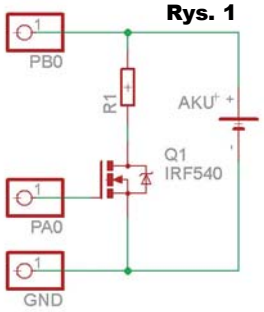
Kurs AVR – lekcja 13

W pierwszym zadaniu domowym chodziło o przepisanie wyłącznika zmierzchowego tak, aby nie opierał się o komparator analogowy, a o przetwornik analogowo-cyfrowy. Wówczas progi histerezy można ustawiać, np. wpisując odpowiednią wartość z klawiatury lub też przy danym natężeniu oświetlenia zapamiętywać bieżącą wartość jako górny lub dolny próg przełączania. Zrealizujmy tę drugą opcję. Przykładowy kod źródłowy zamieszczony jest na **listingu 1**.

```
#include <avr/io.h>
int main(void) {
ADCSRA |= _BV(ADEN); //włącz ADC
ADMUX |= _BV(REFS0); //AVCC preskaler128
ADCSRA |=_BV(ADPS0)|_BV(ADPS1)|_BV(ADPS2);
 uint16_t high = 0;
 uint16_t low = 0;
 DDRB |= _BV(DDB0);
 PORTB |= _BV(PORTB1) | _BV(PORTB2);
 while(1) {
  ADCSRA |= _BV(ADSC);
  while(ADCSRA & _BV(ADSC));
  uint16_t adc = ADCW;
  if (adc > high) {
   PORTB |= _BV(PORTB0);    }
  if (adc < low) {
   PORTB &= ~_BV(PORTB0);   }
  if (!(PINB & _BV(PORTB1))) {
   low = adc;    }
  if (!(PINB & _BV(PORTB2))) {
   high = adc;    }  }  }
```

Po standardowym skonfigurowaniu ADC następuje inicjalizacja zmiennych przechowujących progi przełączania, konfiguracja pinu wyjściowego (pin 0 portu B) oraz pinów dla przycisków (piny 1 i 2 portu B). W pętli głównej odczytywane jest natężenie światła z ADC a następnie porównywane z progami. Tak jak poprzednio zakładamy, że fotorezystor jest włączony od strony masy i jego rezystancja spada wraz ze wzrostem natężenia światła. Jeśli więc napięcie jest wyższe od wyższego progu, znaczy to, że natężenie światła jest bardzo małe i należy ustawić pin wyjściowy w stan wysoki. Jeśli napięcie będzie niższe od niższego progu, pin przestawiany jest w stan niski. Jeśli naciśniemy przycisk podłączony do pinu 1, bieżące natężenie światła zostanie zapamiętane jako próg niższy. Jeśli naciśniemy przycisk podłączony do pinu 2 portu B, zapamiętany zostanie wyższy próg.

Jak skonfigurować progi? Niski próg powinien być ustawiony przy wyższym natężeniu światła, a wysoki przy niższym. Czyli przycisk podłączony do pinu 1 powinien zostać wciśnięty, gdy natężenie światła będzie nieco większe, a przycisk dla pinu 2, gdy będzie nieco mniejsze. Wtedy histereza będzie działała prawidłowo.

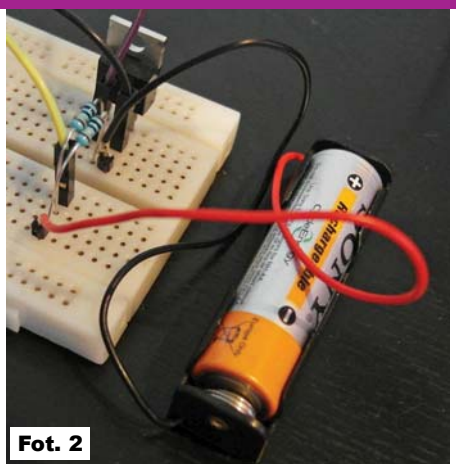


Rys. 1

Aby zrealizować ćwiczenie na płytce testowej trzeba wyjście dzielnika z fotorezystorem podłączyć do pinu 0 portu A. Pin 1 portu B łączymy z pinem K1 złącza KEYB, a pin 2 portu B z pinem K2. Natomiast pin W1 złącza KEYB łączymy z masą. W ten sposób wciskanie przycisków S1 i S2 będzie powodowało ściąganie pinów 1 i 2 portu B do masy. Piny te są wewnętrznie podciągnięte do plusa dzięki wpisaniu jedynek do rejestru PORTB, na pozycjach bitów PORTB1 i PORTB2. Pin 0 portu B, pełniący funkcję wyjścia, można np. podłączyć do którejś z diod dostępnych na złączu LED.

Drugie zadanie domowe polegało na napisaniu programu obliczającego pojemność akumulatorków, np. NiMH lub Li-Ion. Aby ją obliczyć, musimy zmierzyć czas jaki jest potrzebny do rozładowania świeżo naładowanego akumulatorka do minimalnego bezpiecznego napięcia. Obciążeniem akumulatorka może być aktywne obciążenie utrzymujące stały prąd rozładowania lub zwykły rezystor. W tym drugim przypadku prąd rozładowania będzie spadał wraz ze spadającym napięciem z akumulatora. Konieczne jest wtedy ciągłe obliczanie prądu. Nie jest to jednak duża trudność a część analogowa jest znacznie prostsza, dlatego przyjrzymy się właśnie tej opcji.

