Marcin Głód 164004

Systemy Wbudowane - Sprawozdanie

Spis Treści

Cel Ćwiczeń

- Zapoznanie z działaniem mikrokontrolera
- Poznanie procesu prototypowania systemów opartych o mikrokontrolery
- Zaznajomienie z procesem debugowania oprogramowania mikrokontrolerów

- Podstawy tworzenia oprogramowania w oparciu o płytki rozwojowe

Opis Środowiska Pracy

MPLAB X IDE

Do programowania i debugowania mikrokontrolerów używałem środowiska MPLAB X IDE, głównie ze względu na wbudowane narzędzie do debugowania kodu i programowania podłączonego mikrokontrolera. Pomimo iż jest to narzędzie dedykowane dla systemów wbudowanych, niestety wymaga użycia dodatkowych narzędzi, by pracować z nim efektywnie. Celem uzupełnienia środowiska deweloperskiego do pisania kodu użyłem również edytora neovim uzupełnionego o clangd LSP oraz systemu kontroli wersji git, aby poprawić komfort pracy i efektywność programowania.

Explorer 16/32

Płytka rozwojowa Explorer 16/32 jest wszechstronnym narzędziem do prototypowania systemów opartych na mikrokontrolerach. Oferuje szereg modułów i funkcji, dzięki którym rozwój oprogramowania jest procesem szybszym i wygodniejszym. Głównymi wyróżnikami opisywanej płytki są:

- **Funkcje i przeznaczenie:** Umożliwia szybkie testowanie i rozwój oprogramowania dla różnych mikrokontrolerów produkowanych przez Microchip.
- **Zastosowanie:** Używana do edukacji, projektów badawczych i rozwoju prototypów.
- **Moduły wykorzystane na zajęciach:** Płytka Explorer 16/32 zawiera wbudowane moduły takie jak zasilacz, konwerter UART/I2C na USB(dzięki któremu mogliśmy komunikować się z komputerem), diody, przyciski, potencjometr, oraz ekran LCD (alfanumeryczny 16x2 znaki).
- **pozostałe Moduły:** Wśród modułów niewykorzystanych podczas ćwiczeń znaleźć możemy: piny GPIO, czujnik temperatury, złącze portu Serial, oraz JTAG.
- **Kompatybilne procesory:** Obsługuje szeroką gamę mikrokontrolerów z rodzin PIC24, dsPIC33, oraz PIC32.

Procesor PIC24FJ128GA010

Mikrokontroler PIC24FJ128GA010 charakteryzuje się następującymi parametrami:

- **Taktowanie:** Maksymalna częstotliwość taktowania wynosi 32 MHz.
- **RAM:** 16 KB pamięci RAM.
- **Parametry:** Oferuje 128 KB pamięci Flash, 21 kanałów ADC, 5 Timerów 16-bitowych oraz 3 interfejsy UART.

- **Zastosowanie IRL (In Real Life):** Znajduje zastosowanie w aplikacjach wymagających średniej mocy obliczeniowej, takich jak kontrolery przemysłowe, urządzenia medyczne i systemy wbudowane w automatyce domowej.

Opis Zrealizowanych Zadań

Liczniki Binarne

Program przygotowany w tym zadaniu zajmuje się wyświetlaniem kolejnych liczb przy użyciu diod dostępnych na płytce. Naciśnięcie przycisków S3 oraz S4 powoduje zmianę systemu binarnego w jakim wyświetlane są liczby. Dostępnymi trybami są:

- licznik binarny zliczający w górę
- licznik binarny zliczający w dół
- licznik w kodzie Graya zliczający w górę
- licznik w kodzie Graya zliczający w dół
- dwucyfrowy licznik BCD zliczający w górę
- dwucyfrowy licznik BCD zliczający w dół

Poniżej możemy się przyjrzeć implementacji zastosowanych funkcji zliczających: **licznik binarny:**

```
void zlicz_gora(void) {
        static int zliczg = 0; // zliczanie zaczynamy od 0
        zliczg++; // przy każdym przejściu pętli zwiększamy liczbę o 1
        if (zliczg>255){ // jeżeli osiągniemy 255, zmieniamy wartość z powrotem na 0
            zliczg = 0;
        }
        LATA = zliczg; // do Latch A przypisujemy wartość zmiennej zliczającej
        sleep(TIME); // czekamy moment, by kolejne liczby były zauważalne
}
```

licznik Graya:

```
void zlicz_gray_gora(void) {
        static int zliczg = 0; // zliczanie zaczynamy od 0
        zliczg++; // przy każdym przejściu pętli zwiększamy liczbę o 1
        if (zliczg>255){ // jeżeli osiągniemy 255, zmieniamy wartość z powrotem na 0
            zliczg = 0;
        }
        LATA = zliczg ^ (zliczg >> 1); // wartość zmiennej zliczającej jest poddawar
        sleep(TIME); // czekamy moment, by kolejne liczby były zauważalne
}
```

