Bernard WYRWOŁ

Politechnika Śląska, Wydział Automatyki, Elektroniki i Informatyki
ORCID: 0000-0001-7325-8094

doi:10.15199/48.2023.11.03

Implementacja interfejsu 1-Wire w systemie wbudowanym z systemem operacyjnym FreeRTOS

Streszczenie. *Implementacja interfejsu 1-Wire w systemie wbudowanym, pracującym pod kontrolą systemu operacyjnego czasu rzeczywistego FreeRTOS, wymaga realizacji opóźnień o czasie trwania mniejszym, niż udostępnia to sam system operacyjny. Wykorzystanie opóźnień o pożądanym czasie trwania, realizowanych zwykle na drodze programowej, może prowadzić do niekorzystnego zwiększenia czasu reakcji takiego systemu. W artykule przedstawiono przykładowe rozwiązanie obsługi interfejsu 1-Wire w takim systemie wykorzystujące sprzętowy interfejs SPI.*

Abstract*. Implementation of the 1-Wire interface in the embedded system, controlled by the FreeRTOS real-time operating system, requires the realization of delays with a duration shorter than that provided by the operating system itself. The use of delays of a desired duration, usually implemented by software, can lead to an unfavorable increase in the response time of such a system. The article presents an example of the 1-Wire interface implementation in such a system using the hardware SPI interface. (**1-Wire interface implementation in embedded system with FreeRTOS operating system**).*

Słowa kluczowe: system wbudowany, mikrokontroler, 1-Wire, SPI, FreeRTOS.
Keywords: embedded system, microcontroller, 1-Wire, SPI, FreeRTOS.

Wprowadzenie

Interfejs 1-Wire jest jednym z rozwiązań wykorzystywanych do podłączania układów peryferyjnych w systemach wbudowanych [1, 2]. Zaletą jego jest prosta architektura magistrali (jednoprzewodowa, dwukierunkowa magistrala danych z rezystorem polaryzującym podłączonym do napięcia zasilania) oraz możliwość podłączenia wielu układów (adres 64-bitowy) podrzędnych (slave) przy jednym wyróżnionym układzie nadrzędnym (master). Zasilanie układów slave może odbywać się z linii danych magistrali (tzw. zasilanie pasożytnicze) lub wprost z napięcia zasilania systemu wbudowanego. Do jego wad można zaliczyć niewielką prędkość transmisji (typowo 16kbps lub 115,2kbps w trybie overdrive), co może się uwidaczniać szczególnie w przypadku obsługi dużej liczby układów dołączonych do magistrali.

Na rynku dostępnych jest wiele układów wyposażonych w interfejs 1-Wire takich jak [3]: układy programowalne wejść-wyjść cyfrowych ogólnego przeznaczenia (np. 8-bitowy DS2408 lub 2-bitowy DS2413), klucze dostępu typu iButton (np. DS1990A), układy zegara czasu rzeczywistego (np. DS2415), pamięci (np. 4-kbitowa pamięć RAM DS2423 o organizacji 16 stron, każda o pojemności 256 bitów lub 1024-bitowa pamięć EEPROM DS2431 o organizacji 4 stron, każda o pojemności 256 bitów), przetworniki analogowo-cyfrowe (np. 4-wejściowy przetwornik ADC sukcesywnej aproksymacji DS2450), potencjometry cyfrowe (np. DS2890), układy konwerterów interfejsów (np. konwerter interfejsu 1-Wire na interfejsy I2C/SPI DS28E18) czy też czujniki, z których najczęściej spotykane to czujniki temperatury (np. MAX30207, MAX31820, MAX31888, DS18B20), w tym także współpracujące z termoparami (np. MAX31850/MAX31851). Układy te można spotkać w wielu praktycznych aplikacjach opisywanych w literaturze [4-8].

Protokół komunikacyjny magistrali 1-Wire oparty jest na przesyłaniu linią danych impulsów o czasie trwania rzędu od kilku do kilkuset mikrosekund, przy czym każdą transmisję pakietu zawsze inicjuje układ nadrzędny master. Podstawowe operacje realizowane w protokole to zerowanie jednostek podrzędnych slave oraz przesyłanie bitu o stanie logicznym niskim lub wysokim. W oparciu o te elementarne sekwencje generowanych stanów formowane są ramki danych, zwykle 8-bitowe.

Obecnie dostępne mikrokontrolery ogólnego przeznaczenia, wykorzystywane w budowie systemów wbudowanych, nie posiadają sprzętowych układów do obsługi interfejsu 1-Wire. Z tego powodu konieczne jest zastosowanie dodatkowych, zewnętrznych układów sterujących (układy logiki programowalnej, mikrokontrolery, układy dedykowane), pełniących funkcję układu nadrzędnego master interfejsu 1-Wire (np. DS2485, do mikrokontrolera podłączony jest za pośrednictwem interfejsu I2C [3]), lub wykorzystanie zasobów sprzętowych lub programowych samego mikrokontrolera [1, 4].

W przypadku systemów wbudowanych z oprogramowaniem typu „bare-metal" najczęściej do realizacji zależności czasowych, wymaganych do programowej implementacji protokołu 1-Wire, wykorzystuje się opóźnienia w postaci pętli programowych (funkcje opóźniające blokujące np. *_delay_us()* ze środowiska GCC). Rozwiązania takie nie są efektywne, gdyż znacznie absorbują zasoby mikrokontrolera (czas jednostki centralnej). Bardziej złożone rozwiązania, pozwalające na obsługę interfejsu w tle aplikacji głównej, opierają się na wykorzystaniu liczników sprzętowych wbudowanych w mikrokontroler, pozwalających odmierzać wymagane interwały czasowe, oraz systemu przerwań. Zamiast liczników ogólnego przeznaczenia do odmierzania czasu mogą być wykorzystywane także inne zasoby sprzętowe mikrokontrolera takie jak np. interfejsy szeregowe, w których to czas transmisji pojedynczej ramki danych trwa ściśle określony odcinek czasu.