Prąd akumulatorka zmienia się i trzeba go mierzyć co jakiś czas. Ile powinien on wynosić? Jeśli mierzymy pojemność typowego akumulatorka NiMH, chcielibyśmy otrzymać wartość wyrażoną w miliamperogodzinach. Przy prądzie rozładowania wynoszącym 1A, jedna miliamperogodzina energii zostanie pobrana w 3,6 sekundy. Możemy właśnie co taki okres mierzyć napięcie, obliczać prąd oraz aktualnie pobraną energię i wyświetlać wynik. Oczywiście prąd rozładowania nie musi wynosić 1A. W zależności od zalecanego przez producenta maksymalnego prądu rozładowania, mocy wydzielającej się na obciążeniu oraz od tego, jak szybko będziemy chcieli rozładować akumulatorek, dobierać będziemy prąd rozładowania i rezystancję obciążenia. Ponadto prąd będzie się zmniejszał w trakcie rozładowania akumulatorka. W związku z tym co 3,6 sekundy potrzebujemy do aktualnego wyniku pomiaru pojemności wyrażonego w mAh dodawać wartość będącą liczbą amperów wypływających z akumulatorka. Jeśli np. w danej chwili na akumulatorku jest napięcie 1,2V, a rezystor obciążający ma 5Ω, to wynik pomiaru w odstępie 3,6 s zwiększymy o 0,24 mAh.


Fot. 2

Nasz program pomiarowy nie może dopuścić do nadmiernego rozładowania akumulatorka. Powinien więc odłączyć akumulatorek od obciążenia po osiągnięciu napięcia minimalnego. Potrzebny jest więc jakiś element, za pomocą którego mikrokontroler będzie mógł podłączać i odłączać obciążenie. W tej roli może wystąpić tranzystor MOSFET, który będzie sterowany z pinu mikrokontrolera (**rysunek 1**). W przykładzie z listingu 2 jest to pin 0 portu B. Przykładowe połączenie na płytce stykowej z dwoma rezystorami 10Ω połączonymi równolegle w roli obciążenia 5Ω przedstawione jest na **fotogtafii 2**.

```
#include <avr/io.h>
#include <util/delay.h>
#include "lcd.h"
#define LOADR 5.0
#define UMIN 0.9

int main(void) {
 ADCSRA |= _BV(ADEN); //włącz ADC
 ADMUX |= _BV(REFS0); //AVCC
 //preskaler 128
   ADCSRA   |=   _BV(ADPS0)|_BV(ADPS1)|_
BV(ADPS2);   //włącz MOSFET
 DDRB = _BV(DDD0);
 PORTB |= _BV(PB0);
 lcdInit();
 lcdInitPrintf();
 float capacity = 0.0;
 while (1) {     //zmierz napięcie
  ADCSRA |= _BV(ADSC);
  while(ADCSRA & _BV(ADSC));
  float u = ADCW * 5.0 / 1024;
  lcdGotoXY(0, 0);
  printf("U=%.2fV", u);
  //zaktualizuj pojemność
  float i = u / LOADR;
  capacity += i;
  printf(" I=%.3f", i);
  lcdGotoXY(0, 1);
  printf("C=%.2f mAh", capacity);
  //jeśli napięcie jest za niskie,
  //odłącz obciążenie i zakończ
  if (u < UMIN) {
   lcdGotoXY(13, 1);
   printf("end");
   PORTB &= ~_BV(PB0);
   while(1);
  }
  _delay_ms(3600);
 }
}
```

Przykładowe rozwiązanie rozpoczyna się inicjalizacją ADC i wyświetlacza (na porcie D). Konfigurowany jako wyjście jest też pin, który steruje tranzystorem dołączającym obciążenie do akumulatorka. W głównej pętli odczytywane jest napięcie i obliczany prąd płynący przez obciążenie. Z prądu obliczana jest liczba mAh pobranych od ostatniej iteracji pętli, przechowywana w zmiennej capacity. Napięcie i wynik pomiaru są wyświetlane na bieżąco na LCD. Gdy napięcie spadnie poniżej wartości granicznej, wyświetlany jest napis „end", tranzystor zostaje wyłączony, a program zatrzymuje się, wpadając w pętlę nieskończoną. Parametry takie jak rezystancja obciążenia i napięcie minimalne definiowane są za pomocą makr LOADR i UMIN. Dzięki temu, aby je zmienić, nie trzeba ich szukać po całym kodzie. Sam kod też staje się czytelniejszy, bo nie trzeba zgadywać, co dana liczba oznacza. Program jest przykładową, prostą realizacją zadania. Warto go rozbudować, np. dodając możliwość restartu pomiaru. Obecnie wymagany jest reset procesora, a akumulatorek musi być podłączony w momencie startu programu.

Na koniec kilka słów o dokładności pomiaru. W kodzie wynik powstaje z wielu dodawań wykonywanych na zmiennej capacity. Należy pamiętać, że użyty tryb float nie zapewnia idealnej dokładności obliczeń, np. gdy do dużej liczby dodawana jest względnie mała liczba. Popatrzmy na przykład:

```
float a = 0.0;
for(int i = 0; i < 1000000; i++) {
 a += 0.23;  }
```

Po wykonaniu się pętli zmienna a będzie mieć wartość ok. 230987, a nie równe 230000. Różnica wynosi 0,43%. W naszym programie liczba iteracji pętli zależy od tego, jak długo będziemy rozładowywać akumulatorek. Zwykle tak dobierzemy rezystor, aby pomiar trwał maksymalnie kilka godzin, a więc kilka tysięcy iteracji pętli. Błąd będzie więc dużo mniejszy niż w powyższym przykładzie z milionem iteracji. W naszym mierniku takie ułamki procenta nie mają znaczenia, jednak w innych programach, np. przetwarzających dane finansowe, taki błąd będzie niedopuszczalny. Wtedy konieczne będzie użycie typu całkowitego (np. int).

Drugie źródło niedokładności leży w odmierzaniu czasu. Funkcja opóźniająca _delay_ms() dla opóźnień powyżej 262 ms ma dokładność rzędu 0,1 s, a więc tutaj ok. 3%. Ponadto opóźnienie generowane jest też przez wykonujący się pozostały kod w pętli, szczególnie pomiar wykonywany przez ADC oraz sterowanie LCD. Obejściem byłoby zastosowanie timera. Na dokładność wpływa też źródło taktowania procesora: rezonator kwarcowy lub wewnętrzny generator RC. Ten pierwszy oczywiście ma częstotliwość dużo dokładniejszą i stabilniejszą.

