Systemy wbudowane (zaoczne)

Kameleon Board

Konfiguracja środowiska STMCube

Device: STM32L496ZGT6
Ścieżka repozytorium: Windows → Preferences → STM32Cube → Firmware Updater → Repository Setup → Browse → ustawić ścieżkę „C:\STM32Cube\Repository”

Informacje dot. Płytki Kameleon Board

https://kameleonboard.org/
Nota katalogowa https://kameleonboard.org/wp-content/uploads/STM32L496ZGT6-Datasheet.pdf
Manual https://kameleonboard.org/wp-content/uploads/STM32L496ZGT6-Reference-Manual.pdf

→ Drivers → STM32L4xx_HAL_Driver – definicje funkcji dla odpowiednich peryferiów
→ Drivers → Cmsis – definicje adresów rejestrów

Ćwiczenie 1

Temat: Obsługa portu wejście/wyjście – diody LED (2h)

Zadanie: Napisać program, który będzie zapalał po kolei diody w nieskończonej pętli.
Skonfigurować odpowiednie piny do których podłączone są diody:
LED 0 → GPIO_C[6]
LED 1 → GPIO_C[7]
LED 2 → GPIO_C[8]
LED 3 → GPIO_C[9]
LED 4 → GPIO_E[4]
LED 5 → GPIO_D[3]
LED 6 → GPIO_E[5]
LED 7 → GPIO_E[6]
Wykorzystać do tego funkcję GPIO_Init(), utworzyć odpowiednią strukturę podając parametry:
MODE → Output push pull mode (wyjście cyfrowe)
SPEED → FREQ_LOW (niska częstotliwość <5MHz)
PULL → odpowiednie użycie rezystorów podciągających (sprawdzić schemat połączenia elektrycznego).
Włączyć zegar dla odpowiednioego portu korzystając z funkcji (konfiguracja w RCC) *__HAL_RCC_GPIOx_CLK_ENABLE(). (*x odpowiada numerowi portu*)*
Opóźnienie zrealizować poprzez wykorzystanie funkcji *HAL_Delay()*. Należy pamiętać o wywołaniu funkcji *HAL_Init()*, która konfiguruje timer SysTick wykorzystywany w funkcji opóźniającej.
Dla ułatwienia wykorzystać definicje z plików nagłówkowych dotyczących odpowiednich peryferii.

Język: C
Zasoby: porty I/O
Pliki do modyfikacji: Core → Src → main.c

Ćwiczenie 2

Temat: Obsługa przerwań i zegara SysTick – dioda RGB (4h)

Zadanie: Napisać program, który będzie wyświetlał różne kolory z wykorzystaniem diody RGB. Zmiana kolorów powinna odbywać z wykorzystaniem joysticka, Skonfigurować piny (konfiguracja analogicznie jak w ćwiczeniu 1), do których podłączona jest dioda RGB:
LED 0 → GPIO_D[12]
LED 1 → GPIO_D[13]
LED 2 → GPIO_B[8]
oraz piny, do których podpięty jest joystick:
JOY_RIGHT → GPIO_E[0]
JOY_LEFT → GPIO_E[1]
JOY_UP → GPIO_E[2]
JOY_DOWN → GPIO_E[3]
JOY_PUSH → GPIO_E[15]
używając następującej konfiguracji:
MODE → Input (wejście cyfrowe)
PULL → odpowiednie użycie rezystorów podciągających (sprawdzić schemat połączenia elektrycznego)
Wywołać funkcję *HAL_Init()*, która konfiguruje timer SysTick wykorzystywany w funkcji opóźniającej.
Wykorzystać przerwanie od timera SysTick do obsługi joysticka. Stan joysticka należy sprawdzać z odpowiednią częstotliwością by nie było zauważalne opóźnienie. Należy pamiętać o uwzględnieniu możliwości użycia joysticka w jednej pozycji przez dłuższy czas. Aktualizację sygnałów sterujących diodą RGB należy wykonywać przy każdym sprawdzaniu stanu joysticka.
Dioda RGB składa się z trzech oddzielnie sterowanych diod R, G i B. Jasność danego koloru realizuje się poprzez podanie sygnału PWM o zadanej częstotliwości (odpowiednio dużej by niewidoczne było miganie) oraz wypełnieniu D odpowiadającym jasności świecenia. Sprawdzić który stan odpowiada maksymalnej jasności, D=0% czy D=100%. Do generowania sygnału PWM wykorzystać przerwanie od timera SysTick (funkcje są dostępne w pliku Core → Src → stm32_it.c). Jeśli będzie wymagane to należy zmienić częstotliwość generowania przerwania od timera SysTick.
Sposób zmiany kolorów dowolny np.:

