Internet de las Cosas

Práctica 1.A

Introducción

Esta primera práctica se estructura en dos bloques. La primera parte, objeto de este guión de prácticas, estará orientada a presentar el dispositivo físico sobre el que basaremos las prácticas de esta asignatura, así como a introducir de manera sumaria algunos de los entornos de desarrollo disponibles para la misma. Abordaremos también la programación de algunos recursos de interés que se encuentran integrados en la placa de desarrollo, tales como el reloj de tiempo real (RTC), el chip de memoria FLASH, etc.

El segundo bloque estará orientado a presentar el uso de un sistema operativo de tiempo real, específicamente diseñado para microcontroladores.

Con el objetivo de mantener la extensión de este guion de prácticas dentro de unos límites razonables, se usará un estilo compacto para el texto y se emplearán con frecuencia referencias externas y ejemplos dirigidos. Por esta misma razón, no será posible abordar una descripción detallada de cada uno de los recursos hardware disponibles en el microcontrolador y, siempre que sea posible, se usarán librerías previamente desarrolladas con el objetivo de nivelar la curva de aprendizaje del uso de estos elementos.

Objetivos

- Conocer la arquitectura funcional de la tarjeta Arduino MKR 1310
- Presentar algunos de los entornos de desarrollo disponibles para el MKR 1310
- Resolver pequeñas tareas de programación con el MKR 1310. Dichas tareas se focalizarán en la programación de algunos dispositivos de interés, RTC, watchdog, memoria FLASH, …, disponibles en esta placa.

Arduino MKR 1310

Basaremos las prácticas de este curso en el Arduino MKR 1310. Se trata de placa de pequeñas dimensiones que incluye el microcontrolador Cortex M0+ SAMD21G como controlador principal, el módulo LoRA Murata CMWX1ZZABZ para comunicaciones inalámbricas, un chip criptográfico ECC508 y un chip W25Q16 de memoria FLASH de 2MByte. La tarjeta incluye también un módulo BQ24195L para controlar la recarga de una batería de ión-litio o polímero de litio.

El procesador principal SAMD21G puede operar hasta 48 MHz. Ofrece 256 KB de memoria Flash (programa), 32 KB de memoria SRAM, hasta 5 contadores/temporizadores de 16 bits, hasta 3 contadores/temporizadores de 24 bits, 12 bit 350ksps ADC, 10 bit 350ksps DAC, 12 canales DMA, watchdog y un RTC de 32 bits. Incorpora también 6 puertos series que pueden proyectarse sobre diferentes pines del micro para ofrecer diferentes puertos y buses seriales (UART, I^2C o SPI).

El Arduino MKR 1310 **soporta hasta una tensión de 3.3VDC** en todos sus pines de entrada/salida (E/S). Conectar cualquiera de sus pines a un tensión superior, por ejemplo 5VDC, puede dañarla de forma definitiva.

Recursos de interés:

- Página de producto del Arduino MKR 1310
- Manual de usuario del MKR 1310
- Manual de Microchip del microcontrolador SAMD21G
- Pinout del MKR 1310
- Esquema de la placa

Entornos de desarrollo

Existe un amplio número de alternativas a la hora de escoger un entorno de programación para el Arduino MKR 1310. La preferencia puede estar determinada por criterios de simplicidad de uso, el espacio en disco requerido por la instalación, si incluye o no un herramientas de depuración, si es posible usar un cierto editor, o si la interfaz es de tipo gráfico o es posible usarla en modo línea de comandos (CLI).

En esta introducción nos centraremos en cuatro de las opciones más populares.

- Arduino IDE 1.8.19: Es la última versión del IDE de Arduino “clásico”. Disponible para los principales sistemas operativos, Arduino lo considera “legacy”, es decir, obsoleto. Incluye una interfaz gráfica muy simple de usar, pero carece de cualquier tipo de soporte para depuración (debug) hardware.
- Arduino IDE 2.0.0: Versión actual del IDE de Arduino, es una evolución de la interfaz gráfica 1.X.X. Conserva el estilo minimalista típico de Arduino, pero incorpora algunos cambios y mejoras como la integración del Serial Plotter o diferentes opciones de depuración hardware. Es claramente la opción a futuro dentro de las opciones proporcionadas por Arduino.
- Arduino CLI: Arduino ofrece también la posibilidad de programar y depurar a través de una interfaz en línea de comando. Puede ser una opción interesante para los que necesiten un mayor control durante el proceso de compilación y carga del firmware.
- PlatformIO: Ofrece un entorno integrado de programación con una interfaz gráfica potente. Es una opción muy popular que admite ajustar múltiples preferencias que abarcan desde el editor, hasta el entorno de programación (librerías) con las que se puede desarrollar para una

cierta tarjeta. El número de tarjetas y de entornos de programación soportados es muy amplio e incluye herramientas para depuración hardware. Posiblemente, es el entorno más potente y desarrollado.

Es necesario advertir que, si bien el Arduino MKR1310 viene preparado para que se pueda soldar un conector JTAG/SWD (estándar de depuración), actualmente carecemos de los módulos hardware de depuración. Pueden escoger el entorno de su preferencia.

Ejemplos

Abordaremos ahora el desarrollo de una serie de pequeñas aplicaciones que tienen como objetivo poner en valor aspectos del hardware disponible en el Arduino MKR 1310 y aprender cómo usarlos.

Un “hola mundo” por interrupción

Este primer ejemplo muestra como es posible usar interrupciones para generar una variante del clásico “Hola Mundo” con el que se comienzan siempre todo tipo de cursos de programación. En este programa se usan interrupciones generadas por flanco de bajada en el pin digital 5 (`INTERRUPT_PIN`). Este pin se ha configurado en modo pull-up y se provoca el flanco de bajada (`FALLING`) cuando se lleva a tierra. La macro `WITH_HYSTERESIS`, definida por defecto contribuye a eliminar múltiples interrupciones por transitorios en el pin `INTERRUPT_PIN`.

```
/* ------------------------------------------------------------------------
 *  Ejemplo 1:
 *  - Un ejemplo sencillo sobre cómo programar una rutina de atención
 *    de interrupciones
 *  Asignatura (GII-IoT)
 * ------------------------------------------------------------------------
 */
// Descomentar para impedir múltiples interrupciones por rebote
#define WITH_HYSTERESIS

#if defined(WITH_HYSTERESIS)
  const uint32_t inhibitionTime_ms = 250;
#endif

#define ellapsedTime_ms(since_ms) (uint32_t)(millis() - since_ms)

#define INTERRUPT_PIN     5

// Declaramos la variable bandera como volátil y la inicializamos
volatile uint32_t int_flag = 0;
```

```
// ----------------------------------------------------------------------------
// ----------------------------------------------------------------------------
void setup()
{
  SerialUSB.begin(9600);
  while(!SerialUSB) {;}

  // Ajustamos el modo del INTERRUPT_PIN
  // Lo configuramos en modo pullup para que se produzca una
  // interrupción por flanco de bajada (FALLING) al llevarl a tierra
  pinMode(INTERRUPT_PIN, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), my_isr, FALLING);

  SerialUSB.println("\nLooping ...");
}

// ----------------------------------------------------------------------------
// ----------------------------------------------------------------------------
void loop()
{
  #if defined(WITH_HYSTERESIS)
    static uint32_t last_int_ms = 0;
    static boolean int_enabled = true;
  #endif

  if (int_flag) {

    #if defined(WITH_HYSTERESIS)
      detachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN));
      last_int_ms = millis();
      int_enabled = false;
      int_flag = 0;
    #else
      int_flag--;
    #endif
    SerialUSB.print("At ");
    SerialUSB.print(millis());
    SerialUSB.print(": Hello world!");
    SerialUSB.print(" --- remaining: ");
    SerialUSB.println(int_flag);
  }

  #if defined(WITH_HYSTERESIS)
    if ((!int_enabled) &&
        (ellapsedTime_ms(last_int_ms) >= inhibitionTime_ms) &&
```

```
        (digitalRead(INTERRUPT_PIN) == LOW)) {
      attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), my_isr, FALLING);
      int_enabled = true;
    }
  #endif
}

// ----------------------------------------------------------------------------
// Rutina de servicio de la interrupción
// ----------------------------------------------------------------------------
void my_isr()
{
  int_flag++;
}
```