Na pomiar wpływa też oczywiście ADC, w tym jego rozdzielczość, liniowość i stabilność temperaturowa. W naszym przykładzie decydujące znaczenie dla wyniku pomiaru i tak ma przyjęte przez nas minimalne napięcie rozładowania. Przykład ten pokazuje jednak, że na pomiar może wpływać wiele czynników i realizując takie elementy programu, jak odmierzanie czasu czy obliczenia matematyczne, trzeba brać pod uwagę, jak bardzo mogą one zmienić wynik.

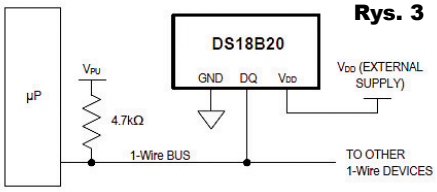
Protokół 1-Wire

W tej lekcji zajmiemy się protokołem 1-Wire. Został on opracowany przez firmę Dallas, przejętą później przez Maxim i umożliwia komunikację z takimi układami scalonymi jak termometry cyfrowe, pamięci, timery czy monitory baterii. Jego charakterystyczną cechą jest to, że do komunikacji oprócz masy potrzebny jest tylko jeden przewód. Stąd nazwa 1-Wire. Ponadto niektóre układy 1-Wire występują w obudowach pastylkowych, wyglądających jak baterie. Tego typu układy nie są lutowane w płytkę lub do przewodów, ale przykładane do odpowiednich styków pełnią np. funkcję kluczy do domofonu. Ponadto każdy układ 1-Wire ma unikalny 48-bitowy numer pozwalający odróżniać od siebie poszczególne egzemplarze. Daje to nie tylko możliwość adresowania układów 1-Wire, ale też ogólnie dodawania funkcji unikalnego numeru seryjnego do budowanych układów elektronicznych. Typowym przykładem jest tutaj układ DS2401, którego jedyną funkcją jest właśnie zwracanie numeru seryjnego.

Od strony elektrycznej linia/szyna transmisyjna 1-Wire wygląda w ten sposób, że jest ona podciągnięta do plusa zasilania przez rezystor 4,7kΩ, a podłączone do niej układy mają wyjścia z otwartym kolektorem/drenem. W spoczynku szyna jest podciągnięta do plusa (V_{PU}) rezystorem (**rysunek 3**). W trakcie transmisji danych układy 1-Wire w odpowiednich momentach ściągają ją do masy.

Niektóre układy 1-Wire mogą być zasilane z linii komunikacyjnej i wówczas nawet nie jest wymagane oddzielne podłączanie zasilania. Ułatwia to wykonywanie połączeń, bo wystarczy zwykły dwużyłowy kabel. Takie zasilanie pasożytnicze realizowane jest dzięki wbudowanemu kondensatorowi o pojemności 800pF, który pozwala układowi działać także w momencie, gdy szyna jest zwarta do masy i nie otrzymuje on zasilania. Należy jednak pamiętać, że gdy jest napięcie na szynie 1-Wire, to pochodzi ono z rezystora podciągającego i ilość prądu, jaką układ może pobrać, jest ograniczona. Najprościej

Rys. 3

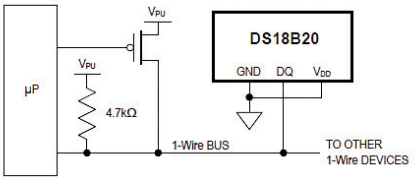


wtedy zmniejszyć rezystor podciągający do np. 3,3 lub 2,7kΩ, ale zalecanym rozwiązaniem jest zwarcie linii do plusa za pomocą tranzystora (**rysunek 4**). Oczywiście takie podciągnięcie może mieć miejsce tylko wtedy, gdy układ 1-Wire wykonuje operacje wymagające większego prądu i nie jest prowadzona transmisja danych. Zalecane jest standardowe połączenie trójprzewodowe, a połączenie dwuprzewodowe z zasilaniem pasożytniczym powinno być traktowane jako wyjątek. Należy też pamiętać, że standard 1-Wire nie został zaprojektowany do wykonywania połączeń na większe odległości, rzędu dziesiątek metrów.

Z logicznego punktu widzenia transmisją 1-Wire zarządza urządzenie pełniące funkcję master (nadrzędną), którym zwykle jest mikrokontroler. Pozostałe układy pełnią funkcję slave (podrzędną). Master inicjuje każdy rodzaj transmisji, zarówno wysyłanie, jak i odbieranie danych. Może on wykonać cztery rodzaje operacji:

- Reset/Presence – resetuje szynę 1-Wire oraz pozwala wykryć, czy są do niej podłączone jakiejś układy slave. Master ściąga szynę w dół na przynajmniej 480 mikrosekund, a następnie pozwala, aby powróciła do stanu wysokiego. Wtedy układy slave, jeśli są obecne, w ciągu 60 mikrosekund muszą zareagować, ściągając szynę na okres 60 mikrosekund. Jeśli master nie wykryje stanu niskiego spowodowanego przez jeden lub więcej układów slave, musi uznać, że nie są one podłączone i dalsza komunikacja jest bezcelowa.
- Wysłanie bitu 1 – master ściąga szynę w dół na 1–15μs, a następnie pozostawia ją podciągniętą na pozostałą część 60-mikrosekundowego slotu (45–59μs).
- Wysłanie bitu 0 – master ściąga szynę w dół na 60–120μs
- Odczyt bitu – wygląda jak wysłanie bitu 1: master ściąga szynę na 1–15μs, a przez pozostałe 60 mikrosekund układ slave może kontynuować ściągnięcie szyny w dół. Jeśli to zrobi, oznacza to transmisję zera, jeśli pozwoli na powrót szyny do stanu wysokiego, będzie to oznaczało transmisję jedynki.

