Katedra Biosensorów i Przetwarzania Sygnałów Biomedycznych

Wydział Inżynierii Biomedycznej

POLITECHNIKA ŚLĄSKA

Systemy Wbudowane i Mobilne w Biomedycynie

Laboratorium 2

Sekcja 1: Suchanek Kamil, Piecuch Edyta

Kierunek studiów: Inżynieria Biomedyczna

Specjalność: *Przetwarzanie i Analiza Informacji Biomedycznej*

Prowadzący laboratorium: mgr inż. Marek Czerw

ZABRZE – 2020

Fragment instrukcji z notatkami:

“Laboratorium z Systemów Wbudowanych i Mobilnych w Biomedycynie

Ćwiczenie 2

Zadanie dotyczy zastosowania jednego z najbardziej podstawowych wbudowanych zasobów w mikrokontrolerze jakim jest Timer.(czyli można użyć tylko jednego) Zadanie wspólne dla wszystkich sekcji dotyczy rozpoznania typów timera(??), aby przy ich użyciu (TIM1,TIM3, TIM6 oraz Systick) zaprogramować regulowane odcinki czasowe które będą generować sygnały charakterystyczne dla kodu Morsea, w tym przypadku błysków świetlnych diody LED. Podstawowym do zdefiniowania parametrem jest czas trwania kropki (dt), ponieważ pozostałe parametry są od tego uzależnione. I tak

- Kreska powinna trwać co najmniej tyle czasu, co trzy kropki. ok
- Odstęp pomiędzy elementami znaku powinien trwać jedną kropkę. ok (w tym, że chodzi o “znaki” kropka/kreska, “intra-character space”)
- Odstęp pomiędzy poszczególnymi znakami trzy kropki. ok (w tym, że chodzi o znaki alfanumeryczne, “inter-character space”)
- Odstęp pomiędzy grupami znaków (słowami) – siedem kropek. ok

Proszę dla diody LED zaprogramować zdanie charakterystyczne dla danej sekcji:

- Sekcja 1 – „Halo tu sekcja pierwsza”,

Dodatkowo (dla lepszej oceny) przy użyciu aplikacji Morse Tool, niech każda sekcja wykaże jaki można ustawić minimalny czas trwania kropki.

Jak poprzednio zadanie należy wykonać przy użyciu zestawu ewaluacyjnego STM32F0Discovery, który posiada elementy składowe interfejsu użytkownika przewidziane do wykorzystania:

- Przycisk użytkownika USER - B1 (Blue PushButton)
- Dioda LED zielona – Led3 (Green Led)

Przy użyciu przycisku należy rozpocząć proces generowania błysków świetlnych, z których powstanie jedno zdanie charakterystyczne dla każdej sekcji.
ok, czyli zadanie polega na generowaniu jednego komunikatu po wciśnięciu przycisku, szybkość generowania powinna być konfigurowalna, tzn podstawowy okres czasu przypadający na jedną “ciszę”, czy jedną “kropkę”, musi dać się zmieniać (chociażby dzieląc odpowiednio krok Timera czy dodając licznik do uchwytu przerwania). Prędkość w WPM nie ma w przypadku tego zadania zastosowania. Kontrola czasu generowanych sygnałów ma zostać wprowadzona przy pomocy timera bez dynamicznego obliczania prędkości migania dla konkretnego słowa jak to ma miejsce dla szybkości w WPM.

SPIS TREŚCI:

1. Projekt realizacji zadania

W celu realizacji ćwiczenia zaimplementujemy małą maszynę stanów. Program uruchamiany w przerwaniu będzie sprawdzał aktualny stan (Dioda zapalona, zgaszona, postój) oraz dopasowywał do tego wykonywane czynności.

Program będzie zaopatrzony w gotowy wektor wartości z wartościami 1, 3, 2, 6, oznaczających odpowiednio kropkę, kreskę, odstęp między znakami alfanumerycznymi, oraz odstęp pomiędzy słowami. Podstawowy odstęp pomiędzy znakami kodu Morse'a nie zostały zawarte w tej reprezentacji ze względu na sposób działania programu. Po każdym znaku kropki/kreski musi nastąpić przynajmniej jeden okres bez sygnału i zostało to uwzględnione w wartościach wektora dla odstępów, tzn.: odstęp 3 okresów ma wartość 2 zamiast 3, a odstęp siedmiu okresów ma wartość 6 zamiast 7.

