Programowanie mikrokontrolerów

w języku C

Marcin Engel Marcin Peczarski

Instytut Informatyki Uniwersytetu Warszawskiego

27 listopada 2012

Elementarz

- Jak zwykle, trzeba włączyć pliki nagłówkowe.
- Najpotrzebniejsze definicje są w

  `#include <avr/io.h>`
- Funkcja `main` jest bezargumentowa i nie kończy działania.
- Najprostsza funkcja `main`:

```
int main(void) {
  for (;;);
}
```

Manipulowanie bitami w rejestrach wejścia-wyjścia

- Skonfigurowanie wyprowadzenia PA0 jako wyjście:

```
DDRA |= 1 << PA0;
```

- Ustawienie stanu wysokiego na wyprowadzeniu PA0:

```
PORTA |= 1 << PA0;
```

- Ustawienie stanu niskiego na wyprowadzeniu PA0:

```
PORTA &= ~(1 << PA0);
```

Opóźnienia programowe

- Aby móc ich używać, trzeba zdefiniować częstotliwość taktowania:

```
#define F_CPU 8000000UL /* 8 MHz */
```

- Następnie należy włączyć plik nagłówkowy, kolejność jest istotna!

```
#include <util/delay.h>
```

- Teraz łatwo już można uzyskać potrzebne opóźnienie:

```
_delay_ms(500);
```

- Pętle opóźniające są dobrze skalibrowane tylko przy włączonej optymalizacji!
- Włączenie optymalizacji:
  Project → GCC / WinAVR flags → Optimization → Level 2.

Cały pierwszy program

```
#define F_CPU 8000000UL /* 8 MHz */

#include <avr/io.h>
#include <util/delay.h>

int main(void) {
  DDRA |= 1 << PA0;
  for (;;) {
    PORTA |= 1 << PA0;
    _delay_ms(500);
    PORTA &= ~(1 << PA0);
    _delay_ms(500);
  }
}
```

Jak to skompilować w VMLAB

- W laboratorium jest zainstalowany kompilator WinAVR bazujący na GCC.
- Podczas tworzenia nowego projektu w trzecim kroku (Step 3) należy wybrać GNU C Compiler i podać ścieżkę dostępu do niego (GCC path).
- W laboratorium kompilator jest zainstalowany w katalogu C:\Program Files\WinAVR-20100110

Plik projektu dla pierwszego programu

```
.MICRO "ATmega16"
.TOOLCHAIN "GCC"
.GCCPATH   "C:\Program Files\WinAVR-20100110"
.GCCMAKE   AUTO
.TARGET    "led.hex"
.SOURCE    "led.c"

.TRACE
.POWER VDD=5 VSS=0
.CLOCK 8meg
.STORE 2000m

D1 VDD N1
R1 N1 PA0 680
.PLOT V(PA0)
```

Obsługa przerwań (1)

- Przerwanie INT0 będzie zgłaszane przy zboczu opadającym:

```
uint8_t tmp;
tmp = MCUCR;
tmp &= ~(1 << ISC00);
tmp |= 1 << ISC01;
MCUCR = tmp;
```

- Przerwanie INT2 też będzie zgłaszane przy zboczu opadającym:

```
MCUCSR &= ~(1 << ISC2);
```

- Włączenie przerwań INT0 i INT2:

```
GICR |= 1 << INT0 | 1 << INT2;
```

- Wyzerowanie znaczników przerwań:

```
GIFR |= 1 << INTF0 | 1 << INTF2;
```

- Włączenie przerwań (ustawienie globalnego znacznika przerwań):

```
sei();
```

Obsługa przerwań (2)

- Potrzebny jest plik nagłówkowy:

```
#include <avr/interrupt.h>
```

- Przykładowa obsługa przerwania INT2 polega na zmianie stanu wyjścia na przeciwny (ang. *toggle*):

```
ISR(INT2_vect) {
  PORTA ^= 1 << PA2;
}
```

- Jeśli do wyjścia PA2 podłączymy diodę świecącą, a do wejścia INT2 (PB2) klawisz, to jego naciskanie będzie powodowało na przemian włączanie i wyłączanie diody.
- Można zaobserwować drgania styków klawisza.

Obsługa przerwań (3)

- Nieco wydumana procedura obsługi przerwania INT0 zmienia stan wyjścia co drugi raz:

```
ISR(INT0_vect) {
  static uint8_t toggle = 0;
  volatile uint16_t t;

  for (t = 0; t < 2000; ++t);

  if ((PIND & (1 << PD2)) == 0)
    toggle = !toggle;
  if (toggle)
    PORTA ^= 1 << PA0;
}
```

- Aktywne czekanie w procedurze obsługi przerwania jest wyjątkowo złym rozwiązaniem!
- Zostało użyte, aby uniknąć problemu drgania styków klawisza i zademonstrować użycie słowa kluczowego `volatile`.

