Instrukcja do laboratorium Akademii ETI 26.03.2014*

I. Logowanie do systemu

Aby zalogować się do komputera należy podać następującego użytkownika i hasło:

- w sali 308: ***lab1/lab1***
- w sali 325: ***auslab/auslab***

Po zalogowaniu użytkownicy mają dostęp do następujących zasobów:

- w sali 308: ***C:\Designs***
- w sali 325: ***D:\auslab***

II. Uruchomienie środowiska programistycznego

Jako środowisko programistyczne wykorzystany jest pakiet WIN AVR wraz ze zintegrowanym edytorem tekstowym. W celu jego uruchomienia należy kliknąć na plik:

- w sali 308: ***C:\Designs\avr\pn.bat***
- w sali 325: ***D:\auslab\avr\pn.bat***

Po kliknięciu uruchomi się program „Programmer's Notepad", który stanowi mini środowisko IDE. Konfiguracja środowiska **wymaga umieszczania katalogów projektu:**

- dla sali 308 w katalogu: ***C:\Designs\avr\"katalog projektu"***
- dla sali 325 w katalogu: ***D:\auslab\avr\"katalog projektu"***

III. Utworzenie przykładowego projektu i wykonanie kompilacji

Aby stworzyć pierwszy prosty program i go skompilować należy kolejno:

1) Utworzyć (spod systemu) katalog w którym chcemy zapisać projekt np: ***C:\Designs\avr\kat_proj***, Położenie folderu musi koniecznie być zgodne z pkt. II powyżej.

2) W programie „Programmer's Notepad" należy wybrać kolejno: ***File/New/Project*** a następnie należy podać nazwę projektu np.:"***proj_pierwszy***" a jako folder utworzony w pkt. powyżej.

3) W kolejnym kroku należy utworzyć plik programu, polecenia: ***File/New/C++***. Pojawi się okno w którym powinniśmy wpisać kod programu. Poniżej w przykładzie jest minimalny kod programu:

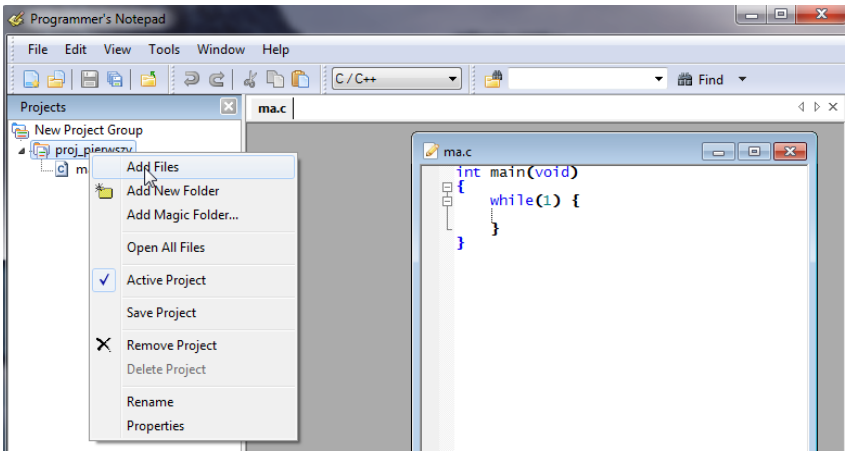
```
int main(void)
{
 while(1) {

 }
}
```

---

* Sponsorzy:

Utworzony plik należy następnie zapisać z rozszerzeniem „.c” (klawisze Ctrl-S) w katalogu utworzonym w pkt 2). W następnym kroku dodajemy plik do projektu poprzez najechanie prawym klawiszem myszki nad nazwę projektu i wybranie utworzonego pliku poleceniem ***Add Files*** jak na rysunku poniżej.



