

Procesory i Systemy Operacyjne w Zastosowaniach Przemysłowych

Konfiguracja środowiska Keil

Poniższe wartości należy ustawić lub zmienić, niewymienione należy pozostawić bez zmian

Device: NXP LPC4350

Target: Xtal: 12MHz;

IROM1: 0x80000000, 0x90000, startup

IROM2: 0x80090000, 0x10000

IRAM1: 0x10080000, 0xA000

Use FPU

Linker: Use memory target layout

C/C++: Define: CORE_M4, USE_FPU

C99 mode

Debug: Use J-Link, Load application at startup, Run to main()

~~Initialization file: LPC18xx-43xx_SPIFI.ini~~

J-Link: Port: JTAG, RAM for Algorithm: start 0x10000000, size 0x8000, Programming

Algorithm SPI_Flash_LPC18xx@80000000

Utilities: Use debug driver

Initialization file: LPC18xx-43xx_SPIFI.ini

Ćwiczenie 1

Temat: Obsługa rejestrów i portów mikrokontrolera – operacje podstawowe

Zadanie: Skonfigurować odpowiednie piny, do których podłączone są diody LED (instrukcja płytki MYD). Ustawić odpowiednią konfigurację pinów (rejstry SCU) oraz konfigurację portów GPIO. Zapalać po kolei diody w nieskończonej pętli. Dla ułatwienia zdefiniować nazwy dla wszystkich potrzebnych rejestrów i przypisać im odpowiednie adresy oraz zdefiniować nazwy do wszystkich wykorzystywanych peryferiów (np. LED1, LED2). Wykorzystać adresy bazowe oraz odpowiednie przesunięcia. Zdefiniować dodatkowe nazwy dla konkretnych bitów (np. dla diod LED, bitów w rejestrach konfiguracyjnych).

Stworzyć procedury do konfiguracji oraz zapalania i gaszenia diod. W funkcji main nie mogą znajdować się instrukcje poza skokami do odpowiednich podprocedur. Poprawność programu sprawdzić za pomocą debuggera (lub przejść do ćwiczenia 2 z pętlą opóźniającą).

main

BL konfiguracja

BL turn_on_led1

BL turn_off_led1

...

Język: Asembler

Zasoby: porty I/O

Weryfikacja: Przy użyciu środowiska Keil

Ćwiczenie 2

Temat: Pętla opóźniająca, skoki warunkowe i obsługa stosu

Zadanie: Do programu z ćwiczenia 1 dodać pętlę opóźniającą, która opóźni zapalenie kolejnych diod o czas równy ok. $(500+50*N)$ ms. Domyślną częstotliwość zegara sprawdzić w karcie katalogowej mikrokontrolera (w przypadku braku konfiguracji pętli PLL)*. Wszystkie instrukcje wykonujące określoną funkcjonalność umieścić w funkcjach, które będą wywoływane w programie głównym (w funkcji main mogą znajdować się wyłącznie skoki do podprocedur oraz pętla główna). Wykorzystać do tego instrukcję skoku z zachowaniem adresu powrotu. Należy zwrócić uwagę na ilość skoków do funkcji aby nie utracić adresu powrotu (rejestr LR będzie nadpisywany). Zaimplementować obsługę stosu do zachowania rejestru LR w każdej podprocedurze.

led_loop

```
PUSH ...  
BL    turn_on_led1  
BL    delay  
BL    turn_off_led1  
POP   ...  
...
```

main

```
BL    konfiguracja  
BL    led_loop  
...
```

Czas który upłynął można zweryfikować w oknie Registers, wartość Sec. Jednocześnie można zweryfikować ile cykli potrzebnych jest na wykonanie danej instrukcji (wartość States).

Zweryfikować które instrukcje są wykorzystywane po kompilacji (16 czy 32-bit) oraz sprawdzić ile cykli jest wymagane na ich wykonanie. Informacje dostępne są na stronie przedmiotu. W kodzie umieścić obliczenia dla uzyskania odpowiedniego opóźnienia pętli.

Język: Asembler

Zasoby: porty I/O

Weryfikacja: Przy użyciu środowiska Keil

* zweryfikować w ustawieniach projektu, zakładka Target, czy częstotliwość oscylatora Xtal jest ustawiona na 12MHz

Ćwiczenie 3

Temat: Pamięć RAM i FLASH

Zadanie: Do programu z ćwiczenia 2 dodać nowy plik asemblera, w którym będą umieszczone wartości początkowe zmiennych dla pamięci RAM. Zdefiniować za pomocą dyrektywy zmienną w pamięci FLASH (IROM1 – zobacz konfiguracja środowiska) oraz odpowiadającą zmienną w pamięci RAM (IRAM1). W ustawieniach nowo utworzonego pliku (prawy klawisz myszy na pliku → options for file → memory assignment) nadpisać miejsce zapisu jako pamięć IROM1. Zmienne mają być wykorzystywane do przechowywania licznika dla funkcji opóźniającej z ćwiczenia 2 (*). Przypisać zadaną wartość początkową do zmiennej w pamięci FLASH za pomocą dyrektywy. Skopiować wartość zmiennej z pamięci FLASH do pamięci RAM. Skonfigurować dwa piny procesora jako wejście, do którego podłączone są przyciski (microswitch) pamiętając o włączeniu w rejestrach SCU bufora wejściowego (input buffer) i filtra przeciwzakłóceń (glitch filter). Przyciski mają dwukrotnie zwiększać/zmniejszać czas opóźnienia poprzez modyfikację zmiennej w pamięci RAM. Ograniczyć zakres zmiany wartości zmiennej: np. trójkrotne zwiększenie/zmniejszenie o 2x poczynając od wartości początkowej.