Programación del reloj de tiempo real (RTC)

El procesador SAMD21G, microcontrolador principal en la tarjeta MKR 1310, incluye un reloj de tiempo real (Real Time Clock, RTC). Se trata de un contador de 32 bits que puede vincularse a diferentes osciladores (relojes del microcontrolador) y usarse como contador de tiempo o "reloj". Se puede emplear para "despertar" o sacar el microcontrolador de un estado de baja energía (sleep modes). Evidentemente, el RTC puede seguir operando cuando el SAMD21G entra en un estado de baja energía (**sección 16 del manual** del microcontrolador).

En función del oscilador al que se conecte y cómo se programe (ajuste del pre-escalado) puede usarse como un contador ordinario de tiempo, con una resolución de segundos) y calendario, o como contador de tiempo con una resolución inferior a un milisegundo. Para más información consultar la **sección 19 del manual** del microcontrolador.

En esta introducción abordaremos el uso de este dispositivo como reloj-calendario usando la librería RTCZero. El ejemplo 2 muestra cómo es posible poner en hora y fecha el RTC, y cómo consultarlo.

Nótese que esta forma de consultar la hora y la fecha no garantiza una lectura "atómica" o protegida de la hora o la fecha. Es decir, en circunstancias excepcionales podríamos leer una hora o una fecha que fueran incoherentes.

```
/* ------------------------------------------------------------
 *  Ejemplo 2: Puesta en hora del RTC
 *  - Requiere la librería RTCZero
 *    https://www.arduino.cc/reference/en/libraries/rtczero/
 *  Asignatura (GII-IoT)
 * ------------------------------------------------------------
 */
#include <RTCZero.h>

// rtc object
RTCZero rtc;
```

```
// Ponemos una hora y fecha arbitrarias
const uint8_t seconds = 50;
const uint8_t minutes = 59;
const uint8_t hours = 23;
const uint8_t day = 31;
const uint8_t month = 12;
const uint8_t year = 22;

// ----------------------------------------------------------------------------
// ----------------------------------------------------------------------------
void setup()
{
  SerialUSB.begin(115200);

  // Habilitamos el uso del rtc
  rtc.begin();

  // Fijamos la hora y la fecha
  // Existen funciones para fijar cualquiera de estos campos de forma independiente
  rtc.setTime(hours, minutes, seconds);
  rtc.setDate(day, month, year);
}

// ----------------------------------------------------------------------------
// ----------------------------------------------------------------------------
void loop()
{
  // Generamos e imprimimos la fecha y la hora
  char dateTime[25];
  snprintf(dateTime, sizeof(dateTime),"%4u/%02u/%02u %02u:%02u:%02u",
           (uint16_t)rtc.getYear() + 2000, rtc.getMonth(), rtc.getDay(),
           rtc.getHours(), rtc.getMinutes(), rtc.getSeconds());
  SerialUSB.println(dateTime);
  delay(1000);
}
```

El ejemplo 3 es una ligera variación del ejemplo anterior que ilustra cómo resolver esta cuestión usando la función **getEpoch()** incluida en la librería ZeroRTC. Dicha función devolverá el tiempo de época, en ocasiones denominado tiempo UNIX o POSIX, que es el número de segundos transcurridos desde el 1 de enero de 1970. Alternativamente, se puede seleccionar el 1 de enero de 2000 como origen de tiempo.

```
/* ----------------------------------------------------------------------
 *  Ejemplo 3: Puesta en hora del RTC
 *             Lectura atómica del tiempo
```

```
 *             Conversión de Epoch al formato habitual de fecha/hora
 *  - Requiere la librería RTCZero
 *    https://www.arduino.cc/reference/en/libraries/rtczero/
 *  Asignatura (GII-IoT)
 * ---------------------------------------------------------------------
 */
#include <time.h>
#include <RTCZero.h>

// rtc object
RTCZero rtc;

// Ponemos una hora y fecha arbitrarias
const uint8_t seconds = 50;
const uint8_t minutes = 59;
const uint8_t hours = 23;

const uint8_t day = 31;
const uint8_t month = 12;
const uint8_t year = 22;

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------
void setup()
{
  SerialUSB.begin(115200);

  // Habilitamos el uso del rtc
  rtc.begin();

  // Fijamos la hora y la fecha
  // Existen funciones para fijar cualquiera de estos campos de forma independiente
  rtc.setTime(hours, minutes, seconds);
  rtc.setDate(day, month, year);
}

// ---------------------------------------------------------------------------
// ---------------------------------------------------------------------------
void loop()
{
  char dateTime[32];
  static bool useEpoch = false;

  if (!useEpoch) {
    // Generamos e imprimimos la fecha y la hora
```

```
    snprintf(dateTime, sizeof(dateTime),"_____ %4u/%02u/%02u %02u:%02u:%02u",
             (uint16_t)rtc.getYear() + 2000, rtc.getMonth(), rtc.getDay(),
             rtc.getHours(), rtc.getMinutes(), rtc.getSeconds());
  }
  else {
    struct tm stm;
    // Obtenemos el tiempo en posixTime, segundos desde el 1 de enero de 1970
    time_t epoch = rtc.getEpoch();

    // Covertimos a la forma habitual de fecha y hora
    gmtime_r(&epoch, &stm);

    // Generamos e imprimimos la fecha y la hora
    snprintf(dateTime, sizeof(dateTime),"EPOCH %4u/%02u/%02u %02u:%02u:%02u",
             stm.tm_year + 1900, stm.tm_mon + 1, stm.tm_mday,
             stm.tm_hour, stm.tm_min, stm.tm_sec);
  }

  SerialUSB.println(dateTime);
  useEpoch = !useEpoch;

  delay(1000);
}
```

Puesta en hora del RTC desde fuentes externas

La puesta en hora del RTC es evidentemente necesaria al inicio de una aplicación, pero puede ser una capacidad necesaria para corregir periódicamente la deriva inherente a este tipo de dispositivos. En cualquiera de estas situaciones se requiere alguna forma de sincronización con el exterior. Las formas más habituales de lograr esta capacidad de puesta en hora consiste en explotar el tiempo proporcionado por dispositivos de posicionamiento global por satélite (Global Navigation Satellite System, GNSS) o por infraestructuras de comunicación tales como Internet (Network Time protocol servers, NTP), redes de telefonía móvil o señales de radio.

