2022

Sprawozdanie na przedmiot Systemy Wbudowane

ISI GRUPA 1 (ZAJĘCIA Z GRUPĄ 2)

SZYMON ŚLIWA 155471

Spis treści

Zadanie 1

Obsługa:

- Optymalne ustawienie zegara: 2MHz
- RB3: następne podzadanie
- RB4: poprzednie podzadanie

Przejście pomiędzy trybami

```
int setTask(int task, int change) {
    task = task + change;
    if (task == 10) {
        task = 1;
    } else if (task == 0) {
        task = 9;
    }
    return task;
}

unsigned char setDisplay(int task) {
    if (task == 2) {
        return 255;
    } else if (task == 9) {
        return 1;
    } else if (task == 1 || task == 3 || task == 7){
        return 0;
    }
}
```

Do przechodzenia pomiędzy trybami (podzadaniami) stworzyłem dwie funkcje. Pierwsza *setTask* przyjmuje jako parametry aktualne zadanie oraz stronę w którą będziemy się przesuwać, tzn. jeżeli chcemy przejść do kolejnego podzadania podajemy 1, jeżeli chcemy się cofnąć -1. Funkcja obsługuje przypadki w których jesteśmy aktualnie na pierwszym lub na ostatnim zadaniu. Wtedy odpowiednio przechodzi do ostatniego lub pierwszego zadania. Funkcja *setDisplay* ustala wartość zmiennej display odpowiednią dla początku działania odpowiedniego podzadania.

Implementacja tych dwóch funkcji w *mainie*, w nieskończonej pętli wygląda następująco:

```
if (PORTBbits.RB3 == 0) {    //task do przodu
    task = setTask(task, 1);
    display = setDisplay(task);
    if (task == 3) {
        gray = 1;
    } else if (task == 4) {
        gray = 255;
    } else if (task == 5) {
        bcd = 0;
    } else if (task == 6) {
        bcd = 99;
    } else if (task == 7) {
        direction = 1;
    } else if (task == 8) {
        sum = 1;
        added = 1;
    }
} else if (PORTBbits.RB4 == 0) {    //task do tylu
    task = setTask(task, -1);
    display = setDisplay(task);
    if (task == 3) {
        gray = 1;
    } else if (task == 4) {
        gray = 255;
    } else if (task == 5) {
        bcd = 0;
    } else if (task == 6) {
        bcd = 99;
    } else if (task == 7) {
        direction = 1;
    } else if (task == 8) {
        sum = 1;
        added = 1;
    }
}

if (task == 1) {
    display = bin_up(display);
} else if (task == 2) {
    display = bin_down(display);
} else if (task == 3) {
    display = gray_up();
} else if (task == 4) {
    display = gray_down();
} else if (task == 5) {
    display = bcd_up();
} else if (task == 6) {
    display = bcd_down();
} else if (task == 7) {
    display = snake(display);
} else if (task == 8) {
    display = queue();
} else {
    display = prng(display);
}
```

Jak można zauważyć na załączonym zrzucie ekranu, implementacja ta ustala również wartości zmiennych pomocniczych wykorzystywanych w odpowiednich zadaniach. Po sprawdzeniu czy przyciski odpowiadające zmianie podzadania na kolejne oraz poprzednie zostały wciśnięte, program rozpoczyna wywoływanie funkcji odpowiadającej działaniu odpowiedniego podzadania.

Odliczanie binarne do przodu

```
unsigned char bin_up(unsigned char display) {
    return (display + 1);
}
```

Funkcja przyjmuje jako argument aktualny stan zmiennej *display* i zwraca kolejną liczbę, która jest przypisywana do zmiennej *display* wewnątrz nieskończonej pętli.

Odliczanie binarne do tyłu

```
unsigned char bin_down(unsigned char display) {
    return (display - 1);
}
```

Funkcja identycznie jak w przypadku odliczania binarnego w górę przyjmuje jako argument aktualny stan zmiennej *display* i zwraca kolejną liczbę, która jest przypisywana do zmiennej *display* wewnątrz nieskończonej pętli.