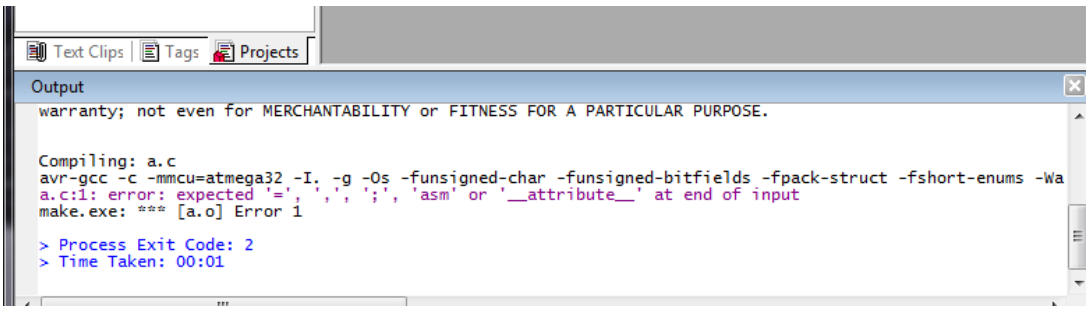
4) Do kompilacji niezbędny jest plik ***makefile***, którego reguły składni wykraczają poza materiał tych zajęć. W katalogu ***/avr/test_project*** przygotowany jest stosowny plik ***makefile*** i należy go skopiować do własnego katalogu projektu.

Kompilację i wgranie na płytkę można wykonać poprzez wydanie poleceń:

Tools/[WinAVR] Make All oraz

Tools/[WinAVR] Program

Oczywiście aby próbować wgrać kod na płytkę należy najpierw skompilować program bez błędów, które jeśli się pojawią będą wyszczególnione w oknie ***Output***.



Zadanie 1. Należy zapalić 3 wybrane przez siebie diody LED.

Przejdziemy teraz do nauki obsługi portów mikrokontrolera. Pierwszym krokiem jest zapalanie wybranych przez nas diod. W tym celu należy skorzystać z poniższego kodu.

Wersja dla posiadaczy płytki typ I:

```
#define F_CPU 16000000L
#include <avr/io.h>

int main(void){
     DDRB|=(1<<PB7)|(1<<PB5)|(1<<PB3)|(1<<PB1);
     PORTB|=(1<<PB7)|(1<<PB5)|(1<<PB3)|(1<<PB1);
```

```
    while(1){

    }

}
```

Wersja dla posiadaczy płytki typ II:

```
#define F_CPU 8000000L
#include <avr/io.h>

int main(void){
    DDRC|=(1<<PC2)|(1<<PC4);
    while(1){

    }

}
```

Jak nie trudno zauważyć efektem działania programu będzie zaświecenie diod podłączonych do odpowiednich pinów wybranego przez nas portu mikrokontrolera.

Program rozpoczynamy od zdefiniowania z jaką częstotliwością pracuje nasz układ. Kolejnym krokiem jest dołączenie odpowiedniego pliku nagłówkowego, który pozwoli nam w łatwy sposób odwoływać się do rejestrów mikrokontrolera. Następnie zgodnie z informacjami znajdującymi się w dokumentacji technicznej naszego układu dotyczącymi konfiguracji GPIO ustawiamy stan wysoki wybranych bitów w rejestrze DDRX mikrokontrolera gdzie X odpowiada oznaczeniu portu do którego podłączone są diody, co konfiguruje odpowiadające ustawianym bitom, piny układu jako wyjścia.

Dla układu typu I:

Kolejna linia kodu ustawia stan wysoki na wybranych przez nas wyjściach mikrokontrolera co skutkuje pojawieniem się na nich napięcia zasilania. Efektem poprawnie przeprowadzonego programowania układu powinny być cztery święcące diody.

Dla układu typu II:

Po ustawieniu wybranych pinów jako wyjściowe pojawia się na nich automatycznie stan niski który powoduję świecenie diod.

Zadanie 2. Należy spowodować cykliczne zapalanie się i gaszenie diod nr 1 i 2 co 500 ms.