Lamentablemente, con los recursos disponibles para esta primera práctica, no podemos emplear ninguno de estos mecanismos de sincronización. Sin embargo, podemos conseguir una primera aproximación explotando las macros __TIME__ y __DATE__ que cualquier compilador incluye en el código objeto que genera. Evidentemente, estos datos, en forma de cadenas de texto, solo contienen el tiempo y la fecha en la que se compiló el código fuente.

El siguiente ejemplo indica cómo explotar esta posibilidad para – al menos – lograr una puesta en hora inicial, asumiendo que el firmware siempre se regenera (v.g. se compila) antes de grabarlo en el microcontrolador. Ver la función **bool setDateTime(const char * date_str, const char * time_str)**.

```
/* ------------------------------------------------------------------------
 *  Ejemplo 4: Puesta en hora del RTC usando la fecha y hora de la compilación
```

```
 *             Lectura atómica del tiempo
 *             Conversión de Epoch al formato habitual de fecha/hora
 *  - Requiere la librería RTCZero
 *    https://www.arduino.cc/reference/en/libraries/rtczero/
 *  Asignatura (GII-IoT)
 * -----------------------------------------------------------------------------
 */
#include <time.h>
#include <RTCZero.h>

// rtc object
RTCZero rtc;

// -----------------------------------------------------------------------------
// -----------------------------------------------------------------------------
void setup()
{
  SerialUSB.begin(115200);
  while(!SerialUSB) {;}

  SerialUSB.print(__DATE__);
  SerialUSB.print(" ");
  SerialUSB.println(__TIME__);

  // Habilitamos el uso del rtc
  rtc.begin();

  // Analizamos las dos cadenas para extraer fecha y hora y fijar el RTC
  if (!setDateTime(__DATE__, __TIME__))
  {
    SerialUSB.println("setDateTime() failed!\nExiting ...");
    while (1) { ; }
  }
}

// -----------------------------------------------------------------------------
// -----------------------------------------------------------------------------
void loop()
{
  char dateTime[32];
  struct tm stm;
  // Obtenemos el tiempo en posixTime, segundos desde el 1 de enero de 1970
  time_t epoch = rtc.getEpoch();

  // Convertimos a la forma habitual de fecha y hora
```

```
  gmtime_r(&epoch, &stm);

  // Generamos e imprimimos la fecha y la hora
  snprintf(dateTime, sizeof(dateTime),"EPOCH %4u/%02u/%02u %02u:%02u:%02u",
           stm.tm_year + 1900, stm.tm_mon + 1, stm.tm_mday,
           stm.tm_hour, stm.tm_min, stm.tm_sec);

  SerialUSB.println(dateTime);
  delay(1000);
}


bool setDateTime(const char * date_str, const char * time_str)
{
  char month_str[4];
  char months[12][4] = {"Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug",
                        "Sep", "Oct", "Nov", "Dec"};
  uint16_t i, mday, month, hour, min, sec, year;

  if (sscanf(date_str, "%3s %hu %hu", month_str, &mday, &year) != 3) return false;
  if (sscanf(time_str, "%hu:%hu:%hu", &hour, &min, &sec) != 3) return false;

  for (i = 0; i < 12; i++) {
    if (!strncmp(month_str, months[i], 3)) {
      month = i + 1;
      break;
    }
  }
  if (i == 12) return false;
  rtc.setTime((uint8_t)hour, (uint8_t)min, (uint8_t)sec);
  rtc.setDate((uint8_t)mday, (uint8_t)month, (uint8_t)(year - 2000));
  return true;
}
```

Uso de la alarma del RTC

El último ejemplo sobre el uso del RTC muestra cómo es posible programar una alarma en el RTC y asociar esa alarma a una rutina de servicio o callback. Es muy frecuente usar la alarma para despertar al microcontrolador de un estado de “suspensión”, sleep mode, o de baja energía. De hecho, la librería ZeroRTC proporciona algún ejemplo donde se usa la función standby(), con el objetivo de “dormir” al microcontrolador, y se le despierta con la alarma del RTC. No obstante, por si alguien decide intentar adaptar este ejemplo en ese sentido, hay que advertir que el puerto SerialUSB queda inoperativo al entrar en el modo de suspensión y no se recuperará al despertarlo. Esta es una limitación conocida del IDE de Arduino para este microcontrolador.

