S - Aplikace nad vestavným OS

Autor: Gleb Litvinchuk
Login: xlitvi02

Datum: 15.12.2024

Contents

1 Úvod

Režim **bare-metal** nebo **RTOS** lze vybrat úpravou direktivy v kódu:

- Aktivace režimu bare-metal: odkomentujte `#define BARE_METAL`.
- Aktivace režimu RTOS: odkomentujte `#define WITH_RTOS`.

1.1 Motivace

Motivací tohoto projektu je seznámit se s návrhovými vzory a vývojem vestavných aplikací pomocí vestavných OS, se službami a API zvoleného OS, zkusit si vyvinout vestavnou aplikaci bez/s OS. Dále projekt poskytuje hlubší pochopení vestavných systémů a ukazuje, jak se různé přístupy (s operačním systémem a bez něj) liší z hlediska návrhu, implementace a výkonu. Vestavné systémy jsou klíčové v moderním světě, kde reálný čas a efektivita hrají zásadní roli. Tento projekt slouží jako praktický příklad aplikace teoretických konceptů.

1.2 Cíle projektu

Hlavním cílem projektu je vytvořit aplikaci, která zahrnuje:

- Implementaci základní aplikace (bare-metal) pro detekci událostí a řízení periferií.
- Implementaci rozšířené aplikace s využitím operačního systému reálného času (RTOS).
- Použití synchronizačních mechanismů (např. semafory) a služeb pro práci s časem.
- Demonstraci rozdílů mezi bare-metal a RTOS přístupy.

1.3 Použité nástroje a platformy

Pro realizaci projektu byla zvolena platforma ESP32, která nabízí:

- Výkonné dvoujádrové jádro s podporou RTOS.
- Širokou škálu periferií (GPIO, přerušení, PWM, atd.).
- Open-source SDK (ESP-IDF) pro vývoj aplikací.

Operační systém použitý v projektu je FreeRTOS, který poskytuje flexibilní nástroje pro správu úloh, synchronizaci a práci s časem.

1.4 Očekávané výsledky

Očekává se, že tento projekt:

- Umožní porozumět základům práce s vestavnými systémy.
- Ukáže praktické použití operačního systému reálného času.
- Poskytne srovnání mezi přístupy s OS a bez něj.

2 Rozbor a návrh řešení

V této části bude představen přehled dostupných prostředků a metod použitých při realizaci projektu, návrh architektury a popis hlavních komponent systému.

2.1 Použité prostředky a metody

Pro řešení projektu byly zvoleny následující prostředky:

- Platforma **ESP32 WROOM 32D**: Vestavný mikrokontrolér s podporou dvoujádrového procesoru, Wi-Fi a Bluetooth.
- **FreeRTOS**: Operační systém reálného času, který umožňuje efektivní správu úloh a synchronizaci.
- Periferní zařízení:
  - Dvě LED diody:
    - Zelená LED dioda pro úlohu *led1_task*.
    - Modrá LED dioda pro úlohu *led2_task*.
  - Tlačítko pro generování vstupních událostí.
  - Rezistory (220 Ω) pro ochranu LED diod.
- **Breadboard**: Pro uspořádání komponent (LED, tlačítko, rezistory).
- Propojovací vodiče (female-male jumpery).
- Programovací jazyk **C**: Použití nízkoúrovňového programovacího jazyka pro optimalizaci výkonu a přesné řízení hardwaru.
- Vývojové prostředí **ESP-IDF**: Oficiální SDK pro ESP32 s podporou FreeRTOS.

2.2 Návrh architektury systému

Systém je navržen tak, aby simuloval běžné aplikace vestavných systémů s použitím RTOS a bare-metal přístupu. Architektura obsahuje následující komponenty:

- **Úloha 1 (led1_task)**: Úloha s vysokou prioritou pro ovládání první LED diody na základě vstupu z tlačítka.
- **Úloha 2 (led2_task)**: Úloha s nižší prioritou, která periodicky bliká druhou LED diodou.
- **Přerušení (ISR)**: Zajišťuje detekci stisku tlačítka a synchronizaci s úlohou led1_task pomocí semaforu.
- **Synchronizační mechanismy**: Semafory slouží k synchronizaci mezi ISR a úlohami.

2.3 Popis chování systému

Systém funguje následovně:

1. Stisk tlačítka vyvolá přerušení (ISR), které nastaví semafor a deaktivuje blikání druhé LED diody.
2. Úloha led1_task detekuje semafor a přepíná stav první LED diody (zapnuto/vypnuto).
3. Po ukončení zpracování tlačítka (led1_task) se obnoví blikání druhé LED diody.

Tato architektura umožňuje demonstrovat rozdíly mezi přístupy bare-metal a RTOS, včetně práce s prioritami úloh a synchronizací.

3 Vlastní řešení

3.1 Popis řešení

Hlavním cílem bylo vytvořit systém, který demonstruje rozdíly mezi přístupy bare-metal a RTOS. Systém obsahuje následující komponenty:

- **LED1**: Ovládána úlohou s vysokou prioritou (`led1_task`), která reaguje na stisk tlačítka.
- **LED2**: Ovládána úlohou s nižší prioritou (`led2_task`), která bliká v pravidelných intervalech.
- **Tlačítko**: Používá se k vyvolání přerušení (ISR) a změně stavu LED1.
- **Semafory**: Slouží k synchronizaci mezi ISR a úlohami.