System wbudowany, z oprogramowaniem użytkownika pracującym pod kontrolą systemu operacyjnego czasu rzeczywistego RTOS, stawia pewne warunki i ograniczenia na programową implementację interfejsu 1-Wire. Sam system operacyjny udostępnia funkcje opóźniające nieblokujące (np. *vTaskDelay()* w systemie FreeRTOS), których minimalny czas opóźnienia jest narzucony przez system RTOS (w systemach wbudowanych zwykle 1ms), więc nie nadają się do realizacji opóźnień o dużo mniejszych wartościach. Z kolei wykorzystanie funkcji w postaci pętli programowych może z jednej strony zwiększyć czas reakcji systemu i tym samym zwiększyć jego podatność na błędy oprogramowania (szczególnie dla systemu RTOS pracującego w trybie kooperacji), z drugiej zaś trudne do określenia są dokładne wartości rzeczywistych opóźnień generowanych w ten sposób (dla systemu RTOS pracującego w trybie z wywłaszczaniem). Jedyne w tym przypadku rozwiązanie to wykorzystanie

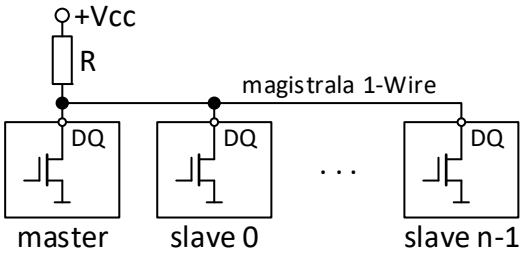
zasobów sprzętowych mikrokontrolera (liczniki, interfejsy szeregowe, system przerwań) i integracja programów ich obsługi z systemem operacyjnym.

Sposoby wykorzystania sprzętowo-programowych zasobów mikrokontrolera w obsłudze interfejsu 1-Wire opisane zostały w literaturze. Pozycja [9] pokazuje sposób programowy jego obsługi, natomiast nota aplikacyjna [10] przedstawia metodę wykorzystania układu asynchronicznej transmisji szeregowej UART do generowania wymaganych przebiegów czasowym na magistrali interfejsu. Z kolei w artykule [11] zaproponowano wykorzystanie licznika ogólnego przeznaczenia oraz systemu przerwań w realizacji tego interfejsu.

W niniejszym artykule przedstawiona została jeszcze jedna, nowatorska metoda obsługi interfejsu 1-Wire, wykorzystująca sprzętowy układ SPI (układ szeregowej synchronicznej transmisji danych) mikrokontrolera.

Interfejs oraz protokół 1-Wire

Interfejs szeregowy 1-Wire składa się z jednej, dwukierunkowej linii danych, z rezystorem polaryzującym R podłączonym do napięcia zasilania, do której dołączane są wyjścia układów realizujących na niej iloczyn logiczny (tj. z wyjściami typu open-colector lub open-drain). Stan wysoki na linii wymusza rezystor R, natomiast stan niski jest wymuszany przez jeden (lub kilka, w przypadku wystąpienia konfliktu) z układów. Wartość tego rezystora, np. dla układów DS18B20, zgodnie z zaleceniem noty aplikacyjnej [12], powinna wynosić ok. 4,7kΩ. Warstwę fizyczną interfejsu pokazano na rysunku 1. Dla czytelności pominięto na nim linie zasilające układów.



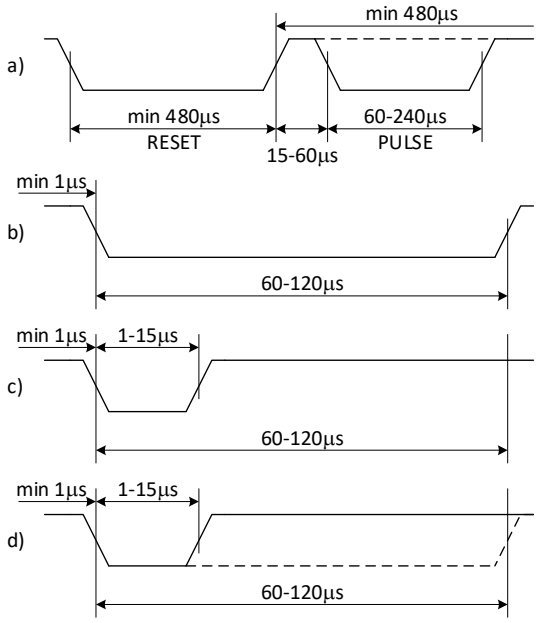
Rys.1. Podłączenie urządzeń do magistrali 1-Wire

Podłączone do magistrali układy slave mogą być zasilane ze swoich źródeł lub czerpać energię z samej magistrali (tzw. zasilanie pasożytnicze). Układ master zwykle posiada własne zasilanie.

Protokół interfejsu 1-Wire definiuje następujące niskopoziomowe sekwencje przebiegów generowane na magistrali: sekwencja zerująca układy slave, zapis stanu logicznego niskiego lub wysokiego oraz odczyt stanu logicznego niskiego lub wysokiego. Wszystkie operacje inicjowane są przez układ master, a każda z nich złożona jest ze stanów logicznych trwających ściśle określony czas, jak to pokazano na rysunku 2, przy czym stanem nieaktywnym magistrali jest zawsze stan wysoki.

