Sistemas Embebidos
Trabajo práctico Nº4 - Año 2023
Memoria EEPROM

Objetivos

- Leer y escribir memoria EEPROM.
- Interactuar aplicaciones en sistemas embebidos con aplicaciones web.

Metodología

Trabajo individual o grupal. 2 estudiantes por grupo máximo.
Tiempo de realización estimado: 3 clases.

Aprobación

- Mostrar en clases la aplicación funcionando correctamente.
- Enviar los programas de computación implementados a través de la plataforma Moodle.
- Escribir un breve informe y enviarlo a través de la plataforma Moodle.

Materiales necesarios

- Placas Arduino UNO (provistas por la cátedra).
- Entorno de desarrollo de Arduino UNO (puede descargarse de https://www.arduino.cc/en/software).
- Librerías para Arduino IDE:
    - Opción 1: si utiliza FreeRTOS:
        - FreeRTOS de Richard Barry (ver TP3).
    - Opción 2: si no utiliza FreeRTOS:
        - TimerOne creada por Paul Stoffregen.

Introducción

El presente trabajo práctico puede ser realizado utilizando FreeRTOS (recomendado) o no.
En caso de utilizar FreeRTOS deberá utilizar funciones para crear tareas, suspenderlas, reanudarlas, temporizadores y semáforos de FreeRTOS. La dificultad de utilizar FreeRTOS será asignar cantidad de memoria adecuada a la pila de cada tarea. En FreeRTOS el multitasking es natural.
En caso de no utilizar FreeRTOS, deberá utilizar temporizadores. Puede utilizar la librería TimerOne creada por Paul Stoffregen (se sugiere ver ejemplos). La dificultad será simular una aplicación multitasking con un algoritmo secuencial, lo que producirá errores muy difíciles de corregir.

Actividades

Escribir una aplicación que realice las siguientes tareas:

1. Implemente un reloj en tiempo real que cuente años, meses, días, horas, minutos y segundos usando un Timer del Arduino Uno. Por simplicidad, suponga que todos los meses poseen 30 días. Para el año, utilice el formato “AA” (los últimos dos dígitos del año). Debe implementarse mediante una tarea de alta prioridad o una interrupción que se ejecute cada segundo, e incrementar un segundo el reloj en cada ejecución. Vea en el Anexo 1 sugerencias de implementación. El valor del reloj debe mostrarse en la página web.
2. El reloj debe poder actualizarse desde la página web.
3. Implementar interrupciones en los pines 2 y 3 del Arduino Uno, de modo que cuando se presionen los pulsadores conectados a dichos pines, se almacene en la memoria EEPROM la hora completa (aa/mm/dd-hh:mm:ss) a la cual ocurrió el evento, y el evento ocurrido. La función debe ser tal que, si el Arduino se desconecta de la fuente de alimentación y se vuelve a conectar, nuevas escrituras en la EEPROM no deben sobreescribir escrituras anteriores.
4. Implementar funciones que permitan recuperar la información almacenada en la EEPROM para mostrarse en la página web cuando se presione un botón desde la misma..
5. La memoria EEPROM debe poder borrarse completamente cuando se presione un botón para tal fin implementado en la página web.
6. Toda la interacción del usuario con la aplicación debe realizarse a través de una página web (puede reutilizar los mecanismos implementados en trabajos prácticos anteriores).

Anexo 1

Sugerencias de implementación

A continuación se brindan sugerencias de implementación. No es obligatorio seguirlas.

Implementación del reloj en tiempo real

Un reloj en tiempo real suele implementarse mediante un contador que cuenta la cantidad de segundos desde una fecha inicial, por ejemplo, el segundo cero puede ser el 1/1/2000 a las 00:00:00 horas. Además, se agregan funciones que realicen las siguientes tareas:

- Transforman los segundos totales en cadenas de texto que indiquen fechas y horas en formatos como “aa/mm/dd-hh/mm/ss”.
- Transformen cadenas de texto con información de la fecha y hora en formatos como “aa/mm/dd-hh/mm/ss” a una variable entera que represente el total de segundos desde la fecha inicial.

