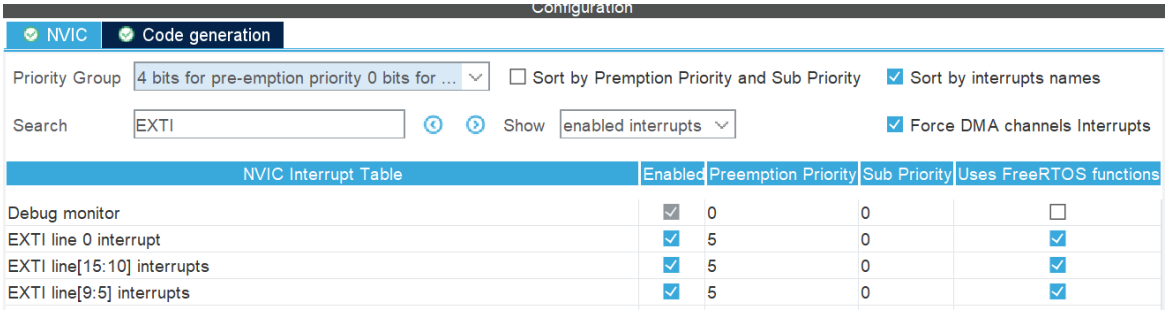
FreeRTOS 3/3

Przebieg ćwiczenia

1. **Wstęp**
   Zmodyfikuj projekt dodając:
   - zadania: Task1, Task2, Task3, Task4 o normalnych priorytetach
   - semafory binarne: Sem1, Sem2, Sem3, Sem4
   - muteksy: Mux1, Mux2, Mux3, Mux4.
2. **Semafor binarny**
   Dodaj poniższą funkcję w pliku *freertos.c*. Jej zadaniem jest czasowe zapalenie cyfry na wskazanej pozycji. Ogranicz dostęp do wspólnego zasobu.
   void **DISP_SetDigitOnTime**(uint8_t digit, uint8_t pos, uint32_t dispTime)
3. **Semafor binarny i zliczający**
   Wykorzystując funkcję z poprzedniego zadania przygotuj zadania wg następującego schematu:
   Task1 zapala segment 1 na 500ms z przerwą 2500ms.
   Task2 zapala segment 2 na 500ms z przerwą 2500ms.
   Task3 zapala segment 3 na 500ms z przerwą 2500ms.
   Task4 zapala segment 4 na 500ms z przerwą 2500ms.
   Przetestuj działanie i wyjaśnij wydłużone świecenie jednego z segmentów.
   Nastepnie ustaw parametr USE_COUNTING_SEMAPHORE na Enabled (FreeRTOS->Config parameteres)
   Zmodyfikuj rozmiar bufora semafora(pojemność). Modyfikacje przeprowadź w miejscu inicjalizacji semafora (plik freertos.c). Za pomocą debuger wyjaśnij działanie programu.
4. **Muteks a semafor – różnice**
   Dodaj przerwania od pinów BUT_LEFT i BUT_RIGHT. Ustawienie pinu jako GPIO_EXITx (nie GPIO_Input).
   Skonfiguruj w zakładce GPIO piny BUT_LEFT i BUT_RIGHT jak poniżej.

PB5	n/a	n/a	External Interrupt Mode with Falling edge ...	Pull-up	n/a	BUT_LEFT	☑
PB0	n/a	n/a	External Interrupt Mode with Falling edge ...	Pull-up	n/a	BUT_RIGHT	☑

   Odblokuj przerwania w kontrolerze NVIC zgodnie z poniższym zrzutem ekranu.



   Zadeklaruj w pliku *freertos.c* funkcję:
   __weak void **HAL_GPIO_EXTI_Callback**(uint16_t GPIO_Pin)
   Na podstawie otrzymanego argumentu w powyższej funkcji przygotuj:
   - reakcja na przycisk BUT_LEFT zwalnia Sem1.
   - reakcja na przycisk BUT_RIGHT zwalnia Sem2.

   Zadanie Task1 skonfiguruj do działania w oparciu o Sem1 i Sem2. Po pojawieniu się Sem1 wyświetl dowolną liczbę. Następnie zadanie oczekuje na Sem2, po jego odblokowaniu wyłącza wyświetlacz. Po poprawnym uruchomieniu przykładu podmień mechanizm semafora na muteks. Wyjaśnij różnice.