W przypadku sekwencji zerowania (określanej jako RESET) układów slave oraz odczytu stanu magistrali 1-Wire układ master wymusza stan niski na magistrali przez określony czas (odpowiednio 480µs lub więcej oraz 1-15µs) po czym wprowadza magistralę w stan wysoki i oczekuje odpowiedzi z układu slave. W pierwszym przypadku zmiana stanu magistrali na niski oznacza, że do magistrali został podłączony przynajmniej jeden układ podrzędny slave (stan określany jako PULSE lub PRESENCE). W drugim przypadku stan magistrali interpretowany jest jako wartość pojedynczego bitu odczytanego z układu slave.

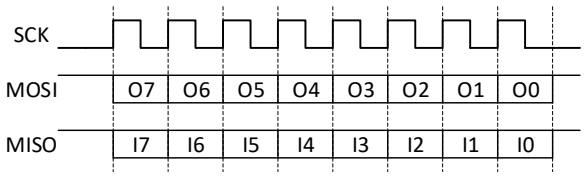
Niskopoziomowe sekwencje operujące na pojedynczych bitach magistrali można zestawić w bajty, które z kolei mogą być przesyłane pomiędzy układem master oraz konkretnym, odpowiednio zaadresowanym układem slave. Znaczenie poszczególnych, przesyłanych pomiędzy układami bajtów zależy już od konkretnego układu, dlatego tutaj nie będzie to omawiane. Szczegóły techniczne można znaleźć w [3]. Każda transmisja sekwencji pojedynczych bitów powinna być zawsze poprzedzona wysłaniem sekwencji RESET-PULSE przez układ master, zerującej układy slave.



Rys.2. Sekwencje stanów generowanych na magistrali 1-Wire: a) zerowanie układów slave, b) zapis stanu niskiego, c) zapis stanu wysokiego, d) odczyt stanu magistrali (odpowiedź układu slave)

Interfejs SPI

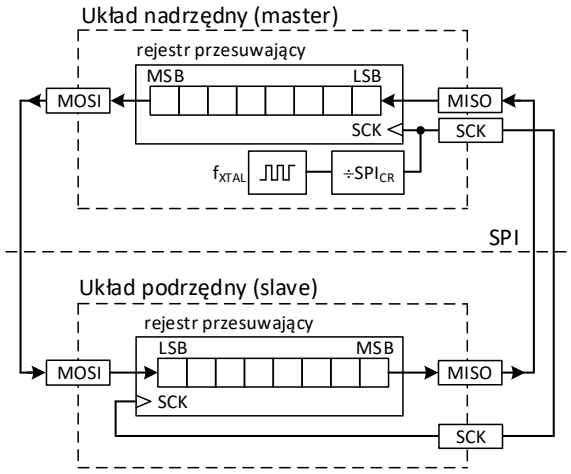
Jest to jeden z najprostszych interfejsów szeregowych umożliwiających przesyłanie danych w sposób synchroniczny. W danym cyklu transmisji danej (zwykle o rozmiarze jednego bajtu, w niektórych układach dwóch bajtów) występuje równoczesne, synchronizowane sygnałem zegarowym SCK, wysyłanie danej bit po bicie (za pośrednictwem linii MOSI), jak i odbieranie danej (za pośrednictwem linii MISO) zgodnie z tym, co pokazano na rysunku 3 (przesyłanie odbywa się zwykle począwszy od najbardziej znaczącego bitu).



Rys.3. Przesyłanie danych za pośrednictwem magistrali SPI

Podstawowym elementem interfejsu jest rejestr przesuwający, w który wyposażony jest zarówno układ nadrzędny master jak i podrzędny slave jak to przedstawiono na rysunku 4. Są one połączone liniami MOSI/MISO, wchodzącymi w skład magistrali SPI, w taki sposób, że tworzą pierścień. W przypadku wielu układów podrzędnych slave z reguły tylko jeden z nich jest w danej chwili aktywny (wybierany przez układ nadrzędny master) lub tworzą wielorejestrowy pierścień.

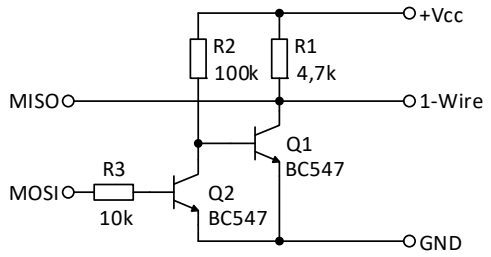
Układ master steruje przesyłaniem danych za pośrednictwem magistrali 1-Wire i posiada wbudowany generator sygnału zegarowego. Sygnał zegarowy pojawia się na linii SCK interfejsu wyłącznie podczas przesyłania danych. Prędkość transmisji (częstotliwość sygnału zegarowego SCK) zwykle można konfigurować jedynie w pewnym zakresie poprzez ustawienie współczynnika podziału (dzielnika) podstawowego sygnału zegarowego układu master (mikrokontrolera).



Rys.4. Magistrala SPI z układami master-slave

Implementacja interfejsu 1-Wire

Wykorzystanie interfejsu UART, do generowania sekwencji stanów na magistrali 1-Wire, zostało szczegółowo opisane w [10]. Metodę tam zaprezentowaną można zastosować również w przypadku interfejsu SPI. Wymagany jest tutaj konwerter, pokazany na rysunku 5, którego zadaniem jest przekształcenie wysyłanej ramki danych na odpowiednie stany magistrali 1-Wire oraz umożliwienie odczytu jej aktualnego stanu.



Rys.5. Układ konwertera interfejsu SPI na interfejs 1-Wire, na podstawie [10]

Interfejs SPI mikrokontrolera powinien być tak skonfigurowany, by czas wysyłania pojedynczej sekwencji stanów magistrali 1-Wire wynosił maksymalnie 120µs, dla operacji bitowych, oraz minimalnie 960µs, dla sekwencji zerującej RESET-PULSE. O ile w przypadku układu UART możliwa jest konfiguracja prędkości transmisji w dość szerokim zakresie, o tyle w przypadku układu SPI konfiguracja ta jest bardzo ograniczona. Sprowadza się jedynie do wyboru dzielnika sygnału zegarowego mikrokontrolera spośród tylko kilku wartości, np. dla mikrokontrolerów rodziny AVR są to dzielniki SPI$_{CR}$ przez 2, 4, 8, 16, 32, 64, 128 (przy ramce danych 8-bitów), z kolei dla mikrokontrolerów ARM rodziny STM32 wynoszą one 2, 4, 8, 16, 32, 64, 128, 256 (przy ramce danych 8- lub 16-bitów).

