecución de las rutinas `TSensorAdquition`, `TReactiveSystem`, `TTxRxData`, `TSensorFusion`, `TGuidance` de acuerdo a periodicidad de cada tarea, definida en la siguiente tabla. **Suponed que el tick de reloj del sistema tiene un periodo de 100 ms.** Emplead para la implementación las macros, funciones y la estructura de la función `main` utilizadas en la práctica 4.0 **(2.5 puntos)**

Task	Execution Time (sec)	Period (sec)
TSensorAdquisition	0.1	1
TReactiveSystem	0.3	2
TTxRxData	0.4	3
TSensorFusion	0.5	3
TGuidance	0.8	6




```
// Código de la tarea TSensorAdquisition
void TSensorAdquisition(void){

    leon3_print_string(" Do TSensorAdquisition\n");

    //Suponer que está completo el código de la tarea

}


// Código de la tarea TReactiveSystem
void TReactiveSystem(void){

    leon3_print_string(" Do TReactiveSystem\n");
    //Suponer que está completo el código de la tarea

}

// Código de la tarea TTxRxData
void TTxRxData(void){
```

```
    leon3_print_string(" Do TTxRxData\n");
    //Suponer que está completo el código de la tarea
}

// Código de la tarea TSensorFusion
void TSensorFusion(void){

    leon3_print_string(" Do TSensorFusion\n");
    //Suponer que está completo el código de la tarea
}

// Código de la tarea TGuidance
void TGuidance(void){

    leon3_print_string(" Do TGuidance\n");
    //Suponer que está completo el código de la tarea
}
```

```
//Definición de las macros de configuración
#define CYClIC_EXECUTIVE_PERIOD_IN_TICKS                10
#define CYClIC_EXECUTIVE_HYPER_PERIOD                       6
#define CYClIC_EXECUTIVE_TASKS_NUMBER                   5

void (*cyclic_executive[CYClIC_EXECUTIVE_HYPER_PERIOD][CYClIC_EXECUTIVE_TASKS_NUMBER+1](void)={
     {TSensorAdquisition,TReactiveSystem,TSensorFusion,NULL,NULL,NULL},
     {TSensorAdquisition,TGuidance,NULL,NULL,NULL,NULL},
     {TSensorAdquisition,TReactiveSystem,TTxRxData,NULL,NULL,NULL}
     {TSensorAdquisition,TSensorFusion,NULL,NULL,NULL,NULL},
     {TSensorAdquisition,TReactiveSystem,NULL,NULL,NULL,NULL},
     {TSensorAdquisition,TTxRxData,NULL,NULL,NULL,NULL}
};
```

También valdría ajustar el array al máximo número de tareas a ejecutar por ciclo

```
//Definición de las macros de configuración
#define CYClIC_EXECUTIVE_PERIOD_IN_TICKS                10
#define CYClIC_EXECUTIVE_HYPER_PERIOD                       6
#define CYClIC_EXECUTIVE_TASKS_NUMBER                   3

void (*cyclic_executive[CYClIC_EXECUTIVE_HYPER_PERIOD][CYClIC_EXECUTIVE_TASKS_NUMBER+1](void)={
     {TSensorAdquisition,TReactiveSystem,TSensorFusion,NULL},
     {TSensorAdquisition,TGuidance,NULL,NULL},
     {TSensorAdquisition,TReactiveSystem,TTxRxData,NULL}
     {TSensorAdquisition,TSensorFusion,NULL,NULL},
     {TSensorAdquisition,TReactiveSystem,NULL,NULL},
     {TSensorAdquisition,TTxRxData,NULL,NULL}
};
```

```
// Función principal
int main()
{
    //Install handlers for enable and disable irqs
    leon3_set_trap_handler(0x83,  leon3_trap_handler_enable_irqs);
    leon3_set_trap_handler(0x84 , leon3_trap_handler_disable_irqs);

    //Declaración de variables de control del ejecutivo cíclico
    uint8_t current_period=0;
    int task_index=0;
    uint64_t next_period_in_ticks_from_reset;

    //Inicialización del servicio de temporización y toma de referencia absoluta
    //del número de ticks desde el reset del sistema

    //Init Timing Service
    init_timing_service( date_time_to_Y2K(18, 3, 22, 0, 0, 0 ));

    //Get Absolute Time reference
    next_period_in_ticks_from_reset=get_tick_counter_from_reset();

    while(1){
       task_index=0; //poner a 0 al inicio de cada periodo básico

       leon3_print_string("\nStart period\n");
       print_date_time_from_Y2K(get_universal_time_Y2K());

       // Control de la ejecución de las tareas de cada período básico
       while(cyclic_executive[current_period][task_index]){
              cyclic_executive[current_period][task_index]();
              task_index++;
       }

       //Sincronización con el inicio del siguiente período básico
       //Update Absolute reference with next period
       next_period_in_ticks_from_reset+=CYClIC_EXECUTIVE_PERIOD_IN_TICKS;
       wait_until(next_period_in_ticks_from_reset);

       //Next basic period
       current_period++;
       if(current_period==CYClIC_EXECUTIVE_HYPER_PERIOD){

              current_period=0;
              leon3_print_string("\n****************\n");
              leon3_print_string("\nNext hyperperiod\n");
              leon3_print_string("\n****************\n");
       }

    }
    return 0;
}
```

3. Completar a continuación el código de las tareas `TSensorAdquition`, `TReactiveSystem`, `TTxRxData`, `TSensorFusion`, `TGuidance` de acuerdo a periodicidad de cada una de ellas, pero utilizando la primitiva `rtems_task_wake_after` de RTEMS tal como se hizo en la práctica 5_1 .

