Podstawowe parametry:
- zakres pomiarowy temperatury: –40...125°C
- rozdzielczość pomiaru temperatury: 0,5°C
- dokładność pomiaru temperatury: ±2°C
- obsługiwane rozkazy: SEARCH_ROM, READ_ROM, MATCH_ROM, SKIP_ROM, CONVERT_T oraz READ_SCRATCHPAD,
- napięcie zasilania: 2,7...5 V, pobierany prąd: 6 mA.

Dodatkowe materiały do pobrania ze strony www.ulubionykiosk.pl/media

AVT5952	eT – wielokanałowy, bezprzewodowy system pomiaru temperatury (EP 9/2022)
AVT5949	Energooszczędny termometr LED (EP 8/2022)
AVT5892	Energooszczędny termometr z kalibracją (EP 10/2021)
AVT5635	Bezprzewodowy, energooszczędny system pomiaru temperatury (EP 8-9/2018)
AVT1999	2-kanałowy termometr MIN-MAX z alarmem (EP 8/2018)
AVT5623	4-kanałowy termometr z interfejsem Wi-Fi (EP 4/2018)
AVT5566	THPStation – rozbudowany termometr z Wi-Fi (EP 1/2017)
AVT5535	Termometr 2-kanałowy z interfejsem Bluetooth (EP 4/2016)
AVT5518	Termometr bezprzewodowy (EP 11/2015)
AVT1863	Termometr z interfejsem Bluetooth (EP 8/2015)
AVT1790	Termometr XXL (EP 2/2014)
AVT5489	8-kanałowy termometr z alarmem i wyświetlaczem LCD (EP 11/2013)
AVT5420	Wielopunktowy termometr z rejestracją (EP 10/2013)
AVT1734	Termometr do wędzarni (EP 4/2013)
AVT5373	Tlogger – rejestrator temperatury (EP 12/2012)
AVT1705	Moduł do pomiaru temperatury z interfejsem RS485 (EP 9/2012)

W ofercie AVT*
AVT5965

* **Uwaga!** Elektroniczne zestawy do samodzielnego montażu. Wymagana umiejętność lutowania! Podstawową wersją zestawu jest wersja **[B]** nazywana potocznie KIT-em (z ang. zestaw). Zestaw w wersji **[B]** zawiera elementy elektroniczne (w tym **[UK]** – jeśli występuje w projekcie), które należy samodzielnie wlutować w dołączoną płytkę drukowaną (PCB). Wykaz elementów znajduje się w dokumentacji, która jest podlinkowana w opisie kitu. Mając na uwadze różne potrzeby naszych klientów, oferujemy dodatkowe wersje:
- **wersja [C]** – zmontowany, uruchomiony i przetestowany zestaw **[B]** (elementy wlutowane w płytkę PCB),
- **wersja [A]** – płytka drukowana bez elementów i dokumentacji.

Kity, w których występuje układ scalony wymagający zaprogramowania, mają następujące dodatkowe wersje:
- **wersja [A+]** – płytka drukowana **[A]** + zaprogramowany układ **[UK]** i dokumentacja,
- **wersja [UK]** – zaprogramowany układ.

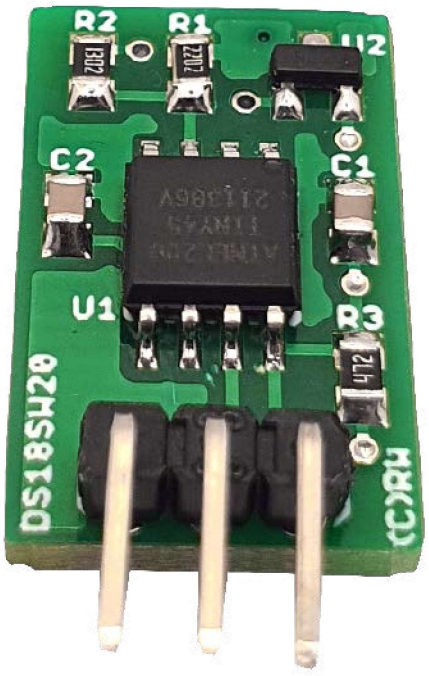
Nie każdy zestaw AVT występuje we wszystkich wersjach! Każda wersja ma załączony ten sam plik PDF! Podczas składania zamówienia upewnij się, którą wersję zamawiasz! **http://sklep.avt.pl**

W przypadku braku dostępności na stronie sklepu osoby zainteresowane zakupem płytek drukowanych (PCB) prosimy o kontakt via e-mail: **kity@avt.pl**.

DS18SW20 – emulator czujnika temperatury DS1820

Przykład programowej realizacji urządzenia 1-Wire slave (1)

Pomysł na ten projekt narodził się w dość nietypowych okolicznościach. Otóż, w jednym z urządzeń dostarczonych przez klienta byłem zmuszony wymienić uszkodzony termometr scalony pod postacią bardzo popularnego układu typu DS1820 firmy Maxim Integrated (dawniej Dallas Semiconductor) pracującego pod kontrolą interfejsu 1-Wire. Ku mojemu wielkiemu zaskoczeniu dość szybko okazało się, że zdobycie tego rodzaju, jakby nie patrzeć prostego, elementu graniczy z cudem, a ceny u dystrybutorów sięgają astronomicznych kwot rzędu 50 do nawet 120 zł za sztukę (!).



To niewyobrażalne, aby element tego typu osiągał tak kuriozalne ceny, gdy jeszcze jakiś czas temu można było go kupić za równowartość 1$. Rozumiem bieżące problemy z dostępnością jakichkolwiek półprzewodników (a zwłaszcza mikrokontrolerów) lecz nic nie uzasadnia tak absurdalnej sytuacji. Na pocieszenie można dodać, że znacznie przystępniejsze, powiedziałbym normalne, ceny ma inny termometr tego producenta a mianowicie DS18B20. Niestety oba wspomniane peryferia nie są bezpośrednimi odpowiednikami, gdyż różnią się dokładnością, a co najważniejsze mają inną organizację wbudowanej pamięci (*scratchpad*) przechowującej wartość mierzonej temperatury a co za tym idzie nie można ich stosować zamiennie. I właśnie wtedy do głowy wpadł mi dość oryginalny, jak mi się wydaje, pomysł skonstruowania własnego termometru DS1820, którego projekt nazwałem dość sugestywnie, a mianowicie DS18SW20 (SW od słowa *software*).

Jednak zaznaczam, że nie będę realizował całej dostępnej funkcjonalności termometru DS1820 (choć zapewne z 90%) a skupię się wyłącznie na tych możliwościach, które zazwyczaj potrzebne są w systemach mikroprocesorowych. Zaprezentuję jednak szczegółowy opis implementacji w języku C, w związku z czym zainteresowani Czytelnicy bez problemu będą mogli dodać brakujące funkcje.

Magistrala 1-Wire

Zanim jednak przejdę do opisu samego urządzenia, jak i szczegółów implementacyjnych, nie sposób choćby skrótowo nie przypomnieć informacji na temat bardzo interesującej magistrali 1-Wire, tym razem jednak z punktu widzenia układu podrzędnego typu Slave. Tak jak wspomniano wcześniej, komunikacja na magistrali 1-Wire odbywa się przy udziale wyłącznie jednego przewodu (stąd nazwa interfejsu) oznaczonego jako DQ, który może jednocześnie pełnić rolę przewodu zasilającego w konfiguracji tzw. zasilania pasożytniczego (de facto nieobsługiwanego przez nasze urządzenie DS18SW20). W przypadku magistrali 1-Wire, tak jak w przypadku większości interfejsów szeregowych, transmisja przebiega w konfiguracji Master↔Slave.

Układ nadrzędny (Master) steruje wyszukiwaniem i adresowaniem układów podrzędnych (Slave), steruje przepływem danych oraz

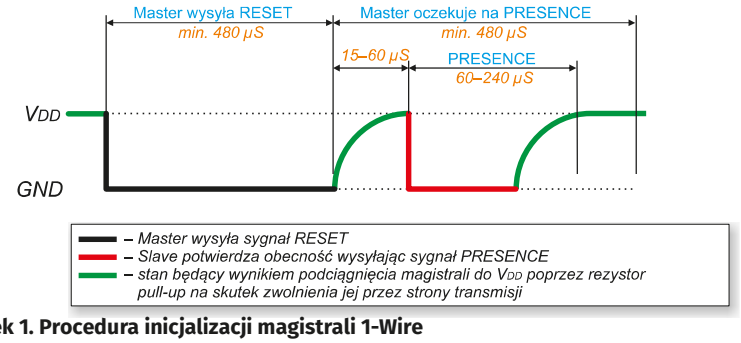
generuje sygnał zegarowy (inicjuje wysyłanie i odbieranie danych). Dane przesyłane są synchronicznie z prędkością do 16,3 kbps w trybie standard oraz do 115 kbps w trybie *overdrive*. Należy szczególnie podkreślić, że przesłanie każdego bitu informacji niezależnie od kierunku transmisji inicjowane jest wyłącznie przez układ Master za pomocą wygenerowania opadającego zbocza sygnału (ściągnięcie magistrali do logicznego „0” przez czas z zakresu 1…5 µs). Po wystąpieniu takiego zbocza sygnału układ Slave podejmuje różne działania, których scenariusz zależy od oczekiwanego kierunku transmisji. Tego typu organizacja protokołu transmisji zapewnia prawidłową synchronizację przesyłanych danych bez potrzeby stosowania dodatkowych linii sterujących.