Czas trwania transmisji pojedynczej ramki danych wynosi

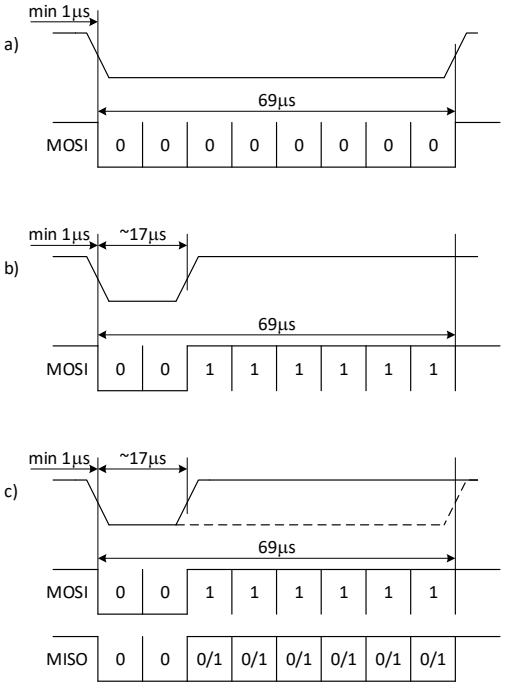
$$t_{frame} = \frac{N_{FR} \cdot SPI_{CR}}{f_{XTAL}}, \quad (1)$$

gdzie f_{XTAL} jest częstotliwością sygnału zegarowego mikrokontrolera (układu master), N_{FR} długością ramki danych wyrażoną w bitach (8 lub 16), a SPI$_{CR}$ wartością dzielnika częstotliwości.

Z reguły długość ramki danych, o ile jest taka możliwość, ustala się na stałą wartość i potem już nie zmienia. W celu wygenerowania impulsu RESET-PULSE (ok. 960µs, rysunek 2a) oraz sekwencji stanów (ok. 60-120µs, rysunki 2b-2d) można wykorzystać zmianę wartości dzielnika SPI$_{CR}$, podobnie jak w [10]. Ze wzoru (1) można wyznaczyć maksymalną częstotliwość sygnału zegarowego f_{XTAL}, dla której takie rozwiązanie jest możliwe. Wynosi ona 2MHz dla ramki 8-bitowej i pozwala wygenerować sekwencję RESET-PULSE tylko dla SPI$_{CR}$=256 oraz sekwencje bitowe dla SPI$_{CR}$=16. W przypadku ramki 16-bitowej częstotliwość ta może być dwukrotnie większa. Biorąc jednak pod uwagę możliwości współczesnych mikrokontrolerów to częstotliwość taktowania na poziomie 2-4MHz powoduje znaczne obniżenie wydajności systemu, w szczególności jak wykorzystywany jest w nim system operacyjny czasu rzeczywistego.

Z tego powodu, o ile nie jest możliwe obniżenie częstotliwości taktowania mikrokontrolera, zaproponowane zostało rozwiązanie oparte nie na zmianie częstotliwości sygnału zegarowego SCK tj. wartości SPI$_{CR}$, ale na sposobie generowania sekwencji zerującej RESET-PULSE poprzez zwielokrotnienie liczby wysyłanych ramek za pośrednictwem interfejsu SPI.

Przyjmując przykładową, typową wartość częstotliwości f_{XTAL}=14,7456MHz, dla 8-bitowej ramki danych oraz dzielnika SPI$_{CR}$=128 (wartości dla mikrokontrolera AVR), czas trwania transmisji jednej ramki wynosi 69,4µs, czyli mieści się w granicach podanych na rysunku 2 dla operacji bitowych.



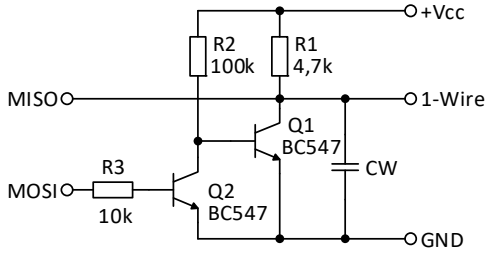
Rys.6. Sekwencje stanów generowanych na magistrali 1-Wire przy wykorzystaniu interfejsu SPI: a) zapis stanu niskiego, b) zapis stanu wysokiego, c) odczyt stanu magistrali (odpowiedź układu slave)

Na rysunku 6 pokazano zawartości rejestrów danych interfejsu SPI przy generowaniu podstawowych operacji bitowych na magistrali 1-Wire. Podane wartości bitowe ramek należy, przed rozpoczęciem transmisji, wpisać do rejestru nadajnika interfejsu SPI. Będą one przesyłane linią MOSI. Po zakończeniu wysyłania jednej ramki odebrane dane (dla odczytu) z linii MISO będą dostępne w rejestrze odbiornika interfejsu SPI (zwykle jest to fizycznie ten sam rejestr dla nadawania i odbierania danych, por. rysunek 4).

