Actividad TDS

1)

Funcionamiento:

La función **inicializar_despachador** setea las funciones para iniciar y detener el timer. Luego a través de la función **agragar_tarea** se agregan tres funciones (tarea_calculo, tarea_serie, tarea_led) y sus respectivos parámetros temporales.

Una vez hecho esto entro al loop infinito, donde reseteo el contador de ciclos de programa (CYCCNT). Luego, cada 2ms (tick del sistema) entro a la función **despachar_tareas**, donde se van a ir ejecutando las tareas de a una en el orden que fueron colocadas y respetando el periodo de ejecución. En esta función se verifica que el tiempo de ejecución de cada una sea mayor al mejor tiempo de ejecución (BCET) y menor al peor tiempo de ejecución (WCET) previamente seteados. En caso de que sea mayor a WCET, este adoptara el valor del último tiempo medido. También, si cualquiera de estas condiciones no se cumple entonces se ejecutará la tarea **falla_sistema**, que hará titilar un LED infinitamente.

La tarea que se ejecuta primero es **tarea_calculo**. Esta tarea chequea que se haya presionado el pulsador y que el puerto serie esté disponible. Si estas condiciones se cumplen se genera un numero aleatorio que luego es convertido a hash y guardado en una variable global. Por último, se levanta un flag para indicar que se puede enviar por puerto serie.

Luego se ejecuta la **tarea_serie**, en donde primero se verifica que se pueda enviar por puerto serie y que no se esté mandando nada. Si esto se cumple, se llena un buffer con el hash creado anteriormente (si es la primera vez que entra). Luego se manda cada byte del buffer a través de la UART (un byte por cada llamada a la tarea), hasta que se envía el ultimo y se resetean los flags.

La **tarea_led** cambia el estado de un LED luego de que se llama una cantidad determinada de veces (43) y se llama luego de llamar a tarea_serie, pero esta tiene un periodo 3 veces mayor a las otras, ósea que las otras 2 funciones se ejecutaran 3 veces antes de que se ejecute tarea_led.

2)

Hay tres tareas y existe intercomunicación entre ellas ya que la **tarea_calculo** genera el hash que luego es enviado en **tarea_serie**. El hash es la variable global que es usada por estas dos tareas.

3)

El tick del sistema es de 2ms, mientras que el Hiperperiodo es de 3*2ms = 6ms. El tick del sistema está definido por la macro TICKS_SISTEMA

$T_{clk} * prescaler = \frac{1}{72MHz} * 71 = \mathbf{986.1\ nS}$ (es lo que se ve en el programa: TIM2->PSC = (SystemCoreClock / (1000000*divisor_us)) - 1;)

Los ciclos secundarios serán de $\boldsymbol{2mS}$

La carga media del CPU se calcula como:

$$U = \sum_{1}^{3} \left(\frac{C_i}{T_i}\right)$$

Donde Ci es el WCET y Ti es el periodo de cada tarea. Habiendo ejecutado el programa, obtuve los WCET de cada tarea:

lista_tareas	TaskStat [8]	[8]
lista_tareas[0]	TaskStat	{...}
(x)= bcet	int	0
(x)= wcet	int	60000
(x)= et	int	3
(x)= et_wcet	int	3196
(x)= offset	int	0
(x)= period	int	1
(x)= task	void (*)(void *)	0x8000315 <tarea_calculo>
(x)= param	void *	0x0
lista_tareas[1]	TaskStat	{...}
(x)= bcet	int	0
(x)= wcet	int	60000
(x)= et	int	1
(x)= et_wcet	int	144
(x)= offset	int	0
(x)= period	int	1
(x)= task	void (*)(void *)	0x80003d1 <tarea_serie>
(x)= param	void *	0x0
lista_tareas[2]	TaskStat	{...}
(x)= bcet	int	0
(x)= wcet	int	60000
(x)= et	int	1
(x)= et_wcet	int	2
(x)= offset	int	1
(x)= period	int	3
(x)= task	void (*)(void *)	0x80004ed <tarea_led>
(x)= param	void *	0x0

Viendo la tabla podemos determinar el WCET de cada tarea:

tarea_calculo: 3196*1uS = 3.196 mS

tarea_serie: 144*1uS = 0.144 mS

tarea_led: 2*1uS = 2 uS

tick 1: $\frac{3.196\ \text{mS}}{2\ mS} = 159{,}8\%$

tick 2: $\frac{0.144\ \text{mS}}{2\ mS} = 7{,}2\%$

tick 3: $\frac{2\ \text{uS}}{2\ mS} = 0{,}1\%$

$$U = \frac{3.196\ \text{mS}}{2\ mS} + \frac{0.144\ \text{mS}}{2\ mS} + \frac{2\ \text{uS}}{6\ mS} = 1.671 > 1$$

4) Podemos ver que el sistema no está planificado correctamente ya que el WCET de la primera tarea es mayor que cada tick del sistema.

Si se quiere que la carga media sea menor al 75% se debe hacer que el tick del sistema sea de al menos 5 ms.

5) En esta aplicación se utiliza una tarea multiciclo para hacer la transmisión por puerto serie, ya que no se envían todos los datos en una sola llamada a tarea_serie. Si se hiciera de esa forma, el WCET sería mucho mayor y el tick del sistema debería aumentar también. Para nuestra aplicación no hay problema con esta implementación ya que la velocidad de transmisión es configurable, en nuestro caso es de 4800 bauds.

6) Ahora se utilizó el timer 3 para implementar demoras “sándwich”. Se agrego a la función despachar_tarea lo siguiente:

```
int despachar_tarea(TaskStat *estado, uint32_t i)
{
    int ret = 0;
    int valor_us;
    uint16_t wcet [] = {3200,150,3};


    if (!estado->offset)
    {
        sandwich_delay_T3_start(wcet[i]);//me aseguro de que cada tarea termine siempre en el mismo momento
        monitor_start();

        estado->offset = estado->period - 1;
        estado->task(estado->param);

        sandwich_delay_T3_wait();
        valor_us = monitor_stop();//siempre dara el mismo tiempo

        estado->et = valor_us;
        if (valor_us < estado->bcet || valor_us > estado->wcet)
            ret--;
        if (estado->et_wcet < estado->et)
            estado->et_wcet = estado->et;

    }
    else
    {
        estado->offset--;
    }


    return ret;
}
```

El vector WCET tiene los datos de los correspondientes peores tiempos de ejecución (obtenidos anteriormente) y además se le pasa el índice de la tarea a ejecutar. De esta forma nos aseguramos de finalizar cada tarea siempre en el mismo periodo de tiempo, y así ecualizar la duración de las tareas.

Ahora siempre ejecutaremos todas las tareas en la misma cantidad de ticks (aproximadamente 242150), que pasado a segundos es: $t_{ejec} = 242150 * \left(\frac{1}{72MHz}\right) = 3.363\ mS$, lo cual es lo esperado ya que ahora tenemos un tick cada 5 ms.

(x)= ticks	uint32_t	242139