Zastosowany sposób nie jest sztywnym rozwiązaniem, każdą wartość 1, 3, 2, 6 można uznać za "zadanie" i analizując zadaną wiadomość jako tablicę typu char można przechodząc element po elemencie dodawać to "zadanie" do schedulera. Jednak zadaniem na laboratorium jest poradzić sobie z Timerem w celu generowania odpowiednich, kontrolowanych czasowo zdarzeń.

1.1. Konfiguracja graficzna

W tej części tworzenia projektu w STM32CubeIDE skonfigurowano jedynie niebieski przycisk w celu wywoływania przerwania zewnętrznego, jak na poprzednim laboratorium. Dodatkowa konfiguracja graficzna Timerów nie była wymagana.

2. Realizacja zadania

2.1. Plik main.c

Plik main.c uzupełniono jedynie o ustawienie kroku Timera, inicjalizację diody zielonej i załączenie dodatkowego pliku do projektu.

```
#include "stm32f0308_discovery.h"
SysTick_Config((SystemCoreClock/10))
BSP_LED_Init(LED_GREEN);
```

Ustawiono Timer tak aby wykonywał przerwanie 10 razy na sekundę.

2.2. Plik stm32f0xx_it.c

Dodanie pliku z wygodnymi funkcjami do obsługi diod:

```
#include "stm32f0308_discovery.h"
```

Zdefiniowanie stanów:

```
typedef uint8_t STATE;
#define LIGHT_OFF 0
#define LIGHT_ON 1
#define WAITING 2
```

Tablica z wiadomością, rozciągnięta aby wykluczyć proste błędy.

```
uint8_t msg_t[] = {
        1,1,1,1,2,      /* H */
        1,3,2,          /* a */
        1,3,1,1,2,      /* l */
        3,3,3,6,        /* o */
        3,2,            /* t */
        1,1,3,6,        /* u */
        1,1,1,2,        /* s */
        1,2,            /* e */
        3,1,3,2,        /* k */
        3,1,3,1,2,      /* c */
        1,3,3,3,2,      /* j */
        1,3,6,          /* a */
        1,3,3,1,2,      /* p */
        1,1,2,          /* i */
        1,2,            /* e */
        1,3,1,2,        /* r */
        1,3,3,2,        /* w */
        1,1,1,2,        /* s */
        3,3,1,1,2,      /* z */
        1,3,6           /* a */
};
```

Zmienne globalne:

```
uint8_t send = 0; //  0 albo 1, nie wysyłaj, wyślij wiadomość
uint8_t el = 0; // aktualny element wektora
uint8_t size_of_msg = sizeof(msg_t)/sizeof(msg_t[0]); // rozmiar wektora wiadomości
int8_t counter = -1; // licznik po elementach wektora
STATE state = LIGHT_OFF; // inicjalizacja pierwszego stanu
uint8_t my_delay = 0; // dodatkowy licznik wydłużający okresy pomiędzy którymi
                      wywoływane są funkcje “send_message()”
```

Funkcja send_message() wywoływana wewnątrz przerwania wewnętrznego SysTick:

```
void send_message(){
    switch(state){
    case LIGHT_OFF:
                        counter = counter + 1; // zwiększ licznik
                        el = msg_t[counter]; // odczytaj kolejny element
                        ziel(); // wykonaj akcje, action() nazwa była zajęta...
                        break; // następny okres
    case LIGHT_ON:
                        if(el < 1){ // jeśli okres świecenia się skończył..
                                BSP_LED_Off(LED_GREEN); // zgaś
                                state = LIGHT_OFF; // zmień stan
                        }else{ // jeśli dalej trzeba świecić (kreska):
                                el = el - 1;
                        }
                        break; // następny okres
    case WAITING:
                        if(el < 1){ // jeśli okres postoju się skończył
                                counter = counter + 1;
                                if(counter <= size_of_msg){ // i nie dotarliśmy do końca
                                        el = msg_t[counter];
                                        ziel();
                                }else{ // jeśli jednak to ostatni to po prostu zatrzymaj
                                        send = 0;
                                }
                        }else{ // czekaj dalej
                                el = el - 1;
                        }
                        break; // następny okres
    }
}
```

Funkcja ziel() (nazwa action() była chyba zajęta) uruchamiana w poszczególnych stanach, w których znajdzie się aplikacja:

```
void ziel(){
        switch(el){
                case 1:
                        BSP_LED_On(LED_GREEN);
                        state = LIGHT_ON;
                        el = el - 1;
                        break;
                case 3:
                        BSP_LED_On(LED_GREEN); // zapal
                        state = LIGHT_ON; // zmień stan
                        el = el - 1; // odlicz element
                        break; // wyjdź
                case 2:
                        state = WAITING;
                        el = el - 1;
                        break;
                case 6:
                        state = WAITING;
                        el = el - 1;
                        break;
        }
 }
```