W przypadku generowania sekwencji zerującej i sprawdzającej obecność układów slave, dołączonych do magistrali 1-Wire, wymaganych jest wysłanie kolejno kilku ramek danych, by uzyskać pożądane czasy trwania tych sekwencji. Dla impulsu RESET będzie to ok. 483µs (7·69µs, tj. wysyłanych jest 7 bajtów o wartości 0x00), natomiast dla odczytania PULSE wyniesie 207µs (3·69µs, tj. wysyłane są 3 bajty o wartościach 0xFF i równocześnie odczytywane są odbierane dane, przy czym dwa ostatnie bajty o wartości 0x00 odpowiadają obecności przynajmniej jednego układu slave). Podczas generowania impulsu RESET wymagane jest utrzymanie stanu niskiego na linii danych przez określony czas, pojawienie się nawet krótkich impulsów przerywa ten proces. Okazuje się, że podczas wysyłania sekwencji kilku ramek, pomiędzy nimi pojawia się stan wysoki jak to pokazano na rzeczywistych przebiegach na rysunku 7 (pomiary zrealizowane analizatorem stanów logicznych Saleae-Logic). Maksymalna wartość tego czasu nie przekracza ok. 2µs (pomiary doświadczalne).

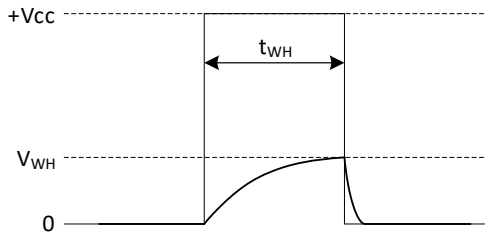
Rys.7. Przebiegi czasowe interfejsu 1-Wire przy generowaniu sekwencji zerującej RESET-PULSE wraz z sygnałem zegarowym

Pojawiające się niepożądane stany wysokie można w prosty sposób wyeliminować modyfikując nieznacznie układ konwertera, pokazanego na rysunku 8, poprzez dołączenie dodatkowego kondensatora o niewielkiej pojemności. Wartość jej nie może być zbyt mała, tak by skutecznie eliminować niepożądane impulsy podczas sekwencji RESET-PULSE z jednej strony, z drugiej zaś nie może być zbyt wysoka, by nie wprowadzać zakłóceń podczas transmisji pojedynczych bitów, a tym samym uniknąć konieczności nadmiernego wydłużenia czasów trwania występujących w odpowiednich sekwencjach impulsów (zob. rysunek 2).



Rys.8. Układ konwertera interfejsu SPI na interfejs 1-Wire pozwalający poprawnie generować sekwencję zerującą RESET-PULSE

Przy czasie trwania niepożądanego impulsu o logicznym stanie wysokim t_{WH} przebieg czasowy napięcie na linii danych interfejsu 1-Wire pokazano na rysunku 9. By układ spełniał swoją rolę, napięcie na kondensatorze C po czasie t_{WH} nie powinno przekroczyć wartości V_{WH} traktowanej jako stan niski (zgodnie z notą katalogową [12] maksymalna wartość tego napięcia wynosi 0.8V lub 0.5V przy zasilaniu pasożytniczym układu slave z linii danych).



Rys.9. Przebieg napięcia na magistrali 1-Wire podczas występowania niepożądanych stanów nieaktywnych w czasie transmisji danych za pośrednictwem interfejsu SPI

Czas potrzebny na rozładowanie kondensatora można pominąć w rozważaniach, gdyż jest on rozładowywany bardzo szybko przez klucz tranzystorowy Q1. Z kolei ładowanie kondensatora odbywa się poprzez rezystancję R1 (o znacznej wartości) i jedynie czas ładowania odgrywa tu dominującą rolę w działaniu konwertera.

Ze wzoru opisującego przebieg napięcia na linii danych interfejsu można wyznaczyć wartość pojemności C_W, zgodnie ze wzorem

$$C_W = -\frac{t_{WH}}{R_1 \cdot \ln\left(1-\frac{V_{WHmax}}{V_{CC}}\right)}, \tag{2}$$

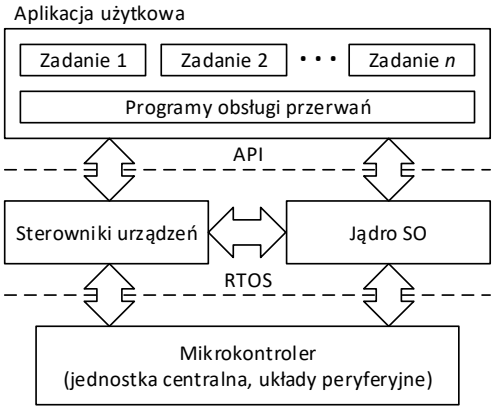
Po podstawieniu do wzoru wartości, wyznaczona pojemność wynosi C_W=2,4nF. Zastosowany w układzie kondensator, dobrany z szeregu E12, o najbardziej zbliżonej do obliczonej wartości pojemności równej 2,2nF pozwolił uzyskać przebiegi na linii danych interfejsu pokazane na rysunku 10. Widać w tym przypadku poprawnie generowany sygnał zerowania RESET, odpowiedź P układu podrzędnego slave (PULSE-PRESENCE) oraz niezakłócone przesyłanie kolejnych bitów danych (polecenie 0xCC).

Rys.10. Przebiegi czasowe interfejsu 1-Wire przy zastosowaniu kondensatora i poprawnej detekcji ramek danych

System FreeRTOS

System Operacyjny czasu rzeczywistego RTOS, którego przykładem jest FreeRTOS [13], jest często wykorzystywany w oprogramowaniu dla systemów wbudowanych. Umożliwia on uruchamianie w systemie wielu niezależnych zadań (w tym programów obsługi przerwań), wchodzących w skład aplikacji użytkowych, oraz zapewnia mechanizmy komunikacji pomiędzy nimi. Ogólna architektura systemu wbudowanego pokazana została na rysunku 11. Interfejs systemu operacyjnego z aplikacją użytkową stanowi warstwa API (ang. Application Programming Interface), w skład której wchodzą standardowe funkcje i struktury danych zapewniające możliwość wykorzystania w prosty sposób mechanizmów RTOS [14]. Samo jądro systemu złożone jest z takich modułów jak: zarządzanie pamięcią (obsługa sterty i administracja danymi w pamięci), funkcje zależne od sprzętu (integrują oprogramowanie z dostępnymi układami peryferyjnymi wykorzystanego w projekcie mikrokontrolera), komunikacja między zadaniami (zapewnia wymianę danych i synchronizację pomiędzy zadaniami - kolejki, semafory itp.), planista (przełącza zadania i udostępnia dla każdego z

nich zasoby mikrokontrolera, podejmuje decyzję o kolejności wykonywania zadań).