Odliczanie kodem Graya w górę

```
unsigned char gray_up() {
    unsigned char display;
    display = (gray >>1) ^ gray;
    gray = gray + 1;
    return display;
}
```

Funkcja posługując się globalną zmienną pomocniczą *gray* przypisuje do wewnętrznej zmiennej *display* wynik operacji XOR pomiędzy przesunięciem bitowym o jeden w prawo wartości zmiennej *gray*, a wartością zmiennej *gray*, następnie dodaje do zmiennej *gray* 1 i zwraca wartość zmiennej *display*.

Odliczanie kodem Graya w dół

```
unsigned char gray_down() {
    unsigned char display;
    display = (gray >>1) ^ gray;
    gray = gray - 1;
    return display;
}
```

Funkcja wykonuje identyczne operacje jak funkcja *gray_up*, jedyną różnicą jest odejmowanie 1 od wartości zmiennej globalnej *gray*. Zwraca nową wartość *display*, która w nieskończonej pętli jest przypisywana do zmiennej *display*.

Odliczanie BCD w górę

```
unsigned char bcd_up() {
    unsigned int display;
    if (bcd > 99) {
        bcd = 1;
    } else {
        display = ((bcd / 10) << 4) | (bcd % 10);
        bcd = bcd + 1;
    }
    return display;
}
```

Funkcja tworzy tymczasową zmienną wewnętrzną *display*. Przypisuje do niej przekonwertowaną na kod BCD wartość zmiennej globalnej *bcd*, następnie inkrementuje tą zmienną o jeden w górę. Na koniec zwraca wartość tymczasowej zmiennej wewnętrznej *display*, która następnie jest przypisywana w nieskończonej pętli do zmiennej *display*. Funkcja ta obsługuje sytuacje, w której maksymalna wartość możliwa do wyświetlenia w kodzie BCD (99) jest aktualnie odzwierciedlona na diodach. W takim przypadku odliczanie rozpocznie się ponownie od 1.

Odliczanie BCD w dół

```
unsigned char bcd_down() {
    unsigned int display;
    if (bcd == 0) {
        bcd = 99;
    } else {
        display = ((bcd / 10) << 4) | (bcd % 10);
        bcd = bcd - 1;
    }
    return display;
}
```

Funkcja analogicznie jak *bcd_up* konwertuje wartość globalnej zmiennej *bcd* do takiej, która jest możliwa do wyświetlenia na diodach, następnie dekrementuje wartość tej zmiennej. W przypadku, gdy aktualny stan zmiennej globalnej *bcd* jest równy 0, jest ona zmieniana na 99 i odliczanie zaczyna się od początku.

Snake (wąż)

```
unsigned char snake(unsigned char display){
    if (display < 7) {
        display = (display << 1) ^ 1;
    } else {
        if (direction == 1) {
            if (display == 224) {
                direction = 0;
                display = display >> 1;
            } else {
                display = display << 1;
            }
        } else {
            if (display == 7) {
                direction = 1;
                display = display << 1;
            } else {
                display = display >>1;
            }
        }
    }
    return display;
}
```

Funkcja przyjmuje jako argument aktualną wartość zmiennej *display*. Jest minimalnie bardziej rozbudowana niż odpowiednik, który mieliśmy zrobić, ponieważ trzyledowy wąż najpierw led po ledzie wysuwa się, a następnie odbija się od ścianki do ścianki. Do poprawnego działania funkcji użyłem zmiennej globalnej *direction*, dzięki której mogłem ustalać w którą stronę będą zachodzić przesunięcia bitowe. Funkcja zwraca nową wartość zmiennej *display*, która w nieskończonej pętli jest przypisywana do zmiennej *display*.

Queue (kolejka)

```
unsigned char queue() {
    if (sum == 128 || sum == 192 || sum == 224 || sum == 240 || sum == 248 || sum ==252 || sum == 254) {
        added = 1;
        sum = sum ^ added;
    } else if (sum == 255) {
        sum = 1;
        added = 1;
    } else {
        sum = sum ^ added;
        added = added << 1;
        sum = sum ^ added;
    }
    return sum;
}
```

Funkcja posługuje się dwoma zmiennymi globalnymi *added* i *sum*. W przypadku kiedy w kolejce pojawia się nowy bit, jest on przesuwany w lewą stronę, aż do momentu kiedy dotrze do „ściany” lub innego bitu, który jest blokowany przez bit lub „ścianę”. W przypadku zajęcia wszystkich bitów kolejka jest czyszczona i kolejkowanie ledów zaczyna się od początku. Wartość zmiennej *sum* zwracana na końcu funkcji jest przypisywana w pętli nieskończonej do zmiennej *display*.