Rys. 4



Odstępy między operacjami muszą wynosić co najmniej 1 mikrosekundę. Kolejne bity, transmitowane od najmłodszego, tworzą bajty. Zalecany algorytm dla mastera z konkretnymi okresami wygląda następująco:

- Reset/Presence – ściągnąć linię na 480µs, sprawdzić jej stan po 70µs, odczekać 410µs.
- Wysłanie 1 – ściągnąć linię na 6µs, pozostawić podciągniętą na 64µs.
- Wysłanie 0 – ściągnąć linię na 60µs, pozostawić podciągniętą przez 10µs.
- Odczyt bitu – ściągnąć linię na 6µs, sprawdzić jej stan po 9µs, odczekać 55µs.

Wiemy już, jak przesyłać pojedyncze bity. Jak natomiast komunikacja wygląda na wyższym poziomie? Ponieważ na magistrali 1-Wire może znajdować się wiele układów slave, po resecie linii master wydaje jedną z tzw. komend ROM służących do adresacji. Dopiero potem może zostać wydana opcjonalnie komenda wywołująca jakąś funkcję na urządzeniu slave. Mówimy o komendach ROM (ROM Commands), ponieważ odnoszą się do unikalnego adresu urządzenia, zapisanego w jego pamięci ROM. Jest pięć komend ROM:

- Read ROM (33h) – odczytuje numer seryjny urządzenia 1-Wire pod warunkiem, że jest ono jedynym urządzeniem slave na linii.
- Match ROM (55h) – adresuje konkretne urządzenie. Po wysłaniu tej komendy trzeba wysłać numer seryjny danego urządzenia. Wysłana potem komenda spowoduje wykonanie określonej funkcji na tym właśnie urządzeniu.
- Skip ROM (CCh) – adresuje pojedyncze urządzenie, jeśli jest tylko jedno na linii.
- Alarm Search (ECh) – pozwala wyszukać urządzenia, na których uaktywnił się alarm.
- Search ROM (F0h) – za pomocą specjalnego algorytmu pozwala wykryć wszystkie urządzenia slave na linii 1-Wire.

Obsługa 1-Wire w mikrokontrolerach

W przeciwieństwie do standardów takich jak I²C, SPI czy port szeregowy, w popularnych mikrokontrolerach nie znajdziemy sprzętowej obsługi 1-Wire. Musimy więc radzić sobie inaczej i mamy do dyspozycji trzy główne metody. Pierwsza z nich to generowanie odpowiednich sygnałów programowo, czyli „machając" nogą mikrokontrolera, analogicznie jak robiliśmy to, obsługując LCD. Można też wykorzystać port szeregowy, który przy odpowiednim oprogramowaniu można zmusić do wysyłania i odbierania danych zgodnie ze standardem 1-Wire. Trzecia metoda to użycie gotowego sterownika, np. DS2465.

W tej lekcji skupimy się na pierwszym podejściu. Jak widać, przedstawiony powyżej algorytm obsługi 1-Wire jest prosty i napisanie odpowiedniej biblioteki komunikacyjnej nie będzie skomplikowane, szczególnie po doświadczeniach z obsługą wyświetlacza alfanumerycznego. Zacznijmy od podstawowej rzeczy, czyli ustawianiu wysokiego lub niskiego stanu na linii 1-Wire. Została ona zaprojektowana z myślą o układach z wyjściem z otwartym drenem, jednak nasza ATmega32 nie ma takich wyjść. Jej porty pracują w trybie push-pull, który wymusza określony stan logiczny. Jeśli nasz mikrokontroler ustawi wysoki stan na linii, a jakiś inny układ będzie chciał ją ściągnąć do masy, nastąpi zwarcie. Aby więc inne układy mogły ustawiać niski stan na linii, pin mikrokontrolera musi wtedy pracować jako wejście. Natomiast gdy to nasz mikrokontroler ma wymuszać stan niski, jego pin musi pracować jako wyjście. Musimy więc manipulować odpowiednim bitem rejestru DDRn.

A co z rejestrem PORTn? W zasadzie bit odpowiadający pinowi podłączonemu do szyny 1-Wire powinien być cały czas wyzerowany. Gdy za pomocą DDRn pin będzie wyjściem, zero pozwoli ściągnąć linię w dół. Gdy natomiast będzie wejściem, wyzerowany bit PORTn nie będzie przeszkadzał. Gdyby był ustawiony, spowodowałby włączenie wewnętrznego podciągnięcia do plusa zasilania, ale jest to zbędne, bo już mamy rezystor podciągający. Tutaj pewnie Czytelnicy zapytają, czy to wewnętrzne podciągnięcie nie zastąpi nam rezystora podciągającego. Można by wtedy teoretycznie mikrokontroler podłączyć bezpośrednio do układu 1-Wire. Ten wewnętrzny rezystor podciągający wewnątrz ATmega32 ma jednak rezystancję w okolicach 36 kΩ, a więc dużo więcej niż zalecany 4,7 kΩ. Zapewne niektóre układy 1-Wire będą działać poprawnie, ale takie pójście na skróty nie jest zalecane, szczególnie przy zasilaniu pasożytniczym. Jak się okazuje, brak wyjść z otwartym drenem nie jest w naszym mikrokontrolerze przeszkodą. Jest nawet zaletą, bo przy zasilaniu pasożytniczym znika potrzeba stosowania dodatkowego tranzystora. Można po prostu przestawić pin na pracę jako wyjście na czas, gdy układ 1-Wire przeprowadza operacje wymagające zwiększonego prądu zasilania. W tej lekcji skupimy się na zasilaniu standardowym.

Biblioteka 1-Wire

Znając już algorytm komunikacji oraz zasady sterowania pinem mikrokontrolera, przystąpmy do pisania biblioteki obsługującej 1-Wire. W tym celu w projekcie dodajmy nowe pliki 1wire.c oraz 1wire.h. Tradycyjnie w pliku .h będziemy umieszczać makra i deklaracje funkcji, a w pliku .c definicje funkcji. Tak jak w przypadku LCD, rozpoczniemy od funkcji najniższego poziomu, przechodząc coraz wyżej.