Rys.11. Architektura systemu wbudowanego z systemem operacyjnym czasu rzeczywistego

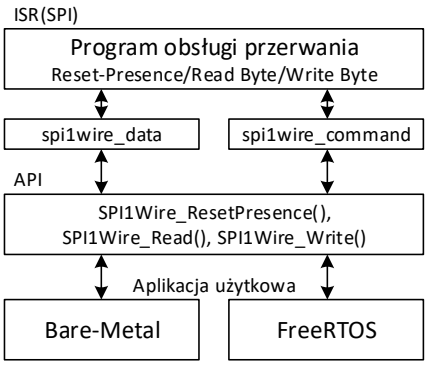
Sterowniki urządzeń, opracowane dla konkretnego systemy wbudowanego, pozwalają oddzielić użytkowe oprogramowanie od warstwy sprzętowej systemu i w pewnym stopniu się od niej uniezależnić. Posiadają one własne funkcje do obsługi układów peryferyjnych, z których korzysta zarówno aplikacja jak i system operacyjny.

Podstawowym układem peryferyjnym, z jakiego korzysta system operacyjny, jest licznik sprzętowy mikrokontrolera używany do odmierzania interwałów czasowych (tzw. System-Tick). Po każdym odmierzeniu kwantu czasu przerywana jest aplikacja użytkowa (konkretne zadanie) i następuje przejście do systemu operacyjnego, który realizuje obsługę komunikacji między zadaniami, zarządza pamięcią i przełącza zadania. Licznik jest tak skonfigurowany, by umożliwić na wystąpienie cyklicznego przerwania typowo co 1ms. Jest to minimalny interwał czasu rozróżniany przez system operacyjny. Jego wartość można zmieniać, ale zbyt mały powoduje, że mniej czasu jednostki centralnej mikrokontrolera jest poświęcanych na realizację zadań, z kolei zbyt duża wartość zmniejsza reakcję systemu operacyjnego na zdarzenia jakie pojawiają się w systemie, a które muszą być obsłużone w ściśle określonym czasie. W przypadku konieczności wykorzystania możliwości obsługi zdarzeń w krótszym czasie, należy ich obsługę przenieść poza system operacyjny i wykorzystać dodatkowy układ peryferyjny mikrokontrolera (np. licznik). Przykładem może być tutaj opisywany interfejs 1-Wire i dostęp do niego z poziomu systemu operacyjnego RTOS.

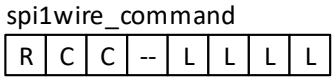
Integracja interfejsu 1-Wire w systemie FreeRTOS

Implementacja interfejsu 1-Wire w systemie wbudowanym obejmuje dwa podstawowe elementy pokazane na rysunku 12: program obsługi przerwania ISR od interfejsu SPI oraz zestaw funkcji API, które stanowią element pośredniczący pomiędzy aplikacją użytkową, a wspomnianym programem obsługi przerwania. Podejście takie umożliwia oddzielić warstwę programową od warstwy sprzętowej, tym samym pozwala ukryć sprzętowe szczegóły implementacyjne obsługi interfejsu 1-Wire.

Program obsługi przerwania wywoływany jest po wysłaniu (i równoczesnym odebraniu) danej z układu transmisji szeregowej SPI. Podstawowym jego zadaniem jest, w omawianym zastosowaniu, generowanie sekwencji ciągu ramek, umożliwiających utworzenie sekwencji RESET-PRESENCE oraz odczyt i zapis pojedynczych bitów protokołu 1-Wire, a następnie formowanie z nich pełnych bajtów danych. Wymiana danych pomiędzy programem obsługi przerwania oraz programem głównym odbywa się za pośrednictwem dwóch zmiennych (typu bajtowego): *spi1wire_data*, wykorzystywanej do przekazywania danych (podczas zapisu i odczytu), oraz *spi1wire_command*, która określa typ wykonywanej operacji oraz jej status. Format tej ostatniej przedstawia rysunek 13.



Rys.12. Integracja obsługi interfejsu SPI-1Wire z aplikacją użytkową



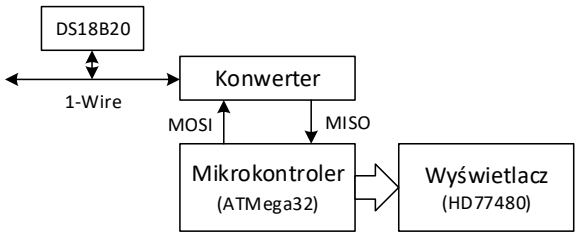
Rys.13. Zmienna *spi1wire_command* wykorzystywana do obsługi interfejsu SPI-1Wire

Bit oznaczony jako R określa zakończenie jednej z sekwencji tj. RESET-PULSE, tj. zapis lub odczyt danej. Bity CC określają typ wykonywanej sekwencji na magistrali 1-Wire, natomiast LLLL wykorzystywane są przez program obsługi przerwania do odliczania wysłanych bajtów danych (generowania poszczególnych bitów protokołu 1-Wire).