PRNG

```
unsigned char prng(unsigned char display) {
    int ans, xored;
    xored = (((display >> 0) & 1) ^ ((display >> 1) & 1)) ^ (((display >> 4) & 1) ^ ((display >> 5) & 1));
    ans = (xored << 5) | (display >> 1);
    return ans;
}
```

Funkcja jako argument przyjmuje aktualną wartość zmiennej *display*, a następnie przy użyciu implementacji funkcji z pierwszych zajęć używa jej jako seed (ziarno) z którego jest tworzona kolejna pseudolosowa wartość. Jest ona zwracana na końcu funkcji, a następnie w nieskończonej pętli przypisywana do zmiennej *display*.

Zadanie 2

Obsługa:

- Optymalne ustawienie zegara: 8MHz
- P1: gdy wartość odczytywana przekroczy połowę zakresu uruchamia się alarm
- RB3: resetuje działanie alarmu

```
unsigned int isInformed = 0;

void alarm(){
    int i = 0;
    while (i < 10){
        if (PORTBbits.RB3 == 0) {
            PORTD = 0;
            i = 0;
        }
        if (((unsigned int)adc(0)/10) < 51){
            i = 10;
        }
        if (i%2 != 0){
            PORTD = 1;
        } else {
            PORTD = 0;
        }
        delay(500);
        i++;
    }
    isInformed = 1;
}
```

Do poprawnego działania sterownika została stworzona zmienna globalna *isInformed*, która przetrzymuje informację o tym, czy użytkownik został zaalarmowany, oraz funkcja alarm. Działanie funkcji opiera się na sprawdzaniu czy w trakcie trwania alarmu został wciśnięty przycisk RB3, jeżeli tak, alarm zaczyna ponowne odliczanie. Odliczanie polega na przejściu przez pętlę while, standardowo zawartość pętli wykona się 10 razy (dioda zapali się 5 razy i 5 razy zgaśnie), każde z 10 przejść pętli zakończy się półsekundową przerwą (co skutkuje łącznie 5 sekundową informacją przed uruchomieniem alarmu polegającego na zapaleniu wszystkich diód). Funkcja alarm została zaimplementowana w nieskończonej pętli w następujący sposób:

```
if (PORTBbits.RB3 == 0) {
    if (isInformed == 1){
        isInformed = 0;
    }
}
if (((unsigned int)adc(1)/10) >= 51){
    if (isInformed == 0){
        alarm();
    } else {
        PORTD = 255;
    }
} else {
    PORTD = 0;
    isInformed = 0;
}
```

Zadanie 3

Obsługa:

- Optymalne ustawienie zegara: 8MHz
- Reklama działa w nieskończonej pętli, nie obsługuje żadnych przycisków

W celu wykorzystania w reklamie niestandardowych znaków, korzystając z wymienionych na końcu materiałów dydaktycznych, stworzyłem funkcję, która zapisuje w pamięci wyświetlacza nowe, niestandardowe znaki.

```
void lcd_custom_char(unsigned char *Pattern, const char Location){
    unsigned int i;
    lcd_cmd(0x40+(Location*8));
    for (i=0; i<8; i++){
        lcd_dat(Pattern[i]);
    }
}
```

Następnie przy użyciu generatora (link do generatora również znajduje się w sekcji „Pomoce dydaktyczne”) stworzyłem potrzebne w mojej reklamie znaki i wpisałem je przy użyciu wcześniej napisanej funkcji do pamięci wyświetlacza.

```
lcd_init();
lcd_cmd(L_CLR);

unsigned char full_bottle[] = {0x4,0x4,0xe,0xe,0xe,0xe,0xe,0xe};
unsigned char half_bottle[] = {0x4,0x4,0xe,0xa,0xa,0xe,0xe,0xe};
unsigned char empty_bottle[] = {0x4,0x4,0xe,0xa,0xa,0xa,0xa,0xe};
unsigned char smile[] = {0x0,0xa,0xa,0xa,0x0,0x11,0xe,0x0};
unsigned char sad[] = {0x0,0xa,0xa,0xa,0x0,0xe,0x11,0x0};
unsigned char ciapki[] = {0x1b,0x1b,0x1b,0x0,0x0,0x0,0x0,0x0};

lcd_custom_char(full_bottle, 0);
lcd_custom_char(half_bottle, 1);
lcd_custom_char(empty_bottle, 2);
lcd_custom_char(smile, 3);
lcd_custom_char(sad, 4);
lcd_custom_char(ciapki, 5);
```