Do przedstawienia kodu BCD musimy wpierw utworzyć odpowiednią strukturę:

```
struct BCD_Counter {
    unsigned int cyfra1 : 4; // Pierwsza cyfra BCD
    unsigned int cyfra10 : 4; // Druga cyfra BCD
};
```

kod BCD

```
void inkrementujBCD(struct BCD_Counter *licznik) {
    if (licznik->cyfra1 < 9) { // jeżeli cyfra jednostek nie jest 9, to podnosimy ja
        licznik->cyfra1++;
    } else { // jeżeli cyfra jedności jest 9, to jedności przestawiamy na 0, a cyfrę
        licznik->cyfra1 = 0;
        if (licznik->cyfra10 < 9) {
            licznik->cyfra10++;
        } else {
            licznik->cyfra10 = 0;
        }
    }
    LATA = (licznik->cyfra10 << 4) | licznik->cyfra1; // do wyświetlania, cyfrę dzie
    __delay32(2000000); // czekamy
```

główna pętla

```
int main(void) {
unsigned char portValue = 0x53;
struct BCD_Counter licznik = {0, 0}; // zerujemy strukturę, z której korzysta BCD
AD1PCFG = 0xFFFF;
TRISA = 0x0000;
while(1){
    if (BUTTON_IsPressed (BUTTON_S3) == true) { // gdy naciśnięty jest przycisk S3,
        shift = shift + 1;
        if (shift > 5){
            shift = 0;
```

```
            }
        __delay32(1000);
        } else if ((BUTTON_IsPressed (BUTTON_S4) == true)) { // gdy naciśnięty jest przy
            shift = shift - 1;
            if (shift < 0){
                shift = 5;
            }
        __delay32(1000); // czekamy chwilę, by uniknąć efektu debounce'u przycisków
        }
        if (shift == 0){ // w zależności od wartości shift wybieramy tryb wyświetlania
            zlicz_gora();
        } else if (shift == 1){
            zlicz_dol();
        } else if (shift == 2){
            zlicz_gray_gora();
        } else if (shift == 3){
            zlicz_gray_dol();
        } else if (shift == 4){
            inkrementujBCD(&licznik);
        } else if (shift == 5){
            dekrementujBCD(&licznik);
        } else{
            __delay32(100);
        }
    }
    return 0; // zamykamy main
    }
```

Potencjometr

Ten program implementuje system alarmowy, który monitoruje wartość odczytaną z potencjometru. Alarm włącza się, gdy odczytana wartość przekroczy określony próg i jest wyłączany przyciskiem S3.

W pętli odczytujemy wartość potencjometru, przesuwając jego wartość o dwa bity w prawo, co zapewnia, że wartość jaką uzyskamy jest w przedziale (0-255). Po przekroczeniu wartości 128 włącza się alarm, który na początku mruga pięciokrotnie, następnie świeci ciągle. Alarm przerywa zejście poniżej 128 na potencjometrze, lub naciśnięcie przycisku S3.

Główną pętlę programu stworzyłem w następujący sposób:

```
int main(void) {
    unsigned char portValue = 0x53;
    int alarm = 0;

    ADC_SetConfiguration(ADC_CONFIGURATION_DEFAULT);
    ADC_ChannelEnable(ADC_CHANNEL_POTENTIOMETER);
```

```
TRISA = 0x0000;

unsigned int value;

while (1) {
    // Odczyt wartości potencjometru
    value = ADC_Read10bit(ADC_CHANNEL_POTENTIOMETER);

    // Sprawdzenie poprawności odczytu
    if (value == 0xFFFF) {
        continue;
    }

    // Normalizacja wartości do 8 bitów
    unsigned char normValue = value >> 2;

    // Sprawdzenie progu alarmowego
    if (normValue >= 128) {
        alarm = 1;
    }

    // Obsługa alarmu
    while (alarm == 1) {
        // Sprawdzenie przycisku do wyłączenia alarmu
        if (BUTTON_IsPressed(BUTTON_S3) == true) {
            alarm = 0;
            break;
        }
        value = ADC_Read10bit(ADC_CHANNEL_POTENTIOMETER);
        normValue = value >> 2;
        if (normValue < 128){
            alarm = 0;
            break;
        }

        // Miganie diodą LED
        for (int i = 0; i < 5; i++) {
            LATA = 1;
            sleep(100);
            LATA = 0;
            sleep(100);
        }

        // Włączenie wszystkich diod
        LATA = 255;
        // Wszystkie diody świecą aż wyłączymy alarm
        while(alarm == 1){
            if(BUTTON_IsPressed(BUTTON_S3) == true){
                alarm = 0;
            }
            value = ADC_Read10bit(ADC_CHANNEL_POTENTIOMETER);
```

```
                normValue = value >> 2;
                if (normValue < 128){
                    alarm = 0;
                    break;
                }
            }
            // Wyświetlenie wartości na porcie Latch A
            LATA = normValue;
        }
    }
    return 0;
}
```