Es importante destacar el modificador ***volatile***, requerido por las variables que son accedidas desde rutinas de servicio de interrupciones (ISR). ***LED_BUILTIN*** es una macro que identifica el pin DIO usado para controlar el LED indicador incluido en la placa.

```
/* ------------------------------------------------------------------------
 *  Ejemplo 5: Puesta en hora del RTC usando la fecha y hora de la compilación
 *             Lectura atómica del tiempo
 *             Conversión de Epoch al formato habitual de fecha/hora
 *             Programación de una alarma
 *
 *  - Requiere la librería RTCZero
 *    https://www.arduino.cc/reference/en/libraries/rtczero/
 *  Asignatura (GII-IoT)
 * ------------------------------------------------------------------------
 */
#include <time.h>
#include <RTCZero.h>

// rtc object
RTCZero rtc;

// OJO al carácter volátil de estas variables
volatile uint32_t _period_sec = 0;
volatile uint16_t _rtcFlag = 0;

// Macro útil para medir el tiempo transcurrido en milisegundos
#define elapsedMilliseconds(since_ms)  (uint32_t)(millis() - since_ms)

// ------------------------------------------------------------------------
// ------------------------------------------------------------------------
void setup()
{
  // Registramos el tiempo de comienzo
  uint32_t t_start_ms = millis();

  // Activamos el pin de control del LED
  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, HIGH);

  SerialUSB.begin(9600);
  while(!SerialUSB) {;}

  SerialUSB.print("Starting at: ");
  SerialUSB.print(__DATE__);
  SerialUSB.print(" ");
  SerialUSB.println(__TIME__);
```

```
  // Habilitamos el uso del rtc
  rtc.begin();

  // Analizamos las dos cadenas para extraer fecha y hora y fijar el RTC
  if (!setDateTime(__DATE__, __TIME__))
  {
    SerialUSB.println("setDateTime() failed!\nExiting ...");
    while (1) { ; }
  }

   // Activar la alarma cada 10 segundos a partir de 5 secs
   setPeriodicAlarm(10, 5);

   // Apagamos el led, pero esperamos a que hayan pasado 3 segundos
   // como mínimo desde el principio
   while(elapsedMilliseconds(t_start_ms) > 3000) { delay(1); }
   digitalWrite(LED_BUILTIN, LOW);

   // Limpiamos _rtcFlag
   _rtcFlag = 0;

   // Activar la rutina de atención
   rtc.attachInterrupt(alarmCallback);
}

// ----------------------------------------------------------------------------
// ----------------------------------------------------------------------------
void loop()
{
  if ( _rtcFlag ) {

    // Imprimimos la fecha y la hora
    printDateTime();

    // Decrementamos _rtcFlag
    _rtcFlag--;
    if ( _rtcFlag) SerialUSB.println("WARNING: Unattended RTC alarm events!");

    // Apagamos el led
    digitalWrite(LED_BUILTIN, LOW);
  }
}

// ----------------------------------------------------------------------------
```

```
// Fija fecha y hora a partir de dos cadenas con el formato de __DATE__ y __TIME__
// -----------------------------------------------------------------------------
bool setDateTime(const char * date_str, const char * time_str)
{
  char month_str[4];
  char months[12][4] = {"Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug",
                        "Sep", "Oct", "Nov", "Dec"};
  uint16_t i, mday, month, hour, min, sec, year;

  if (sscanf(date_str, "%3s %hu %hu", month_str, &mday, &year) != 3) return false;
  if (sscanf(time_str, "%hu:%hu:%hu", &hour, &min, &sec) != 3) return false;

  for (i = 0; i < 12; i++) {
    if (!strncmp(month_str, months[i], 3)) {
      month = i + 1;
      break;
    }
  }
  if (i == 12) return false;

  rtc.setTime((uint8_t)hour, (uint8_t)min, (uint8_t)sec);
  rtc.setDate((uint8_t)mday, (uint8_t)month, (uint8_t)(year - 2000));
  return true;
}

// -----------------------------------------------------------------------------
// Imprime la hora y fecha en un formato internacional estándar
// -----------------------------------------------------------------------------
void printDateTime()
{
  const char *weekDay[7] = { "Sun", "Mon", "Tue", "Wed", "Thr", "Fri", "Sat" };

  // Obtenemos el tiempo Epoch, segundos desde el 1 de enero de 1970
  time_t epoch = rtc.getEpoch();

  // Covertimos a la forma habitual de fecha y hora
  struct tm stm;
  gmtime_r(&epoch, &stm);

  // Generamos e imprimimos la fecha y la hora
  char dateTime[32];
  snprintf(dateTime, sizeof(dateTime),"%s %4u/%02u/%02u %02u:%02u:%02u",
           weekDay[stm.tm_wday],
           stm.tm_year + 1900, stm.tm_mon + 1, stm.tm_mday,
           stm.tm_hour, stm.tm_min, stm.tm_sec);
```

```
  SerialUSB.println(dateTime);
}

// ------------------------------------------------------------------------
// Programa la alarma del RTC para que se active de en period_sec segundos a
// partir de "offset" en segundos desde el instante actual
// ------------------------------------------------------------------------
void setPeriodicAlarm(uint32_t period_sec, uint32_t offsetFromNow_sec)
{
  _period_sec = period_sec;
  rtc.setAlarmEpoch(rtc.getEpoch() + offsetFromNow_sec);

  // Ver enum Alarm_Match en RTCZero.h
  rtc.enableAlarm(rtc.MATCH_YYMMDDHHMMSS);
}

// ------------------------------------------------------------------------
// Rutina de servicio asociada a la interrupción provocada por la expiración de la
// alarma.
// ------------------------------------------------------------------------
void alarmCallback()
{
  // Incrementamos la variable bandera
  _rtcFlag++;

  // Encendemos el led
  digitalWrite(LED_BUILTIN, HIGH);

  // Reprogramamos la alarma usando el mismo periodo
  rtc.setAlarmEpoch(rtc.getEpoch() + _period_sec);
}
```

Programación del temporizador del Watchdog (WatchDog Timer, WDT)

El wacthdog incluido en el microcontrolador SAMD21G es un recurso que permite activar un reset, como medida de continuidad, en el caso de que el programa pueda quedar atrapado en una situación de bloqueo que impida su correcto funcionamiento. El principio de operación es muy sencillo: el watchdog se programa con un tiempo de expiración máximo, como un cronometro de cuenta-atrás, que debe reiniciarse antes de que expire. Si este reinicio o reset no se verifica dentro del tiempo máximo establecido, el microcontrolador recibirá una señal de reset.

El código que pretende activar el uso del watchdog se diseña siempre teniendo en cuenta es necesario reiniciar el contador de tiempo de forma periódica, con un tiempo de expiración ajustado al tiempo máximo de ciclo de la aplicación.

El WDT del SAMDG21 puede vincularse a diferentes relojes del microcontrolador y usarse también para sacar el microcontrolador de un estado de baja energía o sleep mode. Una particularidad del WDT de este microcontrolador es la posibilidad de ofrecer un aviso “temprano” o Early Warning sobre una próxima expiración del contador mediante un callback. Esta posibilidad puede ser útil y otorgar más flexibilidad en la arquitectura del firmware. Consultar la **sección 18 del manual** del SAMD21G para más información.
La librería WDTZero proporciona un conjunto de funciones para un acceso simplificado a la programación del watchdog del SAMD21G. El siguiente ejemplo simula el comportamiento de un programa que realiza un primer ciclo de forma correcta, reiniciando el watchdog periódicamente, y un segundo ciclo donde este reinicio se omite y el watchdog se activa tras el periodo de salvaguarda preestablecido. Ajustando la macro WDT_SOFT es posible seleccionar el modo “soft” en el que se invoca un una función de servicio (shutdownCallback() en este ejemplo) antes de reiniciar el microcontrolador, o en el modo “hard” donde esta función no llega a invocarse si se activa el reset.

```
/* ------------------------------------------------------------------------
 *  Ejemplo 6: Uso del watchdog del SAMD21G
 *
 *     Usa la librería WDTZero
 *     https://github.com/javos65/WDTZero
 *
 *  Asignatura (GII-IoT)
 * ------------------------------------------------------------------------
 */
#include <WDTZero.h>
#define WDT_SOFT  false // true

#define timeoutExpired_ms(start_ms, timeout_ms)  \
                         (((millis() - timeout_ms) <= start_ms) ? false : true)

// Declaramos el objeto para manejear el watchdog
WDTZero wdt;

// ------------------------------------------------------------------------
// ------------------------------------------------------------------------
void setup()
{
  SerialUSB.begin(115200);
  while(!SerialUSB) {;}

  wdt.attachShutdown(shutdownCallback);

#if WDT_SOFT

  // Inicializamos el WDT en modo "soft" (Early Warning).
  SerialUSB.println("\n\nWDTZero-Demo : Setup Soft Watchdog at 32S interval");
  wdt.setup(WDT_SOFTCYCLE32S);
```

```
#else

  // Inicializamos el WDT en modo "hard".
  SerialUSB.println("\n\nWDTZero-Demo : Setup Hard Watchdog at 16S interval");
  wdt.setup(WDT_HARDCYCLE16S);

#endif
}

// ------------------------------------------------------------------------------
// ------------------------------------------------------------------------------
void loop()
{
  static int cycle = 0;
  static uint32_t lapse_init_time_ms;

  SerialUSB.print("\nThis is cycle: ");
  SerialUSB.println(cycle);
  SerialUSB.print("WDTZeroCounter: ");
  SerialUSB.println(WDTZeroCounter);

  for (int i = 0; i < 60; ) {
    lapse_init_time_ms = millis();

    // Refrescamos el watchdog para impedir que salte
    if (!cycle) wdt.clear();

    if (i < 10) SerialUSB.print(0);
    SerialUSB.print(i);
    SerialUSB.print(" ");

    if ( !(++i % 10) ) {
      SerialUSB.print("  WDTZeroCounter: ");
      SerialUSB.println(WDTZeroCounter);
    }
    while (!timeoutExpired_ms(lapse_init_time_ms, 1000))  delay(1);
  }
  cycle++;
}
```

```
// ------------------------------------------------------------------------
// Función que se invoca con anterioridad al reset en el modo "soft"
// ------------------------------------------------------------------------
void shutdownCallback()
{
  SerialUSB.println("\nWatchdog will reset the board now ...");
}
```

Uso de la memoria FLASH externa (W25Q16)

La placa Arduino MKR1310 incluye una memoria externa de 2 Mbyte (W25Q16) accesible a través del segundo bus SPI (SPI1). El uso de esta memoria no volátil es muy interesante como recurso para el almacenamiento de datos ya que esta placa carece de otros periféricos de almacenamiento, por ejemplo un zócalo para tarjetas de memoria.