Jak szybko zauważycie działanie naszego programu nie jest zbyt efektowne dlatego postaramy się tchnąć w niego odrobinę życia. Z pomocą przychodzi nam funkcja **_delay_ms(X);** gdzie X oznacza liczbę milisekund na jaką nasz układ ma zatrzymać swoją pracę. Aby móc wykorzystać wymieniona wyżej funkcję należy do programu dołączyć do programu odpowiedni plik nagłówkowy w następujący sposób:

#include <util/delay.h>

Przykład działania funkcji **_delay_ms();**

Dla układu typu I:

Układ pierwszy ma do dyspozycji 8 diod dołączonych do portu B mikrokontrolera.

```
PORTB &= ~(1<<PB6);
PORTB|=(1<<PB7);
_delay_ms(500);

PORTB &= ~(1<<PB7);
PORTB|=(1<<PB6);
_delay_ms(500);
```

Dla układu typu II:
Układ drugi ma do dyspozycji 4 diody dołączone do pinów PC2, PC3, PC4 i PC5.

```
PORTC &= ~(1<<PC2);
PORTC|=(1<<PC3);
_delay_ms(500);

PORTC &= ~(1<<PC3);
PORTC|=(1<<PC2);
_delay_ms(500);
```

Efektem działania programu będzie naprzemienne świecenie i gaśnięcie dwóch diod. Spróbujcie teraz rozbudować program tak aby uzyskać ciekawą sekwencje odpowiednio manipulując dostępnymi diodami.

Zadanie 3. Należy zapalać diodę po przyciśnięciu przycisku.

Kolejnym krokiem jest dodanie możliwości interakcji układu z użytkownikiem czyli obsługi przycisku. Dzięki automatycznej konfiguracji portów jako wejścia nie musimy zmieniać zawartości rejestru DDR.

Dla układu typu I:
Układ pierwszy ma do dyspozycji 8 przycisków dołączonych do portu D mikrokontrolera.
Przykład odczytu stanu przycisku z wykorzystaniem instrukcji warunkowej:

```
if((PIND & (1<<PD0)))
     //instrukcje po wykonywane po wykryciu wciśnięcia
przycisku
else
     //instrukcje wykonywane kiedy przycisk nie jest wciśnięty
```

Dla układu typu II:
Układ drugi ma do dyspozycji 4 przyciski dołączone do pinów PC6, PC7, PD2 i PD6.

```
if(!(PINC & (1<<PC7)))
     //instrukcje wykonywane po wykryciu wciśnięcia przycisku
else
     //instrukcje wykonywane kiedy przycisk nie jest wciśnięty
```

Znając sposób odczytu stanu przycisku oraz potrafiąc zapalać i gasić diody możemy napisać program, który będzie zapalał diodę odpowiadającą wciśniętemu przyciskowi.

Zadanie 4. Należy zapalać liczbę diod proporcjonalną do wartości odczytanego z potencjometru napięcia.

Ostatnim zadaniem będzie wykorzystanie wbudowanego w układ przetwornika ADC który pozwoli nam zmierzyć napięcie wyjściowe z potencjometru. Do tego celu wykorzystamy dwie funkcje,

Funkcja inicjalizująca przetwornik która należy wywołać jednokrotnie na początku kodu programu przed wykonaniem pomiaru:

```
void adc_init(){

     ADMUX |= (1 << REFS0);
     ADCSRA |= (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1) | (1
<< ADPS0);
}
```

Funkcja pomiar która zwraca wynik pomiaru:

```
uint16_t pomiar(uint8_t kanal) {
     ADMUX |= (ADMUX & 0xF8) | kanal;
     ADCSRA |= (1 << ADSC);
     while ( ADCSRA & (1 << ADSC))
          ;
     return ADCW;
}
```

Argumentem przekazywanym do funkcji jest liczba z zakresu od 0 do 7, która decyduje o wyborze kanału na którym będzie wykonywany pomiar. W praktyce jeśli wywołamy funkcje z argumentem równym 1 to przetwornik będzie wykonywał pomiar napięcia znajdującego się na pinie pierwszym portu A, ponieważ na tym porcie wyprowadzone są wejścia przetwornika. W przypadku układu typu I wybieramy kanał 7, ponieważ potencjometr oznaczony jako POT1 jest na stałe dołączony do kanału 7, w przypadku układu drugiego wybieramy kanał 0. Zwracana przez funkcje wartość zawiera się w zakresie od 0 do 1023, gdzie 0 to napięcie równe masie a 1023 to napięcie równe napięciu zasilania.
Waszym zadaniem będzie napisanie programu który zapala coraz większą ilość diod wraz ze wzrostem mierzonego napięcia.

Materiał przygotowali:
Bogdan Pankiewicz
Dariusz Wika
Gdańsk 26.03.2014