Te najbardziej niskopoziomowe operacje to ustawienie stanu niskiego oraz wysokiego na szynie 1-Wire. Tutaj dla wygody zdefiniujemy sobie makra, aby nie wpisywać na sztywno określonych rejestrów i bitów.

```
#define OW_PIN PB0
#define OW_DDR DDRB
#define OW_INPUT PINB
#define OW_LOW OW_DDR |= _BV(OW_PIN)
#define OW_HIGH OW_DDR &= ~_BV(OW_PIN)
```

Za pomocą pierwszy trzech makr będziemy mogli wygodnie konfigurować pin, którym będziemy kontrolować magistralę 1-Wire. Pierwsze określa sterowany pin, drugie rejestr kierunku (wejście/wyjście), trzecie rejestr odczytu stanu portu. Za pomocą makr OW_LOW i OW_HIGH możemy wygodnie przestawiać nasz pin w stan niski i wysoki. Zastosowano nazwy zaczynające się od liter OW, a nie 1W czy 1WIRE, gdyż nazwy makr, funkcji czy zmiennych nie mogą zaczynać się od cyfr.

Teraz na podstawie zamieszczonego wcześniej algorytmu możemy napisać funkcje do wykonywania podstawowych operacji na linii 1-Wire.

```
void oneWireSendZero() {
 OW_LOW;
 _delay_us(60);
 OW_HIGH;
 _delay_us(10);  }

void oneWireSendOne() {
 OW_LOW;
 _delay_us(6);
 OW_HIGH;
 _delay_us(64);  }

uint8_t oneWireReset() {
 uint8_t ret = 0;
 OW_LOW;
 _delay_us(480);
 OW_HIGH;
 _delay_us(70);
 if (!(OW_INPUT & _BV(OW_PIN))) ret = 1;
 _delay_us(410);
 return ret;  }

uint8_t oneWireReceiveBit() {
 uint8_t ret = 0;
 OW_LOW;
 _delay_us(6);
 OW_HIGH;
 _delay_us(9);
 if (OW_INPUT & _BV(OW_PIN)) ret = 1;
 _delay_us(55);
 return ret;  }
```

Jak widać, funkcje są prostym przeniesieniem założeń algorytmu. Funkcja resetująca zwraca 1, jeśli wykryje ściągnięcie linii w dół przez urządzenie slave. Bardzo podobna do niej jest funkcja odbierająca jeden bit. Tutaj nie ma odwracania stanu logicznego i zwracana jest jedynka jeśli urządzenie nie ściągnęło linii w dół. Oczywiście odpowiednio ustawione są też czasy trwania poszczególnych faz.

Napiszmy prosty program testujący wykrywanie układu 1-Wire. Na **listingu 4** zawarto samą funkcję main(), bez definicji funkcji i makr.

```
int main(void) {
 DDRA = _BV(DDA0);
 PORTB |= _BV(PORTB1);
 while(1)
 {
  if (oneWireReset()) {
   PORTA |= _BV(PORTA0);
  } else {
   PORTA &= ~_BV(PORTA0);
  }   }  }
```

Program ma za zadanie zaświecać LED podłączony do pinu 0 portu A, gdy na linii 1-Wire, podłączone do pinu 0 portu B, obecne jest urządzenie slave. Aby sprawdzić jego działanie na płytce testowej, pin 0 portu A należy połączyć z którymś z LED-ów, a pin 0 portu B z pinem Temp na złączu MISC. Pin Temp jest połączony z popularnym układem 1-Wire DS18B20 i ma podciągnięcie do plusa zasilania. Przy takim połączeniu dioda powinna świecić, sygnalizując obecność układu. Jeśli jednak pin 0 portu B podłączymy do miejsca podciągniętego do plusa, ale niepołączonego z żadnym układem 1-Wire, dioda zgaśnie. Można to sprawdzić, łącząc ten pin np. z pinem SCL lub SDA na złączu RTC.

Przetestujmy też wysyłanie jedynki. 1-Wire wymaga krótkich impulsów, a funkcje opóźniające nie dają idealnej dokładności. Wykonywanie instrukcji między opóźnieniami też wprowadza opóźnienie. Tutaj kod testowy będzie następujący:

```
int main(void)
{  while(1)  { oneWireSendOne(); }  }
```

Mając pin Temp złącza MISC połączony z mikrokontrolerem, podłączamy do niego oscyloskop. Okazuje się, że przy taktowaniu wewnętrznym generatorem RC 1 MHz opóźnienie mające wynosić 6 mikrosekund wynosi 8 µs (**rysunek 5**). Jest to błąd akceptowalny, ponieważ maksymalnie stan niski może trwać 15 µs. Możemy jeszcze sprawdzić, jak sytuacja wygląda przy taktowaniu kwarcem 16 MHz. Tutaj opóźnienie wynosi dokładnie tyle, ile powinno (**rysunek 6**). Tak więc nasza funkcja będzie działała prawidłowo nawet przy taktowaniu mikrokontrolera wolnym zegarem. Oczywiście jak zwykle należy pamiętać o odpowiednim ustawieniu makra F_CPU.

Mając napisane funkcje najniższego poziomu, możemy przejść wyżej, do wysyłania i odbierania bajtów.

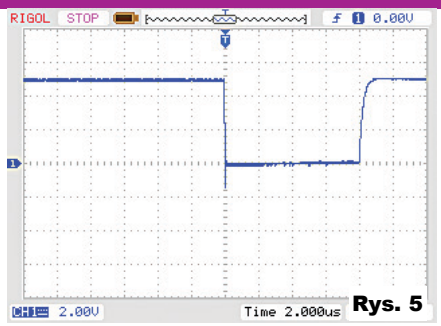
```
void OneWireSendByte(uint8_t data) {
 for (uint8_t i = 0; i < 8; i++) {
  if(data & 1) { oneWireSendOne();
  } else {
   oneWireSendZero();   }
  data >>= 1;
 }  }

uint8_t oneWireReceiveByte() {
 uint8_t data = 0;
 for (uint8_t i = 0; i < 8; i++) {
  data >>= 1;
     if  (oneWireReceiveBit())  data  |=
0b10000000;
 }
 return data;
}
```

