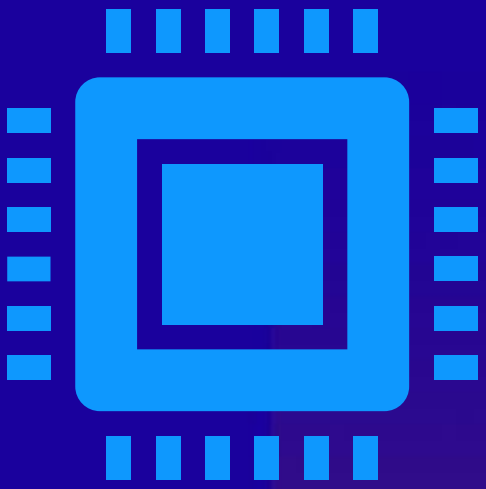
PRÁCTICA 1B

2023

Antonio Aparicio González

Daniel Díaz Martel



INTERNET DE LAS COSAS

ÍNDICE DE CONTENIDOS

Introducción

La práctica se desarrolla sobre una placa Arduino MKR 1310, un dispositivo de pequeñas dimensiones está equipado con una serie de componentes que lo hacen idóneo para una amplia gama de aplicaciones relacionadas con IoT, como por ejemplo:

- **Microcontrolador Cortex M0+ SAMD21G:** Controlador principal que gestiona todas las operaciones y lógicas del dispositivo.
- **RTC de 32 bits:** Un Reloj en Tiempo Real (RTC) integrado, esencial para el seguimiento preciso del tiempo y la fecha en el sistema.
- **Memoria Flash W25Q16:** Un chip con 2 MBytes de memoria FLASH, que se utiliza para almacenar datos de manera persistente.

Objetivos de la Práctica

La tarea principal consiste en desarrollar un firmware para el Arduino MKR 1310 que cumpla con los siguientes objetivos:

A partir del último código de ejemplo contemplado en el guión de prácticas “Ejemplo 3: Estimación de la carga computacional” hay que realizar un esquema de monitorización que permita balancear la carga computacional en el microprocesador y llevarla a una condición “óptima”, definida como aquella en la que la utilización del procesador no supera el 85% (15% del tiempo en la tarea idle/loop) y las tareas se ejecutan respetando el periodo establecido, pero con el máximo posible de intensidad computacional.

Teniendo en cuenta algunos detalles:

- A diferencia del ejemplo anterior, en el que usábamos una intensidad fija para cada hebra o tarea, definida mediante el vector threadLoad[NUM_THREADS], ahora cada tarea dispondrá de un rango ajustable de intensidad o carga computacional y será necesario declarar un valor mínimo y otro máximo.
- Se recomienda comenzar verificando que se respetan las frecuencias de operación de cada hebra de trabajo con las hebras configuradas al valor mínimo de intensidad o carga (mínimo consumo de tiempo de procesador).
- Una vez “estabilizado” el periodo/frecuencia de repetición de las hebras, la hebra top puede ajustar de forma cíclica la intensidad de carga de cada hebra y evaluar la nueva configuración verificando que: a) Se respetan las frecuencias de repetición de las hebras b) No se supera el umbral de carga del sistema En caso contrario, deberá reducirse la intensidad computacional de las hebras y revaluar la situación.

Extra

- Contemplar la posibilidad de que el tiempo de procesador de cada hebra pueda incluir un cierto porcentaje de carga variable (aleatoria). En este supuesto, el margen que se ha reservado de tiempo de CPU debería servir para absorber estos “picos” momentáneos de carga. Es evidente que este supuesto exige cierta tolerancia o histéresis para distinguir sobrecargas puntuales de otras situaciones de sobrecarga permanente, entendiendo como sobrecarga aquella situación en la que el uso de procesador sobrepasa el umbral establecido (p.e. del 85%).

