Programowy emulator DS1990

Oprogramowanie dla trybu 1-Wire Slave

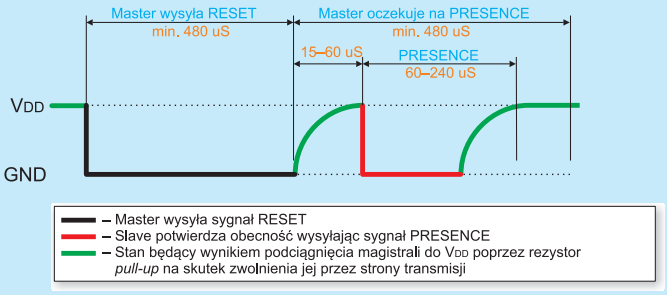
Chyba każdy z elektroników zna interfejs 1-Wire. Wystarczy wspomnieć, jaką popularnością cieszyły się niegdyś układy DS1990 powszechnie stosowane we wszelkiego rodzaju systemach kontroli dostępu czy też popularne i chętnie stosowane także dzisiaj termometry scalone typu DS18x20. Nie była to popularność przypadkowa. Oprócz funkcjonalności, którą miały, nie bez znaczenia był fakt względnie łatwej, programowej implementacji zastosowanego w nich niezmiernie ciekawego i oryginalnego interfejsu komunikacyjnego, który nie ma swoich odpowiedników w świecie elektroniki.

Każdy element z interfejsem 1-Wire ma swój unikalny numer seryjny i dlatego szereg układów wyposażonych w ten interfejs komunikacyjny może pracować na jednej i – co ważne – pojedynczej magistrali danych będącej jednocześnie źródłem zasilania. Stanowiło to o niespotykanej oryginalności interfejsu opracowanego przez firmę Dallas Semiconductor (w tej chwili Maxim). Pomimo tych niepodważalnych zalet, nie spotkałem się z systemami, które interfejs 1-Wire wykorzystywałyby do komunikacji pomiędzy własnymi rozwiązaniami sprzętowymi, a spotykane rozwiązania ograniczały się do programowej realizacji obsługi wspomnianej magistrali wyłącznie w celu umożliwienia współpracy z firmowymi peryferiami producenta, mimo że udostępnił on (co oczywiste) pełną specyfikację interfejsu. Z jednej strony nie budzi to zdziwienia, ponieważ firma Maxim dysponuje prawem patentowym na specyfikację tej magistrali, z drugiej zaś nic nie stoi na przeszkodzie, by ten arcyciekawy interfejs komunikacyjny z jego wszystkimi zaletami wykorzystać we własnych konstrukcjach w celu umożliwienia obustronnej współpracy urządzeń niebędących produktami firmy Maxim. Sprowadza się to do opracowania własnego elementu typu *Slave*, który obsługiwałby oferowaną przez interfejs 1-wire funkcjonalność, włączając w to mechanizm wyszukiwania adresów podłączonych do magistrali peryferiów (tzw. *SEARCH ROM*). Zademonstrujemy to na przykładzie programowej realizacji układu DS1990, opartej o dowolny mikrokontroler wyposażony w przerwanie zewnętrzne oraz timer sprzętowy (w naszym przypadku wybrano popularny mikrokontroler AVR typu ATmega8 taktowany wewnętrznym oscylatorem 8 MHz).