Transmisja bajtów wykonywana jest od najmłodszego bitu do najstarszego. Wysyłając bajt, sprawdzamy, czy jego najmłodszy bit jest jedynką. Jeśli tak, wywoływana jest funkcja wysyłająca jedynkę, w przeciwnym wypadku funkcja wysyłająca zero. Po wysłaniu bitu bity w bajcie są przesuwane w prawo tak, że teraz najmłodszym bitem staje się bit drugi. Cykl powtarzany jest osiem razy, aby wysłać wszystkie bity w bajcie. Analogicznie działa odbieranie. Na początku wynikowy bajt jest zerowany. Potem osiem razy odbierany jest jeden bit, przy czym najpierw wykonywane jest jeszcze przesunięcie bitowe, aby zrobić miejsce dla nowego bitu. Jeśli odebrana zostanie jedynka, zostaje ustawiona w bajcie wynikowym na najstarszej pozycji. W ten sposób do bajtu wynikowego trafiają wszystkie odebrane bity.

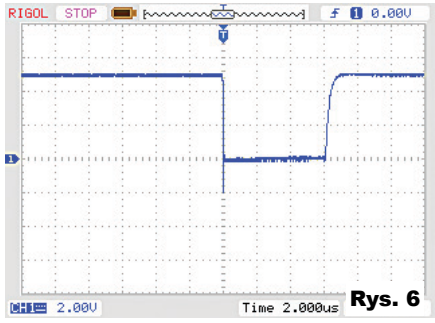
Gdy nasza biblioteka ma funkcje do wysyłania i odbierania bajtów, możemy wykonać pierwsze operacje. Na początek odczytajmy identyfikator egzemplarza układu DS18B20, który wlutowany jest w płytkę testową. W tym celu po resecie linii 1-Wire musimy wydać komendę Read ROM, czyli wysłać bajt o wartości 33h. Dostaniemy odpowiedź składającą się z 8 bajtów: bajt rodziny układów DS18B20, mający wartość 28h, następnie 6 bajtów identyfikatora i bajt sumy kontrolnej CRC. Choć właściwy identyfikator to 6 bajtów (48 bitów), w wyszukiwaniu czy adresowaniu urządzeń używa się 8-bajtowego (64-bitowego) identyfikatora, czyli także kodu rodziny oraz sumy kontrolnej. Przykładowy kod testowy, korzystający z właśnie napisanej biblioteki 1-Wire oraz biblioteki LCD, zamieszczony jest na **listingu 5**.

Program jest bardzo prosty i polega na inicjalizacji LCD, resecie magistrali 1-Wire, wysłaniu komendy Read ROM oraz odebraniu i wyświetleniu odpowiedzi (**rysunek 7**). Aby sprawdzić, czy poprawnie odebraliśmy dane, musimy zweryfikować sumę kontrolną. Możemy w tym celu posłużyć się gotową funkcją dostarczaną przez firmę Atmel. Dla spójności nazewnictwa nazwiemy ją oneWireComputeCRC8(). Przetwarza ona po jednym bajcie i za każdym razem zwraca zaktualizowaną wartość sumy kontrolnej. Jako pierwszy parametr podajemy nowy bajt, a jako drugi dotychczas obliczoną wartość sumy CRC. Aby wyświetlić poprawność sumy kontrolnej, ostatnie wywołanie funkcji printf() zastąpmy następującymi linijkami:

```
uint8_t crcExpected = oneWireComputeCRC8(family, 0);
for (uint8_t i = 0; i < 6; i++)
 crcExpected = oneWireComputeCRC8(rom[i], crcExpected);
printf("family:%02x CRC:%s", family, crc == crcExpected ? "OK" : "E");
```



Rys. 5



Rys. 6

Listing 5

```
#include <avr/io.h>
#include "lcd.h"
#include "wire.h"

int main(void)
{
 lcdInit();
 lcdInitPrintf();
 oneWireReset();
 oneWireSendByte(0x33);
 uint8_t family = oneWireReceiveByte();
 uint8_t rom[6];
 for (uint8_t i = 0; i < 6; i++)
        rom[i] = oneWireReceiveByte();
 uint8_t crc = oneWireReceiveByte();
 printf("id: ");
 for (uint8_t i = 0; i < 6; i++)
         printf("%02X", rom[i]);
 lcdGotoXY(0, 1);
 printf("family:%02x CRC:%02X", family, crc);
 while(1) { }
}
```

Rys. 7

Suma kontrolna liczona jest ze wszystkich odebranych bajtów, z wyjątkiem jej samej. Zaczynamy więc od pierwszego odebranego bajtu, jakim jest kod rodziny układów. Ponieważ jest to początek obliczania sumy, jako jej wartość startową podajemy zero. Obliczana suma przechowywana jest w zmiennej crcExpected. Następnie obliczamy sumę dalej, korzystając z bajtów identyfikatora układu. Dla każdego bajtu przekazujemy do funkcji oneWireComputeCRC8() tenże bajt oraz aktualną sumę i pobieramy nową wartość sumy. Na koniec, jeśli obliczona suma kontrolna jest taka sama jak odebrana suma w zmiennej crc, wyświetlany jest napis CRC:OK. W przeciwnym wypadku pojawi się napis CRC:E, oznaczający błąd.