La librería oficial de Arduino para usar este tipo de dispositivos, MKRMEM, solo soporta este chip de memoria instalado en el shield MKR MEM Shield y no funciona correctamente con el chip disponible en el MKR 1310, por lo que es necesario “retocarla” ligeramente antes de poder usarla con esa tarjeta.

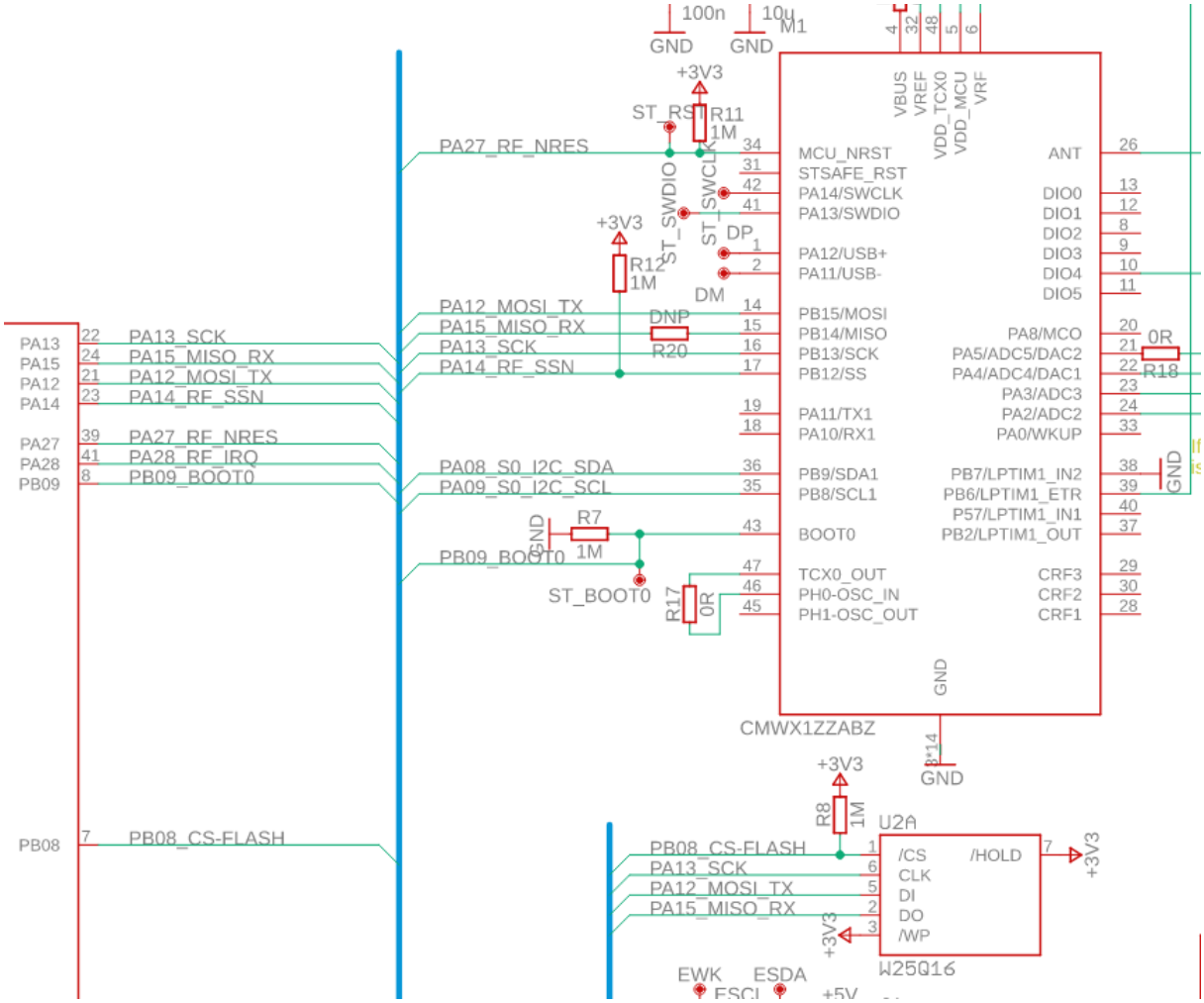

Figura 1: Detalle de las conexiones módulo Murata CMWX1ZZABZ y de la memoria W25Q16 al microcontrolador según el esquema de la tarjeta Arduino MKR 1310.

La incompatibilidad tiene su origen en que es necesario apagar el módulo LoRA Murata CMWX1ZZABZ antes de poder usar la memoria W25Q16. De acuerdo con el esquema de la tarjeta (ver detalle en la siguiente figura), ambos dispositivos está conectados al bus SPI implementado sobre los pines PA12(MOSI), PA13(SCK) y PA15(MISO). El pin de selección SPI para el módulo Murata es el PA14(RF_SSN); el pin de selección del módulo de memoria Flash es el PB08(CS-FLASH).

Esta denominación de pines es la que corresponde al microcontrolador SAMD21G. Para saber cómo localizarlos en el contexto del IDE de Arduino, es necesario verificar el siguiente proceso:

- Primero localizar la carpeta de instalación del software de Arduino. En Linux es una carpeta oculta que suele ubicarse en la carpeta personal del usuario que realizó la instalación. **$HOME/.arduino15/packages/arduino/hardware/samd/18.13**. Es fácil apreciar que está organizada por arquitecturas. En nuestro caso nos interesa la SAMD.
- En dicha carpeta, consultar (abrir solo para lectura) el fichero **boards.txt**, que contiene la parametrización de todas las tarjetas soportadas por Arduino. Localizar el bloque correspondiente al modelo mkrwan1310. Debemos encontrar una línea como esta:

mkrwan1310.build.variant=mkrwan1300, que indica la tarjeta mkr1310 está asimilado a la "variante" mkr1300 (el modelo anterior).