Kod tego zadania na pewno dałoby się zoptymalizować przez podział funkcji main() na mniejsze komponenty, jednak zastosowane podejście było prostsze i bardziej przejrzyste dla oczu nie wprawionych w tworzenie oprogramowania w C.

Mikrofala

Program ten jest symulatorem mikrofalówki. Składa się z prostego timera wyświetlającego na ekranie LCD pozostały czas i wybraną moc mikrofali. Sterowanie odbywa się za pomocą kilku przycisków.

Kod tego zadania wyszedł w sposób następujący:

funkcje opóźnienia - w sumie nie robią nic, poza czekaniem. Oddzielne nazwy funkcji dla różnych czasów były po prostu łatwiejsze w użyciu.

```
void __delay_us(unsigned long us) { __delay32(us * FCY / 1000000); }
void __delay_ms(unsigned long us) { __delay32(us * FCY / 1000); }
```

obsługa wyświetlacza LCD Z racji iż są one zapożyczone ze skryptu dostarczonego na zajęciach, pominę opis następujących funkcji:

```
LCD_sendCommand()

LCD_sendData()

LCD_print()

LCD_setCursor()

LCD_init()

LCD_clear()
```

wyświetlanie wybranych wartości

```
void wyswietlaj(unsigned int time_left, unsigned char current_power) {
  char bufor_wyswietlacza[16];
  unsigned char sec = time_left % 60;
  unsigned char min = time_left / 60;
  sprintf(bufor_wyswietlacza, "%02u:%02u MOC: %u", min, sec, current_power);
  LCD_clear();
  LCD_print((unsigned char *)bufor_wyswietlacza);
}
```

obsługa przycisków

```
void buttonierka(unsigned int *time_left, unsigned char *current_power, unsigned cha
  // Sprawdź, czy przycisk S3 jest wciśnięty i aktualna moc jest mniejsza niż 5
  if (BUTTON_IsPressed(BUTTON_S3) && *current_power < 5) {
    (*current_power)++; // Zwiększ moc o 1
    while (BUTTON_IsPressed(BUTTON_S3)); // Poczekaj, aż przycisk S3 zostanie zwolni
  }

  // Sprawdź, czy przycisk S4 jest wciśnięty, licznik nie jest aktywny i czas pozost
  if (BUTTON_IsPressed(BUTTON_S4) && !*aktywny && *time_left < 5999) {
    *time_left += 30; // Dodaj 30 sekund do pozostałego czasu
    while (BUTTON_IsPressed(BUTTON_S4)); // Poczekaj, aż przycisk S4 zostanie zwolni
  }

  // Sprawdź, czy przycisk S6 jest wciśnięty
  if (BUTTON_IsPressed(BUTTON_S6)) {
    *aktywny = !*aktywny; // Zmień stan aktywności licznika na przeciwny
    while (BUTTON_IsPressed(BUTTON_S6)); // Poczekaj, aż przycisk S6 zostanie zwolni
  }

  // Sprawdź, czy przycisk S5 jest wciśnięty
  if (BUTTON_IsPressed(BUTTON_S5)) {
    *time_left = 0; // Ustaw pozostały czas na 0
    *aktywny = 0; // Dezaktywuj licznik
    while (BUTTON_IsPressed(BUTTON_S5)); // Poczekaj, aż przycisk S5 zostanie zwolni
  }
}
```

Zegar Szachowy

W tym zadaniu utworzyłem zegar szachowy, który na ekranie LCD pokazuje ile czasu pozostało każdemu z graczy. Dodatkowo, zegar posiada dodaktowy przycisk, którego naciśnięcie powoduje zakończenie gry niezależnie od pozostałego czasu.