Wyszukiwanie układów 1-Wire

Ważną funkcją szyny 1-Wire jest możliwość wyszukiwania urządzeń, czyli utworzenia listy ich identyfikatorów (64 bity, razem z kodem rodziny i CRC). W tym celu master wydaje komendę Search ROM. W tym momencie każde urządzenie slave wystawia pierwszy bit swojego identyfikatora. Ze względu na sposób działania 1-Wire (podciągnięcie do plusa i wyjścia z otwartym drenem) stan linii będzie de facto wynikiem działania logicznej operacji AND na tych wszystkich bitach. Następnie master wykonuje drugi odczyt bitu. Tym razem urządzenia wystawiają jeszcze ten sam bit, ale zanegowany. Znów sprzętowo wykona się operacja AND. Jeśli wszystkie urządzenia miały jedynkę jako pierwszy bit, master odczyta bity 1 i 0. Jeśli wszystkie miały zero, master odczyta bity 0 i 1. W obu tych wypadkach master zna już pierwszy bit, bo wszystkie urządzenia mają taki sam. Wysyła wtedy ten właśnie bit, co jest informacją dla urządzeń slave, że mają wystawić kolejny bit. Jeśli natomiast odczytane zostaną bity 0 i 0, oznacza to, że master musi podjąć decyzję, czy wysłać bit 0, czy1. Jeśli wyśle 0, wyszukiwanie będzie kontynuowane z tymi urządzeniami slave, które mają akurat 0 na danej pozycji w identyfikatorze. Jeśli wyśle 1, wówczas wybrane zostaną urządzenia z jedynką. W ten sposób następuje rozgałęzienie i wybranie jednej z dwóch opcji. Gdy master odczyta wszystkie bity, będzie miał adres/identyfikator jednego urządzenia slave. Następnie musi wykonać wyszukiwanie od początku, ale pójść inną ścieżką. Tak więc wyszukiwanie nie polega na sprawdzeniu wszystkich 2^{64} kombinacji, ale na przejściu po drzewie rozgałęziającym się tam, gdzie identyfikatory zaczynają się różnić.

Jako funkcję wykonującą wyszukiwanie wykorzystamy lekko zmodyfikowany kod udostępniany przez firmę Atmel. Funkcja jest zawarta w bibliotece dołączonej do materiałów do tego numeru EdW i pozostawiam ją do samodzielnej analizy. Skupimy się natomiast krótko na jej wykorzystaniu. Deklaracja funkcji wygląda następująco:

```
int8_t oneWireSearchRom(uint8_t * bitPattern, uint8_t lastDeviation);
```

Jako pierwszy parametr podajemy 8-bajtową tablicę, do której zwrócony zostanie znaleziony identyfikator. Jeśli jest to kolejne wywołanie, podajemy poprzedni identyfikator. Drugi parametr to numer bitu, przy którym w poprzednim wywołaniu została podjęta decyzja co do wybranej gałęzi przeszukiwania. Funkcja zwraca nową pozycję decyzji. Jeśli wyszukany został ostatni układ, zwracane jest zero. Jeśli nie było żadnego układu na linii 1-Wire, zwracana jest wartość –1. Popatrzmy na **listing 6**, pokazujący identyfikatory dwóch pierwszych układów. Jeśli pierwsze wyszukanie nie zakończyło się błędem, zostaje wyświetlony znaleziony identyfikator. Jeśli nie był on ostatni, wykonywane jest drugie wyszukanie a wynik zostaje wyświetlony, jeśli nie było błędu. Ponieważ na płytce testowej znajduje się tylko jeden układ 1-Wire, ten kod najlepiej przetestować, dołączając drugi układ, DS18B20 lub inny 1-Wire, za pomocą płytki stykowej.

```
#include <avr/io.h>
#include "lcd.h"
#include "1wire.h"
int main(void)
{
 lcdInit();
 lcdInitPrintf();
 uint8_t rom[] = {0, 0, 0, 0, 0, 0, 0, 0};
 int8_t lastDeviation = oneWireSearchRom(rom, 0);
 if (lastDeviation > -1) {
  for (uint8_t i = 0; i < 8; i++) printf("%02X", rom[i]);
  lcdGotoXY(0, 1);  }
 if (lastDeviation > 0) {
  lastDeviation = oneWireSearchRom(rom, lastDeviation);
  if (lastDeviation > -1) {
   for (uint8_t i = 0; i < 8; i++) printf("%02X", rom[i]);
  }  }
 while(1) { }
}
```

Listing 6

Listing 7

```
#include <avr/io.h>
#include <util/delay.h>
#include "lcd.h"
#include "1wire.h"
#define DS18B20_CONVERT_T 0x44
#define DS18B20_READ_SCRATCHPAD 0xBE
struct scratchpad_struct {
 int16_t temperature;
 int16_t temperatureAlarm;
 uint8_t config;
 int8_t reserved[3];
 uint8_t crc;    };

int main(void) {
 lcdInit();
 lcdInitPrintf();
 oneWireReset();
 oneWireSendByte(OW_SKIP_ROM);
 oneWireSendByte(DS18B20_CONVERT_T);
 _delay_ms(750);
 oneWireReset();
 oneWireSendByte(OW_SKIP_ROM);
 oneWireSendByte(DS18B20_READ_SCRATCHPAD);
 //deklaracja zmiennej strukturalnej
 struct scratchpad_struct scratchpad;
 //stworzenie wskaźnika pokazującego na zmienną
strukturalną
 uint8_t * byte = (uint8_t *) &scratchpad;
 //odczyt scratchpada i zapis do zmiennej strukturalnej
 //przez wskaźnik bajtowy
 for (uint8_t i = 0; i < sizeof(scratchpad); i++)
  byte[i] = oneWireReceiveByte();
 //obliczenie temperatury
 float t = scratchpad.temperature / 16.0;
 printf("T=%.3f", t);  // a teraz weryfikacja CRC:
 uint8_t crc = 0;
 byte = (uint8_t *) &scratchpad;
 for (uint8_t i = 0; i < sizeof(scratchpad) - 1; i++)
  crc = oneWireComputeCRC8(byte[i], crc);
 lcdGotoXY(0, 1);
 printf("CRC: %s", crc == scratchpad.crc ? "OK" : "E");
 while(1) { }  }
```

Pomiar temperatury

DS18B20 to najpopularniejszy termometr cyfrowy. Pozwala on łatwo wykonać pomiar temperatury, bez potrzeby kalibracji i przetwarzania wartości analogowych na cyfrowe. Aby odczytać z niego temperaturę, musimy wykonać następujące czynności:

- Zresetować linię
- Wysłać komendę Skip ROM, jeśli jest to jedyny układ, lub też Match ROM a następnie identyfikator, jeśli układów jest więcej
- Wysłać komendę Convert T (44h)
- Odczekać 750 ms
- Zresetować linię
- Analogicznie jak wcześniej zaadresować układ
- Wysłać komendę Read Scratchpad (BEh)

Scratchpad to 9-bajtowa pamięć przechowująca wynik pomiaru, ustawienia temperatury alarmowej, pozostałe ustawienia i sumę kontrolną. Organizacja scratchpada przedstawiona jest w **tabeli 1**. Master nie musi odczytywać wszystkich bajtów, może np. odczytać tylko pierwsze dwa.