Ważne jest, aby wpisywanie znaków do pamięci odbywało się po inicjalizacji LCD przy użyciu funkcji lcd_init. W innym przypadku znaki mogą się nie wyświetlać, lub wyświetlać się w nieodpowiedni sposób (mimo, nie powinny się wyświetlać wcale, w trakcie testowania przerobiłem również taki scenariusz).

Treść reklamy jest długa, bo około 130 linii kodu. Będzie ona dostępna w pliku zadania z rozszerzeniem .c i możliwa do uruchomienia na symulatorze przy użyciu pliku z rozszerzeniem .hex na symulatorze.

Zadanie 4

Obsługa:

- Optymalne ustawienie zegara: 8MHz
- RB5: zmiana trybu działania pomiędzy 200W, 350W, 600W i 800W
- RB4: dodanie 1 minuty do zegara
- RB3: dodanie 10 sekund do zegara
- RB2: START / STOP zegara (bez restartowania czasu na zegarze)
- RB1: Reset zegara i mocy

Do poprawnego działania sterownika stworzyłem następujące funkcje i zmienne:

```
unsigned int power[2] = {2,0};
unsigned int timer[4] = {0,0,0,0};
```

Dwie zmienne globalne *power* i *timer* przechowujące aktualne wartości mocy i zegara wyświetlane następnie na ekranie.

```
void change_power(){
    if (power[0] == 2){
        power[0] = 3;
        power[1] = 5;
    } else if (power[0] == 3){
        power[0] = 6;
        power[1] = 0;
    } else if (power[0] == 6){
        power[0] = 8;
    } else {
        power[0] = 2;
    }
}
```

Funkcja *change_power*, która obsługuje zmianę aktualnej mocy mikrofalówki. Każde jednorazowe użycie przycisku odpowiadającego za wywołanie tej funkcji będzie skutkowało przejściu do kolejnej wartości. Po osiągnięciu 800W, następne wywołanie funkcji spowoduje przejście do 200W.

```
void timer_add_minute(){
    timer[1] = timer[1] + 1;
    if (timer[1] > 9) {
        timer[1] = 0;
        if (timer[0] < 6) {
            timer[0] = timer[0] + 1;
        } else {
            timer[0] = 6;
            timer[1] = 0;
            timer[2] = 0;
            timer[3] = 0;
        }
    }
}

  void timer_add_seconds(){
      if (timer[2] < 6){
          timer[2] = timer[2] + 1;
      } else {
          timer[2] = 0;
          timer_add_minute();
      }

  }
```

Funkcje *timer_add_minute* i *timer_add_seconds* powodują odpowiednio dodanie minuty lub 10 sekund do zegara kuchenki mikrofalowej. Funkcje te obsługują sytuacje, w których przekraczane są pełne minuty, lub pełne dziesiątki sekund i odpowiednio ustawiają stan zegara.

```
void decreese_timer(){
    if (timer[3] == 0){
        if (timer[2] == 0){
            if (timer[1] == 0){
                if (timer[0] > 0){
                    timer[0] = timer[0] - 1;
                    timer[1] = 9;
                    timer[2] = 5;
                    timer[3] = 9;
                }
            } else {
                timer[1] = timer[1] - 1;
                timer[2] = 5;
                timer[3] = 9;
            }
        } else {
            timer[2] = timer[2] - 1;
            timer[3] = 9;
        }
    } else {
        timer[3] = timer[3] - 1;
    }
}
```

Funkcja *decreese_timer* służy do odliczania w dół na zegarze.

```
char first_line[] = "MOC:        200W";
char second_line[] = "CZAS:      00:00";
```