Kod wewnątrz przerwania SysTick:

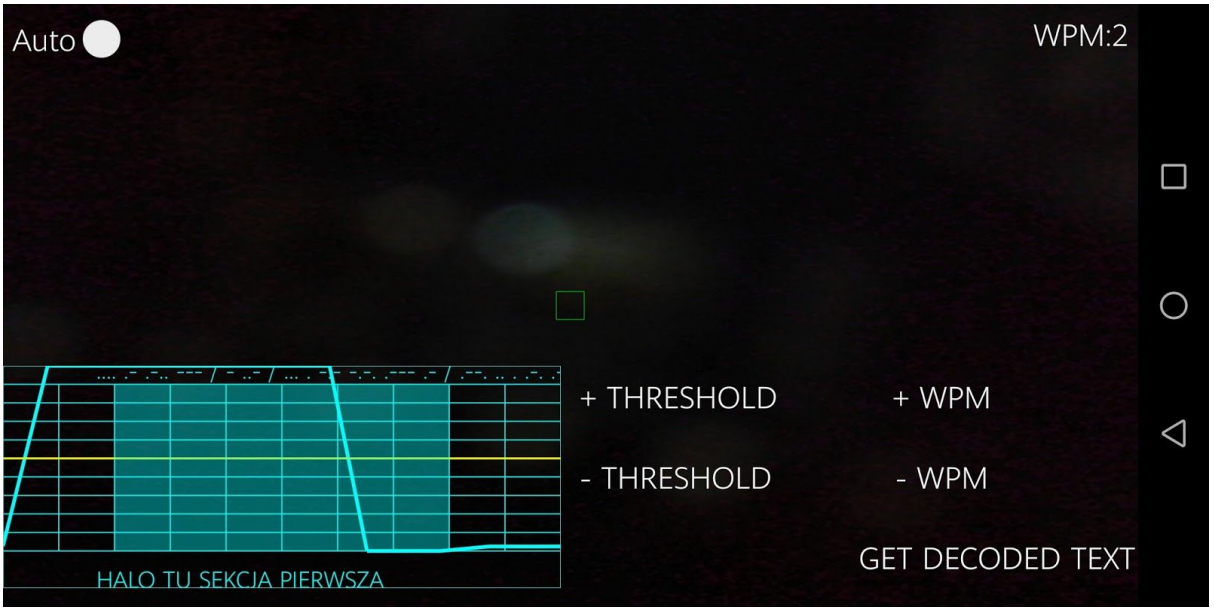
```
/* USER CODE BEGIN SysTick_IRQn 0 */
	my_delay++;
	if(my_delay > 4){
		if(send > 0){
			send_message();
		}
		my_delay = 0;
	}
/* USER CODE END SysTick_IRQn 0 */
```

Kod wewnątrz przerwania zewnętrznego:

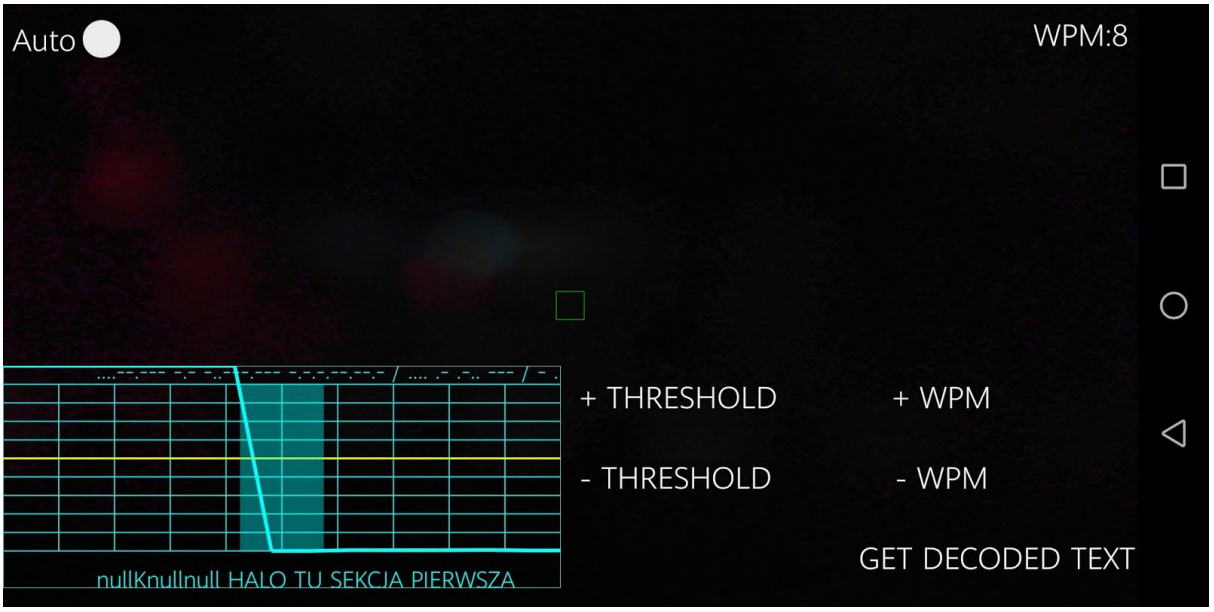
```
/* USER CODE BEGIN EXTI0_1_IRQn 0 */
    if(counter>=size_of_msg || counter < 0){
		send = 1;
		state = LIGHT_OFF;
		counter = -1;
    }
/* USER CODE END EXTI0_1_IRQn 0 */
```