Oto opis kodu źródłowego:

wyświetlanie danych na LCD

```
// Funkcja wyświetlająca czas na LCD
void displayTime(unsigned int time, unsigned char row, unsigned char col) {
    char timeStr[6];
    unsigned int minutes = time / 60;
    unsigned int seconds = time % 60;
    intToStr(minutes, timeStr, 2); // Konwertowanie minut na tekst
    timeStr[2] = ':'; // Dodanie dwukropka
    intToStr(seconds, timeStr + 3, 2); // Konwertowanie sekund na tekst
    LCD_setCursor(row, col); // Ustawienie kursora na LCD
    LCD_print((unsigned char*)timeStr); // Drukowanie czasu na LCD
}
```

obsługa przycisków

```
void obsluga_przyciskow(unsigned char* currentPlayer, unsigned char* gameActive) {
    if (BUTTON_IsPressed(BUTTON_S4)) {
        __delay_ms(300); // Opóźnienie debouncingu
        while (BUTTON_IsPressed(BUTTON_S4)); // Czekanie na zwolnienie przycisku
        *gameActive = !(*gameActive); // Przełączenie stanu aktywności gry
        if (!(*gameActive)) {
            LCD_sendCommand(LCD_CLEAR); // Wyczyść LCD
            LCD_setCursor(1, 0); // Ustawienie kursora na początku
            LCD_print((unsigned char*)"Game Stopped"); // Wyświetlenie komunikatu
        }
    }

    if (*gameActive && BUTTON_IsPressed(BUTTON_S3)) {
        __delay_ms(300); // Opóźnienie debouncingu
        while (BUTTON_IsPressed(BUTTON_S3)); // Czekanie na zwolnienie przycisku
        *currentPlayer = 3 - *currentPlayer; // Przełączenie gracza (z 1 na 2 lub z
    }
}
```

logika zegara

```
void chessClock() {
    unsigned int time1 = 300; // 5 minut dla gracza 1
    unsigned int time2 = 300; // 5 minut dla gracza 2
    unsigned char currentPlayer = 1;
    unsigned char gameActive = 0;

    LCD_setCursor(1, 0);
    LCD_print((unsigned char*)"P1:");
    displayTime(time1, 1, 3); // Wyświetlenie czasu dla gracza 1
```

```
    LCD_setCursor(2, 0);
    LCD_print((unsigned char*)"P2:");
    displayTime(time2, 2, 3); // Wyświetlenie czasu dla gracza 2

    while (1) {
        obsluga_przyciskow(&currentPlayer, &gameActive); // Obsługa przycisków

        if (gameActive) {
            if (currentPlayer == 1) {
                if (time1 == 0) {
                    LCD_sendCommand(LCD_CLEAR); // Wyczyść LCD
                    LCD_setCursor(1, 0);
                    LCD_print((unsigned char*)"Player 1 loses"); // Wyświetlenie kom
                    break;
                }
                time1--;
                displayTime(time1, 1, 3); // Wyświetlenie zaktualizowanego czasu dla
            } else {
                if (time2 == 0) {
                    LCD_sendCommand(LCD_CLEAR); // Wyczyść LCD
                    LCD_setCursor(1, 0);
                    LCD_print((unsigned char*)"Player 2 loses"); // Wyświetlenie kom
                    break;
                }
                time2--;
                displayTime(time2, 2, 3); // Wyświetlenie zaktualizowanego czasu dla
            }
        }

        __delay_ms(1000); // Opóźnienie 1 sekundy
    }
}
```

główna pętla programu

```
int main(void) {
    TRISB = 0x7FFF;
    TRISD = 0x0000; // ustawienie wartości początkowych
    TRISE = 0x0000;

    LCD_init();

    while (1) { // logika zegara szachowego w nieskończonej pętli
        chessClock();
    }

    return 0;
}
```

Problemy i Trudności

Oto główne trudności z którymi spotkałem się podczas rozwiązywania zadań w ramach przedmiotu Systemy Wbudowane:

- **Problemy z MPLAB X IDE na Linuxie:** Środowisko działa niestabilnie i często sprawia problemy podczas pracy na systemie Linux (oficjalna dokumentacja zatrzymuje się na Ubuntu 14.04 (End of Life od 04.2019), oraz na Fedorze 20 (End of Life 06.2015)).
- **Instalacja:** Instalacja za pomocą pliku .sh o wadze 900MB jest nieefektywna i stwarza problemy związane z bezpieczeństwem.
- **Niszowa, droga i niedostępna platforma:** Platforma jest mniej dostępna i droższa w porównaniu do bardziej popularnych i tańszych alternatyw jak ATmega328, ESP32 czy PIC16F877A, które są łatwiej dostępne dla studentów do użytku domowego i hobbystycznego.

Źródła

- Dokumentacja od Microchip
- Materiały udostępnione w ramach zajęć
- kod Demo z oficjalnej strony Microchip
- Kanał YouTube BinderTronics
- learnxinyminutes.com/docs/c/
- Microcontroller Embedded C Programming Absolute Beginners Tutorial