- predefiniowane n kolorów, zmiana w lewo i w prawo
- inkrementacja jasności każdego z kolorów (lewo, prawo, góra) + reset (dół)
- inne…

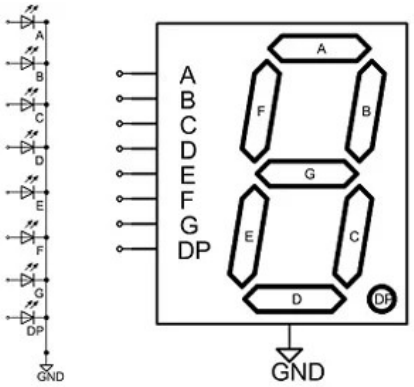
Wykorzystać ogólnodostępne tablice kodowania kolorów w palecie RGB. Wskazówka: sygnał PWM dopasować do liczby bitów opisujących każdą ze składowych. Dla ułatwienia standardowe 8bit na składową można zmniejszyć na mniejszą liczbę np. 2bit.
W programie użyć definicji nazw, do których przypisać wartości odpowiadające częstotliwości sprawdzania stanu joysticka oraz częstotliwości generowanego sygnału PWM.

Język: C
Zasoby: porty I/O, SysTick
Pliki do modyfikacji: Core → Src → main.c; Core → Src → stm32_it.c

Ćwiczenie 3

Temat: Zegar cyfrowy – wyświetlacz 7-segmentowy (2h)

Zadanie: Napisać program, który będzie realizował funkcję zegara cyfrowego z wykorzystaniem wyświetlacza 7-segmentowego. Format zegara powinien być następujący:
HH.MM. → godzina.minuta; prawa kropka powinna migać wskazując na sekundy
MM.SS → minuta.sekunda
Środkowa kropka powinna być zapalona na stałe. Przełączanie powinno odbywać się za pomocą przycisku. Sterowanie jednej cyfry jest zgodne ze schematem poniżej:
Połączenie segmentów:



A → GPIO_G[0]
B → GPIO_G[1]
C → GPIO_G[2]
D → GPIO_G[3]
E → GPIO_G[4]
F → GPIO_G[5]
G → GPIO_G[6]
DP → GPIO_G[9]
Sterowanie cyframi jest realizowane poprzez multipleksowanie czyli w danej chwili można wyświetlać tylko jedną cyfrę. Linie dla kazdego z segmentów są zatem sterowane z tych samych pinów dla każdej z cyfr. Wybór cyfry polega na włączeniu odpowiadającego tranzystora NPN (patrz schemat elektryczny → Prints of displays). Bazy tranzystorów są podłączone następująco:
DIG_1 → GPIO_B[2] (cyfra z lewej strony wyświetlacza)
DIG_2 → GPIO_B[3]
DIG_3 → GPIO_B[4]
DIG_4 → GPIO_B[5] (cyfra z prawej strony wyświetlacza)
Przełączanie pomiędzy cyframi powinno odbywać się z częstotliwością taką by ludzkie oko nie obserwowało włączania/wyłączania cyfry.
Do uzyskania bardziej przejrzystego kodu należy zdefiniować w tablicy wartości, które po wysłaniu na port będą odpowiadały segmentom potrzebnym do wyświetlenia zadanej cyfry:

```
...
#define     SEG_C     2     //pin do którego pospięty jest dany segment
…

const uint8_t segments[] = {
    1<<SEG_A | 1<<SEG_B | 1<<SEG_C | 1<<SEG_D | 1<<SEG_E | 1<<SEG_F, // 0
    …    //1
    …
    …    //9
    }
```

Zdefiniowane tablicy w kolejności odpowiadającej cyfrom ułatwi proste odwołanie do tablicy, którego indeksem będzie cyfra do wyświetlenia.
Wykorzystać przerwanie od timera SysTick zarówno do przełączania pomiędzy cyframi jak i aktualizacji wyświetlanego czasu na wyświetlaczu oraz obsługi przycisku. Nie wolno korzystać z funkcji opóźniającej. Kod reazliujący powyższe funkcjonalności powinien zostać

umieszczony wyłącznie w funkcji obsługi przerwania. Pętla główna w funkcji main powinna być pusta.
UWAGA! Aby port GPIO_G[2:15] działał poprawnie, należy włączyć dodatkowe zasilanie VDDIO2 za pomocą następującej funkcji:
HAL_PWREx_EnableVddIO2();