Minimalny czas trwania pojedynczego bitu jest ściśle określony i wynosi 60 µs+1 µs na tak zwany czas odtworzenia zasilania (*recovery time*). Wyznacza on maksymalną prędkość transmisji w trybie standard (1/61 µs=16,3 kbps). Co ważne, każde z urządzeń podłączonych do magistrali musi mieć wyjście typu otwarty dren lub otwarty kolektor, a linia danych połączona jest do zasilania przez rezystor podciągający o typowej wartości 4,7 kΩ, co w stanie bezczynności powoduje utrzymywanie się stanu wysokiego na tej linii zapewniającego zasilanie urządzeń podrzędnych (jeśli pracują w trybie zasilania pasożytniczego).

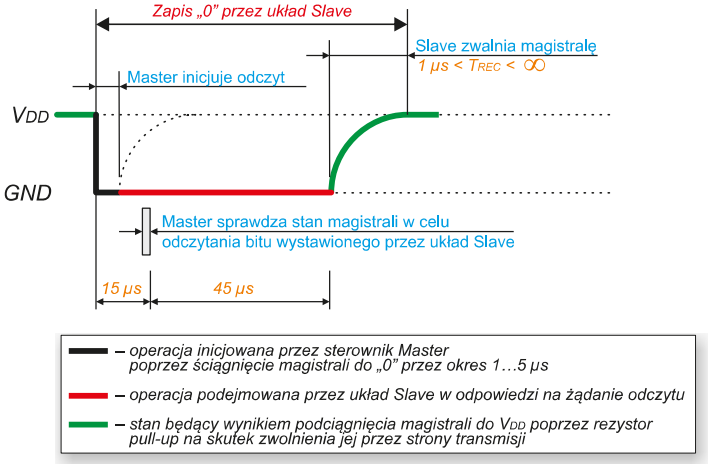
Sama magistrala nie ma ustalonego formatu danych a sposób przesyłania informacji zależy od konfiguracji i właściwości układów podrzędnych. Przesyłane słowa są zawsze jednobajtowe a jako pierwszy transmitowany jest bit mniej znaczący. Dodatkową i jedną z najważniejszych cech urządzeń z interfejsem 1-Wire, o czym wspomniano na wstępie, odróżniającą je jednocześnie np. od urządzeń standardu I²C, jest unikatowy, ośmiobajtowy adres zapisany w pamięci ROM peryferium. Adres ten jest niepowtarzalny i właściwy tylko i wyłącznie pojedynczemu układowi scalonemu (dla elementów produkowanych przez firmę Maxim/Dallas zapisywany jest na etapie produkcji). Najmniej znaczący bajt tego adresu zawiera kod rodziny układów (Family code), kolejne 6 bajtów zawiera unikatowy kod konkretnego egzemplarza (właściwy adresu układu) a najbardziej znaczący bajt zawiera sumę kontrolną CRC8 (*Cyclic Redundancy Check*). Suma ta wyliczana jest na podstawie poprzednich siedmiu bajtów i jest ustalana na etapie produkcji (służy do kontroli poprawności transmisji).

Protokół transmisji danych interfejsu 1-Wire definiuje kilka, podstawowych sygnałów sterujących i stanów pracy magistrali:

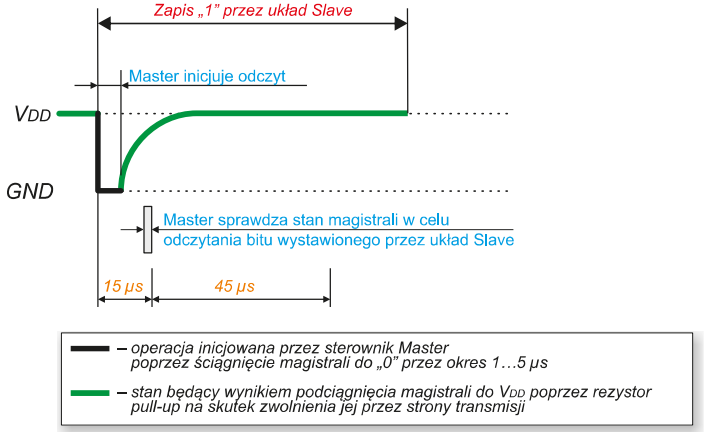
- sygnał *Reset*, wysyłany przez układ Master, będący żądaniem zgłoszenia się układów Slave,



Rysunek 1. Procedura inicjalizacji magistrali 1-Wire



Rysunek 2. Sekwencja sygnałów sterujących obrazująca operację zapisu logicznego „0” wykonywaną przez układ Slave a inicjowaną przez układ Master



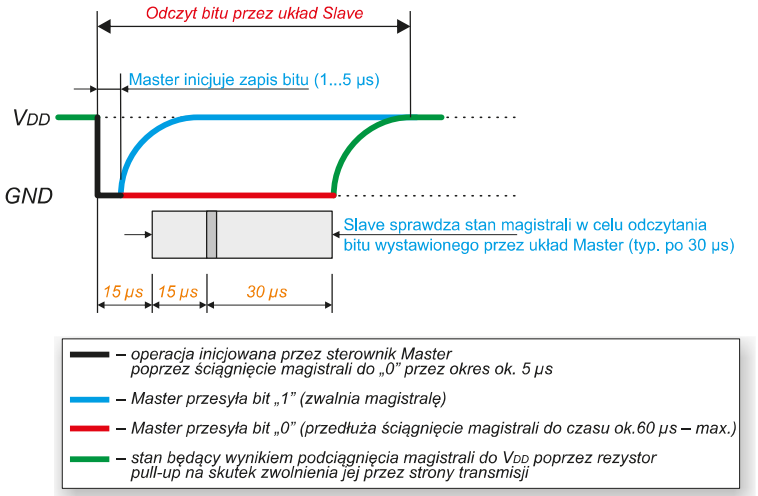
Rysunek 3. Sekwencja sygnałów sterujących obrazująca operację zapisu logicznej „1” wykonywaną przez układ Slave a inicjowaną przez układ Master

- sygnał *Presence*, wysyłany przez układy Slave, będący potwierdzeniem obecności tych układów na magistrali danych,
- zapis logicznej „1” i „0”,
- odczyt logicznej „1” i „0”.

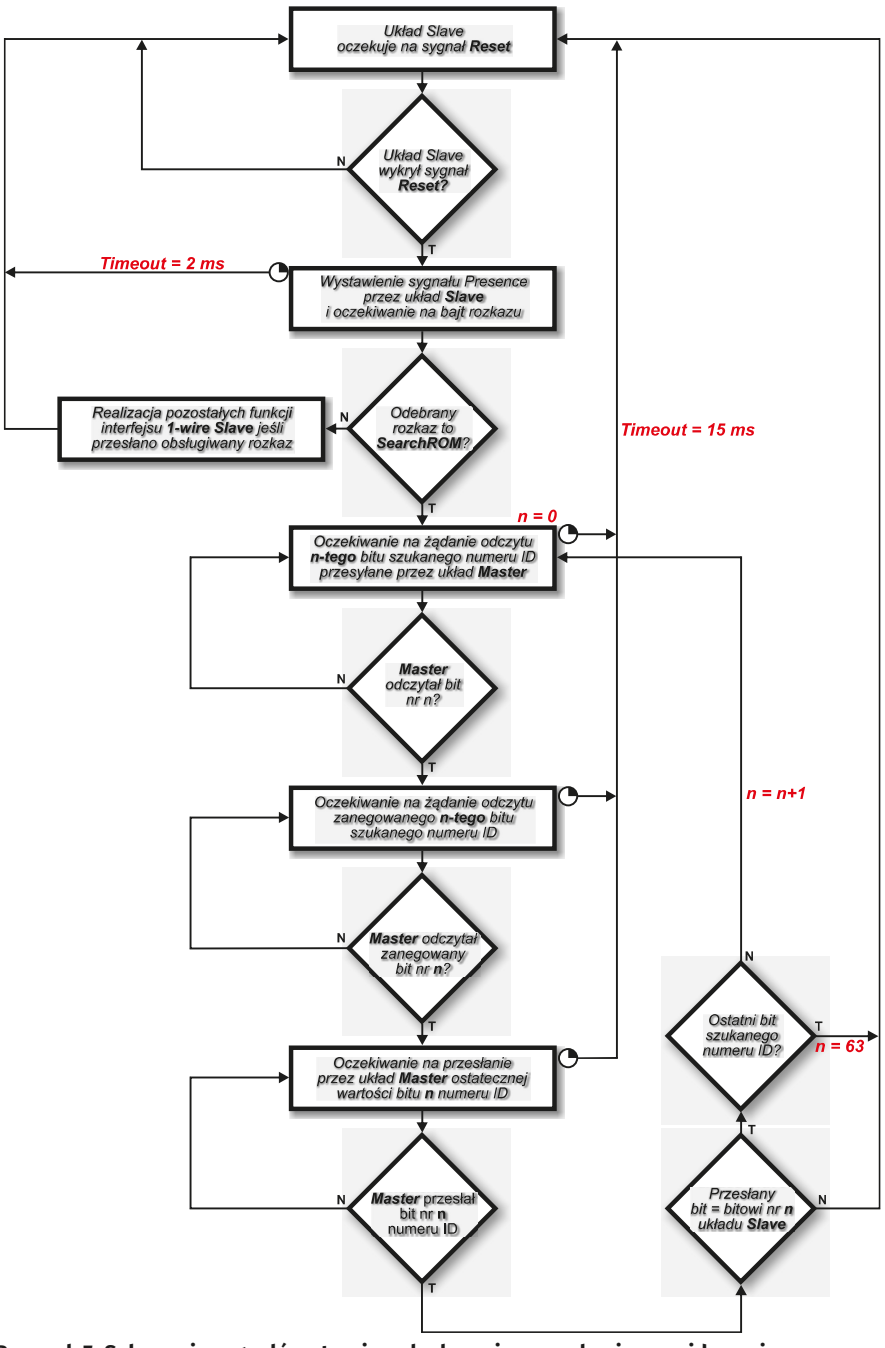
Najlepszym sposobem na zrozumienie zależności czasowych dla poszczególnych stanów pracy magistrali jest zobrazowanie ich na rysunkach pokazujących sekwencje tych sygnałów widziane z perspektywy układu Slave. Na **rysunku 1** pokazano sekwencję inicjalizacji magistrali 1-Wire, która to, jak wspomniano wcześniej, umożliwia układowi Master wykrycie podłączonych do magistrali układów Slave. Sekwencja ta rozpoczyna się poprzez wysłanie przez układ Master sygnału *Reset* (ściągnięcie magistrali do masy przez czas 480 µs) i po odczekaniu czasu 15…60 µs, odpowiedzią układów Slave poprzez wysłanie sygnału *Presence* (ściągnięcie magistrali do masy przez czas 60…240 µs).