- Ahora que hemos confirmado cómo se reconoce esta tarjeta, podemos consultar el fichero que contiene la identificación de los pines del procesador en el IDE de Arduino. Para ello hemos de consultar el fichero **variant.cpp**, localizado en la carpeta **variants/mkrwan1300** es decir,

  $HOME/.arduino15/packages/arduino/hardware/samd/18.13/variants/mkrwan1300/variant.cpp

- Ahora localizamos en dicho fichero los pines PA12, PA13, PA14, PA15 y PB08, y vemos que en Arduino se identifican, respectivamente, como los pines 26 (PA12), 27 (PA13), 28 (PA14), 29 (PA15) y 32 (PB08). Vemos también que los pines PA12, PA13 y PA15 son los que implementan el segundo bus SPI de esta tarjeta.

```
125 /*
126  +------------+------------------+--------+-----------------+-
127  | Pin number |  MKR  Board pin  |  PIN   | Notes           |
128  |            |                  |        |                 |
129  |            |                  |        |                 |
130  +------------+------------------+--------+-----------------+-
131  |            | SD SPI           |        |                 |
132  |  26        |                  |  PA12  | UART TX (MOSI)  |
133  |  27        |                  |  PA13  | (SCK)           |
134  |  28        |                  |  PA14  | (SSN)           |
135  |  29        |                  |  PA15  | UART RX (MISO)  |
136  |  30        |                  |  PA27  | MODULE RESETN   |
137  |  31        |                  |  PA28  | UNUSED          |
138  |  32        |                  |  PB08  | ADC_BATTERY     |
139  |  33        |                  |  PB09  | MODULE BOOT0    |
140  +------------+------------------+--------+-----------------+-
141  |            | 32768Hz Crystal  |        |                 |
142  |  34        |                  |  PA00  | XIN32           |
143  |  35        |                  |  PA01  | XOUT32          |
144  +------------+------------------+--------+-----------------+-
145  |            |       SPI1       |        |                 |
146  |  36        | MOSI             |  PA12  |                 |
147  |  37        | SCK              |  PA13  |                 |
148  |  38        | MISO             |  PA15  |                 |
149  +------------+------------------+--------+-----------------+-
```

Figura 2: Identificación de algunos pines del SAMD21G en variants/mkrwan1300/variant.cpp

Puede ser instructivo echar un vistazo al fichero variant.h, en ese mismo fichero. Allí encontramos la confirmación de la identidad de los pines anteriores y otros, como LORA_RESET, LORA_BOOT0, … que serán de interés un poco más adelante. Vemos también como la macro FLASH_CS (FLASH Chip Select) aparece vinculada al pin 32, según la nomenclatura de Arduino.

En realidad, el retoque de la librería MKRMEM es muy simple y requiere solamente comentar la declaración de la variable flash, líneas 193 a 199 en el fichero **src/Arduino_WQ16DV.cpp**. Adicionalmente, tal y como muestra el siguiente ejemplo, declaramos el objeto flash para que use el bus SPI1 y FLASH_CS como pin CS y apagamos el módulo LoRA manteniendo el pin de RESET, LORA_RESET a nivel bajo.

```
/* ----------------------------------------------------------------------------
 *  Ejemplo 7:
 *  - Este programa verifica el acceso al chip de memoria flash del MKR1310,
 *    lo formatea e informa de su capacidad.
 *  - Ejecutar este programa es necesario para poder usar un sistema de archivos
 *    SPIFFS (SPI Flash File System) sobre la memoria.
 *
 *  - Usa la librería MKRMEM
 *    https://github.com/arduino-libraries/Arduino_MKRMEM
 *  - Pero es necesario comentar la declaración de la variable flash como en el
 *    fichero src/Arduino_WQ16DV.cpp de la librería MKRMEM (líneas 193 - 199)
 *   *
 * Adaptado del programa SPIFFSFormat.ino de Alexander Entinger
 *----------------------------------------------------------------------------
*/
#include <Arduino_MKRMEM.h>

// ----------------------------------------------------------------------------
// IMPORTANTE: Trasladamos aquí la declaración de la variable flash para ajustar
//             el bus SPI y el pin CS de la FLASH
// ----------------------------------------------------------------------------

Arduino_W25Q16DV flash(SPI1, FLASH_CS);

// ----------------------------------------------------------------------------
// ----------------------------------------------------------------------------
void setup()
{
  // Recordar que LORA_RESET está definida en
  // .arduino15/packages/arduino/hardware/samd/1.8.13/variants/mkrwan1300/variant.h
  pinMode(LORA_RESET, OUTPUT);    // Declaramos LORA reset pin como salida
  digitalWrite(LORA_RESET, LOW);  // Lo ponemos a nivel bajo para desactivar el
                                  // módulo LoRA

  SerialUSB.begin(9600);
  while(!SerialUSB) { ; }

  flash.begin();

  SerialUSB.println("Erasing chip ...");
  flash.eraseChip();

  // filesystem is declared inArduino_SPIFFS.cpp
  SerialUSB.println("Mounting ...");
  int res = filesystem.mount();
```

```
  if (res != SPIFFS_OK && res != SPIFFS_ERR_NOT_A_FS) {
    SerialUSB.println("mount() failed with error code ");
    SerialUSB.println(res); return;
  }

  SerialUSB.println("Unmounting ...");
  filesystem.unmount();

  SerialUSB.println("Formatting ...");
  res = filesystem.format();
  if(res != SPIFFS_OK) {
    SerialUSB.println("format() failed with error code ");
    SerialUSB.println(res); return;
  }

  SerialUSB.println("Mounting ...");
  res = filesystem.mount();
  if(res != SPIFFS_OK) {
    SerialUSB.println("mount() failed with error code ");
    SerialUSB.println(res); return;
  }

  SerialUSB.println("Checking ...");
  res = filesystem.check();
  if(res != SPIFFS_OK) {
    SerialUSB.println("check() failed with error code ");
    SerialUSB.println(res); return;
  }

  SerialUSB.println("Retrieving filesystem info ...");
  unsigned int bytes_total = 0,
               bytes_used  = 0;
  res = filesystem.info(bytes_total, bytes_used);
  if(res != SPIFFS_OK) {
    SerialUSB.println("check() failed with error code ");
    SerialUSB.println(res); return;
  } else {
    char msg[64] = {0};
    snprintf(msg, sizeof(msg), "SPIFFS Info:\nBytes Total: %d\nBytes Used:  %d",
           bytes_total, bytes_used);
    SerialUSB.println(msg);
  }
  SerialUSB.println("Unmounting ... (formatting finished!");
  filesystem.unmount();
}
```

```
void loop() { }
```

El SPI Flash File System (SPIFFS) permite definir un sistema de archivos sobre el chip W25Q16 y usarlo como una tarjeta de memoria con capacidad de 2 Mbytes de almacenamiento no volátil. El primer paso es, evidentemente, formatear "la unidad" (el chip de memoria). Es lo que acabamos de hacer al ejecutar el ejemplo anterior. Puede resultar instructivo leer el README.md de la librería MKRMEM para conocer algunos detalles sobre el uso de estos chips de memoria FLASH.