3.2 Nastavení mikrokontroléru

Pro správnou funkčnost projektu bylo nutné nakonfigurovat následující části mikrokontroléru ESP32:

- GPIO piny:
  - **LED1_PIN (GPIO2)**: Konfigurován jako výstup pro zelenou LED.
  - **LED2_PIN (GPIO4)**: Konfigurován jako výstup pro modrou LED.
  - **BUTTON_PIN (GPIO25)**: Konfigurován jako vstup s přerušeními (ISR).
- Přerušení:
  - Tlačítko bylo konfigurováno k vyvolání přerušení na sestupné hraně signálu (`GPIO_INTR_NEGEDGE`).
  - ISR bylo implementováno s kontrolou debounce (časový filtr) pomocí konstanty `DEBOUNCE_TIME_MS` a proměnné `last_interrupt_time`.
  - ISR zajišťuje synchronizaci mezi přerušením a úlohami pomocí binárního semaforu (`xSemaphoreGiveFromISR`).
- Synchronizační mechanismy:
  - Použití binárního semaforu pro synchronizaci ISR a úlohy `led1_task`.

3.3 Použité API FreeRTOS

V projektu byly využity následující funkce a služby FreeRTOS:

- **xSemaphoreTake**: Používá se k čekání na semafor, který je nastaven ISR.
- **xSemaphoreGiveFromISR**: Používá se k nastavení semaforu přímo z přerušení.
- **vTaskDelay**: Používá se k implementaci zpoždění v úlohách (např. blikání LED).
- **xTaskCreate**: Používá se k vytvoření úloh. Úloha `led1_task` má prioritu 10 a velikost zásobníku 2048 bajtů. Úloha `led2_task` má prioritu 5 a velikost zásobníku 2048 bajtů.

Tyto funkce umožňují synchronizaci a správu času v reálném čase, což je zásadní pro implementaci RTOS přístupů.

3.4 Implementace kódu

K implementaci byla použita platforma ESP32 s vývojovým prostředím ESP-IDF. Zde je ukázka klíčových částí kódu:

```
void led1_task(void *pvParameter) {
    while (1) {
        if (xSemaphoreTake(buttonSemaphore, portMAX_DELAY)) {
            printf("Button pressed, LED1 toggled.\n");
            isLampOn = !isLampOn;
            gpio_set_level(LED1_PIN, isLampOn);
        }
    }
}
```

Listing 1: Úloha led1_task

Tato úloha zpracovává stisk tlačítka a přepíná stav LED1. Níže je uveden kód úlohy pro blikání LED2:

```
void led2_task(void *pvParameter) {
    while (1) {
        if (!can_blink) {
            vTaskDelay(pdMS_TO_TICKS(2000));
            can_blink = 1;
            printf("Switching to blinking LED2.\n");
        }
        gpio_set_level(LED2_PIN, 1);
        vTaskDelay(pdMS_TO_TICKS(500));
        gpio_set_level(LED2_PIN, 0);
        vTaskDelay(pdMS_TO_TICKS(500));
    }
}
```

Listing 2: Úloha led2_task

ISR pro zpracování tlačítka je implementováno následujícím způsobem:

```
static void IRAM_ATTR button_isr_handler(void *arg) {
    uint32_t current_time = xTaskGetTickCountFromISR() *
       portTICK_PERIOD_MS;
    if ((current_time - last_interrupt_time) > DEBOUNCE_TIME_MS) {
        last_interrupt_time = current_time;
        can_blink = 0;
        BaseType_t xHigherPriorityTaskWoken = pdFALSE;
        xSemaphoreGiveFromISR(buttonSemaphore,
           &xHigherPriorityTaskWoken);
        portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
    }
}
```

Listing 3: ISR tlačítka

3.5 Diagram architektury

Níže je diagram, který ukazuje architekturu systému a vazby mezi komponentami:

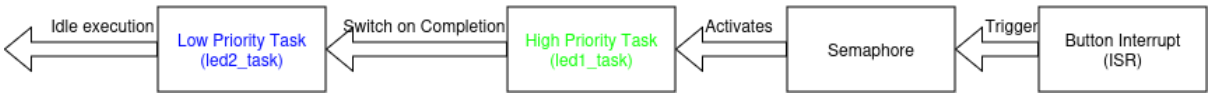

Figure 1: Architektura úloh a synchronizace.

3.6 Řešení problémů

Během realizace projektu vznikly následující problémy a jejich řešení:

- **Drebězg tlačítka**: Byla implementována kontrola debounce pomocí konstanty `DEBOUNCE_TIME_MS` (200 ms). ISR zajišťuje, že přerušení není vyvoláno zbytečně často.
- **Synchronizace úloh**: Pro synchronizaci ISR s úlohami byly použity semafory FreeRTOS.
- **Zpoždění při blikání LED2**: Bylo přidáno inicializační zpoždění 2 sekundy před začátkem blikání.

3.7 Shrnutí

Implementace splňuje všechny požadavky zadání. Úloha `led1_task` demonstruje práci s prioritami a synchronizací, zatímco `led2_task` ukazuje fungování úloh s nižší prioritou.

4 Závěrečné zhodnocení

Projekt splňuje všechny požadavky zadání:

- Úspěšně byly implementovány dvě úlohy s rozdílnými prioritami (`led1_task` a `led2_task`).

- Byly využity synchronizační mechanismy FreeRTOS (semafory) a služby pro správu času (`vTaskDelay`).
- Byly demonstrovány rozdíly mezi bare-metal a RTOS přístupy v rámci implementace.

Hlavní výzvou během projektu bylo správné nastavení přerušení a eliminace problémů se zpožděním (debounce). Projekt ukázal, že použití RTOS přístupů výrazně zlepšuje organizaci kódu a umožňuje efektivní správu více úloh.

5 Použité informační zdroje

Při realizaci projektu byly použity následující informační zdroje:

- Dokumentace ESP-IDF: `https://docs.espressif.com/projects/esp-idf/en/latest/`
- Dokumentace FreeRTOS (Espressif API): `https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/freertos_idf.html`
- Diskusní fóra Stack Overflow.