Na **rysunkach 2 i 3** pokazano z kolei sekwencje sygnałów sterujących obrazujące operację zapisu danych (logicznego „0” i „1”) przez układ Slave na magistralę 1-Wire a będącą wynikiem żądania odczytu ze strony układu Master (ściągnięcie magistrali do masy przez czas 1…5 µs). Dalej, na **rysunku 4** pokazano sekwencję sygnałów sterujących obrazującą operacje odczytu danych wykonywaną przez układ Slave a będącą wynikiem zapisu tychże danych dokonanego przez układ Master (jak wcześniej, inicjowaną poprzez ściągnięcie magistrali do masy przez czas 1…5 µs).

Na koniec, na **rysunku 5**, zaprezentowany jest graf funkcjonalny bodajże najciekawszego mechanizmu charakterystycznego wyłącznie dla magistrali tego typu a przeznaczonego do wyszukania adresów wszystkich układów do niej przyłączonych nazywany SEARCH ROM. Należy bowiem pamiętać, że układ Master nie musi znać adresów przyłączonych do magistrali układów, w związku z czym musi istnieć pewny mechanizm na ich ustalenie. Operacja taka inicjowana jest poprzez wysłanie przez układ Master rozkazu 0xF0 (zwanego SEARCH ROM). Następnie każdy układ Slave wystawia na magistralę wartość pierwszego, najmniej znaczącego bitu swojego numeru ID (oczywiście zapis inicjowany jest zawsze przez układ Master). Mając na uwadze specyfikę interfejsu 1-Wire, polegającą na tym, że wszystkie układy podłączone są do tej samej magistrali danych należy pamiętać, że na odebraną komendę przeszukiwania odpowiedzą dokładnie w tym samym czasie wszystkie układy Slave w związku, z czym rezultat wystawienia bitów docierający do układu Master jest iloczynem logicznym stanów wyjść wszystkich tych układów (tzw. *wired and*). Kolejnym krokiem jest wystawianie przez układy Slave zanegowanego, pierwszego bitu swojego numeru ID (oczywiście jak zwykle, na żądanie układu Master). W tym momencie układ Master decyduje o przyjętej wartości pierwszego bitu adresu i wystawia go na magistralę, a układy Slave odczytują go. Jeśli przesłana przez układ Master wartość (w tym trzecim kroku) pierwszego bitu numeru ID jest zgodna z rzeczywistą wartością pierwszego, najmniej znaczącego bitu numeru ID wybranego układu Slave, układ ten kontynuuje proces aż do przebrnięcia przez wszystkie 64 bity numeru. Jeśli taka zgodność nie występuje, układ Slave pozostaje nieaktywny aż do następnego sygnału Reset i żądania wyszukiwania numerów ID (rozkazu 0xF0). W ten prosty sposób układ Master, po przejściu przez każde 64 bity numeru ID, dysponuje każdorazowo kolejnym, znalezionym numerem ID układu pracującego na magistrali 1-Wire. Oczywiście opis ten pokazano z punktu widzenia układu Slave, gdyż algorytm dla układu Master jest nieco bardziej skomplikowany (bazuje na algorytmie tzw. drzewa binarnego).



Rysunek 4. Sekwencja sygnałów sterujących obrazująca operację odczytu stanu magistrali 1-Wire wykonywaną przez układ Slave a inicjowaną przez układ Master



Rysunek 5. Sekwencja sygnałów sterujących obrazująca mechanizm znajdowania numerów ID układów Slave (z punktu widzenia tychże układów)

Budowa i działanie

W tym miejscu dysponujemy już niezbędną wiedzą z zakresu protokołu 1-Wire przejdźmy zatem do schematu ideowego naszego urządzenia, który pokazano na **rysunku 6**. Jak widać, zaprojektowano bardzo prosty, wręcz trywialny, system mikroprocesorowy, którego sercem jest niewielki mikrokontroler ATtiny25 taktowany wewnętrznym oscylatorem 8 MHz odpowiedzialny za realizację całej, założonej funkcjonalności.

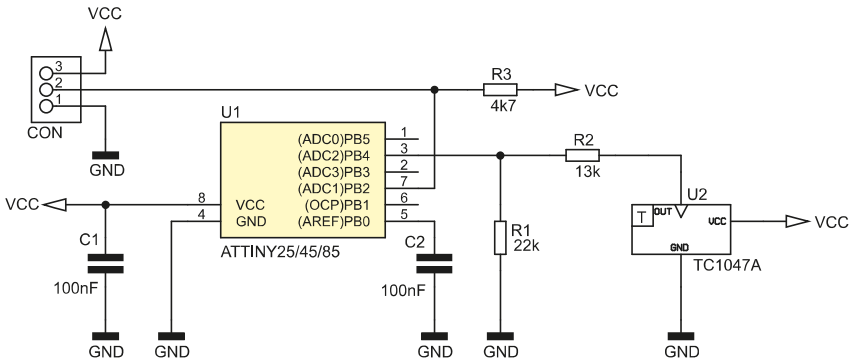
Wykaz elementów, kupuj na stronie sklep.avt.pl (Warszawa, ul. Leszczynowa 11, tel. +48222578451, e-mail: handlowy@avt.pl)

Rezystory: (SMD0805)
R1: 22 kΩ 1%
R2: 13 kΩ 1%
R3: 4.7 kΩ

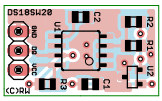
Kondensatory: (SMD0805)
C1, C2: 100 nF ceramiczny X7R

Półprzewodniki:
U1: ATtiny25/45/85 (SOIC-8)
U2: TC1047A (SOT-23)

Pozostałe:
CON: złącze GOLDPIN kątowe 3×1 pin



Rysunek 6. Schemat ideowy urządzenia DS18SW20



Rysunek 7. Schemat montażowy urządzenia DS18SW20

Jako element pomiarowy (termometr) zastosowano scalony przetwornik temperatura-napięcie pod postacią układu scalonego TC1047A produkcji Microchip cechujący się doskonałą liniowością (z nachyleniem 10 mV/°C) oraz akceptowalną dokładnością na poziomie ±2°C. Nie jest to co prawda dokładność emulowanego termometru typu DS1820 (lub DS18S20), ale moim zdaniem jest jak najbardziej akceptowalna, zaś sam układ jest łatwo dostępny i niedrogi. Co oczywiste, do odczytu wskazań termometru użyto wbudowanego w strukturę mikrokontrolera 10-bitowego przetwornika ADC (kanał ADC2) oraz wewnętrznego napięcia odniesienia o wartości 1,1 V. Odczytywane 10-bitowe dane z przetwornika ADC przeskalowano w taki sposób, aby uzyskać rozdzielczość pomiaru równą 0,5°C oraz zakres mierzonych temperatur rzędu –40…+125°C (jak dla emulowanego termometru). Jako, że poziom napięć wyjściowych układu TC1047A (100…1750 mV) dla pełnego zakresu mierzonych temperatur nie mieści się w zakresie dopuszczalnych napięć wejściowych przetwornika ADC (w przypadku korzystania z wewnętrznej referencji 1,1 V) zastosowano prosty dzielnik napięcia pod postacią rezystorów R1/R2 (koniecznie dokładnych – 1%) niwelujący tą niedogodność.