Dwie zmienne globalne *first_line* i *second_line* zapisane jako lista charów przechowują aktualny stan obu linii wyświetlanych na ekranie. Zostały stworzone aby ułatwić mi pracę z wartościami wyświetlanymi na wyświetlaczu LCD.

```
void put_power_on_display(){
    first_line[12] = power[0] + '0';
    first_line[13] = power[1] + '0';
}

void put_time_on_display(){
    second_line[11] = timer[0] + '0';
    second_line[12] = timer[1] + '0';
    second_line[14] = timer[2] + '0';
    second_line[15] = timer[3] + '0';
}
```

Funkcje *put_power_on_display* i *put_time_on_display* służą do edycji wcześniej zadeklarowanych zmiennych globalnych *first_line* i *second_line*. Przy ich użyciu w odpowiednie miejsca wstawiam wartości odpowiadające aktualnemu stanowi mocy i czasu na zegarze.

```
void reset_display(){
    power[0] = 2;
    power[1] = 0;
    timer[0] = 0;
    timer[1] = 0;
    timer[2] = 0;
    timer[3] = 0;
    put_power_on_display();
    put_time_on_display();
}
```

Funkcja *reset_display* restartuje moc i czas na zegarze do wartości standardowych (początkowych) a następnie przy użyciu wcześniej opisanych funkcji *put_*_on_display* umieszcza je w tablicach charów wyświetlanych na ekranie LCD.

W funkcji *main* tworzę zmienną *working* przechowującą informację o tym, czy odliczanie jest rozpoczęte, czy nie, oraz inicjalizuję wyświetlacz wyświetlając na nim wartości początkowe. Wygląda to następująco:

```
unsigned int working = 0;

lcd_init();
lcd_cmd(L_CLR);
lcd_cmd(L_L1);
lcd_str(first_line);
lcd_cmd(L_L2);
lcd_str(second_line);
```

W nieskończonej pętli napisałem kod, dzięki któremu poprawnie działa obsługa przycisków funkcyjnych. Ich działanie zostało opisane w podpunkcie „Obsługa” zawartym na samym początku opisu tego zadania.

```
if (PORTBbits.RB5 == 0){
    change_power();
    put_power_on_display();
}
if (PORTBbits.RB4 == 0){
    timer_add_minute();
    put_time_on_display();
}
if (PORTBbits.RB3 == 0){
    timer_add_seconds();
    put_time_on_display();
}
if (PORTBbits.RB2 == 0){
    if (working == 1){
        working = 0;
    } else {
        working = 1;
    }
}
if (PORTBbits.RB1 == 0){
    reset_display();
}
```

Działanie mikrofalówki, oraz wyświetlanie komunikatu (po zakończeniu odliczania) informującego o tym, że posiłek jest gotowy zostało zaprogramowane w następujący sposób:

```
if (working == 1){
    if ((timer[0] == 0) && (timer[1] == 0) && (timer[2] == 0) && (timer[3] == 0)){
        lcd_cmd(L_CLR);
        lcd_cmd(L_L1);
        lcd_str("  POSILEK JEST  ");
        lcd_cmd(L_L2);
        lcd_str("     GOWOTY!    ");
        working = 0;
        delay(5000);
    } else {
        decreese_timer();
        put_time_on_display();
        delay(1000);
    }
} else {
    delay(250);
}

lcd_cmd(L_CLR);
lcd_cmd(L_L1);
lcd_str(first_line);
lcd_cmd(L_L2);
lcd_str(second_line);
```

Warto zwrócić uwagę, że wyświetlanie nowego stanu ekranu jest umieszczone na końcu każdego przejścia pętli while.