Funcionamiento y Partes del Programa

Variables Globales y Estructuras

- **thread_name**: Array bidimensional de caracteres que almacena los nombres de los hilos en el sistema. Su tamaño es determinado por **NUM_THREADS** y el tamaño máximo de nombre de hilo (15 caracteres en este caso).
- **threadPeriod_ms**: Array de enteros sin signo que define los períodos de cada hilo en milisegundos. Los valores son inicializados de acuerdo a la constante **CYCLE_MS** y otros valores específicos.
- **threadMinLoad** y **threadMaxLoad**: Arrays de enteros volátiles que definen los valores mínimo y máximo de carga para cada hilo, respectivamente.
- **threadLoad**: Array de enteros volátiles que almacena la carga actual de cada hilo. Inicialmente, se establece igual a **threadMinLoad**.
- **threadEffectivePeriod_ms** y **threadCycle_ms**: Arrays de enteros sin signo volátiles que almacenan, respectivamente, el período efectivo y el tiempo de ciclo de cada hilo.
- **period_difference**: Array de enteros con signo volátiles que define el estado de conformidad de las hebras con su período establecido.
- **sysLoad**: Variable de tipo **systemLoad_t** que almacena la información de carga del sistema, incluyendo la carga de cada hilo y la carga en estado inactivo.
- **systemLoad_t**: Estructura que contiene un Array de **threadLoad_t** para cada hilo y un entero sin signo para la carga en estado inactivo.
- **threadLoad_t**: Estructura que almacena la información de carga de un hilo específico, incluyendo punteros al hilo, tiempos de muestra, ticks totales y por ciclo, y la carga por ciclo.

Macros

- **EXTRA**: Define si se activa el apartado extra, que implica carga de trabajo aleatoria. Establecido en **true** para activar.
- **USE_DOUBLE**: Define si se utiliza precisión doble para los cálculos matemáticos. Establecido en **false** para usar precisión simple.
- **CYCLE_MS**: Define el período base para los hilos en milisegundos.
- **NUM_THREADS**: Define el número total de hilos en el sistema.
- **LOAD_THRESHOLD**: Define el umbral de carga de trabajo como 85%.
- **BALANCED_STATUS**: Calcula el estado equilibrado de carga de trabajo dividiendo el umbral por la cantidad de hilos de trabajo.
- **TOTAL_SYSTEM_LOAD**: Calcula la carga total de trabajo del sistema, restando la carga en estado inactivo del 100%.

Funciones de las hebras

1. Hebra de Máxima Prioridad o Monitor del Sistema (top)

Esta hebra se encarga de la estimación de la carga del sistema. Se ejecuta periódicamente con una alta prioridad para recopilar información de todas las hebras, procesar esta información para equilibrar la carga del sistema y mostrar estadísticas. Además, controla el estado de un LED como indicador visual.

- **Variables**:
  - **lastTime_i**: Almacena el último momento en el que se ejecutó la hebra.
  - **period_i**: El período de la hebra, convertido a ticks del sistema.
  - **ledState**: Estado actual del LED.
- **Funciones Principales**:
  - **collectThreadTicks()**: Recoge los ticks consumidos por cada hebra.
  - **balanceLoad()**: Equilibra la carga de las hebras si es necesario.
  - **printThreadStats()**: Imprime estadísticas del sistema.
  - **toggleLedState()**: Cambia el estado del LED.
- **Comportamiento**: Espera durante su período establecido, recoge ticks, balancea la carga, imprime estadísticas y cambia el estado del LED.

2. Hebra Trabajadora(worker)

Esta hebra realiza cálculos matemáticos pesados para simular una carga de trabajo. Se ejecuta periódicamente y su carga de trabajo puede ser ajustada dinámicamente.

- **Variables**:
  - **worker_ID**: Identificador único de la hebra.
  - **period_i**: El período de la hebra, convertido a ticks del sistema.
  - **deadline_i**: Momento en el que la hebra debe terminar su ejecución actual.
  - **lastBeginTime_i**: Marca de tiempo del inicio de la última ejecución.
  - **beginTime_i**: Marca de tiempo del inicio de la ejecución actual.
  - **num**: Variable utilizada para los cálculos, su tipo depende de la macro **USE_DOUBLE**.
  - **niter**: Número de iteraciones de cálculo, basado en la carga de la hebra.
- **Funciones Principales**: Realiza cálculos matemáticos y ajusta su período de ejecución basado en la carga asignada.
- **Comportamiento**: Calcula su período efectivo, realiza cálculos, actualiza estadísticas de tiempo y duerme hasta su próximo período si es necesario.