Montaż i uruchomienie

W tym momencie przejdźmy do szczegółów dotyczących montażu układu. Schemat płytki PCB urządzenia DS18SW20 pokazano na **rysunku 7**. Jak widać zaprojektowano bardzo kompaktowy (12×20 mm) dwustronny obwód drukowany z wyłącznym montażem elementów SMD. Montaż urządzenia rozpoczynamy od przylutowania półprzewodników (o nieproblematycznych obudowach), następnie lutujemy elementy bierne a na samym końcu złącze GOLDPIN. Poprawnie zmontowane i zaprogramowane urządzenie nie wymaga żadnego uruchamiania i powinno działać tuż po podłączeniu napięcia zasilającego.

Jedynym procesem, jaki możemy wykonać jest zaprogramowanie opcjonalnego adresu urządzenia Slave w pamięci EEPROM mikrokontrolera. W przypadku problemów z dostępnością przetwornika TC1047A u dystrybutorów elektroniki bez wahania możemy go zamówić na stronie producenta układu (z resztą w niższej cenie, niż u dystrybutorów) lub otrzymać za darmo (i to kilka sztuk) w ramach programu bezpłatnych próbek (sample) z czego skorzystałem podczas montażu urządzenia.

Tyle w kwestiach sprzętowych, których nie ma nazbyt wiele, gdyż cała magia dzieje się w oprogramowaniu. W drugiej części artykułu, która ukaże się w kolejnym wydaniu EP, dokładnie omówimy najważniejsze fragmenty kodu programu sterującego urządzeniem.

Robert Wołgajew, EP

Podstawowe parametry:
- zakres pomiarowy temperatury: –40...125°C
- rozdzielczość pomiaru temperatury: 0,5°C
- dokładność pomiaru temperatury: ±2°C
- obsługiwane rozkazy: SEARCH_ROM, READ_ROM, MATCH_ROM, SKIP_ROM, CONVERT_T oraz READ_SCRATCHPAD,
- napięcie zasilania: 2,7...5 V, pobierany prąd: 6 mA.

Dodatkowe materiały do pobrania ze strony www.ulubionykiosk.pl/media

AVT5952	eT – wielokanałowy, bezprzewodowy system pomiaru temperatury (EP 9/2022)
AVT5949	Energooszczędny termometr LED (EP 8/2022)
AVT5892	Energooszczędny termometr z kalibracją (EP 10/2021)
AVT5635	Bezprzewodowy, energooszczędny system pomiaru temperatury (EP 8-9/2018)
AVT1999	2-kanałowy termometr MIN-MAX z alarmem (EP 8/2018)
AVT5623	4-kanałowy termometr z interfejsem Wi-Fi (EP 4/2018)
AVT5566	THPStation – rozbudowany termometr z Wi-Fi (EP 1/2017)
AVT5535	Termometr 2-kanałowy z interfejsem Bluetooth (EP 4/2016)
AVT5518	Termometr bezprzewodowy (EP 11/2015)
AVT1863	Termometr z interfejsem Bluetooth (EP 8/2015)
AVT1790	Termometr XXL (EP 2/2014)
AVT5489	8-kanałowy termometr z alarmem i wyświetlaczem LCD (EP 11/2013)
AVT5420	Wielopunktowy termometr z rejestracją (EP 10/2013)
AVT1734	Termometr do wędzarni (EP 4/2013)
AVT5373	Tlogger – rejestrator temperatury (EP 12/2012)
AVT1705	Moduł do pomiaru temperatury z interfejsem RS485 (EP 9/2012)

W ofercie AVT* **AVT5965**

* **Uwaga!** Elektroniczne zestawy do samodzielnego montażu. Wymagana umiejętność lutowania! Podstawową wersją zestawu jest wersja **[B]** nazywana potocznie KIT-em (z ang. zestaw). Zestaw w wersji **[B]** zawiera elementy elektroniczne (w tym **[UK]** – jeśli występuje w projekcie), które należy samodzielnie wlutować w dołączoną płytkę drukowaną (PCB). Wykaz elementów znajduje się w dokumentacji, która jest podlinkowana w opisie kitu. Mając na uwadze różne potrzeby naszych klientów, oferujemy dodatkowe wersje:
- **wersja [C]** – zmontowany, uruchomiony i przetestowany zestaw **[B]** (elementy wlutowane w płytkę PCB),
- **wersja [A]** – płytka drukowana bez elementów i dokumentacji.

Kity, w których występuje układ scalony wymagający zaprogramowania, mają następujące dodatkowe wersje:
- **wersja [A+]** – płytka drukowana **[A]** + zaprogramowany układ **[UK]** i dokumentacja,
- **wersja [UK]** – zaprogramowany układ.

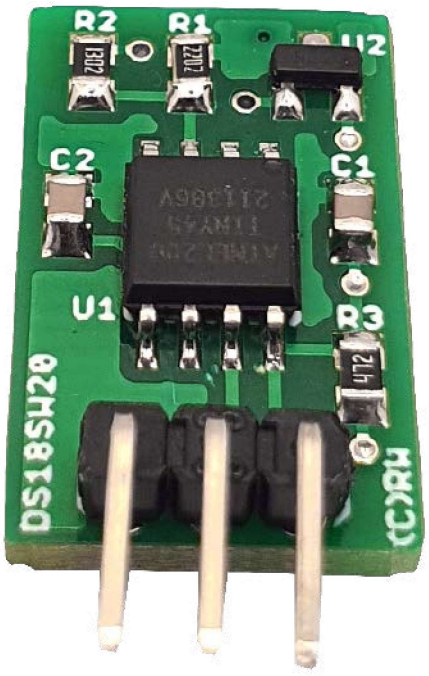
Nie każdy zestaw AVT występuje we wszystkich wersjach! Każda wersja ma załączony ten sam plik PDF! Podczas składania zamówienia upewnij się, którą wersję zamawiasz! http://sklep.avt.pl

W przypadku braku dostępności na stronie sklepu osoby zainteresowane zakupem płytek drukowanych (PCB) prosimy o kontakt via e-mail: kity@avt.pl.

DS18SW20 – emulator czujnika temperatury DS1820

Przykład programowej realizacji urządzenia 1-Wire slave (2)

Realizacja urządzenia 1-Wire slave wymaga nieszablonowego podejścia. W przeważającej większości projektów mikrokontroler działa jako urządzenie nadrzędne – inicjuje każdy stan i steruje przebiegiem transmisji danych. Taka konfiguracja była wiele razy dokładnie opisywana na łamach EP. Natomiast praca mikrokontrolera w roli układu slave, to relatywnie rzadko spotykane rozwiązanie. Dlatego po ogólnym omówieniu interfejsu 1-Wire w pierwszej części artykułu, przechodzimy do szczegółowego opisu kodu programu emulatora.



Program sterujący

Cała magia emulatora czujnika temperatury DS1820 dzieje się w oprogramowaniu urządzenia. Jak wiemy, każda operacja mająca miejsce na magistrali 1-Wire inicjowana jest przez układ Master poprzez ściągnięcie tejże magistrali do logicznego „0” (przez czas 1…5 µs). W związku z tym naturalnym sposobem na obsłużenie protokołu ze strony układu Slave jest wykorzystanie przerwania zewnętrznego (np. INT0) skonfigurowanego w ten sposób aby zachodziło przy opadającym zboczu sygnału. Użycie przerwania zewnętrznego jest o tyle niezbędne, że magistrala 1-Wire wymusza dość rygorystyczne wymogi czasowe, więc zastosowanie typowego portu I/O i tzw. *pollingu* jest niewystarczająca w przypadku, gdyby układ Slave miał wykonywać jeszcze inne czasowo istotne operacje niezwiązane z obsługą magistrali. Wykorzystanie przerwania zewnętrznego niesie za sobą tą dodatkową zaletę, że cała obsługa protokołu zostanie zamknięta w ramach jednej funkcji ISR a program główny może realizować inną, niezbędną z punktu widzenia konkretnej aplikacji, funkcjonalność.

Zadeklarujmy, zatem kilka podstawowych zmiennych globalnych: numer seryjny naszego urządzenia Slave, scratchpad (pamięć termometru), zmienną przechowującą aktualny stan procedury obsługi układu Slave oraz zmienną odpowiedzialną za żądanie pomiaru temperatury. Jak wiemy, zmienne globalne to w dużym uproszczeniu zło (z przymrużeniem oka), gdyż utrudniają optymalizację kodu, lecz w przypadku styku na poziomie program główny/funkcje narzędziowe a funkcje obsługi przerwań systemowych są niezbędną koniecznością. Specyfikację zmiennych globalnych naszego urządzenia pokazano na **listingu 1**. Dalej, na **listingu 2**, pokazano plik nagłówkowy programu obsługi aplikacji definiujący zarówno ustawienia sprzętowe, jak i wprowadzający niezbędne definicje stałych poprawiających czytelność kodu (jak i ułatwiających jego modyfikacje).