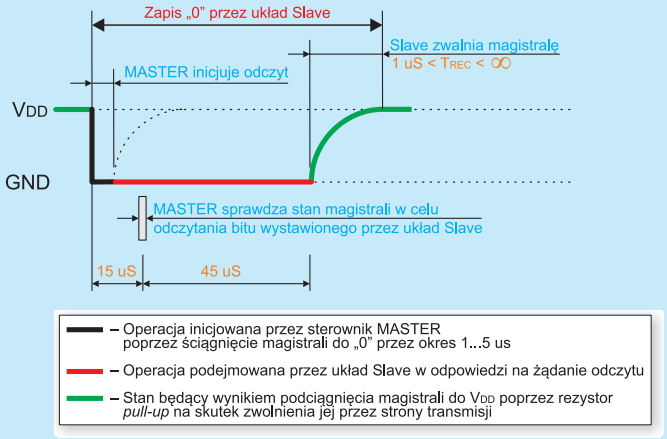
Podstawowe wiadomości o 1-Wire

Zanim jednak przejdę do opisu warstwy programowej (tym razem w języku C) niezbędne jest przypomnienie choćby podstawowych informacji na temat samego interfejsu będącego przedmiotem naszych rozważań, tym razem jednak z punktu widzenia układu podrzędnego typu Slave. Jak wspomniano, komunikacja na magistrali 1-Wire odbywa się za pomocą pojedynczego przewodu (stąd nazwa interfejsu), który oprócz linii danych może jednocześnie pełnić rolę przewodu zasilającego w konfiguracji tzw. zasilania pasożytniczego. Na magistrali 1-Wire 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, iż przesłanie każdego bitu informacji – niezależnie od kierunku transmisji – jest inicjowane wyłącznie przez układ Master za pomocą 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 dołączonych do magistrali musi mieć wyjście typu otwarty dren lub otwarty kolektor, a linia danych jest dołączona do zasilania przez rezystor podciągający o typowej wartości 4,7 kΩ. W stanie bezczynności powoduje to utrzymywanie się poziomu wysokiego na tej linii i zapewnia zasilanie urządzeń z 1-Wire pracujących w trybie zasilania pasożytniczego.

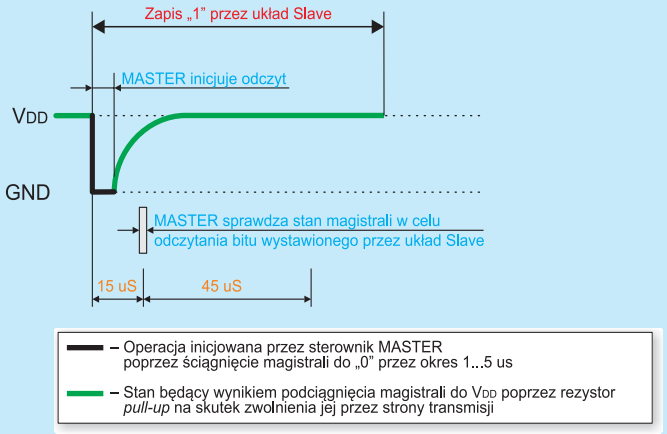
Przesyłane słowa są zawsze 1-bajtowe, a jako pierwszy jest transmitowany bit najmniej znaczący. Dodatkową i jedną z najważniejszych cech urządzeń z interfejsem 1-Wire jednocześnie odróżniającą je np. od urządzeń z interfejsem I²C, jest unikatowy, 8-bajtowy identyfikator układu zapisany w pamięci ROM. Jest on niepowtarzalny i właściwy tylko, i wyłącznie pojedynczemu układowi scalonemu (w pamięci komponentów produkowanych przez firmę Maxim-Dallas jest zapisywany na etapie produkcji). Najmniej znaczący bajt tego identyfikatora za-



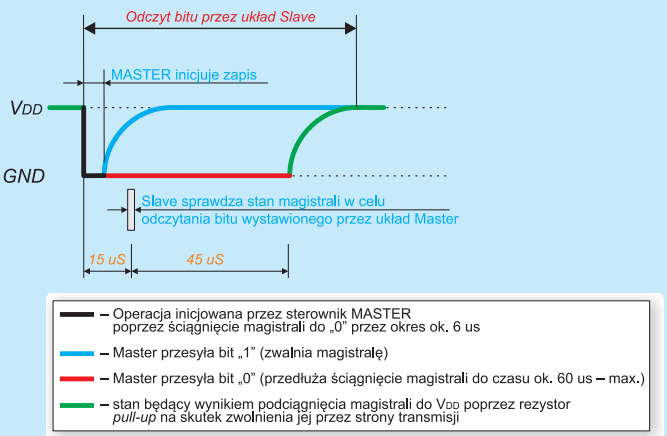
Rysunek 1. Procedura inicjalizacji magistrali 1-wire



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.



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

wiera kod rodziny układów (*Family code*), kolejne 6 bajtów to unikatowy kod konkretnego egzemplarza (właściwy adresu układu), a najbardziej znaczący bajt zawiera sumę kontrolną CRC (*Cyclic Redundancy Check*). Suma ta wyliczana jest na podstawie poprzednich siedmiu bajtów i 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.
- Sygnał Presence, wysyłany przez układy Slave, będący potwierdzeniem obecności tych układów na magistrali danych.
- Zapis logicznej „1" i „0".
- Odczyt logicznej „1" i „0".