Funciones Auxiliares para las hebras

En el programa desarrollado para esta práctica, se han creado una serie de funciones auxiliares que permiten obtener un código más modular y organizado. A continuación se aporta una descripción un poco más detallada:

1. **printThreadLoad():**
   - **Descripción**: Esta función imprime en el puerto serie la carga de trabajo de las hebras worker_1, worker_2 y worker_3, indicando si están en estado "S" (Stable) o "U" (Unstable). En caso de estar Unstable, se indica mediante un + o un – si el periodo efectivo de la hebra es superior o inferior al deseado.
   - **Uso**: Se llama desde la función top() para mostrar el estado de carga de las hebras.
2. **printThreadStats(uint32_t accumTicks):**
   - **Descripción**: Esta función imprime en el puerto serie estadísticas detalladas de cada hebra. Muestra el nombre de la hebra, los ticks consumidos en el último ciclo, el porcentaje de CPU utilizado y la duración del ciclo. Realiza además la llamada a **printThreadLoad().**
   - **Uso**: También se llama desde la función top() para mostrar estadísticas detalladas de las hebras.
3. **checkPeriodDifference():**
   - **Descripción**: Esta función ajusta la carga de trabajo de las hebras trabajadoras en función de la diferencia entre su periodo efectivo y su periodo objetivo. Calcula esta diferencia para cada hebra y modifica su carga para intentar alinear el periodo efectivo con el objetivo, respetando los límites de carga mínima y máxima. Actualiza el array **period_difference**.
   - **Uso**: Se llama desde la función **balanceLoad()** para determinar si las hebras están operando dentro de los límites establecidos.
4. **tryIncreaseAllThreadsLoad():**
   - **Descripción:** Examina la carga de trabajo por ciclo de cada hebra trabajadora y la ajusta con el objetivo de balancear la carga entre ellas. Si la carga de una hebra supera el estado balanceado definido por **BALANCED_STATUS,** se intenta disminuir su carga. En caso contrario, se incrementa, añadiendo un valor aleatorio entre 1 y 15 si **EXTRA** está activado, o simplemente 1 si no lo está. Todo esto respetando los límites mínimos y máximos definidos para cada hebra.
   - **Uso:** Se invoca desde **balanceLoad()** para redistribuir la carga de trabajo entre las hebras.
5. **decreaseAllThreadsLoad():**
   - **Descripción**: Esta función disminuye la carga de todas las hebras en una unidad.
   - **Uso**: Se llama desde la función balanceLoad() cuando la carga total es igual o superior al umbral establecido.

6. **balanceLoad():**
    - **Descripción**: Esta función determina la estabilidad de los periodos, intenta aumentar la carga si la carga total es baja y disminuye la carga si la carga total es alta.
    - **Uso**: Se llama desde la función top() para realizar el balanceo de carga.
7. **initializeLed():**
    - **Descripción**: Esta función inicializa el pin del LED incorporado y lo configura como salida.
    - **Uso**: Se llama desde la función top() para inicializar el LED.
8. **toggleLedState(bool ledState):**
    - **Descripción**: Esta función cambia el estado del LED. Si el estado actual es HIGH, lo cambia a LOW y viceversa.
    - **Uso**: Se llama desde la función top() para alternar el estado del LED.
9. **collectThreadTicks(systime_t lastTime_i):**
    - **Descripción**: Esta función recolecta los ticks consumidos por cada hebra durante el último ciclo y calcula la carga de trabajo.
    - **Uso**: Se llama desde la función **top()** para obtener información sobre la carga de las hebras.

Funciones Principales: setup() y loop()

El programa no sigue la estructura típica de un programa de Arduino, que incluye las funciones **setup()** y **loop()**. En lugar de eso, se basa en la biblioteca ChibiOS para gestionar las hebras y la planificación de tareas. Vamos a discutir cómo funciona en términos generales:

- **chSetup()**:
  Esta es la función que se encarga de la configuración inicial del sistema. Se llama después de inicializar el sistema operativo ChibiOS con **chBegin()**. Verifica la configuración del sistema operativo, imprime el valor de **CH_CFG_TIME_QUANTUM**, y crea e inicia las hebras de trabajo, incluida la hebra de alto nivel (top) y las tres hebras trabajadoras (workers).
- **setup()**:
  Esta función es llamada una vez al inicio de la ejecución del programa. Inicializa el pin LED en la placa, establece la comunicación serial y espera una entrada del usuario antes de inicializar el sistema operativo ChibiOS mediante la llamada a **chBegin(chSetup)**. Esta función nunca retorna y tras su ejecución, el control se traslada a la función **loop()**.
- **loop()**:
  En el contexto de Arduino, la función **loop()** es el bucle principal del programa. Se ejecuta continuamente después de que **setup()** haya finalizado. Sin embargo, en este programa funciona como la hebra de inactividad (idle thread) del sistema. No realiza ninguna acción y su ejecución indica que el sistema no tiene otras tareas para ejecutar y está en un estado inactivo.