Język: C
Zasoby: porty I/O, SysTick

Pliki do modyfikacji: Core → Src → main.c; Core → Src → stm32_it.c

Ćwiczenie 4

Temat: Obsługa Timera i przerwań

Zadanie: Zmodyfikować program z ćwiczenia 3 dodając obsługę licznika ogólnego przeznaczenia Timer2 wraz z przerwaniami i przenieść całą funkcjonalnośc programu do przerwania od tego Timera. Konfigurację licznika zrealizować za pomocą automatycznej konfiguracji dostępnej w środowisku. W drzewie projektu otworzyć plik "nazwa_projektu.ioc". W zakładce Pinout &Configuration:

- włączyć i skonfigurować Timer2 (Timers → TIM2)
  - tryb pracy „Output Compare No Output"
  - Ustawić wartość do której ma liczyć licznik (Counter Period) zgodnie z ustawionym zegarem dla tego licznika (sprawdzić w zakładce Clock Configuration) oraz preskalerem
  - włączyć automatyczne uzupełnianie powyższej wartości oraz poziom wysoki przy poprawnym porównaniu
- włączyć globalne przerwanie od Timera 2 w ustawieniach kontrolera przerwań (System Core → NVIC)

Po zapisaniu ustawień program powinien uaktualnić kod programu dodając odpowiednie funkcje konfiguracyjne.
Korzystając z pliku nagłówkowego dostępnego w drzewie projektu Drivers → STM32_HAL_Driver → Inc → stm32_hal_tim.h znaleźć nazwy odpowiednich funkcji wymaganych poniżej. Dla ułatwienia funkcje są pogrupowane zgodnie z ich funkcjonalnością i oznacvzone odpowiednim komentarzem.
W funkcji głównej programu należy uruchomić Timer 2 z obsługą przerwań za pomocą odpowiedniej funkcji (Time Base functions).
W fukcji głównej obsługi przerwania od Timera 2 → TIM2_IRQHandler() wywoływana jest główna funkcja obsługi przerwania, która przekierowuje do odpowiedniej funkcji żależnie od źródła przerwania (czyli ustawionej flagi) dla danego Timera. Należy zatem znaleźć nazwę funkcji, która odpowiada ustawionemu trybowi pracy (Callback in non blocking modes (Interrupt and DMA)) i zdefiniować taką funkcję, w której należy umieścić wywołanie kodu realizującego całą funkcjonalnośc programu.
Można skorzystać z poniższego dostęponego przykładu: https://deepbluembedded.com/stm32-timer-interrupt-hal-example-timer-mode-lab/ .

Język: C
Zasoby: porty I/O, SysTick, Timer2, przerwania

Pliki do modyfikacji: Core → Src → main.c; Core → Src → stm32_it.c

Ćwiczenie 5

Temat: Obsługa akcelerometru MEMS

Zadanie: Napisać program do obsługi akcelerometru MEMS. Wykorzystać do tego bibliotekę dostarczaną przez producenta płytki, która zawiera funkcje do obsługi tego czujnika. Wartość przyspieszenia w danej osi należy wyświetlić na wyświetlaczu 7-segmentowym. Zmianę osi zrealizować za pomocą joysticka lub przycisków. Sposób wyświetlania dowolny. Wartość może być wyświetlana w jednostkqach *m/s* lub w *g*. Pierwszy znak może oznaczać wybraną oś, przecinek może się zmieniać zależnie od liczby cyfr np.: Xab.c, Ya.bc. Odświeżanie wartości zrealizować z dowolną częstotliwością. Wykorzystać przerwanie od SysTicka lub Timera2. Wykorzystać kod z ćwiczenia 3 i/lub 4.
Biblioteki znajdują się na stronie płytki w zakładce Kameleon Library. Wykorzystqać bibliotekę *kamami_l496_mems.h*. Pliki h i c należy skopiować do odpowiadających katalogów w katalogu projektu (include i source). W pliku main.c należy dołączyć plik h a plik c powinien być widoczny w drzewie projektu.
Na stronie płytki w zakładce Kameleon library reference znajduje się opis funkcji wbudowanych. Można również wykorzystać przykładowy projekt *kamami_l496_mems_vcom* znajdujący się w pliku z bibliotekami.

Język: C
Zasoby: porty I/O, SysTick, Timer2, przerwania

Pliki do modyfikacji: Core → Src → main.c; Core → Src → stm32_it.c