Zadanie dodatkowe

Task1 odpowiada za cykliczne odczytywanie jasności oświetlenia (cykl 1s) i zapisywanie go w zmiennej globalnej.
Task2 odpowiada za przygotowanie danych do wyświetlenia w Task3. Posiada trzy tryby: bezpośrednia wartość z ADC, zmapowana do skali 0-100%, czas w sekundach od uruchomienia mikrokontrolera.
Task3 odpowiada za multipleksowanie wyświetlacza.
Do zmiany aktualnego trybu wyświetlania wykorzystaj mechanizm przerwań.

Funkcje

HAL

- void **HAL_GPIO_WritePin** (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)
- GPIO_PinState **HAL_GPIO_ReadPin** (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)
- void **HAL_Delay** (uint32_t Delay)
- uint32_t **HAL_GetTick** (void)
- HAL_StatusTypeDef **HAL_ADC_Start** (ADC_HandleTypeDef * hadc)
- HAL_StatusTypeDef **HAL_ADC_PollForConversion** (ADC_HandleTypeDef * hadc, uint32_t Timeout)
- uint32_t **HAL_ADC_GetValue** (ADC_HandleTypeDef * hadc)
- __weak void **HAL_GPIO_EXTI_Callback**(uint16_t GPIO_Pin)

FreeRTOS

- osStatus **osDelay** (uint32_t millisec)
- int32_t **osSemaphoreWait** (osSemaphoreId semaphore_id, uint32_t millisec)
- osStatus **osSemaphoreRelease** (osSemaphoreId semaphore_id)
- osStatus **osMutexWait** (osMutexId mutex_id, uint32_t millisec);
- osStatus **osMutexRelease** (osMutexId mutex_id);

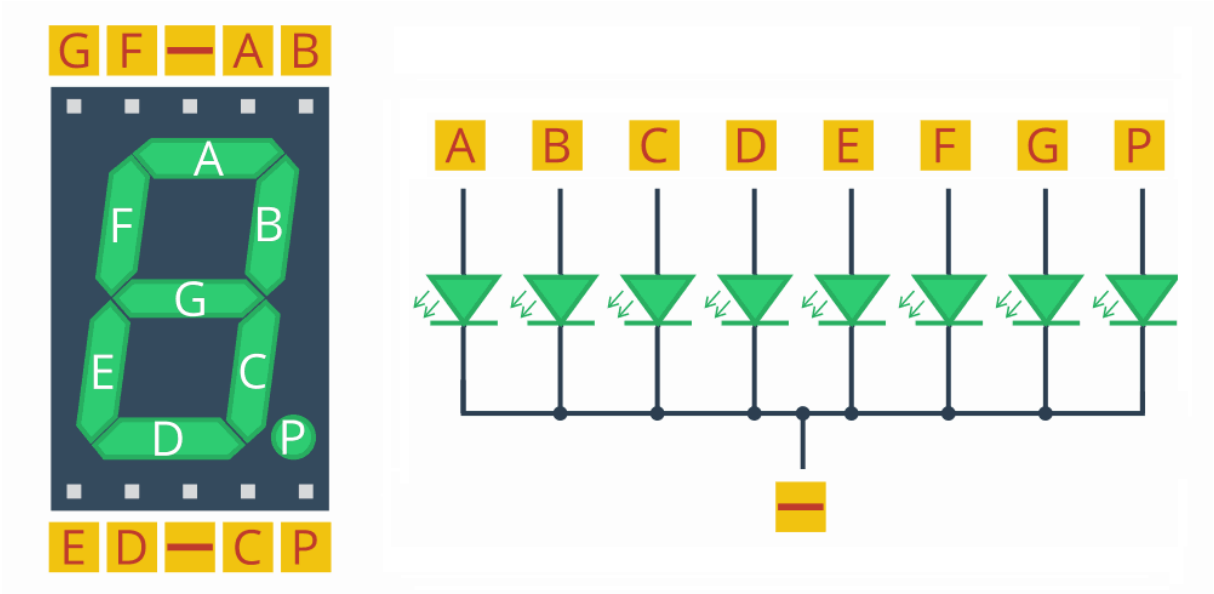
Display

- void **DISP_SetSegments**(enum dispSeg seg)
- void **DISP_SetPositions**(enum dispPos pos)
- void **DISP_SetDigit**(uint8_t digit)

Materiały dodatkowe

- Description of STM32F4 HAL and low-layer drivers - User manual
- FreeRTOS API categories
- KA-Nucleo-Multisensor (PL) (kamamilabs.com)

Wyświetlacz 7-segmentowy



STM32CubeMX