W związku z tym, że obsługa protokołu odbywa się w całości w ramach procedury obsługi przerwania INT0 (oraz jak się okaże, przerwania Timera0) funkcja główna naszego programu obsługi ogranicza się wyłącznie do konfiguracji niezbędnych peryferiów mikrokontrolera

Listing 1. Specyfikacja zmiennych globalnych urządzenia DS18SW20

```
//Status Slave'a 1-wire
volatile uint8_t Status;
//ID układu Slave. 8. bajt to CRC8
volatile uint8_t ID[8] = {'R', 'o', 'b', 'e', 'r', 't', 'W', 0x00};
//Scratchpad układu Slave. 1. i 2. bajt to LSB i MSB temperatury. 9. bajt to CRC8
volatile uint8_t Scratchpad[9] = {0xAA, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0x0C, 0x10, 0x00};
//Żądanie pomiaru temperatury
volatile uint8_t measureTemp;
```

Listing 2. Plik nagłówkowy programu obsługi aplikacji DS18SW20

```
//Definicje portu 1-wire
#define ONE_WIRE_DDR DDRB
#define ONE_WIRE_PIN PINB
#define ONE_WIRE_NR PB2  //INT0
#define ONE_WIRE_CLEAR ONE_WIRE_DDR |= (1<<ONE_WIRE_NR)
#define ONE_WIRE_RELEASE ONE_WIRE_DDR &= ~(1<<ONE_WIRE_NR)
#define ONE_WIRE_READ ((ONE_WIRE_PIN & (1<<ONE_WIRE_NR))>>ONE_WIRE_NR)

//Definicje dla Timera0
#define STOP_TIMER TCCR0B = 0x00
//Preskaler = 1024, 7812Hz
#define START_TIMER TCCR0B = (1<<CS02)|(1<<CS00)
#define RESET_TIMER TCNT0 = 0
#define TIMEOUT_2MS (256-15)
#define TIMEOUT_5MS (256-39)
#define TIMEOUT_15MS (256-117)

//Definicje statusów układu Slave
#define STATUS_IDLE 0x00
#define STATUS_WAITING_FOR_CMD 0x01
#define STATUS_SEARCH_ROM_ACTIVE 0x02
#define STATUS_READ_ROM_ACTIVE 0x03
#define STATUS_MATCH_ROM_ACTIVE 0x04
#define STATUS_SLAVE_SELECTED 0x05
#define STATUS_READ_SCRATCHPAD_ACTIVE 0x06

//Definicje komend 1-wire
#define CMD_SEARCH_ROM 0xF0
#define CMD_READ_ROM 0x33
#define CMD_MATCH_ROM 0x55
#define CMD_SKIP_ROM 0xCC
#define CMD_READ_SCRATCHPAD 0xBE
#define CMD_CONVERT_T 0x44

//Definicje dla mechanizmu wyszukiwania numeru ID
#define SEARCH_SEND_BYTE 0x00
#define SEARCH_SEND_NEGATED_BYTE 0x01
#define SEARCH_COMPARE_BYTE 0x02

//Definicje czasów 1wire – us
#define INTR_HANDLING_TIME 6
#define RESET_PULSE_MIN 3  //Takty zegara Timer0
#define PRESENCE_PULSE 240
#define SLAVE_SAMPLE_TIME (25-INTR_HANDLING_TIME)
#define SLAVE_ZERO_TIME 20
```

Listing 3 Funkcja główna programu obsługi aplikacji DS18SW20.

```
int main(void){
    //Redukcja poboru mocy przez wyłączenie modułów
    //(lub ich zegarów): TIMER1, USI
    PRR = (1<<PRTIM1)|(1<<PRUSI);
    //Wyłączenie komparatora analogowego
    //dla zmniejszenia poboru mocy
    ACSR = (1<<ACD);
    //Konfiguracja przetwornika ADC
    //Ref = 1.1V, wejście ADC2
    ADMUX = (1<<REFS1)|(1<<MUX1);
    //Konfiguracja i uruchomienie przerwania INT0
    //wyzwalane zboczem opadającym – obsługa 1wire
    MCUCR |= (1<<ISC01);
    GIMSK = (1<<INT0);
    //Uruchomienie przerwania od przepełnienia Timera0
    //obsługa timeout-ów
    TIMSK = (1<<TOIE0);
    //Odczytujemy numer ID lub zostajemy przy domyślnym
    readID();

    sei();
    while(1){
        //Obsługa żądania pomiaru temperatury
        if(measureTemp){
            measureTemp = 0;
            readADC();
        }
    }
}
```

i wykonywania pomiaru temperatury na żądanie użytkownika. Ciało tejże funkcji pokazano na **listingu 3**. Jak widać, i o czym nie wspomniano wcześniej, w programie obsługi używany jest także timer sprzętowy (w tym przypadku 8-bitowy Timer0) jak i przerwanie od jego przepełnienia (*TIMER0_OVF_vect*). Użycie tego peryferium konieczne jest w przypadku, gdyby układ Master z jakiś powodów wysłał niepełną sekwencję sygnałów sterujących (np. tylko sygnał *Reset* bez jakichkolwiek dalszych komend) co spowodowałoby zmianę stanu procedury obsługi magistrali 1-Wire na predefiniowaną wcześniej wartość *STATUS_WAITING_FOR_CMD* i oczekiwanie na rozkaz sterujący uniemożliwiając tym samym dalsze, poprawne funkcjonowanie algorytmu obsługi. Timer ten każdorazowo ustawiany jest w taki sposób aby po upłynięciu zadanego czasu (zależnego od oczekiwanych operacji po stronie układu Master) przywrócić spoczynkowy stan procedury obsługi (*STATUS_IDLE*) w przypadku błędów po stronie układu nadrzędnego.

Przejdźmy zatem do właściwego programu obsługi układu Slave magistrali 1-Wire. Jak wspomniano wcześniej, stanem spoczynkowym układu Slave jest oczekiwanie na sygnał *Reset* a po jego wykryciu, wystawienie sygnału *Presence* i oczekiwanie na rozkaz sterujący. Tą część algorytmu programu obsługi pokazana na **listingu 4** (o wykonaniu poszczególnych części programu obsługi decyduje wartość zmiennej Status). Po wykonaniu tych czynności, układ Slave oczekuje (przez czas 2 ms) na odebranie komendy sterującej i w zależności od jej rodzaju realizuje pozostałe funkcje typowe dla emulowanego układu typu DS1820.

Nasze urządzenie obsługuje 6 rodzajów komend sterujących: CMD_SEARCH_ROM, CMD_READ_ROM, CMD_MATCH_ROM, CMD_SKIP_ROM, CMD_CONVERT_T oraz CMD_READ_SCRATCHPAD. Odbiór komendy sterującej realizuje część programu obsługi pokazana na **listingu 5**. Następnie, w zależności od rodzaju odebranej komendy sterującej, realizowana jest odpowiednia część programu obsługi (o wykonaniu poszczególnych części programu obsługi jak zwykle decyduje wartość zmiennej Status). Rozkaz READ_ROM (czyli odczyt numeru seryjnego przez układ Master) realizuje część programu obsługi aplikacji pokazana na **listingu 6**. Rozkaz MATCH_ROM (czyli wysyłania przez układ Master numeru ID aby zaadresować naszego Slave-a) realizuje część programu obsługi aplikacji

Wykaz elementów, kupuj na stronie sklep.avt.pl (Warszawa, ul. Leszczynowa 11, tel. +48222578451, e-mail: handlowy@avt.pl)

Rezystory: (SMD0805)
R1: 22 kΩ 1%
R2: 13 kΩ 1%
R3: 4.7 kΩ

Kondensatory: (SMD0805)
C1, C2: 100 nF ceramiczny X7R

Półprzewodniki:
U1: ATtiny25/45/85 (SOIC-8)
U2: TC1047A (SOT-23)

Pozostałe:
CON: złącze GOLDPIN kątowe 3×1 pin

pokazana na **listingu 7**. Jak widać, przesłanie niepoprawnego adresu układu Slave powoduje dezaktywację naszego urządzenia (stan domyślny STATUS_IDLE wymuszający oczekiwanie na sygnał RESET), zaś przesłanie adresu zgodnego powoduje adresację urządzenia (STATUS_SLAVE_SELECTED) i oczekiwanie na przesłanie bajta realizowanej funkcjonalności (rozkazy CMD_CONVERT_T i CMD_READ_SCRATCHPAD). Skądinąd takie samo zachowanie powoduje przesłanie rozkazu CMD_SKIP_ROM (pokazane na **listingu 5**), który wymusza pominięcie adresacji konkretnego urządzenia i ma sens wyłącznie wtedy, gdy na magistrali 1-Wire znajduje się tylko jedno urządzenie typu Slave.