Zadanie 5

Obsługa:

- Optymalne ustawienie zegara: 8MHz
- P2: ustawienie czasu partii (nie działa w trakcie partii – tak jak powinno)
- RB3: zakończenie ruchu pierwszego gracza
- RB5: zakończenie ruchu drugiego gracza
- Rozpoczęcie partii przyciskiem RB3 powoduje rozpoczęcie tury gracza drugiego, analogicznie rozpoczęcie przyciskiem RB5 powoduje rozpoczęcie tury gracza pierwszego

Do poprawnego działania zegara do gry w szachy stworzyłem następujące zmienne globalne i funkcje:

```
char timer[] = "1:00        1:00";
unsigned int time1[] = {1,0,0};
unsigned int time2[] = {1,0,0};
```

Zmienna *timer*, podobnie jak zmienne *first_line* i *second_line* z Zadania 4 przechowuje ciąg znaków, który następnie będzie wyświetlony na ekranie. Zmienne *time1* i *time2* przechowują aktualny stan czasu na zegarze odpowiednio pierwszego i drugiego gracza.

```
void change_time(unsigned int value){
    if (value < 34){
        time1[0] = 1;
        time1[1] = 0;
        time1[2] = 0;
        time2[0] = 1;
        time2[1] = 0;
        time2[2] = 0;
    } else if ((value < 68) && (value >= 34)){
        time1[0] = 3;
        time1[1] = 0;
        time1[2] = 0;
        time2[0] = 3;
        time2[1] = 0;
        time2[2] = 0;
    } else {
        time1[0] = 5;
        time1[1] = 0;
        time1[2] = 0;
        time2[0] = 5;
        time2[1] = 0;
        time2[2] = 0;
    }
}
```

Funkcja *change_time* przyjmuje jako argument wartość odpowiadającą podanemu stanowi potencjometru P2, a następnie w zależności od tego w jakim zakresie wartości się ona mieści ustala czas na zegarach obu graczy na 1, 3 lub 5 minut.

```
void put_time_on_display(){
    timer[0] = time1[0] + '0';
    timer[2] = time1[1] + '0';
    timer[3] = time1[2] + '0';
    timer[12] = time2[0] + '0';
    timer[14] = time2[1] + '0';
    timer[15] = time2[2] + '0';
}
```

Funkcja *put_time_on_display* jest odpowiednikiem funkcji *put_*_on_display* z Zadania 4, dostosowanym do wymogów aktualnego zadania. Przenosi wartości z tablic przechowujących aktualny czas obu graczy do ciągu znaków, który później jest wyświetlany na ekranie LCD.

```
void decreese_timer1(){
    if (time1[2] == 0){
        if (time1[1] == 0){
            if (time1[0] > 0){
                time1[0] = time1[0] - 1;
                time1[1] = 5;
                time1[2] = 9;
            }
        } else {
            time1[1] = time1[1] - 1;
            time1[2] = 9;
        }
    } else {
        time1[2] = time1[2] - 1;
    }
}

void decreese_timer2(){
    if (time2[2] == 0){
        if (time2[1] == 0){
            if (time2[0] > 0){
                time2[0] = time2[0] - 1;
                time2[1] = 5;
                time2[2] = 9;
            }
        } else {
            time2[1] = time2[1] - 1;
            time2[2] = 9;
        }
    } else {
        time2[2] = time2[2] - 1;
    }
}
```

Funkcje *decreese_timer1* i *decreese_timer2* są odpowiedzialne za obsługę upływu czasu na zegarach graczy.

```
unsigned int turn = 1;
void change_turn(){
    if (PORTBbits.RB3 == 0){
        turn = 2;
    }
    if (PORTBbits.RB5 == 0) {
        turn = 1;
    }
}
```

Zmienna globalna *turn* przechowuje informację o tym, który gracz aktualnie wykonuje ruch, a więc również informację o tym, z którego zegara odejmujemy czas. Funkcja *change_turn* odpowiada za zmianę osoby wykonującej ruch, została ona wydzielona ponieważ jest wielokrotnie wykonywana w nieskończonej pętli.

W funkcji main został zainicjalizowany ekran oraz zostały na niego naniesione podstawowe wartości. Została również zadeklarowana zmienna *running*, która przechowuje informację o tym, czy partia trwa.

```
lcd_init();
lcd_cmd(L_CLR);
lcd_cmd(L_L1);
lcd_str("GRACZ 1  GRACZ 2");
lcd_cmd(L_L2);
lcd_str(timer);

unsigned int running = 0;
```