Z programem obsługi przerwania współpracują funkcje API zorganizowane w bibliotekę SPI1Wire. Są to: *SPI1Wire_ResetPresence()*, która zwraca wartość różną od zera jak przynajmniej jeden układ slave jest podłączony do magistrali 1-Wire, *SPI1Wire_Read()*, zwracającą wartość bajtu odczytanego z magistrali 1-Wire, oraz *SPI1Wire_Write(byte)*, zapisującą do układu slave wartość 8-bitową byte. Każda z tych funkcji wpisuje odpowiednią wartość do zmiennej *spi1wire_command*, odblokowuje przerwanie od interfejsu SPI i inicjuje transmisję poprzez wysłanie pierwszego bajtu danych, a następne wysyłane już są z programu obsługi przerwania. Po zakończeniu transmisji danych przez układ SPI w programie obsługi blokowane jest przerwanie i ustawiany jest znacznik R w zmiennej *spi1wire_command* (rysunek 13).

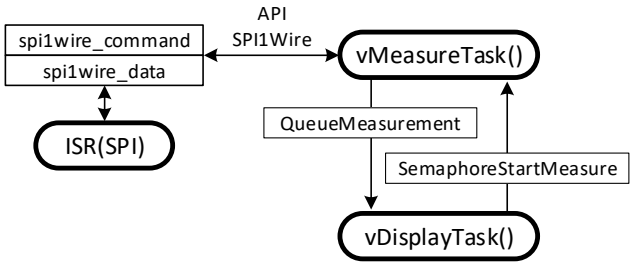
Funkcje API biblioteki SPI1Wire mogą być wykorzystane nie tylko w aplikacjach uruchamianych w środowisku systemu operacyjnego FreeRTOS, ale także w aplikacjach dedykowanych typu „bare-metal”.

W celu praktycznego sprawdzenia zaproponowanego rozwiązania, opracowany został układ prototypowy z interfejsem 1-Wire, do którego podłączony został czujnik temperatury DS18B20 [12]. Wykorzystano w nim mikrokontroler AVR rodziny ATMega typu ATMega32 [15] oraz wyświetlacz alfanumeryczny LCD ze sterownikiem HD77480 i interfejsem szeregowym TWI. Bazuje on na rozwiązaniu zaproponowanym w [16]. Do sterowania interfejsem 1-Wire wykorzystany został układ transmisji szeregowej SPI mikrokontrolera oraz omówiony wcześniej konwerter (pokazany na rysunku 8). Schemat blokowy systemu wbudowanego, wykorzystanego do testów, pokazany został na rysunku 14.



Rys.14. System wbudowany realizujący funkcję pomiaru i wyświetlania temperatury

Oprogramowanie dla tego systemu opracowane zostało z wykorzystaniem systemu operacyjnego FreeRTOS (w wersji 7.0.2). Aplikacja użytkowa składa się z dwóch zadań oraz programu obsługi przerwania, jak to pokazano na rysunku 15. Do komunikacji między zadaniami wykorzystane zostały kolejka oraz semafor binarny.



Rys.15. Schemat blokowy aplikacji FreeRTOS

Zadaniem pośredniczącym pomiędzy aplikacją użytkową a oprogramowaniem obsługującym interfejs 1-Wire z wykorzystaniem interfejsu SPI jest *vMeasureTask()*. W odpowiedzi na ustawienie semafora binarnego *SemaphoreStartMeasure* zadanie to inicjuje odczyt temperatury z czujnika DS18B20 przy wykorzystaniu opisanych funkcji API biblioteki SPI1Wire. Odczyt temperatury sprowadza się do wykonania następujących sekwencji rozkazów protokołu 1-Wire [12]:

- wysłanie rozkazu RESET-PULSE i sprawdzenie (PRESENCE), czy do magistrali jest podłączony układ slave,
- wysłanie rozkazu pomijającego adresowanie układu slave (SkipROM),
- wysłanie rozkazu inicjującego pomiar temperatury (ConvertT) i odczekanie na jego zakończenie,
- wysłanie rozkazu RESET-PULSE,
- wysłanie rozkazu pomijającego adresowanie układu slave (SkipROM),
- wysłanie rozkazu odczytu pamięci RAM układu slave (ReadScratchpad),
- odczyt mniej znaczącego bajtu wartości temperatury,
- odczyt bardziej znaczącego bajtu wartości temperatury.

Odczytana wartość temperatury (słowo 16-bitowe) przekazywane jest do kolejki *QueueMeasurement*. W przypadku, kiedy brak jest układu slave (nie odpowiada impulsem PRESENCE), do kolejki jest wprowadzany kod błędu (wartość umowna 0xFFFF).

Zadanie *vDisplayTask()* realizuje funkcję wyświetlania temperatury na wyświetlaczu LCD. Cyklicznie inicjuje wykonanie pomiaru temperatury poprzez ustawienie semafora binarnego *SemaphoreStartMeasure*, po czym oczekuje na pobranie danej z kolejki *QueueMeasurement*. W przypadku odebrania kodu błędu, zamiast wartości temperatury, wyświetlany jest odpowiedni komunikat.

Wnioski

Obsługa interfejsu 1-Wire, w systemie wbudowanym zrealizowanym w oparciu o mikrokontroler, zwykle implementowana jest na drodze programowej, rzadziej z użyciem wybranych układów peryferyjnych mikrokontrolera (liczniki, interfejs UART) i systemu przerwań. W artykule została przedstawiona nowa metoda oparta na wykorzystaniu interfejsu SPI, systemu przerwań i zmodyfikowanego układu konwertera interfejsu SPI na interfejs 1-Wire.