Obsługa przerwań (4)

- Wyprowadzenie INT0 jest współdzielone z wyprowadzeniem PD2, które w zestawie jest domyślnie podłączone do linii D4 LCD.
- Mamy jeszcze do dyspozycji wejście przerwanie INT1 wyprowadzone na PD3, które w zestawie jest domyślnie podłączone do linii D5 LCD.
- Dlatego ważne jest napisanie procedur obsługi LCD w taki sposób, aby łatwo można było skonfigurować LCD na dowolnych wyprowadzeniach.
- Procedura obsługi przerwania może być pusta.

```
EMPTY_INTERRUPT(INT1_vect);
```

- Więcej o procedurach obsługi przerwań można dowiedzieć się w pomocy do pliku avr/interrupt.h.

Cały drugi program

```
#include <avr/io.h>
#include <avr/interrupt.h>

ISR(INT0_vect) {
  static uint8_t toggle = 0;
  volatile uint16_t t;

  for (t = 0; t < 2000; ++t);
  if ((PIND & (1 << PD2)) == 0)
    toggle = !toggle;
  if (toggle)
    PORTA ^= 1 << PA0;
}

ISR(INT2_vect) {
  PORTA ^= 1 << PA2;
}
```

Cały drugi program, cd.

```
int main(void) {
  uint8_t tmp;

  DDRA = 1 << PA0 | 1 << PA2;
  tmp = MCUCR;
  tmp &= ~(1 << ISC00);
  tmp |= 1 << ISC01;
  MCUCR = tmp;
  MCUCSR &= ~(1 << ISC2);
  GICR |= 1 << INT0 | 1 << INT2;
  GIFR |= 1 << INT0 | 1 << INT2;
  sei();

  for (;;);
}
```

Plik projektu drugiego programu

```
.MICRO "ATmega16"
.TOOLCHAIN "GCC"
.GCCPATH   "C:\Program Files\WinAVR-20100110"
.GCCMAKE   AUTO
.TARGET    "int.hex"
.SOURCE    "int.c"

.TRACE
.POWER VDD=5 VSS=0
.CLOCK 8meg
.STORE 2000m
```

Plik projektu drugiego programu, cd.

```
D1  VDD N1
RD1 N1 PA0 680

D3  VDD N3
RD3 N3 PA2 680

K0  PD2 GND ; PD2 = INT0
RK0 PD2 VDD 10k

K2  PB2 GND ; PB2 = INT2
RK2 PB2 VDD 10k

.PLOT V(PA0) V(PD2) V(PA2) V(PB2)
```

Umieszczanie danych w pamięci FLASH

- Mikrokontroler ma mało RAM, a stosunkowo dużo FLASH.
- Szkoda przeznaczać cenny RAM na trzymanie stałych, które lepiej jest umieścić w pamięci FLASH:

```
#include <avr/pgmspace.h>

const prog_char foo[] = "Foo";
const prog_uint16_t bar[] = {1, 2, 3};
```

- Na powyższych zmiennych nie można używać funkcji ze standardowej biblioteki C.
- Plik nagłówkowy avr/pgmspace.h dostarcza deklaracji potrzebnych funkcji, np.: `strcpy_P`, `strcmp_P`, `pgm_read_byte`, `pgm_read_word` itd.

Gdzie szukać dalszych informacji?

- VMLAB: Help → WinAVR → AVR LIBC
  - Main Page
  - User Manual
  - Library Reference
  - FAQ
  - Alphabetical Index
  - Example Projects
- Internet:
  - `http://www.nongnu.org/avr-libc`
  - `http://www.nongnu.org/avr-libc/user-manual/modules.html`

AVR Studio 4

- Dostępne jest też zintegrowane środowisko programistyczne firmy Atmel.
- Używa najnowszej wersji Asemblera.
- Automatycznie integruje się z kompilatorem C WinAVR.
- Ma lepszą niż VMLAB diagnostykę błędów kompilacji.
- Dla ATmega16 oferuje bardzo słaby emulator.
- Sprawdza się, jeśli posiadamy JTAG.
- Aby zdefiniować nowy projekt w C, należy w menu Project wybrać New project i podać m.in.:
  - Project type: AVR GCC
  - Debug platform: AVR Simulator, AVR Simulator 2 lub typ JTAG-a
  - Device: ATmega16 lub inny mikrokontroler
- Ustawienia kompilatora można zmienić w Project $\rightarrow$ Configuration Options.
- Warto zajrzeć do pomocy w Help $\rightarrow$ avr-libc Reference Manual.