La hora del reloj en tiempo real debe actualizarse a través de una tarea de máxima prioridad que cada segundo sume un segundo a la hora.

Tenga en cuenta que necesitará una variable de varios bytes para almacenar los segundos necesarios (por ejemplo: long int).

No puede usar la función `vTaskDelay` de FreeRTOS, porque el tiempo entre ejecuciones de `vTaskDelay` se perderían. En su lugar, debe utilizar `vTaskDelayUntil(TickType_t *pxPreviousWakeTime, TickType_t *xTimeIncrement);` que almacena automáticamente en la variable `pxPreviousWakeTime` la cantidad de tiks desde la última ejecución de `vTaskDelayUntil`.

Implementación de interrupciones en los pines 2 y 3

La siguiente primitiva configura las interrupciones de los pines 2 y 3.

attachInterrupt(número de la fuente de interrupción, nombre de la rutina de servicio, evento que dispara la interrupción);

- Número de la fuente de interrupción:
  - Pin 2: Interrupción número 0
  - Pin 3: Interrupción número 1
- Eventos que disparan las interrupciones:
  - LOW: para activar la interrupción cuando el pin es bajo
  - CHANGE: para activar la interrupción cuando el pin cambia de estado.
  - RISING: dispara la interrupción cuando se detecta un flanco de subida.

- FALLING: dispara la interrupción cuando se detecta un flanco de bajada.

- Nombre de la rutina de servicio: Debe indicarse el nombre de la rutina de servicio, sin agregar (). La rutina de servicio debe ser escrita como una función que no recibe argumentos ni retorna ningún resultado.

Ejemplo: attachInterrupt(digitalPinToInterrupt(2),pin2_ISR,FALLING);

Leer y escribir la memoria EEPROM

Cada posición de la memoria EEPROM almacena un byte. Se sugiere analizar los ejemplos eeprom_clear, eeprom_write y eeprom_read para comprender el funcionamiento de la memoria EEPROM. Estos ejemplos se encuentran en la carpeta de ejemplos EEPROM, instalada por defecto en Arduino IDE.
Considerar que la hora requiere una variable de varios bytes, mientras que cada posición de la memoria EEPROM almacena un byte.
Tener en cuenta que la escritura de la memoria flash para cargar un nuevo programa no sobreescribe la memoria EEPROM. Cuando una posición de la memoria EEPROM está vacía, tiene escrito el número binario 11111111 (0xFF).

Modos de bajo consumo

Se sugiere instalar y analizar los ejemplos de la librería LowPower_LowPowerLab creada por LowPowerLab.
La primitiva para poner el microcontrolador Atmel328P en modo bajo consumo es:
LowPower.idle(SLEEP_8S, ADC_OFF, TIMER2_OFF, TIMER1_OFF, TIMER0_OFF , SPI_OFF, USART0_OFF, TWI_OFF);
Donde el flag SLEEP_8S indica que el microcontrolador debe despertarse luego de 8 segundos (vea la documentación de la librería para otros intervalos de tiempo).
Los demás flags indican que periféricos deben permanecer funcionando, y por lo tanto pueden interrumpir y despertar al microcontrolador. Tenga en cuenta que el Timer1 y el puerto USART deben permanecer funcionando.
En cuanto al puerto serie, si se utiliza la bandera USART0_ON, permanece activo el receptor esperando datos. Por otro lado, el puerto serie trabaja de forma asíncrona con el procesador, por lo tanto, si intenta transmitir un dato desde el Arduino a través del puerto serie e inmediatamente coloca el microcontrolador en modo bajo consumo, la transmisión serial podría no completarse. Por lo tanto, antes de ingresar en modo bajo consumo, bebe agregar una espera de resguardo luego de un envío de datos (se sugiere utilizar delayMicroseconds(10000)).

Sugerencia: Debe crear una función cuyo propósito sea solo recibir datos. Los datos deben ser leídos solo cuando llegaron la totalidad de los caracteres del string. Para interpretar el comando que llegó, crear una tarea diferente.