Dalej, na **listingu 8** pokazano fragment algorytmu programu obsługi odpowiedzialny za realizację funkcji CMD_SEARCH_ROM, czyli wyszukiwania przez układ Master numerów ID podłączonych do magistrali układów Slave. Tak jak wspomniano wcześniej poprawne zaadresowanie układu Slave powoduje przejście urządzenia w tryb oczekiwania (STATUS_SLAVE_SELECTED) na przesłanie bajta realizowanej funkcjonalności (rozkazy CMD_CONVERT_T i CMD_READ_SCRATCHPAD). Za ten etap programu obsługi odpowiada fragment algorytmu programu obsługi pokazany na **listingu 9**. Jak widać, odbiór rozkazu CMD_CONVERT_T powoduje ustawienie zmiennej globalnej *measureTemp* wymuszającej wykonanie pomiaru temperatury realizowane w pętli głównej aplikacji i powodujące wpisanie (atomowe) wyników pomiaru do tablicy Scratchpad, zaś odebranie rozkazu CMD_READ_SCRATCHPAD powoduje przejście programu obsługi do fragmentu odpowiedzialnego za wysłanie (na żądanie Mastera) przez układ Slave zawartości tablicy Scratchpad, przy czym liczba bajtów, jaka zostanie przesłana zależy wyłącznie od przebiegu transmisji inicjowanej przez układ nadrzędny (od 1 do 9 bajtów). Odczyt przez układ Master liczby bajtów mniejszej niż cała zawartość scratchpad-a (9 bajtów) powoduje wystąpienie timeout-u i powrót stanu urządzenia Slave do pozycji wyjściowej (STATUS_IDLE) i oczekiwanie na sygnał RESET.

Uważny Czytelnik zastanowi się z pewnością, dlaczego rozkazy CMD_CONVERT_T i CMD_READ_SCRATCHPAD nie są obsługiwane od razu po detekcji sygnału RESET i wysłaniu sygnału PRESENCE (listing 5). Otóż nie jest to możliwe, gdyż wykonanie tych rozkazów wymaga wcześniejszej adresacji układu

Ustawienia Fuse-bitów:
CKSEL3...0: 0010
SUT1...0: 10
CKDIV8: 1
CKOUT: 1
DWEN: 1
EESAVE: 0

Listing 4. Część algorytmu programu obsługi odpowiedzialna za detekcję sygnału RESET i wygenerowanie sygnału PRESENCE

```
//Gotowy na sygnał Reset – stan wyjściowy
case STATUS_IDLE:
//Sprawdzamy długość sygnału niskiego oczekując sygnału Reset
    RESET_TIMER;
    START_TIMER;
    while(!ONE_WIRE_READ);
    STOP_TIMER;
    if(TCNT0 >= RESET_PULSE_MIN) {
        //Wysyłamy sygnał Presence
        delay_us(15);
        ONE_WIRE_CLEAR;
        _delay_us(PRESENCE_PULSE);
        ONE_WIRE_RELEASE;
        //Przygotowanie zmiennych i zmiana statusu
        bitNr = Byte = 0;
        Status = STATUS_WAITING_FOR_CMD;
        //Uruchomienie Timera0 jako układu
        //odmierzającego timeout
        setTimeout(TIMEOUT_2MS);
    }
break;
```

Listing 5. Część algorytmu programu obsługi odpowiedzialna za odbiór komendy sterującej

```
//Gotowy na przyjęcie rozkazu – po wyemitowaniu sygnału Presence
case STATUS_WAITING_FOR_CMD:
    //Czekamy 30us aż znajdziemy się w środku
    //przedziału czasu na emisję bitu
    _delay_us(SLAVE_SAMPLE_TIME);
    //Odczyt bitu
    if(ONE_WIRE_READ) Byte |= (1<<bitNr);
    //Jeśli mamy kompletny bajt rozkazu to sprawdzamy
    // jaki to rozkaz i decydujemy o dalszym toku programu
    if(++bitNr == 8) {
        switch(Byte){
            case CMD_SEARCH_ROM:
                Status = STATUS_SEARCH_ROM_ACTIVE;
                //Uruchomienie Timera0
                //jako układu odmierzającego timeout
                setTimeout(TIMEOUT_15MS);
            break;

            case CMD_READ_ROM:
                Status = STATUS_READ_ROM_ACTIVE;
                //Uruchomienie Timera0
                //jako układu odmierzającego timeout
                setTimeout(TIMEOUT_5MS);
            break;

            case CMD_MATCH_ROM:
                Status = STATUS_MATCH_ROM_ACTIVE;
                //Wyzerowanie bufora danych
                for(uint8_t i = 0; i<8; i++) Buffer[i] = 0;
                //Uruchomienie Timera0
                //jako układu odmierzającego timeout
                setTimeout(TIMEOUT_5MS);
            break;

            case CMD_SKIP_ROM:
                Status = STATUS_SLAVE_SELECTED;
                //Uruchomienie Timera0
                //jako układu odmierzającego timeout
                setTimeout(TIMEOUT_2MS);
            break;

            default:
                //Nieobsługiwany rozkaz
                //wracamy do stanu wyjściowego
                Status = STATUS_IDLE;
                STOP_TIMER;
            break;
        }
        bitNr = byteNr = searchNr = Byte = 0;
    }
break;
```

Listing 6. Część algorytmu programu obsługi odpowiedzialna za realizację funkcji READ_ROM

```
//Jesteśmy w trybie STATUS_READ_ROM_ACTIVE,
//czyli odczytywania numeru ID przez układ Master
case STATUS_READ_ROM_ACTIVE:
    //Czekamy na zwolnienie magistrali przez Mastera
    //aby nie liczyć tego samego bitu jako kolejnego
    while(!ONE_WIRE_READ);
    //Wysyłamy kolejny bit adresu układu Slave.
    //Jeśli bit=0 to przedłużamy stan niski
    if(!(ID[byteNr] & (1<<bitNr))) {
        ONE_WIRE_CLEAR;
        _delay_us(SLAVE_ZERO_TIME);
        ONE_WIRE_RELEASE;
    }

    //Jeśli wysłaliśmy kompletny bajt adresu
    //to przechodzimy do kolejnego
    if(++bitNr == 8) {
        bitNr = 0;
        //Sprawdzamy, czy wysłaliśmy wszystkie bajty adresu.
        //Jeśli tak to przechodzimy do stanu spoczynkowego.
        if(++byteNr == 8) {
            Status = STATUS_IDLE;
            STOP_TIMER;
        }
    }
break;
```

Slave dokonywanej dzięki obsłudze rozkazów CMD_SKIP_ROM i CMD_MATCH_ROM. Na **listingu 10** pokazano fragment algorytmu programu obsługi odpowiedzialny za realizację funkcji READ_SCRATCHPAD. Na koniec tej tematyki wspomniana wcześniej funkcja obsługi przerwania od przepełnienia licznika Timer0 niezbędna z punktu widzenia odmierzania tzw. czasu timeout, której to ciało pokazano na **listingu 11**.

Listing 7. Część algorytmu programu obsługi odpowiedzialna za realizację funkcji MATCH_ROM

```
//Jesteśmy w trybie STATUS_MATCH_ROM_ACTIVE,
//czyli wysyłania przez układ Master numeru ID by zaadresować naszego Slave-a
case STATUS_MATCH_ROM_ACTIVE:

    //Czekamy 30us aż znajdziemy się
    //w środku przedziału czasu na emisję bitu
    _delay_us(SLAVE_SAMPLE_TIME);
    //Odczyt bitu
    if(ONE_WIRE_READ) Buffer[byteNr] |= (1<<bitNr);

    //Jeśli mamy kompletny bajt adresu
    // to przechodzimy do kolejnego
    if(++bitNr == 8)
        bitNr = 0;
        //Sprawdzamy, czy odebraliśmy
        //kompletny numer ID.
        if(++byteNr == 8){
            //Odebraliśmy kompletny numer ID,
            //więc porównujemy go z naszym i
            //podejmujemy decyzję o dalszym toku programu
            Byte = 0;
            for(uint8_t i = 0; i<8; i++) if(Buffer[i] != ID[i]) Byte = 1;
            if(Byte) {
                Status = STATUS_IDLE;
                STOP_TIMER;
            } else {
                Status = STATUS_SLAVE_SELECTED;
                //Uruchomienie Timera0
                //jako układu odmierzającego timeout
                setTimeout(TIMEOUT_2MS);
                //Przygotowanie zmiennych
                bitNr = byteNr = Byte = 0;
            }
        }
    }
break;
```

Listing 8. Część algorytmu programu obsługi odpowiedzialna za realizację funkcji SEARCH_ROM

```
//Jesteśmy w trybie STATUS_SEARCH_ROM_ACTIVE,
//czyli wyszukiwania przez układ Master numeru ID
case STATUS_SEARCH_ROM_ACTIVE:

    //Dla każdego bitu przeprowadzamy 3 operacje
    switch(searchNr) {
        case SEARCH_SEND_BYTE:
            //Wysyłamy bit adresu układu Slave.
            //Jeśli bit=0 to przedłużamy stan niski
            if(!(ID[byteNr] & (1<<bitNr))) {
                ONE_WIRE_CLEAR;
                _delay_us(SLAVE_ZERO_TIME);
                ONE_WIRE_RELEASE;
            }
        break;

        case SEARCH_SEND_NEGATED_BYTE:
            //Wysyłamy zanegowany bit adresu układu Slave.
            //Jeśli bit=1 to przedłużamy stan niski
            if(ID[byteNr] & (1<<bitNr)) {
                ONE_WIRE_CLEAR;
                _delay_us(SLAVE_ZERO_TIME);
                ONE_WIRE_RELEASE;
            }
        break;

        case SEARCH_COMPARE_BYTE:
            //Układ Master przesyła w odpowiedzi bit
            //a Slave porównuje ze swoim bitem w tym miejscu
            //Jeśli się zgadza to proces idzie dalej
            //a jeśli nie to Slave się resetuje i czeka na sygnał Reset
            _delay_us(SLAVE_SAMPLE_TIME);
            //Odczyt bitu i porównanie z przesyłanym wcześniej
            if(ONE_WIRE_READ != ((ID[byteNr] & (1<<bitNr))>>bitNr)) {
                Status = STATUS_IDLE;
                STOP_TIMER;
            }
        break;
    }

    //Sprawdzamy, czy wszystkie 3 operacje wykonano dla bieżącego bitu
    if(++searchNr > SEARCH_COMPARE_BYTE) {
        searchNr = SEARCH_SEND_BYTE;

        //Jeśli przesłaliśmy kompletny bajt adresu
        //to przechodzimy do kolejnego
        if(++bitNr == 8) {
            bitNr = 0;
            //Sprawdzamy, czy przesłaliśmy
            //kompletny numer ID.
            if(++byteNr == 8) {
                Status = STATUS_IDLE;
                STOP_TIMER;
            }
        }
    }
break;
```