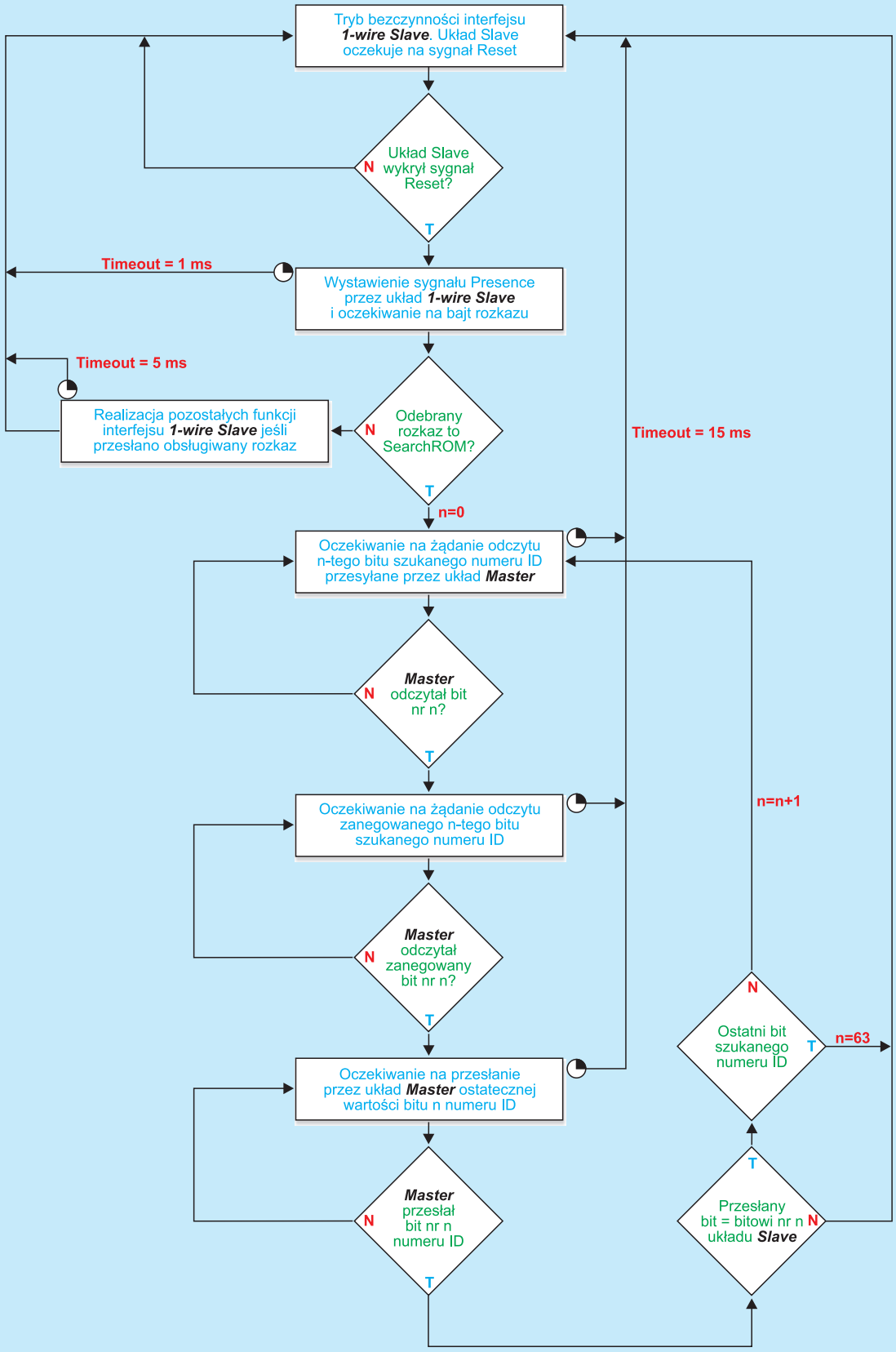
Najlepszym sposobem na zrozumienie zależności czasowych dla poszczególnych stanów pracy magistrali jest zobrazowanie ich na rysunkach przedstawiających sekwencje tychże sygnałów widziane z punktu widzenia układu Slave. Na **rysunku 1** przedstawiono sekwencję inicjalizacji magistrali 1-wire, która to, 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…960 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** przedstawiono 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).