3. Sprawdzenie minimalnego czasu trwania kropki

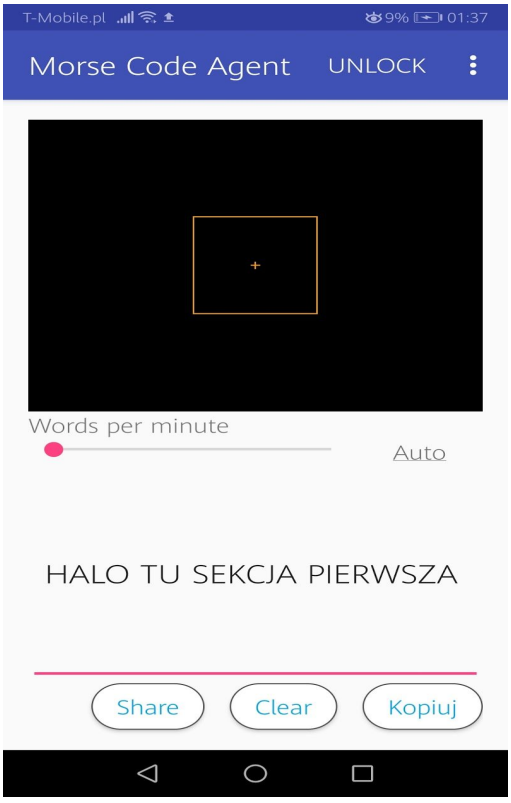
- Dla 2Hz (SystemCoreClock/10 i licznik = 5):



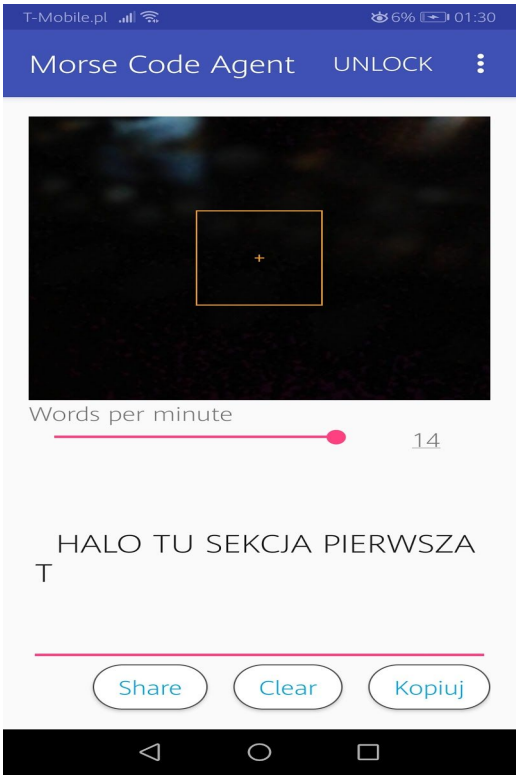
- Dla 5Hz (SystemCoreClock/10 i licznik = 2):



❖ Dla 10 Hz (sprawniejsza aplikacja, SystemCoreClock/10 bez licznika):



❖ Dla około 13Hz (ustawiono SystemCoreClock/1000 oraz licznik na 60) :



- Dla około 17Hz (ustawiono krok na SystemCoreClock/1000 oraz licznik na 75):

- Dla większych wartości aplikacja już nie rozróżniała błysków, czasem tylko strzępy słów.

3.1. Podsumowanie

Najkrótszy czas kropki uzyskano dla częstotliwości 16.6Hz.