Sistemas de Tiempo Real

Práctica 2: Planificación dinámica

Fecha:
23 de diciembre de 2022

Alumnos:
Álvaro Morata Hontanaya (100405846)
Carlos Montero Gómez de las Heras (100405884)
Ángel Daniel Pinheiro Cabrera (100428986)

Tabla de contenidos

Parte 1: Fichero codificado con 4.000 muestras de 1 bit por muestra

Arduino

Para la realización de la planificación de las tareas del módulo de Arduino se ha hecho uso del Timer 1 para programar la tarea playBit() como una interrupción. A continuación se muestra una descripción de las tareas programadas:

- **playBit (Tarea 1):** Dado que se va a reproducir un solo bit por muestra, se aplica una máscara a cada byte recibido de modo que sólo un 1 o un 0 sea el resultado para pasar a la función digitalWrite. En caso de que el altavoz esté muteado (muted == 1) digitalWrite escribirá siempre un 0. Esta función se activa mediante el *Timer 1* (interrupciones).
- **muteLed(Tarea 2):** Esta función recibe la señal de pulsación del switch de la placa, y actua como un toggle de la variable muted dentro del código. Tras cambiar el estado dependiendo de las pulsaciones del botón, hace que el led brille dependiendo del estado de la variable mute (0 = apagado, 1 = encendido). Esta tarea se realiza cada 250 microsegundos.

INTERRUPCIÓN ARDUINO

La reproducción de cada fragmento de muestreo se realiza mediante un evento de interrupción en el TIMER 1.
Necesitamos que el evento se realice 4000 veces por segundo, o lo que es lo mismo, cada 250 microsegundos. Necesitamos encontrar una configuración de preescalado + registros de comparación que encaje con esta frecuencia.
Teniendo una cpu de 16000000 Hz, preescalaremos a 8, para tener ciclos de 2 GHz.
Con un preescalado de 8 y un registro de comparación en 499 (0-499 = 500 ciclos) obtenemos un tiempo de 500*1/2000000 = 250 microsegundos, justo el tiempo que necesitamos.

$CPU = 16\,GHz$

$PREESCALADO = 8$

$f_{ciclo} = \frac{16}{8} = 2\,GHz$

$T_{ciclo} = \frac{1}{2 \cdot 10^6} = 0{,}5\,\mu s$

$250\,\mu s = T_{ciclo} \cdot Reg$

$Reg = 500$

$[0, 499]$

Para aplicar la configuración deseada, modificaremos TCCR1 y el registro de comparación OCR1A:

```
OCR1A = 499;
TCCR1A = _BV(COM1A0);
TCCR1B = _BV(WGM12) | _BV(CS11);
```

Ciclos de tareas arduino

A continuación se muestra una tabla con las tareas y los respectivos tiempos de ciclo.

Tarea	**Tiempo de ciclo**
Tarea 1: playBit	Se activa mediante evento (interrupción)
Tarea 2: muteLed	250 microsegundos

RTEMS

Se han implementado las 3 tareas siguientes:

- task_pause(): se realiza el cambio de la variable paused según se quiera pausar o no la canción.
- read_music(): se pasa el buffer de la canción al arduino.
- status_music(): se muestra el estado de la reproducción.

Tarea	**Periodo(ms)**	**Prioridad**
Tarea 1: read_music	512	3
Tarea 2: task_pause	2000	2
Tarea 3: status_music	5000	1

Las prioridades han sido asignadas por el periodo, atendiendo al periodo, aquellas con uno mayor serán realizadas posteriormente. Es decir, las tareas con menor periodo tienen mayor prioridad.

Parte 2: Fichero codificado con 4.000 muestras de 8 bit por muestra

Arduino

Para la parte dos mantenemos el mismo código que en la parte 1, con el TIMER 1 actuando como activador del evento que escucha el botón de mute y silencia el sonido si es activado.

Pero esta vez en vez de guardar el bit a reproducir en una variable y escribirlo mediante digitalWrite() el sonido se reproducirá mediante un PWM generado por el TIMER 2 y se cambiará la nota modificando el registro correspondiente (OCR2)

```
// parte II
TCCR2A = _BV(COM2A1) | _BV(WGM21) | _BV(WGM20);
TCCR2B = _BV(CS20);
OCR2A = 0;
```

Usaremos OCR2A como registro donde almacenaremos el bit a reproducir a continuación.

El resto del funcionamiento es idéntico al del apartado uno, solo que teniendo en cuenta que ya no usaremos una variable para almacenar el siguiente bit a reproducir.

Ciclos de tareas arduino

A continuación se incluye la tabla para los tiempos de ciclo del módulo de Arduino de esta segunda parte.

Tarea	**Tiempo de ciclo**
Tarea 1: playBit	Se activa mediante evento (interrupción)
Tarea 2: muteLed	250 microsegundos

RTEMS

Se han implementado las 3 tareas siguientes:

- task_pause(): se realiza el cambio de la variable paused según se quiera pausar o no la canción.
- read_music(): se pasa el buffer de la canción al arduino.
- status_music(): se muestra el estado de la reproducción.

Tarea	**Periodo(ms)**	**Prioridad**
Tarea 1: read_music	64	3
Tarea 2: task_pause	2000	2
Tarea 3: status_music	5000	1

Las prioridades han sido asignadas por el periodo, atendiendo al periodo, aquellas con uno mayor serán realizadas posteriormente. Es decir, las tareas con menor periodo tienen mayor prioridad.

Problemas encontrados

Para la parte del RTEMS la comprobación no ha sido posible debido a diversos problemas encontrados a la hora de reconfigurar todo el módulo RTEMS para esta práctica, se realizó el código previo a las pruebas pero una vez hecho y compilado, la build y configuración tanto de RTEMS como de Qemu han resultado en errores imprevistos y fatales para la correcta progresión del módulo de software. Esto ha influido negativamente a la hora de corroborar que el diseño realiza lo que se pretende y a la hora de poder validar el planificador por propiedades ya que no ha sido posible recuperar los tiempos de cómputo ni de cerrojo de los elementos y tareas correspondientes.