Na **rysunku 4** przedstawiono 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 przedstawię graf funkcjonalny bodajże najciekawszego mechanizmu charakterystycznego wyłącznie dla magistrali tego typu a przeznaczony do wyszukania adresów wszystkich układów do niej przyłączonych nazwany *Search ROM*. Należy pamiętać, że początkowo układ Master nie zna adresów i liczby układów przyłączonych do magistrali, w związku, z czym musi istnieć pewny mechanizm ich ustalenia. Operacja taka jest inicjowana przez układ Master za pomocą rozkazu 0xF0 (*Search ROM*). Następnie każdy układ Slave wysyła na magistralę wartość pierwszego, najmniej znaczącego bitu swojego numeru ID (oczywiście zapis jest zawsze inicjowany przez układ Master). Mając na uwadze specyfikę interfejsu 1-Wire polegająca na tym, że wszystkie układy podłączone są do tej samej magistrali danych należy pamiętać, iż na odebraną komendę przeszukiwania odpowiedzą dokładnie w tym samym czasie wszystkie układy Slave. W związku z tym rezultat odbierany przez układ Master jest iloczynem logicznym stanów wyjść wszystkich tych układów (*tzw. wired and*). Kolejnym krokiem jest wystawianie przez układy Slave zanegowanego, pierw-

szego bitu swojego numeru ID (oczywiście jak zwykle, na żądanie układu Master). W wyniku wspomnianych wcześniej dwóch operacji zapisu układ Master decyduje i wystawia, a układ Slave odczytuje, pierwszy, najmniej znaczący bit, znalezionego (w tej chwili przez układ Master) numeru ID. 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



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

Listing 1. Deklaracja identyfikatora oraz zmiennej stanu
```
//Numer ID naszego emulatora DS1990
uint8_t ID[8] = {,R',’o’,’b’,’e’,’r’,’t’,’W’,0};
//Stan procedury obsługi magistrali 1-wire
volatile uint8_t slaveStatus = IDLE;
```

Listing 2. Deklaracja typu wyliczeniowego opisującego stan pracy procedury
```
enum
{
  IDLE,  //Stan bezczynności układu Slave - oczekiwanie na sygnał Reset
  WAIT_FOR_COMMAND,  //Oczekiwanie na rozkaz sterujący
  READ_ROM,  //Oczekiwanie na rozkaz READ_ROM (odczyt numeru ID)
  SEARCH_ROM  //Oczekiwanie na procedurę SEARCH_ROM
};
```

Listing 3. Konfigurowanie obsługi przerwania INT0
```
int main(void)
{
  //Aktualizacja sumy CRC8 z pierwszych 7 bajtów tablicy numeru ID naszego układu Slave
  for(uint8_t i=0; i<7; i++) ID[7] = _crc_ibutton_update(ID[7], ID[i]);
//Konfiguracja przerwania INT0: zbocze opadające
  MCUCR |= (1<<ISC01);
//Odblokowanie przerwania INT0
  GICR |= (1<<INT0);
//Odblokowanie przerwania od przepełnienia licznika Timer1 (obsługa timeout’u)
  TIMSK |= (1<<TOIE1);
  sei();
  while(1);
}
```

Listing 4. Definicje zawarte w pliku *defines.h*
```
#define TIMER1_PRESCALER 8  //Wartośc preskalera licznika Timer1
#define START_TIMER1 TCNT1 = 0; TCCR1B |= (1<<CS11)  //Konfiguracja Timer1: tryb normalny, preskaler = 8
#define STOP_TIMER1 TCCR1B = 0  //Zatrzymanie licznika Timer1

//Przeliczanie mikrosekund na tyknięcia zegara Timer1 przy zadanej wartości preskalera
#define microSeconds(us) ((us)*(F_CPU/1000000)/TIMER1_PRESCALER)

//Przeliczanie mikrosekund na tyknięcia zegara Timer1 pozostałe do przeładowania licznika (uruchomienia
procedury TIMER1_OVF_vect)
#define timeOut(us) (0xFFFF - ((us)*(F_CPU/1000000)/TIMER1_PRESCALER))
//Definicja umożliwiająca konfigurację licznika Timer1 w celu odmierzenia zadanego czasu do jego przeładowania
[w us]
#define START_TIMEOUT(us) TCNT1 = timeOut(us); TCCR1B |= (1<<CS11)  //Tryb normalny, preskaler = 8

#define WIRE1_PIN PIND
#define WIRE1_DDR DDRD
#define WIRE1_NR PD2

#define RESET_1WIRE WIRE1_DDR |= (1<<WIRE1_NR)  //Ściągnięcie magistrali 1-wire do „0” poprzez ustawienie
kierunku portu jako wyjściowy
#define SET_1WIRE WIRE1_DDR &= ~(1<<WIRE1_NR)  //Zwolnienie magistrali 1-wire i tym samym podciągnięcie
rezystorem do plusa zasilania
```