Działanie układu przetestowane zostało w przykładowym systemie wbudowanym, opartym na mikrokontrolerze AVR rodziny ATMega, realizującym pomiar temperatury. Oprogramowanie do obsługi interfejsu 1-Wire, wykorzystujące interfejs szeregowy SPI (biblioteka SPI1Wire), oraz oprogramowanie użytkowe opracowane zostało dla systemu operacyjnego FreeRTOS. Przeprowadzone liczne badania i testy potwierdziły poprawność działania zaproponowanego rozwiązania. Opracowane oprogramowanie można także wykorzystać w aplikacjach typu „bare-metal", co również zostało praktycznie przetestowane. Kody źródłowe biblioteki SPI1Wire, opracowane aplikacje, dokumentacja oraz materiały dodatkowe zostały udostępnione na stronie [17].

Zaproponowane rozwiązanie może być dalej rozwijane, zarówno od strony programowej jak i sprzętowej. Opracowane oprogramowanie warto uzupełnić o funkcje do obsługi wielu układów slave, dołączonych do magistrali 1-Wire, jak również zapewnić możliwość jego przenoszenia na inne platformy sprzętowo-programowe. Jedynym warunkiem jest, by dany mikrokontroler wyposażony był w układ transmisji szeregowej SPI. Zaproponowane rozwiązanie układowe, z kondensatorem eliminującym niepożądane impulsy podczas sekwencji RESET-PULSE, również może zostać zmodyfikowane. Zamiast kondensatora może zostać wykorzystane dodatkowe wyprowadzenie mikrokontrolera, zadaniem którego będzie wymuszenie permanentnego stanu niskiego, wyłącznie podczas generowania sekwencji RESET-PULSE. W tym przypadku, do obsługi interfejsu 1-Wire, wymagane będzie użycie trzech wyprowadzeń mikrokontrolera zamiast tylko dwóch.

Praca została sfinansowana ze środków Ministerstwa Nauki i Szkolnictwa Wyższego na działalność statutową BK-237/RAU12/2023.

***Autor**: dr inż. Bernard Wyrwoł, Politechnika Śląska, Wydział Automatyki, Elektroniki i Informatyki, ul. Akademicka 16, 44-100 Gliwice, email: bwyrwol@polsl.pl*

LITERATURA

[1] Borowik, B., Reading the identification code from the 1-Wire iButton using the PIC16F628 microcontroller. *Przeglad Elektrotechniczny*, 84 (2008), nr 3, 250-251

[2] Wu Tao, Zhou Xiaomin, He Xiaonan, Xu Yan, Design of distributed temperature-measuring system based on 1-wire bus for ultra-kilometre frozen deep well, *Evolutionary Intelligence*, 4 (2022), 2505-2514

[3] Analog Devices, Overview of 1-Wire Technology and Its Use Analog Devices. [web page] www.analog.com, Jun. 2008. [Accessed on 27 Jan. 2023]

[4] Dudak Juraj, Tanuska Pavol, Gaspar Gabriel, Fabo Peter, ARM-Based Universal 1-Wire Module Solution, *Journal of Sensors*, Hindawi, 2018 (2018), ID 5268247, 1-16

[5] Vavrla Tomas, Koziorek Jiri, Temperature Measurement in Boreholes by Programmable Logic Controller B&R and Temperature Sensors 1-Wire, *IFAC Proceedings Volumes*, 45 (2012), 382-387

[6] Xiao Shangli, Xu Weisheng, Yu Youling, A simulative building fire spread tracking system based on FPGA and 1-wire bus

sensor network, *7th International Conference on System Simulation and Scientific Computing*, (2008), 1482-1486
[7] Cambronero María Emilia, Macia Hermenegilda, Valero Valentín, Orozco-Barbosa Luis, Modeling and Analysis of the 1-Wire Communication Protocol Using Timed Colored Petri Nets, *IEEE Access*, 6 (2018), 27356-27372
[8] Xue Hongmei, Research and Development of an Intelligent Temperature-Measuring System Based on 1-Wire Bus, *2008 International Conference on Intelligent Computation Technology and Automation*, 2 (2008), 30-33
[9] Maniyar Sashavalli, Microchip, 1-Wire Communication with PIC Microcontroller. Application Note, [web page] 1-wire-pic.pdf, Feb. 2008. [Accessed on 23 Feb. 2023]
[10] Analog Devices, Reading and Writing 1-Wire Devices Through Serial Interfaces, [web page] www.analog.com, Jun. 2009, [Accessed on 27 Jan. 2023]
[11] Wyrwoł, B., Implementacja interfejsu 1-Wire w systemie FreeRTOS dla mikrokontrolera AVR, *Elektronika: konstrukcje, technologie, zastosowania*, 55 (2014), 76-78(5)
[12] Analog Devices, DS18B20: Programmable Resolution 1-Wire Digital Thermometer Data Sheet (Rev. 6). [web page] DS18B20.pdf, Sep. 2019. [Accessed on 22 Feb. 2023]
[13] Barry Richard, Mastering the FreeRTOS Real Time Kernel A Hands - On Tutorial Guide, [web page] FreeRTOSTutorial.pdf, 2016, [Accessed on 23 Feb. 2023]
[14] Amazon Web Services, The FreeRTOS Reference Manual API Functions and Configuration Options, [web page] FreeRTOSManual.pdf, 2017, [Accessed on 23 Feb. 2023]
[15] Microchip: ATmega32(L) - Complete Datasheet. [web page] ATMega32.pdf, Feb. 2011. [Accessed on 22 Feb. 2023]
[16] Wyrwoł, B., System prototypowania aplikacji wykorzystujących logikę rozmytą AVR-FPGA-FIS, *Przegląd Elektrotechniczny*, 87 (2011), nr 10, 60–63, 2011
[17] Wyrwoł, B., Repozytorium SPI1Wire - Obsługa interfejsu 1-Wire z wykorzystaniem układu SPI, [web page] https://github.com/bwyrwol/SPI1Wire, Jan. 2023, [Accessed on 21 Feb. 2023]