Rejestr konfiguracyjny pozwala modyfikować rozdzielczość termometru za pomocą dwóch bitów, zgodnie z **tabelą 2**. Im mniejsza rozdzielczość, tym pomiar temperatury trwa krócej.

Temperatura przechowywana jest w dwóch bajtach, a liczby ujemne zapisywane są w kodzie uzupełnień do 2, znanym nam z ADC. Można więc rozpatrywać bajty temperatury jako typ int16_t, ale cztery najmłodsze bity stanowią część ułamkową. Trzeba więc wartość jeszcze podzielić przez 16 (2^4). Popatrzmy na przykładowy kod termometru – **listing 7.**

Po inicjalizacji LCD wykonujemy pomiar temperatury. Zakładamy, że podłączony jest tylko jeden układ 1-Wire, stąd użycie komendy Skip ROM. Następnie wydawana jest komenda rozpoczęcia pomiaru. Po odczekaniu odpowiedniego czasu odczytywany jest scratchpad. Tym razem odczytane dane nie trafiają do tablicy typu uint8_t, ale do zmiennej strukturalnej, o polach odpowiadających strukturze scratchpada. Dzięki temu nie musimy operować na pojedynczych bajtach, ale mamy wydzielone logicznie pola.

Na początku kodu definiowany jest typ scratchpad_struct, który opisuje, jakie pola mają się znaleźć w zmiennej przechowującej scratchpad. Definicja bazuje na tabeli 1. Na podstawie zdefiniowanego typu deklarujemy następnie zmienną o nazwie scratchpad. Chcemy ją wypełnić danymi zwracanymi przez DS18B20. Ponieważ funkcja oneWireReceiveByte() zwraca pojedyncze bajty, musimy jakoś wypełnić nimi naszą zmienną strukturalną scratchpad. W tym celu tworzo-

na jest zmienna wskaźnikowa pokazująca właśnie na typ bajtowy uint8_t. Wskaźnik ten inicjalizujemy adresem naszej zmiennej strukturalnej. Wykonując operacje z wykorzystaniem tego wskaźnika, będziemy operować na obszarze pamięci, w którym jest zmienna scratchpad. Ponieważ wskaźnik pokazuje na dane typu uint8_t, a zmienna jest typu scratchpad_struct, w inicjalizacji wykonywane jest tzw. rzutowanie, czyli określenie docelowego typu wewnątrz nawiasu. Dzięki temu kompilator nie będzie zgłaszał niezgodności typów.

Do danych wskazywanych przez wskaźnik możemy odwoływać się jak do elementów tablicy. W ten sposób odbieranie bajtów z DS18B20 możemy napisać tak, jakbyśmy odbierane dane zapisywali do kolejnych komórek zmiennej tablicowej o nazwie byte. Odbierane dane będą trafiały do kolejnych bajtów składających się na zmienną scratchpad. Takie mazanie po pamięci jest wielką zaletą języka C, ale wymaga ostrożności, aby niechcący czegoś nie nadpisać. Dlatego nasza pętla for() wykonuje się tyle razy, ile bajtów ma zmienna scratchpad.

Gdy dane są odebrane, możemy obliczyć temperaturę. Wystarczy wartość pola temperature podzielić przez 16. Wynik wyświetlany jest na LCD. Tak jak w poprzednich lekcjach, trzeba oczywiście pamiętać o ustawieniu flag linkera, aby działało wyświetlanie liczb zmiennoprzecinkowych. Na koniec sprawdzana jest poprawność sumy kontrolnej.

Zadania

W ramach oswajania się z protokołem 1-Wire oraz układem DS18B20 proponuję następujące ćwiczenia:

1. Wyświetlanie na bieżąco liczby podłączonych układów
2. Termometr z wyświetlaczem LED
3. Termometr z funkcją alarmu
4. Termometr o rozdzielczości 9 bitów. Zmianę rozdzielczości wykonujemy, modyfikując bajt 4 w scratchpadzie. Komenda Write Scratchpad (4Eh) modyfikuje bajty 2–4, trzeba więc wysłać najpierw dwa dowolne bajty, a następnie bajt konfiguracyjny, z bitami 5 i 6 ustawionymi według tabeli 2.
5. Zamiast czekać ustalony okres po rozpoczęciu pomiaru, układ niezasilany pasożytniczo można odpytywać o zakończenie pomiaru poprzez odczytywanie pojedynczego bitu. Odczyt bitu 1 będzie oznaczał koniec konwersji. Napisać kod, który będzie wykrywał koniec pomiaru w ten właśnie sposób.

Bajt	Opis
0	Temperatura (młodszy bajt)
1	Temperatura (starszy bajt)
2	Temperatura alarmowa (młodszy bajt)
3	Temperatura alarmowa (starszy bajt)
4	Konfiguracja
5	Zarezerwowany (FFh)
6	Zarezerwowany
7	Zarezerwowany (10h)
8	CRC

Tabela 1

Bit 6	Bit 5	Rozdzielczość	Czas pomiaru
0	0	9 bitów	94 ms
0	1	10 bitów	188 ms
1	0	11 bitów	375 ms
1	1	12 bitów	750 ms

Tabela 2

W materiałach dodatkowych znajduje się projekt miernika pojemności akumulatorków, projekt termometru i pozostałe listingi. Zachęcam też do zapoznania się z kartą katalogową DS18B20.

Grzegorz Niemirowski
grzegorz@grzegorz.net