W nieskończonej pętli obsłużona jest zmiana czasu trwania partii. Warto zauważyć, że czas trwania partii może zostać zmieniony jedynie przed jej rozpoczęciem, dzięki czemu zapobiegamy nieoczekiwanym zachowaniom zegara. W przypadku upłynięcia całego czasu na zegarze któregoś z graczy zostanie wyświetlony odpowiedni komunikat.

```
if (running == 0){
    change_time(((unsigned int)adc(1)/10));
    put_time_on_display();
    if (PORTBbits.RB3 == 0){
        running = 1;
        turn = 2;
    }
    if (PORTBbits.RB5 == 0) {
        running = 1;
        turn = 1;
    }
    delay(200);
    lcd_cmd(L_CLR);
    lcd_cmd(L_L1);
    lcd_str("GRACZ 1  GRACZ 2");
    lcd_cmd(L_L2);
    lcd_str(timer);
} else {
    change_turn();
    if (turn == 1){
        if ((time1[0] == 0) && (time1[1] == 0) && (time1[2] == 0)){
            lcd_cmd(L_CLR);
            lcd_cmd(L_L1);
            lcd_str("GRACZ 1 PRZEGRAL");
            lcd_cmd(L_L2);
            lcd_str("   PRZEZ CZAS   ");
            delay(5000);
            running = 0;
            change_time(((unsigned int)adc(1)/10));
        } else {
            decreese_timer1();
        }
    } else if (turn == 2) {
        if ((time2[0] == 0) && (time2[1] == 0) && (time2[2] == 0)){
            lcd_cmd(L_CLR);
            lcd_cmd(L_L1);
            lcd_str("GRACZ 2 PRZEGRAL");
            lcd_cmd(L_L2);
            lcd_str("   PRZEZ CZAS   ");
            delay(5000);
            running = 0;
            change_time(((unsigned int)adc(1)/10));
        } else {
            decreese_timer2();
        }
    }
```

```
    put_time_on_display();
    lcd_cmd(L_CLR);
    lcd_cmd(L_L1);
    lcd_str("GRACZ 1  GRACZ 2");
    lcd_cmd(L_L2);
    lcd_str(timer);
    delay(125);
    change_turn();
    delay(125);
    change_turn();
    delay(125);
    change_turn();
    delay(125);
    change_turn();
    delay(125);
    change_turn();
    delay(125);
    change_turn();
    delay(125);
    change_turn();
    delay(125);
    change_turn();
}
```

Z racji tego, że każda sekunda w takich rozgrywkach ma znaczenie postanowiłem obsłużyć odczytywanie przycisku zmiany tury w „dziwny” ale spełniający swoje założenia sposób. Jedna sekunda czasu oczekiwania pomiędzy odjęciem tej sekundy od zegara została podzielona na 8 segmentów trwających 125 ms. podczas których sprawdzane jest czy przycisk zmiany kolejki został wciśnięty. Takie rozwiązanie powoduje niemalże natychmiastowe wychwycenie wciśniętego przycisku.

Pomoce dydaktyczne

1. Generator niestandardowych znaków: quinapalus.com
2. Poradnik umieszczania znaków na LCD: openlabpro.com

Uwagi autora

Na etapie tworzenia sprawozdania i rozmowy z moim przyjacielem **Maciejem Szymborskim** (również studentem tego samego roku ISI GR1) dostrzegłem kilka poprawek oraz rozwiązań, które mogłyby pozytywnie wpłynąć np. na ilość kodu. Poniżej zwrócę uwagę na kilka z nich.

Zadanie 4 i Zadanie 5: czas, który aktualnie jest przechowywany w tablicach gdzie odpowiednie indeksy odpowiadają minutom, dziesiątkom sekund i jednościom sekund można by przechowywać jako jedna wartość odpowiadająca sumarycznej ilości sekund, a następnie przy użyciu wartości floor z dzielenia, modulo 60 i modulo 10 rozbijana na minuty, dziesiątki sekund i sekundy. To rozwiązanie wydaje się dużo prostsze i zajmujące mniej linii kodu.

Zadanie 5: funkcje *decreese_timer1* i *decreese_timer2* można zastąpić jedną funkcją, która w argumencie przyjmuje wskaźnik na odpowiednią tablicę czasów (w przypadku aktualnego rozwiązania, lub na zmienną przechowującą czas w sekundach w przypadku zmian z poprzedniego podpunktu uwag).