Kilka niezbędnych słów uwagi wymaga tematyka czasu latencji procedury obsługi przerwania INT0 realizującej obsługę interfejsu 1-Wire, które to zagadnienie jest szczególnie istotne w przypadku wysyłania danych przez układ Slave. Jak wiemy odczyt danych przez układ Master inicjowany jest poprzez wygenerowanie opadającego zbocza sygnału (ściągnięcie magistrali do logicznego „0") przez czas z zakresu 1…5 µs. Po wystąpieniu takiego zbocza sygnału układ Slave wystawia na magistralę bieżący bit danych a układ Master dokonuje jego odczytu, przy czym czas od wystąpienia opadającego zbocza sygnału do operacji odczytu nie może przekroczyć 15 µs. Zwykle implementuje się, że układ Master dokonuje odczytu przesłanego bitu na końcu okna odczytu, czyli po ok. 15 µs od wystąpienia opadającego zbocza sygnału. Jest to dla nas o tyle istotne, że procedura obsługi przerwania INT0 musi zostać napisana w taki sposób, aby zapewnić wystawienie bitu na magistralę w nieprzekraczalnym czasie, o którym wspomniałem powyżej a najlepiej jak najszybciej po zwolnieniu magistrali przez układ Master (po ustąpieniu ściągnięcia magistrali).

Jak wiemy standardowy czas latencji od wystąpienie przerwania do skoku do wektora przerwania dla mikrokontrolerów AVR wynosi 4 takty zegara taktującego, a więc bardzo mało. W takim razie to nie stanowi potencjalnego problemu przy taktowaniu zegarem o częstotliwości 8 MHz. Jest jednak małe, acz istotne „ale". Skok do wektora przerwania nie jest równoznaczny z natychmiastową reakcją programu obsługi przerwania na zainicjowany proces odczytu. Zanim program przejdzie do tego miejsca wykonywanych jest szereg innych instrukcji, których obecność wynika z pracy kompilatora, który dba o integralność rejestrów procesora używanych w procedurze obsługi przerwania tak, aby ich wartości po wyjściu z tejże procedury pozostały niezmienione (odkłada je na stos). Co więcej, na liczbę tych niezbędnych operacji nie mamy większego wpływu, gdyż piszemy w języku wysokiego poziomu, a co za tym idzie, ten aspekt programowania pozostawiamy kompilatorowi. Na szczęście możemy podejrzeć plik deasemblacji (*.lss) i przekonać się, jak wygląda wygenerowany kod maszynowy co jednocześnie pozwala ocenić czas wykonania poszczególnych partii programu. Innym sposobem jest wykorzystanie wbudowanego w środowisko programistyczne symulatora.

Właśnie użycie pierwszej z możliwości pozwoliło mi ocenić czas odpowiedzi naszego urządzenia na żądanie odczytu przesłane przez układ Master, który w przybliżeniu wynosi 50 taktów zegara, czyli w granicach 6,25 µs. Potwierdziłem to również w symulatorze środowiska Microchip Studio. Jest to wartość w zupełności akceptowalna i pozwalająca zmieścić się w 15 µs oknie odczytu urządzenia

Master. W przypadku zapisu do układu Slave latencja nie jest aż tak krytycznym elementem, gdyż odczyt stanu magistrali dokonywany przez układ podrzędny zachodzi dopiero po upłynięciu czasu 30 µs od wystąpienia opadającego zbocza sygnału. Niemniej jednak nawet w tym wypadku, jak i w ogóle w przypadku implementacji funkcji ISR, ważny jest sumaryczny, maksymalny czas obsługi zdarzenia (czyli od wejścia do wyjścia z funkcji ISR), który nie może przekroczyć wartości 60 µs, czyli czasu trwania pojedynczego bitu (gdyż z takim interwałem będzie wywoływane przerwanie INT0).

Przeprowadzone testy praktyczne potwierdzone wcześniejszą symulacją w środowisku Microchip Studio wykazały, że maksymalny czas obsługi zdarzenia wynosi około 51 µs, a więc całkiem sporo. Wynika to poniekąd z czasu oczekiwania (30 µs) na odczyt stanu magistrali dokonywany przez układ podrzędny (w przypadku zapisu przez układ Master) i jest pokłosiem przyjętego rozwiązania programowego implementacji magistrali 1-Wire w wersji Slave. Czas ten można byłoby wydatnie skrócić poprzez zrezygnowanie z oczekiwania wspomnianych 30 µs, lecz wymagałoby to wykorzystania dodatkowego timera (oraz przerwania od jego przepełnienia) oraz gruntownej modyfikacji i komplikacji kodu obsługi wszystkich przerwań, co z pewnością odbiłoby się na wielkości kodu wynikowego aplikacji. Nie zdecydowałem się na ten krok, gdyż zachowujemy odpowiedni margines bezpieczeństwa a jedyne zmiany, jakie postanowiłem wprowadzić to skrócenie czasu oczekiwania na odczyt stanu magistrali dokonywany przez układ podrzędny do wartości 25 µs (opcjonalnie można ustawić 20 µs), przez co sumaryczny, maksymalny (dla najgorszego przypadku) czas obsługi przerwania INT0 wynosi 46 µs. Jest to wartość w pełni bezpieczna. Co prawda podczas obsługi transmisji 1-Wire pozostaje wyłącznie około 24% czasu procesora na wykonywanie innych zadań (np. w pętli głównej), ale nasze urządzenie w zasadzie żadnych innych zadań nie wykonuje, gdyż pomiar temperatury inicjowany jest tylko i wyłącznie rozkazem przesłanym magistralą przez co w czasie pomiaru nie odbywa się żaden „ruch” w ramach tego medium.

Poza tym pomiar dokonywany przy udziale przetwornika ADC nie koliduje w żaden sposób z implementacją magistrali 1-Wire. Niemniej jednak Czytelnicy, którzy chcieliby samodzielnie zastosować omówione

Listing 11. Funkcja obsługi przerwania od przepełnienia licznika Timer0 odpowiedzialna za obsługę timeout-ów

```
ISR(TIM0_OVF_vect) {
    Status = STATUS_IDLE;
    STOP_TIMER;
}
```

Listing 9. Część algorytmu programu obsługi odpowiedzialna za odbiór bajta realizowanej funkcjonalności

```
//Jesteśmy w trybie STATUS_SLAVE_SELECTED,
//czyli nasz Slave został zaadresowany poleceniem MATCH_ROM lub
//wybrany poprzez polecenie SKIP_ROM. W takim wypadku oczekujemy
//na rozkaz dotyczący wykonywanej operacji
case STATUS_SLAVE_SELECTED:
//Czekamy 30us aż znajdziemy się w środku przedziału czasu na emisję bitu
    _delay_us(SLAVE_SAMPLE_TIME);
    //Odczyt bitu
    if(ONE_WIRE_READ) Byte |= (1<<bitNr);

    //Jeśli mamy kompletny rozkaz to przechodzimy
    // do kolejnego etapu programu
    if(++bitNr == 8) {
        switch(Byte) {
            case CMD_CONVERT_T:
                //Żądanie pomiaru temperatury
                measureTemp = 1;
                Status = STATUS_IDLE;
                STOP_TIMER;
            break;

            case CMD_READ_SCRATCHPAD:
                Status = STATUS_READ_SCRATCHPAD_ACTIVE;
                //Uruchomienie Timera0
                //jako układu odmierzającego timeout
                setTimeout(TIMEOUT_5MS);
                //Przygotowanie zmiennych
                bitNr = byteNr = 0;
            break;

            default:
                //Nieobsługiwany rozkaz
                //wracamy do stanu wyjściowego
                Status = STATUS_IDLE;
                STOP_TIMER;
            break;
        }
    }
break;
```