El siguiente y último ejemplo muestra cómo puede crearse un fichero sobre el chip de memoria FLASH, tanto en modo escritura como lectura. Es evidente de la inspección del código que el diseño de la librería, más concretamente de las clases **Arduino_SPIFFS_File** y **Arduino_SPIFFS**, complica innecesariamente la posibilidad de exportar una variable de tipo File desde una función. Esta circunstancia explica el diseño de la función loop(), donde es necesario abrir y cerrar el fichero en cada ciclo.

```
/* ------------------------------------------------------------------------
 *  Ejemplo 8:
 *  - Este programa muestra la forma de usar el SPIFFS sobre el chip de memoria.
 *  - Ejecutar este programa es necesario para poder usar un sistema de archivos
 *    SPIFFS (SPI Flash File System) sobre la memoria.
 *
 *  - Usa la librería MKRMEM
 *    https://github.com/arduino-libraries/Arduino_MKRMEM
 *  - Pero es necesario comentar la declaración de la variable flash como en el
 *    fichero src/Arduino_WQ16DV.cpp de la librería MKRMEM (líneas 193 - 199)
 *
 *------------------------------------------------------------------------
 */

#include <Arduino_MKRMEM.h>

// ------------------------------------------------------------------------
// IMPORTANTE: Trasladamos aquí la declaración de la variable flash para ajustar
//             el bus SPI y el pin CS de la FLASH
// ------------------------------------------------------------------------
Arduino_W25Q16DV flash(SPI1, FLASH_CS);
char filename[] = "datos.txt";

// ------------------------------------------------------------------------
// ------------------------------------------------------------------------
void setup()
{
  // Recordar que LORA_RESET está definida en
```

```
  // .arduino15/packages/arduino/hardware/samd/1.8.13/variants/mkrwan1300/variant.h
  pinMode(LORA_RESET, OUTPUT);   // Declaramos LORA reset pin como salida
  digitalWrite(LORA_RESET, LOW); // Lo ponemos a nivel bajo para desactivar el
                                 // módulo LoRA

  SerialUSB.begin(9600);
  while(!SerialUSB) { ; }

  // Inicializamos la memoria FLASH
  flash.begin();

  // Montamos el sistema de archivos
  SerialUSB.println("Mounting ...");
  int res = filesystem.mount();
  if(res != SPIFFS_OK && res != SPIFFS_ERR_NOT_A_FS) {
    SerialUSB.println("mount() failed with error code ");
    SerialUSB.println(res);
    exit(EXIT_FAILURE);
  }

  // Creamos un nuevo fichero
  // Podríamos usar create(), pero open() proporciona más flexibilidad (flags)
  File file = filesystem.open(filename,  CREATE | TRUNCATE);
  if (!file) {
    SerialUSB.print("Creation of file ");
    SerialUSB.print(filename);
    SerialUSB.print(" failed. Aborting ...");
    on_exit_with_error_do();
  }
  file.close();
}

// ----------------------------------------------------------------------------
// ----------------------------------------------------------------------------
void loop()
{
  static int line = 0;
  char data_line[50];

  if (line < 10) {

    // Reabrimos el fichero en cada iteración para añadir datos
    File file = filesystem.open(filename,  WRITE_ONLY | APPEND);
    if (!file) {
      SerialUSB.print("Opening file ");
```

```
    SerialUSB.print(filename);
    SerialUSB.print(" failed for appending. Aborting ...");
    on_exit_with_error_do();
  }

  // Escribimos una línea en el fichero por segundo
  snprintf(data_line, sizeof(data_line), "At %lu: this is line no. %02d\n",
           millis(), ++line);

  SerialUSB.print("Writing: ");
  SerialUSB.print(data_line);

  int const bytes_to_write = strlen(data_line);
  int const bytes_written = file.write((void *)data_line, bytes_to_write);
  if (bytes_to_write != bytes_written) {
    SerialUSB.print("write() failed with error code ");
    SerialUSB.println(filesystem.err());
    SerialUSB.println("Aborting ...");
    on_exit_with_error_do();
  }
  // Cerramos el fichero
  file.close();
  delay(998);
}
else {

  // Abrimos el fichero para lectura
  File file = filesystem.open(filename,  READ_ONLY);
  if (!file) {
    SerialUSB.print("Opening file ");
    SerialUSB.print(filename);
    SerialUSB.print(" failed for reading. Aborting ...");
    on_exit_with_error_do();
  }
  SerialUSB.print("Reading file contents:\n\t ");

  // Leemos el contenido del fichero hasta alcanzar la marca EOF
  while(!file.eof()) {
    char c;
    int const bytes_read = file.read(&c, sizeof(c));
    if (bytes_read) {
      SerialUSB.print(c);
      if (c == '\n') SerialUSB.print("\t ");
    }
  }
```

```
    // Cerramos el fichero
    file.close();
    SerialUSB.println("\nFile closed");

    // Desmontamos el sistema de archivos
    SerialUSB.println("Unmounting filesystem ... (program finished)");
    filesystem.unmount();
    exit(0);
  }
}

// ------------------------------------------------------------------------
// Utilidad para desmontar el sistema de archivos y terminar en caso de error
// ------------------------------------------------------------------------
void on_exit_with_error_do()
{
  filesystem.unmount();
  exit(EXIT_FAILURE);
}
```

Uso de los modos de bajo consumo del microcontrolador

El microcontrolador SAMD21G incluido en la placa Arduino MKR1310 permite varios modos de operación, entre ellos algunos donde es posible suspender la operación del micro para reducir de forma significativa su consumo de energía. Existen varias librerías, como RTCZero o Arduino Low Power, que simplifican el uso de estos modos de bajo consumo.

Por su mayor flexibilidad, en el siguiente ejemplo usaremos esta última librería para mostrar cómo es posible hacer uso de estos modos. Más concretamente, la librería permite activar el modo sleep mediante la función `LowPower.sleep(sleepingTime_ms)`, donde el parámetro indica el tiempo durante el cuál deseamos que el microcontrolador esté suspendido en el modo sleep. La función programa la alarma del RTC para que emita una interrupción al final de este periodo. Si se omite este parámetro, el micro permanecerá en este estado hasta que se produzca una interrupción en alguna de las líneas habilitadas para ello. La librería ofrece aparentemente tres modos, idle, sleep y deepSleep. Sin embargo, si revisamos el código fuente de la librería podemos comprobar que los métodos sleep y deepSleep hacen exactamente lo mismo, esto es, activar el modo deep sleep del microcontrolador.

Adicionalmente, la función `LowPower.attachInterruptWakeup(RTC_ALARM_WAKEUP, dummy, CHANGE)` permite registrar una función de servicio o ISR, en este caso denominada dummy(), que se ejecutará justo cuando se reciba la interrupción que ha despertado al microcontrolador. En este ejemplo, dummy() incrementará la variable volátil *iterations*. El valor de esta variable se usará para hacer parpadear el LED de la placa un número de ciclos similar al valor de *iterations*, hasta un máximo de 9 ciclos.

```
/* ------------------------------------------------------------------------
 *  Ejemplo 9: Ilustra cómo es posible usar interrupciones internas para
 *             activar periódicamente el modo sleep con el consiguiente
 *             ahorro de energía. El RTC del microcontrolador lo despertará
 *             cada 2s.
 *
 *  - Importante: El uso del SerialUSB es cuasi-incompatible con el modo sleep
 *  - Requiere la librería ArduinoLowPower
 *      https://www.arduino.cc/reference/en/libraries/arduino-low-power/
 *
 *  NOTA: Mientras el microcontrolador esté en el modo sleep no será posible
 *        cargar un nuevo firmware. En ese caso, se puede pulsar dos veces el
 *        botón de reset.
 *
 *  Asignatura (GII-IoT)
 * ------------------------------------------------------------------------
 */
#include "ArduinoLowPower.h"

const uint32_t halfPeriod_ms = 500;
const uint32_t sleepingTime_ms = 2000;
volatile int iterations = 0;

void flash_n_times(uint16_t repetitions)
{
  if (!repetitions) repetitions = 1;
  for (uint16_t k = 0; k < repetitions; k++) {
    digitalWrite(LED_BUILTIN, HIGH);
    delay(halfPeriod_ms);
    digitalWrite(LED_BUILTIN, LOW);
    delay(halfPeriod_ms);
  }
}

void setup()
{
  pinMode(LED_BUILTIN, OUTPUT);

  // Comentar esta función evitará que se invoque la función dummy() tras despertar
  // al micro y la variable repetitions no cambiará
  // LowPower.attachInterruptWakeup(RTC_ALARM_WAKEUP, dummy, CHANGE);
}
```

```
void loop()
{
  // Hacemos parpadear el LED del MKR1310
  flash_n_times(iterations%10);

  // Pone a dormir el microcontrolador durante el tiempo consignado en la variable
  // sleepingTime_ms con el consiguiente ahorro de energía. Al expirar este tiempo,
  // el RTC emitirá una alarma que despertará al micro. La librería oculta muchos
  // detalles, como el registro de los tipos de eventos que pueden sacar
  // al microcontrolador del modo sleep
  LowPower.sleep(sleepingTime_ms);
}

void dummy()
{
  // Esta función se ejecuta una vez cuando el dispositivo se despierte
  // Como se ejecuta en un contexto de interrupción, el procesamiento debe
  // ser muy rápido. En consecuencia, no deben invocarse funciones que
  // consuman mucho tiempo (e.g delay())
  iterations++;
}
```