Listing 5. Czynności wykonywane przez program w stanie *IDLE*
```
case IDLE:
//Uruchamiamy Timer odmierzający czas oczekiwanego impulsu RESET
  START_TIMER1;
//Czekamy, aż Master zwolni magistralę 1-wire
  while(!(WIRE1_PIN & (1<<WIRE1_NR)));
  STOP_TIMER1;
//Odebrano oczekiwany sygnał RESET
  if(TCNT1 > microSeconds(460))
  {
//Wystawiamy sygnał PRESENCE by zasygnalizować obecność
//układu Slave na magistrali 1-wire
    RESET_1WIRE;
    _delay_us(240);
    SET_1WIRE;
//Zerowanie zmiennych pomocniczych
    Command = 0;
    bitIndex = 0;
//Zmiana stanu procedury obsługi magistrali 1-wire
    slaveStatus = WAIT_FOR_COMMAND;
//Uruchamiamy zegar odmierzający timeout = 1ms
//(oczekiwanie na rozkaz)
    START_TIMEOUT(1000);
  }
break;
```

Programowy emulator 1-Wire

W tym miejscu dysponujemy już niezbędna wiedzą z zakresu protokołu 1-wire, aby zrealizować zamierzone założenia w związku, z czym przejdźmy do realizacji programowej na przykładzie emulacji układu DS1990. Jak wiemy, każda operacja mająca miejsce na magistrali 1-wire inicjowana jest przez układ Master poprzez zciągnięcie tejże magistrali do logicznego „0”. W związku z tym naturalnym sposobem na obsłużenie protokołu ze strony układu Slave jest wykorzystanie przerwania zewnętrznego (np. INT0) skonfigurowanego w ten sposób by zachodziło przy opadającym zboczu sygnału. Wykorzystanie przerwania zewnętrznego jest o tyle niezbędne, iż magistrala 1-wire wymusza dość rygorystyczne wymogi czasowe, więc wykorzystanie typowego portu I/O i tzw. pollingu jest niewystarczające w przypadku, gdy układ Slave miałby wykonywać jeszcze inne czasowo zależne operacje niezwiązane z obsługą magistrali. Wykorzystanie przerwania zewnętrznego niesie za sobą tę dodatkową zaletę, iż cała obsługa protokołu zostanie „zamknięta” w ramach jednej, krótkiej funkcji ISR, a program główny może realizować inną funkcjonalność, niezbędną z punktu widzenia konkretnej aplikacji. Zadeklarujmy, zatem dwie podstawowe zmienne globalne: numer seryjny naszego urządzenia Slave oraz zmienną przechowującą aktualny stan procedury obsługi układu Slave magistrali 1-wire (**listing 1**). Dodatkowo, aby poprawić czytelność programu obsługi, wprowadzimy typ wyliczeniowy opisujący stan pracy wspomnianej procedury (**listing 2**). W związku z tym, że obsługa protokołu odbywa się w całości w ramach procedury obsługi przerwania INT0 (*INT0_vect*) funkcja główna naszego programu obsługi ogranicza się wyłącznie do konfiguracji niezbędnych peryferiów mikrokontrolera (**listing 3**)

W programie obsługi jest używany także timer sprzętowy (w tym przypadku 16-bitowy Timer1) oraz przerwanie od jego przepełnienia (*TIMER1_OVF_vect*). Użycie tego 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 przedstawiono z punktu widzenia układu Slave, gdyż algorytm dla układu Master jest nieco bardziej skomplikowany (bazuje na algorytmie tzw. drzewa binarnego).

peryferium konieczne jest w przypadku, gdy układ Master z jakiś powodów wysłałby niepełną sekwencję sygnałów sterujących (np. tylko sygnał Reset bez jakichkolwiek dalszych komend), co spowodowałoby zmianę stanu procedury obsługi magistrali 1-Wire na predefiniowaną wartość *WAIT_FOR_COMMAND* i oczekiwanie na rozkaz sterujący uniemożliwiając dalsze poprawne funkcjonowanie algorytmu obsługi. Timer ten każdorazowo jest inicjowany w taki sposób, by po upłynięciu zadanego czasu (zależnego od oczekiwanych zadań po stronie układu Master) przywrócić stan spoczynkowy procedury obsługi (*Idle*) w przypadku błędów po stronie układu nadrzędnego. Aby poprawić czytelność programu obsługi oraz uprościć zapis niektórych wartości, wprowadzono dodatkowe definicje (dodatkowy plik *defines.h*). Jego zawartość pokazano na **listingu 4**.