Listing 10. Część algorytmu programu obsługi odpowiedzialna za realizację funkcji READ_SCRATCHPAD

```
//Jesteśmy w trybie STATUS_READ_SCRATCHPAD_ACTIVE,
//czyli układ Master będzie czytał zawartość scratchpad-a.
//Maksymalnie 9 bajtów.
case STATUS_READ_SCRATCHPAD_ACTIVE:
    //Czekamy na zwolnienie magistrali przez Mastera
    //aby nie liczyć tego samego bitu jako kolejnego
    while(!ONE_WIRE_READ);
    //Wysyłamy kolejny bit scratchpad-a układu Slave.
    //Jeśli bit=0 to przedłużamy stan niski
    if(!(Scratchpad[byteNr] & (1<<bitNr))) {
        ONE_WIRE_CLEAR;
        _delay_us(SLAVE_ZERO_TIME);
        ONE_WIRE_RELEASE;
    }

    //Jeśli wysłaliśmy kompletny bajt scratchpad-a
    //to przechodzimy do kolejnego
    if(++bitNr == 8) {
        bitNr = 0;
        //Sprawdzamy, czy wysłaliśmy wszystkie bajty scratchpad-a.
        //Jeśli tak to przechodzimy do stanu spoczynkowego.
        if(++byteNr == 9) {
            Status = STATUS_IDLE;
            STOP_TIMER;
        }
    }
break;
```

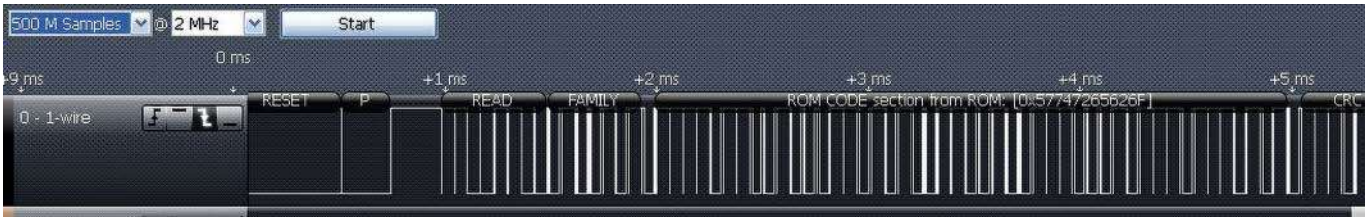
Listing 12. Funkcja odpowiedzialna za pomiar temperatury układu TC1047A

```
void readADC(void) {
    int16_t Temperature;
    uint8_t Sign, CRC8 =0;
    //Start konwersji – Prescaler= 64 (125kHz)
    ADCSRA = (1<<ADEN)|(1<<ADSC)|(1<<ADPS2)|(1<<ADPS1);
    //Czekamy na jej zakończenie – 120us
    while(ADCSRA & (1<<ADSC));

    //Przeliczenie na zakres -80...250 (jednostka 0.5'C)
    Temperature = -100 + ((ADC * (int32_t) 3418) / (int32_t) 10000);
    if(Temperature < 0) Sign = 0xFF; else Sign = 0x00;
    Temperature &= 0xFF;

    //Obliczamy CRC8 z pierwszych 8 bajtów
    //Aktualizacja CRC8
    _crc_ibutton_update(CRC8, Temperature);
    //Aktualizacja CRC8
    _crc_ibutton_update(CRC8, Sign);
    //Aktualizacja CRC8
    for(uint8_t i = 2; i<8; i++)
        CRC8 = _crc_ibutton_update(CRC8, Scratchpad[i]);

    //Atomowa aktualizacja scratchpad-a
    ATOMIC_BLOCK(ATOMIC_RESTORESTATE) {
        Scratchpad[0] = Temperature;
        Scratchpad[1] = Sign;
        Scratchpad[8] = CRC8;
    }
}
```



Rysunek 8. Rzeczywiste przebiegi sygnałów sterujących magistrali 1-Wire dla przypadku rozkazu READ_ROM wysłanego do naszego urządzenia DS18SW20

```
Listing 13. Funkcja odpowiedzialna za odczyt adresu ID układu Slave z pamięci EEPROM
mikrokontrolera

void readID(void) {
    uint8_t CRC8 = 0;
    //Sprawdzamy czy w EEPROMie zapisano unikalny numer ID.
    //Jeśli tak, to czytamy zamiast domyślnego.
    if(eeprom_read_byte(&IDEE[0]) != 0xFF)
        eeprom_read_block((uint8_t *) ID, IDEE, 8);
    //Obliczamy CRC8 umieszczony w 8. bajcie
    for(uint8_t i = 0; i<7; i++)
        //Aktualizacja CRC8
        CRC8 = _crc_ibutton_update(CRC8, ID[i]);
    ID[7] = CRC8;
}
```

Tabela 1. Pseudokod funkcji obsługujących urządzenie DS18SW20

Rodzaj operacji	Pseudokod
Sekwencja konwersji temperatury dla wybranego układu Slave	RESET MATCH_ROM Przesłanie 8 bajtów adresu CONVERT_T
Sekwencja odczytu Scratchpad-a dla wybranego układu Slave	RESET MATCH_ROM Przesłanie 8 bajtów adresu READ_SCRATCHPAD Odczytanie (do) 9 bajtów scratchpad-a
Sekwencja konwersji temperatury dla dowolnego układu Slave	RESET SKIP_ROM CONVERT_T
Sekwencja odczytu Scratchpad-a dla dowolnego układu Slave	RESET SKIP_ROM READ_SCRATCHPAD Odczytanie (do) 9 bajtów scratchpad-a
Sekwencja odczytu numeru seryjnego jedynego układu Slave	RESET READ_ROM Odczytanie 8 bajtów adresu
Sekwencja dla mechanizmu wyszukiwania adresów układów Slave	RESET SEARCH_ROM Start mechanizmu wyszukiwania adresów

rozwiązania w swoim oprogramowaniu muszą te kwestie szczegółowo przeanalizować nawet wtedy, gdy zdecydują się na zwiększenie częstotliwości taktowania mikrokontrolera, co samo w sobie minimalizuje ryzyko potencjalnych problemów będąc tak naprawdę rozwiązaniem najprostszym, lecz nie pozbawionym wad. Główną wadą takiej solucji jest wydatne zwiększenie zapotrzebowania na moc co niejednokrotnie nie jest pożądane.

Uff, tyle w kwestii implementacji magistrali 1-Wire w wersji Slave i emulacji termometru DS1820. Wiem, że było to dość długie opracowanie, ale moim zdaniem bardzo wartościowe z punktu widzenia poznania zasad działania tego medium transmisyjnego i zastosowania go do swoich potrzeb. Już na sam koniec 2 funkcje niezwiązane bezpośrednio z samą realizacją obsługi magistrali 1-Wire, ale niezbędne do emulacji układu DS1820. Pierwsza z nich to funkcja dokonująca pomiaru napięcia przetwornika TC1047A i konwertująca otrzymane dane na format temperatury zgodny ze specyfikacją układu DS1820 a następnie zapisująca je w pamięci Scratchpad. Ciało tej funkcji pokazano na **listingu 12**. Dla dociekliwych dodam, że pomiar napięcia przetwornika ADC, jego konwersja na temperaturę w standardzie układu DS1820, obliczenie CRC8 i aktualizacja scratchpad-a zajmuje ok. 270 µs, więc dokładnie tyle potrzeba aby po wysłaniu rozkazu CONVERT_T przystąpić do odczytu scratchpad-a, zapominając o magicznych 750 ms, jakie potrzebował układ DS1820 na konwersję temperatury. Druga i zarazem ostatnia funkcja to funkcja odpowiedzialna za odczyt adresu ID układu Slave z pamięci EEPROM mikrokontrolera (jeśli został tam zapisany) lub pozostanie przy adresie domyślnym utworzonym podczas definicji tablicy ID. Ciało tej funkcji pokazano na **listingu 13**. Fakt istnienia indywidualnego (zamiast domyślnego) adresu urządzenia w pamięci EEPROM oceniany jest na podstawie wartości pierwszego bajta pamięci EEPROM. Jeśli jest on różny od wartości 0xFF to przyjmuje się, że użytkownik wpisał swój własny numer do pamięci EEPROM, w przeciwnym wypadku pozostawiany jest numer domyślny (inicjowany na wstępie programu obsługi aplikacji). Powyższa funkcja oblicza ponadto wartość ósmego bajta adresu, który to za każdym razem powinien być sumą CRC8. Ustalenie adresu urządzenia dokonywane jest jednorazowo na początku programu obsługi aplikacji.

Na koniec, w **tabeli 1**, pokażę pseudokod funkcji obsługujących nasze urządzenie DS18SW20 (uruchamiany po stronie układu Master). Jak widać, programowa realizacja układu DS1820 jak i interfejsu 1-Wire jest niezmiernie prosta i zarazem bardzo ciekawa i skłania do stosowania tego interesującego interfejsu komunikacyjnego we własnych zastosowaniach czego przykładem niech będą chociażby dwa z moich wcześniejszych projektów zrealizowane w tamtym czasie przy użyciu pakietu Bascom: c-button (kopiarka pastylek DS1990, EP 2/2009) oraz 1-Wire LED (4 segmentowy, 3-kolorowy wyświetlacz LED wyposażony w sprzęg 1-Wire, EP 04/2011). Jako ciekawostkę, na **rysunku 8** pokazano rzeczywiste przebiegi sygnałów sterujących magistrali 1-Wire dla przypadku rozkazu READ_ROM wysłanego do naszego urządzenia DS18SW20.

Robert Wołgajew, EP