El ejemplo 10 es una variación del anterior para incluir la posibilidad de despertar al microcontrolador bien cuando expire un periodo de tiempo, como en el ejemplo anterior, bien cuando se produzca una interrupción externa a través de un pin digital de entrada (como en el primer ejemplo de este tema). La librería `ArduinoLowPower` hace uso internamente de la librería RTCZero y permite combinar el uso del RTC con otras fuentes de interrupción.

```
/* ---------------------------------------------------------------------------
 *  Ejemplo 10: Este ejemplo muestra cómo es posible combinar dos fuentes de
 *              interrupción para despertar al microcontrolador y sacarlo de
 *              un estado deep sleep.
 *
 *  - Importante: El uso del SerialUSB es cuasi-incompatible con el modo sleep
 *  - Requiere la librería ArduinoLowPower
 *      https://www.arduino.cc/reference/en/libraries/arduino-low-power/
 *
 *  NOTA: Mientras el microcontrolador esté en el modo sleep no será posible
 *        cargar un nuevo firmware. En ese caso, se puede pulsar dos veces el
 *        botón de reset.
 *
 *  Asignatura (GII-IoT)
 * ---------------------------------------------------------------------------
 */
#include <ArduinoLowPower.h>
```

```
// ArduinoLowPower.h incluye internamente RTCZero.h
RTCZero rtc;

const int externalPin = 5;
const uint32_t alarm_halfPeriod_ms = 100;
const uint32_t external_halfPeriod_ms = 500;

volatile int alarm_iterations = 0;
volatile int external_iterations = 0;
volatile uint32_t _period_sec = 0;

void flash_n_times(uint16_t repetitions, uint32_t halfPeriod_ms)
{
  if (!repetitions) repetitions = 1;
  for (uint16_t k = 0; k < repetitions; k++) {
    digitalWrite(LED_BUILTIN, HIGH);
    delay(halfPeriod_ms);
    digitalWrite(LED_BUILTIN, LOW);
    delay(halfPeriod_ms);
  }
}

void setup()
{
  pinMode(LED_BUILTIN, OUTPUT);

  // Ajustamos el modo del externalPin
  // Lo configuramos en modo pullup para que se produzca una
  // interrupción por flanco de bajada (FALLING) al llevarl a tierra
  pinMode(externalPin, INPUT_PULLUP);
  LowPower.attachInterruptWakeup(externalPin, externalCallback, FALLING);

  // Ponemos en hora el RTC
  rtc.begin();
  rtc.setTime(10, 0, 0);
  rtc.setDate(21, 9, 23);

  // IMPORTANTE: Mantener el orden de estas dos sentencias porque
  // LowPower.attachInterruptWakeup() reinicializa la configuración del
  // RTC
  LowPower.attachInterruptWakeup(RTC_ALARM_WAKEUP, alarmCallback, CHANGE);
  setPeriodicAlarm(10,5);
```

```
  // Pone a dormir el microcontrolador INDEFINIDAMENTE hasta que se genere
  // alguna de las interrupciones
  LowPower.sleep();
}

void loop()
{
  if (external_iterations) {
    // Si se recibe una interrupción externa, el LED parpadeará dos veces
    // de forma "lenta" (semiperiodo de 500 ms)
    external_iterations = 0;
    flash_n_times( 2, external_halfPeriod_ms );
  }

  if (alarm_iterations) {
    // Si expira el periodo programado en el RTC, el LED parpadeará dos veces
    // de forma "rápida" (semiperiodo de 100 ms)
    alarm_iterations = 0;
    flash_n_times( 2, alarm_halfPeriod_ms );
  }

  // Pone a dormir el microcontrolador hasta que se genere una interrupción
  LowPower.sleep();
}

// ---------------------------------------------------------------------------
// Callback cuando se genera una interrupción externa
// ---------------------------------------------------------------------------
void externalCallback()
{
  external_iterations++;
}

// ---------------------------------------------------------------------------
// Callback cuando se activa la alarma del RTC
// ---------------------------------------------------------------------------
void alarmCallback()
{
  // Increment the counter
  alarm_iterations++;

  // Reprogramamos la alarma usando el mismo periodo
  rtc.setAlarmEpoch(rtc.getEpoch() + _period_sec);
}
```

```
// ---------------------------------------------------------------------------
// Programa la alarma del RTC para que se active cada  period_sec segundos a
// partir de "offsetFromNow_sec" en segundos desde el instante actual
// ---------------------------------------------------------------------------
void setPeriodicAlarm(uint32_t period_sec, uint32_t offsetFromNow_sec)
{
  _period_sec = period_sec;
  rtc.setAlarmEpoch(rtc.getEpoch() + offsetFromNow_sec);

  // Ver enum Alarm_Match en RTCZero.h
  rtc.enableAlarm(rtc.MATCH_YYMMDDHHMMSS);
}
```

Ejercicio propuesto

Desarrollar el firmware de una tarjeta Arduino MKR 1310 para implementar los siguientes objetivos:

1. Ajustar el RTC a partir de la hora y fecha de generación del firmware.
2. Programar la alarma del RTC para simular la lectura de un sensor de forma periódica (cada 10 segundos). Cada vez que la alarma se active, se generará una cadena de texto con la fecha y la hora.
3. Esta cadena de texto deberá salvarse a un fichero que habrá creado en el chip de memoria externa FLASH de la tarjeta.
4. Poner el microcontrolador en modo sleep por tiempo indefinido. Se despertará cuando se active la alarma del RTC.

Extra

5. Permitir que el microcontrolador pueda registrar otra interrupción (un flanco de bajada) a través de un pin digital, configurado como entrada en modo pull-up, cuando se lleva a tierra. Esta interrupción puede ocurrir en cualquier momento y el firmware deberá proceder como en el objetivo 2, pero indicando que esa línea se añade debido a una interrupción externa.