Przejdźmy zater 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. Ten algorytm opisuje część programu obsługi zaprezentowana na **listingu 5** (o wykonaniu poszczególnych części programu obsługi decyduje wartość zmiennej *slaveStatus*). Po wykonaniu powyższych czynności pokazanych na list. 5, układ Slave oczekuje (przez czas 1 ms) na odebranie komendy sterującej i w zależności od jej rodzaju realizuje pozostałe funkcje typowe dla sprzętowego układu DS1990. Nasze urządzenie obsługuje 2 rodzaje komend sterujących: *READ_ROM* i *SEARCH_ROM*. Odbiór komendy sterującej realizuje część programu obsługi urządzenia pokazana na **listingu 6**. Następnie, w zależności od rodzaju odebranej komendy sterującej, realizowana jest stosowna część programu obsługi.

Rozkaz *READ_ROM*, czyli odczyt numeru seryjnego przez układ Master realizuje niewielka część programu obsługi pokazana na **listingu 7**.

Rysunek 6. Rzeczywiste przebiegi sygnałów sterujących magistrali 1-wire dla przypadku rozkazu READ_ROM wysłanego do naszego układu Slave.

Listing 6. Cześć programu służąca do odbioru komendy sterującej

```
case WAIT_FOR_COMMAND:
   _delay_us(15);
  if(WIRE1_PIN & (1<<WIRE1_NR)) Command |= (1<<bitIndex);
  if(++bitIndex == 8)  //Odebrano pełen rozkaz
  {
    STOP_TIMER1;  //Zatrzymujemy zegar odmierzający timeout
    switch(Command)
    {
      case 0x33:
//Zmiana stanu procedury obsługi magistrali 1-wire
        slaveStatus = READ_ROM;
//Uruchamiamy zegar odmierzający timeout = 5ms (oczekiwanie na
//odczyt numeru ID)
        START_TIMEOUT(5000);
        break;
      case 0xF0:
//Zmiana stanu procedury obsługi magistrali 1-wire
        slaveStatus = SEARCH_ROM;
//Uruchamiamy zegar odmierzający timeout = 15ms (oczekiwanie na
//procedurę SEARCH_ROM)
        START_TIMEOUT(15000);
        break;
      default: slaveStatus = IDLE;
    }
//Zerowanie zmiennych pomocniczych
    bitIndex = 0; byteIndex = 0; searchIndex = 0;
  }
break;
```

Rozkaz *SEARCH_ROM*, czyli mechanizm wyszukiwania numeru seryjnego wykonywany przez układ Master, realizuje fragment programu pokazany na **listingu 8**.

Wspomniana wcześniej procedura obsługi przerwania od przepełnienia licznika Timer1 niezbędna z punktu widzenia odmierzania tzw. czasu timeout zamieszczono na **listingu 9**.

Dla porządku przedstawię deklaracje zmiennych lokalnych procedury obsługi magistrali 1-Wire (*INT0_vect*). Zamieszczono je na **listingu 10**.

Podsumowanie

Programowa realizacja układu Slave interfejsu 1-wire nie jest trudna. Przedstawiony opis programu może być dobrym wstępem do samodzielnego eksperymentowania i umożliwia zastosowanie tego ciekawego interfejsu komunikacyjnego we własnych aplikacjach. Przykładami mogą być dwa z moich wcześniejszych projektów wykonane przy użyciu pakietu Bascom:

- ***c-button*** (kopiarka pastylek DS1990),
- ***1-wire LED*** (4 segmentowy, 3-kolorowy wyświetlacz LED wyposażony w interfejs 1-wire).