Język: Asembler

Zasoby: porty I/O, RAM, Flash

Weryfikacja: Przy użyciu środowiska Keil

*Pamięć RAM nie umożliwia inicjowania zadaną wartością dlatego należy skorzystać z pamięci FLASH, która jest inicjowana podczas programowania. Z kolei pamięć FLASH nie umożliwia zapisu więc musi zostać wykonana kopia do pamięci RAM.

Ćwiczenie 4

Temat: Przerwania

Zadanie: Zmodyfikować program z ćwiczenia 3 dodając obsługę przerwania od przycisków (GPIO pin interrupt). Przypisać przerwanie od portu GPIO do odpowiedniego pinu (rejestr SCU), odpowiednio skonfigurować przerwania tj. sposób wyzwalania zgodnie z pożądanym efektem. Etykieta funkcji obsługi przerwania jest zdefiniowana w pliku startup.s. Wszystkie operacje związane z obsługą przycisku powinny znajdować się w funkcji obsługi przerwania. Należy pamiętać o obsłudze stosu w każdej podprocedurze.

Język: Asembler

Zasoby: porty I/O, RAM, Flash, stos, przerwania

Weryfikacja: Przy użyciu środowiska Keil

Ćwiczenie 5

Temat: Przetwornik ADC, jednostka FPU

Zadanie: Zapoznać się z projektem realizującym obsługę przetwornika ADC na przerwaniu `01_ADC / adc_interrupt`. Do projektu dodać obsługę portu GPIO realizującego wyświetlanie wartości odczytanej z przetwornika na diodach LED w postaci słupkowej (bar graph). Wykorzystać projekt `12_GPIO`, w którym znajduje się wykorzystanie funkcji do obsługi portu GPIO. Wyświetlanie wartości należy zrealizować w funkcji obsługi przerwania od przetwornika. Do obliczeń należy wykorzystać zmienne typu float.

W projekcie należy zakomentować następujące funkcje w funkcji głównej `c_entry()`: `CGU_init()`; `debug_frmwrk_init()`; `print_menu()`; oraz wszystkie funkcje służące do obsługi UART z pętli głównej (`_DBG`, `_DBD`, itp.).

W przypadku problemów z obsługą zmiennych zmiennoprzecinkowych tj. zawieszenie się procesora po wykonaniu instrukcji `VPUSH`, `VPOP`, przed wywołaniem funkcji `c_entry()`; należy wywołać funkcję `fpuEnable()`. Należy pamiętać o zdefiniowaniu symbolu `USE_FPU` w ustawieniach projektu (→ konfiguracja środowiska).

W przypadku gdy w procesie kompilacji nie zostaje znaleziona funkcja `fpuEnable()`, należy ją umieścić w kodzie:

```
void fpuEnable()
{
    SCB->CPACR |= ((3UL << 10*2) | /* set CP10 Full Access */
                  (3UL << 11*2)); /* set CP11 Full Access */
}
```

Sprawdzić w debuggerze czas wykonywania jednego z obliczeń na liczbach zmiennoprzecinkowych w przypadku wykorzystania jednostki FPU oraz bez niej. W ustawieniach projektu → Target włączyć/wyłączyć jednostkę FPU pamiętając o rekompilacji projektu.

Język: C

Zasoby: porty I/O, przerwania, ADC, FPU

Weryfikacja: Przy użyciu środowiska Keil

Ćwiczenie 6

Temat: Licznik SysTick, tryb uśpienia

Zadanie: Zapoznać się z działaniem licznika SysTick, który najczęściej jest wykorzystywany w systemach operacyjnych. Napisać program, który w pętli głównej wprowadza mikrokontroler w stan uśpienia (sleep mode) i jedynym źródłem wybudzenia jest przerwanie od licznika SysTick. Skonfigurować licznik SysTick tak by przerwanie wywoływało się co jedną milisekundę (funkcja *SysTick_Config()* znajdująca się w pliku *core_cm4.h*). Należy sprawdzić z jaką częstotliwością taktowany jest ten licznik oraz jak jest skonfigurowany układ do generowania sygnału zegarowego (o ile taka konfiguracja została użyta np. za pomocą funkcji *SystemInit()*, w przeciwnym wypadku należy użyć domyślnej konfiguracji). W funkcji obsługi przerwania od tego licznika należy wyłącznie dekrementować zmienną globalną msec, która w programie głównym ma być ustawiana na wartość ms, po której ma się zmieniać stan na wyjściu, do którego podłączona jest dioda LED.

Język: C

Zasoby: porty I/O, przerwania, SysTick, sleep

Weryfikacja: Przy użyciu środowiska Keil