   Suponed que en RTEMS el tick de reloj tiene un periodo de 100 ms (2.0 puntos)

```
rtems_task TSensorAdquisition (rtems_task_argument unused) {

// Complete:

  while(1){   //o for(;;;){
     rtems_status_code status;
     puts("Do Sensor Adquisition");
     printf(" - rtems_ticks_since_boot - %i",get_ticks_since_boot());

     status=rtems_task_wake_after(10);
  }
}

rtems_task TReactiveSystem (rtems_task_argument unused) {

// Complete:
  while(1){ //o for(;;;){
     rtems_status_code status;
     puts("Do Reactive System");
     printf(" - rtems_ticks_since_boot - %i",get_ticks_since_boot());
     status=rtems_task_wake_after(20);
  }
}

rtems_task TTxRxData (rtems_task_argument unused) {

// Complete:
  while(1){  //o for(;;;){
     rtems_status_code status;
     puts("TTxRxData");
     printf(" - rtems_ticks_since_boot - %i",get_ticks_since_boot());
     status=rtems_task_wake_after(30);
  }
}


rtems_task TSensorFusion (rtems_task_argument unused) {

// Complete:
  while(1){ //o for(;;;){
     rtems_status_code status;
     puts("Do Sensor Fusion");
     printf(" - rtems_ticks_since_boot - %i",get_ticks_since_boot());
     status=rtems_task_wake_after(30);
  }
}
```

```
rtems_task TGuidance rtems_task_argument argument){


// Complete:
  while(1){   //o for(;;;){
     rtems_status_code status;
     puts("Do Guidance");
     printf(" - rtems_ticks_since_boot - %i",get_ticks_since_boot());
     status=rtems_task_wake_after(60);
  }
}
```

4. Completad de nuevo la implementación de las tareas `TSensorAdquition`, `TReactiveSystem`, `TTxRxData`, `TSensorFusion`, pero utilizando ahora la función `task_delay_until` que se utilizó en la práctica 5_2. **Suponed que el periodo del tick de reloj es también de 100 ms (2.0 puntos)**

Se requería usar delay_until tal como se ha explicado en clase, permitiendo evitar el problema de la deriva. (It was required to use delay_until as explained in class, allowing to avoid the drift/bias problem.

```
rtems_task TSensorAdquisition (rtems_task_argument unused) {
// Complete:
  rtems_interval ticks = get_ticks_since_boot();
  while(1){ //o for(;;;){
     puts("Do Sensor Adquisition");
     printf(" - rtems_ticks_since_boot - %i",get_ticks_since_boot());
     ticks+=10;
     task_delay_until(ticks);
  }
}

rtems_task TReactiveSystem (rtems_task_argument unused) {
// Complete:
  rtems_interval ticks = get_ticks_since_boot();
  while(1){ //o for(;;;){
     puts("Do Reactive System");
     printf(" - rtems_ticks_since_boot - %i",get_ticks_since_boot());
     ticks+=20;
     task_delay_until(ticks);
  }
}

rtems_task TTxRxData (rtems_task_argument unused) {
// Complete:
  rtems_interval ticks = get_ticks_since_boot();
  while(1){ //o for(;;;){
     puts("TTxRxData");
     printf(" - rtems_ticks_since_boot - %i",get_ticks_since_boot());
     ticks+=30;
     task_delay_until(ticks);
  }
}
```

```
rtems_task TSensorFusion (rtems_task_argument unused) {

// Complete:
  rtems_interval ticks = get_ticks_since_boot();
  while(1){ //o for(;;;){
     puts("Do Sensor Fusion");
     printf(" - rtems_ticks_since_boot - %i",get_ticks_since_boot());
     ticks+=30;
     task_delay_until(ticks);
  }
}

rtems_task TGuidance (rtems_task_argument unused){


// Complete:

  rtems_interval ticks = get_ticks_since_boot();
  while(1){ //o for(;;;){
     puts("Do Guidance");
     printf(" - rtems_ticks_since_boot - %i",get_ticks_since_boot());
     ticks+=60;
     task_delay_until(ticks);
  }

}
```

5. ¿Qué código tendría la tarea `TSensorAdquition` si **el tick de reloj que configura RTEMS es ahora de 10 ms**? Utilizar la función `task_delay_until` como en el apartado anterior**. (0.5 puntos)**

```
rtems_task TSensorAdquisition (rtems_task_argument unused) {

// Complete:
  rtems_interval ticks = get_ticks_since_boot();
  while(1){ //o for(;;;){
     puts("Do Sensor Adquisition");
     printf(" - rtems_ticks_since_boot - %i",get_ticks_since_boot());
     ticks+=100;  //Al ser el tick de 10ms, 1 seg= 100 ticks
     task_delay_until(ticks);
  }
}
```

6. Utilizad las primitivas de RTEMS que creas necesarias para crear la función get_seconds_since_boot que devuelve el número de segundos que han transcurrido desde el power-on del sistema **(1.0 puntos)**

```
uint32_t  get_seconds_since_boot(){
  rtems_interval ticks_per_second;
  rtems_interval ticks_since_boot;
  rtems_clock_get(RTEMS_CLOCK_GET_TICKS_SINCE_BOOT,&ticks_since_boot);
  rtems_clock_get( RTEMS_CLOCK_GET_TICKS_PER_SECOND,&ticks_per_second);
  return (ticks_since_boot / ticks_per_second);
}
```

Otra posibilidad, utilizando las funciones de la práctica 4:

```
uint32_t  get_seconds_since_boot(){
   return (get_ticks_since_boot()/ get_ticks_per_second());
}
```