Z zastosowaniem tego ostatniego projektu wykonałem prosty system mikroprocesorowy, w którym na jednej magistrali 1-Wire znajdowały się dwa termometry DS18S20 oraz wyświetlacz, a wszystko sterowane przez niewielki mikrokontroler ATtiny2313. Na **rysunku 6** jako ciekawostkę przedstawiono rzeczywiste przebiegi sygnałów sterujących magistrali 1-Wire dla przypadku rozkazu READ_ROM wysłanego do urządzenia.

Robert Wołgajew, EP

Listing 7. Programowa realizacja rozkazu READ_ROM

```
case READ_ROM:
//Jeśli aktualnie transmitowany bit=0 -> ściągamy magistralę
//1-wire przez kolejne 45us. Jeśli ten bit=1 -> nie robimy nic,
//magistrala zostanie podciągnięta do VCC
if((ID[byteIndex]&(1<<bitIndex)) == 0)
  {
    RESET_1WIRE;
    _delay_us(45);
    SET_1WIRE;
  }
//Przesłano właśnie 8 bitów bieżącego bajta
  if(++bitIndex == 8)
  {
    bitIndex = 0;
//Przesłano właśnie wszystkie bajty numeru ID - zatrzymujemy
//zegar odmierzający timeout
    if(++byteIndex == 8)
    {
      slaveStatus = IDLE;
      STOP_TIMER1;
    }
  }
  break;
```

Listing 8. Programowa realizacja rozkazu *SEARCH_ROM*

```
case SEARCH_ROM:
//W trakcie procedury SEARCH_ROM wykonywane są 3 operacje
//(określone wartością zmiennej searchIndex) dla każdego bitu
//adresu ID: wysłanie bitu, wysłanie zanegowanego bitu oraz
//odczyt bitu przesłanego przez układ Master i następujące po
//tym porównanie odczytanego bitu z bitem adresu ID urządzenia
//Slave
  switch(searchIndex)
  {
    case 0:  //Wysłanie bitu z bieżącej pozycji numeru ID
      if((ID[byteIndex]&(1<<bitIndex)) == 0)
      {
        RESET_1WIRE;
        _delay_us(45);
        SET_1WIRE;
      }
      break;
    case 1:  //Wysłanie zanegowanego bitu z bieżącej pozycji numeru ID
      if((ID[byteIndex]&(1<<bitIndex)) != 0)
      {
        RESET_1WIRE;
        _delay_us(45);
        SET_1WIRE;
      }
      break;
    case 2:  //Odczyt bitu przesłanego przez układ Master
//Układ Master przesyła w odpowiedzi bieżący bit a Slave porównuje
//go ze swoim bitem w tym miejscu numeru ID i jeśli się zgadza, to
//proces podąża dalej a jeśli nie to Slave przechodzi do bezczynności
      _delay_us(15);
      if((((ID[byteIndex]&(1<<bitIndex)) == 0) &&
      ((WIRE1_PIN & (1<<WIRE1_NR)) != 0)) ||
      (((ID[byteIndex]&(1<<bitIndex)) != 0) &&
      ((WIRE1_PIN & (1<<WIRE1_NR)) == 0)))
      {
        slaveStatus = IDLE;
        STOP_TIMER1;
      }
      break;
  }
  if(++searchIndex == 3)  //Komplet operacji dla bieżącego bitu
  {
    searchIndex = 0;
    if(++bitIndex == 8)  //Przetworzono już 8 bitów bieżącego bajta
    {
      bitIndex = 0;
//Przetworzono właśnie wszystkie bajty numeru ID - zatrzymujemy zegar
//odmierzający timeout
      if(++byteIndex == 8)
      {
        slaveStatus = IDLE;
        STOP_TIMER1;
      }
    }
  }
  break;
```

Listing 9. Procedura obsługi przerwania służąca do odmierzania czasu *timeout*

```
ISR(TIMER1_OVF_vect)
{
  slaveStatus = IDLE;
  STOP_TIMER1;
}
```

Listing 10. Deklaracje zmiennych lokalnych procedury obsługi magistrali 1-Wire

```
//Rozkaz odebrany magistralą 1-wire
static uint8_t Command;
//Zmienna pomocnicza indeksująca kolejno odebrane lub wysłane bity
static uint8_t bitIndex;
//Zmienna pomocnicza indeksująca kolejno odebrane lub wysłane bajty
static uint8_t byteIndex;
//Zmienna pomocnicza indeksująca kolejne kroki procedury SEARCH_ROM
//dla każdego bitu adresu ID
static uint8_t searchIndex;
```